HEWLETT-PACKARD

Dealer Configuration File Creation Guide

Notice

The information contained in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

Personal Computer Group
974 E. Arques Avenue
P.O. Box 3486
Sunnyvale, CA 94086, USA

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

HP PC Forum on CompuServe

The HP PC Forum is an easy way to obtain up-to-date information and answers to your questions about HP personal computers. The HP PC Forum is an online bulletin board messaging system maintained jointly by Hewlett-Packard and HP PC users. HP system operators answer questions and maintain libraries which contain contributed articles and software. Conferences are scheduled periodically for online discussions of selected topics.

The HP PC Forum is available through the CompuServe Information Service, the largest electronic information service in the world. To access the HP PC Forum, you must have an account with CompuServe and a PC with a 300, 1200, or 2400 baud modem. As a preferred Hewlett-Packard customer, you are invited to join the Hewlett-Packard forum on CompuServe at no charge. Simply call 1-800-848-8990 (614-457-0802 if outside U.S. and Canada) and ask for Representative #133. CompuServe will send you a free introductory membership immediately.

INTEREX, the International HP Users Group

INTEREX, the International Association of Hewlett-Packard Computer Users, is an independent global professional association offering a comprehensive range of services and activities, including publications, conferences, a library of current programs, and up-to-date information about the HP computing world. For a free prospective member packet, direct questions to the Member Services Department of INTEREX, 585 Maude Ct., Sunnyvale, CA 94088-3439. USA phone: (408) 738-4848, Telex: 4971527, FAX: (408) 736-2156.

1

Creating Configuration (CFG) Files

Configuration (CFG) files contain information about resources (IRQ lines, I/O port addresses) required by a board or option. Most boards and options require their own configuration file. EASY CONFIG uses the information in configuration files to create a configuration for your computer that is free of resource conflicts.

Chapter 1 of this guide contains general instructions for creating configuration files. Two examples of creating a configuration file are included at the end of chapter 1. Chapter 2 includes the EISA Configuration Language Specification. Appendix A contains information about using types and subtypes.

Use the information in this guide to create a configuration file only if you cannot obtain the file from another source.

Gathering Information

Before you begin to create a configuration file for a particular board or option, you need to find out a certain amount of information about the board or option that you are going to configure.

Some of this information may be found by reading the manual provided by the board or option's manufacturer. Other information may be found by examining the board or option.

Three types of information are required:

- General information: the name of the board, the name of the manufacturer, the physical characteristics of the board, and how the board will function in the computer.

- Switch/jumper/software/port initialization information: presence of switches or jumpers, type of switches and jumpers, software initialization requirements.
- Resource information: DMA, IRQ, port, and memory information.

After this information is gathered, it must be organized so that it may be used properly. Only after the information is properly organized should it be entered into the configuration file.

Organizing Information

The key to developing a configuration file for a board or option is understanding how functions, choices, and resources interact. Boards contain functions; functions have configuration alternatives called choices; choices define the set of resources needed for that configuration alternative; INIT statements within the choice specify the switch, jumper, I/O port, and software initializations required for that configuration alternative.

Functions

You must determine what the functions of this board or option are. A function is a certain capability provided to the computer by a board or option. Examples of common functions provided by boards are parallel printer interface, asynchronous interface, disk controllers, network interfaces, memory, and video control. If the board has several different capabilities (such as a serial/parallel board), or if a board has major identifying features which can be used in different ways (such as speed), several functions may need to be defined.

Choices and Resources

Each function may have one or more configuration alternatives; these are the choices for each function. Each choice may need to identify resources required for that choice. Resources are defined as memory, IRQs, DMA channels, and I/O port addresses. The specific initialization required is identified for

each choice also. These can be switch, jumper, software settings, or port initializations.

At times, it may be desirable to allow a choice to have multiple resource possibilities. This is called having resource alternatives. Use of resource alternatives is desired when the list of resource possibilities is very large and would require many choices, and when the end user does not care which resource is chosen (such as picking a range of I/O port addresses for the board to use).

The choices for each function, and alternatives for each choice, should be ordered sequentially, with the most recommended selection being the first one.

You also need to define the switch, jumper, and port settings needed to designate the board as a particular choice. For example, setting a four-position DIP switch to xx01 may indicate that the serial port is to function as COM1; setting the DIP switch to xx10 may indicate that the serial port is to function as COM2; setting the DIP switch to xx00 may indicate that the serial port is disabled; setting the DIP switch to 01xx may indicate that the parallel port is to function as LPT1; and setting the DIP switch to 10xx may indicate that the parallel port is to function as LPT2. Setting the DIP switch to 00xx may indicate that the parallel port is disabled.

One or more of the initialization methods (switches, jumpers, I/O ports, and software) may be used by the configuration.

INIT statements are used to initialize the switches and jumpers to the settings required to perform the desired choice or function. For example, (using the example of the serial/parallel board above), if you wish to use the board as serial port COM1, the switch must be initialized to xx01. Providing an initialization statement to this effect will automatically initialize this switch to the correct switch display when this particular use of the port is selected.

Defining Functions, Choices, and Resources

In order to correctly define functions, choices, and resources for a board or option, you should ask the following questions:

- What varieties of functionality are contained on this board? How many different ways can this board be used? For example, a serial/parallel board has two varieties of functionality: a serial port, and a parallel port. If you should see this capability, it is a function.
- How can each function be configured? For example, the serial port can act as COM port #1 or COM port #2, or can be disabled. The parallel port can act as line printer port (LPT) #1 or line printer port (LPT) #2, or can be disabled. Each of these two *functions* (serial port and parallel port) has three *choices*—COM1, COM2 and disabled, and LPT1, LPT2, and disabled.
- What things need to occur to select a particular choice? Which resources are used (such as an IRQ)? Which jumper or switch should be set to a particular setting in order to indicate to the computer that this is the way that you wish to use this board?
- What are logical groups of functions? For example, a serial and parallel port may be included in a communication ports group.
- Are the types of functions you have defined listed in appendix A, "Using Types and Subtypes?"

An Example of Defining Functions, Choices, and Resources

The following diagram illustrates the hierarchy of configuration information:

Board

Function	Serial Port			Parallel Port		
Choice	COM1	COM2	Disabled	LPT1	LPT2	Disabled
Resource	IRQ	IRQ		IRQ	IRQ	
Resource	Port	Port		Port	Port	
Resource	INIT	INIT		INIT	INIT	
Switch	xx01	xx10	xx00	01xx	10xx	00xx

The board illustrated above is a board which has two functions: serial port and parallel port. Each of these functions has several choices: the serial port can be used either as COM1 or COM2, and the parallel port can be used either as LPT1 or LPT2. Depending upon the choice, the computer may need to allocate different system resources. Therefore, the corresponding system resources must be defined for each individual choice (that is, for COM1, COM2, serial port disabled, LPT1, LPT2, and parallel port disabled).

Naming a Configuration File

To determine a name for a configuration file, use the following format.

!*AAAPPPr*.CFG

!	A configuration file name begins with an exclamation point.
AAA	The three uppercase character manufacturer ID.
PPP	The three-digit (0-Fh) product number. Each digit can be a hexidecimal value.
r	The one-digit revision number. The digit can be a hexidecimal value.
.CFG	The file name extension (CFG).

For example:

- !ACE1234.CFG
- !XYZ5678.CFG
- !ABC0000.CFG

The file name convention is the same for a system board, other board, embedded device, or virtual device. For example, a board with a product ID of ACE0101 has a configuration file named !ACE0101.CFG.

Creating a Configuration File

When you create a configuration file, you can either create a new configuration file, or you can modify an existing, similar configuration file. This chapter assumes that you will use a word processor or text editor to create a new configuration file.

The following general steps should be followed when creating a configuration file. Each step is explained in detail later in this section. (Refer to chapter 2 for further detailed information.) Two examples of creating a configuration file are included at the end of this chapter.

1. Create a Board Identification Block to identify and characterize the board.
2. Create an Initialization Information Block to define switches, jumpers, programmable I/O ports, and software used to configure the board.
3. Create a Function Statement Block for each function that contains choices for each function and initialization values and resources for each choice.
4. Verify the configuration file with the CHECKCFG utility.

Creating a Board Identification Block (Step 1)

The Board Information Block is used to identify the board or option to be installed in the computer, and to identify the type of slot required for this board or option.

BOARD
    ID = "*AAAPPPr*"
    NAME = "*text*"
    MFR = "*text*"
    CATEGORY = "*text*"
    [SLOT = *slot type* [,"*text*"]]
    [LENGTH = *value*]
    [SKIRT = YES | NO]
    [READID = YES | NO]
    [AMPERAGE = *value*]
    [BUSMASTER = *value*]
    [SIZING = *value*]
    [IOCHECK = VALID | INVALID]
    [DISABLE = SUPPORTED | UNSUPPORTED]
    [COMMENTS = "*text*"]
    [HELP = "*text*"]

The SLOT, LENGTH, and SKIRT statements are used to determine the recommended slot for the board or option.

