# PREFACE

This reference manual is intended for system managers and application programmers who are using or are going to use PCIF/1000 to supervise and interact with Programmable Controllers (PCs).

The manual assumes a knowledge of the HP 1000 computer and its RTE-A operating system, plus the operating principles of Programmable Controllers. It is divided into nine chapters, and can be used as both a reference and a user guide.

New users of PCIF/1000 should read Chapters 1, 2, 3 and 5. System Managers will find Chapters 5 and 6 useful when installing PCIF/1000 for the first time, and Chapters 7 and 8 when configuring a PC-computer system. The reference sections for application programmers are Chapters 4 and 9.

The handler manuals (e.g. Using PCIF/1000 with Allen-Bradley Programmable Controllers), provide information specific to the particular handler software purchased along with the core PCIF software. Use your handler manual in conjunction with this manual to install and operate your PCIF system.

- CHAPTER 1 provides a definition of the product, its purpose, what it consists of and a very brief description of how it works. Also listed are the hardware and software requirements of PCIF/1000.

- CHAPTER 2 describes in more detail the operation of PCIF/1000, including definitions of PCIF/1000 terms, such as configuration, run-time and PC Access Routines.

- CHAPTER 3 is a review of Programmable Controllers. It describes the functions that PCs perform in the industrial automation area, gives an example of a PC-computer system and outlines the logical structure of a PC's memory.

- CHAPTER 4 is intended for application programmers and is a reference guide that lists all the PC Access Routines available with PCIF/1000. An analysis is provided with each routine, plus the routine's calling sequence, entry/return parameters and return codes.

- CHAPTER 5 outlines the installation and configuration processes of PCIF/1000, and provides a definition of the preconfigurator and the configuration programs.

- CHAPTER 6 describes the installation procedure, and provides examples of the three possible connection methods between a computer and a PC system, i.e. RS232C; 20mA current loop; and RS232C/current loop. The product's software installation is divided into two procedures, one for the configurator programs and one for generating an RTE-A run-time system.

- CHAPTER 7 is a step-by-step description of the preconfigurator program, the screens it displays to the operator and the information to be entered on these screens.

- CHAPTER 8 provides a similar step-by-step description of the configuration editor program.

- CHAPTER 9 analyzes for application programmers the run-time operation of the PCIF/1000 utility programs. It indicates in detail how to start and stop the PCIF/1000 subsystem, possible errors that might occur and how to deal with them.

- APPENDIX A lists and explains all the error codes that may be returned by the PCIF/1000 monitor, at either configuration or run-time.

- APPENDIX B is a description and listing of PCTST, a program to assist in testing, verification, program debugging and learning PCIF/1000.

# CONTENTS

# HP Computer Museum
# www.hpmuseum.net

**For research and education purposes only.**

Chapter 4
PC ACCESS ROUTINES



Chapter 5
INSTALLATION OVERVIEW

Chapter 8
CONFIGURATION



Chapter 9
RUN-TIME OPERATION

Appendix A
ERROR MESSAGES



Appendix B
PCTST LISTING

# Chapter 1
# GENERAL INTRODUCTION

## 1.1 PCIF/1000 DEFINITION & PURPOSE

PCIF/1000 is a software interface that allows the connection of Programmable Controllers (P/Cs) of different manufacturers and types to an HP 1000 A-Series supervisory computer. The product provides an application programmer with standard and transparent access to a range of P/Cs, via the computing power and facilities offered by the operating system of the HP 1000, and thus enables easier supervision of programmable controllers in an industrial automation environment.

## 1.2 BRIEF SPECIFICATION


| | |
|---|---|
| Product Name : | PCIF/1000 |
| Product Number : | 94200B |
| Operating System Requirements : | RTE-A |
| Supported Program Languages : | Pascal and/or FORTRAN |
| Supported P/C Manufacturers : | Allen-Bradley<br>Gould-Modicon<br>Siemens<br>Telemecanique |

## 1.3 WHAT IT DOES

PCIF/1000 allows real-time access from an HP 1000 to the data and programs of P/Cs. This means that data and programs from one P/C may be uploaded and downloaded to another P/C in the same system, under instructions from an application program running on the HP 1000.

PCIF/1000 is not a program development tool for P/Cs. Its main purpose is to allow HP 1000 users to exchange data and programs between P/Cs connected in a system, or collect data and programs from these P/Cs. PCIF/1000 provides a means of achieving this aim which avoids alteration of an application program each time the number or types of P/Cs in the system are changed.

## 1.4 HOW IT WORKS

The following is a brief summary of how PCIF/1000 allows application programs to interact with programmable controllers.

(1) The user's P/Cs are connected to the supervisory HP 1000 via a multiplexer (MUX). (See your handler manual for the specific MUX to be used with your P/C brand.)

(2) A logical picture (in fact a file) of the types and manufacturers of the connected P/Cs is created by an operator feeding the relevant information to configuration programs. These programs are part of the PCIF/1000 package and may be used to update the configuration file if the connected P/Cs are changed.

(3) An application program runs on the HP 1000, activates a PCIF/1000 monitor program and uses the previously created configuration file to identify target P/Cs. The application program does not know the type of the target P/C or its connection method. This P/C type and protocol independence allows the constituents of the P/C system to be changed without the subsequent need to rewrite the application program.

(4) Information exchanges occur between the application program and the target P/Cs, and these P/Cs reply to requests from the program. The requests may be "read program" to one P/C, and "write program" to another. The information exchange is supervised by the PCIF/1000 monitor program.

## 1.5 WHAT IT CONSISTS OF

Generally speaking, PCIF/1000 is composed of four basic parts:

- A menu-driven configuration program that allows the system manager to describe the characteristics of the various P/Cs installed.

- A library of high-level routines that implement the application program interface to PCIF/1000 at run-time.

- A monitor program that controls the overall operation of the PCIF/1000 software at run-time.

- A set of handler programs for the interface between PCIF/1000 and the P/Cs. Note that these programs are a separate product from the core and are documented in separate handler-specific manuals.

## 1.6 PCIF/1000 REQUIREMENTS

* HP 1000 A-Series computer (1Mb minimum memory and 10 Mb minimum disc storage space).

* RTE-A operating system.

* Macro assembler program resident in the operating system.

* FORTRAN 77 or Pascal language.

* At least one of the following HP block-mode terminals: 2622A; 2623A; 2624B; 2626A; 2627A; 2382A; 2647A/F; 2648A; 150A. NOTE: The HP 2645A terminal is not supported.

* HP multiplexer. (See your handler manual for the specific MUX to be used with your P/C brand.)

# Chapter 2
# OPERATING PRINCIPLES

## 2.1 INTRODUCTION

This explanatory chapter is written for operators who are using PCIF/1000 for their first time. For this explanation it is assumed that the product is to be installed and configured, and then used with application programs.

It is important to realize the distinction between the configuration phase and the run-time phase of PCIF/1000. Section 2 (GENERAL) includes, therefore, a definition of the two phases and an outline of the actions required to facilitate a progression from installation to configuration, and from configuration-time to run-time.

The third section (PC-COMPUTER INFORMATION EXCHANGE) analyzes what is happening inside PCIF/1000 at run-time, and explains the various modules that comprise the PCIF/1000 subsystem.

## 2.2 GENERAL

### 2.2.1 PC-Computer System Overall Appearance

A typical P/C and computer system consists of at least one P/C connected to a computer via an interface and a connection cable. For convenience the data communication path is referred to as the "highway", whether the physical implementation of this connection is point to point, multipoint, or whatever communication protocol is used.

The P/C and the computer can exchange information, programs, status or data through the communication cable and the appropriate hardware interfaces. These interfaces must be electrically compatible and use the same protocol. A multiplexer (MUX) is used as the interface at the HP computer end of the communication highway. On the P/C side the interface used is determined by the manufacturer of the P/C.

The following figure shows the general appearance and the constituent parts of a PC-computer system using PCIF/1000.

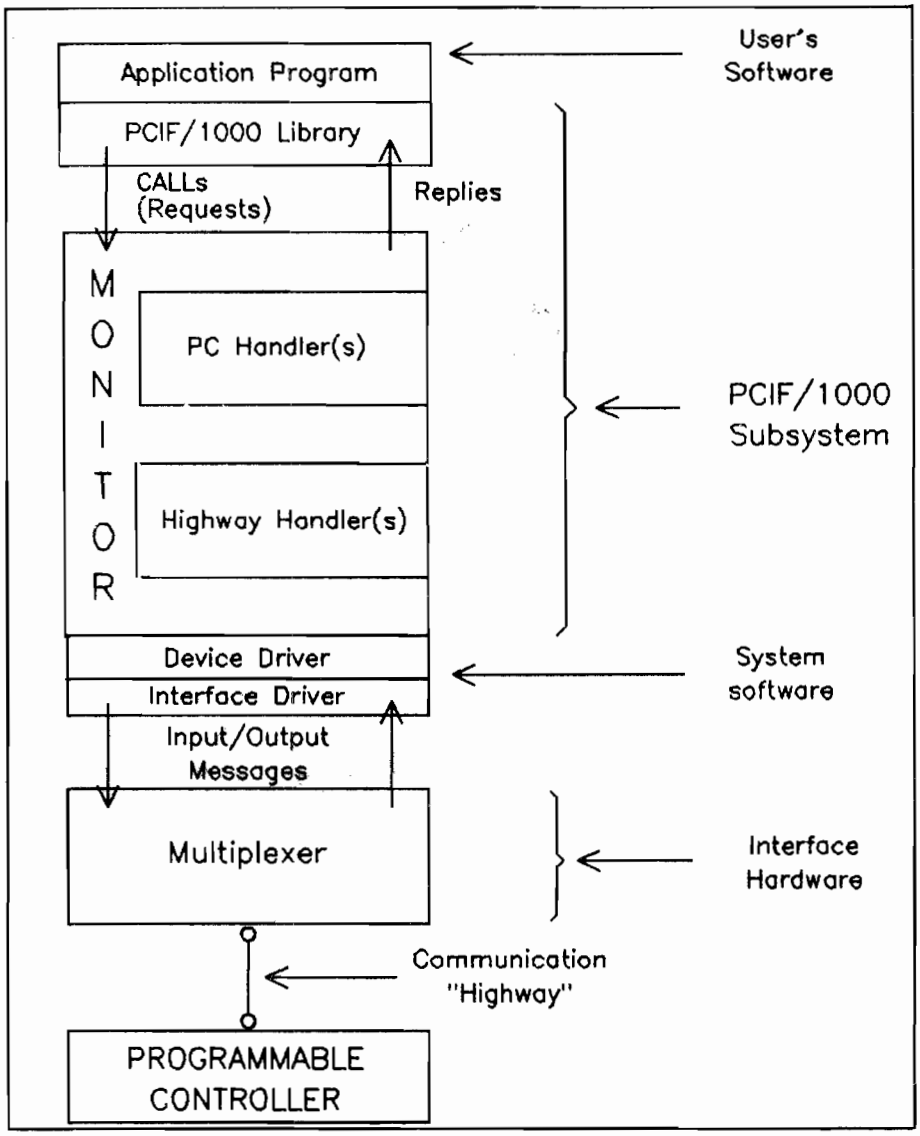



Figure 2.1 General Appearance of a PC-Computer System Using PCIF/1000

With reference to the figure above, note that:

Application Program is the user written software that interacts with programmable controllers via PCIF/1000 at run-time. More than one application program may be handled by PCIF/1000 during the same run-time session.

PCIF/1000 Library is located on disc and contains the P/C Access Routines. During the linking process of an application program the required P/C Access Routines are found in this library and appended to the application program for use at run-time. The routines are further explained in section 2.3.

PCIF Subsystem comprises the monitor plus the P/C and highway handlers, and is brought into the memory of the supervisory HP 1000 at run-time. The "E" is a representation of how these modules interact within the memory when processing requests and replies between application programs and P/Cs. The components of the subsystem are explained in section 2.3.

Device Driver is a software interface module provided with the PCIF software.

Interface Driver is a software interface module provided with the RTE-A operating system of the HP 1000.

Multiplexer (MUX) is a microprocessor based interface. The 12041A and 12041B MUXs contain firmware that is altered at run-time by protocol code downloaded from the highway handler. In addition, these MUXs accept one composite input signal and convert it to eight outputs, each containing the relevant protocol code for its target P/C.

Programmable Controller is a P/C or P/Cs whose manufacturer and type are defined by the user. Information on the connected P/Cs is provided to PCIF/1000 by an operator at configuration-time. The P/Cs themselves are further discussed in Chapter 3.

## 2.2.2 Getting PCIF/1000 Ready For Use

From an operator's viewpoint, PCIF/1000 can be divided into two main areas that require understanding. These are:

- the configuration environment
- the run-time environment

These two environments are created from the supplied media (i.e tapes, cartridges etc) by the installer of PCIF/1000. The operator and installer of PCIF/1000 may of course be the same person, but for explanatory purposes their tasks may be separately considered. The installer, therefore, must complete the following tasks in order to create the two environments for the operator:

- Verify that the target PC-computer system meets the hardware requirements of PCIF/1000, such as compatible protocol, interface and connection details.

- Generate an RTE-A operating system for PCIF/1000 on the supervisory HP 1000.

- Install the PCIF/1000 software onto the the HP 1000.

The above procedure is a very broad picture of the installation process, which is fully described in Chapters 5 and 6.

Now the operator takes over. Remember that PCIF/1000 provides an adaptable software interface between P/Cs (number and type defined by the user) and application programs running on the HP 1000. So the first task of the operator is to configure the installed PCIF/1000 software to meet the requirements of the target PC-computer system. This is achieved with two interactive programs, collectively referred to as the configuration environment or configuration-time.

At the end of configuration-time the operator is ready to load application programs on the HP 1000 and actually use the facilities of PCIF/1000 to command, obtain and transfer information to the target P/Cs. This phase is referred to as run-time.

## 2.2.3 Definition of Configuration-Time

Configuration-time occurs after installation and before run-time, and refers to the use of two programs: PRECONFIGURATOR followed by CONFIGURATION EDITOR. In these programs the operator supplies information (relating to the target P/C system upon which PCIF/1000 will be expected to run) to a sequence of formatted screens. These screens are displayed and managed by an internal program called F/1000, which is supplied with PCIF/1000.

In the preconfigurator program, the operator defines the manufacturer(s) of the P/Cs installed on the target system. This information is used to build the second program, the configuration editor. Here, specific details on the previously defined P/Cs are supplied, and a logical P/C number is attached to each installed P/C.

All the information supplied by the operator during both programs is checked for validity to ensure that the operator is configuring a logically feasible P/C system. If invalid information is supplied, the operator is warned by error messages, and the configuration editor program will not be completed.

The end result of configuration-time is a unique configuration file that logically describes the target P/C system. The file is given a name by the operator and is called with the command string used to initiate run-time, as shown below:

CI> XQ,PCIF,<configuration file name>...

Note that the "PCIF" in this command string refers to the PCIF/1000 monitor program, which is also unique for the target P/C system and is created at the end of the preconfigurator program.

The above description of configuration-time is an overview of the complete configuration process. Each program is described in detail by Chapter 7 (preconfigurator) and Chapter 8 (configuration editor). It is also advisable to read Chapter 5 (Installation Overview) if you are using the preconfigurator (and therefore the configuration editor) for the first time.

Note that configuration is only necessary the first time PCIF/1000 is used with a particular P/C system, unless the amount or type of P/Cs in this system are subsequently changed. If the amount or types are changed, then only the configuration editor needs to be run again in order to edit the configuration file. However, if a P/C is installed whose manufacturer is different from those previously defined for this target system, then both the preconfigurator and the configuration editor must be run again to create a new PCIF monitor and a new configuration file.

## 2.2.4 Definition of Run-Time

For PCIF/1000, run-time starts with the "XQ,PCIF,<configuration file name>" command, which may be issued by an application program or by an operator from a scheduling terminal.

By run-time, PCIF/1000 comprises a number of different modules that together make up a unique PCIF/1000 subsystem for the target P/Cs. A constant part of this subsystem is the PCIF monitor, and the other parts are the P/C and highway handlers whose construction depends upon the types and manufacturers of the P/Cs in the target system, in fact the information supplied to PCIF/1000 during configuration-time.

Figure 2.1 provides a very simple view of what the PCIF/1000 subsystem looks like at run-time. In fact, a number of utility programs are automatically called after the issue of the run-time command, but before the interaction between application programs and P/Cs can take place. A complete description of the run-time operation appears in Chapter 9.

Again turn your attention to Figure 2.1. Note that the PCIF Library contains high level routines (called the P/C Access Routines) which contain instructions that are converted by the PCIF/1000 subsystem into the required function to be performed on or by the PC(s). There are 23 different PC Access Routines provided in this library by PCIF/1000. Each is designed to implement a different function in the P/C (read/write data, stop/start PC, etc) as instructed from the application program, but is of the same construction irrespective of the type of P/C being accessed. Generally speaking, the application program only needs to logically identify the target P/C, and can ignore the P/Cs type or manufacturer.

## 2.3 PC-COMPUTER INFORMATION EXCHANGE

In this section the description of the principles of operation relate only to what is happening in the PCIF/1000 subsystem at run-time, and therefore how the HP 1000 and P/Cs exchange information.

### 2.3.1 The PCIF/1000 Subsystem

When an application program wants to interact with a P/C, four different modules are activated for the execution of the desired function:

- A particular P/C Access Routine (e.g. PC_WRITED), found in the PCIF Library and appended to the calling application program.
- The PCIF/1000 monitor program.
- The P/C handler corresponding to the target P/C (i.e. the P/C logically identified by the application program).
- The highway handler corresponding to the data communication protocol used for transferring information between the HP 1000 and the target P/C.

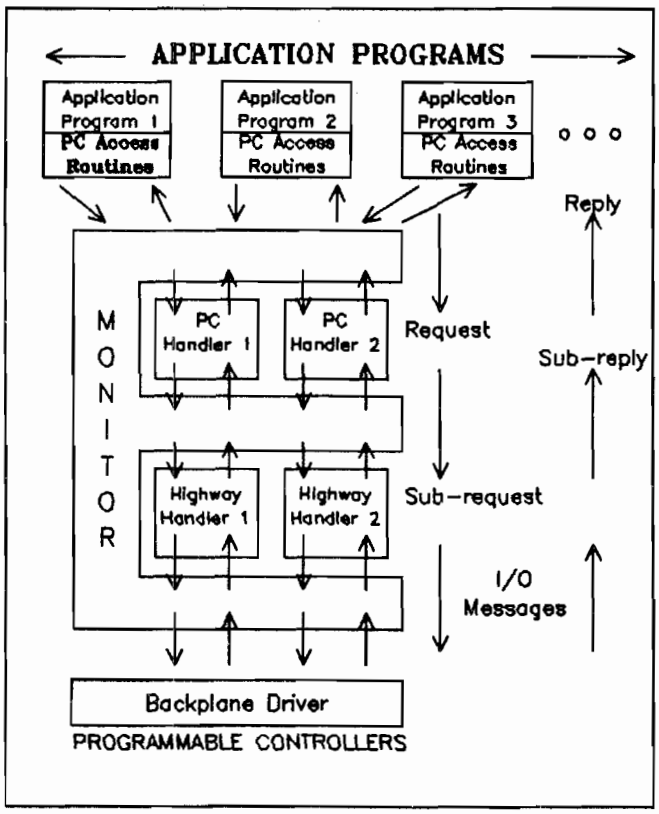



Figure 2.2 PCIF/1000 Subsystem Information Flow

As can be seen from Figure 2.2, the monitor is the main program of PCIF/1000. It manages the different parts, activates the appropriate handlers and supervises the flow of information through the PCIF/1000 subsystem.

With reference to Figure 2.2, the flow of information between the HP 1000 and the P/Cs is as follows:

- An application program issues a REQUEST to exchange data with a P/C by calling the appropriate P/C ACCESS ROUTINE and supplying the target PC's logical identification. This P/C number would have been allocated to the target P/C at configuration-time.

- The PCIF/1000 MONITOR receives the request and verifies that the supplied P/C identification is valid, and that the request is valid for this P/C.

- The monitor then calls the appropriate P/C HANDLER sub-program which translates the request into specific P/C commands, called SUB-REQUESTS. One request from an application program can generate any number of sub-requests (including zero), depending upon the nature of the request and the target PC's type. For instance, if a data buffer destined for the target P/C is longer than the maximum allowed by this P/C, then it will be divided into several messages. These messages can then be sent to the highway handler using several sub-requests.

- Once a sub-request is built, the P/C handler calls a routine in the monitor to find the correct HIGHWAY HANDLER to implement the correct communication protocol between the HP 1000 and the target P/C. The highway handler also manages the I/O instruction for the LUs associated with the target PC's highway.

For the return flow of information, from the P/C to the HP 1000:

- An I/O message is detected by the monitor which recognizes this message as a SUB-REPLY to a corresponding sub-request. The sub-reply is changed into a REPLY by the P/C handler, from which the monitor delivers the appropriate return parameters for the P/C Access Routine that originally made the request. At all times the monitor keeps track of the flow of information in both directions, together with the message tree structure associated with the requests, sub-requests and I/O messages.

In cases of multiple access by several (or only one) application programs to the same P/C, the monitor queues the requests according to instructions from the P/C and highway handlers.

All communication between the PCIF/1000 subsystem and the "outside world", i.e. the P/Cs and their highways, passes through a backplane driver which is part of the RTE-A operating system. (For your specific backplane driver name, see your handler reference manual.)

## 2.3.2 Definition of P/C Access Routines

The P/C Access Routines are an application program's key to the P/C management facilities of PCIF/1000. There are 23 routines stored in a library file on disc. Each has a unique name that accurately reflects the required function to be performed on the target P/C, and each is available for use by an application program written in either Pascal or FORTRAN.

When called, a routine is seen to perform a specific function in the target P/C, such as:

PC_READD : Allows the application program that calls this routine to read (upload) data from the target P/C.

An individual P/C is identified by a logical P/C number that is a required parameter for most P/C Access Routines. This P/C then becomes the specified or target P/C.

The P/C Access Routine returns information to the application program by using return parameters in the routine. The first of these parameters (STAT) informs the application program (that has called the routine) whether or not the routine has successfully completed its task, for example uploading data. If the routine is successful, subsequent parameters will contain the required information from the target P/C.

All the available P/C Access Routines, their parameters, their calling sequences and possible error codes are detailed in Chapter 4.

## 2.3.3 P/C Handler Definition

As a part of the PCIF/1000 subsystem, the P/C handler is a specific subprogram of the monitor. It transforms, for a certain P/C type (or range of P/C types) the application user request into commands for the target P/C. There may be two different P/C handlers configured in the same PCIF/1000 subsystem at the same time, one for each P/C brand installed on the target system.

A P/C handler performs the P/C dependent part of an exchange of information between an application program and a P/C. It is called when a user request (in a P/C Access Routine) is received by the monitor and this request has been recognized as valid from an external and system viewpoint, i.e. the supplied P/C logical identifier exists, the target P/C is connected, etc.

The P/C handler called is the one associated with the P/C whose logical identifier was supplied in the P/C Access Routine. The subrequest generated by the PC handler is given to the monitor for transmission to the target P/C. This transmission is accomplished via the associated highway handler. When the reply to the subrequest comes back from the P/C, the monitor calls the appropriate PC handler again.

## 2.3.4 Highway Handler Definition

The highway handler is a subprogram of the monitor that implements the required data communication protocol needed to transport a request from the PCIF/1000 subsystem to the target P/C.

As with the P/C handler, there may be two different highway handlers configured on the same PCIF/1000 subsystem at the same time, one for each highway type (related to P/C manufacturer) installed in the target P/C system.

The highway handler accepts subrequests from the P/C handler, transmits them to the appropriate P/C, and returns replies coming from a PC to the P/C handler. In general, they create or receive the various I/O messages sent to or received from the communication highway. For example, a data communication protocol may generate several exchanges on the highway for one received subrequest, in addition to the data to be transferred.

# Chapter 3
# REVIEW OF PROGRAMMABLE CONTROLLERS

## 3.1 INTRODUCTION

Programmable Controllers are often used for the direct control of production machinery. They realize the sequencing and the control logic required for the functioning of this type of equipment and can be programmed, using their typical programming language, for providing production information and details on the functioning of the machine itself. This type of information is of great interest for production management, and the connection of P/Cs to supervisory computers is required more and more for providing managers with accurate and timely information on their operations. Such a system includes at least one P/C connected to a computer via an interface card and a so called communication "Highway". A system of this type is described by the next figure:

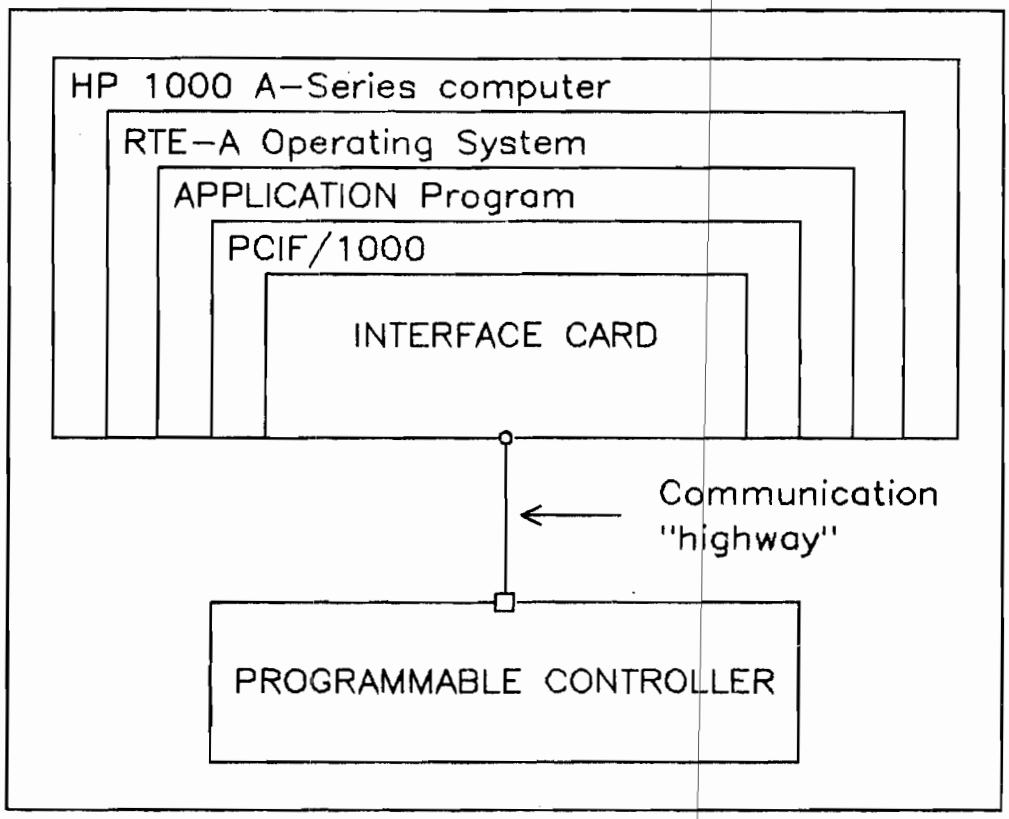



Figure 3.1 P/C and Supervisory Computer Connection

## 3.2 STRUCTURE OF A PC-COMPUTER SYSTEM

Very often a real application will need to connect several P/Cs to the same computer, and can even include several P/Cs from different manufacturers.

Every P/C is physically connected to the computer through an interface card and one HIGHWAY cable, in a point-to-point or a multipoint manner. In the case of a multipoint connection, every P/C is given an address on this highway. This address depends on the P/C manufacturer's hardware, as HP does not provide this type of equipment. The next figure explains these types of connection.

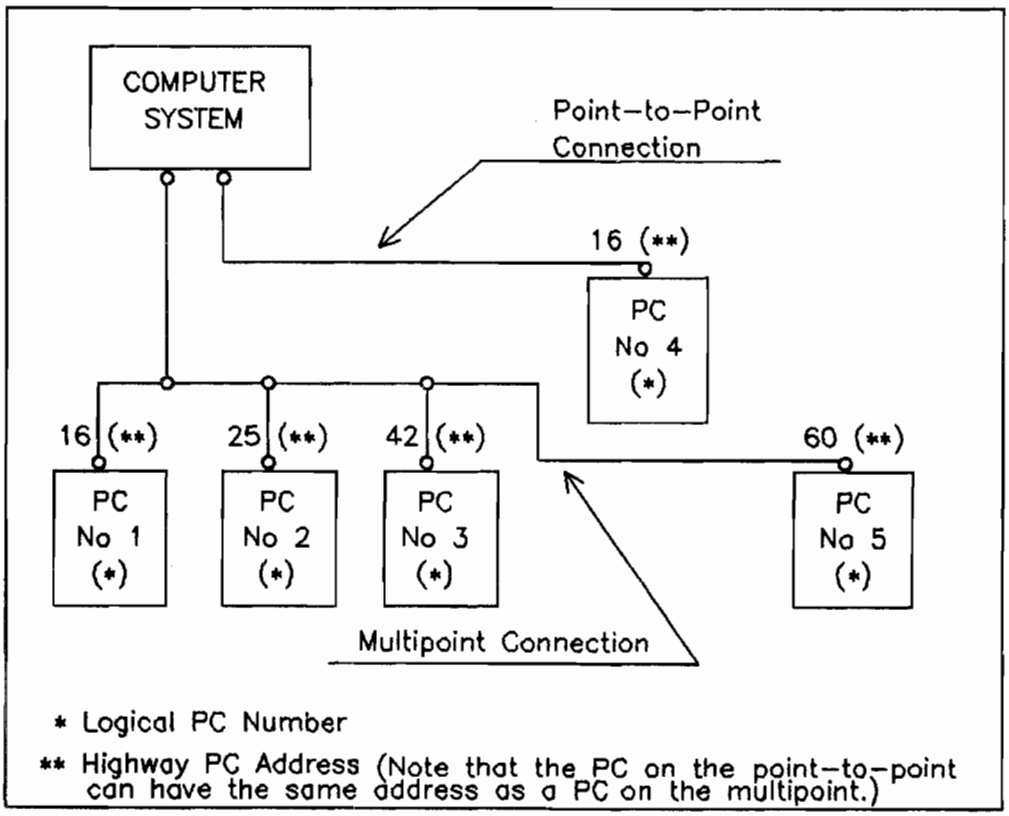



Figure 3.2 P/C Connection Paths

# 3.3 STRUCTURE OF A PROGRAMMABLE CONTROLLER

## 3.3.1 P/C Hardware Structure and Functioning Cycle

Very briefly explained, a P/C is a memory based computer which runs the same machine control program permanently. Its hardware structure includes a CPU, a memory system and industrial input and output circuits, often integrated into a rugged package for resistance to shocks and vibrations. The I/O points are directly connected to the sensors and actuators installed on the machine they are controlling.

At a first glance, P/Cs seem to be running in a permanent loop that involves:

- Reading the input points;
- Calculating the outputs for machine control;
- Performing other tasks (table updates, communication to external devices etc...);
- Applying the calculated outputs to the actuators;
- Return to reading the input points.

## 3.3.2 Logical Memory Structure of a P/C

The memory map of a programmable controller can be described as a list of words used for storing different types of information in various areas. These areas are usually defined by P/C manufacturers as follows:

- The operating system and internal system tables.
- The user written application program (i.e the P/Cs program).
- The program's variables and tables.
- The images of the I/O points.

Other types of areas exist in some P/Cs, depending upon the manufacturer and type. But it is unnecessary to list them all for an understanding of the functioning of PCIF/1000.

The main role of PCIF/1000 is to access variables and programs of the connected P/Cs via their communication interface. This access is achieved by asking the P/C to execute specific functions which are very often memory area type dependent. Figure 3.3 shows the memory map of a theoretical P/C. Program areas, timers, counters, tables and I/O images are not accessed by the same functions. In addition, it could be very dangerous to write into the program memory area when the P/C is running. Therefore, this type of access is very often protected and, in some cases, forbidden by P/C manufacturers.

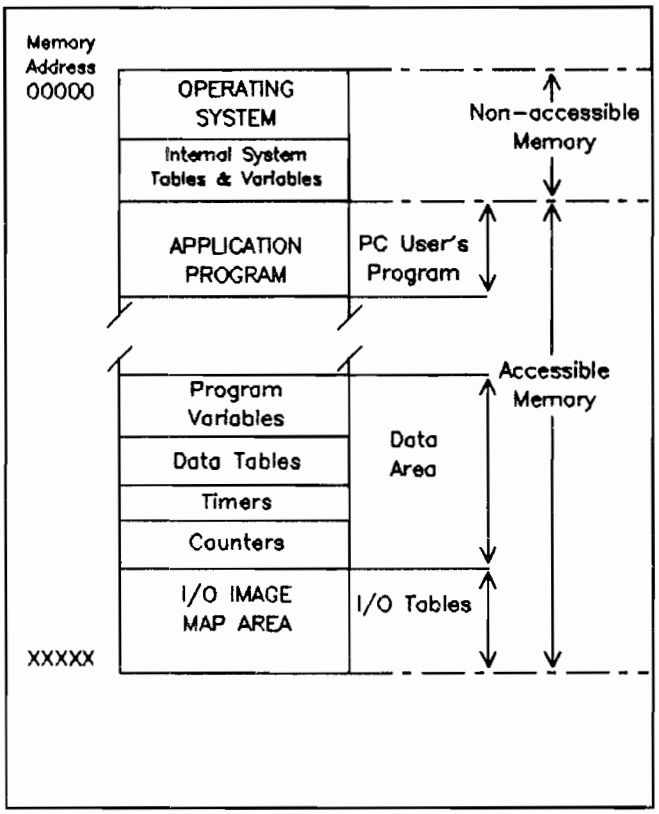



Figure 3.3 Example of a P/C Memory Map

## 3.4 P/C COMMUNICATION FUNCTIONS

For communicating with external devices (computers, peripherals, other P/Cs, etc.) the P/Cs can use either their main processor (e.g. Allen-Bradley) or a separate unit which shares the memory with the main processor. The communication functions they can perform are of type read, write or device control and are differentiated according to the memory area which is addressed. These areas can be classified in three main types:

- Data (timers, counters, I/O points, etc.)
- Programs
- Device status

For PCIF/1000, therefore, any P/C is the equivalent of a LOGICAL P/C. The correspondence between this logical P/C and the real P/C memory is defined by the user during the PCIF/1000 configuration phase.

# Chapter 4
# P/C ACCESS ROUTINES

## 4.1 GENERAL

### 4.1.1 Introduction

This chapter describes how a programmer may use PCIF/1000 with application programs.

The programming interface between application programs and the P/C management facilities of PCIF/1000 is a set of high level routines collectively known as the P/C Access Routines or PCARs.

Once the installation and configuration procedures are complete, the PCARs allow application programs to issue requests and commands to specific P/Cs, and to retrieve information from these P/Cs. The complete range of facilities offered by the PCARs is given in section 4.2.

NOTE: This chapter describes all PCARs. Refer to your handler-specific manual for the PCARs supported by your particular interface.

### 4.1.2 Operating Principles

One of the results of the configuration process is the definition of several maps and reference tables, to be used by the PCIF monitor at run-time.

a) Logical P/C Identification

A unique logical P/C identifier is assigned to each specific combination of a data highway and a physical P/C along this highway.

b) Logical P/C Memory

The logical P/C memory map defines for each logical P/C the various data areas (registers, counters, etc..) and program areas in this P/C, according to the logical P/C address to be used by the PCARs.

c) Logical P/C State Table

The P/C state table includes static information defined at configuration-time, plus dynamic information which is alterable by the PCARs. For each P/C, this state table lists the time-out value (static), maximum allowable number of waiting requests (static), P/C capabilities (static), lock/unlock state (dynamic) and connect/disconnect state (dynamic).

*Time-out Value:* The time-out value is the maximum time that can elapse between the moment a request is accepted by PCIF/1000 and the request's completion. Uncompleted requests that exceed the time-out value are destroyed by PCIF/1000, and a message is returned to the application program. If a reply to a request is received after exceeding the time-out value, it is ignored by PCIF/1000. This value is user definable and initialized at configuration-time.

*Number of Waiting requests:* This number specifies the maximum number of requests which can be queued on each P/C. If this number is reached for a P/C, all additional requests for this P/C are rejected by PCIF/1000 and a message is sent back to the user program (as a completion status for the request). This value is user definable and initialized at configuration-time.

*P/C Capabilities:* Certain requests may be not be allowed for certain P/Cs, as there may not be the required function in the real P/C. For example, a P/C may not be capable of sending unsolicited interrupts. Similarly, the capability to stop a P/C may be restricted to those operators who know a security code.

*P/C Lock/Unlock:* Using this capability a programmer is able to reserve a PC for private use. This P/C becomes unavailable for the other application programs until it is unlocked by the application program that first locked it.

*PC Connect/Disconnect:* If needed, a programmer is able to connect/disconnect a P/C from a PCIF/ point of view. No request (except a connect) can be sent to a disconnected P/C.

Each static value is specified at configuration time. When a P/C access routine is called by an application program, the PCIF/1000 monitor verifies that it may process this request and will then issue the appropriate request to the P/C and/or highway handler, which in turn interfaces with the physical P/C station. The reverse path is followed when the P/C replies to an application program.

## 4.1.3 Logical versus Physical P/C Status

From the application programmers point of view a P/C can have two states:

- The physical status is defined by the condition of the P/C processor. The P/C may be in run-mode, programming mode, stopped or down. This physical status is interpreted by the specific P/C handlers and may be obtained by the program using the PC_PCSTAT routine.

- The logical status describes the PC's condition from a PCIF/1000 point of view, such as whether the P/C is connected/disconnected, locked/unlocked, enabled/disabled for unsolicited requests, etc. The program may obtain the logical status of a specific P/C by using the PC_SYSTAT routine.

These two states are not related to one another. Any PCAR call modifying one state will not have any effect on the other.

## 4.1.4 Simultaneous Access by Different APs

The PCIF monitor can deal with PCARs arriving from different application programs, as each program must start a dialog with PCIF by using the routine PCIF_OPEN. This prompts the PCIF monitor to allocate the calling program a unique class number which is used to identify all calls coming from this program. Furthermore, the class number is removed from an application program when it finishes the PCIF dialog with the routine PCIF_CLOSE.

## 4.1.5 PCIF Transparent Access

Direct access to a P/C is possible via the PCIF system by-pass provided by the P/C Access Routine PC_TRANS. The use of this routine requires an in-depth knowledge of the P/C to be addressed, and allows the application program to generate requests in accordance with the target PC's protocol.

The capability of a P/C to receive a PC_TRANS request is defined at configuration time.

## 4.1.6 Using the Wait/No-Wait Option

When using the PCARs the programmer has the choice of waiting for the reply to each P/C request, and therefore suspending the application program until the reply arrives, or taking the no-wait option whereby the application program proceeds without the reply, retrieving the answer at a later time. Note that if the application program includes more than one data highway, the overall system throughput can be greatly increased by issuing all reads together without wait, and looping to read the replies.

If the programmer wants to take the no-wait option, an ACCESS KEY must first be obtained from the PCIF monitor by using the PC_GETKEY routine. The returned access key value will be unique each time an application program calls PC_GETKEY. The number of available access keys is defined at configuration time, but is limited to a maximum of 16 per application program.

A P/C Access Routine can be issued with "no-wait" by the insertion of the access key value into the routine's KEY parameter. This parameter may also be set to zero if the routine must wait for the reply.

The subsequent reply to a no-wait routine is found by using the PC_ENQUIRY routine with the relevant access key value inserted in the CONTKEY parameter.

## 4.1.7 Reading Unsolicited Requests

The application program may allow any unsolicited (i.e. unprovoked) requests arriving from specific P/Cs to be read. This is also achieved with a unique access key value obtained by a PC_GETKEY routine. Subsequently, a specific P/C is enabled to issue unsolicited requests by receiving a PC_ENUNSOL routine containing this access key value. If any unsolicited requests are then issued by the P/C, they can be retrieved by using a PC_ENQUIRY call containing the same access key.

NOTE: The same access key value is usually sufficient to retrieve answers to requests "without wait" plus any unsolicited requests.

## 4.1.8 Pascal and FORTRAN Compatibility

The PCARs can be called by application programs written in either FORTRAN 77 or PASCAL.

Table 4.1 gives the PCAR parameter data types used and their corresponding data type declaration in both languages.

| PCAR | FORTRAN 77 | PASCAL/1000 | Length |
|---|---|---|---|
| Integer | INTEGER | -32768..32767 | 16 bits |
| Long Integer | INTEGER*4 | INTEGER | 32 bits |
| String | CHAR | PACKED STRING | 8 bits |

Table 4.1 Pascal and FORTRAN Parameter Types

Note that in this chapter the term "integer" always refers to a 16 bit integer (-32768..32767) unless stated otherwise.

## 4.1.9 Linking an Application Program with PCARs

When written and compiled, an application program should be linked with the PCIF/1000 library ( PCLIB or PCLBC). As this library is written in Pascal, the following commands should be given to the link command file:

```
Non-CDS Application Program     CDS Application Program
LI,/LIBRARIES/PASCAL.LIB        LI,/LIBRARIES/PASCAL_CDS.LIB   <Pascal Library>
LI,$PCLIB::<crn>                LI,$PCLBC::<crn>               <PCIF Library>
....                            ....
....                            ....

....                            ....
RE,------                       RE,------                      <User's
EN                              EN                              relocatable>
```

## 4.1.10 Handling PCIF Library Errors in the Application Program

When $PCLIB or $PCLBC is called from an application program written in Pascal, the programmer can refer to:

"PASCAL/1000 Reference Manual"
Part Number: 92833-90005

for using either the Pascal provided error management (Pas.ErrorCatcher) or a user-defined routine.

When $PCLIB or $PCLBC is used from an application program written in FORTRAN, the user may not want to use the Pascal provided error management to control program size. If this is the case, an entry point must be added in the program and called

Pas.ErrorCatcher

where Pascal errors will branch. Then the user can either STOP or take any desired action. If $PCLIB or $PCLBC is the only PASCAL part of the user program, branching into "Pas.ErrorCatcher" means that there is a failure inside $PCLIB or $PCLBC. The first action to be taken in this case is to verify that the $PCLIB or $PCLBC version used to link the application program is the appropriate version (revision code, integrity, etc...).

## 4.2 SUMMARY OF PCARs

All the PCARs are listed below and classified by their function. Refer to section 4.3 for a detailed analysis of each routine.