The READID, BUSMASTER, IOCHECK, and DISABLE statements should be used if the board or option is an EISA board or option.

The AMPERAGE, SIZING, COMMENT, and HELP statements are not required.

Creating an Initialization Information Block (Step 2)

The Initialization Information Block allows you to define switches, jumpers, programmable I/O ports, and software used to configure the board. These items must occur in the configuration file before the function in which they are referenced.

Creating a Programmable I/O Port Identification Block

Programmable I/O Port Identification Blocks are used to identify ports required to configure a board. A Programmable I/O Port Identification Block is needed for each port addressed.

IOPORT(i) = *address*
    [SIZE = BYTE | WORD | DWORD]
    [INITVAL = [LOC (*bitlist*)] *value*]

IOPORT(i) = PORTVAR(*n*)
    [SIZE = BYTE | WORD | DWORD]
    [INITVAL = [LOC (*bitlist*)] *value*]

The SIZE and INITVAL statements are not required.

Creating a Switch Identification Block

Switch Identification Blocks are used to identify each set of switches on a board. A Switch Identification Block is needed for each set of switches on the board.

SWITCH(i) = *n*
    NAME = "*text*"
    STYPE = DIP | ROTARY | SLIDE
    [VERTICAL = YES | NO]
    [REVERSE = YES | NO]
    [LABEL = LOC(*switchlist) list*]
    [INITVAL = LOC(*switchlist) list*]
    [FACTORY = LOC(*switchlist) list*]
    [COMMENTS="*text*"]
    [HELP="*text*"]

The VERTICAL, REVERSE, INITVAL, FACTORY, COMMENTS, and HELP statements are not required.

Creating a Jumper Identification Block

Jumper Identification Blocks are used to identify sets of jumpers on a board. A Jumper Identification Block is needed for each set of jumpers on the board.

JUMPER(i) = *n*
    NAME = "*text*"
    JTYPE = INLINE | PAIRED | TRIPOLE
    [VERTICAL = YES | NO]
    [REVERSE = YES | NO]
    [LABEL = LOC(*jumperlist) list*]
    [INITVAL = LOC(*jumperlist) list*]
    [FACTORY = LOC(*jumperlist) list*]
    [COMMENTS="*text*"]
    [HELP="*text*"]

The VERTICAL, REVERSE, LABEL, INITVAL, FACTORY, COMMENTS, and HELP statements are not required.

Creating a Software Identification Block

Software Identification Blocks are used to identify software programs that are used to initialize the board. They will display a set of instructions for adding software statements to existing drivers or programs that are executed in order to use this board or option. These drivers or programs are typically those placed in the CONFIG.SYS or AUTOEXEC.BAT files.

SOFTWARE(i) = "*information*"

Creating a Function Statement Block (Step 3)

The Function Statement Block is used to define the functions of this board or option, the choices belonging to each function, and the resources which are needed by each choice.

CHOICE statements are used to place different resources and their corresponding information into logical groups.

RESOURCE statements are used to describe the system resources used by the configuration. The global system resources used by ISA computers are DMA channels, IRQ levels, I/O port address space, and memory address space. In addition to this, EISA computers also have specific resources associated with every slot.

Initialization statements provide the values needed to initialize programmable boards or options, or the switches and jumpers used on switch-programmable boards.

Organizing Functions

By definition of the configuration language, each configuration file must be structured in such a way as to observe a certain hierarchy of statements.

Groups. The GROUP statement is optional.

FUNCTION statement blocks can be grouped together using the GROUP statement, if desired. The GROUP statement is used to group functions that belong together under one title for display. For example:

```
GROUP = "Disk Storage"
    FUNCTION = "Flexible Disk A"
        .
        .
        .
    FUNCTION = "Flexible Disk B"
        .
        .
        .
    FUNCTION = "Fixed Drive 1"
        .
        .
        .
    FUNCTION = "Fixed Drive 2"
        .
        .
        .
ENDGROUP
```

A group of function blocks must contain at least one function. Each function in the group has one entry in EISA nonvolatile memory (CMOS), and the group's type string is prepended to each of the function's type strings. The group's name is used for display purposes.

Functions. The FUNCTION statement names a function of the board. The information that follows details the choices of resources used to implement the function and the steps used to initialize the function. Functions may or may not belong to a group.

A FUNCTION statement block can contain CHOICE statements or SUBFUNCTION statements, but not both.

Subfunctions. SUBFUNCTION statements are optional.

Ordinarily, FUNCTION statements are used to describe a single device or function of the board. The different alternatives for configuring the function are then listed in subsequent CHOICE statements. There may be cases, however, in which a single function of the board can be broken down into smaller pieces, each of which may be configured separately. In such cases you should be able to view and modify the configurations of each of these pieces individually, even though they were presented as a single function.

Individually configurable pieces of a single function can be grouped together under a FUNCTION statement using SUBFUNCTION statements. For example:

```
FUNCTION = "Serial Port"
    SUBFUNCTION = "COM Address"
        .
        .
        .
    SUBFUNCTION = "Baud Rate"
        .
        .
        .
    SUBFUNCTION = "Parity"
        .
        .
        .
```

Each SUBFUNCTION statement has all the components of a FUNCTION statement. A FUNCTION statement may not have CHOICE statement blocks followed by SUBFUNCTIONS.

SUBFUNCTION types are appended to the FUNCTION type.

A FUNCTION with SUBFUNCTIONS has only one entry in EISA nonvolatile memory (CMOS). This single entry contains the information from all of the subfunctions.

Choices. CHOICE statements are used to place different resource statements and the corresponding initialization into logical groups. Each FUNCTION or SUBFUNCTION must have at least one CHOICE statement, and can have as many as is necessary. Groups of related resource and initialization statements are called a resource/init groups. CHOICES can contain SUBCHOICES following the resource/init groups for the choice (if any). For example:

```
FUNCTION = "Serial Port"
    CHOICE = "COM1"
        .
        .
        .
    CHOICE = "COM2"
        .
        .
        .
    CHOICE = "Disable Serial Port"
        .
        .
        .
```

Subchoices. SUBCHOICES are used for complex statements that cannot be done in a single CHOICE statement. They should only be used in complex situations, such as handling certain memory configurations. SUBCHOICES are configuration alternatives that are not seen. You can still scroll through the possible configurations, including those present in SUBCHOICES, but a list of these alternatives is never presented, as is the case with separate, named choices.

A CHOICE can have as many SUBCHOICES as is necessary.

Resource/init groups can be placed before the SUBCHOICES in a CHOICE. These global resource/init groups are allocated regardless of the SUBCHOICE selected.

Each SUBCHOICE contains the resource and init statements required for that configuration.

SUBCHOICES are mutually exclusive. Only one SUBCHOICE will be selected per CHOICE by EASY CONFIG.

For example, you wish to add 1 MB of memory to the computer. If the computer only has a 512 KB base memory, you want the computer to allocate an additional 128 KB of base memory with the remainder as extended memory. If the computer has 640 KB base memory, you want the computer to allocate the entire 1 MB as extended memory. You do not wish this decision to be presented. The configuration file would be as follows:

Creating Resource Statements

Creating a DMA Statement. DMA statements are used to define DMA channels. This statement is required only by functions that use DMA channels. DMA values can be specified in decimal or hex.

DMA = *list* [SHARE = YES | NO | "*text*"] [SIZE = BYTE | WORD | DWORD] [TIMING = DEFAULT | TYPEA | TYPEB | TYPEC]

The SHARE, SIZE, and TIMING statements are not required.

Creating an IRQ Statement. The IRQ statement is used to define IRQ addresses. This statement is required only by functions that use IRQ addresses. IRQ values may be specified in decimal or hex.

IRQ = *list*
[SHARE = YES | NO | "*text*"]
[TRIGGER = LEVEL | EDGE]

The SHARE and TRIGGER statements are not required.

Creating a PORT Statement. A PORT statement is used to identify I/O ports. This statement is only required by functions that use I/O ports. PORT addresses must be given in hex. STEP and COUNT values can be given in decimal or hex.

PORT = *list* | *range* STEP = *value* [COUNT = *value*]
[SHARE = YES | NO | "*text*"]
[SIZE = BYTE | WORD | DWORD]

The SHARE, and SIZE statements are not required.

Creating a MEMORY Statement. MEMORY statements are used to define memory addresses. This statement is required only by functions that use memory addresses. If a linked, combined, or free-form resource group contains a MEMORY statement, it can contain no other RESOURCE statements. Memory amounts, addresses and STEP values may be given in decimal, hex, or decimal with a unit abbreviation.