Read/Write Data

```
PC_READD (STAT,TAG,KEY,PC,BUFFR,LENGR,PCADR)
PC_WRITED (STAT,TAG,KEY,PC,BUFFW,LENGW,PCADR)
```

Read/Write Program

```
PC_READP (STAT,TAG,KEY,PC,BUFFR,LENGR,PCADR)
PC_WRITEP (STAT,TAG,KEY,PC,BUFFR,LENGW,PCADR)
```

Lock/Unlock P/C Request

```
PC_LOCK (STAT,PC)
PC_UNLOCK (STAT,PC)
```

Connect/Disconnect P/C Request

```
PC_CONNECT (STAT,TAG,KEY,PC)
PC_DISC (STAT,PC,SECURITY_CODE)
```

P/C Physical Status Request

```
PC_PCSTAT (STAT,TAG,KEY,PC,BUFFR)
PC_IDENT (STAT,TAG,KEY,PC,BUFFR,LENGR)
```

P/C Logical Status Request

```
PC_SYSTAT (STAT,PC,BUFFR)
```

Cancel Request

```
PC_CANCEL (STAT,PC,OLDTAG,TYPEC)
```

Start/Stop P/C Request

```
PC_START (STAT,TAG,KEY,PC)
PC_STOP (STAT,TAG,KEY,PC)
```

Enquiry

```
PC_ENQUIRY(STAT,OLDSTAT,OLDTAG,CONTKEY,PC,BUFFR,LENGR,TYPER,LOGR)
```

Request in Transparent Mode

```
PC_TRANS (STAT,TAG,KEY,PC,SUBFCT,BUFFW,LENGW,BUFFR,LENGR)
```

Access Key Operation

```
PC_GETKEY (STAT,AKEY)
PC_RELKEY (STAT,AKEY)
```

Enable/Disable Unsolicited Requests

```
PC_ENUNSOL (STAT,PC,AKEY)
PC_DIUNSOL (STAT,PC)
```

PCIF/1000 Open and Close

```
PCIF_OPEN (STAT)
PCIF_CLOSE (STAT)
```

Error Message Management

```
PCIF_ERROR (STAT,BUFFR,LENGTH)
```

# 4.3 DETAILED DESCRIPTION OF THE ROUTINES

## 4.3.1 General Information

With the exception of the access routines PCIF_OPEN, PCIF_CLOSE and PCIF_ERROR, each routine is listed in this section in alphabetical order. All the routines are described using the following format:

- Name of the routine
- Function of the routine
- Calling sequence, e.g. PC_EXAMPLE (P1,P2,...Pn)
- Parameter Analysis
- Additional Comments (where applicable).

All the parameters (P1,P2,...Pn) must be supplied for every routine.

The number of parameters varies with the routine used, and each is described in the order they appear in the calling sequence.

## 4.3.2 Entry versus Return Parameters

With the exception of PCIF_OPEN and PCIF_CLOSE where only one return parameter is required, each routine divides into "entry" and "return" parameters.

Entry parameter values are defined by the application program before a call to the routine is made, whereas return parameter values are returned by the routine to the application program.

For easier reference, the return parameters are underlined in the calling sequence.

In a Pascal program, return parameters should be declared with VAR.

## 4.3.3 Status Parameter

With all the PCARs the first parameter P1 is a return parameter called STAT. This indicates whether the routine's call was completed successfully. It is then set to 0.

A positive returned value indicates an error condition that in every case aborts the call and returns control to the application program.

A returned value equal to -1 indicates that a no-wait type of request was made and has been accepted. The status of the request itself is returned later (with the Enquiry call) and will be a positive value or 0.

Refer to Appendix A for a list of all the errors and their corresponding return code values.

# PCIF_OPEN

Allows the application program to initialize a dialog with PCIF/1000. Upon receiving this request, PCIF/1000 asks RTE to allocate some resources for its internal use, such as a class number.

## Calling Sequence

PCIF_OPEN (STAT)

## Parameters

**STAT**

Type: Integer. Value: 0 or a non-zero value.

This return parameter value is set to zero if the call is completed successfully, otherwise to a non-zero value if any error occurs. The possible non-zero values are: (also refer to Appendix A)

| | |
|---|---|
| 17 | Contact with PCIF monitor lost. |
| 20 | Temporary memory shortage for request transmission. |
| 22 | RTE EXEC error while dialoging with PCIF monitor. |
| 23 | PCIF_OPEN already made by this program. |
| 24 | Lacking RTE resources for using PCIF. |
| 25 | Maximum number of possible OPENs exceeded. |

## Comments

An application program can only issue one PCIF_OPEN. If it issues another before a PCIF_CLOSE is made then a STAT value of 23 is returned.

If a program was previously executing with the same ID segment name (and the same session number in case of RTE-A with VC+), and has never closed the dialog with PCIF/1000 monitor, the PCIF_OPEN will do a PCIF_CLOSE of the oldest program, and then do a PCIF_OPEN for the new execution. This can only occur if the application was ended (normally or abnormally) without doing a PCIF_CLOSE. In this situation, the monitor generates the warning, MK050 (see Appendix A for details).

# PCIF_CLOSE

Terminates the dialog between PCIF/1000 and the application program. This routine also disables any outstanding P/C requests and releases all the RTE resources that were allocated for the application program.

## Calling Sequence

PCIF_CLOSE (STAT)

## Parameters

**STAT**

Type: Integer. Value: 0 or a non-zero value.

This return parameter value is set to zero if the call is completed successfully, otherwise to a non-zero value if any error occurs. The possible non-zero values are: (also refer to Appendix A)

16 Missing PCIF_OPEN for this program
17 Contact with the PCIF monitor lost.
20 Temporary memory shortage for request transmission.
22 RTE EXEC error while dialoging with PCIF monitor.

## Comments

PCIF_CLOSE routine unlocks all P/Cs locked by this application program, disables all unsolicited enabled P/Cs (for this program) and also releases all access keys previously allocated to this application program (i.e. a PC_GETKEY was made but no corresponding PC_RELKEY was made). When PCIF_CLOSE is accepted (STAT=0) then any subsequent PCAR will be refused by the monitor except PCIF_OPEN.

# PCIF_ERROR

This is an extra routine provided to allow the application program to transform a completion status parameter from a 16 bit integer value into an ASCII string. The ASCII strings corresponding to status codes are found in an FMGR file.

## Calling Sequence

PCIF_ERROR (STAT,BUFFR,LENGTH)

## Parameters

### STAT

Type: Integer. Value: 0 or a non-zero value.

For this routine, STAT is an entry parameter containing the completion status code as provided from the P/C Access Routine. A complete list of these is given in Appendix A.

### BUFFR

Type: Integer/string array. Value: HP or user provided.

This return parameter will contain the ASCII message corresponding to the STAT value.

### LENGTH

Type: Integer. Value: Between 0 and 80.

This is a return parameter that will contain the length (in bytes) of the message. Any remaining space after the message and up to 80 characters is padded with blanks.

## Comments

The file used to store the messages is called "PCMSG. If the routine is unable to find the message corresponding to the current value of STAT, the following character string will be put in BUFFER : PCxxx . Where xxx is the last three digits of the value of STAT. e.g. if STAT = 3 then xxx = 003.

If the value of STAT is negative or greater than 999 then the following message is given:

PC999 invalid PCIF ERROR value

# PC_CANCEL

Allows the application program to flush either one or all of its requests previously sent to a specified P/C. Only those requests that are uncompleted at the time of this call are cancelled. A completed request and associated reply are unaffected.

Calling Sequence

PC_CANCEL (STAT,PC,OLDTAG,TYPEC)

Parameters

**STAT**

Type: Integer. Value: 0 or a non-zero value.

This return parameter value is set to zero if the call is completed successfully, otherwise to a non-zero value if any error occurs. Possible non-zero values for this routine are: (also refer to Appendix A)

16 Missing PCIF_OPEN for this program.
17 Contact with PCIF monitor lost.
20 Not enough SAM.
21 Not enough EMA.
22 RTE EXEC error while dialoging with PCIF monitor.
33 Disconnected P/C.
35 Unknown P/C logical identifier.

**PC**

Type: Integer. Value: Configuration dependent.

This entry parameter is the logical identifier of a physical P/C station. The parameter value is the one defined for the target P/C at configuration-time.

**OLDTAG**

Type: Integer. Value: Defined by TAG parameter.

This entry parameter is only significant when used with a TYPEC different from zero. It is a user defined tag for selective cancel.

**TYPEG**

Type: Integer. Value: zero or 1,

For this entry parameter, the value zero indicates that all the waiting P/C requests are cancelled. Other values specify that only the requests having a TAG equal to the OLDTAG parameter are to be cancelled. There may be one or more requests made by the application program with the same TAG.

Comments

Cancelled requests will never send a reply to the application program. Be careful of cancelling requests made with "no-wait", as PC_CANCEL MAY or MAY NOT flush them. This is because a reply may be pending on an access key and therefore will not be affected.

WARNING

If a PC_CANCEL is issued for a request while this request is being processed by PCIF/1000, the physical effect of the cancellation cannot be guaranteed. The cancelled request may have already been partially treated.

# PC_CONNECT

Allows exchange of information between any application program and a logical P/C.

Calling Sequence

PC_CONNECT (STAT,TAG,KEY,PC)

Parameters

**STAT**

Type: Integer. Value: 0 or a non-zero value.

This return parameter value is set to zero if the call is completed successfully, otherwise to a non-zero value if any error occurs. Possible non-zero values for this routine are: (also refer to Appendix A)

-1 Request accepted but not completed (no-wait call).
16 Missing PCIF_OPEN for this program.
17 Contact with PCIF monitor lost.
20 Not enough SAM.
21 Not enough EMA.
22 RTE EXEC error while dialoging with PCIF monitor.
35 Unknown P/C logical identifier.

**TAG**

Type: Integer. Value: User defined.

This entry parameter has a user defined value which identifies this particular request. It is never modified by PCIF/1000 and is carried along with the reply, for instance it is returned in a PC_ENQUIRY call as OLDTAG.

**KEY**

Type: Integer. Value: 0 or provided by PC_GETKEY.

This entry parameter should be set to zero if the routine is to be used with the wait option. If the no-wait option is used, the value entered will signify on which access key the answer can be found at a later stage with a PC_ENQUIRY call. This access key value should have been allocated with a previous PC_GETKEY call.

PC

Type: Integer. Value: Configuration dependent

This entry parameter is the logical identifier of a physical P/C station. The parameter value is the one defined for the target P/C at configuration-time.

Comments

When PCIF/1000 monitor is started, all the P/Cs found in the configuration file are in the disconnected state. Once a PC_CONNECT has been successfully made, another PC_CONNECT request for this P/C will still be successful, and will not change the current status of the P/C.

# PC_DISC

Logically disconnects the specified P/C from all application programs making this P/C unavailable for all requests except connect and status enquires.

## Calling Sequence

PC_DISC (STAT,PC,SECURITY_CODE)

## Parameters

### STAT

Type: Integer. Value: 0 or a non-zero value.

This return parameter value is set to zero if the call is completed successfully, otherwise to a non-zero value if any error occurs. Possible non-zero values for this routine are: (also refer to Appendix A)

16 Missing PCIF_OPEN for this program.
17 Contact with PCIF monitor lost.
20 Not enough SAM.
21 Not enough EMA.
22 RTE EXEC error while dialoging with PCIF monitor.
32 Locked P/C.
33 Disconnected P/C.
35 Unknown P/C logical identifier.
43 Invalid PCIF security code.

### PC

Type: Integer. Value: Configuration dependent.

This entry parameter is the logical identifier of a physical P/C station. The parameter value is the one defined at configuration time for the target P/C.

### SECURITY_CODE

Type: Integer. Value: Configuration dependent.

This entry parameter must be equal to the PCIF/1000 security code for this configuration as defined at configuration-time (see screen 11 of configuration).

Comments

The disconnect request has an immediate action when accepted. To be accepted the SECURITY_CODE parameter must be correct and the P/C should not be LOCKED to any application program including the one making the PC_DISC.

The requests waiting to be processed on this P/C are completed with a status value of 33.

When disconnected, if the P/C is enabled for unsolicited requests it will be disabled.

Therefore a disconnected P/C is:

- Unsolicited request disabled.
- Unlocked.

# PC_DIUNSOL

Disables all unsolicited requests from the specified P/C to the application program.

## Calling Sequence

PC_DIUNSOL (STAT,PC)

## Parameters

### STAT

Type: Integer. Value: 0 or a non-zero value.

This return parameter value is set to zero if the call is completed successfully, otherwise to a non-zero value if any error occurs. Possible non-zero values for this routine are: (also refer to Appendix A)

16 Missing PCIF_OPEN for this program.
17 Contact with PCIF monitor lost.
20 Not enough SAM.
21 Not enough EMA.
22 RTE EXEC error while dialoging with PCIF monitor.
32 Locked P/C.
33 Disconnected P/C.
35 Unknown P/C logical identifier.
36 P/C does not have the capability to perform user request.
41 P/C not previously enabled for UNSOL with this program.

### PC

Type: Integer. Value: Configuration dependent.

This entry parameter is the logical identifier of a physical P/C station. The parameter value is the one defined at configuration time for the target P/C.

## Comments

This request can only be made by the same application program as the one which made the PC_ENUNSOL request.

This PCAR call will immediately disable unsolicited requests from this P/C, but all previous unsolicited request for this P/C will stay in the associated access key queue, unless they were removed from the queue with a PC_ENQUIRY or PC_RELKEY request.

# PC_ENQUIRY

Allows the retrieval of replies from previous "no-wait requests", i.e. requests made with their KEY parameter not equal to zero. Also the retrieval of unsolicited data or requests from specified P/Cs, which have previously received a PC_ENUNSOL with the same KEY parameter.

## Calling Sequence

```
PC_ENQUIRY (STAT,OLDSTAT,OLDTAG,CONTKEY,PC,BUFFR,LENGR,
                                          TYPER,LOGR)
```

## Parameters

### STAT

Type: Integer. Value: 0 or a non-zero value.

This return parameter value is set to zero if the call is completed successfully, otherwise to a non-zero value if any error occurs. Possible non-zero values for this routine are: (also refer to Appendix A)

| | |
|---|---|
| 2 | Length of buffer too long or zero. |
| 3 | Invalid length unit. |
| 16 | Missing PCIF_OPEN for this program. |
| 17 | Contact with PCIF monitor lost. |
| 22 | RTE EXEC error while dialoging with PCIF monitor. |

NOTE: If STAT parameter does not equal 0, all other return parameters have no meaning.

### OLDSTAT

Type: Integer. Value: 0 or a non-zero value.

This return parameter contains the status of the retrieved request or reply. Its value will be zero if the retrieved request was completed successfully, otherwise it can be any non-zero value as listed in Appendix A or in each corresponding section.

This return parameter is only significant if TYPER is not equal to 0.

### OLDTAG

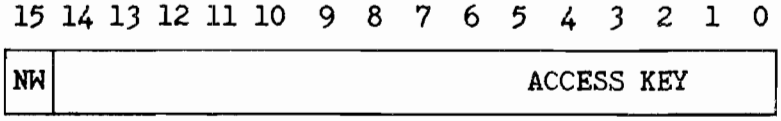
Type: Integer. Value: Defined by TAG parameter.

This return parameter was previously defined as the TAG parameter with the P/C request that has been retrieved, and therefore identifies this request.

This return parameter has only a meaning if TYPER is not equal 0.

**CONTKEY**

Type: Integer. Value: wait bit + access key (provided by PC_GETKEY).




This entry parameter is set to the relevant access key value (ACCESS KEY) as shown above. NW is a no-wait bit. When set (NW=1), control is returned immediately to the application program if no information is currently stored in this associated access key queue. Information is returned in the STAT word describing the action taken by the PCIF subsystem. This information must be checked by the application program. If NW is set to 0 a return will not come to the application program until the request PC_ENQUIRY is completed.

Note that replies to "no-wait" requests may be made in a different order than the request's emission order. The access key keeps reply order, not emission order.

**PC**

Type: Integer. Value: Retrieved request dependent

This return parameter is the logical identifier of the P/C to which the retrieved request was previously sent.

**BUFFR**

Type: Integer/byte/bit array. Value: data retrieved.

This return parameter is used to store either the data, if there is any associated with an unsolicited request, or a reply to a no-wait request.

The data area indicated by BUFFR is ONLY modified if STAT=0, OLDSTAT=0 AND TYPER is not equal 0.

Note: For more PC-specific information, see your handler reference manual.

**LENGR**

Type: Integer. Value: length and type of BUFFR.

This entry parameter contains the maximum length of BUFFR that can be transferred to the application program, and is coded as follows:

| 15 14 | 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |
|---|---|
| BI | LENGTH OF DATA TO RECEIVE |

where: BI = 0, the length is expressed in 16 bit words;
= 1, the length is expressed in 8 bit bytes;
= 2, the length is expressed in bit units.

LENGTH OF DATA TO RECEIVE is the maximum number of units expected to be received from the P/C, in accordance with the BI information.

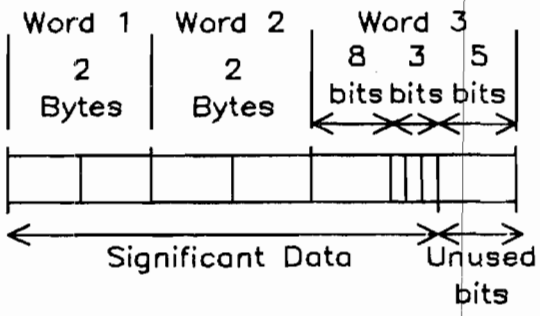
Note also that if the length is expressed in bits or bytes the result always an EVEN number of bytes, and the possible remaining bits/bytes are undefined. For example,

with LENGR = | 2 | 43 | 43=(5bytes*8)+3bits

then 43 bits = 2.69 words are needed.

This must be rounded up, however, to the nearest word. At least three words, therefore, are required in BUFFR to store the returned data.

BUFFR would then look like this:




Refer to the specific handler reference manual for more information.

**TYPER**

Type: Integer. Value: zero, >0 or <0.

This is a return parameter that defines what type of message has been retrieved, and is coded as follows:

Zero = No reply or unsolicited request found.

> 0 = The returned message is a reply to a no-wait request.
Listed below are the possible values:

| | |
|---|---|
| 1 PC_READD | 16 PC_START |
| 2 PC_WRITED | 17 PC_STOP |
| 3 PC_READP | 20 PC_CONNECT |
| 4 PC_WRITEP | 22 PC_PCSTAT |
| 8 PC_TRANS | 30 PC_IDENT |

< 0 = The returned message is an unsolicited request.
Listed below are the possible values:

| |
|---|
| -1 Unsolicited PC_READD |
| -2 Unsolicited PC_WRITED |
| -22 Unsolicited PC_PCSTAT |

**LOGR**

Type: Integer. Value: Positive or zero.

This return parameter provides the returned message data length. It is coded in BYTE units. This parameter is only significant if TYPER is not equal to 0.

Comments

The user must analyze the validity of the reply (or unsolicited request) found by the PC_ENQUIRY routine.

When requests are made with a CONTKEY parameter not equal to zero or when unsolicited requests arrive from P/Cs, the replies or unsolicited request are stored in memory in an area associated with the access key. They are kept in reply chronological order (and not in request order). This information is purged on PC_ENQUIRY which gets the oldest reply or unsolicited request. It may also be purged with a PC_RELKEY (on this KEY) or on PCIF_CLOSE (for this application program). Therefore, to avoid using too much memory with these replies and unsolicited requests, it is a good idea to make regular PC_ENQUIRY calls during the run-time period of PCIF/1000.

When a PC_ENQUIRY request is issued on an access key and the queue is not empty, and if the retrieved information has associated data (from a PC_READD, for example), then this data will also be transferred into the BUFFR of the the PC_ENQUIRY request. Therefore, one PC_ENQUIRY call gets all the information pertaining to a no-wait reply or unsolicited request.

# PC_ENUNSOL

Enables unsolicited requests issued by a physical P/C, logically identified as a target P/C, to be sent into an access key provided in the PC_ENUNSOL.

## Calling Sequence

PC_ENUNSOL (STAT,PC,AKEY)

## Parameters

### STAT

Type: Integer. Value: 0 or a non-zero value.

This return parameter value is set to zero if the call is completed successfully, otherwise to a positive non-zero value if any error occurs. Possible non-zero values for this routine are: (also refer to Appendix A)

| | |
|---|---|
| 16 | Missing PCIF_OPEN for this program. |
| 17 | Contact with PCIF monitor lost. |
| 18 | Illegal access key. |
| 20 | Not enough SAM. |
| 21 | Not enough EMA. |
| 22 | RTE EXEC error while dialoging with PCIF monitor. |
| 32 | Locked P/C. |
| 33 | Disconnected P/C. |
| 35 | Unknown P/C logical identifier. |
| 36 | P/C does not have the capability to perform user request. |
| 40 | P/C already enabled for UNSOL with this program. |
| 42 | P/C already enabled for UNSOL with another program. |
| 128..255 | P/C handler detected error. See Appendix A. |
| 256..511 | Highway handler detected error. See Appendix A. |
| 512.. | Specific P/C manufacturer error. |

No action is made by the call unless STAT is equal to zero.

### PC

Type: Integer. Value: Configuration dependent.

This entry parameter is the logical identifier of a physical P/C station. The parameter value is the one defined at configuration time for the target P/C.

**AKEY**

Type: Integer. Value: Provided by PC_GETKEY.

This entry parameter must contain an access key assigned to the application program by a previous PC_GETKEY request. PCIF/1000 associates this key with a data area for storing unsolicited requests from the specified P/C station. This key will be used by a PC_ENQUIRY (in the CONTKEY parameter) to retrieve the unsolicited requests.

Comments

A PC_ENUNSOL request can only be performed on a P/C which supports the "EU" capability (See PC_SYSTAT request explanation).

If a physical P/C emits an unsolicited request and the corresponding logical P/C is not enabled for unsolicited requests, then the unsolicited requests are FLUSHED by PCIF/1000.

A P/C can be "unsolicited request enabled" for only one application program at any given time.

# PC_GETKEY

Obtains ownership of an access key which will be associated with memory areas that are used to store data. This data may be unsolicited requests, or replies to no-wait requests.

## Calling Sequence

PC_GETKEY (STAT,AKEY)

## Parameters

### STAT

Type: Integer. Value: 0 or a non-zero value.

This return parameter value is set to zero if the call is completed successfully, otherwise to a non-zero value if any error occurs. Possible non-zero values for this routine are: (also refer to Appendix A)

16 Missing PCIF_OPEN for this program.
17 Contact with PCIF monitor lost.
20 Not enough SAM.
21 Not enough EMA.
22 RTE EXEC error while dialoging with PCIF monitor.
24 Not enough class numbers available in RTE.
39 Maximum number of available access keys exceeded.

### AKEY

Type: Integer. Value: A class number.

This return parameter contains the access key assigned by PCIF/1000 to this PC_GETKEY request. PCIF/1000 associates this key with a data area for storing unsolicited requests from the specified P/C station, plus replies to other P/C requests made with the no-wait option (via their parameter KEY in each appropriate call).

This parameter has a significant value only if STAT = 0.

## Comments

It is up to the user to remember the allocation of access keys used. The maximum number per run-time session is determined at P/C configuration-time; however, for any application program it may never be more than 16 associated access keys at any given time.

# PC_IDENT

Allows the application program to identify the brand name, model number, and physical station ID of the target P/C.

## Calling Sequence

PC_IDENT (STAT,TAG,KEY,PC,BUFFR,LENGR)

## Parameters

### STAT

Type: Integer. Value: 0 or non-zero value.

This return parameter value is set to zero if the call is completed successfully, otherwise to a non-zero value if any error occurs. Possible non-zero values for this routine are: (also refer to Appendix A)

| | |
|---|---|
| -1 | Request accepted but not completed (no wait call). |
| 16 | Missing PCIF_OPEN for this program. |
| 17 | Contact with PCIF monitor lost. |
| 20 | Not enough SAM. |
| 21 | Not enough EMA. |
| 22 | RTE EXEC error while dialoging with PCIF monitor. |
| 32 | Locked P/C. |
| 33 | Disconnected P/C. |
| 35 | Unknown P/C logical identifier. |
| 37 | Maximum number of waiting requests reached on this P/C. |
| 38 | Time-out. |
| 128..255 | P/C handler detected error. See Appendix A. |
| 256..511 | Highway handler detected error. See Appendix A. |
| 512.. | Specific P/C manufacturer error. |

### TAG

Type: Integer. Value: User defined.

This entry parameter has a user defined value which identifies this particular request. It is never modified by PCIF/1000 and is carried along with the reply; for instance, it is returned in PC_ENQUIRY call as OLDTAG.

**KEY**

Type: Integer. Value: 0 or provided by PC_GETKEY.

This entry parameter may be set to zero if this routine call is to be used with the wait option. If the no-wait option is used, the value entered will signify on which access key the answer can be found at a later stage with a PC_ENQUIRY call. This access key value should have been allocated with a previous PC_GETKEY call.

**PC**

Type: Integer. Value: Configuration dependent.

This entry parameter is the logical identifier of a physical P/C station. The parameter value is defined at configuration time for the target P/C.

**BUFFR**

Type: Integer. Value: P/C status dependent.

This return parameter is a 10-word buffer containing data that identifies the specified P/C. The identity is defined as follows:

| | 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |
|---|---|
| word 1 | VENDOR |
| 2 | MODEL NUMBER |
| 3 | STATION ID |
| 4 | //////////////////////////////////////// |
| 5 | //////////////////////////////////////// |
| 6 | //////////////////////////////////////// |
| 7 | //////////////////////////////////////// |
| 8 | //////////////////////////////////////// |
| 9 | //////////////////////////////////////// |
| 10 | //////////////////////////////////////// |

Where: VENDOR is the two-character mnemonic representing the brand of the target P/C (e.g. AB for Allen-Bradley). (See your handler reference manual for details.)

MODEL is the PC's model number (e.g. 584 for Gould-Modicon). (See your handler reference manual for details.)

STATION ID is the PC's physical station id.

// signifies that these bits have undefined values and are reserved for future use.

**LENGR**

Type: Integer. Value: Length of BUFFR.

This entry parameter defines the amount of data to be received from the P/C. For PC_IDENT this amount is always 10 words.

# PC_LOCK

Allows the application program to lock a target P/C and prevent access to this P/C by any other application programs.

## Calling Sequence

PC_LOCK (STAT,PC)

## Parameters

### STAT

Type: Integer. Value: 0 or a non-zero value.

This return parameter value is set to zero if the call is completed successfully, otherwise to a non-zero value if any error occurs. Possible non-zero values for this routine are: (also refer to Appendix A)

16 Missing PCIF_OPEN for this program.
17 Contact with PCIF monitor lost.
20 Not enough SAM.
21 Not enough EMA.
22 RTE EXEC error while dialoging with PCIF monitor.
32 Locked P/C.
33 Disconnected P/C.
35 Unknown P/C logical identifier.

The lock status of the P/C is not changed unless STAT is equal to zero.

### PC

Type: Integer. Value: Configuration dependent.

This entry parameter is the logical identifier of a physical P/C station. The parameter value is the one defined at configuration time for the target P/C.

## Comments

This request takes effect at the completion of any pending requests on the specified P/C.

If other application programs try to access the specified P/C after the PC_LOCK routine has been completed, the request will be refused and the PCIF monitor will return a message in the appropriate STAT parameter. "Other" application programs are those associated by different PCIF_OPEN calls.

# PC_PCSTAT

This routine allows the application program to obtain information on the physical status of a target P/C.

## Calling Sequence

PC_PCSTAT (STAT,TAG,KEY,PC,BUFFR)

## Parameters

### STAT

Type: Integer. Value: 0 or a non-zero value.

This return parameter value is set to zero if the call is completed successfully, otherwise to a positive non-zero value if any error occurs. Possible non-zero values for this routine are: (also refer to Appendix A)

| | |
|---|---|
| -1 | Request accepted but not completed (no-wait call). |
| 16 | Missing PCIF_OPEN for this program. |
| 17 | Contact with PCIF monitor lost. |
| 20 | Not enough SAM. |
| 21 | Not enough EMA. |
| 22 | RTE EXEC error while dialoging with PCIF monitor. |
| 32 | Locked P/C. |
| 33 | Disconnected P/C. |
| 35 | Unknown P/C logical identifier. |
| 37 | Maximum number of waiting requests reached on this P/C. |
| 38 | Time-out. |
| 128..255 | P/C handler detected error. See Appendix A. |
| 256..511 | Highway handler detected error. See Appendix A. |
| 512.. | Specific P/C manufacturer error. " |

### TAG

Type: Integer. Value: User defined.

This entry parameter has a user defined value which identifies this particular request. It is never modified by PCIF/1000 and is carried along with the reply, for instance it is returned in PC_ENQUIRY call as OLDTAG.

**KEY**

Type: Integer  Value: 0 or provided by PC_GETKEY.

This entry parameter may be set to zero if this routine call is to be used with the wait option. If the no-wait option is used, the value entered will signify on which access key the answer can be found at a later stage with a PC_ENQUIRY call. This access key value should have been allocated with a previous PC_GETKEY call.

**PC**

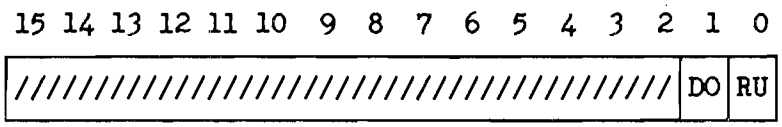
Type: Integer.  Value: Configuration dependent.

This entry parameter is the logical identifier of a physical P/C station. The parameter value is defined at configuration time for the target P/C.

**BUFFR**

Type: Integer.  Value: P/C status dependent.

This is a return parameter and is a 16 bit word that holds the returned data describing the physical status of the specified P/C. The status is defined as follows:




Where: RU = 0, the P/C processor is in run mode.
= 1, the P/C processor is not in run mode.

DO = 0, the P/C can accept a program download.
= 1, the P/C cannot accept a program download.

// signifies that these bits have undefined values and are reserved for future use.

The contents of BUFFR are not modified if the PC_PCSTAT call is made in no-wait mode or the STAT parameter is not equal to zero.
Refer to the specific handler manual for a PC-specific description of the status bits.

# PC_READD

Allows the application program to read data from a target P/C.

## Calling Sequence

PC_READD (STAT,TAG,KEY,PC,BUFFR,LENGR,PCADR)

## Parameters

### STAT

Type: Integer. Value: 0 or a non-zero value.

This return parameter value is set to zero if the call is completed successfully, otherwise to a non-zero value if any error occurs. Possible non-zero values for this routine are: (also refer to Appendix A)

| | |
|---|---|
| -1 | Request accepted but not completed (no-wait call). |
| 2 | Length of buffer to transmit too long or null. |
| 3 | Invalid length unit. |
| 16 | Missing PCIF_OPEN for this program. |
| 17 | Contact with PCIF monitor lost. |
| 18 | Illegal access key. |
| 19 | Illegal buffer address. |
| 20 | Not enough SAM. |
| 21 | Not enough EMA. |
| 22 | RTE EXEC error while dialoging with PCIF monitor. |
| 32 | Locked P/C. |
| 33 | Disconnected P/C. |
| 35 | Unknown P/C logical identifier. |
| 37 | Maximum number of waiting requests reached for this P/C. |
| 38 | Time-out |
| 128..255 | P/C handler detected error. See Appendix A. |
| 256..511 | Highway handler detected error. See Appendix A. |
| 512.. | Specific P/C manufacturer error. |

### TAG

Type: Integer. Value: User defined.

This entry parameter has a user defined value which identifies this particular request. It is never modified by PCIF/1000 and is carried along with the reply, for instance, returned with a PC_ENQUIRY call as OLDTAG.

**KEY**

Type: Integer  Value: 0 or provided by PC_GETKEY.

This entry parameter may be set to zero if this routine call is to be made with the wait option. If the no-wait option is used, the value entered will signify on which access key the answer can be found at a later stage with a PC_ENQUIRY call. This access key value should have been allocated with a previous PC_GETKEY call.

**PC**

Type: Integer.  Value: Configuration dependent.

This entry parameter is the logical identifier of a physical P/C station. The parameter value is the one defined at configuration time for the target P/C.

**BUFFR**

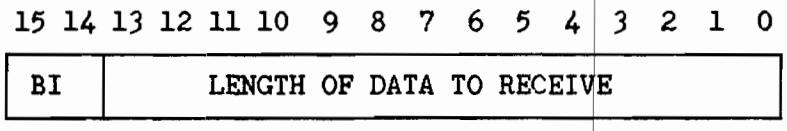
Type: Integer/byte/bit array.  Value: Data coming from P/C.

This return parameter is the application program defined buffer to hold the returned data. If STAT is not equal to zero, then no DATA is written into the BUFFR area. If the returned data is of byte or bit type, it is in BUFFR in a packed format. For example, if the data is in byte format, each word in BUFFR contains two bytes of data.

**LENGR**

Type: Integer. Value: Length and type of BUFFR.

This entry parameter defines the type and amount of data to be received from the P/C. This may be specified in a 16 bit word, a byte, or in bit units and is coded as follows:

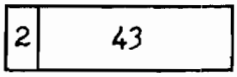



where: BI = 0, the length is expressed in 16 bit words;
= 1, the length is expressed in 8 bit bytes;
= 2, the length is expressed in bit units.

LENGTH OF DATA TO RECEIVE is the number of units to be received from the P/C, in accordance with the BI information.

Note that the length (in words) of the buffer area in the application program must not be less than the value of LENGR converted into 16 bit word units.

Note also that if the length is expressed in bit or byte the result always has an EVEN number of bytes, and the possible remaining bits/bytes are undefined. For example:

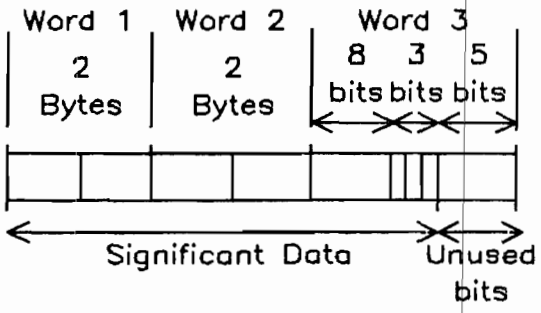
with LENGR = 

43=(5bytes*8)+3bits

then 43 bits = 2.69 words are needed.

This must be rounded up, however, to the nearest word. At least three words, therefore, are required in BUFFR to store the returned data.

BUFFR would then look like this:



**KEY**

Type: Integer Value: 0 or provided by GETKEY.

This entry parameter may be set to zero if this routine is to be used with the wait option. If the no-wait option is used, the value entered will signify on which access key the answer can be found at a later stage with a PC_ENQUIRY call. This access key value should have been allocated with a previous PC_GETKEY call.

**PC**

Type: Integer. Value: Configuration dependent.

This entry parameter is the logical identifier of a physical P/C station. The parameter value is the one defined at configuration time for the target P/C.

**BUFFR**

Type: Integer/byte/bit. Value: Data read from physical P/C.

This is a return parameter and is the application program defined buffer to hold the returned program.

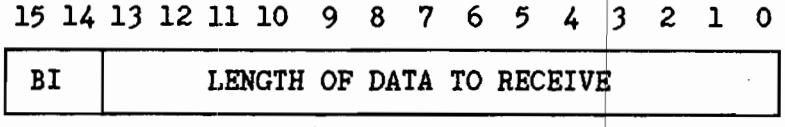
No data is put into this buffer unless STAT is equal to zero.

If the returned data is of byte or bit type, it is in BUFFR in a packed format. For example, if the data is in byte format, each word in BUFFR contains two bytes of data.

**LENGR**

Type: Integer. Value: length and type of BUFFR.

This entry parameter contains the maximum amount of the program to be received from the real P/C. It may be specified in a 16 bit word.




where: BI = 0, the length is expressed in 16 bit words.

LENGTH OF DATA TO RECEIVE is the number of words to be received from the P/C.

Note that the length (in words) of the buffer area in the application program must not be less than the length in words of LENGR.

**PCADR**

Type: 32 bit integer. Value: See specific P/C brand.

This entry parameter contains a value that allows the retrieval of the physical memory address of the program to be read from the specified real P/C. For the correspondence between this parameter value and the real memory address, see your handler-specific manual.

Comments

Refer to the relevant handler manual for specific information on using PC_READP with particular P/C brands.

# PC_RELKEY

Releases the assignment of ownership of an access key which was previously obtained by this application program using a PC_GETKEY request.

## Calling Sequence

PC_RELKEY (STAT,AKEY)

## Parameters

### STAT

Type: Integer. Value: 0 or a non-zero value.

This return parameter value is set to zero if the call is completed successfully, otherwise to a non-zero value if any error occurs. Possible non-zero values for this routine are: (also refer to Appendix A)

16 Missing PCIF_OPEN for this program.
17 Contact with PCIF monitor lost.
18 Illegal access key.
20 Not enough SAM.
21 Not enough EMA.
22 RTE EXEC error while dialoging with PCIF monitor.

If STAT is not equal to zero then the access key is not released, and none of the possible associated actions are made.

### AKEY

Type: Integer. Value: User defined

This entry parameter contains the access key allocated to the application program with a previous PC_GETKEY routine, and which is now required to be released. If some routines are currently associated with this access key, either replies to requests or unsolicited requests, and for which no PC_ENQUIRY has been made, they will be flushed from memory and it will not be possible to reach them by any means. It is also impossible to know their completion status.

## Comments

This call disables unsolicited requests from those P/Cs associated with the AKEY parameter. Therefore, any unsolicited requests sent from these P/Cs after the PC_RELKEY call will be lost, until a program sends another PC_ENUNSOL.

# PC_START

Physically starts the specified P/C. The use of this routine can be forbidden for some P/Cs, either they lack the capability to be remotely started or this capability was restricted at configuration-time.

## Calling Sequence

PC_START (STAT,TAG,KEY,PC)

## Parameters

### STAT

Type: Integer. Value: 0 or a non-zero value.

This return parameter value is set to zero if the call is completed successfully, otherwise to a non-zero value if any error occurs. Possible non-zero values for this routine are: (also refer to Appendix A)

| | |
|---|---|
| -1 | Request accepted but not completed (no-wait call). |
| 16 | Missing PCIF_OPEN for this program. |
| 17 | Contact with PCIF monitor lost. |
| 19 | Illegal buffer address. |
| 20 | Not enough SAM. |
| 21 | Not enough EMA. |
| 22 | RTE EXEC error while dialoging with PCIF monitor. |
| 32 | Locked P/C. |
| 33 | Disconnected P/C. |
| 35 | Unknown P/C logical identifier. |
| 36 | P/C does not have the capability to perform user request. |
| 37 | Maximum number of waiting requests reached for this P/C. |
| 38 | Time-out. |
| 128..255 | P/C handler detected error. See Appendix A. |
| 256..511 | Highway handler detected error. See Appendix A. |
| 512.. | Specific P/C manufacturer error. |

### TAG

Type: Integer. Value: User defined.

This entry parameter has a user defined value which identifies this particular request. It is never modified by PCIF/1000 and is carried along with the reply, for instance, returned in a PC_ENQUIRY call as OLDTAG.

**KEY**

Type: Integer Value: 0 or provided by GETKEY.

This entry parameter may be set to zero if this routine is to be used with the wait option. If the no-wait option is used, the value entered will signify on which access key the reply can be found at a later stage with a PC_ENQUIRY call. This access key value should have been allocated with a previous PC_GETKEY call.

**PC**

Type: Integer. Value: Configuration dependent.

This entry parameter is the logical identifier of a physical P/C station. The parameter value is the one defined at configuration time for the target P/C.

# PC_STOP

Physically stops the target P/C. The use of this routine may be forbidden for some P/Cs, either they lack the capability to be remotely stopped or this capability was restricted at configuration-time.

## Calling Sequence

PC_STOP (STAT,TAG,KEY,PC)

## Parameters

### STAT

Type: Integer. Value: 0 or a non-zero value.

This return parameter value is set to zero if the call is completed successfully, otherwise to a positive non-zero value if any error occurs. Possible non-zero values for this routine are: (also refer to Appendix A)

| | |
|---|---|
| -1 | Request accepted but not completed (no-wait call). |
| 16 | Missing PCIF_OPEN for this program. |
| 17 | Contact with PCIF monitor lost. |
| 18 | Illegal access key. |
| 20 | Not enough SAM. |
| 21 | Not enough EMA. |
| 22 | RTE EXEC error while dialoging with PCIF monitor. |
| 32 | Locked P/C. |
| 33 | Disconnected P/C. |
| 35 | Unknown P/C logical identifier. |
| 36 | P/C does not have the capability to perform user request. |
| 37 | Maximum number of waiting requests reached for this P/C. |
| 38 | Time-out. |
| 128..255 | P/C handler detected error. See Appendix A. |
| 256..511 | Highway handler detected error. See Appendix A. |
| 512.. | Specific P/C manufacturer error. |

### TAG

Type: Integer. Value: User defined.

This entry parameter has a user defined value which identifies this particular request. It is never modified by PCIF/1000 and is carried along with the reply, for instance, it is returned in a PC_ENQUIRY call as OLDTAG.

**KEY**

Type: Integer. Value: 0 or provided by GETKEY.

This entry parameter is set to zero if the routine is to be used with the wait option. If the no-wait option is used, the value entered will signify on which access key the reply can be found at a later stage with a PC_ENQUIRY call. This access key value should have been allocated with a previous PC_GETKEY call.

**PC**

Type: Integer. Value: Configuration dependent.

This entry parameter is the logical identifier of a physical P/C station. The parameter value is the one defined at configuration time for the target P/C.

# PC_SYSTAT

Allows the application program to obtain information on the logical state of a target P/C. The logical P/C state contains two sets of information, these are the system status and the system capability.

## Calling Sequence

PC_SYSTAT (STAT,PC,BUFFR)

## Parameters

### STAT

Type: Integer. Value: 0 or a non-zero value.

This return parameter value is set to zero if the call is completed successfully, otherwise to a positive non-zero value if any error occurs. Possible non-zero values for this routine are: (also refer to Appendix A)

16 Missing PCIF_OPEN for this program.
17 Contact with PCIF monitor lost.
20 Not enough SAM.
21 Not enough EMA.
22 RTE EXEC error while dialoging with PCIF monitor.
35 Unknown P/C logical identifier.

### PC

Type: Integer. Value: Configuration dependent.

This entry parameter is the logical identifier of a physical P/C station. The parameter value is the one defined at configuration time for the target P/C.

**BUFFR**

Type: Integer/bit array. Value: Request dependent.

This return parameter contains data that defines the system and capability status of the P/C station. It is only significant if STAT is equal to zero. The buffer is 16 bits long and is coded as follows:

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| WR | PR | TR | UN | ST | //// | //// | //// | UE | CO | LO | BU | //// | //// | //// | //// |

*System Status*

Where:

BU = 0, the station is free.
= 1, the station is busy, i.e. one or more P/C requests are pending.

LO = 0, the station is unlocked.
= 1, the station has been locked either by this or another AP.

CO = 0, the station is connected.
= 1, the station is unconnected.

UE = 0, the station is "unsolicited request enabled" for an AP.
= 1, the station is "unsolicited request disabled" for an AP.

*Capability Status*

Where:

ST = 0, any program has the capability to start/stop the P/C station.
= 1, no program can have this capability.

UN = 0, the P/C station can issue unsolicited requests.
= 1, the P/C station cannot issue unsolicited requests.

TR = 0, the transparent mode is allowed for this P/C by any program.
= 1, the transparent mode not allowed for this P/C by any program.

PR = 0, P/C programs can be written into the P/C station memory.
= 1, P/C programs cannot be written into the P/C station memory.

WR = 0, data can be written into the P/C station memory.
= 1, data cannot be written into the P/C station memory.

NOTE: The symbol //// denotes that these bits have undefined values and are reserved for future use.

# PC_TRANS

Allows the application program to interact with the specified P/C station in a PCIF transparent manner. This enables an application program to subsequently transmit all the possible functions of a P/C brand as described in the PC manufacturer's manual.

## Calling Sequence

PC_TRANS (STAT,TAG,KEY,PC,SUBFCT,BUFFW,LENGW,BUFFR,LENGR)

## Parameters

### STAT

Type: Integer. Value: 0 or a non-zero value.

This return parameter value is set to zero if the call is completed successfully, otherwise to a non-zero value if any error occurs. Possible non-zero values for this routine are: (also refer to Appendix A)

| | |
|---|---|
| -1 | Request accepted but not completed (no-wait call). |
| 1 | Invalid SUBFNC parameter. |
| 2 | Length of buffer to transmit is too long. |
| 3 | Invalid length unit. |
| 16 | Missing PCIF_OPEN for this program. |
| 17 | Contact with PCIF monitor lost. |
| 18 | Illegal access key. |
| 19 | Illegal buffer address. |
| 20 | Not enough SAM. |
| 21 | Not enough EMA. |
| 22 | RTE EXEC error while dialoging with PCIF monitor. |
| 26 | Requested function not implemented on current system. |
| 32 | Locked P/C. |
| 33 | Disconnected P/C. |
| 34 | Stopped P/C. |
| 35 | Unknown P/C logical identifier. |
| 36 | P/C does not have the capability to perform user request. |
| 37 | Maximum number of waiting requests reached for this P/C. |
| 38 | Time-out. |
| 128..255 | P/C handler detected error. See Appendix A. |
| 256..511 | Highway handler detected error. See Appendix A. |
| 512.. | Specific P/C manufacturer error. |

**TAG**

Type: Integer. Value: User defined.

This entry parameter has a user defined value which identifies this particular request. It is never modified by PCIF/1000 and is carried along with the reply, for instance, it is returned in a PC_ENQUIRY call as OLDTAG.

**KEY**

Type: Integer. Value: 0 or provided by GETKEY.

This entry parameter is set to zero if the routine is to be used with the wait option. If the no-wait option is used, the value entered will signify on which access key the answer can be found at a later stage with a PC_ENQUIRY call. This access key value will have been allocated with a previous PC_GETKEY call.

**PC**

Type: Integer. Value: Configuration dependent.

This entry parameter is the logical identifier of a physical P/C station. The parameter value is the one defined at configuration time for the target P/C.

**SUBFCT**

Type: Integer. Value: P/C brand and type dependent.

This entry parameter describes the P/C function to be performed by the target P/C, and is coded according to the requirements of the relevant P/C manufacturer. This parameter is specific P/C brand dependent.

**BUFFW**

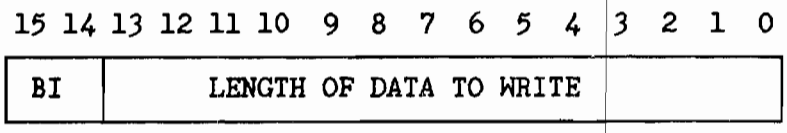
Type: Integer/byte/bit array. Value: Data to be sent to the P/C.

This entry parameter is the application program defined buffer. Its data is P/C type dependent and is composed as described for each specific P/C.

**LENGW**

Type: Integer. Value: length and type of BUFFW.

This entry parameter defines the amount and type of data to be written to the P/C, which may be in words (16 bit), bytes, or bit units, and is coded as follows:




where: BI = 0, the length is expressed in 16 bit words;
= 1, the length is expressed in 8 bit bytes;
= 2, the length is expressed in bit units.

LENGTH OF DATA TO WRITE is the number of units to be sent to the P/C, in accordance with the BI information.

**BUFFR**

Type: Integer/byte/bit array Value: Data read from P/C.

This return parameter is the application program defined buffer that holds a reply when PC_TRANS is called with the "wait" option selected (KEY = 0). If the call is made with no-wait (KEY = a valid access key) the result will be retrieved with a call to PC_ENQUIRY.

This parameter is not modified unless STAT is equal to zero.

**LENGR**

Type: Integer. Value: Length and type of BUFFR.

This entry parameter contains the maximum message length to be received from the P/C station by the application program and is coded as LENGW.

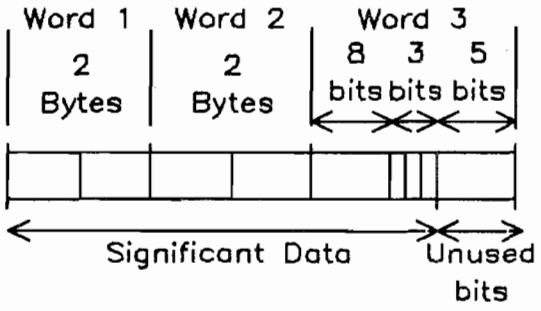
Note also that if the length is expressed in bits or bytes the result always has an EVEN number of bytes, and the possible remaining bits/bytes are undefined. For example:

with LENGR = | 2 | 43 | 43=(5bytes*8)+3bits

then 43 bits = 2.69 words are needed.

This must be rounded up, however, to the nearest word. At least three words, therefore, are required in BUFFR to store the returned data.

BUFFR would then look like this:




Comments

All request parameters are checked by the PCIF monitor to ensure PCIF/1000 compatibility, but they are not interpreted by the PCIF monitor.

Refer to your handler-specific manual for information on using PC_TRANS with your particular P/C brand.

********** **WARNING** **********

During PC_TRANS the PCIF/1000 monitor does not supervise the PC computer exchange, as with the other PCARs. Therefore using PC_TRANS requires a thorough knowledge of the target PC's function within a system, because this PC's physical status may be changed such that it is difficult or impossible to later process other PCIF/1000 functions.

# PC_UNLOCK

Allows the application program to unlock a P/C and therefore enable access to this P/C by any application programs.

## Calling Sequence

PC_UNLOCK (STAT,PC)

## Parameters

### STAT

Type: Integer. Value: 0 or a non-zero value.

This return parameter value is set to zero if the call is completed successfully, otherwise to a positive non-zero value if any error occurs. Possible non-zero values for this routine are: (also refer to Appendix A)

16 Missing PCIF_OPEN for this program.
17 Contact with PCIF monitor lost.
20 Not enough SAM.
21 Not enough EMA.
22 RTE EXEC error while dialoging with PCIF monitor.
32 Locked P/C.
33 Disconnected P/C.
35 Unknown P/C logical identifier.

The logical state of the P/C is unchanged unless STAT is equal to zero.

### PC

Type: Integer. Value: Configuration dependent.

This entry parameter is the logical identifier of a physical P/C station. The parameter value is the one defined at configuration time for the target P/C.

## Comments

Requests are allowed to continue if they have been made in the no-wait mode and are incomplete at the time of PC_UNLOCK. The purpose of the PC_LOCK/PC_UNLOCK pair is to guarantee that no other application program requests are intermixed with the current application program's requests.

# PC_WRITED

Allows the writing of data to the target P/C from the specified application program buffer.

Calling Sequence

PC_WRITED (STAT,TAG,KEY,PC,BUFFW,LENGW,PCADR)

Parameters

**STAT**

Type: Integer. Value: 0 or a non-zero value.

This return parameter value is set to zero if the call is completed successfully, otherwise to a non-zero value if any error occurs. Possible non-zero values for this routine are: (also refer to Appendix A)

| | |
|---|---|
| -1 | Request accepted but not completed (no-wait call). |
| 2 | Length of buffer to transmit too long or null. |
| 3 | Invalid length unit. |
| 16 | Missing PCIF_OPEN for this program. |
| 17 | Contact with PCIF monitor lost. |
| 18 | Illegal access key. |
| 19 | Illegal buffer address. |
| 20 | Not enough SAM. |
| 21 | Not enough EMA. |
| 22 | RTE EXEC error while dialoging with PCIF monitor. |
| 32 | Locked P/C. |
| 33 | Disconnected P/C. |
| 35 | Unknown P/C logical identifier. |
| 36 | P/C does not have the capability to perform the user's request. |
| 37 | Maximum number of waiting requests reached for this P/C. |
| 38 | Time-out. |
| 128..255 | P/C handler detected error. See Appendix A. |
| 256..511 | Highway handler detected error. See Appendix A. |
| 512.. | Specific P/C manufacturer error. |

**TAG**

Type: Integer. Value: User defined.

This entry parameter has a user defined value which identifies this particular request. It is never modified by PCIF/1000 and is carried along with the reply, for instance it is returned in a PC_ENQUIRY as OLDTAG.

**KEY**

Type: Integer. Value: 0 or provided by GETKEY.

This entry parameter is set to zero if the routine is to be used with the wait option. If the no-wait option is used, the value entered will signify on which access key the answer can be found at a later stage with a PC_ENQUIRY call. This access key value should have been allocated with a previous PC_GETKEY call.

**PC**

Type: Integer. Value: Configuration dependent.

This entry parameter is the logical identifier of a physical P/C station. The parameter value is the one defined at configuration time for the target P/C.

**BUFFW**

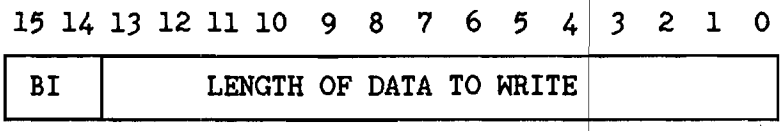
Type: Integer/byte/bit. Value: Data to be sent to the P/C.

This entry parameter is the user defined buffer that will hold the data to be sent to the specified P/C.

**LENGW**

Type: Integer. Value: Length and type of BUFFW.

This entry parameter contains the maximum length of data to be sent to the P/C station. It may be specified in a 16 bit word, a byte, or in bit units and is coded as follows:




where: BI = 0, the length is expressed in 16 bit words;
= 1, the length is expressed in 8 bit bytes;
= 2, the length is expressed in bit units.

LENGTH OF DATA TO WRITE is the number of units to be sent to the P/C, in accordance with the BI information.

Note: For more PC-specific information, see your handler manual.

**PCADR**

Type: 32 bit integer. Value: See specific P/C brand.

This entry parameter contains a value that allows the retrieval of the specified PC's physical memory address where data will be written. The For the correspondence between this parameter value and the real memory address, see your handler-specific manual.

Comments

The capability of the P/C to receive a PC_WRITED request may have been restricted at configuration-time.

# PC_WRITEP

Allows the downloading of P/C programs to a target P/C from an application program.

## Calling Sequence

PC_WRITEP (STAT,TAG,KEY,PC,BUFFW,LENGW,PCADR)

## Parameters

### STAT

Type: Integer. Value: 0 or a non-zero value.

This return parameter value is set to zero if the call is completed successfully, otherwise to a non-zero value if any error occurs. Possible non-zero values for this routine are: (also refer to Appendix A)

| | |
|---|---|
| -1 | Request accepted but not completed (no-wait call). |
| 2 | Length of buffer to transmit too long or null. |
| 3 | Invalid length unit. |
| 16 | Missing PCIF_OPEN for this program. |
| 17 | Contact with PCIF monitor lost. |
| 18 | Illegal access key. |
| 19 | Illegal buffer address. |
| 20 | Not enough SAM. |
| 21 | Not enough EMA. |
| 22 | RTE EXEC error while dialoging with PCIF monitor. |
| 26 | Requested function not implemented on current system. |
| 32 | Locked P/C. |
| 33 | Disconnected P/C. |
| 35 | Unknown P/C logical identifier. |
| 36 | P/C does not have capability to perform user request. |
| 37 | Maximum number of waiting requests reached for this P/C. |
| 38 | Time-out. |
| 128..255 | P/C handler detected error. See Appendix A. |
| 256..511 | Highway handler detected error. See Appendix A. |
| 512.. | Specific P/C manufacturer error. |

### TAG

Type: Integer. Value: User defined.

This entry parameter has a user defined value which identifies this particular request. It is never modified by PCIF/1000 and is carried along with the reply, for instance, it is returned in a PC_ENQUIRY call as OLDTAG.

**KEY**

Type: Integer. Value: 0 or provided by GETKEY.

This entry parameter may be set to zero if the routine is to be used with the wait option. If the no-wait option is used, the value entered will signify on which access key the answer can be found at a later stage with a PC_ENQUIRY call. This access key value should have been allocated with a previous PC_GETKEY call.

**PC**

Type: Integer. Value: Configuration dependent.

This entry parameter is the logical identifier of a physical P/C station. The parameter value is the one defined at configuration time for the target P/C.

**BUFFW**

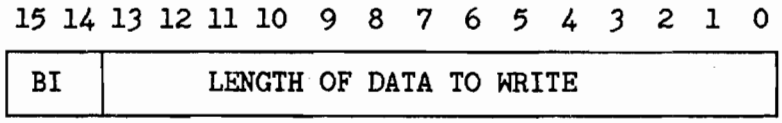
Type: Integer/byte/bit array. Value: P/C program.

This is an entry parameter and is the user defined buffer for storing the program to be sent to the specified P/C.

**LENGW**

Type: Integer. Value: Length and type of BUFFW.

This entry parameter contains the maximum length of program to be sent to the target P/C. This may be specified in a 16 bit word format only, and is coded as follows:




where: BI = 0, the length is expressed in 16 bit words.

LENGTH OF DATA TO WRITE is the number of words to be sent to the P/C.

Note that the length (in words) of the buffer area in the application program must not be less than the length (in words) of LENGW.

**PCADR**

Type: 32 bit integer. Value: See specific P/C brand.

This entry parameter contains a value that corresponds to the specified PC's physical memory address, which is where the P/C program will be loaded. For the correspondence between this parameter value and the real memory address, see your handler-specific manual.

Comments

The user must ensure that the P/C can accept a program before the PC_WRITEP call is made, as this capability may have been restricted at run-time. Also note that the P/C should be stopped and that the P/C outputs deactivated before downloading the program.

Note: For more PC-specific information, refer to your handler manual.

# Chapter 5
# INSTALLATION OVERVIEW

## 5.1 PURPOSE

The aim of the installation and configuration procedure is to achieve a PCIF subsystem ready to interact between application programs and P/Cs. The number and type of application programs and P/Cs are variable and defined by the user. Each user can customize PCIF/1000 to fit the requirements of the target P/C system. The individual variables are defined during the configuration process resulting in a configuration file that is called at run-time, as follows:

CI> XQ, PCIF,<configuration file namr>

The configuration process may be run on a different system than the supervisory system to be used at run-time. However, to enable a simple explanation of the complete process of starting with PCIF/1000 "as shipped" (i.e on magnetic media, and finishing with a ready to use subsystem), it is assumed in this chapter that the configuration and run-time operation will take place on the same computer system.

An overview of the installation and configuration procedures, therefore, may be summarized as follows:

- Generate an RTE-A operating system to support PCIF/1000
- Install the PCIF/1000 software
- Configure the PCIF/1000 software

At the end of configuration, PCIF/1000 will be ready for use.

If the message:

```
PCTMO (System Utility)
PCIF  (System Utility)
Continue, Logoff, Background, or ? [C]?
```

is displayed when logging off after starting up PCIF, simply type "L" to logoff. This should not occur if PCIF is started up from the WELCOME file. Also, it should only happen once, when PCIF is initially started up from a user's session.

NOTE: This chapter is an overview of the detail contained in the following chapters:

- Chapter 6 for INSTALLATION
- Chapter 7 for PRECONFIGURATION
- Chapter 8 for CONFIGURATION
- Chapter 9 for RUN-TIME

Figure 5.1 provides an illustration of the overall procedure required to get the PCIF/1000 subsystem up and running, after receiving the product on shipping media. Subsequent sections in this chapter describe the various stages in this procedure.

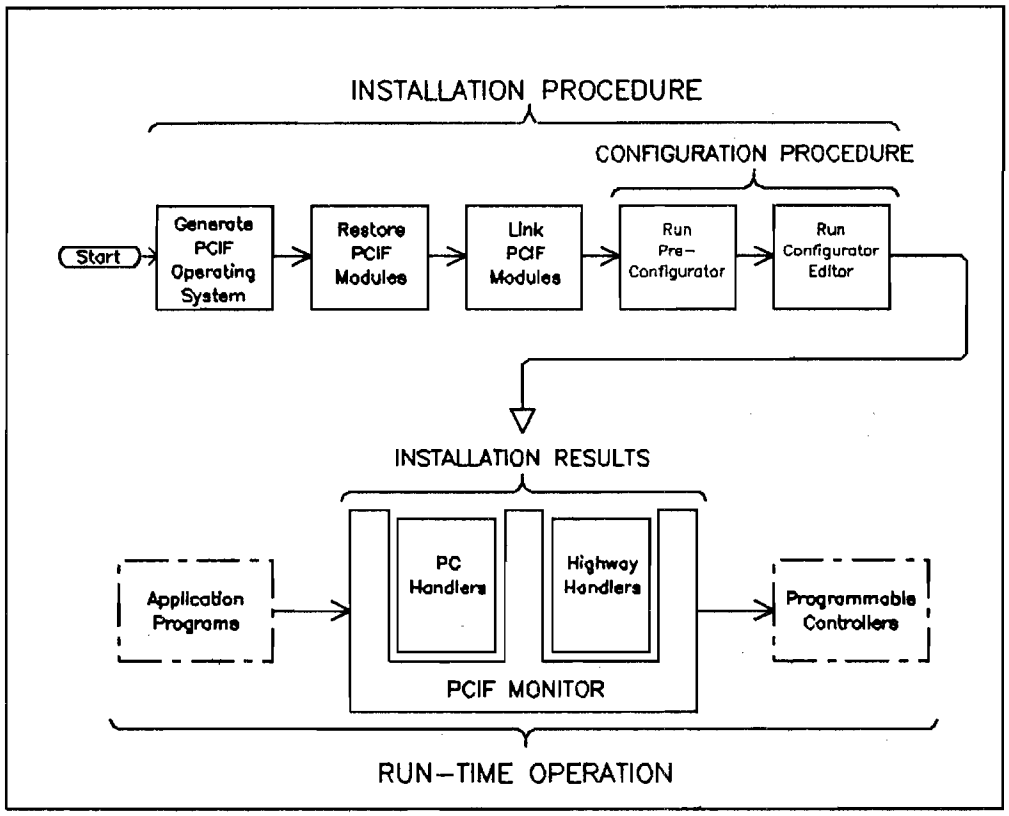



Figure 5.1 Overall Installation Procedure

## 5.2 GENERATING AN RTE OPERATING SYSTEM FOR PCIF/1000

The structure of the operating system supporting PCIF/1000 will vary according to the run-time requirements, such as the number of P/Cs to be supported and the application program requests. The user will have to define the following:

- Specific drivers for the devices and their interfaces
- LUs for the supported PC/highways
- Partition, SAM and EMA size
- Maximum number of class numbers

These various elements are sized according to figures given to the user, and relate to the target P/C system. The minimum requirements are listed in Chapter 6.

A standard RTE-A operating system (with the PCIF additions as stated in Chapter 6) can be used to run the preconfigurator and configuration programs.

## 5.3 INSTALLATION PROCEDURE

The next task in overall installation is to transfer (using the TF utility program) the PCIF software from the shipping media (tape, cartridge or minifloppy) onto the supervisory HP 1000 system.

The contents of the media containing the PCIF monitor must be restored on two places:

1) the PCIF core product files must be restored to a FMGR cartridge

2) the F/1000 runtime files must be restored to a CI directory called /F1000

Once this is achieved, the cartridge reference and security code of the various PCIF files and programs can be changed to suit the user's requirements. Finally, the preconfigurator program is prepared for use by invoking the transfer file command which links the PCIF programs to be used within the user system. This is achieved with the following command:

```
CI> TR,*PCIF::crn,crn
```

where "crn" is the Cartridge Reference Number of the disc cartridge containing the PCIF/1000 software.

The end result is a "ready to run" preconfigurator program called PCGEN.

The contents of the media containing the on-line tutorial, Getting Started with PCIF, must be restored on a CI directory called /GSWPCIF. Then the tutorial is prepared for use by invoking the following transfer file command:

```
CI> TR,XFER.CMD
```

The end result is a "ready to run" tutorial program called TEACHME.

## 5.4 CONFIGURATION PROCEDURE

The configuration procedure can be divided into two phases: the preconfigurator program followed by the configuration editor program. This is because at run-time PCIF is made up of a number of modules. One group of these modules is predefined (by whatever P/C brands are in use) and is created with the preconfigurator to form the run-time monitor. The formation of another group depends upon the number and type of P/Cs and highways to be supported. This group forms the P/C and highway handlers.

The preconfigurator defines a range of P/C and highway types to be used, but specific details for P/Cs (and highways), such as P/C logical identifier, priority and many other characteristics are defined with the configuration editor.

The preconfigurator program is fully described in Chapter 7. Its end result is the construction of a run-time monitor program referred to as the PCIF monitor (PCIF) and the configuration editor (PCCON).

The configuration editor is detailed in Chapter 8. Its end result is a configuration file that describes characteristics of all the P/Cs and highways connected to the

supervisory HP 1000, as defined by the user. The completion of the configuration file signals that PCIF/1000 is ready to run.

The following flow chart shows the complete configuration process.

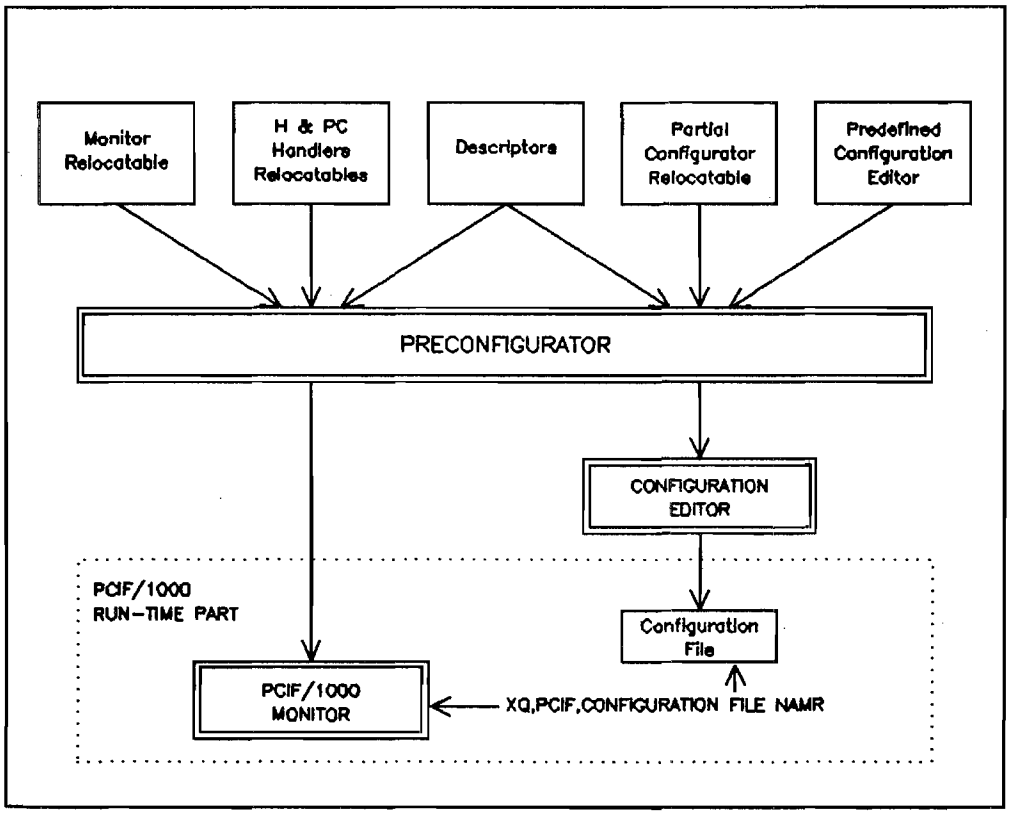



Figure 5.2 Overall Configuration Procedure

When the preconfigurator is executed, the user names descriptor files specific to the types of P/Cs and highways intended for use. All handlers to be used with PCIF/1000 should be configured at this time. The preconfigurator then uses the run-time monitor program (PCIF) relocatables, the appropriate highway, and P/C relocatables to link the run-time monitor program (PCIF). Similarly, the preconfigurator combines predefined configuration editor information to link the configuration editor program (PCCON). The configuration editor can then be run to create the specific application configuration file used at run-time.

If you decide at a later date to include another handler, and thus re-run PCGEN, you must create new configuration files. Do not use old configuration files with a new configurator and PCIF monitor program.

# Chapter 6
# INSTALLATION

## 6.1 INTRODUCTION

This chapter describes the installation requirements of PCIF/1000 and how to install it on an RTE-A operating system.

Due to the modular aspect of PCIF/1000, some of the requirements will be variable depending upon the supported P/C and highway handlers, as well as the real P/C configuration. Only the minimum requirements for installing the basic product are listed here. Supplementary information may be found in the relevant handler reference manual.

PCIF/1000 is divided into two parts. These are:

- The preconfigurator and configuration editor programs.
- The run-time sub-system and P/C Access Routine library.

The two configuration programs may be run on a different computer system than the supervisory system used for run-time. Therefore, the different installation requirements are separated in sections 6.4 and 6.5.

## 6.2 UNPACKING AND INSPECTION

The PCIF/1000 and related products including the multiplexer, multiplexer panel, cables, and required software are shipped in multiple containers. When the shipment arrives, check to ensure the receipt of each container.

# 6.3 HARDWARE REQUIREMENTS

## 6.3.1 General

The following table defines the hardware required for the different parts of PCIF/1000.

| | Preconfigurator & Configurator | Run-time Monitor |
|---|---|---|
| CPU : A/600/A700/A900 | required | required |
| Memory (words) | 256K | 256K (*) |
| Disc storage | required (**) | required(**) |
| Terminal | required (***) | |
| Multiplexer (MUX) | | required (#) |

Table 6.1 Hardware Requirements

* dependent upon configuration and traffic (see section 7.4).
** at least 20 Mbytes of hard disc.
*** must be one of the following: 2622A, 2623A, 2624B, 2626A, 2627A, 2382A, 3092A, 3093A, or HP 150 connected on point-to-point, multiplexed or multipoint connections.
\# MUX panel with MUX interface board. This is only required for HP provided handlers.

## 6.3.2 P/C Connections to the HP 1000

The connection of HP supported P/Cs are made using a MUX interface card with a MUX panel. (See your handler reference manual for the specific MUX to be used with your P/C brand.)

The next three sections describe two possible connection methods, plus a third one which is a mixture of the first two.

## 6.3.3 RS232C MUX/Highway Connection

At run-time, this RS232C connection can be used by any HP supported P/C brand. Also refer to the brand's specific requirements in the relevant handler reference manual.

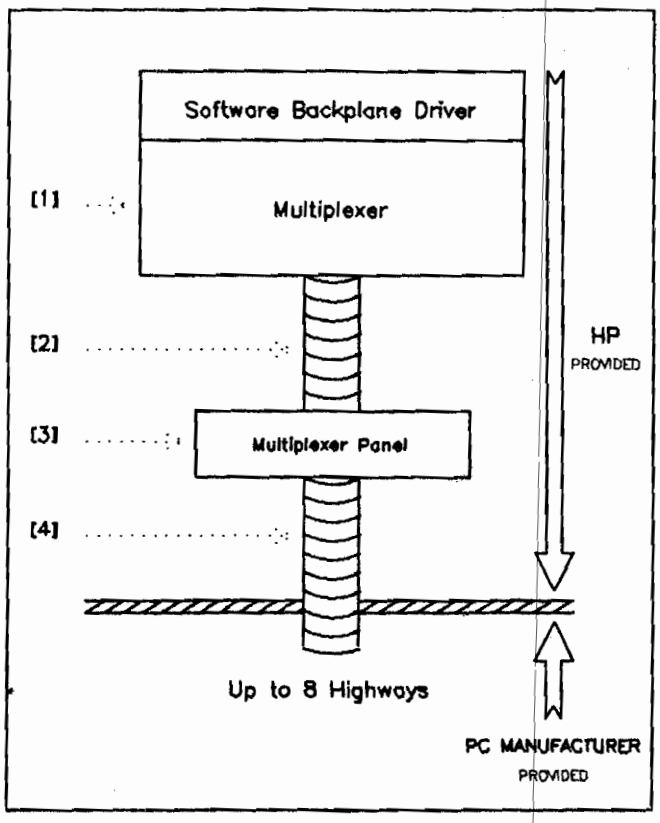



Figure 6.1 RS232C Connection

SUPPLIED BY HEWLETT-PACKARD
[1] MUX interface board
[2] cable provided with MUX panel
[3] MUX panel
[4] cable 25 pin male RS232C to 25 pin male RS232C. Part No. 92140-60001.

SUPPLIED BY P/C MANUFACTURER
Refer to your handler-specific manual.

## 6.3.4 RS232C MUX/Highway 20mA Current Loop Connection

At run-time, this type of connection can be used by certain HP supported PC brands. In each case refer to the individual brand requirements in the relevant handler manual.

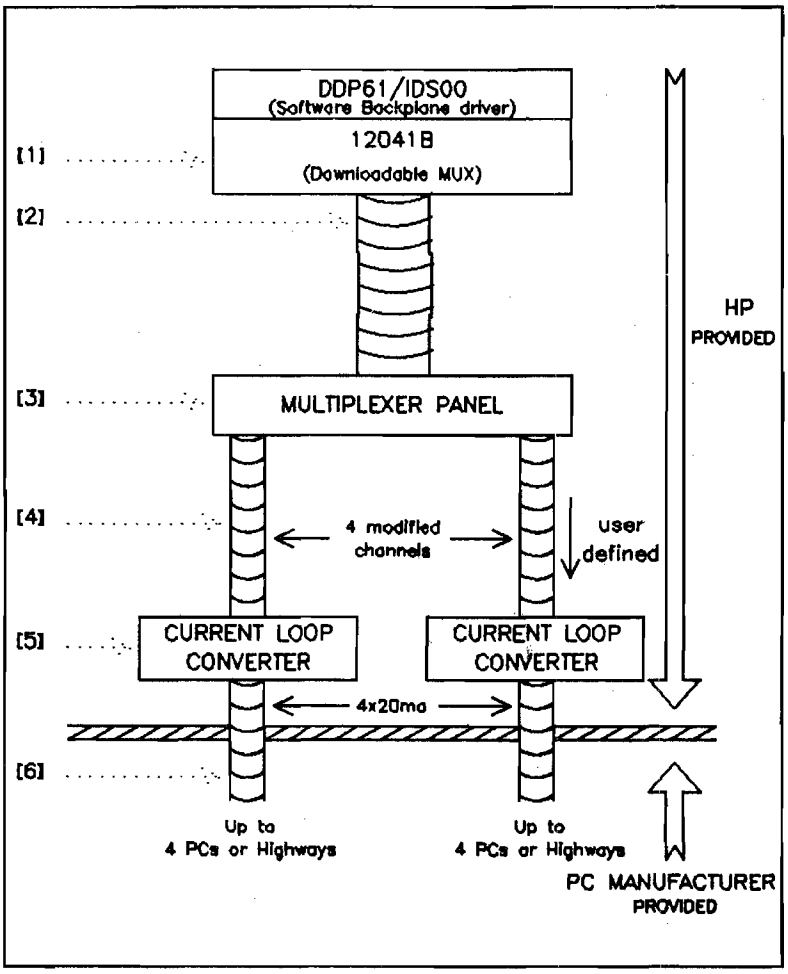



Figure 6.2 Current Loop Connection

SUPPLIED BY HEWLETT PACKARD
[1] MUX interface board.
[2] cable provided with MUX panel.
[3] MUX panel.
[4] customer fabricated cable, 25 pin male RS232C to 3 pin male RS232C.
[5] third party current loop converter; see PCIF/1000 data sheet.
[6] cable for connection between the P/C and the current loop converter.

SUPPLIED BY P/C MANUFACTURER
Refer to relevant handler reference manual.
Connect kit, plug and cables.
Data communication interface.

## 6.3.5 Mixing 20mA and RS232C MUX Connections

At run-time, it is possible to connect RS232 lines and current loop lines on the same MUX card. Note that all P/Cs and highways must be of the same brand and can be downloaded to the same MUX.

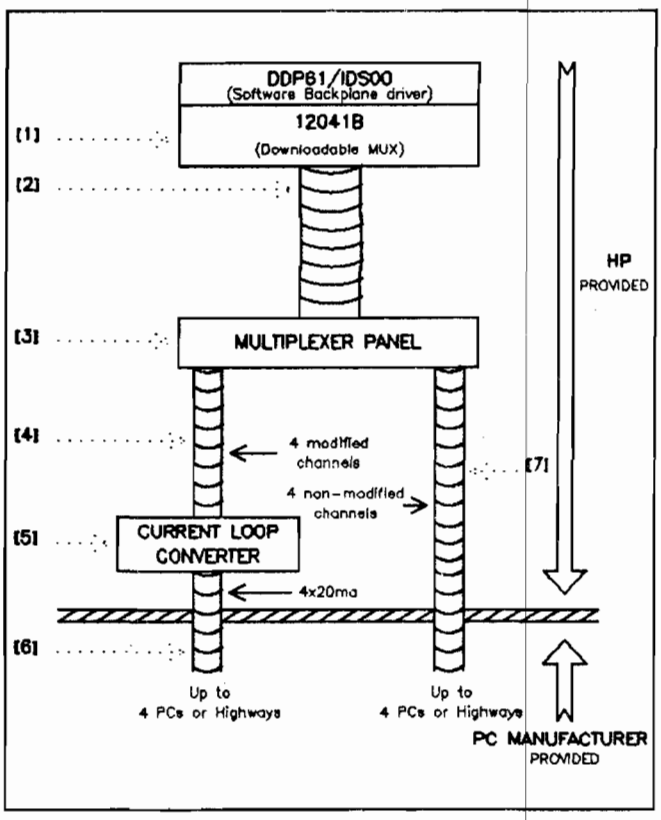



Figure 6.3 Mixing 20mA & RS232C Connected on the Same MUX

SUPPLIED BY HEWLETT PACKARD
[1] MUX interface board.
[2] cable provided with MUX panel.
[3] MUX panel.
[4] customer fabricated cable, 25 pin male RS232C to 3 pin male RS232C.
[5] third party current loop converter; see PCIF/1000 data sheet.
[6] cable for connection between the P/C and the current loop converter.
[7] customer fabricated cable, 25 pin male RS232C to 25 pin male RS232C.

SUPPLIED BY P/C MANUFACTURER
Refer to relevant handler manual.

For this type of connection mix, the adapter cable details will have to be changed so that pin connections are reversed when coupled to a 20mA current loop.

For example, in the following table, if port 0 requires a 20 mA connection and port 1 an RS232 then the adapter cable must be modified as follows:

| | | side A | | side B |
|---|---|---|---|---|
| PORT 0 | RD0+ | 15 | white | 33 |
| (20mA) | RD0- | 33 | black | 15 |
| | | | | |
| PORT 1 | RD1+ | 13 | white | 13 |
| (RS232) | RD1- | 31 | brown | 31 |

Note that RD is data received at the P/C. The transmit data wires are never reversed (as this inversion is made inside the 92922A option D01 if 20 mA conversion is desired).

# 6.4 SOFTWARE REQUIREMENTS

## 6.4.1 General

PCIF/1000 requires certain software packages to be installed on a HP 1000 system before PCIF/1000 can be run on this system. Listed below are these required packages, cross-referenced to the various parts of PCIF/1000:

| | Pre-Configurator | Configuration Editor | Run-time Monitor |
|---|---|---|---|
| RTE-A operating system | required | required | required |
| LINK/1000 loader prgm | required (*) | | |
| PASCAL/1000 library | required (*) | | |
| MACRO/1000 assembler | required (*) | | |
| F/1000 executable code | required (**) | required (**) | |

Table 6.2 General PCIF Software Requirements

* supplied with RTE-A
** supplied with PCIF

## 6.4.2 Preconfigurator & Configuration Editor

The following lists the software requirements for the PCIF/1000 configuration process.

| | | | |
|---|---|---|---|
| Logical Unit (LU) | 1 for the dialog terminal<br>+1 (at least) for a FMGR cartridge | | |
| Class Numbers | 2 for the dialog with program FOCLO | | |
| Resource Number | none | | |
| Memory | program name | size | note |
| | PCGEN<br>PCCON<br>FOCLO | 25 pages<br>24 pages<br>31 pages | (1) (2)<br>(3) |

Table 6.3 Configuration Software Requirements

Notes: [1] The Configurator uses VMA for building configuration files.
[2] The real size depends on the partial configurator editors.

(Figures given here are for HP provided handlers.)
[3] This is the F/1000 forms management monitor used by PCIF/1000.

Table 6.4 lists the default and required directories for the PCIF software.

| NAME | TYPE | COMMENTS |
|---|---|---|
| /LIBRARIES | CI directory | Required to hold $PCIFM (if CDS monitor used). $PCIFM is accessed by MACRO during PCGEN build of PCIF. $PCIFM is built from "CDSLB during load of PCGEN in transfer file *PCIFC. Default directory for PASCAL.LIB and PASCAL_CDS.LIB. Referenced in link command files. |
| /F1000 | CI directory | Required directory for F/1000 software. Referenced in link command files. |
| /PROGRAMS | CI directory | Required directory must contain program FOCLO.RUN. Referenced in link command file #FOCLO. |
| ::P1 | FMGR directory | Required directory for PCIF/1000 software. P1 is the recommended name. |

Table 6.4 Required PCIF Software Directories

Table 6.5, below, lists all the files required by the PCGEN preconfigurator program, and the PCCON configuration editor program. The table also lists the types of access made by the programs to their files. (See your handler reference manual for files specific to your P/C brand.)

| PCIF Programs | Files Used | Access Types |
|---|---|---|
| PCGEN | PCCON | CR (1) |
| | PCIF | CR (1) |
| | [PCPGE, [PCPGF, $PCIFM | RO (4) |
| | &PCCTB, &PCRTB, &PCRTC | SC (2) (5) |
| | #PCLCO, #PCLRT, %PCCTB, %PCRTB, %PCRTC | UP (3) (6) |
| | plus possible user list file | |
| | plus three screens: | |
| | !PCP01, !PCP02, !PCP03 | RO |
| | and three associated help files: | |
| | "PCP01, "PCP02, "PCP03 | |
| | FOCLO.TXT::F1000, "PCERR | RO |
| | scratch files for LINK & MACRO | SC |
| PCCON | configuration editor file | CR |
| | plus possible user list file | UP |
| | and 11 screens: | |
| | !PCC01...!PCC11 | RO |
| | !PCCB5, !PCCB9, "PCCB5, "PCCB9 | |
| | and 11 associated help files: | |
| | "PCC01..."PCC11 | RO |
| | "PCERR | RO |
| | virtual memory file | SC |

Table 6.5 PCGEN and PCCON Required Files and Access

Access Types
CR created file : this file MUST NOT exist prior running PCGEN.
RO read only file : this file is not modified by this PCIF/1000 step.
SC scratch file : this file is deleted at completion of the present step.
UP updated file : this file is created if it does not exist.

(1) These files must not already exist.
(2) These are scratch files used internally and then purged.
(3) These files are kept at the end of preconfiguration (see Chapter 7).
(4) [PCPGE for the non-CDS monitor; [PCPGF and $PCIFM for the CDS monitor. $PCIFM is created from "CDSLB by the transfer file *PCIFC. It must reside on the ::LIBRARIES directory.
(5) &PCRTB for the non-CDS moniotr; &PCRTC for the CDS monitor.
(6) %PCRTB for the non-CDS monitor; %PCRTC for the CDS monitor.

## 6.5 RUN-TIME RTE-A REQUIREMENTS

This section describes the logical unit, answer file, class numbers, memory, and file requirements of PCIF/1000.

### 6.5.1 Logical Units

Each data highway requires a certain number of LUs. This number varies among the different P/C brands. See your handler-specific reference manual for details.

### 6.5.2 Answer File

As far as LU generation is concerned, handler-specific information is to be put into the RTE-A generation answer file. See your handler-specific manual for this information.

### 6.5.3 Class Numbers

The run-time part of PCIF/1000 contains several programs as well as the user's application programs. All of these programs use RTE-A class numbers for communication, Therefore the RTE-A system must be generated with the following minimum requirements:

- one class number for the PCIF/1000 run-time monitor
- one class number for each concurrent application program
- one class number for each access key

The total number of class numbers is at least:

```
N = 1+(max AP)+(max AK)
```

where:

(max AP): is the maximum number of application programs to run concurrently with PCIF/1000. The maximum is 16.

(max AK): is the maximum number of access keys as defined in Screen 11 of the configuration editor program, for all application programs. PCIF/1000 allows a total of 64 access keys for all the application programs, and 16 for each application program.

### 6.5.4 Memory Requirements

PCIF/1000 memory requirements are concentrated in the following areas:

- PCIF/1000 program partitions (code and data);
- EMA size (depending upon the P/C user's configuration);
- SAM size (depending upon the P/C user's configuration).

The various PCIF/1000 run-time programs have the following requirements:

| memory | program name | size | note |
|---|---|---|---|
| PARTITION | PCIF (monitor) | 36 --> 500 pages | (1), (4) |
| | PCOPN (open) | 2 pages | (2) |
| | PCTMO (time out) | 5 pages | (2) |
| | PCDMX (MUX download) | 9 pages | (2), (3) |
| | PCHLT (halt) | 5 pages | |

Table 6.6 Memory Requirements

Note: (1) It is recommended that the PCIF monitor be locked in memory for faster response time. However, this is application dependent and it may not be necessary for specific applications.

(2) These programs are linked as system utility programs and MUST have an ID segment (can be created with RTE A command: RP,...). If the RTE-A system has a session monitor, these programs can only be removed by a super-user. The assignment of an ID segment to these programs can be done in the welcome file.

(3) This program is required for specific P/Cs to download communication protocol to the 12041A or the 12041B downloadable MUX. Refer to your handler reference manual if your system uses either of these MUX cards.

(4) The EMA size needed by the PCIF monitor can be set by the user using an RTE-A "SZ" command. The minimum size can be approximated using Figure 6.4 on the following page.