MEMORY = *list* | *range* STEP = *value*
    [INIT Statements]
[ADDRESS = *list* | *range* STEP = *value*]
    [WRITABLE = YES | NO]
    [MEMTYPE = SYS | EXP | OTH | VIR]
    [SIZE = BYTE | WORD | DWORD]
    [DECODE = 20 | 24 | 32]
    [CACHE = YES | NO]
    [SHARE = YES | NO | "*text*"]

The WRITABLE, MEMTYPE, SIZE, DECODE, CACHE, and SHARE statements are not required.

Creating Resource Groupings

The resources for a choice are allowed to be grouped together in three different ways: *free, linked,* or *combined.*

Free. Resources grouped in a free group are independent of any other resources. They stand alone. For example, a free group may be created as such:

FREE
    DMA = 2
    PORT = 300h-30Fh

This indicates that for DMA value 2, the port address 300h-30Fh is used. However, it should be noted that these two resources are not related to each other in any way.

Linked. Linked resources may contain different resource alternatives. (Resource alternatives are separated by vertical bars.) These alternatives are linked together. For example:

```
LINK
    DMA = 2 | 3
    PORT = 300h-30Fh | 400h-40Fh
```

This group allows two possible combinations of DMA and port:

- DMA value 2 and port address 300h-30Fh
- DMA value 3 and port address 400h-40Fh

Combined. Combined resources contain different resource alternatives; however, unlike the linked group (which, in the example above, only allows two possible combinations), the alternatives are linked together to form all possible combinations. For example:

```
COMBINE
    DMA = 2 | 3
    PORT = 300h-30Fh | 400h-40Fh | 500h-50Fh
```

This group allows six possible combinations:

- DMA value 2 and port address 300h-30Fh
- DMA value 3 and port address 300h-30Fh
- DMA value 2 and port address 400h-40Fh
- DMA value 3 and port address 400h-40Fh
- DMA value 2 and port address 500h-50Fh
- DMA value 3 and port address 500h-50Fh

Creating resource groups is a way to lessen the amount of choices which must be created. For example, a single choice can be created with the linked group illustrated above, rather than two choices with specified resources. To illustrate:

```
CHOICE = First Choice
        DMA = 2
        PORT = 300h-30Fh
CHOICE = Second Choice
    DMA = 3
    PORT = 400h-40Fh
```

can be written as

```
CHOICE = Single Choice
    LINK
        DMA = 2 | 3
        PORT = 300h-30Fh | 400h-40Fh
```

by using a linked group.

Using the combined group example shown above would eliminate having to create six separate CHOICE statements.

You may provide multiple values for initialization statements. For example, you may provide the following:

```
CHOICE = "Option #1"
    LINK
    IRQ = 2 | 3 | 4 | 5
    INIT = JUMPER(1) LOC(4 3 2 1) 1000 | 0100 | 0010 | 0001
```

This indicates the following corresponding combinations when the board is chosen to perform the function designated by the choice named Option #1:

- If IRQ 2 is chosen, jumper #1 is set to 1000, in descending order from jumper location 4 to jumper location 1.
- If IRQ 3 is chosen, jumper #1 is set to 0100, in descending order from jumper location 4 to jumper location 1.
- If IRQ 4 is chosen, jumper #1 is set to 0010, in descending order from jumper location 4 to jumper location 1.
- If IRQ 5 is chosen, jumper #1 is set to 0001, in descending order from jumper location 4 to jumper location 1.

This type of resource specification allows you to select the way in which the board will be used, and allows the computer to automatically select the resource allocation which will be performed.

Creating an INIT Statement

When creating INIT statements, all statements should be treated as required statements, regardless of the type of INIT (I/O port, switch, jumper, or software).

INIT statements provide the values needed to initialize programmable boards, or the settings used on switch- and jumper-programmable boards. INIT statements may also be used to display information needed to properly install software drivers. INIT statements may follow any system resource statement, including DMA, IRQ, PORT, MEMORY, and ADDRESS.

INIT Statement for Programmable I/O Ports. IOPORT(i) indicates which port is to be used. The index, "i", must be given in decimal. The variable list gives a list of values to be output to the port for each resource option. These values must be in binary. Ranges can be used.

INIT = IOPORT(i) [LOC(*bitlist*)] *list*

INIT Statement for Switches and Jumpers. INIT Statements for Switches and Jumpers are used to specify switch and jumper settings. The index, "i", in SWITCH and JUMPER statements indicates which switch or jumper block is to be used. The index must be given in decimal.

INIT = SWITCH(i) LOC(*switchlist*) *valuelist*
INIT = JUMPER(i) LOC(*jumperlist*) *valuelist*

INIT Statement for Software. INIT statements for software are used to pass parameters to the Software Identification Block for display. The index "i", indicates which SOFTWARE statement is displayed during configuration. The index must be given in decimal.

INIT = SOFTWARE(i) *list*

Verifying the Configuration File (Step 4)

The final step in the creation of a configuration file is verifying that the configuration is syntactically correct. This is accomplished by performing the following:

1. Save the configuration file that was created.
2. Exit your word processor or text editor.
3. Insert EASY CONFIG diskette #2 in drive A. Enter the following to verify that the configuration file has been created correctly:

   `A:CHECKCFG` *filename*`.CFG`

   Be sure that you indicate the drive that contains the configuration file if it is not on your current drive.

Creating Custom Configurations with Configuration File Extensions

Configuration file extensions (OVL files) are used when a configuration file alone isn't adequate for configuration; you need the computer to prompt for information, or you wish to customize certain parameters.

Configuration file extensions are able to access vendor-specified locations on the board or option to determine specific information, which is used to help make decisions to determine what the configuration should be and how specific system resources should be allocated.

When text or numeric input is required, configuration file extensions may be used to define this input; for example, you may be required to enter a password or a timeout value.

Configuration file extensions may also be used when there are too many decision paths for you to follow; you need a controlled decision process that may not be provided by using a configuration file alone.

For more information on configuration file extensions, refer to the *CFG File Extensions Programmer's Reference Guide* available from Micro Computer

Systems, Inc., 2300 Valley View Lane, Suite 800, Irving, Texas, 75062, (214) 659-1514.

An Example of Creating a Configuration File

In this example, we are creating a configuration file for the ACME Multi-function Board. An examination of the ACME Multi-function Board and its documentation reveals the following:

- The board has four distinct capabilities:
  - Serial Port
  - Parallel Port
  - Extended Memory (1 MB to 4 MB)
  - Mouse port
- The serial port has three options: COM1, COM2, and Disabled. It is an asynchronous communications port. Its connector is the top 9-pin connector.
- The parallel port has three options: LPT1, LPT2, and Disabled. Its connector is the middle 25-pin connector.
- The extended memory capability provides 1 MB to 4 MB of extended memory, added in 1 MB increments. Addressing is by 1 MB increments; addressing can begin anywhere from 1 MB to 12 MB.
- The mouse port can use IRQ 2, 3, 4, or 5, or may be disabled. It uses the lower 9-pin circular connector.
- A 6-position DIP switch is located on the board, which is used by the serial and parallel ports. Positions 1 and 2 are used by the serial port; a setting of 01 indicates COM1, 10 indicates COM2, and 00 indicates Disabled. Positions 3 and 4 are used by the parallel port; a setting of 01 indicates LPT1, 10 indicates LPT2, and 00 indicates Disabled. Positions 5 and 6 are reserved and must always be set to 00. It is set as 010100 by the factory. The switch is labeled with numbers in descending order.

- A second 6-position DIP switch is located on the board, which is used by the memory expansion capability. It is set as 000000 by the factory. The switch is labeled with numbers in descending order.
- A 4-position paired jumper is located on the board, which is used by the mouse port to indicate the IRQ value. It is numbered in ascending order. Setting location 1 (1000) indicates IRQ 2; setting location 2 (0100) indicates IRQ 3; setting location 3 (0010) indicates IRQ 4; and setting location 4 (0001) indicates IRQ 5. Setting the jumper to 0000 indicates that the mouse port is disabled. Interrupt information is determined by software through I/O port 330h.
- The board itself is a multi-function board, made by ACME Manufacturing Corporation. It is 330 millimeters long, and requires 1500 milliamps of continuous 5V current.

Using this information, the following functions and choices can be defined:

- The first function is the serial port function. It has three choices: COM1, COM2, and Disabled. Positions 1 and 2 on Switch 1 are initialized to 01 in order to select COM1; they are initialized to 10 in order to select COM2; and they are initialized to 00 in order to disable this port.
- The second function is the parallel port function. It has three choices: LPT1, LPT2, and Disabled. Positions 3 and 4 on switch 1 are initialized to 01 to select LPT1; they are initialized to 10 to select LPT2; and they are initialized to 00 to disable this port.
- The third function is the extended memory function. It has only one choice: Enable Extended Memory, with four possible memory amounts (1 MB, 2 MB, 3 MB, and 4 MB) and 12 possible starting locations (1 MB to 12 MB).
- The fourth function is the mouse port function. It has two choices: Enable or Disable the mouse port. The mouse driver reads jumper 1 (via I/O port 330h)to get the IRQ number it uses (2, 3, 4, or 5). You do not care which IRQ the mouse port uses, so these IRQ selections should not be considered choices.

In this example, the following steps would be used to create the configuration file for this board:

1. Create a Board Identification Block.

2. Create an Initialization Information Block.
   - Define Switch 1
   - Define Switch 2
   - Define Jumper 1
3. Create a Function Statement Block.
   - Create the serial port function. Create the three choices, and define the resources required for each choice.
   - Create the parallel port function. Create the three choices, and define the resources required for each choice.
   - Create the extended memory function.
   - Create the mouse port function. Define the two choices, and define the resources required for each.
4. Save the configuration file. Then, verify the configuration file with the CHECKCFG utility.

The configuration file for the above example is shown below. Note that valid values for CATEGORY, TYPE, and SUBTYPE are listed in appendix A.

```
BOARD
    ID = "ACE1010"
    NAME = "ACME Multi-function Board"
    MFR = "ACME"
    CATEGORY = "MFC"
    SLOT = ISA8
    LENGTH = 180 mm
    SKIRT = NO
    AMPERAGE = 1500

SWITCH(1) = 6
    STYPE = DIP
    NAME = "SW1"
    REVERSE = NO
    LABEL = LOC(6 5 4 3 2 1) "6" "5" "4" "3" "2" "1"
    INITVAL = LOC(6-1) 00XXXX
    FACTORY = LOC(6-1) 000101
```

```
    COMMENTS =
"This switch is used to set the serial and parallel port configurations."

SWITCH(2) = 6
    STYPE = DIP
    NAME = "SW2"
    REVERSE = NO
    LABEL = LOC(6 5 4 3 2 1) "6" "5" "4" "3" "2" "1"
    FACTORY = LOC(6-1) 000000
    COMMENTS =
"This switch is used to configure the memory allocation."

JUMPER(1) = 4
    JTYPE = PAIRED
    NAME = "IRQ"
    REVERSE = YES
    LABEL = LOC(1 2 3 4) "IRQ2" "IRQ3" "IRQ4" "IRQ5"
    FACTORY = LOC(1-4) 1000
    COMMENTS =
"This jumper is used to set the mouse port IRQ level."

FUNCTION = "Serial Port"
    TYPE = "COM,ASY"
    CONNECTION = "9-pin, male, D-type connector"
    CHOICE = "COM 1"
        SUBTYPE = "COM1"
        LINK
            IRQ = 4
            PORT = 03f8h-03ffh
            INIT = SWITCH(1) LOC(2 1) 01
    CHOICE = "COM2"
        SUBTYPE = "COM2"
        LINK
            IRQ = 3
            PORT = 02f8h-02ffh
            INIT = SWITCH(1) LOC(2 1) 10
    CHOICE = "Disabled"
        DISABLE = YES
```

```
        FREE
            INIT = SWITCH(1) LOC(2 1) 00

FUNCTION = "Parallel Port"
    TYPE = "PAR"
    CONNECTION = "25-pin, female, D-type connector"
    CHOICE = "LPT1"
        SUBTYPE = "LPT2"
        LINK
            IRQ = 7
            PORT = 378h-37fh
            INIT = SWITCH(1) LOC(4 3) 01
    CHOICE = "LPT2"
        SUBTYPE = "LPT3"
        LINK
            IRQ = 7
            PORT = 278h-27fh
            INIT = SWITCH(1) LOC(4 3) 10
    CHOICE = "Disable Parallel Port"
        DISABLE = YES
        FREE
            INIT = SWITCH(1) LOC(4 3) 00

FUNCTION = "Extended Memory"
    TYPE = "MEM"
    CHOICE = "Enable Extended Memory"
        COMBINE
            MEMORY = 1M - 12M STEP = 1M
            ADDRESS = 1M - 12M STEP = 1M
                SIZE = WORD
                CACHE = YES
            INIT = SWITCH(2) LOC(6 5 4 3 2 1) 000000-101111

FUNCTION = "Mouse Port"
    TYPE = "PTR"
    CONNECTION = "9-pin, circular connector"
    CHOICE = "Enable Mouse Port"
    FREE
```

```
    PORT = 330h
    LINK
        IRQ = 2 | 3 | 4 | 5
        INIT = JUMPER(1) LOC(1 2 3 4) 1000 | 0100 | 0010 | 0001
CHOICE = "Disable Mouse Port"
    DISABLE = YES
    FREE
        PORT = 330h
        INIT = JUMPER(1) LOC(1 2 3 4) 0000
```

An Example of Creating a Memory Board Configuration File

This section provides guidelines for creating a configuration file for a memory board. Memory board configuration is the most complicated of all configuration types.

Memory Board Configuration Cases

A memory board configuration may initialize the memory amount and starting address in two ways:

- Concurrently, in a single switch/jumper/port statement. For example:

```
COMBINE
    MEMORY = 1M | 4M
    ADDRESS = 1M - 12M STEP = 1M
    INIT = SWITCH(1) LOC(5-1) 00000-11000
LINK
    MEMORY = 1M | 4M | 16M
    ADDRESS = 1M | 1M | 16M
    INIT = SWITCH(1) LOC(2 1) 00 | 01 | 10
```

- Independently, in separate switch/jumper/port statements. For example:

```
LINK
    MEMORY = 1M | 4M
    INIT = JUMPER(1) LOC(1) 0 | 1
    ADDRESS = 1M - 12M STEP = 1M
    INIT = SWITCH(1) LOC(4-1) 0000-1100
```

The TOTALMEM Statement

The TOTALMEM statement is used to specify the total amount of memory that is being added to the system by a memory board. The TOTALMEM statement appears under the CHOICE statement; it is optional. However, if the TOTALMEM statement is not used, a hierarchy for memory selection is not established; EASY CONFIG will default to the first memory amount address that does not conflict.

This default value may be changed by using Edit Resources on EASY CONFIG's Edit pull-down menu. Ideally, for complex memory boards, configuration file extensions (OVL files) would determine the amount of memory residing on the board and would set the TOTALMEM statement accordingly. Refer to the *CFG File Extensions Programmer's Reference Guide* available from Micro Computer Systems, Inc., 2300 Valley View Lane, Suite 800, Irving, Texas, 75062, (214) 659-1514 for more information.

If a memory board is capable of adding more than one amount of memory or more than one type of memory to the computer, a TOTALMEM statement should be used. For example, a particular memory board is capable of adding either 1 MB or 4 MB. The TOTALMEM statement should be used:

```
TOTALMEM = 1M | 4M
```

A TOTALMEM statement should also be used if the memory supplied by the board is going to be split into different ranges using several MEMORY statements.

In any case, EASY CONFIG uses the specified totalmem amount to select memory amounts in all of the MEMORY statements in that choice. Virtual memory is not to be included in the totalmem amount.

EASY CONFIG will attempt to configure the function so that the total of all memory (excluding virtual memory) specified in the choice will add up to the total specified in the TOTALMEM statement. If it is unable to do this, EASY CONFIG will allocate an amount of memory less than the totalmem amount.

For example, a particular memory board supplies 1 MB - 16 MB of memory. there may already be some extended memory in the computer, and there is a 256 KB ROM at the top of the extended memory range. It is desirable to put as much memory as possible in the 1 MB - 16 MB range, with the remainder above 16 MB.

```
FUNCTION = "ROM MAPPING"
CHOICE = "ROM MAPPING"
LINK
    MEMORY = 256K
    ADDRESS = FC0000h
    MEMTYPE = OTH

FUNCTION = "MEMORY"
CHOICE = "EXTENDED MEMORY"
TOTALMEM = 1M - 16M STEP = 1M
COMBINE
    MEMORY = 0M - 15M STEP = 256K
    ADDRESS = 1M - FC0000h STEP = 256K
COMBINE
    MEMORY = 0M - 16M STEP = 256K
    ADDRESS = 16M
```

In this example, the first MEMORY statement will use as much memory as is available. If it allocates the maximum 15 MB, any remaining memory will go to the secondary MEMORY statement and will be placed at 16 MB. If the first statement uses all available memory, the second statement will be set to 0.