PCIF/1000 EMA REQUIREMENTS

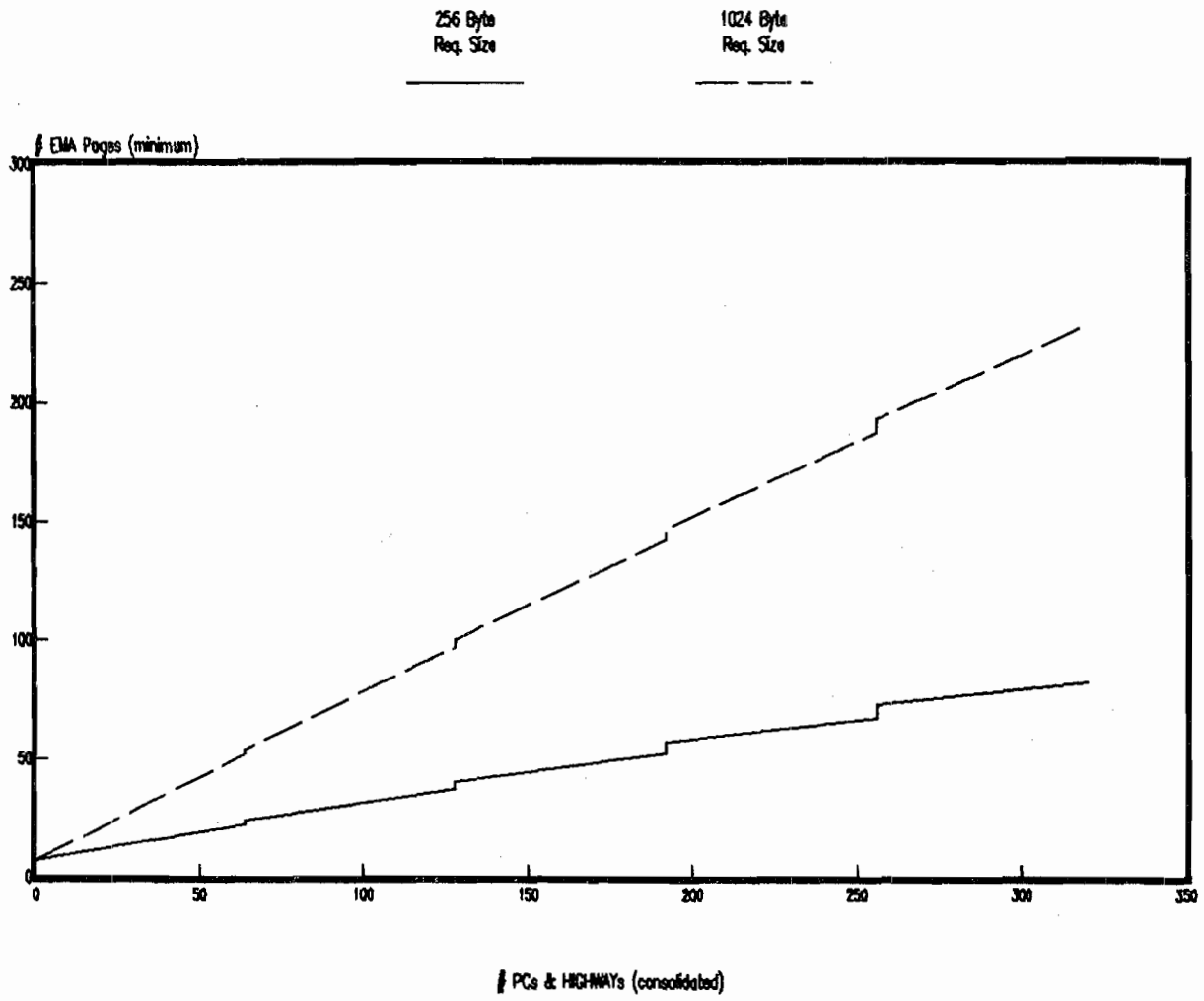



Figure 6.4 Minimum EMA Size

The minimum size is given using the following data:

- One request simultaneously on each P/C, with a length of 256 bytes (Curve 1).
- The Handler used is HP provided with a request length of 1024 bytes (Curve 2).

The minimum SAM size (in pages) can be approximated by the following formula:

| | |
|---|---|
| <number of data highways> | * 0.132 |
| + <general P/C overhead> | 1.000 |

## 6.5.5 Required Files

PCIF/1000 requires the use of some FMGR files at run-time. These files are used in read-only access mode and will not be modified by PCIF/1000. The files are as follows:

| PCIF programs | FMGR files | note |
|---|---|---|
| PCIF (monitor) | <configuration file> | (1) |
| | "PCMER | (2) (3) |
| PCTMO | "PCMER | (2),(3) |
| PCDMX | !PCFxx | (3),(4) |
| | "PCMER | (2),(3) |
| PCHLT | "PCMER | (2),(3) |

Table 6.7 Required Files

Note: (1) This file is provided by the user. It has been built using the configuration editor and is specified in the run string of PCIF/1000.

(2) This run-time error file can be localized by the user or it can be the HP provided file. The file is searched on the FMGR cartridges in cartridge mounted order.

(3) This file is searched on the same FMGR cartridge as that given with the highway descriptor file namr in Screen 1 of the preconfiguration process. The highway descriptor file is [PCHxx (P/C brand dependent). See Chapter 7.2.4 for details about the highway descriptor file.

(4) This file is used only for PC's requiring the 12041A or 12041B downloadable MUX.

## 6.6 SOFTWARE INSTALLATION

This section describes how to load the PCIF/1000 software, prepare it for installation, and install the PCGEN program.

### 6.6.1 Loading Software

The PCIF core software is delivered on magnetic media and must be restored to two places:

1) the PCIF core product files must be restored to a FMGR cartridge

2) the F/1000 run-time files must be restored to a CI directory called /F1000

To restore the software, use the copy command CO of the utility program TF, as in the following example:

```
CI>  TF
TF:  GR
TF:  CO, <media device LU>{/PCIF/CORE/@.@},@.@::<FMGR cartridge name>,V
TF:  CO, <media device LU>{/PCIF/F1000/@.@},/F1000/@.@,V
TF:  EG
TF:  EX
```

where: <media device LU> is the LU number of the device containing PCIF/1000.
<FMGR cartridge name> is the destination cartridge for PCIF/1000.

You must use the ::xx format to specify the FMGR cartridge. The FMGR cartridge must already exist, because, if it does not, TF will create a CI directory of the same name.

The on-line tutorial, Getting Started with PCIF, must be restored to a CI directory called /GSWPCIF.

This is achieved using the TF command described below:

```
CI>  TF
TF:  CO, <media device LU>{/PCIF/GS/@.@},/GSWPCIF/@.@, V
TF:  EX
```

### 6.6.2 Preparing PCIF/1000 Installation

Once restored, PCIF/1000 programs must be linked on the user system. The following LINK command files are provided:

- for the preconfigurator: #PCLGE (or #PCLGC for CDS monitor)
  #PCFOC
- for the run-time subsystem: #PCLOP
  #PCLDM
  #PCLTM

#PCLHL
#PCTST

It is recommended that the transfer file, *PCIF or *PCIFC, (described in subsection 6.6.3) be used to install PCIF. If you do not use these files, you should at least consult them to make sure that you are installing PCIF properly, especially for creating $PCIFM::LIBRARIES.

Do not initialize the MUX port in the WELCOME file.

## NOTE:

These command files use P1 as a default cartridge reference. The default security code, R2, is used for all the files created at the user's site (programs PCGEN, FOCLO, PCOPN, PCDMX, PCTMO and PCHLT). The security code of all other PCIF/1000 files is "0". The operating system snap-shot file uses SNAP as a default name, while the PASCAL library uses /LIBRARIES/PASCAL.LIB.

The user can edit these files for changing default values. See the following page for an example.

In the following example, the command file #PCLGE has been transferred to the user's cartridge (reference US) with the TF command, as explained in 6.6.1. The cartridge references of all the files must now be changed from the default reference (P1) to US. In addition, a security code can also be inserted. In this example it is 31416. These changes are achieved with the "Exchange" EDIT command.

```
EDIT #PCLGE::US                          ** PCIF files are on US **
  .....              EDIT/1000 comments
/1L
  * #PCLGE 94200-17002 REV.2525 <850610.1725>
  * link file for PCIF/1000 PCGEN <preconfigurator>
  * command file needs updating for FMGR cartridge number & security code
  * command file needs updating for the Snap Shot file and PASCAL library.
  SN,SNAP
  LI,/F1000/FOPRL.LIB
  LI,/F1000/FRULB.LIB
  LI,/LIBRARIES/PASCAL.LIB
  RE,/F1000/FCOMM.REL
  RE,$PCGEN::P1
  EN,PCGEN:R2:P1
/1$X/::P1/::US//
  .....              EDIT/1000 comments
/1$X/:R2:P1/:31416:US//
  .....              EDIT/1000 comments
/1L
  * #PCLGE 94200-17002 REV.2525 <850610.1725>
  * link file for PCIF/1000 PCGEN <preconfigurator>
  * command file needs updating for FMGR cartridge number & security code
  * command file needs updating for the Snap Shot file and PASCAL library.
  SN,SNAP
  LI,/F1000/FOPRL.LIB
  LI,/F1000/FRULB.LIB
  LI,/LIBRARIES/PASCAL.LIB
  RE,/F1000/FCOMM.REL
  RE,$PCGEN::US
  EN,PCGEN:31416:US
/ER
```

This operation must be perfomed on all other previously indicated files. A similar operation must be performed if F/1000 is not stored on the CI directory /F1000. In any event, the program FOCLO.RUN must be put in the directory /PROGRAMS.

PCIF/1000 may be split between two computer systems, one for running the preconfigurator and configuration editor, and possibly another for the run-time monitor and utilities. In this case two snap-shot files are used:

* snap-shot (1) for #PCLGE, #PCFOC, and #PCLCO (the link command file built by PCGEN for linking PCCON).

* snap-shot (2) for #PCLOP, #PCLDM, #PCLHL, and #PCLRT (the link command file built by PCGEN for linking PCIF monitor).

REMARKS:

Using the wrong SNAPSHOT FILE may result in an incorrect RPL checksum at execution time.

The following explanations may help the user to correct some unusual situations:

a) The preconfigurator or configuration editor can be aborted with the error message F0106 (FMP RP error on FORM monitor file). This means that the preconfigurator or the configuration editor were unable to start the F/1000 forms management monitor because of RPL checksum. In such a situation, retry the last RTE command (i.e. RU,PCGEN or RU,PCCON). Make sure that your working directory is 0 (type "WD 0") before you re-run PCGEN or PCCON. The error should have disappeared.

b) At the end of preconfiguration (PCGEN), the link process for PCCON or PCIF might be aborted because the specified snapshot file, or the pascal library does not exist under the names SNAP or /LIBRARIES/PASCAL.LIB. In this case, correct the files, #PCLCO and #PCLRT, as described in section 7.3.

## 6.6.3 Installing The Preconfigurator Program

The installation of the preconfigurator (PCGEN) is achieved automatically by calling one of two transfer files, *PCIF or *PCIFC. If you are using only one handler type, you can use the non-CDS monitor, loaded with the transfer file *PCIF. If you have several P/C brands in your system, or if the PCIF monitor program will not load using the PCGEN created by *PCIF, then use the CDS version of the monitor by invoking the transfer file *PCIFC.

TR *PCIF::<PCIF FMGR cartridge>, <PCIF FMGR cartridge>

or

TR *PCIFC::<PCIF FMGR cartridge>, <PCIF FMGR cartridge>

where: <PCIF FMGR cartridge> is the name of the cartridge on which PCIF/1000 was restored.

NOTE: This installation must not be done on a system where PCIF and PCCON are presently running, or where some run-time programs of PCIF/1000 have an ID segment. Should any of these be present on the target system, parts of the *PCIF or *PCIFC operations will be aborted.

When *PCIF or *PCIFC ends, the operator is able to use the preconfigurator program. This is described in Chapter 7.

Note that the preconfigurator and configuration editor use the forms management package F/1000 for controlling the terminal/operator interaction. If the user already owns this package, the installation as described with PCIF/1000 must still be followed. Then either the PCIF provided "forms" monitor FOCLO can be used, or the monitor the user received with F/1000.

# Chapter 7
# PRECONFIGURATION

## 7.1 INTRODUCTION

Preconfiguration is the definition of a source data area from which the PCIF run-time monitor and the configuration editor will be expected to work. This definition is achieved with a menu-driven program called the preconfigurator. The program displays screens which are used to establish the following information:

- Choose the names of the descriptor files. Each P/C brand requires two descriptor files, one for the P/C itself and one for the highway.
- Verify that the required P/C or highway is supported, by checking the contents of the descriptor file.
- Repeat the process for all the PC/highway combinations requiring configuration.
- Confirm that all the required information has been found.

The preconfigurator will use this information to create two programs:

- The configuration editor, PCCON.
- The PCIF run-time monitor, PCIF.

The following CI command is used to start the preconfigurator (be sure to set your working directory to "0" with the CI command "WD 0"):

```
CI> PCGEN::cartridge reference
```

The cartridge reference was previously specified when the program PCGEN was loaded. If it is unknown, it may be found by typing the CI command:

```
CI> DL, PCGEN::0
```

NOTE: ALL FILES should be on an FMGR cartridge; the various screens described here use the FMGR namr format to name files.

## 7.2 SCREEN DESCRIPTIONS

### 7.2.1 General

During the execution of the preconfigurator, information is requested via the screened menus described in this section. There are three screens to be processed, screen 1 and 2 in a loop followed by screen 3. This process may be summarized as follows:

- Enter the name of a descriptor file.
- Verify the content of this descriptor file.
- Enter another descriptor file name for the associated P/C or highway.

- Verify the content of this file.
- Obtain a listing of all the verified information.
- Link the PCIF monitor (PCIF) and the configuration editor (PCCON) programs.

This summary corresponds to the following sequence of screens:

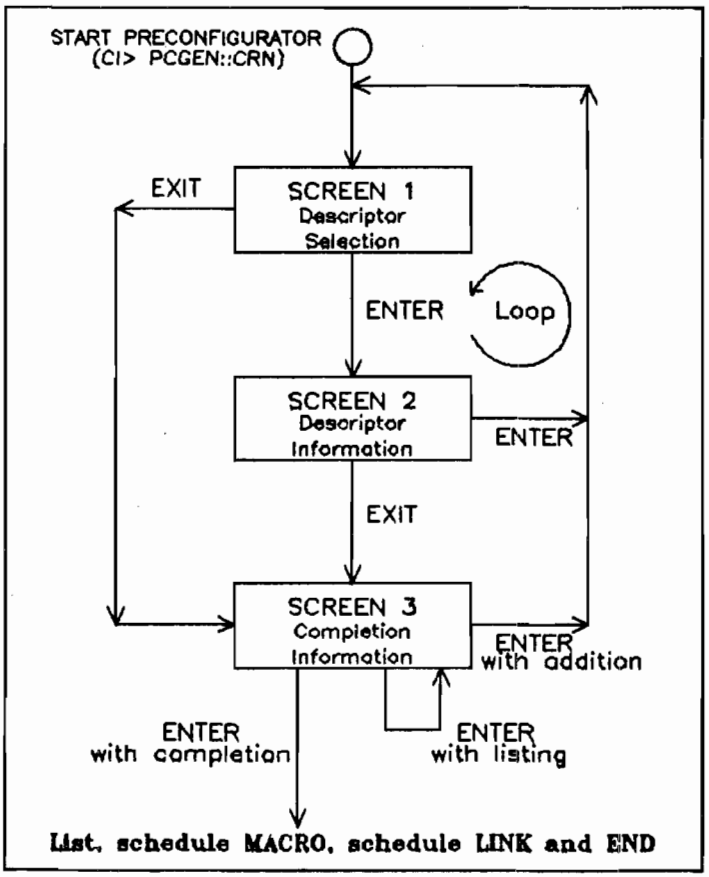



Figure 7.1 Preconfigurator Screen Sequencing

## Screen Sequencing Comments

The loop comprising screen 1/ENTER, followed by screen 2/ENTER must be maintained until all the the required information has been defined for both the P/C and the highway portion of each brand. Then pressing EXIT on screen 2 displays screen 3 which allows a file to be defined for listing this preconfiguration information, plus the allocation of a security code for the run-time monitor and configuration editor program files created by the preconfigurator.

When ENTER (with completion) is pressed on screen 3 the preconfigurator first schedules the MACRO assembler program, followed by the LINK loading program to develop the run-time monitor (PCIF) and configuration editor (PCCON) programs. The successful development of these programs signals the completion of the preconfigurator program.

## 7.2.2 Operation

The screens of the preconfigurator program are managed by the operator interface program F/1000. Attempted use of the screens outside the parameters of F/1000 may result in an error message : FOxxx FORMS ERROR.

The following should be noted when using the screens:

- The operator can only enter values or information in the unprotected fields of the screen. These fields are always enhanced with inverse video, and may already contain data that can be overwritten.

- The cursor is moved between unprotected fields by pressing the TAB key (which moves the cursor to the NEXT field) or pressing TAB and SHIFT keys simultaneously (which moves the cursor to the PREVIOUS field).

- The ENTER key, and sometimes the EXIT softkey are used to validate the information entered on the screen and to initiate the screen sequencing.

- The window line is a display-only field where error messages are displayed when an error has been detected. The entered data is not validated and the current screen will remain displayed until the operator takes corrective action.

## 7.2.3 Softkeys

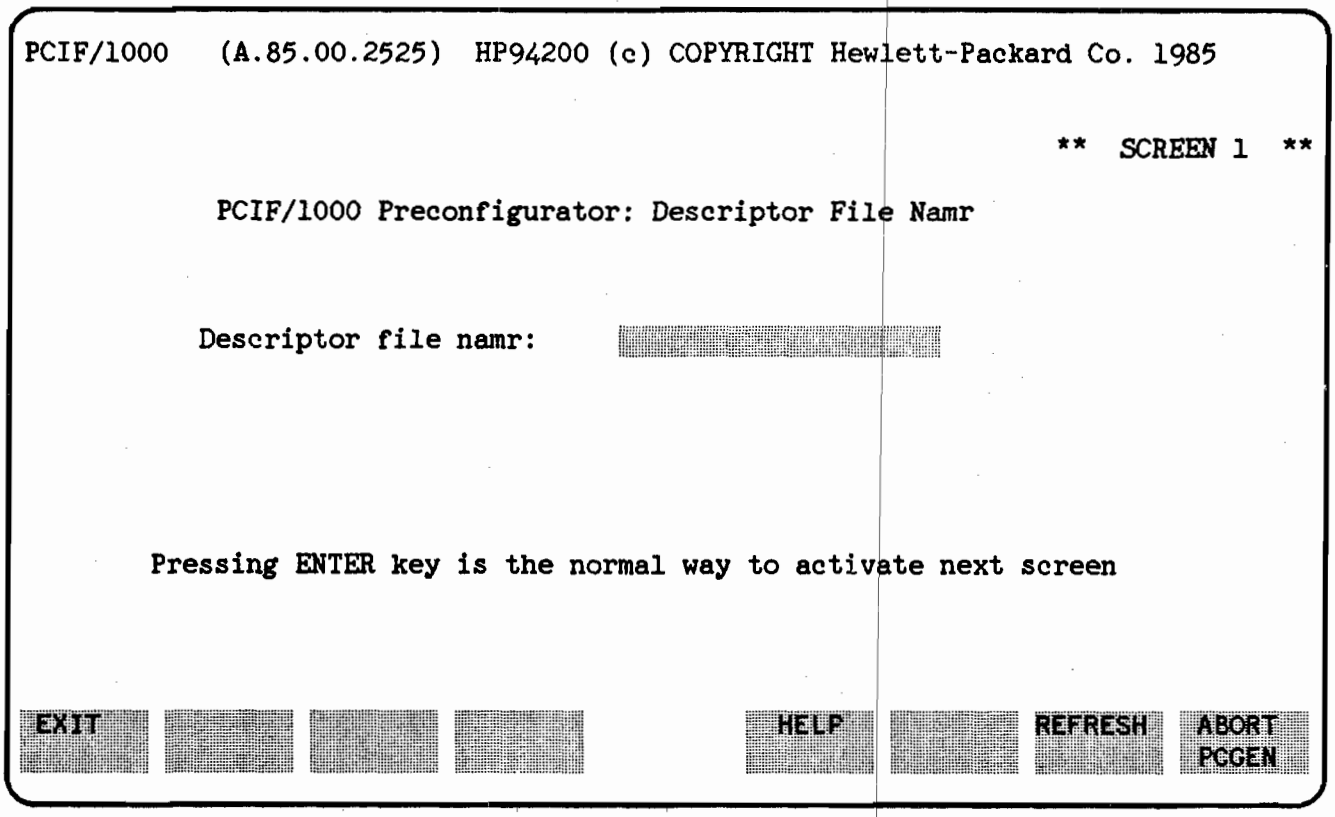
The following softkeys always appear in the same position with each screen.

- HELP — Displays an explanation of the use of the current screen. Pressing the terminal key "home-up" redisplays the current screen in the same state as it was when HELP was requested.

- REFRESH — Clears and resets the values of the currently displayed screen to the values it had on first display.

- ABORT PCGEN — A confirmation will be required if this key is pressed. The key must be pressed again to confirm. The preconfigurator will then be terminated without generating either the run-time monitor or the configuration editor. If the ABORT key is pressed once in error, pressing another softkey (or ENTER key) disables the ABORT request and the action associated with the depressed key is taken.

## 7.2.4 Screen 1: Descriptor Selection




Comments

This screen asks for the FMGR namr of a descriptor file which will contain relevant information on the P/C or highway to be configured. Refer to your handler reference manual for the descriptor files that correspond to your handler.

The handler, possibly the partial configurator, and possibly the download files that are named in a descriptor file must exist on a FMGR cartridge.

The FMGR namr field is made up of twenty characters. It is advisable to enter the cartridge reference in the FMGR namr, as the preconfigurator will only search for associated files (handlers, partial configurators, etc) on this specified cartridge. If no cartridge reference is specified, the preconfigurator will search throughout the FMGR cartridges for these files.

The operator must press ENTER to proceed to screen 2 when the FMGR namr field has been filled.

The descriptor files used must always be consistently all "CDS" descriptor files or "non-CDS" descriptor files for all brands of P/Cs selected. See the appropriate handler manual.

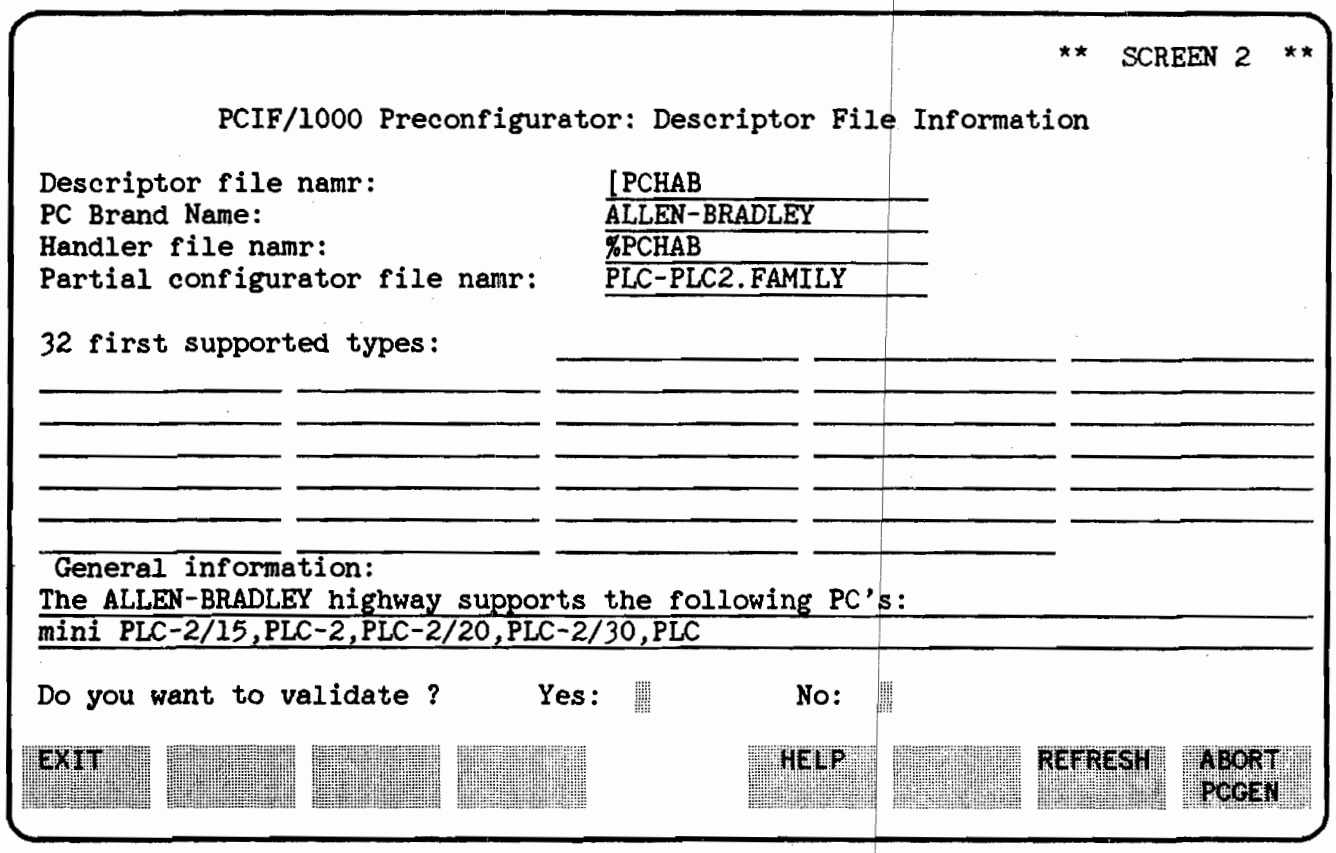
If the softkey EXIT is depressed, screen 3 is displayed. However, the preconfigurator will ignore the file name entered in the FMGR namr field. The acceptance of descriptor file names by the preconfigurator only occurs with the validation of screen 2.

Error Messages

The following error messages may occur with this menu after the ENTER key has been pressed.

- C0001 REQUIRED FIELD MISSING - ENTER VALUE IN FIELD. No file namr has been entered.
- C0003 NON-DESCRIPTOR FILE: The file exists but is not a descriptor file.
- C0035 NO ASSOCIATED HANDLER (OR FMP ERROR ON THIS FILE). A handler file has been named in the specified descriptor file but has not been found.
- C0037 NO ASSOCIATED PARTIAL CONFIGURATOR (OR FMP ERROR ON THIS FILE). A partial configurator file has been named in the specified descriptor file but has not been found.
- C0038 ALREADY ENTERED DESCRIPTOR FILE NAMR.
- C0061 NO ASSOCIATED DOWNLOAD FILE (OR FMP ERROR ON THE DOWNLOAD FILE). A download file has been named in the specified descriptor file but has not been found.
- DCxxx CORRUPTED DESCRIPTOR
- FMxxx FILE MANAGER ERROR.
- XXxxx ?? : No error message has been found.

## 7.2.5 Screen 2: Descriptor File Information




Comments

This screen describes the descriptor file whose FMGR namr was entered in the previous screen. (The screen above shows Allen-Bradley specific information as an example.) The preconfigurator program will search for, open and read the contents of these files.

The operator must confirm whether the displayed information needs to be in the required configuration. This validation is achieved by entering an X in the YES field followed by pressing EXIT or ENTER. The displayed descriptor information will then be held for later use by programs MACRO and LINK (see 7.3). Otherwise, an X in the NO field refuses the information displayed on this screen, i.e. the associated handler file.

ENTER returns the preconfigurator to screen 1 for the definition of another descriptor namr, while EXIT goes to screen 3 (in both cases the action given by X is made).

Note that it is impossible to modify the handler namr or the list of supported types on this screen.

Error Messages

- C0001 REQUIRED FIELD MISSING: ENTER VALUE IN FIELD: Either a yes or no must be selected.

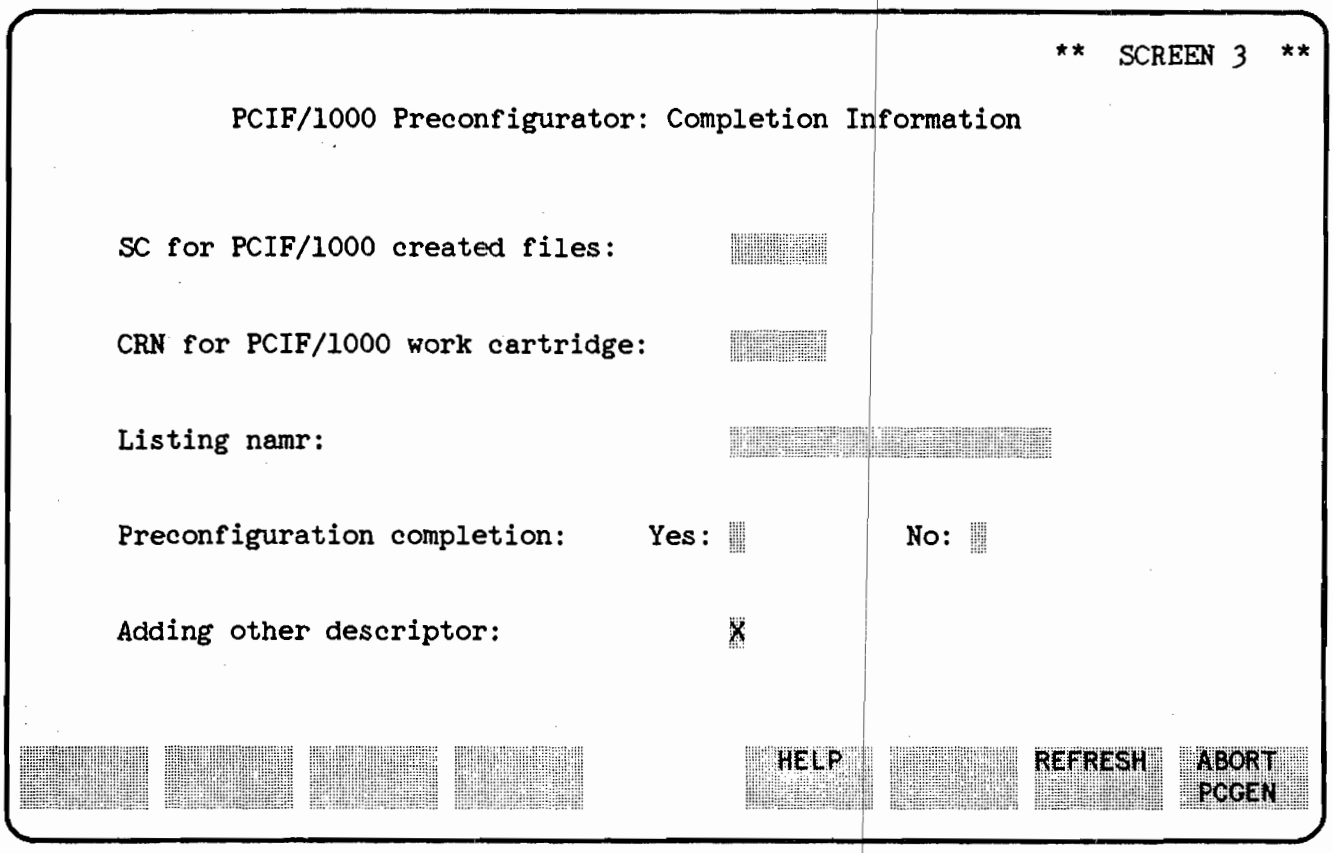
- C0004 X REQUIRED: Only the letter X in either upper or lower case can be entered.

- C0005 CONFLICTING ANSWERS: Both yes and no have been simultaneously selected.

- FOxxx FORMS ERROR

- XXyyy UNDEFINED ERROR: No message has been found for this error in the error message file.

## 7.2.6 Screen 3: Completion Information




Comments

This screen is used to complete the preconfiguration program by defining the security code (SC) and the cartridge reference (CRN) of the run-time monitor and the configuration editor, plus the name of a list file for recording the preconfiguration answers. The required information is as follows:

- SC FOR PCIF/1000 CREATED FILES: This allows the operator to define the security code for the PCIF run-time monitor, for PCCON (the configuration editor), and for the MACRO and LINK files.

- CRN FOR PCIF/1000 WORK CARTRIDGE: Allows the operator to define the cartridge reference for the PCIF monitor file, for PCCON (the configuration editor), and for the MACRO and LINK files.

- LISTING NAMR: a FMGR namr must be entered here if the user decides to end the preconfiguration process. A confirmation request will be displayed after pressing ENTER if the file already exists. Pressing ENTER again will overlay this file with the latest preconfiguration data. This will include the descriptors, the handlers and partial configurator file names that have been chosen. The listing can be selected alone or simultaneously with "adding other descriptor". Screen 3 will be displayed again if it is selected alone, and screen 1 for adding another descriptor.

- PRECONFIGURATION COMPLETION: Enter the letter X in either upper or lower case in the appropriate field. An answer one way or the other is required, and this has been preset to "no". A "yes" is not allowed if "adding another descriptor" is selected in the same screen. If "yes" is selected, subsequently pressing ENTER schedules the generation of the run-time monitor (PCIF) and of the configuration editor (PCCON). For details of this process refer to section 7.3.

- ADDING OTHER DESCRIPTOR: Enter the letter X in either upper or lower case to display Screen 1 again. This can only be selected if the "completion" option has not been taken.

Error Messages

- C0001 REQUIRED FIELD MISSING. ENTER VALUE IN FIELD: Neither "yes", "no" nor "adding another descriptor" have been selected.

- C0004 X REQUIRED: A character other than X has been used to answer the "adding another descriptor" or "completion" questions.

- C0005 CONFLICTING ANSWERS: A "yes" and a "no" answer have been selected simultaneously for the "completion" question.

- C0011 NO TASK SELECTED: None of the questions in the screen have been answered, and a "no" has been selected for the "completion" option.

- C0074 ALREADY EXISTING PCIF MONITOR OR CONFIGURATION EDITOR ON THIS CARTRIDGE.

- FOxxx FORMS ERROR.

- XXyyy ??: No message has been found for this error in the error message file.

## 7.3 COMPLETION OF THE PRECONFIGURATOR

The operator decides that all the required descriptors have been specified and verified. Then having selected the "completion" field in screen 3 and pressed ENTER, the terminal screen is cleared. The preconfigurator program now generates the configuration editor and the run-time monitor by first using the assembler program MACRO followed by the loader program LINK.

During the operation of MACRO and LINK the following sequence of events occur:

- The preconfigurator builds two assembler source files. These are &PCCTB for generating the configuration editor, and &PCRTB for the run-time monitor (or &PCRTC for the CDS run-time monitor).

- MACRO is scheduled twice, assembling &PCCTB which produces a relocatable %PCCTB, and &PCRTB which produces %PCRTB (or assembling &PCRTC which produces %PCRTC). The two source files are then purged.

- The preconfigurator builds two command files for the LINK program. These are #PCLCO for the configuration editor, and #PCLRT for the run-time monitor. Only one link command file name is used for both CDS and non-CDS monitors. This command file reflects what is currently linked as PCIF.

- LINK is scheduled a first time for the configuration editor and displays a list of its segments. The last displayed are the partial configurators just selected at preconfiguration time. If the configuration editor has been successfully assembled and loaded, the following message will be displayed: PCCON READY.

- LINK is scheduled a second time for the run-time monitor and also displays a list of segments. If the run-time monitor has been successfully assembled and loaded, the following message will be displayed: PCIF READY.

The preconfiguration program finishes with the successful creation of the configuration editor (PCCON) and the run-time monitor (PCIF) programs.

The two LINK command files #PCLCO and #PCLRT, and the relocatable files %PCCTB and %PCRTB (or %PCRTC) are not purged after preconfiguration. The files #PCLCO and #PCLRT can be modified and used to reload PCCON and PCIF without running the preconfigurator again. If it is necessary to run the preconfigurator again, the files, PCCON and PCIF, should be purged.

The operator can now run the configuration editor program to complete the configuration process. Refer to Chapter 8 for details.

The file relationship may be summarized as follows:

| | | | |
|---|---|---|---|
| MACRO | &PCCTB | &PCRTB or &PCRTC | source file |
| MACRO | %PCCTB | %PCRTB or %PCRTC | relocatable file |
| LINK | #PCLCO | #PCLRT | command file |
| | Configuration Editor PCCON | Run-time Monitor PCIF | |

All these files are located on the FMGR cartridge which has been specified in screen 3, and with the security code also provided in screen 3.

Any problem in the linking process will cause an abort before completion. The reason for this may be one of the following:

a) Some files were not found: SNAP and PASCAL libraries for example. Edit files #PCLCO and #PCLRT to the file names used (Refer to section 6.6.2 for editing these files). When the corrections are made, then run LINK again.

```
CI> LINK, #PCLCO:<SC>:<CRN>

CI> LINK, #PCLRT:<SC>:<CRN>
```

Where <SC> and <CRN> are those values entered in screen 3.

b) The program to link is too large (this usually happens with the link of PCIF): There are too many P/C and HIGHWAY handlers for this system. Redo the pre-configuration with less brands, or use the CDS version of PCIF by re-installing with the transfer file *PCIFC.

c) Other link messages may appear, see the LINK reference manual (PN 92077-90007) for correcting the errors, then run LINK again as described in step a.

# Chapter 8
# CONFIGURATION

## 8.1 GENERAL


### 8.1.1 Introduction

The configuration editor program allows the P/C user to set or modify values for the parameters which describe the existing physical connection between the P/C stations and the HP 1000. This initialization of required parameters is achieved by the user supplying information, in sequence, to the formatted screens of the configurator program.

The end result of the program is a disc FMGR file which is read by the PCIF monitor at run-time. The configurator uses a temporary Virtual Memory Area file to build or modify a user configuration according to information obtained on screens. The FMGR configuration file is only created or updated at the end of the configuration process (unless the configuration editor is aborted).

### 8.1.2 Overview

The configurator work file contains information derived from the preconfigurator program. This information includes a list of types of highways and P/Cs supported by the current PCIF/1000 monitor program, and may not be altered without using the preconfigurator again.

NOTE WD /PCIF/RUNTIME

The configuration process may be summarized as follows:

Start the configuration editor with the command CI> PCCON::cartridge reference (be sure your working directory is set to zero using the command CI> WD 0).

1. select configuration context (highway, P/C, or exit)
2. enter information specific to each handler:
   - select the current P/C or highway;
   - enter standard PC/highway information;
   - enter type-dependent PC/highway information;
3. enter general information
4. create or update the configuration disc file.

NO LONGER APPLICABLE
As of 26/5/87
Log on as PCIF

### 8.1.3 Proposed Values

When a screen is displayed, a value or values pertaining to the requested information may already be found in some fields. These are proposed values, and will either have been defined before by the previous configuration or will have been proposed by the configuration editor itself if the screen is being used with the defined configuration file for the first time. In either case the values may be altered to suit current requirements or left as proposed. The most important point to remember is that any value found in any field with any screen remains proposed until ENTER (or in some cases, the PREVIOUS SCREEN softkey) is pressed to validate the screen.

## 8.1.4 Validation

At the end of the configuration process a configuration file is created or updated. In this file a validation flag is set by the configuration editor which indicates that the associated configuration is complete and confirmed to be ready for use by the PCIF monitor at run-time.

The flag is not set if insufficient information has been supplied during the configuration or update, and the configuration editor displays a warning to this effect if the operator attempts to create a configuration file that lacks such data. This is to prevent run-time problems and to ensure that PCIF/1000 operates within previously defined parameters. However, it is possible for the operator to temporarily store an incomplete configuration file that may be validated at some later stage.

The following will be regarded as insufficient information and therefore not allowed by the configuration editor:

Example

The port number to be used on the MUX card is required information for the configuration editor. This number must be an integer between 0 and 7. Therefore if 0 is entered this information will be validated, but if a blank is entered it will be considered insufficient information and the validation flag will not be set for this configuration.

NOTE

> It may be useful to temporarily save an incomplete configuration, for example to get more information, or to get a hard copy of a list file.

## 8.1.5 Listing a Configuration

Listing a configuration is useful when searching for undefined parameters.

Making a listing of a configuration is possible by using the softkey LIST. This key is available at screens 2, 3, 4, 5, 7, 9 and 11. It enables the operator to record the configuration at various stages of the process. A list file must be defined in screen 2 at the beginning of the configuration, and subsequent commands to LIST fills this file with the available configuration data associated with the temporary work file.

Printing or displaying the listing can be achieved by one of three methods:

- Using the utility program PRINT.

- With the CI command LI, but this cannot be used while the configuration editor is still creating or modifying the associated configuration file.

- By allocating an LU to a file name with the FMGR command CR before starting the configuration editor. In this case the listing is placed directly into the LU from which it can be printed or displayed during the processing of the associated configuration file.

## 8.2 SCREEN SEQUENCING

### 8.2.1 General

Each screen contains information for the user and presents options. The selection of options enables the progression of the program which then proceeds to the next screen. A screen is generally validated by pressing the ENTER or the PREVIOUS SCREEN softkey. The next screen is then displayed.

The allocation of softkeys will vary slightly between screens, but the following are found in every screen and in the same physical softkey position:

- HELP: Displays an explanation of the use of the current screen. Pressing the terminal key "home-up" redisplays the current screen again, and pressing "shift, home-up" redisplays the help screen again.

- REFRESH: Removes the current screen and redisplays it showing the values it contained when the screen was first displayed.

- ABORT PCCON: Terminates the configuration editor and prevents generation of the configuration file. The operator will be asked to confirm the abort request by pressing the softkey again. If ABORT is not pressed again, the action of the depressed key is taken. If the ABORT is confirmed, the program may be restarted with the RU,PCCON command.

- LIST: Writes configuration data into a file on disc from which a copy can then be produced. There are methods available to achieve this. Refer to subsection 8.1.5 for details.

- PREVIOUS SCREEN: Available at certain screens only, this key also reads the information provided by the operator but provides an alternative screen sequence than ENTER.

- ENTER: This is not a softkey but it may be used at every screen to read the information supplied by the operator and display the next screen in sequence.

## 8.2.2 Operation

The following diagram displays the sequence of the Configuration Editor screens.

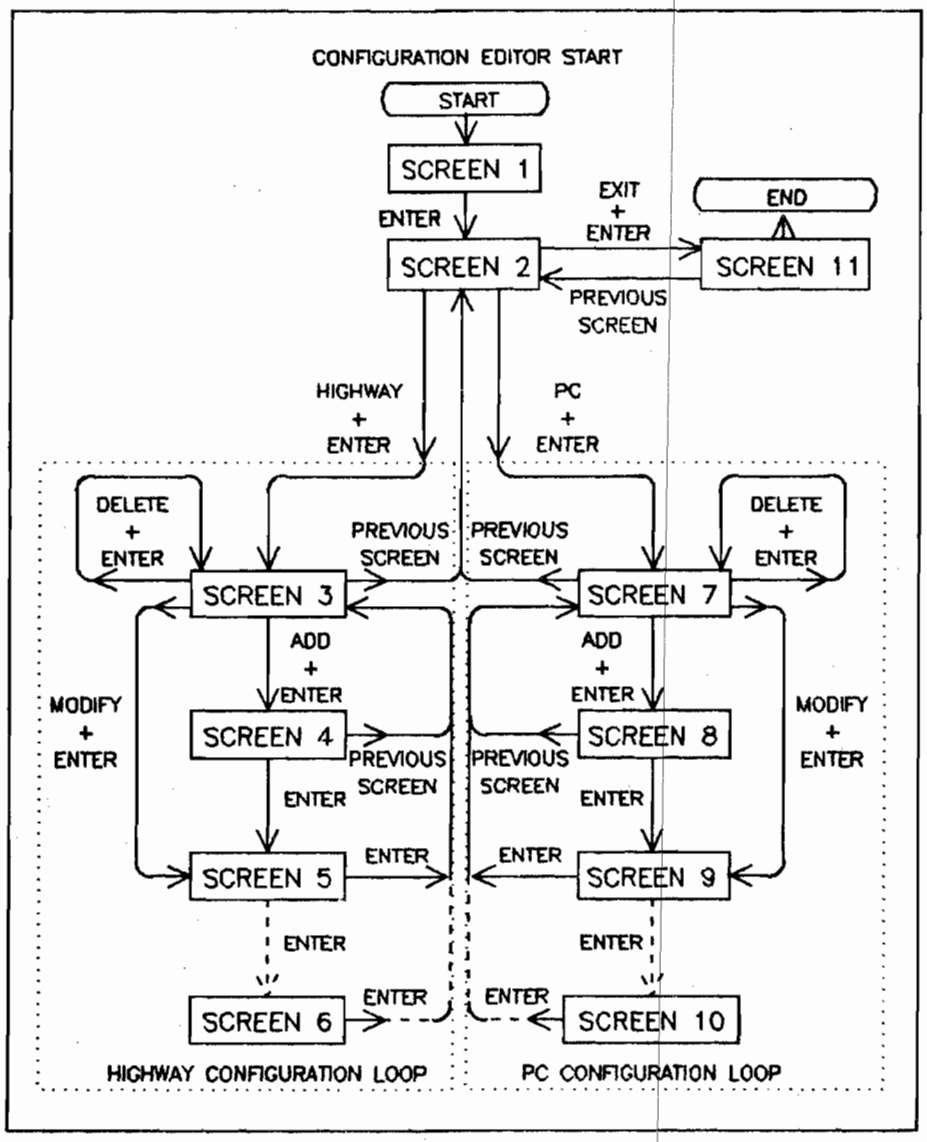
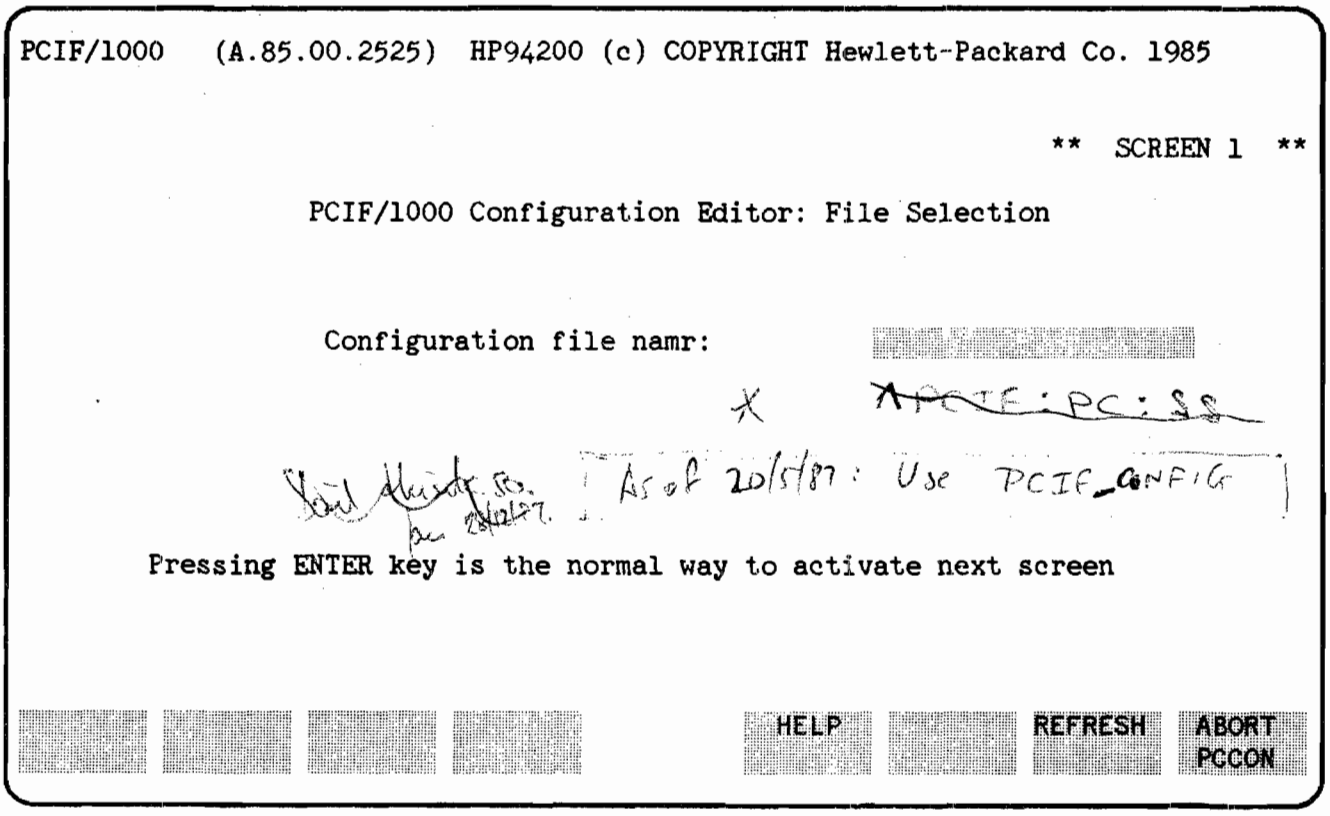



Figure 8.1 Screen Seqencing Diagram

**Note that screens 6 and 10 may not appear when configuring certain P/C brands. Refer to section 7.2.2 for general advice on entering information on screens.**

## 8.3 SCREEN DESCRIPTIONS

### 8.3.1 Screen 1: File Selection




Comments

The operator allocates a name for the configuration file to be subsequently made. By using the entered FMGR namr the configuration editor checks if this will be a configuration update or a new configuration.

- CONFIGURATION FILE NAMR is the name of the FMGR file that will be used by the PCIF run-time monitor for information on this particular configuration. It may have been used for a previous configuration, but requires updating. Also, as it is the namr associated with the call XQ,PCIF,NAMR to be made at run-time, it must be in FMGR namr format. It is advisable to enter the cartridge number. If this namr field is left blank it is assumed that this is a new configuration, of which the namr may be defined in screen 11 at the end of the configuration program. The file is not created, or updated until screen 11 is processed (as the configuration editor works on a work file).

The provided namr is validated by pressing the ENTER key. The program then compiles the required descriptors, and screen 2 is displayed. If a problem occurs, Screen 1 is redisplayed containing a relevant error message. If error FM006 is returned, you need to set your working directory to zero allowing FOCLO to find the FMGR screen files).

## Error Messages

The following error messages may be displayed with this screen after the ENTER key is pressed. Refer to Appendix A for further details.

- CO014 INVALID CONFIGURATION FILE NAME. This file already exists but is not a configuration file.

- CO013 UNDEFINED SOFTKEY

- CO036 FMP ERROR NUMBER -xxx

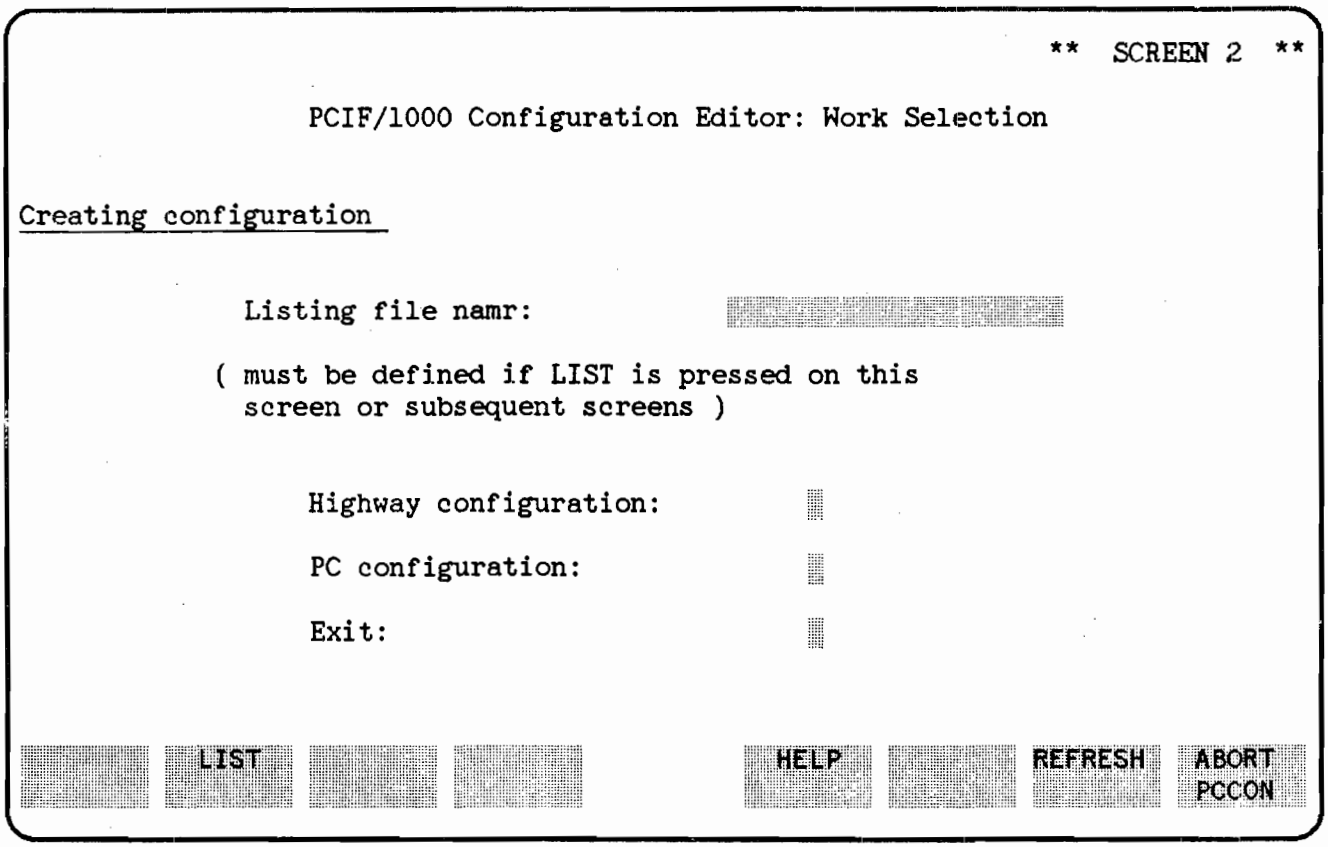
- DC036 FMP ERROR XX ON DESCRIPTOR FILE.

- DC0xx CORRUPTED DESCRIPTOR. One of the descriptor files used by this configuration editor has been corrupted. Run the preconfigurator again to determine which one.

- XXyyy ??. No message has been found for this error in the error message file ("PCERR).

Note: CO = Configuration error; DC = Descriptor error;
FO = Forms error; FM = File Manager error.

## 8.3.2 Screen 2: Work Selection




Comments

This is the main screen in the configuration editor program and it informs the operator whether the current configuration is a creation (CREATING) or a modification (MODIFYING). The screen provides the option for selecting the subsequent screen sequence, that is, for configuration of the highway or the P/C or to exit the editor program upon completion of both of these. Also, the operator may define a namr for providing a listing of the configuration.

A highway must always be configured before its associated P/C. This is because the configuration editor verifies the compatibility of this highway (defined by its number) with the required P/Cs.

- LISTING FILE NAMR is the name of the file that will be used to store a hard-copy listing of the current configuration. It must be in FMGR namr format. The softkey command LIST is used to output to the file. Refer to subsection 8.1.5. for details of this operation.

  PC/HIGHWAY CONFIGURATION selects the configuration of either a P/C or a highway. The operator must enter an X in the appropriate field.

  EXIT provides the opportunity to leave the configuration process through screen 11. The operator must enter an X.

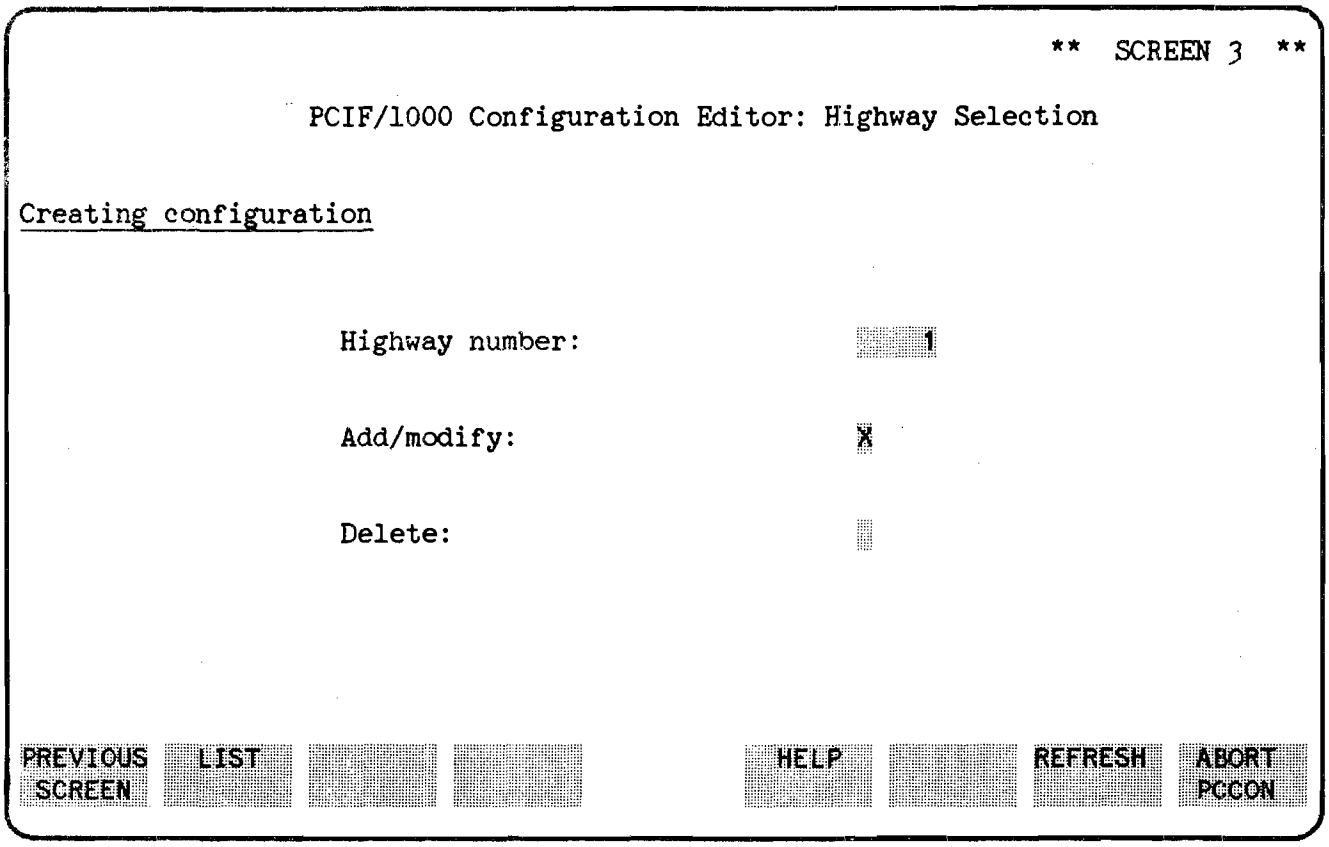
Error Messages

The following error messages may be displayed with this screen after the ENTER key is pressed. Refer to Appendix A for further details.

- C0001 REQUIRED FIELD MISSING - ENTER VALUE IN FIELD.
- C0004 X REQUIRED. Only the character X in upper or lower case is allowed to select options. The erroneous field(s) will be shown blinking.
- C0005 CONFLICTING ANSWER. Two or three alternative options have been selected at the same time. The erroneous fields will be shown blinking.
- C0013 UNDEFINED SOFTKEY.
- C0056 UNDEFINED LISTING FILE.
- FOxxx FORMS ERROR /
- XXyyy ??. No message has been found for this error in the error message file ("PCERR).

Note: CO = Configuration error; DC = Descriptor error;
FM = File Manager error;

## 8.3.3 Screen 3: Highway Selection




Comments

This screen will be the first displayed if the operator has opted to configure the highways. It asks the operator to allocate a number to the highway requiring configuration, which then becomes the highway's internal identifier. The operator also selects the type of highway configuration to be performed, either adding a new configuration, or modifying or deleting a highway from an existing configuration. The subsequent display of screens depends upon this selection.

- HIGHWAY NUMBER. The default value is n+1, where n is the largest highway number already allocated. The number must be a positive non zero integer, smaller than 32767. If the default is entered, the operator creates a new highway, and screen 4 will be displayed. If the value for an existing highway is entered, the operator must specify in the next two fields, whether a highway is to be modified or deleted.

- ADD/MODIFY. This field allows the operator to add a highway to an existing configuration, or to modify the highway defined with the highway number field. If you are modifying an existing highway, the next displayed screen will be screen 5.

- DELETE. Allows deletion of the selected highway from the current configuration, as long as this highway is not referenced by any P/Cs. Otherwise, this deletion will be refused until all the associated P/Cs are deleted (see screen 7). The

character X is entered to request the deletion. Having selected DELETE, the user can then press:

- ENTER to delete the selected highway and display screen 3.

  or

- PREVIOUS SCREEN to delete the selected highway and display screen 2.

Error Messages

The following error messages may be displayed with this screen after the ENTER key is pressed. Refer to Appendix A for further details.

- C0001 REQUIRED FIELD MISSING. ENTER VALUE IN FIELD. The relevant field is shown blinking.

- C0004 X REQUIRED. Only the character X in upper or lower case is allowed to select options. The erroneous field(s) is (are) shown blinking.

- C0005 CONFLICTING ANSWERS. Two or three alternative options have been selected at the same time. The erroneous fields are shown blinking.

- C0013 UNDEFINED SOFTKEY.

- C0019 NON EXISTING HIGHWAY NUMBER. Follows an attempt to delete a highway from a configuration for which it has not previously been defined.

- C0022 PC(s) ALREADY DEFINED WITH THIS HIGHWAY. Follows an attempt to delete a highway for which P/Cs are still defined.

- C0031 OUT OF RANGE DATA. The highway number must be an integer between 0 and 32767.

- C0041 FULL HIGHWAY LIST. No more than 64 highways may be defined.

- C0056 UNDEFINED LISTING FILE.

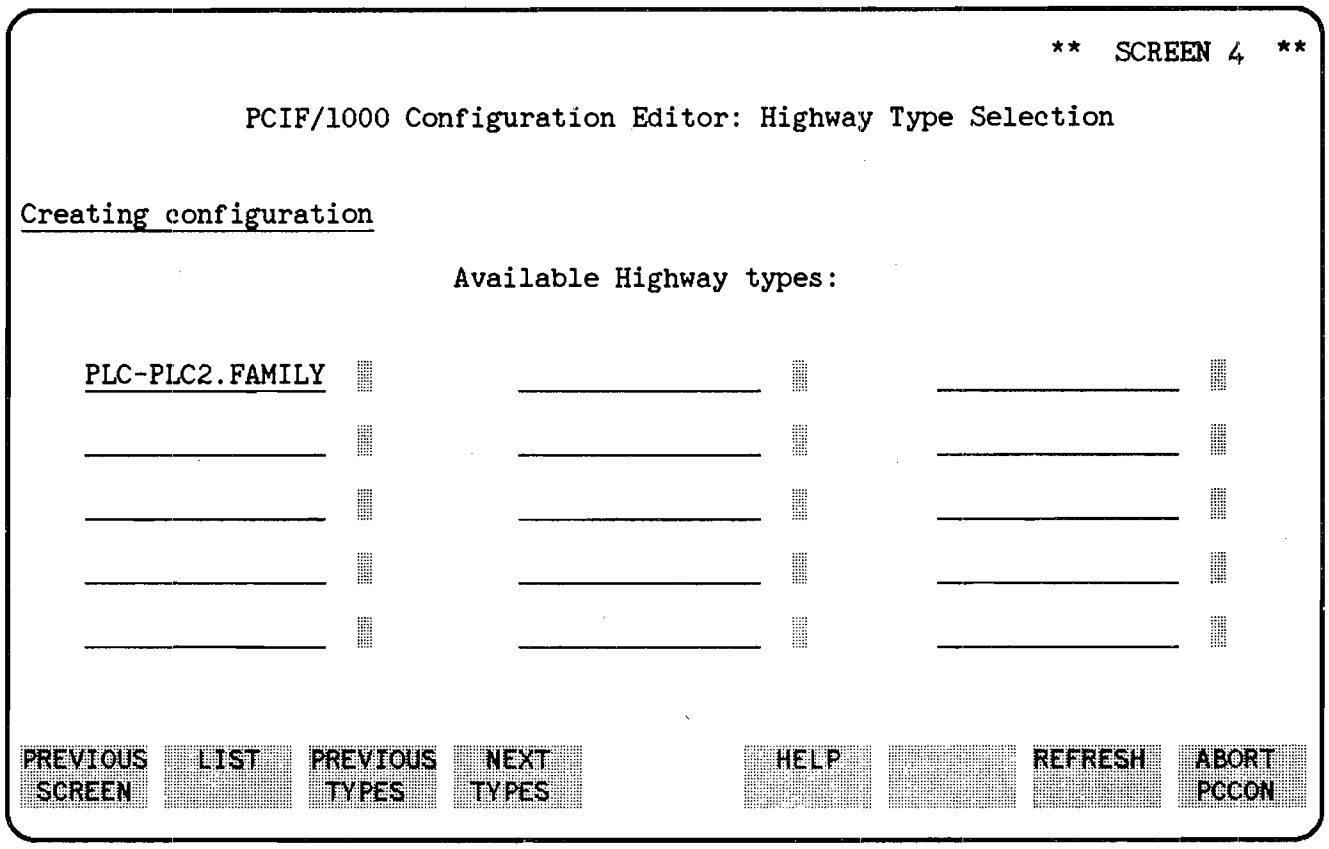
- C0065 DATA ENTERED DOES NOT MATCH FIELD TYPE. RE-ENTER. Concerns the highway number.

- FOxxx FORMS ERROR.

- XXyyy ??. No message has been found for this error in the error message file ("PCERR).

Note: CO = Configuration error; DC = Descriptor error;
FM = File Manager error;

## 8.3.4 Screen 4: Highway Type Selection




Comments

This screen is used to add a highway to an existing configuration. It displays a list of the supported highway types as found in the descriptor files. The screen shown above is an example showing the supported highway types for the Allen-Bradley handler. The names are the same as those displayed in the preconfiguration program at screen 2. The operator must select the highway type by entering an X in the appropriate square. Only one type can be selected at a time.

The list of supported types may be too long to be displayed on a single screen. For this reason two softkeys are defined for this screen. These are NEXT TYPES and PREVIOUS TYPES, and pressing the appropriate softkey displays either the next or the previous list. Once a highway type has been selected the operator must press ENTER or PREVIOUS SCREEN to validate this selection. Pressing PREVIOUS SCREEN redisplays screen 3, and pressing ENTER displays screen 5.

Error Messages

The following error messages may be displayed with this screen after the ENTER, PREVIOUS SCREEN or LIST key is pressed. Refer to Appendix A for further details.

- C0001 REQUIRED FIELD MISSING. ENTER VALUE IN FIELD. The relevant field will be shown blinking.

- C0004 X REQUIRED. Only the character X in upper or lower case is allowed to select options. The erroneous field(s) will be shown blinking.

- C0005 CONFLICTING ANSWERS. Two or three alternative options have been selected at the same time. The erroneous fields will be shown blinking.

- C0013 UNDEFINED SOFTKEY.

- C0056 UNDEFINED LISTING FILE.

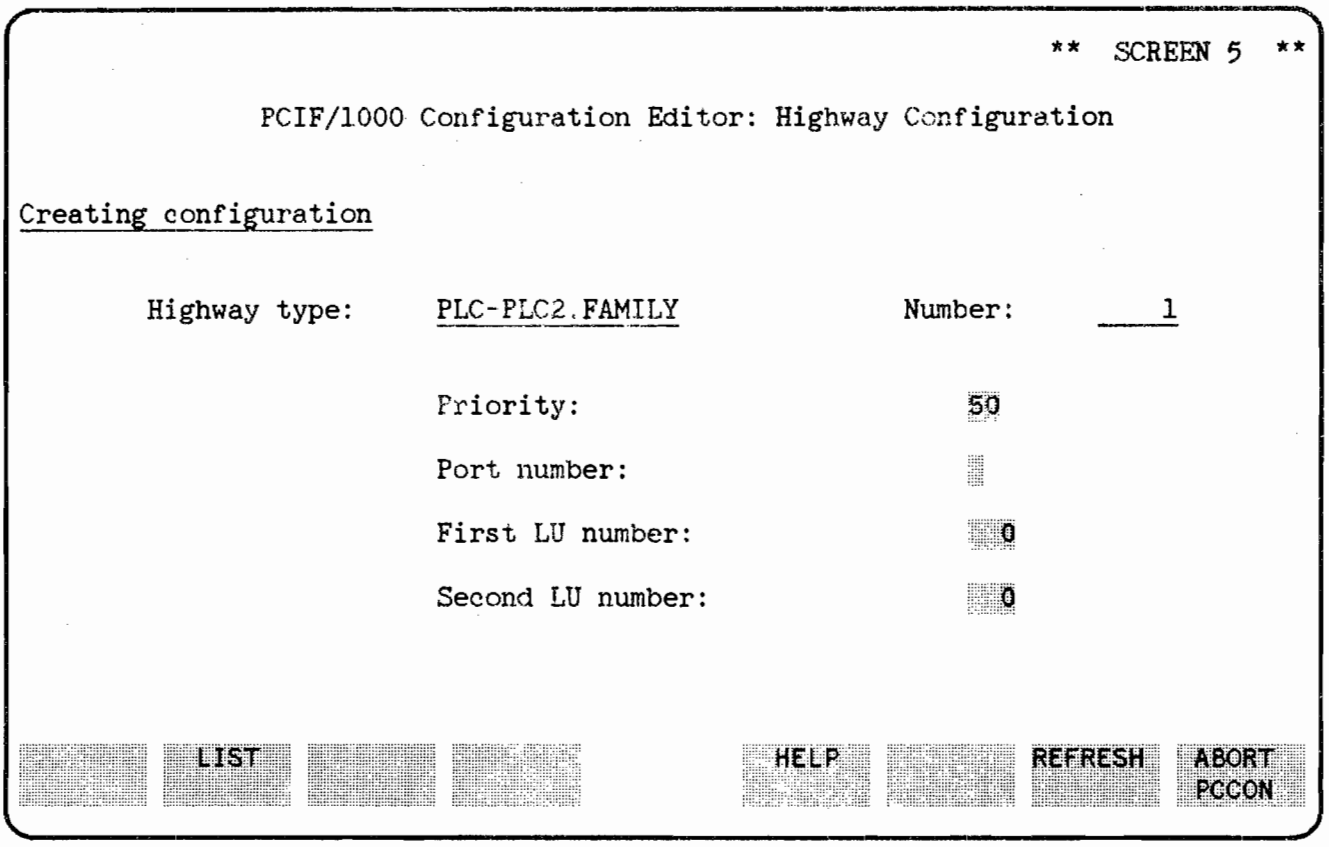
- C0080 NO AVAILABLE TYPES. ABORT PCCON. CHECK PRECONFIGURATION.

- FOxxx FORMS ERROR.

- XXyyy ??. No message has been found for this error in the error message file ("PCERR).

Note: CO = Configuration error; DC = Descriptor error;
FM = File Manager error;

## 8.3.5 Screen 5: Highway Configurationn




Comments

This screen asks for information whose nature is common to all highway types but whose content is specific for the type that has been selected. There will be an indication if this highway has been configured before (MODIFYING) or was never configured (ADDING). The number of required LUs will be dependent on the selected highway type. If only one LU is required then only one LU field will be displayed. The validation flag will not be set unless values are entered for all the fields.

- PRIORITY. This field asks for a parameter that defines the priority of this highway. The priority will be used by the PCIF monitor program at run-time and will be an integer between 1 and 99. The highest priority is 99, and the lowest priority is 1. If two requests are received simultaneously, the highest priority is handled first. This will integrate the PCIF software with the various data flows defined for a given application. The priority is determined by comparison with other modules. A value 0 may be entered temporarily but this must be redefined between 1 and 99, with this screen, to set the validation flag.

- PORT NUMBER. Provides the port number of this highway on the interface card, where this interface card is a MUX. The entered value sets the validation flag and should be an integer between 0 and 7. A blank may be entered if a port

number is not selected but in this case the validation flag will not be set. Any integer between 0 and 7 must be entered even if a MUX card is not used.

- FIRST & SECOND LU. Allocates either one or two LUs to the highway. A warning message will appear, after pressing ENTER, if the LU has already been allocated to another highway. Pressing ENTER a second time confirms the selection. A value 0 may be entered temporarily but this must be redefined between 1 and 255, with this screen. If two LUs are required they must have different numbers.

Error Messages

The following error messages may be displayed with this screen after the ENTER key is pressed. Refer to Appendix B for further details.

- CO010 INVALID LU NUMBER. This will be caused by entering an LU number that is either negative or higher than 255.

- CO026 INVALID PRIORITY NUMBER. The entered value is outside the range 0 to 99.

- CO043 INVALID PORT NUMBER. The port number must be between 0 and 7.

- CO045 INVALID LUs: FOR THIS HIGHWAY TYPE THE LUs MUST BE DIFFERENT.

- CO065 DATA ENTERED DOES NOT MATCH FIELD TYPE. RE-ENTER. Will either concern the priority, the LU number or the port number. The erroneous field will be shown blinking.

- CO013 UNDEFINED SOFTKEY.

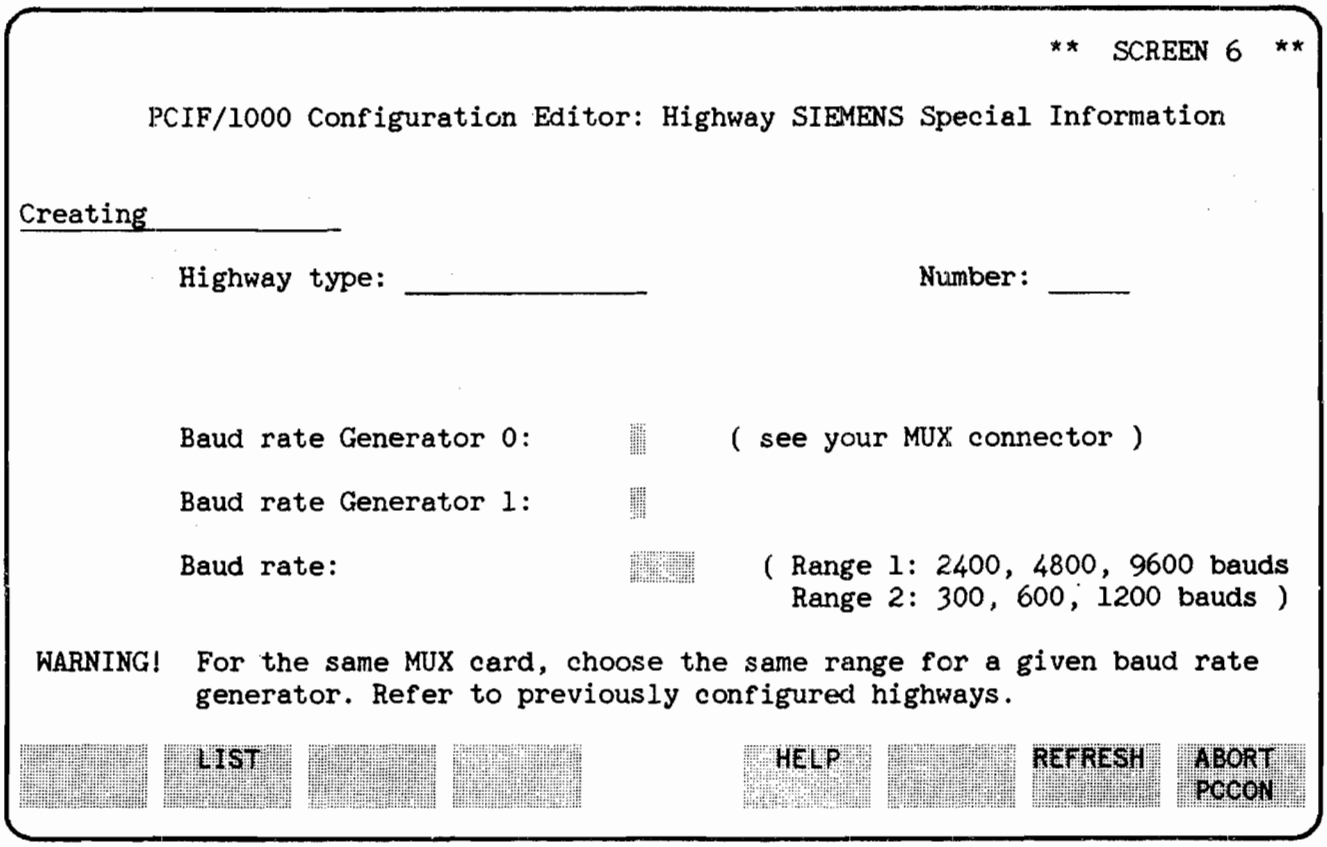
- CO056 UNDEFINED LISTING FILE.

- FOxxx FORMS ERROR.

- XXyyy ??. No message has been found for this error in the error message file ("PCERR).

Note: CO = Configuration error; DC = Descriptor error;
FM = File Manager error;

## 8.3.6 Screen 6: Highway Special Information




Comments

Screen 6 is only displayed for certain P/C brands. The example above applies to Siemens P/Cs. It is used to define the baud rate for the P/C computer dialog and, therefore, to ensure that the MUX and target PC's interface operate at the same baud rate.

If the highway has not already been configured, the message CREATING is displayed; otherwise, MODIFYING is displayed.

- BAUD RATE GENERATOR. Enter an X to define the range of the baud rate generator.

- BAUD RATE. Enter the required rate from either range 1 or 2.

Note that the operator defines the relationship between generator and baud range the first time a highway is configured for the MUX card. For example, if 0 is selected for the generator followed by a baud rate from range 1, then the relationship between 0 and a baud rate from range 1 is established. This relationship must be maintained for subsequent highway configurations on the same MUX card.

The screen may be passed without entering any information, but the configuration will not be validated.

Pressing ENTER redisplays screen 3. This allows either the highway configuration to be repeated or selection of screen 2 (Work Selection).

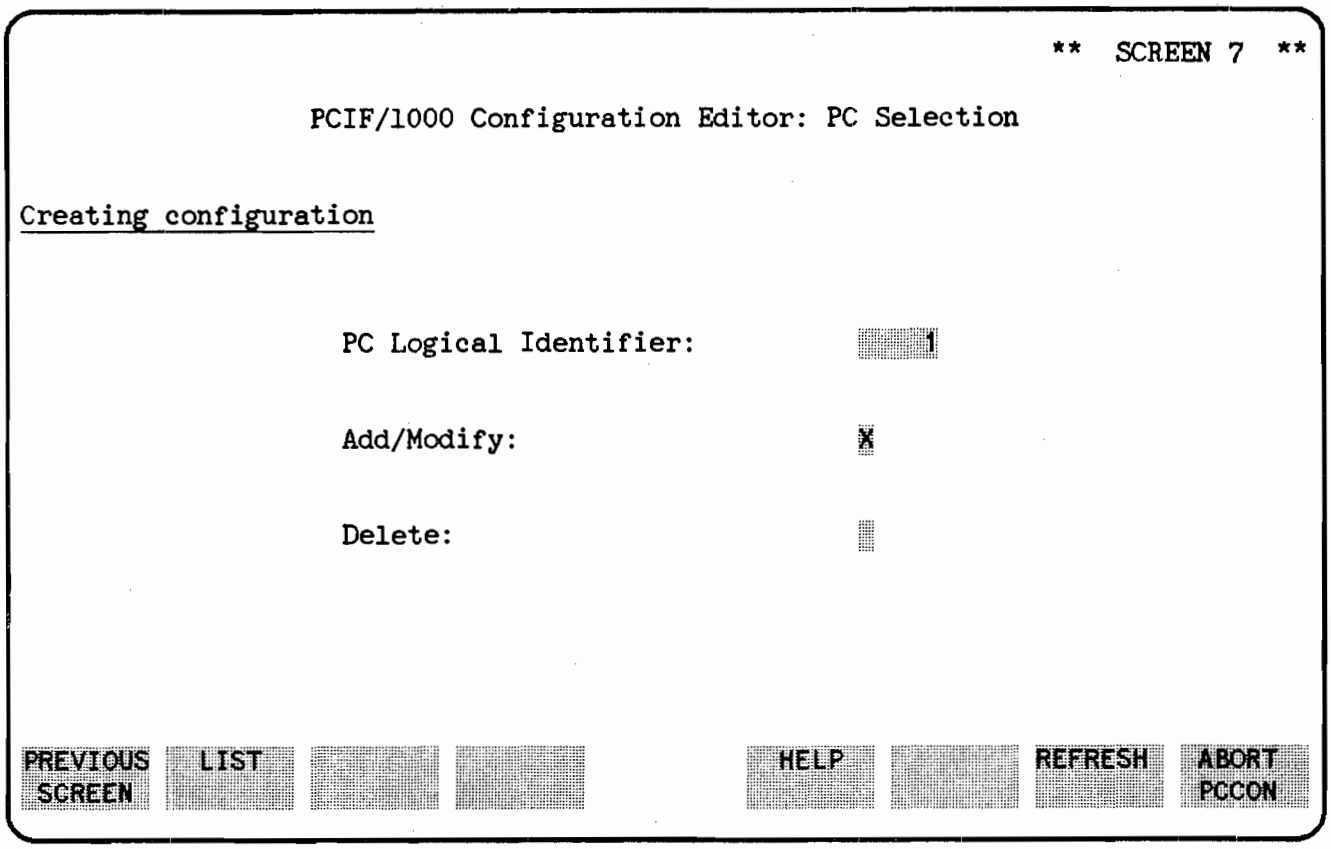
Error Messages

The following error messages may be displayed with this screen after the ENTER key is pressed. Refer to Appendix A for further details.

- DATA ENTERED DOES NOT MATCH FIELD TYPE. RE-ENTER. The baud rate must be an integer.

- INVALID BAUD RATE. The baud rate must be one of the listed values.

## 8.3.7 Screen 7: P/C Selection




Comments

This screen allows the definition of a new P/C by allocation of a logical identifier. The operator may also modify the configuration of a previously defined P/C by entering this PC's logical identifier.

- P/C LOGICAL IDENTIFIER. The default value is n+1, where n is the largest logical ID number already allocated. The number must be a positive integer, smaller than 32767. If the default is entered, the operator adds a new P/C, and screen 8 will be displayed. If an existing P/C identifier is entered, the operator must specify in the next two fields whether the P/C is to be modified or deleted.

- ADD/MODIFY. This field allows the operator to add a P/C to an existing configuration, or to modify the P/C defined with the P/C logical identifier field. If you are modifying an existing P/C, the next displayed screen will be screen 9.

- DELETE. Allows deletion of the selected P/C from the current configuration. The character X is entered to request the deletion. Having selected DELETE, it is possible either to PRESS:

- ENTER to delete the selected P/C and display screen 7.

- or PREVIOUS SCREEN to delete the selected highway and display screen 2.

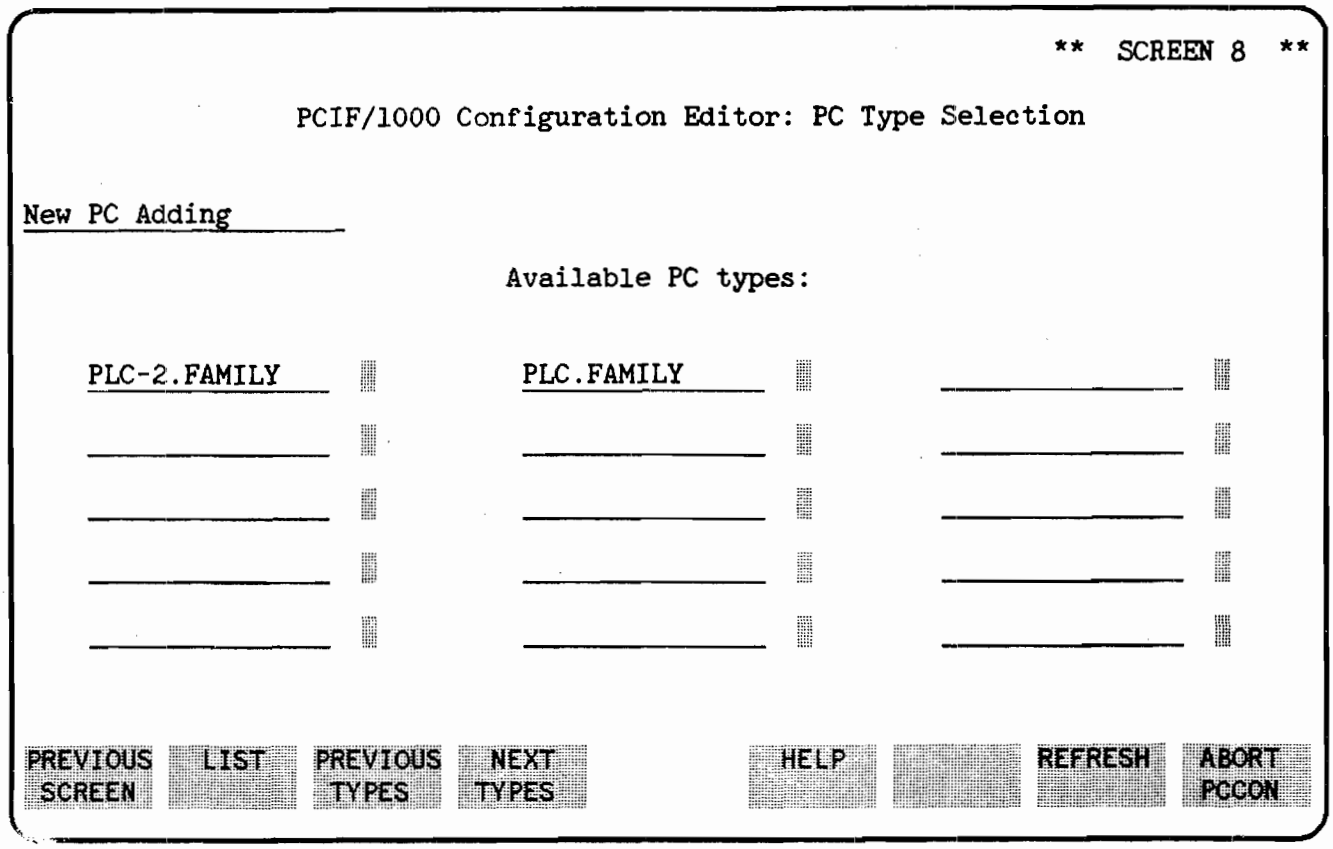
Error Messages

The following error messages may be displayed with this screen after the ENTER, PREVIOUS SCREEN or LIST (message CO056) key is pressed. Refer to Appendix A for further details.