Using Multiple Choices with TOTALMEM Statements

There may be occasions in which more than one choice will be necessary in the memory function. If this is the case, and the TOTALMEM statement is used, each choice must have its own TOTALMEM statement. EASY CONFIG will use only the choice that is selected; it will not automatically move to the second statement (as is does in the case where the TOTALMEM statement is not used), because the TOTALMEM statement for the second choice may be different from that of the first choice. EASY CONFIG also cannot automatically change the totalmem amount.

For example, a board supplies 1 MB - 4 MB of memory that can be configured as either extended memory or expanded memory.

```
FUNCTION = "MEMORY"
CHOICE = "CONFIGURE MEMORY AS EXTENDED"
TOTALMEM = 1M - 4M STEP = 256K
COMBINE
    MEMORY = 1M - 4M STEP = 256K
    ADDRESS = 1M - 16M STEP = 256K
COMBINE
    MEMORY = 0M - 4M STEP = 256K
    ADDRESS = 16M
CHOICE = "CONFIGURE MEMORY AS EXPANDED"
FREE
    MEMORY = 1M - 4M STEP = 1M
    MEMTYPE = EXP
```

Note that the second CHOICE statement in the above example does not require a TOTALMEM statement because there is only one MEMORY statement.

Structuring MEMORY Statements

EASY CONFIG will usually attempt to allocate memory in the order that the alternatives are given in the statement. The presence of the TOTALMEM statement, however, provides the once exception to this rule.

When a TOTALMEM statement is given, EASY CONFIG always tries the highest memory amount first, then the second highest, and so on. Therefore, when a TOTALMEM statement is used, order is not important. However, in the absence of the TOTALMEM statement, the order is critical.

Sizing Gap and System Memory Contiguity

The sizing gap is used when resolving memory conflicts to insure the proper operation of memory sizing algorithms. Sizing algorithms generally require a gap between the end of system memory and the beginning of other types of writable memory to keep them from including the other non-system memory in their computation. The default sizing gap width is 64 KB. This may be increased or decreased by including a SIZING statement in the system board configuration file. Boards which require a larger sizing gap may increase it by including a SIZING statement in their configuration file. Boards cannot decrease the width of the sizing gap.

EASY CONFIG will attempt to find a memory configuration with all system memory contiguous from 0 to 640 KB and from 1 MB to 16 MB. If such a configuration is not possible, however, a non-continuous configuration will be selected. At this point, an "optimal" system memory configuration is not attempted; system memory contiguity is simply ignored.

If a configuration with contiguous system memory is found, EASY CONFIG will make sure that a sizing gap of the proper width exists after the system memory.

System Cache Map and Cache Granularity Conflicts

A system cache map can be provided in the system information block of the system board configuration file. These statements define the regions of memory that are cached by the computer, and the cache granularity for each region. If the cache map is not provided, it is assumed that the computer does not cache memory, and memory will not be marked as cached.

EASY CONFIG will not allow cache granularity conflicts. If two pieces of memory (from two different MEMORY statements) request memory in the same block of cached memory (a block being the smallest block of memory that can be turned on or off) and one MEMORY statement requires caching and the other does not, EASY CONFIG turns caching off for both MEMORY statements.

M16 Conflicts

A MEMORY statement with SIZE = WORD and DECODE = 24 states that the memory board is decoding 24 address lines and doing word transfers. This implies that the board is asserting M16 based on a decode of the address lines.

In this situation, EASY CONFIG will not allow another piece of memory (from another memory statement) to be located in the same 128 KB block of memory if it is doing byte transfers.

Memory and Address Granularity

Memory amounts are stored in nonvolatile memory (CMOS) in increments of 400h (1 KB). Memory values given in a MEMORY statement must be multiples of 1 KB.

Address amounts are stored in nonvolatile memory (CMOS) in increments of 100h (256). Address values given in an ADDRESS statement must be multiples of 256.

2

EISA Configuration Language Specification

The purpose of the EISA Board Configuration (CFG) File Language is to provide rules for creating configuration files which provide information about the option to EASY CONFIG. A configuration file includes a list of system resources used by each possible configuration of the option. System resources include interrupt request (IRQ) lines, direct memory access (DMA) channels, I/O address space, and memory address space. This document describes the configuration language and the rules that govern its use.

Language Syntax

This section describes the syntax of the language. Configuration files consist of free-form ASCII text. White space, including spaces, tabs, carriage returns, and linefeeds, is ignored outside of quoted strings. Use blank lines and indentation as necessary to increase the readability of the file.

Symbols Used In Syntax Description

The following special symbols are used in this description of the configuration file syntax:

`[ ]`	The item or statement is optional.
`x | y`	Either x or y is allowed.
value	Italicized words represent variables.
TYPE	Uppercase characters and words are keynames and must be entered as shown.
#	Spaces that you must enter are denoted by a "#" in the syntax line.

Values And Addresses

A value or address may be given in hexadecimal, decimal, or binary. The radix, or base identifier, is specified by attaching one of the following characters to the end of the value or address:

h	Hexadecimal
d	Decimal
b	Binary

Uppercase or lowercase letters are accepted. The radix character must be placed immediately after the value, with no blanks in between. If no radix is specified, decimal is assumed. For example:

1F00h

Hexadecimal numbers beginning with a letter must have a leading zero. For example:

0C000h

Slot-specific EISA port addresses must be prefixed by 0Z (zero-Z). The Z represents the 4-bit slot-specific portion of these addresses. For example:

0Z400h

Binary values are specified left to right, most significant bit to least significant bit. For example:

xxxxxxxx
76543210

A value or address may also be given in units by attaching one of the following unit abbreviations to the value. Uppercase or lowercase letters are accepted. For example:

k	Kilobytes
m	Megabytes

Units can only be used with decimal numbers. A radix cannot be specified in addition to a unit. For example:

512K

Fractions are not accepted.

A null value is indicated with empty braces: {}. Null values can be used as placeholders for DMA values, IRQ values, port values, memory values, and memory address values. Null values can also be used in INIT statements between memory and address values.

Ranges

A range is defined with a hyphen: value - value.

A range may be either ascending or descending. Ascending ranges are specified as follows: low value - high value. Descending ranges are specified as follows: high value - low value.

Blanks may or may not be used within a range. For example:

300h-30Fh or 300h # - # 30Fh

Lists

Lists consist of items such as values, addresses, and text statements that are delimited with spaces or vertical bars. Vertical bars indicate OR. For example:

"x or y" would be denoted: "x | y"

Switchlists and jumperlists are lists of switch and jumper numbers, respectively. These lists are delimited only with blanks. Vertical bars cannot be used. Ranges of switch or jumper numbers can also be used. For example:

LOC (1 # 2 # 3 # 4)

is equivalent to LOC (1-4).

Text

Many statements in the configuration language contain text fields. Text fields are a fixed length and will be truncated if they exceed this length. Information fields are free-form text and are enclosed in quotation marks. Maximum lengths are given for each instance.

To assist with text display in windows, free-form text can contain embedded tabs, denoted by \t, and embedded linefeeds, denoted by \n. Quotation marks and backslashes can be placed in the text using \ and \\ respectively.

Embedded tabs are expanded to the next tab stop. Tab length is eight characters (tab stops are located at 9, 17, 25, etc.). For example:

```
KEYWORD =
"\t This text contains \n
\t embedded tabs \n
\t and line feeds. \n"
```

White space, defined as ASCII characters 09h - 0Dh and 20h, is ignored at the beginning and end of text fields when multiple lines are required. This allows you to use white space to increase the readability of text fields in a configuration file without affecting the display of the text field. For example:

```
KEYWORD =
"This multiple line text field will be displayed without the multiple
spaces, tabs, carriage returns, and linefeeds found at the
beginning and end of each line."
```

An empty text field is indicated with double quotation marks: ""

Internal Comments

A semicolon must precede internal configuration file comments. Any text following a semicolon is ignored to the end of the line. An example of comment lines follows:

```
;
; The semicolon is used to place comments in the configuration file.
; Comments on each line must be preceded by a semicolon.
; These comments are not displayed.
;
```

Configuration File

A configuration (CFG) file contains information about a board or other option. Most boards and options require their own configuration file. The first step in developing a configuration file is to define the general characteristics of the option. These general characteristics are described as functions and choices. The functions of the board must be clearly defined, keeping in mind that the major purpose of EASY CONFIG is to resolve conflicts between DMA channels, IRQ lines, I/O port addresses, and memory requirements. The secondary purpose is to present the switch and jumper settings needed to specify those resources, and the third purpose is to present settings for other functions in order to make the presentation of the computer's configuration useful.

The key to developing a configuration file is the understanding of how functions, choices and resources interact. Boards contain functions, functions have choices, choices define the resources needed to implement the function, and the INIT statements within the choice specify the switch settings needed to select those resources.