- CO001 REQUIRED FIELD MISSING. ENTER VALUE IN FIELD. The relevant field is shown blinking.
- CO004 X REQUIRED. Only the character X in upper or lower case is allowed to select options. The erroneous field(s) is (are) shown blinking.
- CO005 CONFLICTING ANSWERS. Two or three alternative options have been selected at the same time. The erroneous fields are shown blinking.
- CO013 UNDEFINED SOFTKEY.
- CO025 NON-EXISTING P/C IDENTIFIER. An attempt has been made to delete an an undefined P/C from a configuration.
- CO031 OUT OF RANGE DATA. Either the entered value is larger than the maximum allowed (32767) or it isn't a positive integer value.
- CO042 FULL P/C LIST. No more than 256 P/Cs may be defined.
- CO056 UNDEFINED LISTING FILE.
- CO065 DATA ENTERED DOES NOT MATCH FIELD TYPE. RE-ENTER. The erroneous field is shown blinking.
- FOxxx FORMS ERROR.
- XXyyy ??. No message has been found for this error in the error message file ("PCERR).

Note: CO = Configuration error; DC = Descriptor error;
FM = File Manager error;

## 8.3.8 Screen 8: P/C Type Selection




Comments

This screen is only displayed if a P/C is to be added to the configuration as it shows the supported types of P/C as contained in the descriptor file. The names are the same as those displayed in the preconfiguration at screen 2. The P/C type is selected with either an upper or lower case X character in the appropriate field. Only one type can be selected at a time.

The list of supported types may be too long to be displayed on a single screen. For this reason two softkeys are available, NEXT TYPES and PREVIOUS TYPES, for displaying other parts of the list.

A P/C type selection must be made and is only validated if the ENTER or PREVIOUS SCREEN key is pressed. Screen 9 is displayed after ENTER, and screen 7 after PREVIOUS SCREEN.

See the following page for possible error messages.

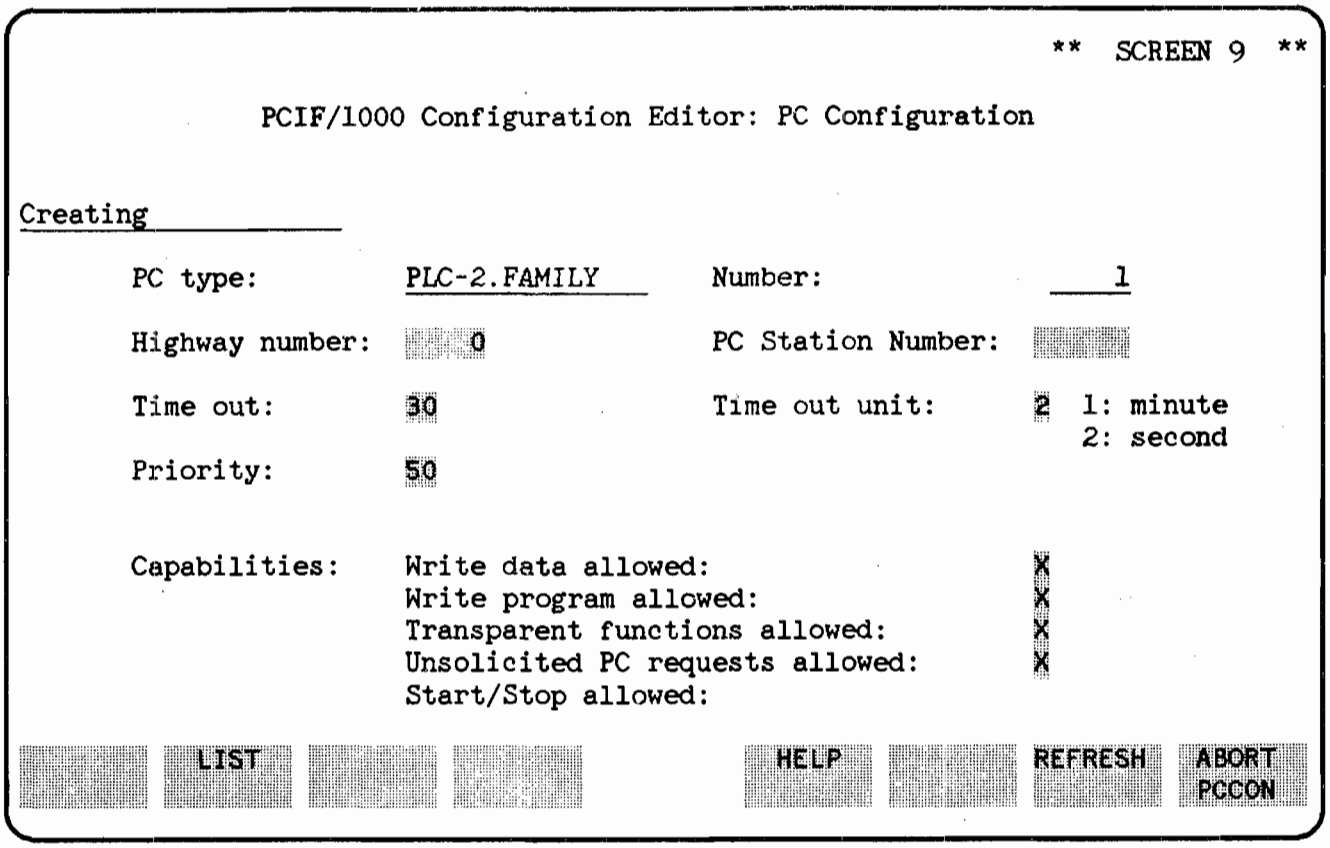
## Error Messages

The following error messages may be displayed with this screen after the ENTER, PREVIOUS SCREEN or LIST (message CO056) key is pressed. Refer to Appendix A for further details.

- CO001 REQUIRED FIELD MISSING. ENTER VALUE IN FIELD. The relevant fields are shown blinking.
- CO004 X REQUIRED. Only the character X in upper or lower case is allowed to select options. The erroneous field(s) will be shown blinking.
- CO005 CONFLICTING ANSWERS. Two or three alternative options have been selected at the same time. The erroneous fields are shown blinking.
- CO013 UNDEFINED SOFTKEY.
- CO056 UNDEFINED LISTING FILE.
- CO080 NO AVAILABLE TYPES. ABORT PCCON. CHECK PRECONFIGURATION
- FOxxx FORMS ERROR.
- XXyyy ??. No message has been found for this error in the error message file ("PCERR).

Note: CO = Configuration error; DC = Descriptor error;
FM = File Manager error;

## 8.3.9 Screen 9: P/C Configuration




Comments

This screen asks for information whose nature is common to any P/C type. The screen informs the operator if this P/C has not been previously configured (CREATING), or that it is to be updated (MODIFYING). It is possible to pass this screen without entering any information, but in this case the configuration will not be validated.

- HIGHWAY NUMBER. Requires the highway number on which this P/C is to be connected. The number was defined at screen 3, or if the highway was previously configured it can be found in the configuration list file. The highway type must be compatible with the P/C being defined. Only one P/C per highway is allowed where point-to-point connections are used.

- PC STATION NUMBER. This will only be displayed when the P/C is to be connected using a multipoint link, in which case the required information will be the physical address of the P/C. This may be 0, will be PC brand dependent and generally defined by switches on the interface card of the P/C. For more information refer to the relevant P/C brand chapter. NOTE: This value must be entered in "decimal".

- TIMEOUT and TIMEOUT UNIT. These two fields in combination define the timeout value for the P/C. This is the time period that a P/C request from an application program can remain unproccessed inside PCIF.

The timeout is a two digit integer, and the unit is selected to be either 1 or 2. If the unit is 1 the timeout value will be in minutes, if it is 2 the value is in seconds. This gives a possible timeout range of 0 seconds to 99 minutes. The proposed value is 30 seconds. A zero timeout value signifies no timeout defined for this P/C. This parameter is not required to set validation of the configuration, however if a timeout value is entered a unit must also be entered.

- PRIORITY. This parameter defines the priority of the associated handler that will be called by the PCIF monitor for this P/C. It may be an integer between 1 and 99 and will be significant when compared with the priority values of the other P/Cs or highways. The proposed value is 50. A zero may be entered in which case the validation flag will not be set. 99 is the highest priority and 1 is the lowest priority. If two requests are received simultaneously, the one with the highest priority will be handled first.

The following capabilities are selected if either an upper or a lower case X is entered in the appropriate field. They are proposed "off", unless a PC of the same type has been previously configured otherwise.

- WRITE DATA and WRITE PROGRAM. The writing of data or PC programs into the PC memory is only permitted if these facilities are allowed. Also refer to the descriptions of PC_WRITED and PC_WRITEP (Chapter 4) for more information.

- TRANSPARENT FUNCTIONS ALLOWED. Allowing this capability permits the application program to use the PC_TRANS access routine with this P/C. Refer to Chapter 4 for further information.

- UNSOLICITED REQUESTS. When this capability is allowed, the P/C can transmit unsolicited requests to the application program. Also refer to the PC_ENUNSOL and PC_DISUNSOL access routines description in Chapter 4.

- START/STOP ALLOWED. When this capability is allowed the P/C can accept PC_START and PC_STOP requests from the application program. Also refer to Chapter 4 for further information.

Before selecting any of these capabilities for a new P/C type, refer to the relevant P/C brand information to check that the selected capability is supported. An error message will be displayed if an unsupported capability is selected.

All supplied information for this screen is validated by pressing the ENTER key.

Error Messages

The following error messages may be displayed with this screen after the ENTER or LIST (message C0056) key is pressed. Refer to Appendix A for further details.

- C0001 REQUIRED FIELD MISSING. ENTER VALUE IN FIELD. The timeout unit field will be shown blinking if a timeout value has been entered without defining a timeout unit.

- C0004 X REQUIRED. Only the character X in upper or lower case is allowed to select options. The erroneous field(s) will be shown blinking.

- CO005 CONFLICTING ANSWERS. Two or three alternative options have been selected at the same time. The erroneous fields will be shown blinking.

- CO013 UNDEFINED SOFTKEY.

- CO019 NON EXISTING HIGHWAY NUMBER. No highway has been defined for the entered number.

- CO023 ALREADY ALLOCATED P/C STATION NUMBER. The entered P/C address has already been allocated to another P/C on the same highway.

- CO026 INVALID PRIORITY NUMBER. The entered priority value must be between 0 and 99.

- CO027 TIMEOUT VALUE OUT OF RANGE. A negative integer has been entered.

- CO030 NON-COMPATIBLE HIGHWAY. The entered highway number cannot support the entered P/C type.

- CO031 OUT OF RANGE DATA. The highway and P/C number must be an integer between 0 and 32767.

- CO046 INVALID TIMEOUT UNIT. Only units 1 or 2 are allowed.

- CO047 ENTERED CAPABILITY DOES NOT MATCH WITH THE FEATURES OF THIS P/C.

- CO056 UNDEFINED LISTING FILE.

- CO063 ALREADY DEFINED PC ON THIS HIGHWAY (POINT TO POINT LINK). The entered highway has already been configured with another P/C on a point to point link.

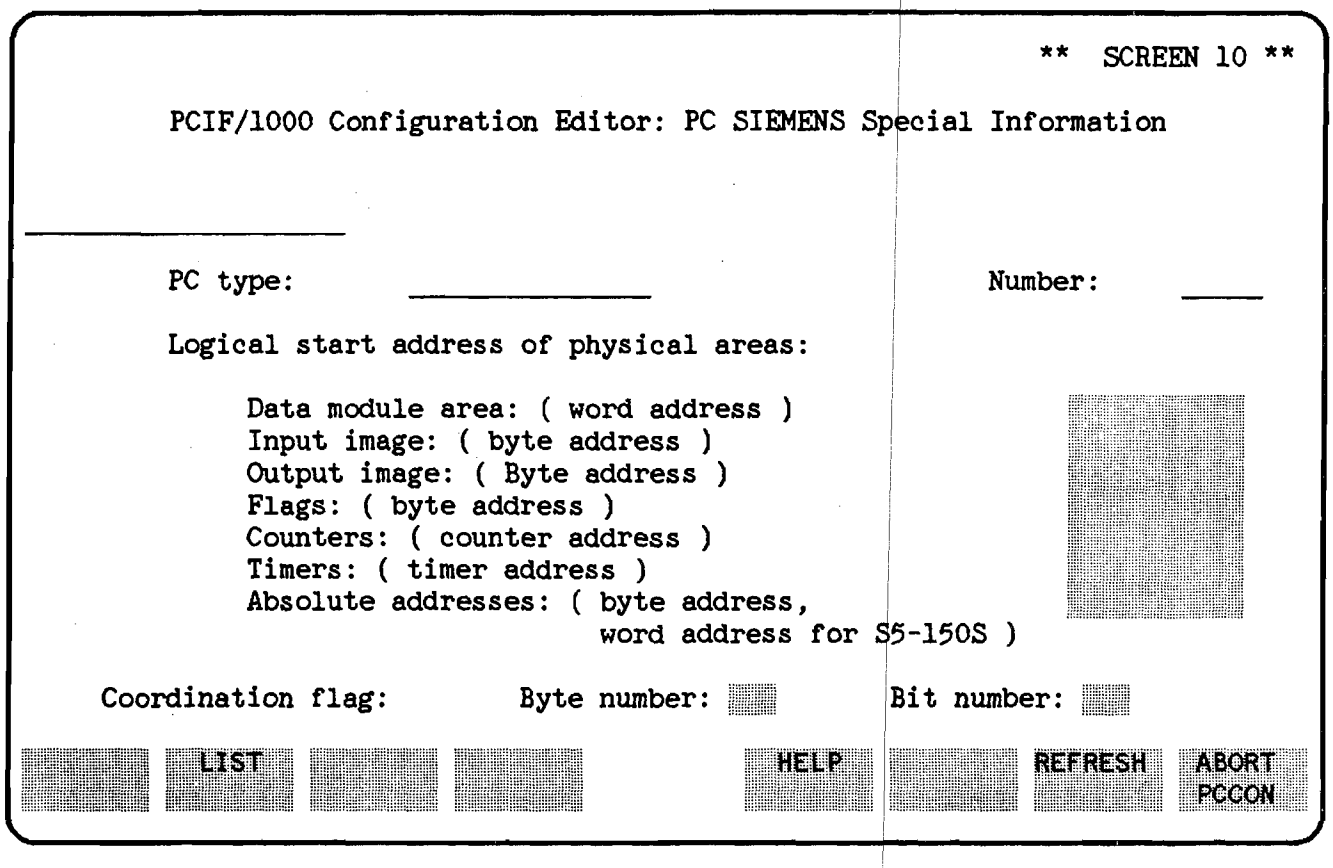
- CO065 DATA ENTERED DOES NOT MATCH FIELD TYPE. RE-ENTER. This may concern any of the following fields: timeout unit; timeout value; priority; highway number or the P/C address. The erroneous field will be shown blinking.

- FOxxx FORMS ERROR.

- XXyyy ??. No message has been found for this error in the error message file ("PCERR).

Note: CO = Configuration error; DC = Descriptor error;
FM = File Manager error;

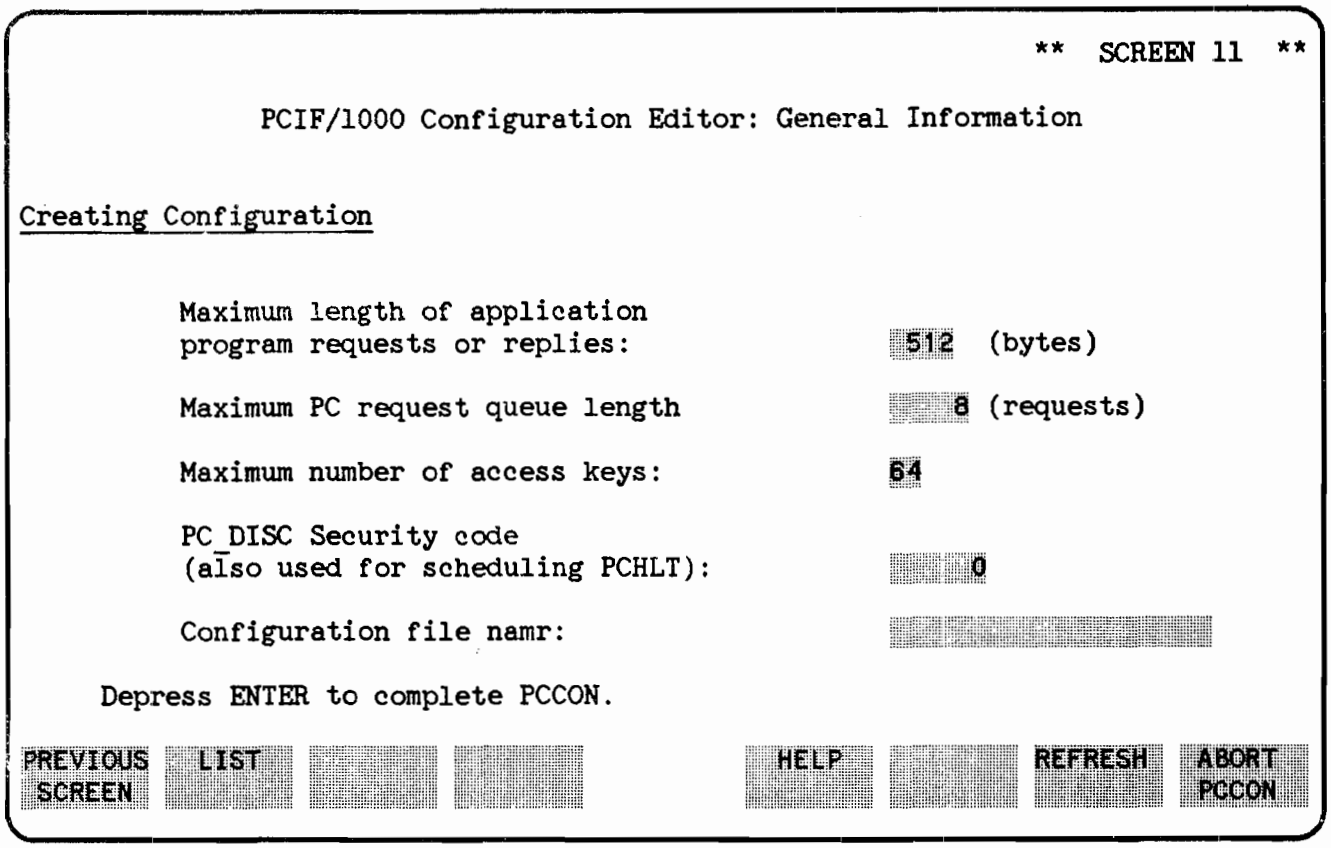
## 8.3.10 Screen 10: P/C Special Information




Comments

Screen 10 is only displayed if the P/C being configured is of a certain brand. For example, P/Cs manufactured by Siemens will require such information as the P/C memory map to be defined. Details of this screen are contained in the specific handler manual. The content of this screen will vary when other P/Cs are being configured.

## 8.3.11 Screen 11: PCIF General Information




Comments

This screen asks for information that is common to all the modules of PCIF/1000 and that generally describes the highway and P/C configuration that has taken place.

NOTE: If you wish to list the values displayed on screen 11 in a list file, you must do the following before pressing the ENTER key:

* Press PREVIOUS SCREEN to return to screen 2. This validates the entered data;

* On screen 2, select the exit option and press ENTER. Screen 11 will be redisplayed;

* On screen 11, press LIST to store the validated data in the list file.

- MAXIMUM LENGTH OF APPLICATION PROGRAM REQUESTS OR REPLIES. The size of memory taken for storage of waiting messages is dependent upon the message length. For this reason the maximum length of the requests and replies must be defined. The information required for this field is the maximum length in bytes. It is a positive integer smaller than 1025 which simultaneously defines request and reply length. If the entered value is 0 the field is assummed not to be selected

and the validation flag is not set. The proposed value is 512 bytes. For presetting the system memory see section 6.5.5.

- PC REQUEST QUEUE LENGTH. The size of memory taken by requests waiting to be processed is dependent upon the amount of the requests and their length. With this field the number of these requests can be limited. A positive integer can be entered to define the maximum number of requests waiting to be completed for the same P/C. If the entered value is 0 the field is assumed not to be selected and the validation flag is not set. The proposed value is 8 requests.

- MAXIMUM NUMBER OF ACCESS KEYS. Access keys are used by certain PC access routines. For details refer to Chapter 4. The required information for this field is a positive integer smaller than 65. If the entered value is 0 the field assumed not to be selected and the validation flag is not set. The proposed value is 64.

- SECURITY CODE. This code is used in conjunction with the P/C access routine PC_DISC (see Chapter 4) and also with the stopping of the PCIF monitor (see Chapter 9). The required information is an integer between -32768 and 32767. The proposed value is 0. This field has no influence on the validation flag.

- CONFIGURATION FILE NAME. This field allows the definition of the name of the configuration data just compiled. The name will be the same as that entered in screen 1 and may be changed if required, although the FMGR namr format must be maintained. A FMGR disc file will be created with the name entered in this field.

When ENTER is depressed the configuration editor will store the data entered during this configuration into the disc file named in the namr field. If this file already exists for a PCIF configuration, a warning is displayed:

DO YOU REALLY WANT TO REPLACE THE CONFIGURATION FILE? YES: SAME KEY.

The ENTER key must be pressed again for the affirmative and the file is overlaid.

The file name in the FMGR namr field must be changed if this configuration is to be stored in a different file from the original file (as given in screen 1).

If the file already exists but not as a PCIF configuration file the configuration editor will not overlay it and a error message will be displayed.

If the configuration contains unvalidated data the operator is reminded by a message before the configuration is stored. The message states:

NON-VALIDATED CONFIGURATION. FOR CONFIRMATION DEPRESS THE SAME KEY.

Pressing ENTER then stores the configuration without validation, which may be useful for recording incomplete configurations awaiting confirmation of detail. Note that the reason for validating configurations is to guarantee compatibility with the application program at run-time.

Error Messages

The following error messages may be displayed with this screen after the ENTER, PREVIOUS SCREEN or LIST (message C0056) key is pressed. Refer to Appendix A for further details.

- C0001 REQUIRED FIELD MISSING - ENTER VALUE IN FIELD. The namr field will be shown blinking if no file name has been defined.

- C0014 INVALID CONFIGURATION FILE NAME. The file already exists but not as a configuration file.

- C0031 OUT OF RANGE DATA. This may concern all the fields except the configuration file namr. The erroneous field will be shown blinking.

- C0056 UNDEFINED LISTING FILE.

- C0065 DATA ENTERED DOES NOT MATCH FIELD TYPE. RE-ENTER. This may concern any of the following fields: maximum request length; P/C request length; maximum number of request keys or the security code. The erroneous field will be shown blinking.

- FOxxx FORMS ERROR.

- XXyyy UNDEFINED ERROR.

Note: CO = Configuration error; DC = Descriptor error; FM = File Manager error;

# Chapter 9
# RUN-TIME OPERATION

## 9.1 INTRODUCTION

This chapter describes how the various components of PCIF/1000 interact at run-time. The chapter assumes the following:

- That the installation and configuration procedures have been followed.
- That a valid configuration file exists.
- That an application program has been written and contains calls to P/C Access Routines.

Figure 9.1 shows the utility programs of PCIF/1000 that are activated at run-time with the call:

XQ,PCIF,<configuration file name>,<emergency file name>

The run-time operation of PCIF/1000 can be divided into three phases.

Initialization Phase
This is the immediate result of the run-time command and is described in subsection 9.2.2.

Run-Time Phase
The period between PCIF_OPEN and PCIF_CLOSE when an application program issues P/C Access Routines. This is described in subsection 9.2.5.

Stopping PCIF/1000
Another program must be scheduled to deactivate the PCIF/1000 monitor. This is called PCHLT and is described in section 9.4.

Section 9.3 describes possible errors that may occur during the three run-time phases. Certain errors may abort the application program or the PCIF monitor, while others are not fatal and only create warning messages. All the error messages are listed in Appendix A, and in some cases advice is given on possible corrective action.

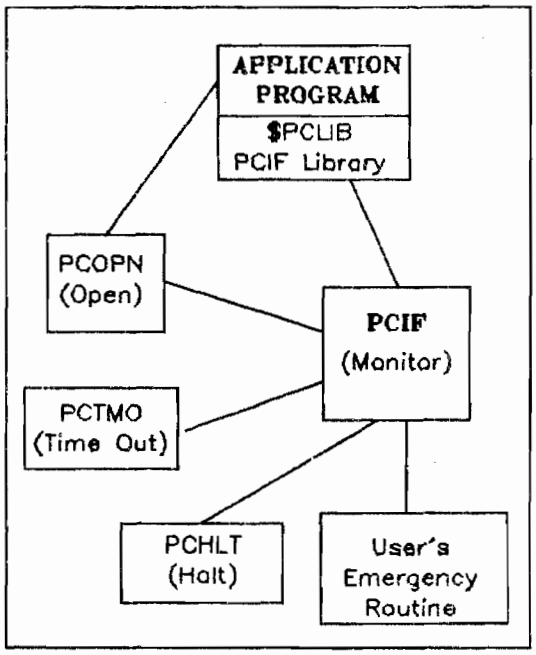



Figure 9.1 Run-Time Utility Programs

The purpose of the programs shown in Figure 9.1 are as follows:

PCIF : This is the central run-time part of the subsystem. It analyzes the various requests initiated by the application program and processes them.

PCOPN: Establishes the connection between the application program and PCIF.

PCHLT: This program is scheduled by the user from a terminal to stop the run-time PCIF subsystem.

PCTMO: This program is scheduled by PCIF and put in the time list on a regular basis. Every time it is activated it sends a message to the MONITOR.

PCDMX: This program is scheduled by PCIF to download Z80 code to the MUX. The required code (P/C brand dependent) is found in a file on disc.

$PCLIB: This is the library to be used by every application program requiring interaction with a P/C through PCIF and will contain the P/C Access Routines listed in Chapter 4. It is appended to the application program during the linking process of this program.

EMERGENCY: This will be scheduled by PCIF when the PCIF run-time subsystem is stopped either normally or abnormally. This program is user written, its name is user definable and must follow the RTE-A naming rules.

## 9.2 STARTING PCIF

### 9.2.1 RTE Command

The PCIF subsystem must be started before any application program access to PCIF can be made. To do this, the various program parts of the PCIF subsystem must be assigned an ID segment as follows:

```
RP,PCOPN::crn    RP,PCHLT::crn
RP,PCTMO::crn    RP,PCDMX::crn
RP,PCTST::crn
RP,PCIF::crn
```

Having assigned ID segments the subsystem can be started with the following command:

XQ,PCIF,<configuration file name>,<emergency file name>

This command can be issued from either the WELCOME file, a terminal or an RTE program written by the user. AUTOR should be RP'ed in the WELCOME file if a power failure recovery is being made with Gould-Modicon P/Cs.

If the command is not issued from the WELCOME file, the following message appears when you are logging off:

```
PCTMO (System Utility)
PCIF  (System Utility)
Continue, Logoff, Background, or ? [C]?
```

In response, simply type an "L" to logoff. This message should only appear once, when PCIF is initially started up from a user's session.

Note: The "RU" command may be used instead of the "XQ" command to start PCIF. The "RU" command, however, is an execute with-wait command, and the CI (the A-Series user interface) will wait for PCIF to finish before issuing another prompt. With "XQ", you get "CI" back immediately because "XQ" means execute without waiting for PCIF to finish.

### 9.2.2 Initialization Phase

The XQ,PCIF command verifies that the configuration file name is valid and that this file contains a complete configuration (i.e. the validation flag has been set), also that an ID segment exists for the user provided emergency program if this program has been called within the XQ,PCIF command.

Program PCIF then loads the configuration description into Extended Memory Area (EMA) of the HP 1000, asks for the various RTE resources (e.g. class numbers) and verifies that the required resources are present, such as LUs and ID segments for the various components of the subsystem.

Next, using information found in the configuration file the appropriate Z80 code is downloaded to the MUX with the PCDMX program for those handlers using the 12041B MUX card. Following this, the PCTMO program is scheduled and subsequently puts itself

into the timelist. Then the program PCOPN is scheduled and passed parameters by the PCIF monitor.

Refer to Figure 9.2 for an overview of these operations.

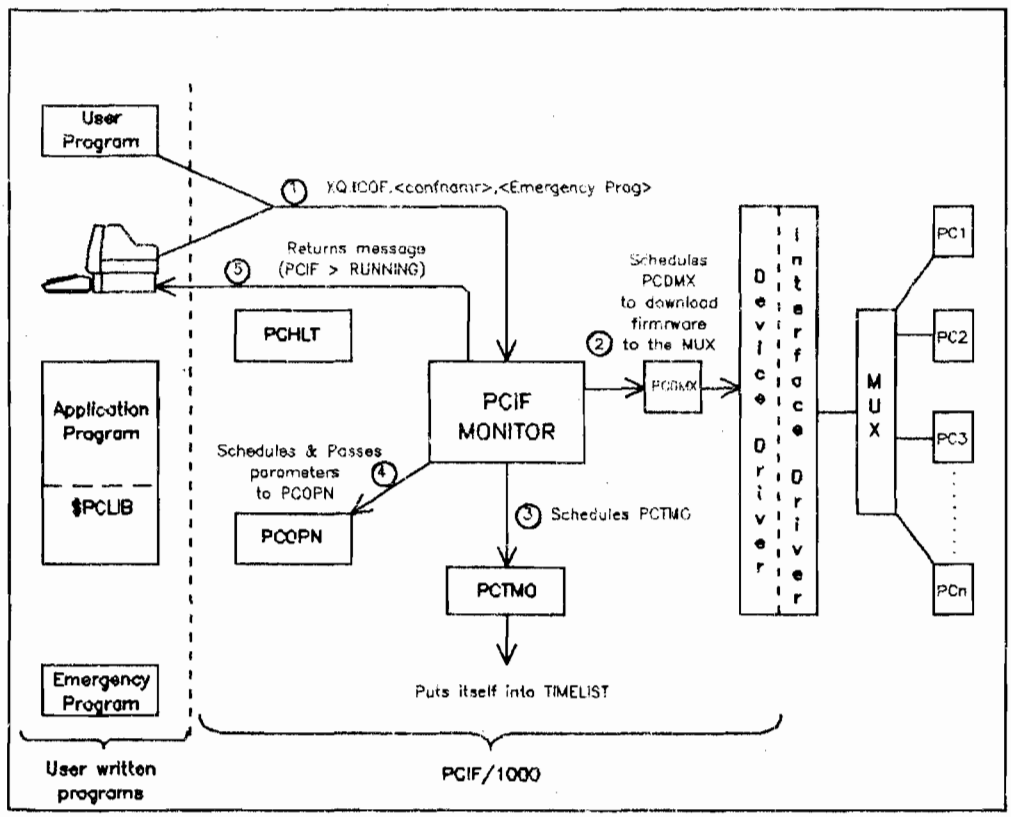



Figure 9.2 Initialization Phase

**Finally**, if everything is correct, the message "PCIF> running" is sent to the terminal that issued the XQ,PCIF command. The initialization phase is then complete. Otherwise an error message is displayed and the initialization process is terminated.

The second parameter of the run-time command (the emergency program name) provides the name of an RTE program that is scheduled when PCIF Monitor ends, either normally or abnormally. The purpose of this optional parameter is to allow a user to make an emergency decision in case of failure or to attract attention when PCIF stops normally.

Note that when the initialization phase is complete, all the P/Cs existing in the configuration are set to a PCIF disconnected state.

## 9.2.3 Messages

During the run-time operation of PCIF/1000, messages are displayed on the scheduling terminal and will have the following format:

```
PCIF>warning : xxxxxxxxxxxxxxx        [1]
or
PCIF>ABORT   : xxxxxxxxxxxxxxx        [2]
```

[1] This is an information message. The run-time PCIF/1000 subsystem will stay operational despite the occurrence of an abnormal event. The general events are described in this section, while a detailed description and the possible action to be taken are listed in Appendix A.

[2] This is an indication that a serious malfunction has occurred and that the PCIF/1000 subsystem has ceased to operate. Reasons for this malfunction are listed below. The problem should be corrected and PCIF restarted as indicated previously (normal start).

Both types of message have the following structure :

XXyyy zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz....

XX is a two character tag indicating the part of the subsystem that has encountered the problem. These characters may be:

MI the error was detected during the initialization part of PCIF. At the time of the error neither the application program nor a P/C was able to start any dialog.

MK the problem has been detected by PCIF during the normal operation of PCIF (i.e. after the message "PCIF>running" was displayed).

DM the download program (PCDMX) has detected a problem.

TM the timeout program has detected a problem.

ST the PCHLT program has detected a problem.

FM this is an extra message that indicates what kind of FMP error has occurred. It is usually displayed as a further explanation of a previous message. For example:

```
PCIF>warning : DM008 FMP error on download file. FMP error -006
PCIF>warning : FM006 File does not exist.
```

yyy is a three digit number that indicates the error number.

zz....is an ASCII string that provides a short explanation of the error.

Appendix A provides a complete explanation of all the possible error codes and messages. The message text can be localized by the user into any local language. (see section 9.2.4).

An error message will be displayed in the following format when PCIF is unable to find the corresponding error file:

PCIF> XXyyy ??

The operator will then need to refer to Appendix A to find an explanation of the message.

## 9.2.4 Localizing Error Messages

The user can localize message files by changing the text that appears after the error number. For example:

The error message "MK042 impossible to answer to application program" can be translated into French to read "MK042 contact rompu avec le programme d'application" or into German to read "MK042 Unmoeglich dem Anwenderprogram zu antworten".

The error message file used by the monitor program is called "PCMER, the one used by the P/C Access Routines is called "PCMSG, and the file used by the configuration programs is called "PCEER.

Note the following when localizing the "PCMER or "PCEER files:

(1) The message format listed in 9.2.3 (XXyyy) must be maintained whatever the local language. The part XXyyy must not be modified and only the next 70 characters (zzzzzzz...) can be changed.

(2) XX036 FMP error number -xxx
Where XX can be a CO or a DC error, the constraint listed in (1) is still true, but these messages must finish with -xxx.

(3) PL xxx preconfiguration listing, LL xxx configuration listing.
For these messages, the number of characters must be:

| | |
|---|---|
| PL001... 39 | LL000... 6 |
| PL002... 74 | LL001 |
| PL003 | to |
| to | LL037... 39 |
| PL007... 33 | LL038 |
| | to |
| | LL042... 16 |

DO NOT CHANGE THE ORDER OF THE MESSAGES IN ANY FILE.

## 9.2.5 Running Application Programs

After successfully completing the initialization process, PCIF/1000 is ready to interact with application programs and so begin the run-time phase. The run-time phase must always begin with the P/C Access Routine PCIF_OPEN . If this call is made before the initialization process is complete (i.e. the "PCIF running" message has not appeared) it will be rejected. The PCOPN program then establishes a link between the application program that issued the PCIF_OPEN and the PCIF monitor.

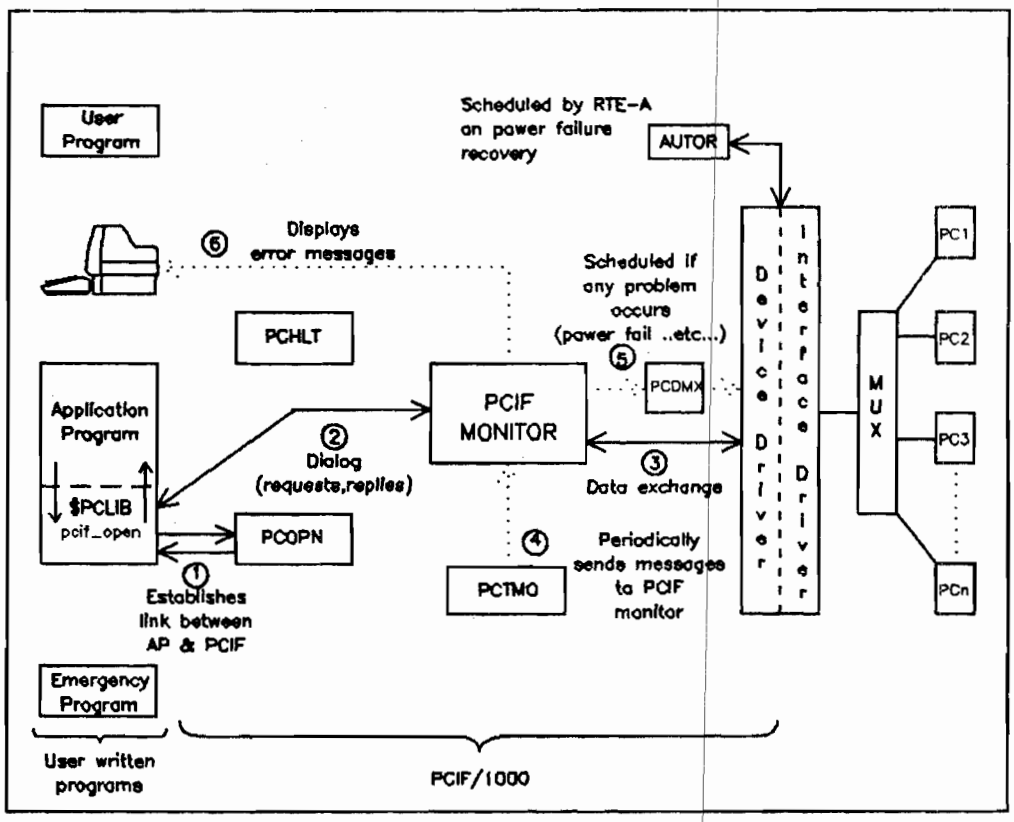



Figure 9.3 Run-Time Phase

Various P/C Access Routines can now be issued by the application program to achieve the required dialog of requests to P/Cs and their associated replies. All the available routines are described in Chapter 4. During the PC-application program dialog the PCTMO program periodically sends messages to the PCIF monitor to check its operational status.

## 9.3 ERRORS DURING PCIF OPERATION

Several types of errors may appear while PCIF is operating:

- error in application program calls to PCIF;
- abort of application program before closing PCIF;
- error in PCIF monitor or other components;
- power fail and P/C problems.

### 9.3.1 Application Program Calls

Any errors that occur during the application program-P/C dialog (i.e. the run-time phase) are reported to the calling application program by the STAT parameter of the appropriate P/C Access Routine. This parameter is set to zero if no errors occur or an integer greater than zero if a problem occurs. The value returned will signify what the problem is, and this STAT value may be converted into an ASCII string with a special P/C Access Routine called PCIF_ERROR. The ASCII string may then be used to display a message on a terminal. These messages are found in a file called "PCMSG which may be localized if necessary.

### 9.3.2 Abort of Application Program

This would be caused by a serious error in the application program such that it is aborted by the RTE operating system. When this occurs the resources allocated for PCIF (such as locked P/Cs, unsolicited requests enabled by this application program, pending P/C requests etc) remain active and must be deactivated, locked P/Cs being the worst case. PCIF will achieve this aim ONLY if it is aware that a serious error has occurred.

Example 1: A request is made by an application program, and when the PCIF monitor tries to return a reply it discovers that the application program is not accessible. The monitor will then clear all current requests, unlock all P/Cs and disable all unsolicited requests made by this program. A message is then sent to the PCIF scheduling terminal:

```
MK032 impossible to send reply to PCIF_OPEN.
MK042 impossible to answer to the application program.
```

Example 2: The application program has been terminated but without the knowledge of the PCIF monitor. If another program with the same name (and the same session number if the RTE-A session monitor is on) sends a PCIF_OPEN call, then the PCIF monitor removes all the oldest requests and sends a message to the user's terminal:

```
MK050 two PCIF_OPEN for the same user, oldest flushed.
```

As the ACCESS KEY which is used for interaction has been lost, PCIF finds all the locked PCIF resources and frees them. From a functional point of view this is equivalent to the concerned application program requesting PCIF_CLOSE.

## 9.3.3 Error in PCIF Components

Some errors may cause PCIF to be terminated by RTE, in which case there is no chance of recovery, or they may be caused by problems with the PCIF components which may be corrected.

When PCIF has been terminated, the "emergency" program (if given in the XQ,PCIF command) is immediately scheduled. After which all the RTE resources used by the PCIF monitor are removed from the system. Application program calls waiting for answers to requests will receive a completion status informing that a loss of contact has occurred with PCIF. However, this may not be possible in severe error situations, such as PCIF monitor terminated by RTE-A or by the operator. For this reason it is advisable to schedule PCIF from a user written program.

Whenever PCIF is stopped or terminated, the emergency program is activated by PCIF with the following command:

```
CALL EXEC (ECODE, NAME, PRAM1, PRAM2)
```


where:

ECODE = 10+no abort bit. This is an immediate schedule without wait.

NAME = a parameter containing the "emergency program name" (as provided in the XQ,PCIF command)

PRAM1 = one word containing 20547 (or 50103 octal) which is a password.

PRAM2 = one word containing a completion code value, as follows:
- (1) PCIF stopped before the end of initialization (the "PCIF>running" message has not been displayed)
- (2) PCIF was stopped normally.
- (3) PCIF was stopped by the operator, but some application programs were still connected to the PCIF monitor. No PCIF_CLOSE was issued.
- (-2) PCIF was terminated due to abnormal conditions and the PCIF monitor was unable to recover.

The emergency program can recover these parameters by using a call RMPAR (refer to the RTE-A programmers manual for details).

When PCIF activates the emergency program it also sends dummy replies to all application programs which have issued a PCIF_OPEN call. These programs will receive a reply status (-17) indicating that contact with PCIF is lost.

WARNING: If the application program is in the process of calling PCIF_OPEN when PCIF stops, it may never receive a reply to this call.

WARNING: Waiting application programs will never receive dummy replies if the PCIF monitor is terminated by the operator or RTE-A.

### 9.3.4 Power Failure

The PCIF monitor will automatically recover from a power failure and subsequent power-up, as long as battery back-up is installed. For Gould-Modicon P/Cs, the AUTOR program provided must be RP'ed from the WELCOME file. If you want to add other power failure recovery processing to AUTOR, you must add to the &AUTOR source provided with PCIF, which calls the routine PCMUX.