For example, a board may specify multiple functions, such as a serial port and a parallel port. The choices for the serial port might be COM1, COM2 or Disabled, and the choices for the parallel port might be LPT1, LPT2 or Disabled. The resources for the choices would be those necessary to implement the function in the chosen manner.

A configuration file contains four types of blocks:

- Board Identification Block
- System Board Identification Block
- Initialization Information Block(s)
- Function Statement Block(s)

The Board Identification Block is used to identify and characterize the option. This block must come first in the configuration file.

The System Board Identification Block describes system board. This section is optional and can only be used in system board configuration files.

Initialization Information Blocks define the programmable ports, switches, jumpers, and software used to configure the option.

Function Statement Blocks define each function of the option. A Function Statement Block lists the different alternatives, or choices, for configuring a function. Each choice lists the system resources associated with its configuration alternative, in addition to the values needed to initialize that configuration. The initialization values can refer to any programmable I/O ports, switches, jumpers, or software defined in the Initialization Information Blocks.

Initialization Information Blocks and Function Statement Blocks may be interspersed in the configuration file. However, any Initialization Information Block referenced by a Function Statement Block must precede the Function Statement Block in which it was referenced.

BOARD IDENTIFICATION BLOCK

.
.
.

INITIALIZATION INFORMATION BLOCK

.
.
.

FUNCTION STATEMENT BLOCK
    CHOICE Statement
FUNCTION STATEMENT BLOCK
    CHOICE Statement
    CHOICE Statement

Multiple definition files may be required for the same product. This might be necessary when different revisions of the same option have different controls or switch settings to implement the functions.

Within a given block, required statements must be placed in front of those statements that are indicated as optional.

Configuration File Names

To determine a name for a configuration file, use the following format.

!*AAAPPPr*.CFG

!	A configuration file name begins with an exclamation point.
AAA	The three uppercase character manufacturer ID.
PPP	The three-digit (0-Fh) product number. Each digit can be a hexidecimal value.
r	The one-digit revision number. The digit can be a hexidecimal value.
.CFG	The file name extension (CFG).

For example:

- !ACE1234.CFG
- !XYZ5678.CFG
- !ABC0000.CFG

The file name convention is the same for a system board, other board, embedded device, or virtual device. For example, a board with a product ID of ACE0101 has a configuration file named !ACE0101.CFG.

You should ensure that the configuration file name is updated to reflect revisions to the board. For example, a product with an ID of ACE0101 may have a configuration file named !ACE0101.CFG. A subsequent revision of the product would have an ID of ACE0102. Therefore, the configuration file should be named !ACE0102.CFG. This ensures that the appropriate configuration file is loaded for the device.

The EISA implementation includes a mechanism to manage duplicate IDs, if the EISA ID scheme breaks down. If it is discovered that two entirely different boards have the same configuration file names, the capability for differentiating between the two board configuration files is provided. For example, the configuration files for two unrelated boards with ID ACE1234 installed in the same computer will be renamed when copied to the system configuration work diskette; the first configuration file detected is copied to !ACE1234.CFG, and the second configuration file is copied and renamed from !ACE1234.CFG to

1ACE1234.CFG. The next configuration file is renamed to 2ACE1234.CFG. However, this is an extremely rare case.

Board Identification Block

The Board Identification Block identifies the board and provides information about the board's physical characteristics.

BOARD
    ID = "*AAAPPPr*"
    NAME = "*text*"
    MFR = "*text*"
    CATEGORY = "*text*"
    [SLOT = *slot type* [,"*text*"]]
    [LENGTH = *value*]
    [SKIRT = YES | NO]
    [READID = YES | NO]
    [AMPERAGE = *value*]
    [BUSMASTER = *value*]
    [SIZING = *value*]
    [IOCHECK = VALID | INVALID]
    [DISABLE = SUPPORTED | UNSUPPORTED]
    [COMMENTS = "*text*"]
    [HELP = "*text*"]

BOARD

This statement appears at the beginning of each configuration file. This statement, along with the other required statements, must appear before the optional statements.

ID

This is a required statement containing the seven-character board ID. This ID is used to identify the board and to name the configuration file. The ID consists of three characters to identify the vendor (AAA), three hex characters to identify the product number (PPP), and one hex character to identify the revision number (r). The ID must contain seven characters and must be placed in quotation marks.

NAME

This is a required statement used to identify the product. Vendor and product name should be included. Revision and part numbers may also be included. A maximum length of 90 characters is allowed. Only the first 55 characters are displayed if truncation or horizontal scrolling is required.

MFR

This is a required statement used to specify the board manufacturer. A maximum length of 30 characters is allowed.

CATEGORY

This is a required statement used to identify the board by category. The board category must consist of three characters enclosed in quotation marks. You should enter the board category in uppercase letters. Some predefined categories are as follows:

COM	Communications Board
CPU	Processor board
JOY	Joystick board
KEY	Keyboard
MEM	Memory Board
MFC	Multi-function Board
MSD	Mass Storage Device
NET	Network Board
NPX	Numeric Coprocessor Board
PAR	Parallel Port Board
PTR	Pointing Device

SYS	System Board
VID	Video Adapter Board
OTH	Other
OSE	Operating System or Environment

SLOT

This is an optional statement that specifies the board's slot type. The default is ISA16.

- ISA8 indicates that the board is 8-bit ISA.
- ISA16 indicates that the board is 16-bit ISA.
- ISA8OR16 indicates that the board is capable of functioning in 8-bit or 16-bit mode. A 16-bit board that is capable of this dual-mode operation can be placed in either an 8-bit or 16-bit slot. Usually, this type of board contains a switch or jumper that must be set to indicate which mode the board is using. 8-bit boards are not capable of 16-bit operation and cannot be classified as ISA8OR16.
- EISA indicates that the board requires an EISA slot.
- OTH indicates that the board requires a non-standard, vendor-specific slot.
- VIR specifies a virtual slot.
- EMB[n] specifies an embedded slot. The slot number, n, is optional, however, an embedded slot must have either a number or a readable ID.

An optional text string of up to 10 characters can also be included with the slot type. This text string is used to match the board with slots on the system board containing the same identifier. EASY CONFIG will not allow the board to be placed in a slot which does not have a matching identifier.

LENGTH

This is an optional statement that specifies the length of the board in millimeters. A unit abbreviation should not be used. Fractions are not accepted. This value must be given in decimal. The default board length is 330 millimeters. If omitted, board length will not be used as a criteria for placement within the computer.

SKIRT

This is an optional statement that specifies whether or not the board has a lower extension that prevents it from fitting into larger size slots. The default is NO.

READID

This is an optional statement that specifies whether or not the board has a readable ID. The default is NO.

AMPERAGE

This is an optional statement that specifies the amount of 5V current used by the board. This value is given in milliamps. A unit abbreviation should not be used. Fractions are not accepted. If omitted, power usage is not used to determine whether a board can be added to the computer. This value must be given in decimal.

BUSMASTER

This is an optional statement that specifies a bus master board's maximum acceptable latency in microseconds. This value must be given in decimal. If omitted, it is assumed that the board is not a busmaster board. There is no default value.

SIZING

This is an optional statement that specifies the minimum gap in memory required for a sizing algorithm to produce correct results. The default is 64K. This statement can be used on a system board to reduce the default size. Other boards can only increase the size. The largest value of all incidences is used as the sizing gap.

IOCHECK

This is an optional statement that specifies whether an EISA board's iocheck bit can be considered valid when an iocheck occurs. Valid values for this statement are VALID and INVALID. The default is VALID.

DISABLE

This is an optional statement that specifies whether the board supports the EISA disable board feature. Valid values for this statement are SUPPORTED and UNSUPPORTED. The default is SUPPORTED.

COMMENTS

This is an optional text field containing additional information about the board. This text field may contain a maximum of 600 characters and will be displayed in a window at least 40 columns wide when the board is selected.

HELP

This is an optional text field containing information that will be displayed if help is requested while configuring the board. This text field may contain a maximum of 600 characters and will be displayed in a window at least 40 columns wide.

System Board Identification Block

The system board is identified in the board identification block as embedded slot #0, using the statement:

SLOT = EMB(0) CATEGORY = SYS

System board configuration files must supply additional information to EASY CONFIG. This information includes the amount of nonvolatile memory available, the number of slots on the system board, and the size and type of each slot. Also included is a memory cache granularity map. This information is provided in the System Board Identification Block, which must be placed immediately following the Board Identification Block.

```
SYSTEM
    [NONVOLATILE = value]
    [AMPERAGE = value]
    SLOT(1) = slot type [,"text"] [,"text"] ... ]
        [LENGTH = value]
        [SKIRT = YES | NO]
        [BUSMASTER = YES | NO]
    .
    .
    SLOT(i) = slot type [,"text"] [,"text"] ... ]
        [LENGTH = value]
        [SKIRT = YES | NO]
        [BUSMASTER = YES | NO]
    [MEMORY = value ADDRESS = value CACHE = YES
    STEP = value]
    .
    .
    [MEMORY = value ADDRESS = value CACHE = NO]
```

SYSTEM

This statement appears at the beginning of the System Board Identification Block.

NONVOLATILE

This is an optional statement listing the amount of nonvolatile memory available for EISA slot- specific use. The value may be specified in decimal, hex, or decimal with a unit abbreviation.

AMPERAGE

This is an optional statement that specifies the amount of 5V current available to boards. This value is specified in milliamps. A unit abbreviation is not used.

SLOT

This statement identifies the number and type of slots available on the system board. The SLOT statements specify the slot types as ISA8 for 8-bit ISA, ISA16 for 16- bit ISA, EISA for EISA boards in addition to 8- and 16-bit ISA boards, or OTHER for manufacturer-specific boards.

One or more optional text strings (10 characters maximum) may be given after each slot type. These text strings are compared to a board's slot text to determine whether the board can be place in the slot. Boards with a matching identifier or a matching slot type will be allowed in the slot. For example, a slot defined as EISA, "ACEMEM" will accept a board defined as EISA or EISA, "ACEMEM."

LENGTH

This is an optional statement that specifies the length of the board in millimeters. The length must be given in decimal. Fractions are not accepted. If omitted, board length will not be used as a criteria for placement within the computer. The default length is 341. For example:

LENGTH = 341 (for a full length slot)

SKIRT

This optional statement specifies whether or not each slot on the system board can accommodate a board that contains a lower extension, or skirt. The default is YES, boards with skirts can be accommodated.

BUSMASTER

This is an optional statement that specifies whether or not each slot on the system board is capable of supporting a bus master. The default is NO for ISA slots and YES for EISA slots.

MEMORY

This optional statement gives the size of the range being specified.

ADDRESS

This optional statement gives the starting address of the range being specified.

CACHE

This optional statement specifies whether this range can be cached by the computer. The STEP parameter specifies the cache granularity within range. STEP should only be given when CACHE = YES is specified.

Initialization Information Block

Initialization Information Blocks are used to identify methods used to configure a board. A board may use any or all of these methods. A configuration file may contain several of each type of these blocks. Each type of Initialization Information Block contains an index used to differentiate it from others of the same type. These indices will be used by INIT statements, discussed later, to reference the blocks. Each type of block should be numbered independently of the other types.

There are four types of Initialization Information Blocks:

- Programmable I/O Port Identification Blocks
- Switch Identification Blocks
- Jumper Identification Blocks
- Software Identification Blocks

Programmable I/O Port Identification Block

Programmable I/O Port Identification Blocks are used to identify ports required to configure a board. A Programmable I/O Port Identification Block is needed for each port addressed. A shorthand form of the INIT statement for I/O ports is provided for boards that must initialize a large number of ports. This form does not require a corresponding IOPORT statement. For more information on the shorthand PORT statement, refer to "Shorthand PORT Statement" in this chapter.

The order in which programmable I/O ports are initialized is defined by the order in which Programmable I/O Port Identification Blocks are placed in the configuration file.

IOPORT(i) = *address*
    [SIZE = BYTE | WORD | DWORD]
    [INITVAL = [LOC (*bitlist*)] *value*]

IOPORT(i) = PORTVAR(*n*)
    [SIZE = BYTE | WORD | DWORD]
    [INITVAL = [LOC (*bitlist*)] *value*]

SIZE

This is an optional statement that gives the addressability of the port. Valid values for this statement are BYTE, WORD, and DWORD. The default is BYTE.

INITVAL

This is an optional statement used to specify the source of a port's bit values. The value is binary only: it is not necessary to specify the radix. A bit position marked with a 1 or 0 indicates that the bit is reserved and should be set to

that value. A bit position marked with an "x" indicates that the bit will be set by EASY CONFIG. A bit position marked with an "r" indicates that the bit value should be read from the port. Each of the port's bits must be defined by one of these characters. If this statement is omitted, all bit values that are not set by EASY CONFIG will be read from the port.

The LOC clause can be used in the INITVAL statement to specify the bits being defined. These bits are specified in descending order from most to least significant. If the LOC clause is not used, the INITVAL statement must contain 8, 16, or 32 bits, depending on the stated size of the port.

The following is an example of a valid INITVAL statement:

INITVAL = rrxx0101

Multiple I/O Port Initialization Blocks

Sometimes a board requires a single port to receive multiple initialization values in sequence. This can be accomplished by defining the same port multiple times. A separate I/O Port Initialization Block must be included for the port for each sequential initialization value it needs to receive. These separate IOPORT statements must have different indices so that they can be individually referenced. During power-up, the initialization values are written to the ports in the order in which they are defined in the configuration file. In this way, the single port can receive the sequence of initialization values it requires.

In the statement, IOPORT(i) = *address*, "i" is an index used to differentiate this Programmable I/O Port Identification Block from others. *Address* is the address of the programmable port. Slot-specific EISA port addresses must be prefixed with 0Z.

In the statement, IOPORT(i) = PORTVAR(n), i is the programmable port index and PORTVAR(n) indicates that the address of the port will be defined at a later time. There must be a statement in a Choice Statement Block that gives the address for PORTVAR(n). The index for the PORTVAR statement (n) is independent of the IOPORT index. Refer to "Choice Statement Block" in this chapter for more information on the PORTVAR statement.

Switch Identification Block

Switch Identification Blocks are used to identify each set of switches on a board. A Switch Identification Block is needed for each set of switches on the board.

SWITCH(i) = *n*
    NAME = "*text*"
    STYPE = DIP | ROTARY | SLIDE
    [VERTICAL = YES | NO]
    [REVERSE = YES | NO]
    [LABEL = LOC(*switchlist) list*]
    [INITVAL = LOC(*switchlist) list*]
    [FACTORY = LOC(*switchlist) list*]
    [COMMENTS="*text*"]
    [HELP="*text*"]

n

This variable tells the number of switches. For DIP switches, this value is the number of individual switches in the set. For rotary and slide switches, this value gives the number of positions on the switch. The maximum value is 16.

NAME

This is a required statement giving either the manufacturer's name or a description for this switch. This is important as a reference. A maximum length of 20 characters is allowed for the NAME statement.

STYPE

This is a required statement used to indicate the type of switch. A DIP switch is defined as a set of switches, each having an ON and OFF position. A ROTARY switch is defined as a round switch that may be set to one of *n* positions. A SLIDE switch is defined as a straight line switch that may be set to one of n positions. All switch positions are numbered beginning with 1.

VERTICAL

This is an optional statement that indicates whether or not the switch is positioned vertically on the board. The default is NO. Refer to "Switch Examples" in this section for more information.

REVERSE

Switches are usually numbered in descending order from left to right or from top to bottom. This is an optional statement that indicates that the switch is numbered in ascending order from left to right or from top to bottom. Switches numbered in ascending order from left to right or top to bottom are considered to be reversed. The default is NO, for not reversed. Refer to "Switch Examples" in this section for more information.

LABEL

This is an optional list of text fields used to specify labels for individual positions on a switch. If the LABEL statement is omitted, the default numbering is ..4321 for normal switches and 1234.. for REVERSE switches. If the switch positions are labeled differently than the default numbering, use the LABEL statement to assign the actual labels to the corresponding positions on the switch. These switch positions are listed in the LOC(*switchlist*) clause. A maximum length of 10 characters is allowed for each text field. In the example below, the switch positions for that switch are displayed as "SW1," "SW2," and "SW3." For example:

LABEL = LOC(3-1) "SW1" "SW2" "SW3"

INITVAL

This is an optional statement used to specify the values of reserved switch positions. The LOC(*switchlist*) clause specifies the switch positions being defined and list gives the values of these positions. If omitted, all switch positions are assumed to be either determined by EASY CONFIG or not used. For example:

INITVAL = LOC(2 # 1) 10

FACTORY

This is an optional statement indicating the default value of the switch positions when the board was shipped. The LOC(switchlist) clause specifies the switch positions being defined and list gives the values of these positions. In the example below, all eight switches are set to ON. For example:

FACTORY = LOC(8-1) 11111111

COMMENTS

This is an optional text field containing additional information that will assist in configuring this particular switch. This field may contain a maximum of 600 characters and will be displayed in a window at least 40 columns wide.

HELP

This is an optional text field containing information that will be displayed if help is requested while configuring the switch. This text field may contain a maximum of 600 characters and will be displayed in a window at least 40 columns wide.