### 9.3.5 Run-Time Utilities

When the PCIF monitor displays the message "PCIF>running" other programs have also been activated for handling:

- request time out management (PCTMO);
- sharing PCIF communication information (PCOPN).

If a problem is detected by one of these utility programs, the "PCIF>" message is still displayed but the first two letters of the message text will indicate which utility has detected an error. See Appendix A for details.

## 9.4 STOPPING PCIF

It is possible for the operator to stop PCIF by scheduling the PCIF STOP program PCHLT. This is achieved by typing (on any connected terminal)

RU,PCHLT,<PCIF security code>

The PCIF security code must be supplied and is as provided during the configuration process (screen 11). The call will cause PCIF to send a message to the the original scheduling terminal (i.e. the one on which the "XQ,PCIF" command was made) that may require an answer from the operator. If there are no connected application programs (no program has done a PCIF_OPEN without a corresponding PCIF_CLOSE) then PCIF is immediately stopped and the message "PCIF>stopped" will appear on the original PCIF scheduling terminal.

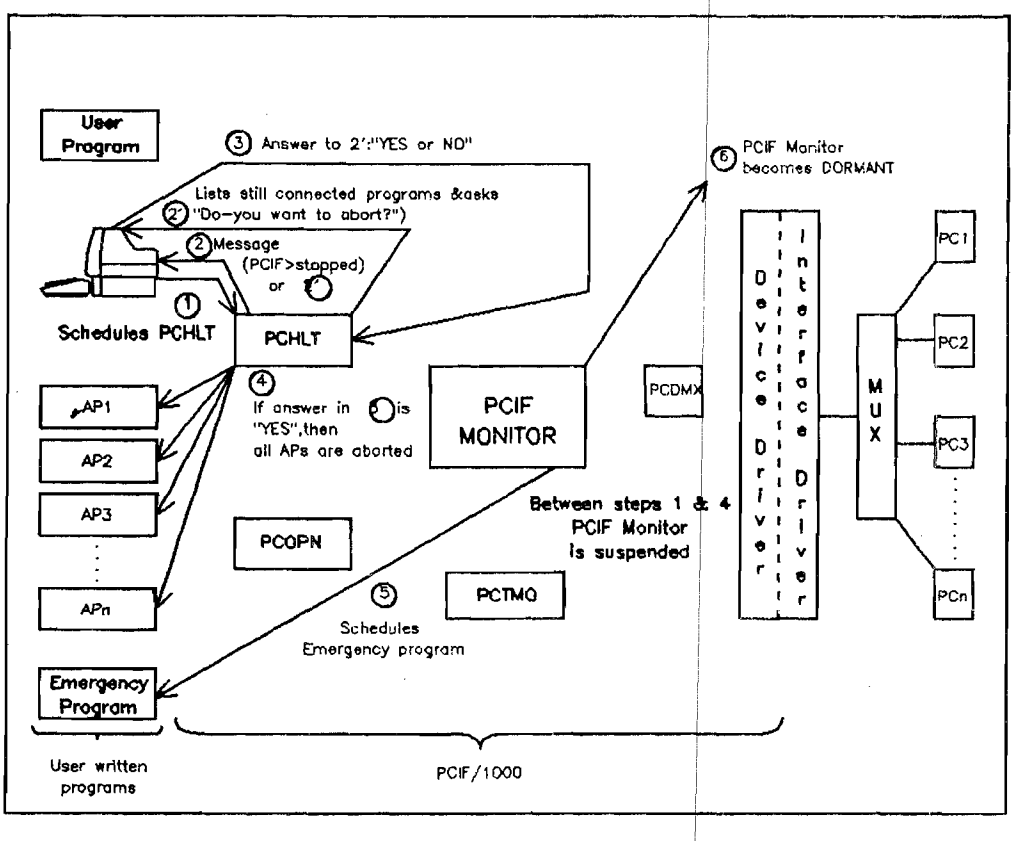



Figure 9.4 Stopping PCIF/1000

The operator will be advised (on the terminal from which PCIF was originally run) if some application programs are still connected. PCIF will wait for confirmation of the stop request, as shown below:

```
PCIF> program currently connected to PCIF: APPLI/1
PCIF> program currently connected to PCIF: USER /78
PCIF> do you want to abort?
```

The operator must answer either yes or no. If the answer is yes then PCIF is stopped, otherwise PCIF will continue to run.

WARNING: PCIF is suspended the moment the PCIF monitor receives a PCHLT call with the correct security code, preventing any interaction with any application program until the request is disabled with the "no" answer. Therefore do not call PCHLT without being sure of the consequences of stopping PCIF.

# Appendix A
# ERROR MESSAGES

This appendix lists all error messages displayed on the scheduling terminal or returned to the application programs by PCIF/1000. The errors are listed under the section of PCIF/1000 which encountered a problem. Sections are provided in the order that a user could encounter problems, starting with configuration; followed by PCIF initialization; continuing with PCIF monitor; and finishing with PCIF utilities.

Error messages appearing on the scheduling terminal may either be warning messages or ABORT messages (see Chapter 9 for a complete explanation of these two cases).

note:

Those errors that cannot be corrected by the user are indicated in this appendix by [*].

# A.1 CONFIGURATION PROCESS ERROR MESSAGES (CO0xx)

These errors are returned by both the preconfigurator and the configuration editor.

CO001 Required field missing - Enter value in field.
CO002 Do you really want to abort? if yes, depress abort again.
CO003 Non descriptor file.
CO004 X required.
CO005 Conflicting answers.
CO007 Non compatible handlers.
CO009 Already existing file.
CO010 Invalid LU number.
CO011 No task selected.
CO013 Undefined soft key.
CO014 Invalid configuration file name.
CO019 Non existing highway number.
CO022 P/Cs still defined with this highway.
CO025 Non existing P/C identifier.
CO026 Invalid priority number.
CO027 Timeout out of range.
CO028 Invalid P/C station number.
CO029 Already allocated P/C station number.
CO030 Non compatible highway.
CO031 Out of range data.
CO034 Corrupted descriptor.
CO035 No associated handler (or FMP error on this file).
CO036 Fmp error number -xxx.
CO037 No associated partial configurator (or FMP error on this file).
CO038 Already entered descriptor file namr.
CO039 Configuration editor linking in process...
CO040 PCIF run-time monitor linking in process...
CO041 Full highway list.
CO042 Full P/C list.
CO043 Invalid port number.
CO044 Already allocated LU. For confirmation, depress the same key again.
CO045 Invalid LU numbers: for this highway type, the LUs must be different.
CO046 Invalid time-out unit: 1 or 2 must be entered.
CO047 Entered capability does not match with the features of this P/C type.
CO048 Non validated configuration. For confirmation, depress the same key.
CO049 Do you really want to replace the configuration file? yes: same key.
CO054 Do you really want to erase the list file ? yes: depress same key again.
CO056 Undefined listing file.
CO057 Can the file #PCLCO be erased ? (if yes, depress ENTER again)
CO058 Can the file &PCCTB be erased ? (if yes, depress ENTER again)
CO059 Can the file #PCLRT be erased ? (if yes, depress ENTER again)
CO060 Can the file &PCRTB be erased ? (if yes, depress ENTER again)
CO061 No associated download file (or FMP error on the download file).
CO062 No entered descriptor file name.
CO063 Already defined P/C on this highway (point-to-point link).
CO064 Only one LU is required for this highway type.
CO065 Data entered does not match field type. Reenter.

C0066 Invalid LSA (the LSAs must be entered in growing order).
C0067 MACRO scheduled, crunch, crunch, crunch ...
C0068 Listing is done. Please continue.
C0069 Unable to access file [PCPGE.
C0070 Unable to schedule MACRO for source file &PCCTB
C0071 Unable to schedule MACRO for source file &PCRTB
C0072 Unable to schedule LINK with command file #PCLCO
C0073 Unable to schedule LINK with command file #PCLRT
C0074 Already existing PCIF monitor or configuration editor on this cartridge.
C0075 Not confirmed answer. Please, continue.
C0076 Writing on list file.
C0077 Can the file %PCCTB be erased ? (if yes, depress ENTER again)
C0078 Can the file %PCRTB be erased ? (if yes, depress ENTER again)
C0079 Writing on configuration file.
C0080 No available types. Abort PCCON. Check preconfiguration.
C0081 Unable to schedule MACRO for source file &PCRTC.
C0082 Can the file &PCRTC be erased ? (if yes, depress ENTER again)
C0083 Can the file %PCRTC be erased " (if yes, depress ENTER again)
C0084 Unable to access file [PCPGF.

## A.2 ERRORS RELATED TO THE DESCRIPTOR FILE (DC0xx)

These errors are returned by both the preconfigurator and the configuration editor.

DC001 Non descriptor file.
DC002 Corrupted descriptor (unable to read next record)
DC003 Corrupted descriptor (invalid description command)
DC004 Corrupted descriptor (file header not in first line)
DC005 Corrupted descriptor (duplicated command)
DC006 Corrupted descriptor (unable to convert ascii string in an integer)
DC007 Corrupted descriptor (invalid handler type declaration)
DC008 Corrupted descriptor (out of range number of types)
DC009 Corrupted descriptor (undefined type)
DC010 Corrupted descriptor (out of range number of comment lines)
DC011 Corrupted descriptor (invalid file name)
DC012 Corrupted descriptor (invalid CAPABILITY command)
DC013 Corrupted descriptor (invalid IO command)
DC014 Corrupted descriptor (out of range number of supported highways types)
DC015 Corrupted descriptor (no defined supported types)
DC016 Corrupted descriptor (no defined compatible highway)
DC017 Corrupted descriptor (undefined handler type)
DC018 Corrupted descriptor (undefined brand)
DC019 Corrupted descriptor (undefined handler)
DC020 Corrupted descriptor (undefined handler entry)
DC021 Corrupted descriptor (undefined partial configurator)
DC022 Corrupted descriptor (undefined partial configurator entry)
DC023 Corrupted descriptor (download file in a P/C descriptor)
DC036 Fmp error number -xxx on descriptor file

## A.3 FORM ERROR MESSAGES (FO0xx)

These errors are returned by F/1000 part of PCIF/1000.

F0001 (Forms error) Non supported operating system
F0002 (Forms error) Terminal already activated
F0003 (Forms error) Unable to read localization file
F0004 (Forms error) Invalid LU
F0005 (Forms error) LU not available
F0006 (Forms error) Driver type not supported
F0007 (Forms error) Terminal type not supported
F0008 (Forms error) Terminal not yet activated
F0009 (Forms error) Form already activated
F0010 (Forms error) Specified file is not a form file
F0011 (Forms error) Internal data buffer overflow
F0012 (Forms error) Field count overflow
F0013 (Forms error) Form not yet activated
F0014 (Forms error) Format error in forms file
F0015 (Forms error) Named field does not exist
F0016 (Forms error) Wrong field data type for call
F0017 (Forms error) Form not yet displayed
F0018 (Forms error) Invalid fields selection flag
F0019 (Forms error) Invalid read selection value
F0020 (Forms error) Invalid edit actions values
F0021 (Forms error) Warning: string truncated
F0022 (Forms error) Field type is not unprotected
F0023 (Forms error) Invalid string length
F0024 (Forms error) Invalid action code
F0025 (Forms error) Read error or terminal malfunction
F0026 (Forms error) Conversion overflow
F0027 (Forms error) Invalid initial value
F0028 (Forms error) Warning: sfk attributes too long
F0029 (Forms error) Non displayable value supplied
F0030 (Forms error) Missing or invalid block data
F0031 (Forms error) Help file not a type 4 file
F0100 (Forms error) Not enough class numbers
F0101 (Forms error) Duplicate monitor clone name
F0102 (Forms error) No ID segments available
F0103 (Forms error) Monitor program not found
F0104 (Forms error) FMP open error on monitor file
F0105 (Forms error) FMP close error on monitor file
F0106 (Forms error) FMP RP error on monitor file
F0107 (Forms error) Monitor program busy
F0109 (Forms error) Monitor program schedule error
F0112 (Forms error) PTOP error: main to clone
F0113 (Forms error) PTOP error: clone to main
F0200 RTE-A I/O error *  (check the RTE-A manual or driver manual)

## A.4 SCREEN ERROR MESSAGES (FR0xx)

These errors are returned by the F/1000 part of PCIF/1000.

FR001 REQUIRED FIELD(S) MISSING. PLEASE ENTER MISSING VALUE(S).
FR002 DATA ENTERED DOES NOT MATCH FIELD TYPE. PLEASE REENTER VALUE.
FR003 DATA ENTERED IS OUTSIDE ACCEPTED RANGE. PLEASE REENTER VALUE.

## A.5 PARTIAL FMP ERROR CODES (FM0xx)

FM001 DISC ERROR : disc down (FMP -1)
FM002 Duplicate file name (FMP -2)
FM005 illegal record length (FMP -5)
FM006 FILE not found (FMP -6)
FM007 bad security code (FMP -76)
FM008 file locked or open (FMP -8)
FM012 EOF or SOF error on file (FMP -12)
FM013 locked cartridge (FMP -13)
FM014 directory full (FMP -14)
FM015 illegal file name (FMP -15)
FM032 cartridge not found (FMP -32)
FM033 not enough room on disc cartridge (-33)
FM036 lock error on device (FMP -36)
FM103 disc directory corrupt (FMP -103)
FM104 extent not found (FMP -104)

## A.6 PCIF INITIALIZATION ERRORS (MI0xx)

note:

Those errors that cannot be corrected by the user are indicated in this appendix by [*].

MI000 monitor not started
[*] MI001 error in dereferencing pointer line # xxx
[*] MI002 undefined CASE (PASCAL) in line # xxxx
MI003 not enough EMA space for this configuration
*EMA size for keeping PCIF monitor information and user's buffers is not big enough. Change this size using RTE-A command : SZ as described in the RTE-A help file.*

[*] MI004 NIL pointer (PASCAL) dereferenced line # xxx
[*] MI005 PASCAL value out of range line # xxx
[*] MI006 PASCAL IO error (error #/line #) xxx yyy
[*] MI007 PASCAL FMP error (error #/line #) xxx yyy
[*] MI008 PASCAL warning error (error #/line #) xxx yyy
[*] MI009 PASCAL undefined error (error #/ line #) xxx yyy
[*] MI010 error in loading monitor segment
MI011 missing configuration file in RUN string
*Restart PCIF by providing the name of a file in the RUN string (first parameter after PCIF).*

MI012 no ID segment exists for the emergency program
*Do RP of the emergency program, then restart PCIF.*

MI013 invalid configuration NAMR
*Retype run string to start PCIF by providing correct file name.*

MI014 too many parameters in run string
MI015 missing information messages (INxxx type)
*Messages of type IN001 to IN011 are not in the error message file. PCIF cannot start if any of this information is missing. Another reason can be the fact that the error message file has not been found.*

[*] MI016 associated handler generates invalid ACTION
[*] MI017 handler context too big
MI019 too many highways are used in this configuration
*Correct the configuration file.*

MI020 configuration file not validated
*Do a listing of the current configuration file (using PCIF Configurator) to find "undefined" statements in PC's or Highway declaration. Define the undefined values, then restart PCIF monitor.*

MI021 file is not a configuration file
MI022 too many MUX or I/O cards on this configuration

MK033 impossible to send reply to PCIF_OPEN
*The user program that requested PCIF_OPEN was removed from the system before the PCIF monitor could reply to this PCIF_OPEN request.*

MK034 problems in sending I/O messages
*See if the system has the correct device driver for the corresponding I/O channel.*

MK035 RTE-A refused to process I/O messages (class I/O)
[*] MK036 invalid I/O message received by PCIF
[*] MK037 MUX message received out of sequence
[*] MK038 monitor class number disappeared
MK039 another program is doing class/get on PCIF monitor class number
MK040 RTE-A class I/O problems
MK041 monitor was aborted by operator
MK042 impossible to answer to application program
*A user program was removed from the system (stopped, aborted) while some requests were still in process inside the PCIF monitor for this user, or this user has made a PC_ENUNSOL and the program stopped without PCIF_CLOSE and the corresponding P/C has sent an unsolicited request. In all these cases all requests and operations (e.g. PC_LOCK, PC_ENUNSOL) made by this user are cleared from the system.*

[*] MK043 wrong action_code emitted by handler
[*] MK044 requested buffer length too big
MK045 emergency program not callable
*When PCIF monitor stopped (for any reason), it tried to schedule the emergency program whose ID name was given in run the string, but PCIF was unable to activate it (no ID segment, program active, ...).lllllllllllll*

MK046 wrong stop password
*PCIF monitor received a message from PCHLT but the password was not equal to the password defined at configuration time.*

MK047 temporary EMA shortage for request buffers
*PCIF monitor was unable to process a request due to insufficient EMA space. If MK047 occurs too frequently or if the system goes into some deadlock (message MK048 does not arrive) then change EMA size of PCIF monitor.*

MK048 EMA shortage RECOVERED
*There is now sufficient space for processing requests inside PCIF monitor.*

[*] MK049 error in handler's call to FLUSH_LU
MK050 two OPEN's for same program, oldest flushed : xxxxx
*If an application program tries to do a PCIF_OPEN and there is already another program connected with the same name (and same session number if a system is used in session mode), then this means that the previous program was removed without doing a PCIF_CLOSE call.*

MK051 MUX power fail recovered

## A.8 PCDMX ERRORS (UTILITY TO DOWNLOAD CARDS) (DM0xx)

note:

Those errors that cannot be corrected by the user are indicated in this appendix by [*].

DM002 not a downloadable card on LU
*Check RTE-A system generation.*

DM003 FMP error on download file.
*An extra message will follow (FMxxx) giving the exact meaning of the most common FMP error codes (otherwise see RTE-A FMP manual).*

DM004 invalid download file
[*] DM005 invalid DMX function code
DM006 card not in download mode on LU
DM007 self test error on card LU
DM009 error during download on card LU xxx
DM010 card status not OK after download on LU xxx
DM011 card does not answer on LU xxx
DM012 no download file name
DM013 invalid download file name xxxxxx
DM014 RTE-A exec call error = xxx
*Something went wrong during an RTE-A EXEC call, the message indicates the RTE-A code (like SC05, CL02...).*

DM015 MUX card not ready on LU xxx
DM016 MUX card dialog into time out on LU xxx
DM017 IDS00 : transmission error on LU xxx
DM018 write protected LU xxx
DM019 IDS00 : datacommunication error on LU xxx
DM020 IDS00 : undefined error on LU xxx
DM021 illegal length of record on file for LU xxx
DM022 incorrect checksum on file for LU xxx
DM023 Z80 address out of range for downloaded code on LU xxx
DM024 incorrect download file length specified within record on LU xxx
DM025 unrecognized download file record format for LU xxx
DM026 no entry address specified in download file for LU xxx
DM027 IDS00 : multiple errors during download on LU xxx
DM028 non existing LU xxx
DM029 impossible to answer to PCIF monitor
*PCDMX program was asked by PCIF monitor to download a MUX card. The job finished but it was not possible to send completion information to monitor.*

## A.9 PCTMO ERRORS (TIME OUT UTILITY) (TM0xx)

TM002 impossible to communicate with PCIF monitor
*PCTMO (time out manager) was unable to communicate with PCIF monitor (aborted, infinite loop,...).*

TM003 RTE-A error
TM004 PCTMO programming (PASCAL) error

## A.10 PCHLT ERRORS (HALT UTILITY PROGRAM) (HT0xx)

HT001 PCOPN program not found
HT002 invalid PCOPN program
HT003 PCIF subsystem not started
HT004 impossible to communicate with PCIF monitor

# Appendix B
# PCTST LISTING

This appendix contains a description and listing of PCTST, a program to assist in testing, verification, program debugging and learning PCIF/1000.

## B.1 USING PCTST

PCTST is an exerciser program which allows the user to execute PCIF subroutine calls in an interactive fashion. For those familiar with the HP 2250, PCTST is to PCIF much like MCX is to MCL/50.

To run PCTST simply input:

PCTST,1,1

from CI, CM, or FMGR. PCTST will then respond as follows:

PCIF/1000 exercizer 94200-16404 REV.2XXX <YYMMDD.HHMM>
(The current day and time)

pctest >

PCIF itself must be running before anything may be done with PCTST. PCIF itself is started by entering XQ PCIF,namr from CI, CM, or FMGR (NOT from PCTST) where namr is the namr of the PCIF configuration file.

## B.2 PCTST DIALOG

PCTST supports all the standard PCIF subroutine calls. In addition to this there are several "special" commands used by the PCTST program for things such as buffer management. These "special" calls will be discussed in section B.5. The standard PCIF calls are executed by entering the name of the routine, excluding the prefix of PC__ or PCIF__.

For example, to execute the routine PCIF__OPEN, the PCTST user simply inputs OPEN, omitting the PCIF__ prefix. PCTST will then prompt the user for any additional input parameters required.

The following is an example dialog in which the operator first opens communication with PCIF, logically connects the PC of interest, and then enquires upon this PC it's status. Operator inputs are indicated with an underline.

```
CI> PCTST

PCIF/1000 exercizer 94200-16301 REV.2340 <831128.1020>
9:55 AM FRI., 21 OCT., 1983

pctest >  open

pctest >  connect
 pc     ? 1

pctest >  pcstat
 pc     ? 1
-> PC run mode=ON; download accept=ON
```

```
pctest >  ex
-> end of PCTEST program

CI>
```

Commands may be entered in either lower or upper case. The parameters for a call may also be entered all at once. For example, when the operator input connect in the above example, PCTST prompted him for the pc to connect. To eliminate the prompting, it is possible to input connect 1 and execute the entire command without the subsequent dialog.

For example, the dialog:

```
pctest >  readd
 pc     ? 1
 length ? 10
 pcadr  ? 32
```

is equivalent to:

```
pctest >  readd 1 10 32
```

When a command is completed normally, the "pctest >" prompt appears, if an error occurs an error message will be displayed. For example, if PCIF was not running when a open request is made to PCTST, the following will be displayed:

```
-> error  : PC017 contact with PCIF monitor lost
```

## B.3 PCTST PCIF COMMAND SUMMARY

The following table summarizes each of the PCIF access routines and how they are accessed from PCTST. Detailed explanations of each routine may be found in chapter 4 of the PCIF manual. The routines are listed in the table in the same order in which they appear in the manual. Following the table is a detailed description of each command.

The table is divided into three columns. The first column contains the name of the PCIF access routine as it would be used in a "Call" statement of an application program. The second column indicates the equivalent PCTST command. The third column briefly describes the function of the call.

| PCIF Routine | PCTST Command | Comments |
|---|---|---|
| PCIF_OPEN | Open | Opens communication with PCIF |
| PCIF_CLOSE | Close | Closes communication with PCIF |
| PCIF_ERROR | Error | Displays the error message for the given _# |

| PCIF Routine | PCTST Command | Comments |
|---|---|---|
| PC__CANCEL | Cancel | Flushes previous requests |
| PC__CONNECT | Connect | Logically connects a PC |
| PC__DISC | Disc | Logically disconnects a PC |
| PC__DIUNSOL | Diunsol | Disables unsolicited requests |
| PC__ENQUIRY | Enquiry | Enquires the status of a call made w/o wait |
| PC__ENUNSOL | Enunsol | Enables unsolicited requests |
| PC__GETKEY | Getkey | Gets a key (class no.) for calls made w/wait |
| PC__IDENT | Ident | Identifies the target PC |
| PC__LOCK | Lock | Locks a PC for exclusive use |
| PC__PCSTAT | Pcstat | Gets status of a PC |
| PC__READD | Readd | Reads PC data |
| PC__READP | Readp | Reads PC programs |
| PC__RELKEY | Relkey | Releases a key (class no.) |
| PC__START | Start | Starts a PC program |
| PC__STOP | Stop | Stops a PC program |
| PC__SYSTAT | Systat | Returns the logical status of a PC |
| PC__TRANS | Trans | Allows transparent communication to a PC |
| PC__UNLOCK | Unlock | Unlocks a PC from exclusive use |
| PC__WRITED | Writed | Writes data to a PC |

| | | |
|---|---|---|
| PC__WRITEP | Writep | Writes a program to a PC |

## B.4 PCTST PCIF COMMANDS

The following pages describe, in detail, each of the PCIF commands as used in PCTST. Included for each command are the name of the PCIF command, the name of the equivalent PCTST command, the required entry parameters, and an explanation of the data, if any, returned. Entry parameters indicated with a "*" are only entered when operationg in asynchronous (I/O without wait) mode. (See the mode command in section B.5.)

* indicates inputs required only in asynchronous mode

PCIF OPEN

PCTST Command: Open

Function: Logically opens communication between the application program (in this case PCTST) and PCIF.

Entry Parameters:

None
Returned Information:

None

PCIF CLOSE

PCTST Command: Close

Function: Logically closes communication with the application program (in this case, PCTST).

Entry Parameters:

None

Returned Information:

None

PCIF ERROR

PCTST Command: Error

Function: Returns an ascii error message for a specified error number.

Entry Parameters: error# ?

Entry Parameter Descriptions:

error# - Enter the PCIF error number in integer format.

Returned Information:

An ascii string defining the particular error number.

## PC CANCEL

PCTST Command: Cancel

Function: To cancel all pending PC requests, or a specified PC request.

Entry Parameters:

```
pc     ?
tag    ?
typec  ?
```

Entry Parameter Descriptions:

pc - Enter the logical PCIF address of the PC to which the command is to be addressed.

tag - Enter the tag of the request to be canceled. This only is used when typec is non-zero.

typec - Enter a zero to cancel all waiting PC requests, enter a one to cancel only the request with the specified tag.

Returned Information:

None

## PC CONNECT

PCTST Command: Connect

Function: Logically connects the specified PC to the PCIF application program (PCTST in this case).

Entry Parameters:

```
  pc     ?
* tag    ?
* key    ?
```

Entry Parameter Descriptions:

pc - Enter the logical PCIF address of the PC to which the command is to be addressed.

tag - Enter any integer value to identify this request. This value is only used in asynchronous mode.

key - Enter a valid access key (obtained with a Getkey). This value is only used in asynchronous mode.

Returned Information:

None

## PC DISC

PCTST Command: Disc

Function: Logically disconnects the specified PC from all application programs.

Entry Parameters: pc ?
passwd ?

Entry Parameter Descriptions:

pc - Enter the logical PCIF address of the PC to which the command is to be addressed.

passwd - This must be the PCIF security code entered at configuration time. It is an integer number.

Returned Information:

None

## PC DIUNSOL

PCTST Command: Diunsol

Function: Disables all unsolicited requests from the specified PC to the application program

Entry Parameters: pc ?

Entry Parameter Descriptions:

pc - Enter the logical PCIF address of the PC to which the command is to be addressed.

Returned Information:

None

## PC ENQUIRY

PCTST Command: Enquiry or Enquiryw

Function: Retrieves replys to previous requests made in asynchronous mode (without wait).

Enquiry checks for a ready reply and finishes if not reply is ready.

Enquiryw will wait until a reply is received if no reply is currently available.

Entry Parameters: key ?

Entry Parameter Descriptions:

key - Enter the key for the request which you wish to enquire upon.

Returned Information:

A reply like the one shown below will be received when an enquiry is made:

```
message get for PC   1 reply to reply to READD typer = 1
               tag 123  logr = 20
```

This reply indicates that for the specified access key, a reply is indeed ready. The type of request that was made was a READD (read data), this is also indicated by the typer (type of request field). The following values are used in typer to indicate the type of request for which information is being returned:

| Typer | Request Type | Typer | Request type |
|---|---|---|---|
| 1 | PC_READD | 17 | PC_STOP |
| 2 | PC_WRITED | 18 | PC_ENUNSOL |
| 3 | PC_READP | 19 | PC_DIUNSOL |
| 4 | PC_WRITEP | 20 | PC_CONNECT |
| 8 | PC_TRANS | 22 | PC_STAT |
| 16 | PC_START | 30 | PC_IDENT |

Also indicated is the tag of the returned request, and the length of returned data. The length of returned data is indicated in bytes in the logr field. The PC_ENQUIRY call made in PCTST uses a length of 256 words. To read longer return buffers, you must change the code in PCTST.

Note that an enquiry may also be made to an access key which is being used for unsolicited requests. If this were the case the message displayed by PCTST would look something like this:

```
message get for PC   1 reply to unsolicited typer = 0
              tag    0  logr =  0
```

If no replies are ready for the indicated access key a message to that effect will be displayed by PCTST:

```
no message currently in access key
To wait on enquiry use function : ENQUIRYW
```

## PC ENUNSOL

PCTST Command: Enunsol

Function: Enables a specified PC to generate unsolicited interrupts to a specified access key.

Entry Parameters:
pc ?
key ?

Entry Parameter Descriptions:

pc - Enter the logical PCIF address of the PC to which the command is to be addressed.

key - Enter a valid access key (obtained with a Getkey). This value will be used in an Enquiry or an Enquiryw to determine if an unsolicited request has occurred.

Returned Information:

None

## PC GETKEY

PCTST Command: Getkey

Function: Obtains an access key (like a RTE class number) to be used in PCIF I/O without wait.

Entry Parameters: None

Returned Information:

A key to be used in subsequent asynchronous PCIF I/O requests.

## PC IDENT

PCTST command: Ident

Function: Identifies the vendor, model number, and station ID of the target PC.

Entry Parameters:
```
    pc    ?
  * tag   ?
  * key   ?
```

Entry Parameter Descriptions:

pc - Enter the logical PCIF address of the PC to which the identity enquiry is to be made.

tag - Enter any integer value to identify this requist. This value is only used in asynchronous mode.

key - Enter a valid access key (obtained with a Getkey.) This value is only used in asynchronous mode.

lengr - Enter the length of the return buffer. This value is always ten words.

Returned Information:

A ten-word buffer identifying the vendor, model number, and station ID of the specified PC.

## PC LOCK

PCTST Command: Lock

Function: Locks the specified PC for exclusive use of the calling program, in this case PCTST.

Entry Parameters: pc ?

Entry Parameter Descriptions:

pc - Enter the logical PCIF address of the PC which is to be locked.

Returned Information:

None

## PC PCSTAT

PCTST Command: Pcstat

Function: Enquires upon a specified PC its status.

Entry Parameters:
```
    pc    ?
  * tag   ?
  * key   ?
```

Entry Parameter Descriptions:

pc - Enter the logical PCIF address of the PC to which the status enquiry is to be made.

tag - Enter any integer value to identify this request. This value is only used in asynchronous mode.

key - Enter a valid access key (obtained with a Getkey). This value is only used in asynchronous mode.

Returned Information:

PCTST will return a message indicating WHETHER the specified PC has run mode on or off, and WHETHER it has download accept on or off:

-> PC run mode=ON/OFF; download accept=ON/OFF

## PC READD

PCTST Command: Readd

Function: Reads data from the specified PC address for a specified length.

Entry Parameters:
```
pc     ?
length ?
pcadr  ?
```

```
* tag    ?
* key    ?
```

Entry Parameter Descriptions:

pc - Enter the logical PCIF address of the PC to which the command is to be addressed.

length - Enter the length of data in 16 bit words.

NOTE: the length may also be specified in bits or bytes, see Chapter 4 of the PCIF manual for details.

pcadr - Enter the physical PC address from which the data is to be read.

tag - Enter any integer value to identify this request. This value is only used in asynchronous mode.

key - Enter a valid access key (obtained with a Getkey). This value is only used in asynchronous mode.

Returned Information:

The specified data is returned to an internal buffer in PCTST (note that this buffer is limited to 256 words). This buffer can be read with the BUFRD command which is described in section B.5.

PC READP

PCTST Command: Readp

Function: Allows the application program (PCTST in this case) to upload a PC program into a buffer.

Entry Parameters:

```
  pc     ?
  length ?
  pcadr  ?
* tag    ?
* key    ?
```

Entry Parameter Descriptions:

pc - Enter the logical PCIF address of the PC to which the command is to be addressed.

length - Enter the length of data in 16 bit words.

NOTE: the length may also be specified in bits or bytes, see Chapter 4 of the PCIF manual for details.

pcadr - Enter the physical PC address of the program to be read.

tag - Enter any integer value to identify this request. This value is only used in asynchronous mode.

key - Enter a valid access key (obtained with a Getkey). This value is only used in asynchronous mode.

Returned Information:

The returned program information is stored in a buffer internal to the PCTST program. This buffer may be displayed with the BUFRD command which is discussed in section B.5. Note that the buffer is limited to 256 words.

## PC RELKEY

PCTST Command: Relkey

Function: Releases an access key back to the system.

Entry Parameters: key ?

Entry Parameter Descriptions:

key - Enter the number of the key to be released.

Returned Information:

None

## PC START

PCTST Command: Start

Function: Physically starts the specified PC. Note that not all PCs have this capability, Allen-Bradley is an example.

Entry Parameters:
```
    pc    ?
  * tag   ?
  * key   ?
```

Entry Parameter Descriptions:

pc - Enter the logical PCIF address of the PC which is to be started.

tag - Enter any integer value to identify this request. This value is only used in asynchronous mode.

key - Enter a valid access key (obtained with a Getkey). This value is only used in asynchronous mode.

Returned Information:

None

PC STOP

PCTST Command: Stop

Function: Physically stops the specified PC.

Entry Parameters:
```
    pc    ?
  * tag   ?
  * key   ?
```

Entry Parameter Descriptions:

pc - Enter the logical PCIF address of the PC which is to be stopped.

tag - Enter any integer value to identify this request. This value is only used in asynchronous mode.

key - Enter a valid access key (obtained with a Getkey). This value is only used in asynchronous mode.

Returned Information:

None

PC SYSTAT

PCTST Command: Systat

Function: Enquires the logical state of a specified PC.

Entry Parameters: pc ?

Entry Parameter Descriptions:

pc - Enter the logical PCIF address of the PC to which the command is to be addressed.

Returned Information:

PCTST will return a small table of information describing the current logical state of the specified PC. Information dispalyed includes WHETHER a PC has unsolicited requests enabled, WHETHER or not it is logically connected, WHETHER it is locked or unlocked, WHETHER or not it has any requests pending, and what capabilities the PC has. An example of this table is shown below:

Example Systat table:

```
-> unsolicited DISABLEd
   CONNECTED
   UNlocked
   NO rqst pending
   capability : WriteData  WriteProgram
                TRansparent  Unsolicited
```

PC_TRANS

PCTST Command: Trans

Function: This command allows transparent communication with a PC. The actual buffer to be written must first be created with the BUFWR command (see section B.5 for details on BUFWR).

********** WARNING **********

The use of PC_TRANS requires from the user a good knowledge of the particular PC addressed. PC_TRANS will not preserve PCIF/1000 transparency and may change the physical status of the real PC, making it difficult or impossible to process further normal PCIF/1000 functions.

Entry Parameters:

```
    pc     ?
    length ?
    logr   ?
    subfnc ?
  * tag    ?
  * key    ?
```

Entry Parameter Descriptions:

pc - Enter the logical PCIF address of the PC to which the command is to be addressed.

length - Enter the length of data in 16 bit words.

NOTE: the length may also be specified in bits or bytes, see Chapter 4 of the PCIF manual for details.

logr - Enter the maximum message length to be returned from the PC in words. Like the length field, logr can also be expressed in bits or bytes, see Chapter 4 of the PCIF manual for details.

subfnc - Enter the PC subfunction to be performed. This subfunction is an integer code which is PC manufacturer and model dependent.

tag - Enter any integer value to identify this request. This value is only used in asynchronous mode.

key - Enter a valid access key (obtained with a Getkey). This value is only used in asynchronous mode.

Returned Information:

A buffer is returned to the PCTST which may be displayed with the BUFRD command.

## PC UNLOCK

PCTST Command: Unlock

Function: Unlocks a PC from exclusive use by an application program, in this case PCTST.

Entry Parameters: pc ?

Entry Parameter Descriptions:

pc - Enter the logical PCIF address of the PC which is to be unlocked.

Returned Information:

None

## PC WRITED

PCTST Command: Writed

Function: Writes data to the specified address of a specified PC. The buffer to be written is created with the PCTST BUFWR buffer write command (see section B.5).

Entry Parameters:

```
  pc     ?
  length ?
  pcadr  ?
* tag    ?
* key    ?
```

Entry Parameter Descriptions:

pc - Enter the logical PCIF address of the PC to which the command is to be addressed.

length - Enter the length of data in 16 bit words.

NOTE: the length may also be specified in bits or bytes, see Chapter 4 of the PCIF manual for details.

pcadr - Enter the physical PC address to which the data is to be written

tag - Enter any integer value to identify this request. This value is only used in asynchronous mode.

key - Enter a valid access key (obtained with a Getkey). This value is only used in asynchronous mode.

Returned Information:
None

## PC WRITEP

PCTST Command: Writep

Function: Downloads a PC program from an application program (PCTST in this case) to the specified address of the specified PC. The program must first be stored in an internal PCTST buffer with the BUFWR command (see section B.5).

Entry Parameters:

```
  pc     ?
  length ?
```

```
  pcadr  ?
* tag    ?
* key    ?
```

Entry Parameter Descriptions:

pc - Enter the logical PCIF address of the PC to which the command is to be addressed.

length - Enter the length of data in 16 bit words.

NOTE: the length may also be specified in bits or bytes, see Chapter 4 of the PCIF manual for details.

pcadr - Enter the physical PC address to which the program is to be written.

tag - Enter any integer value to identify this request. This value is only used in asynchronous mode.

key - Enter a valid access key (obtained with a Getkey). This value is only used in asynchronous mode.

Returned Information:

None

## B.5 PCTST SPECIAL COMMANDS

In addition to the previously defined commands which are based on actual PCIF commands, there are several "special" PCTST commands used for various things. These commands are summarized in the table below and described in detail in the following sections.

| PCTST Command | Comments |
|---|---|
| Bufrd | Displays the internal PCTST buffer. This buffer is filled by, for example, a Readd. |
| Bufwr | Creates an internal PCTST buffer for use with subsequent commands such as Writed. |
| Dwload | Downloads a PC program from a file to a PC. |
| Exit or Ex | Exits PCTST. |
| Mode | Changes the command mode from synchronous (with wait) to asynchronous (without wait) or vice-versa |

Upload — Uploads a PC program from a PC to a file.

BUFRD

PCTST Command: Bufrd

Function: Displays an internal PCTST buffer. This buffer is the result of some sort of PCIF I/O, e.g. a Readd.

Entry Parameters: None

Returned Information:

The current value of the internal buffer is displayed. For example, if a Bufrd command were executed after a Readd request asking for ten words of data, a display like the following would result:

```
0 ->        5
1 ->      100
2 ->        0
3 ->    12538
4 ->       11
5 ->     -576
6 ->      465
7 ->        1
8 ->     1000
9 ->        0
```

Note that the values are displayed in integer format.

BUFWR

PCTST Command: Bufwr

Function: Creates an internal buffer with values input by the operator for use with subsequent PCIF output operations.

Entry Parameters: Array values (see example below).

Entry Parameter Descriptions:

The following is an example dialog in which a three word buffer is created with

a 10 in the first word, a 20 in the second word, and a 30 in the final word:

```
pctest > bufwr
enter values as INTEGER, enter '*' or 'a' to stop
    0   ? 10
    1   ? 20
    2   ? 30
    3   ? a

pctest >
```

Note that values are entered in integer and that an 'a' or an '*' terminates input. Once this buffer is created, it can be used in a subsequent output operation such as a Writed.

Returned Information:

None


## DWLOAD

PCTST Command: Dwload

Function: Downloads a PC program from a file to a PC.

Entry Parameters:

```
    pc      ?
    pcadr   ?
*   tag     ?
*   key     ?
    file    ?
```

Entry Parameter Descriptions:

pc - Enter the logical PCIF address of the PC to which the program is to be downloaded.

pcadr - Enter the physical PC address to which the program is to be written.

tag - Enter any integer value to identify this request. This value is only used in asynchronous mode.

key - Enter a valid access key (obtained with a Getkey). This value is only used in asynchronous mode.

file - Enter the namr of the file containing the PC program.

Returned Information:

None

MODE

PCTST Command: Mode

Function: Changes the PCIF I/O mode from synchronous (with wait) to asynchronous (without wait) or vice-versa.

Entry Parameters: Verification of change (Yes or No)

Entry Parameter Descriptions:

When 'mode' is input, PCTST will display the current mode and ask you if you want to change the mode as follows:

```
current mode is synchronous
to change mode type "YES". All other input will not do any change
YES/NO ?
```

Returned Information:

None

UPLOAD

PCTST Command: Upload

Function: Uploads a PC program from a PC to a file.

Entry Parameters:

```
  pc     ?
  pcadr  ?
  length ?
* tag    ?
* key    ?
  file   ?
```

Entry Parameter Descriptions:

pc - Enter the logical PCIF address of the PC from which the program is to be uploaded.

pcadr - Enter the physical PC address from which the program is to be read.

length - Enter the length of data in 16 bit words.

NOTE: the length may also be specified in bits or bytes, see Chapter 4 of the PCIF manual for details.

tag - Enter any integer value to identify this request. This value is only used in asynchronous mode.

key - Enter a valid access key (obtained with a Getkey). This value is only used in asynchronous mode.

file - Enter the namr of the file into which the PC program is to be written.

Returned Information:

None

## B.6 PCTST INSTALLATION

Installing PCTST is quite simple. First of all, however, PCIF/1000 itself must be installed. Consult Chapters 5 thru 7 of the PCIF manual for details on this.

Next, install the PCTST files on your system. They may be put on any cartridge or volume. The files needed are listed in the table below.

PCTST Files

| File Name | Type | Comments |
|---|---|---|
| #PCTST | 4 | Link command file for linking PCTST |
| &PCTST | 4 | PCTST source code |
| %PCTST | 5 | PCTST relocatable file |
| PCTST | 6 | PCTST program |

In addition to these files the following libraries are required for linking PCTST:

$PCLIB PASCAL.LIB

After the above files are in place, try running the type 6 file, PCTST. If this doesn't work relink PCTST by entering from CI:

```
CI> link,#pctst::<FMGR crn of PCIF software>
```

This will run LINK with the command file provided with PCIF. Note that you may need to edit #PCTST to reflect the directories of your particular system.

## B.7 PCTST SOURCE CODE

```
$PASCAL '94200-16404 REV.2525 <850609.2200>'
{                                                                    }
{     NAME  : PCTST                                                  }
{     SOURCE: 94200-18404                                            }
{     RELOC.: 94200-16404                                            }
{     COMMAND FILE : 94200-17404                                     }
{     PRGMR : SGR                                                    }
{                                                                    }
{    Modified           Added PC_IDENT command processing.           }
{    5-8-85                                                         }
{    by CGY             Added run_string 0 option, so it can be     }
{      (rev.2525)         used serial reusable in a memory based    }
{                         system.                                    }
{                                                                    }
{ ****************************************************************** }
{ * (C) COPYRIGHT HEWLETT-PACKARD COMPANY 1984. ALL RIGHTS * }
{ * RESERVED. NO PART OF THIS PROGRAM MAY BE PHOTOCOPIED * }
{ * REPRODUCED OR TRANSLATED TO ANOTHER PROGRAM LANGUAGE WITHOUT * }
{ * THE PRIOR WRITTEN CONSENT OF HEWLETT-PACKARD COMPANY * }
{ ****************************************************************** }
{                                                                    }
$ RECURSIVE OFF $
$ RUN_STRING 0  $
$ TITLE 'PCIF/1000 exerciser' $


PROGRAM PC_test (input, output) ;



$PAGE
TYPE
   WORD = -32768..32767 ;
   BYTE = -128..127     ;
   BIT_TYPE = 0..1;
    idcb_type               =ARRAY[1..144] OF WORD ;
    ibuf_type               =ARRAY[1..128] OF WORD ;
    name_type               =PACKED ARRAY[1..6] OF CHAR    ;
    isize_type              =ARRAY[1..2] OF WORD   ;
    isecu_icr_type          =WORD ;
   BUF_TYPE = ARRAY [1..512] OF WORD ;
   STRING80 = PACKED ARRAY [1..80] OF CHAR ;
   STRING20 = PACKED ARRAY [1..20] OF CHAR ;
   STRING6 = PACKED ARRAY [1..6] OF CHAR ;
   STRING2 = PACKED ARRAY [1..2] OF CHAR ;
```

```
   STRING3 = PACKED ARRAY [1..3] OF CHAR ;
   STRING9 = PACKED ARRAY [1..9] OF CHAR ;
   STRING11 = PACKED ARRAY [1..11] OF CHAR ;

   EQ_WRD_BYTE = PACKED RECORD
                   CASE INTEGER OF   {*CGY* Added bit_array}
                     1: (wrd: word);
                     2: (bit_array : PACKED ARRAY [1..16] OF BIT_TYPE);
                     3: (bt1:  byte;
                         bt2:  byte);
                 END;

 {*CGY* For printing bufrd with bit lengths.}
   BIT_LENGTH_TYPE  = PACKED RECORD
                        CASE INTEGER OF
                          1: (wd: word);
                          2: (hi_bit : 0..1;
                              lo_bits : 0..32767);
                       END;
VAR
   tempo:         eq_wrd_byte;
   nb_displayed:    WORD;
   zero:WORD;
   abup,abdown:FILE OF WORD;
   key,oldtag,typec,typer,contwd,tag,oldstat:WORD;
   func: STRING20;
   length,
   stat,pc,buffr,lengt,subfnc:WORD;
   bit_length : BIT_LENGTH_TYPE;          {*CGY* Added bit_length}
   pcadr:INTEGER;
   logr : WORD ;
   password : WORD ;
   lengr,i,j:WORD;
   error_buffer : STRING80 ;
   error_length : WORD ;
   eof_flag, found_flag, command_flag : BOOLEAN ;
   command_string : PACKED ARRAY [1..80] OF CHAR ;
   command_ptr : WORD ;
   command_index : WORD ;
   indexl : WORD ;
   number : WORD ;
   idcb:idcb_type ;
   ibuf:ibuf_type ;
   file_name:name_type ;
   file_namr : STRING20 ;
   len,ierr,c,number_of_cut,cut_number,il: WORD ;
   isize: isize_type      ;
   icr,isecu:isecu_icr_type ;
   buff1 : BUF_TYPE ;
   ask_bufval : STRING11 ;
   stop_flag : BOOLEAN ;
   asynch_flag : BOOLEAN ;
   echo_flag : BOOLEAN ;
```

```
CONST
   ask_command = STRING11 [' pctest >  '] ;
   ask_pc      = STRING11 ['  pc     ? '] ;
   ask_lengt   = STRING11 ['  length ? '] ;
   ask_logr    = STRING11 ['  logr   ? '] ;
   ask_pcadr   = STRING11 ['  pcadr  ? '] ;
   ask_file    = STRING11 ['  file   ? '] ;
   ask_subfnc  = STRING11 ['  subfnc ? '] ;
   ask_passwd  = STRING11 ['  passwd ? '] ;
   ask_tag     = STRING11 ['  tag    ? '] ;
   ask_type_ca = STRING11 ['  typec  ? '] ;
   ask_key     = STRING11 ['  key    ? '] ;
   ask_yes     = STRING11 ['  YES/NO ? '] ;
   ask_error   = STRING11 ['  error# ? '] ;
$PAGE
{-------------------------------------------------}
{                                                 }
{   PCIF/1000 LIBRARY : PROCEDURE CALL DEFINITIONS }
{                                                 }
{-------------------------------------------------}

PROCEDURE PC_CALL
   (VAR stat   : WORD ;
        tag    : WORD ;
        contwd : WORD ;
        pc     : WORD ;
        subfnc : WORD ;
    VAR buffr  : BUF_TYPE ;
        lengt  : WORD ;
        pcadr  : INTEGER) ;
EXTERNAL ;

PROCEDURE PC_CANCEL
   (VAR stat   : WORD ;
        pc     : WORD ;
        oldtag : WORD ;
        typec  : WORD) ;
EXTERNAL;

PROCEDURE PCIF_CLOSE
   (VAR stat : WORD);
   EXTERNAL;

PROCEDURE PC_CONNECT
   (VAR stat   : WORD ;
        tag    : WORD ;
        contwd : WORD ;
        pc     : WORD) ;
EXTERNAL;

PROCEDURE PC_DISC
   (VAR stat    : WORD ;
        pc      : WORD ;
```

```
        password : WORD) ;
EXTERNAL;

PROCEDURE PC_DIUNSOL
   (VAR stat : WORD ;
        pc   : WORD);
EXTERNAL;
$PAGE
PROCEDURE PC_ENQUIRY
   (VAR stat    : WORD ;
        oldstat : WORD ;
    VAR oldtag  : WORD ;
        contwd  : WORD ;
        pc      : WORD ;
    VAR buffr   : BUF_TYPE;
        lengr   : WORD ;
    VAR typer   : WORD ;
    VAR logr    : WORD) ;
EXTERNAL;

PROCEDURE PC_ENUNSOL
   (VAR stat : WORD ;
        pc   : WORD ;
        akey : WORD) ;
EXTERNAL;

PROCEDURE PCIF_ERROR
   (    stat   : WORD ;
    VAR buffer : STRING80 ;
    VAR length : WORD) ;
EXTERNAL ;

PROCEDURE PC_GETKEY
   (VAR stat : WORD ;
    VAR akey : WORD);
   EXTERNAL;

PROCEDURE PC_LOCK
   (VAR stat : WORD ;
        pc   : WORD) ;
EXTERNAL;

PROCEDURE PCIF_OPEN
   (VAR stat : WORD);
EXTERNAL;

PROCEDURE PC_PCSTAT
   (VAR stat   : WORD ;
        tag    : WORD ;
        contwd : WORD ;
        pc     : WORD ;
    VAR buffr  : BUF_TYPE) ;
EXTERNAL;
PROCEDURE PC_IDENT
```

```
    (VAR stat    : WORD ;
         tag     : WORD ;
         contwd  : WORD ;
         pc      : WORD ;
     VAR buffr   : BUF_TYPE;
         lengt   : WORD) ;
EXTERNAL;
$PAGE
PROCEDURE PC_READD
    (VAR stat    : WORD ;
         tag     : WORD ;
         contwd  : WORD ;
         pc      : WORD ;
     VAR buffr   : BUF_TYPE ;
         lengt   : WORD ;
         pcadr   : INTEGER);
EXTERNAL;

PROCEDURE PC_READP
    (VAR stat    : WORD ;
         tag     : WORD ;
         contwd  : WORD ;
         pc      : WORD ;
     VAR buffr   : BUF_TYPE ;
         lengt   : WORD ;
         pcadr   : INTEGER);
EXTERNAL;

PROCEDURE PC_RELKEY
    (VAR stat : WORD ;
     VAR akey : WORD);
    EXTERNAL;

PROCEDURE PC_START
    (VAR stat             : WORD ;
         tag, contwd, pc  : WORD) ;
EXTERNAL;

PROCEDURE PC_STOP
    (VAR stat             : WORD ;
         tag, contwd, pc  : WORD) ;
EXTERNAL;

PROCEDURE PC_SYSTAT
    (VAR stat  : WORD ;
         pc    : WORD ;
     VAR buffr : BUF_TYPE);
EXTERNAL;

PROCEDURE PC_TRANS
    (VAR stat                      : WORD ;
         tag, contwd, pc, subfct   : WORD ;
     VAR buffs                     : BUF_TYPE ;
         lengs                     : WORD ;
```

```
    VAR buffr                    : BUF_TYPE ;
        lengr                    : WORD) ;
EXTERNAL;

PROCEDURE PC_UNLOCK
   (VAR stat : WORD ;
        pc   : WORD) ;
EXTERNAL;
$PAGE
PROCEDURE PC_WRITED
   (VAR stat            : WORD ;
        tag, contwd, pc : WORD ;
    VAR buffr           : BUF_TYPE ;
        lengt           : WORD ;
        pcadr           : INTEGER);
EXTERNAL;

PROCEDURE PC_WRITEP
   (VAR stat            : WORD ;
        tag, contwd, pc : WORD ;
    VAR buffr           : BUF_TYPE ;
        lengt           : WORD ;
        pcadr           : INTEGER);
EXTERNAL;

PROCEDURE PCIF_build_error_msg
          (    file_namr : STRING20 ;
               tag       : STRING2 ;
               error     : WORD ;
               promptof  : WORD ;
           VAR buffer    : STRING80 ;
           VAR length    : WORD) ;
EXTERNAL ;

{-----------------------------------------------}
{                                               }
{    RTE LIBRARY : PROCEDURE CALL DEFINITIONS   }
{                                               }
{-----------------------------------------------}

PROCEDURE RTE_CNUMD $ALIAS 'CNUMD'$
          (    number        : WORD ;
           VAR ascii         : STRING6) ;
EXTERNAL ;

PROCEDURE RTE_FTIME $ALIAS 'FTIME'$
          (VAR buffer : STRING80) ;
EXTERNAL ;
$PAGE
{---------------------------------------}
{                                       }
{        f i n d _ i n t e g e r        }
{                                       }
{---------------------------------------}
```

```
FUNCTION find_integer(VAR number : INTEGER) : BOOLEAN ;

CONST
   char_digit ='0123456789' ;

VAR
   index1       : WORD ;
   command_char : CHAR ;
   end_number_flag : BOOLEAN ;
   negative_flag   : BOOLEAN ;

BEGIN

   negative_flag := FALSE ;
   find_integer:= FALSE ;
   WHILE (command_ptr<70) AND (command_string[command_ptr]=' ')
   DO command_ptr := command_ptr + 1 ;
   IF (command_string[command_ptr]='-')
   THEN BEGIN
         negative_flag := TRUE ;
         command_ptr := command_ptr+1 ;
        END ;
   end_number_flag := FALSE ;
   number := 0 ;
   WHILE (NOT end_number_flag) AND (command_ptr<=80) DO
   BEGIN
      command_char := command_string[command_ptr] ;
      index1 := 0 ;
      found_flag := FALSE ;
      REPEAT
         BEGIN
            index1 := index1 + 1 ;
            IF (char_digit[index1]=command_char)
            THEN found_flag := TRUE ;
         END ; { end REPEAT }
      UNTIL
         found_flag OR (index1>=10) ;
$PAGE
      IF found_flag
      THEN BEGIN
              number := number * 10 + index1 - 1 ;
              find_integer:= TRUE ;
           END
      ELSE BEGIN
              end_number_flag := TRUE ;
           END
      ; { end if found_flag }
      IF (command_char='*')
         OR
         (command_char='a')
         OR
         (command_char='A')
      THEN stop_flag := TRUE
      ELSE command_ptr := command_ptr + 1 ;
```

```
   END ; { end WHILE NOT end_number_flag }
   IF (negative_flag) THEN number := -number ;

END ; { end of PROCEDURE find_integer}
$PAGE
{---------------------------------------}
{                                       }
{        f i n d _ n u m b e r          }
{                                       }
{---------------------------------------}

FUNCTION find_number (VAR number : WORD) : BOOLEAN ;

VAR
   long_number : INTEGER ;
   flag        : BOOLEAN ;

BEGIN

   flag := find_integer (long_number) ;
   IF flag AND (NOT stop_flag)
   THEN BEGIN
         find_number := TRUE ;
         IF (long_number>=-32768) AND (long_number<=32767)
         THEN number := long_number
         ELSE find_number := FALSE ;
        END
   ELSE BEGIN
         find_number := FALSE ;
        END
   ; { end IF flag }

END ; { end of PROCEDURE find_number }
$PAGE
{-----------------------------------}
{                                   }
{       g e t _ i n p u t           }
{                                   }
{-----------------------------------}

PROCEDURE get_input (ask : STRING11) ;

VAR
   eof_flag : BOOLEAN ;

BEGIN
  eof_flag := TRUE ;
  WHILE eof_flag DO
  BEGIN
   PROMPT(ask) ;
   IF EOF THEN BEGIN
                  RESET ;
                  eof_flag := TRUE ;
               END
```

```
                ELSE BEGIN
                        READLN(command_string) ;
                        command_ptr := 1 ;
                        eof_flag := FALSE ;
                     END
    ; { end IF EOF }
  END ; { end WHILE ... }

END ; { end of PROCEDURE get_input }
$PAGE
{--------------------------------------}
{                                      }
{       g e t _ f u n c t i o n        }
{                                      }
{--------------------------------------}

PROCEDURE get_function ;

VAR
   flag : BOOLEAN ;
   index : WORD ;
   temp  : CHAR ;
   shift : WORD ;

BEGIN

   FOR index:=1 TO 20 DO
      func[index]:=' ' ;
   flag := TRUE ;
   WHILE (command_ptr<=80) AND flag DO
   BEGIN
      IF (command_string[command_ptr]<>' ') THEN flag:=FALSE
      ELSE command_ptr := command_ptr+1 ;
   END ;
   flag:=TRUE ;
   index := 1 ;
   WHILE (command_ptr<=80) AND flag DO
   BEGIN
      IF (command_string[command_ptr]=' ')
      THEN BEGIN
              flag:=FALSE ;
           END
      ELSE BEGIN
            IF (index<=20)
            THEN BEGIN
                  shift := ORD('a')-ORD('A') ;
                  temp := command_string[command_ptr] ;
                  CASE temp OF
                  'a'..'z': temp := CHR(ORD(temp)-shift);
                  OTHERWISE ;
                  END ; { end CASE ... }
                  func[index]:=temp ;
                 END
           ; { end IF ... }
```

```
            command_ptr := command_ptr+1 ;
            index := index + 1 ;
           END
      ; { end IF ... }
   END ;

END ; { end of PROCEDURE get_function }
$PAGE
{-----------------------------------}
{                                   }
{          g e t _ p c              }
{                                   }
{-----------------------------------}

PROCEDURE get_pc ;

VAR
   flag : BOOLEAN ;

BEGIN

   REPEAT
      flag := find_number(pc) ;
      IF flag OR stop_flag
      THEN BEGIN
           END
      ELSE BEGIN
            get_input (ask_pc) ;
           END
      ; { end IF }
   UNTIL (flag OR stop_flag) ;

END ;
$PAGE
{--------------------------------------}
{                                      }
{          g e t _ s u b f n c         }
{                                      }
{--------------------------------------}

PROCEDURE get_subfnc;

VAR
   flag : BOOLEAN ;

BEGIN

   REPEAT
      flag := find_number(subfnc) ;
      IF flag OR stop_flag
      THEN BEGIN
           END
      ELSE BEGIN
            get_input (ask_subfnc) ;
```

```
            END
      ; { end IF }
   UNTIL (flag OR stop_flag) ;

END ;
$PAGE
{-----------------------------------}
{                                   }
{       g e t _ l e n g t h         }
{                                   }
{-----------------------------------}

PROCEDURE get_length ;

VAR
   flag : BOOLEAN ;

BEGIN

   REPEAT
      flag := find_number(lengt) ;
      IF flag OR stop_flag
      THEN BEGIN
           END
      ELSE BEGIN
            get_input (ask_lengt) ;
           END
      ; { end IF }
   UNTIL (flag OR stop_flag) ;

END ;
$PAGE
{-----------------------------------}
{                                   }
{        g e t _ l o g r            }
{                                   }
{-----------------------------------}

PROCEDURE get_logr ;

VAR
   flag : BOOLEAN ;

BEGIN

   REPEAT
      flag := find_number(logr) ;
      IF flag OR stop_flag
      THEN BEGIN
           END
      ELSE BEGIN
            get_input (ask_logr) ;
           END
```

```
      ; { end IF }
   UNTIL (flag OR stop_flag) ;

END ;
$PAGE
{-------------------------------------}
{                                     }
{       g e t _ p a s s w o r d       }
{                                     }
{-------------------------------------}

PROCEDURE get_password ;

VAR
   flag : BOOLEAN ;

BEGIN

   REPEAT
      flag := find_number(password) ;
      IF flag OR stop_flag
      THEN BEGIN
           END
      ELSE BEGIN
            get_input (ask_passwd);
           END
      ; { end IF }
   UNTIL (flag OR stop_flag) ;

END ;
$PAGE
{-----------------------------------}
{                                   }
{          g e t _ t a g            }
{                                   }
{-----------------------------------}

PROCEDURE get_tag ;

VAR
   flag : BOOLEAN ;

BEGIN

   REPEAT
      flag := find_number(tag) ;
      IF flag OR stop_flag
      THEN BEGIN
           END
      ELSE BEGIN
            get_input (ask_tag) ;
           END
      ; { end IF }
   UNTIL (flag OR stop_flag) ;
```

```
END ;
$PAGE
{-----------------------------------------}
{                                         }
{       g e t _ t y p e _ c a n c e l     }
{                                         }
{-----------------------------------------}

PROCEDURE get_type_cancel ;

VAR
   flag : BOOLEAN ;

BEGIN

   REPEAT
      flag := find_number(typec) ;
      IF flag OR stop_flag
      THEN BEGIN
           END
      ELSE BEGIN
            get_input (ask_type_ca) ;
           END
      ; { end IF }
   UNTIL (flag OR stop_flag) ;

END ;
$PAGE
{-----------------------------------}
{                                   }
{       g e t _ p c a d r           }
{                                   }
{-----------------------------------}

PROCEDURE get_pcadr;

VAR
   flag : BOOLEAN ;

BEGIN

   REPEAT
      flag := find_integer (pcadr) ;
      IF flag OR stop_flag
      THEN BEGIN
           END
      ELSE BEGIN
            get_input (ask_pcadr) ;
           END
      ; { end IF }
   UNTIL (flag OR stop_flag) ;

END ;
$PAGE
```

```
{-----------------------------------------}
{                                         }
{       g e t _ a c c e s s _ k e y       }
{                                         }
{-----------------------------------------}

PROCEDURE get_access_key ;

VAR
   flag : BOOLEAN ;

BEGIN

   REPEAT
      flag := find_number(key) ;
      IF flag OR stop_flag
      THEN BEGIN
           END
      ELSE BEGIN
            get_input (ask_key);
           END
      ; { end IF }
   UNTIL (flag OR stop_flag) ;

END ;
$PAGE
{-------------------------------------------------------------}
{                                                             }
{       g e t _ r e a d _ w r i t e _ p a r a m e t e r s     }
{                                                             }
{-------------------------------------------------------------}

PROCEDURE get_read_write_parameters ;

BEGIN

   get_pc ;
   get_length ;
   get_pcadr ;
   IF asynch_flag
   THEN BEGIN
         get_tag ;
         get_access_key ;
        END ;

END ; { end of PROCEDURE get_read_write_parameters }
$PAGE
{-----------------------------------------}
{                                         }
{       g e t _ e r r o r _ n u m b e r   }
{                                         }
{-----------------------------------------}

PROCEDURE get_error_number ;
```

```
VAR
   flag : BOOLEAN ;

BEGIN

   REPEAT
      flag := find_number(stat) ;
      IF flag OR stop_flag
      THEN BEGIN
           END
      ELSE BEGIN
            get_input (ask_error) ;
           END
      ; { end IF }
   UNTIL (flag OR stop_flag) ;

END ;
$PAGE
{-----------------------------------}
{                                   }
{         g e t _ y e s             }
{                                   }
{-----------------------------------}

FUNCTION get_yes : BOOLEAN ;

CONST
   yes = STRING3 ['YES'] ;

BEGIN

      get_input (ask_yes) ;
      get_function ;
      IF (func=yes) OR (func='Y')
      THEN get_yes := TRUE
      ELSE get_yes := FALSE ;

END ; { end of PROCEDURE get_yes }
$PAGE
{-----------------------------------}
{                                   }
{       g e t _ b u f f r           }
{                                   }
{-----------------------------------}

PROCEDURE get_buffr (index : WORD) ;

CONST
   init_ask = STRING11['          ? '] ;


VAR
   flag : BOOLEAN ;
   value : WORD  ;
```

```
   temp  : STRING6 ;
   index2 : WORD ;

BEGIN

   ask_bufval := init_ask ;
   RTE_CNUMD(index-1,temp) ;
   FOR index2:=1 TO 6 DO ask_bufval[index2]:=temp[index2] ;
   REPEAT
      flag := find_number (value) ;
      IF flag OR stop_flag
      THEN BEGIN
           buff1[index] := value ;
          END
      ELSE BEGIN
           get_input (ask_bufval) ;
          END
      ; { end IF }
   UNTIL (flag OR stop_flag) ;

END ;
$PAGE
{-----------------------------------------}
{                                         }
{       p r i n t _ F M P _ e r r o r     }
{                                         }
{-----------------------------------------}

PROCEDURE print_FMP_msg (error : WORD) ;
CONST
   error_file_namr =STRING20['"PCMER             '] ;

VAR
   err_buff : STRING80 ;
   err_len  : WORD ;

BEGIN

   PCIF_build_error_msg (error_file_namr,
                         'FM',error,0,err_buff,err_len) ;
   WRITELN(' file error : ',err_buff) ;


END ;
$PAGE
{-----------------------------------------}
{                                         }
{         g e t _ f i l e _ n a m r       }
{                                         }
{-----------------------------------------}

PROCEDURE get_file_name ;
CONST
   no_namr = STRING20['                   '] ;
```

```
VAR
   index : WORD ;

BEGIN

   REPEAT
      get_function ;
      IF func=no_namr
      THEN BEGIN
            get_input (ask_file) ;
           END
      ELSE BEGIN
            file_namr := func ;
{-}         FOR index:=1 TO 6 DO file_name[index]:=file_namr[index];
           END
      ; { end IF ... }
   UNTIL (func<>no_namr) ;

END ;
$PAGE
{-------------------------------------------------------}
{                                                       }
{   p r i n t _ P C I F _ f u n c t i o n _ n a m e     }
{                                                       }
{-------------------------------------------------------}

PROCEDURE print_PCIF_function_name (function_code : WORD) ;

BEGIN

   CASE function_code OF
   1 : WRITE ('READD') ;
   2 : WRITE ('WRITED') ;
   3 : WRITE ('READP') ;
   4 : WRITE ('WRITEP') ;
   8 : WRITE ('TRANS') ;
   16: WRITE ('START') ;
   17: WRITE ('STOP') ;
   20: WRITE ('CONNECT') ;
   21: WRITE ('DISC') ;
   22: WRITE ('PCSTAT') ;
   30: WRITE ('IDENT') ;
   OTHERWISE WRITE ('unknown') ;
   END ; { end CASE function_code }

END ; { end of PROCEDURE print_PCIF_function_name }
$PAGE
{-----------------------------------------}
{                                         }
{       p r i n t _ P C _ s t a t u s     }
{                                         }
{-----------------------------------------}

PROCEDURE print_PC_status (status : WORD) ;
```

```
TYPE
   STAT_TYPE =
      RECORD
         CASE INTEGER OF
         0 : (word_access : WORD) ;
         1 : (bit_access  : PACKED ARRAY [1..16] OF BOOLEAN) ;
      END ;

VAR
   temp_stat : STAT_TYPE ;

BEGIN

 IF (stat=0)
 THEN BEGIN
   temp_stat.word_access := status ;
   WRITE(' -> PC run mode=') ;
   IF temp_stat.bit_access[16] THEN WRITE('OFF')
                      ELSE WRITE('ON') ;
   WRITE('; download accept=') ;
   IF temp_stat.bit_access[15] THEN WRITE('OFF')
                      ELSE WRITE('ON') ;
   IF temp_stat.bit_access[14] THEN WRITE('; PC not online') ;
   WRITELN ;
 ENd ; { end IF }

END ; { end of PROCEDURE print_PC_status }
$PAGE
{-------------------------------------------------}
{                                                 }
{       p r i n t _ s y s t e m _ s t a t u s     }
{                                                 }
{-------------------------------------------------}

PROCEDURE print_PC_system_status (status : WORD) ;

TYPE
   STAT_TYPE =
      RECORD
         CASE INTEGER OF
         0 : (word_access : WORD) ;
         1 : (bit_access  : PACKED ARRAY [1..16] OF BOOLEAN) ;
      END ;

VAR
   temp_stat : STAT_TYPE ;

BEGIN

 IF (stat=0)
 THEN BEGIN
   temp_stat.word_access := status ;
   IF (NOT temp_stat.bit_access[4])
   THEN BEGIN
```

```
        WRITE(' -> unsolicited ') ;
        IF temp_stat.bit_access[9] THEN WRITELN('DISABLEd')
                                   ELSE WRITELN('ENABLEd') ;
        WRITE('   ') ;
       END
   ELSE WRITE(' ->') ;
   IF temp_stat.bit_access[10] THEN WRITELN(' DISConnected')
                      ELSE WRITELN(' CONNECTED') ;
   IF temp_stat.bit_access[11] THEN WRITELN('    LOCKED')
                      ELSE WRITELN('    UNlocked') ;
   IF temp_stat.bit_access[12] THEN WRITELN('    pending rqst')
                      ELSE WRITELN('    NO rqst pending') ;
   WRITE('    capability :') ;
   IF (NOT temp_stat.bit_access[1])
      THEN WRITE(' WriteData') ;
   IF (NOT temp_stat.bit_access[2])
      THEN WRITE(' WriteProgram') ;
   IF (NOT temp_stat.bit_access[3])
      THEN WRITE(' TRansparent') ;
   IF (NOT temp_stat.bit_access[4])
      THEN WRITE(' UNsolicited') ;
   IF (NOT temp_stat.bit_access[5])
      THEN WRITE(' STart/STop') ;
   WRITELN ;
 END ; { end IF (stat=0) }

END ; { end of PROCEDURE print_PC_system_status }
$PAGE
{-------------------------------------------------}
{                                                 }
{    p r i n t _ p c _ i d e n t _ s t a t u s    }
{                                                 }
{-------------------------------------------------}

PROCEDURE print_pc_ident_status  (VAR buff : BUF_TYPE) ;

TYPE
   V_TYPE =
     RECORD
       CASE INTEGER OF
       0 : (word_access : WORD) ;
       1 : (char_access  : STRING2) ;
     END ;

VAR
   vendor : V_TYPE ;
   i : WORD;

BEGIN {print_pc_ident_status}

 IF (stat=0)
 THEN BEGIN
   vendor.word_access := buff[1];
   WRITE(' -> Vendor       =    ') ;
```

```
   FOR i := 1 TO 2 DO WRITE (vendor.char_access[i]);
   WRITELN;
   WRITE('   ') ;
   WRITELN(' Model number =',buff[2]:6);
   WRITE('   ');
   WRITELN(' Station ID   =', buff[3]:6) ;
 END ; { end IF (stat=0) }

END ; { end of PROCEDURE print_PC_ident_status }
$PAGE
{*************************************************************}
{                                                             }
{     P C I F / 1 0 0 0   E X E R C I Z E R   :   M A I N     }
{                                                             }
{*************************************************************}

BEGIN
WRITELN ;
WRITELN(' PCIF/1000 exercizer 94200-16404 REV.2525 <850609.2200>'
       ) ;
RTE_FTIME (error_buffer) ;
WRITELN(error_buffer) ;
WRITELN ;
icr:=0 ;
isecu:=0 ;
tag := 0 ;
key := 0 ;
contwd := 0 ;
stop_flag := FALSE ;
echo_flag := FALSE ;
asynch_flag := FALSE ;
j:=1;
FOR i:=1 TO 512 DO
     BEGIN
     buff1[i]:=j;
     IF (j>2) THEN j:=1 ELSE j:=j+1;
     END;
func:='?' ;
$PAGE
WHILE (func<>'EX') AND (func<>'EXIT') DO
     BEGIN
     stat:=0;
     tag := 0 ;
     key := 0 ;
     stop_flag := FALSE ;
     get_input (ask_command) ;
     get_function ;
     IF (func='?') OR (func='??')
     THEN BEGIN
          WRITELN(' PCIF/1000 access routines may be called via') ;
          WRITELN('   their name (without typing PC_ or PCIF_)') ;
          WRITELN ;
          WRITELN('   Calls may be made in :');
          WRITELN('      in synchronous mode (tag & key always=0)');
```

```
          WRITELN('      or asynchronous (tag and access key are asked)') ;
          WRITELN('      Mode is changed by using the function : MODE') ;
          WRITELN ;
          WRITELN('   Parameters may be entered at once on the same line') ;
          WRITELN('      but missing parameters will be explicitly asked') ;
          WRITELN('      Input can be stopped by entering "*","a" or "A"') ;
          WRITELN ;
          WRITELN('  Special functions are : MODE   BUFRD  BUFWR EXIT EX') ;
          WRITELN('                         DWLOAD UPLOAD ERROR ECHO') ;
        END
$PAGE
     ELSE IF (func='MODE')
     THEN BEGIN
          WRITE(' current mode is ') ;
          IF asynch_flag THEN WRITE(' a') ;
          WRITELN('synchronous') ;
          WRITELN(' to change mode type "YES". All other input will',
                  ' not do any change') ;
          IF get_yes
          THEN BEGIN
               IF asynch_flag THEN asynch_flag := FALSE
                              ELSE asynch_flag := TRUE ;
             END
          ELSE BEGIN
               WRITELN(' NO change made') ;
             END ;
        END
$PAGE
     ELSE IF (func='READD')
     THEN BEGIN
          get_read_write_parameters ;
          IF (NOT stop_flag)
           THEN PC_READD (stat,tag,key,pc,buffl,lengt,pcadr)
        END
     ELSE IF (func='READP')
     THEN BEGIN
          get_read_write_parameters ;
          IF (NOT stop_flag)
           THEN PC_READP (stat,tag,key,pc,buffl,lengt,pcadr)
        END
     ELSE IF (func='WRITED')
     THEN BEGIN
          get_read_write_parameters ;
          IF (NOT stop_flag)
           THEN PC_WRITED (stat,tag,key,pc,buffl,lengt,pcadr)
        END
     ELSE IF (func='WRITEP')
     THEN BEGIN
          get_read_write_parameters ;
          IF (NOT stop_flag)
           THEN PC_WRITEP (stat,tag,key,pc,buffl,lengt,pcadr)
        END
$PAGE
     ELSE IF (func='OPEN')
```

```
     THEN BEGIN
          PCIF_OPEN (stat);
          END
     ELSE IF (func='CLOSE')
     THEN BEGIN
          PCIF_CLOSE (stat)
          END
     ELSE IF (func='LOCK')
     THEN BEGIN
          get_pc;
          IF (NOT stop_flag) THEN PC_LOCK (stat,pc);
          END
     ELSE IF (func='UNLOCK')
     THEN BEGIN
          get_pc;
          IF (NOT stop_flag) THEN PC_UNLOCK (stat,pc);
          END
     ELSE IF (func='CONNECT')
     THEN BEGIN
          get_pc;
          IF asynch_flag
          THEN BEGIN
               get_tag ;
               get_access_key ;
              END ;
          IF (NOT stop_flag) THEN PC_CONNECT (stat,tag,key,pc);
          END
     ELSE IF (func='DISC') OR (func='DISCONNECT')
     THEN BEGIN
          get_pc;
          get_password ;
          IF (NOT stop_flag) THEN PC_DISC (stat,pc,password) ;
          END
$PAGE
     ELSE IF (func='PCSTAT')
     THEN BEGIN
          get_pc;
          IF asynch_flag
          THEN BEGIN
               get_tag ;
               get_access_key ;
              END ;
          IF (NOT stop_flag)
          THEN BEGIN
               PC_PCSTAT (stat,tag,key,pc,buff1);
               print_PC_status(buff1[1]) ;
              END ;
          END
     ELSE IF (func='IDENT')
     THEN BEGIN
          get_pc;
          IF asynch_flag
          THEN BEGIN
               get_tag ;
```

```
            get_access_key ;
           END ;
      IF (NOT stop_flag)
      THEN BEGIN
            PC_IDENT (stat,tag,key,pc,buff1,10);
            print_PC_ident_status(buff1) ;
           END ;
      END
 ELSE IF (func='SYSTAT')
 THEN BEGIN
      get_pc;
      IF (NOT stop_flag)
      THEN BEGIN
            PC_SYSTAT (stat,pc,buff1);
            print_PC_system_status(buff1[1]) ;
           END ;
      END
 ELSE IF (func='START')
 THEN BEGIN
      get_pc;
      IF (asynch_flag)
      THEN BEGIN
            get_tag ;
            get_access_key ;
           END ;
      IF (NOT stop_flag) THEN PC_START (stat,tag,key,pc);
      END
 ELSE IF (func='STOP')
 THEN BEGIN
      get_pc;
      IF asynch_flag
      THEN BEGIN
            get_tag ;
            get_access_key ;
           END ;
      IF (NOT stop_flag) THEN PC_STOP (stat,tag,key,pc);
      END
$PAGE
 ELSE IF (func='TRANS')
 THEN BEGIN
      get_pc ;
      get_length ;
      get_logr ;
      get_subfnc;
      IF asynch_flag
      THEN BEGIN
             get_tag ;
             get_access_key ;
           END ;
      IF (NOT stop_flag)
      THEN BEGIN
             PC_TRANS (stat,tag,key,pc,subfnc,
                       buff1,lengt,buff1,logr);
             IF (stat=0) THEN lengt:=logr ;
```

```
                IF ((lengt > -24576) AND (lengt < 0)) OR
                   ((lengt > 512) AND (lengt < 16384)) OR
                   ((lengt > 17384) AND (lengt < 32767))
                   THEN lengt := 512;
             END ;
        END
   ELSE IF (func='GETKEY')
   THEN BEGIN
        PC_GETKEY (stat,key);
        WRITELN (' key : ',key:6);
        END
   ELSE IF (func='RELKEY')
   THEN BEGIN
        get_access_key ;
        IF (NOT stop_flag) THEN PC_RELKEY (stat,key)
        END
   ELSE IF (func='CANCEL')
   THEN BEGIN
        get_pc ;
        get_tag ;
        get_type_cancel ;
        IF (NOT stop_flag) THEN  PC_CANCEL (stat,pc,tag,typec);
        END
$PAGE
   ELSE IF (func='CALL')
   THEN BEGIN
        get_pc ;
        get_length ;
        get_subfnc ;
        get_pcadr ;
        IF asynch_flag
        THEN BEGIN
                get_tag ;
                get_access_key ;
             END ;
        IF (NOT stop_flag)
        THEN BEGIN
                PC_CALL (stat,tag,key,pc,subfnc,
                         buff1,lengt,pcadr);
                IF (stat=0) AND (subfnc>31) THEN lengt := 0 ;
                IF (lengt>512) THEN lengt:=512 ;
             END ;
        END
   ELSE IF (func='ENQUIRY')
   THEN BEGIN
        get_access_key ;
        contwd := key - 32768 ;
        IF (NOT stop_flag)
        THEN BEGIN
             PC_ENQUIRY (stat,oldstat,tag,contwd,
                         pc,buff1,256,typer,logr);
             lengt := logr DIV 2 ;
             IF ((logr MOD 2)<>0)
                THEN lengt := lengt + 1 ;
```

```
            IF (stat=0)
            THEN BEGIN
                 IF (typer<>0)
                 THEN BEGIN
                      WRITE(' message get for PC',pc:6,
                              ' reply to ') ;
                      IF (typer<0)
                      THEN BEGIN
                            WRITE(' unsolicited ');
                            print_PCIF_function_name (-typer) ;
                           END
                      ELSE BEGIN
                            WRITE(' reply to ') ;
                            print_PCIF_function_name (typer) ;
                           END
                      ; { end IF (typer<0) }
                      WRITELN(' typer=',typer:6);
                      WRITELN('                tag',tag:6,
                              ' logr =',logr:6) ;
                      stat := oldstat ;
                     END
                 ELSE BEGIN
                  WRITELN(' no message currently in access key') ;
                  WRITELN(' To wait on enquiry use function : ENQUIRYW');
                      END
                 ; { end IF (typer<>0) }
                END ; { end IF (stat=0) }
            END ; { end IF (NOT stop_flag) }
       END
$PAGE
     ELSE IF (func='ENQUIRYW')
     THEN BEGIN
          get_access_key ;
          contwd := key ;
          IF (NOT stop_flag)
          THEN BEGIN
               PC_ENQUIRY (stat,oldstat,tag,contwd,
                           pc,buff1,256,typer,logr);
               lengt := logr DIV 2 ;
               IF ((logr MOD 2)<>0)
                  THEN lengt := lengt + 1 ;
               CASE stat OF
               0 : BEGIN
                    WRITE(' message get for PC',pc:6) ;
                    IF (typer<0)
                    THEN BEGIN
                          WRITE(' unsolicited ');
                          print_PCIF_function_name (-typer) ;
                         END
                     ELSE BEGIN
                          WRITE(' reply to ') ;
                          print_PCIF_function_name (typer) ;
                         END
                    ; { end IF (typer<0) }
```

```
                    WRITELN(' typer=',typer:6);
                    WRITELN('                tag',tag:6,
                            ' logr =',logr:6) ;
                    stat := oldstat ;
                  END ;
               OTHERWISE BEGIN END ;
               END ; { end CASE stat }
               END ; { end IF (NOT stop_flag) }
          END
$PAGE
     ELSE IF (func='ENUNSOL')
     THEN BEGIN
          get_pc ;
          get_access_key ;
          IF (NOT stop_flag) THEN PC_ENUNSOL (stat,pc,key);
          END
     ELSE IF (func='DIUNSOL')
     THEN BEGIN
          get_pc ;
          IF (NOT stop_flag) THEN PC_DIUNSOL (stat,pc);
          END
$PAGE
     ELSE IF (func='BUFRD')
     THEN BEGIN
          IF (lengt < 16384) AND (lengt > 0)
              THEN  FOR i:=1 TO lengt DO
                    WRITELN(i-1:6,' -> ',buff1[i]:7)
              ELSE
               {*CGY* Added below to print bit length lengt buffers}
               IF (lengt < 0)
                 THEN
                  BEGIN
                    bit_length.wd := lengt;
                    nb_displayed := bit_length.lo_bits DIV 16;
                    FOR i := 1 TO nb_displayed DO
                      writeln (i-1:6,' -> ',buff1[i]:7);
                    IF (bit_length.lo_bits MOD 16 <> 0) THEN
                      BEGIN {print remainder, zero out unused bits}
                        tempo.wrd := buff1[nb_displayed + 1];
                        FOR i := 1 TO (16 - (bit_length.lo_bits MOD 16)) DO
                          tempo.bit_array[(bit_length.lo_bits MOD 16)+i]:=0;
                        writeln (nb_displayed:6,' -> ',
                                 tempo.wrd:7);
                      END;
                  END
                  {*CGY* End of mod.}
                 ELSE
                  BEGIN
                    nb_displayed:= (lengt - 16384) DIV 2;
                    IF (2 * nb_displayed <> lengt-16384) THEN
                       nb_displayed:= nb_displayed + 1;
                    FOR i:=1 TO nb_displayed DO
                       BEGIN
                         tempo.wrd:= buff1[i];
```

```
                        WRITELN (tempo.bt1:7);
                        IF (i<nb_displayed) OR
                           (2 * nb_displayed = lengt-16384) THEN
                               WRITELN (tempo.bt2:7);
                     END;
                END;
          END
     ELSE IF (func='BUFWR')
     THEN BEGIN
          WRITELN(' enter values as INTEGER, enter ''*'' or ''a'' to stop');
          I := 0 ;
          WHILE ((I<=511) AND (NOT stop_flag)) DO
          BEGIN
            I := I + 1 ;
            get_buffr(I) ;
          END ; { end of WHILE ((I<=511) OR (NOT stop_flag)) }
          stop_flag := FALSE ;
          END
$PAGE
     ELSE IF (func='DWLOAD')
     THEN BEGIN
          get_pc         ;
          get_pcadr      ;
          IF asynch_flag
          THEN BEGIN
                get_tag ;
                get_access_key ;
               END ;
          get_file_name ;
          IF (NOT stop_flag)
          THEN BEGIN
                reset(abdown,file_name);
                i:=0;
                WHILE (NOT eof(abdown)) AND (i<512) DO
                BEGIN
                   i:=i+1;
                   read (abdown,buff1[i]);
                END;{WHILE}
                PC_WRITEP (stat,tag,key,pc,buff1,i,pcadr);
                close (abdown);
               END ;
          END
$PAGE
     ELSE IF (func='UPLOAD')
     THEN BEGIN
          get_pc         ;
          get_pcadr      ;
          get_length     ;
          IF asynch_flag
          THEN BEGIN
                get_tag ;
                get_access_key ;
               END ;
          IF (NOT stop_flag)
```

```
          THEN BEGIN
                get_file_name;
                rewrite (abup,file_name);
                IF (lengt>512) THEN lengt:=512;
                PC_READP (stat,0,0,pc,buff1,lengt,pcadr);
                FOR i:=1 TO lengt DO
                BEGIN
                   write (abup,buff1[i]);
                END;{FOR}
                close (abup);
               END ;
          END
$PAGE
     ELSE IF (func='ERROR')
     THEN BEGIN
           get_error_number ;
          END
     ELSE IF (func='**')
     THEN BEGIN
          END
     ELSE IF (func='ECHO')
     THEN BEGIN
            echo_flag := TRUE ;
          END
     ELSE IF (func='EX') OR (func='EXIT')
     THEN BEGIN
           PCIF_CLOSE (stat) ;
           WRITELN ('  -> end of PCTEST program') ;
           stat := 0 ;
          END
     ELSE WRITELN ('  -> unknown test function name');
     IF (NOT stop_flag)
     THEN BEGIN
           IF (stat>0)
           THEN BEGIN
                 PCIF_ERROR(stat,error_buffer, error_length) ;
                 WRITELN(' -> error : ',error_buffer) ;
                END
           ELSE BEGIN
                 WRITELN ;
                END ; { end IF (stat>0) }
          END
     ELSE BEGIN
           WRITELN(' -> function not executed') ;
          END ; { end IF stop_flag }
     END;
END.
```

# INDEX

## A



## C

## R



## S

## T



## U



## W