Hints

In the statement, SWITCH(i) = *n*, "i" is an index used to differentiate this Switch Identification Block from others. SWITCH statements should be numbered beginning with 1.

For DIP switches, *n* gives the number of individual switches in the set. For rotary and slide switches, *n* gives the number of positions on the switch. The maximum value for *n* is 16 for all switch types. The values "i" and *n* must be given in decimal.

Several statements in the Switch Identification Block use the LOC(*switchlist*) clause to specify the location of individual switches within the block. A *switchlist* is defined as a list of switch numbers, delimited with blanks. This list may be either ascending or descending, but not mixed. For example:

LOC(1 # 3 # 5)

The following mixed list is an example of *illegal* syntax.

LOC(1 # 5 # 3)

A range may also be used in place of *switchlist.*

LOC(4-1) or LOC(1-4).

Switches that are defined as reversed (REVERSE=YES) must use ascending position numbers in their LOC clauses. Switches that are defined as not reversed (REVERSE=NO) must use descending position numbers in their LOC clauses.

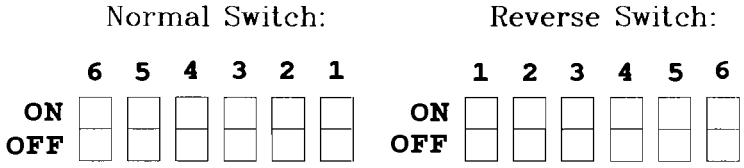
A DIP switch value can be 0, indicating OFF, 1, indicating ON, or "x", indicating that the switch position is not used. A slide or rotary switch value consists of the position number to which the switch is set. Switch values do not require radix identifiers: binary is assumed for all switches.

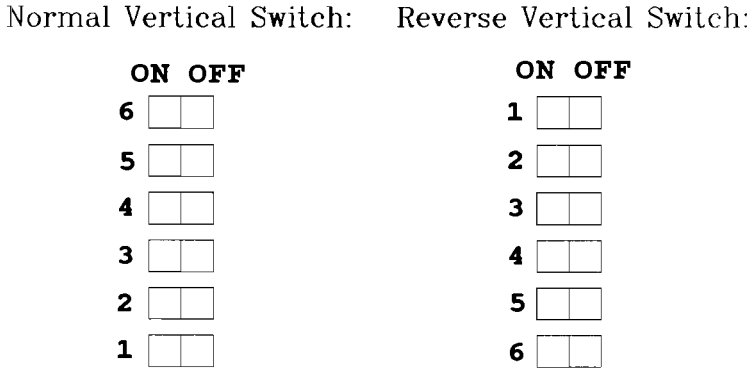
A list of DIP switch values may not contain embedded spaces. For example:

LOC(6-1) xx1101

Switch Examples

The following examples assume the following board orientation: edge connector down, board bracket on the right.





Jumper Identification Block

Jumper Identification Blocks are used to identify sets of jumpers on a board. A Jumper Identification Block is needed for each set of jumpers on the board.

JUMPER(i) = *n*
    NAME = "*text*"
    JTYPE = INLINE | PAIRED | TRIPOLE
    [VERTICAL = YES | NO]
    [REVERSE = YES | NO]
    [LABEL = LOC(*jumperlist) list*]
    [INITVAL = LOC(*jumperlist) list*]
    [FACTORY = LOC(*jumperlist) list*]
    [COMMENTS="*text*"]
    [HELP="*text*"]

n

This variable gives the number of jumper positions in the set. For paired jumpers, each pair is one position. For inline jumpers, each connection is one position. For example, two posts make one connection and three posts make two connections. For tripole jumpers, each tripole set is one position. The maximum value is 16.

NAME

This is a required statement giving either the manufacturer's name or a description for this jumper set. This is important as a reference. A maximum length of 20 characters is allowed for the NAME statement.

JTYPE

This is a required statement used to indicate the type of jumper. INLINE jumpers are arranged in a line, to be connected from one post to the next. PAIRED jumpers are arranged as a series of double posts, to be connected across. TRIPOLE jumpers are arranged as a series of triple posts, so that each jumper has two possible settings. Examples of each type of jumper are given in "Jumper Examples" in this section.

VERTICAL

This is an optional statement that indicates whether or not the jumper block is positioned vertically on the board. The default is NO. Refer to "Jumper Examples" in this section for more information.

REVERSE

Jumpers are usually numbered in descending order from left to right or from top to bottom. This is an optional statement that indicates that the jumper is numbered in ascending order from left to right or from top to bottom. Jumpers numbered in ascending order from left to right or top to bottom are considered to be reversed. The default is NO. Refer to "Jumper Examples" in this section for more information.

LABEL

This is an optional list of text fields used to specify labels for individual jumpers. If the LABEL statement is omitted, the default numbering is ..4321 for normal jumpers and 1234.. for REVERSE jumpers. If the jumpers are labeled differently than the default numbering, use the LABEL statement to assign the actual labels to the corresponding jumpers. These jumpers are listed in the LOC(*jumperlist*) clause. These labels may also be placed between two inline jumpers, numbered i and j, by specifying the jumper location as i^j. A maximum length of 10 characters is allowed. In the example below, the jumpers for that board are displayed as "J1," "J2," and "J3."

LABEL = LOC(3-1) "J1" "J2" "J3"

INITVAL

This is an optional statement used to specify the settings of reserved jumper positions in a set. The LOC(*jumperlist*) clause specifies the jumper positions being defined and *list* gives the settings of these positions. If omitted, all jumper settings are assumed to be either determined by EASY CONFIG or not used. In the example below, on an inline jumper, poles 4 and 3 are connected and poles 2 and 1 are connected.

INITVAL = LOC(4^3 # 2^1) 11

FACTORY

This is an optional statement indicating the default settings of the jumpers when the board was shipped. LOC(*jumperlist*) specifies the jumper positions being defined and *list* gives the settings of these positions. For example:

FACTORY = LOC(6-1) 110000

COMMENTS

This is an optional text field containing additional information that will assist in configuring this particular jumper set. This field may contain a maximum of 600 characters and will be displayed in a window at least 40 columns wide.

HELP

This is an optional text field containing information that will be displayed if help is requested while configuring the jumper. This text field may contain a maximum of 600 characters and will be displayed in a window at least 40 columns wide.

Hints

In the statement, JUMPER(i) = *n*, "i" is an index used to differentiate one Jumper Identification Block from another. JUMPER statements should be numbered beginning with 1. The variable *n* gives the number of jumper positions in the set. The maximum value for *n* is 16. The values "i" and *n* must be given in decimal.

Several statements in the Jumper Identification Block use the LOC(*jumperlist*) clause to specify the location of individual jumpers within the block. A *jumperlist* is defined as a list of jumper numbers, delimited with blanks. This list may be either ascending or descending, but not mixed. For example:

LOC(1 # 3 # 5)

The following mixed list is an example of *illegal* syntax.

LOC(1 # 5 # 3)

A range may also be used in place of *jumperlist*.

LOC(4-1) or LOC(1-4)

Jumpers that are defined as reversed (REVERSE=YES) must use ascending position numbers in their LOC clauses. Jumpers that are defined as not reversed (REVERSE=NO) must use descending position numbers in their LOC clauses.

Paired and tripole jumpers are specified by their numbers, while inline jumpers are specified by the two posts that they connect. A caret is used to show this connection. For example:

LOC(2^1)

Ranges may not be used for inline jumper *jumperlists.*

A list of jumper values may not contain embedded spaces. For example:

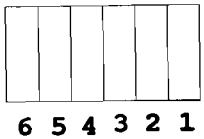
LOC(3-1) 101

Jumper values differ for each type of jumper and are explained in "Jumper Examples" in this section.

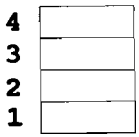
Jumper Examples

Values for paired jumpers can be given in the same way that they are given for switches, with a 1 or 0 for each position. A 1 indicates that the jumper is present; a 0 indicates it is not.

Paired Jumper:



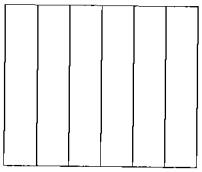
Vertical Paired Jumper:



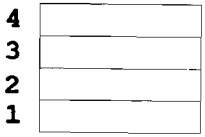
Values for tripole jumpers can also be expressed in the standard way, with some clarification of the meaning of 1 and 0. A 1 indicates that the jumper is in the upper position for horizontal jumpers or the right position for vertical jumpers. A 0 indicates that the jumper is in the lower position for horizontal

jumpers or the left position for vertical jumpers. An uppercase or lowercase N indicates that the jumper is not present.

Tripole Jumper:

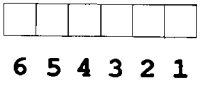


Vertical Tripole Jumper:

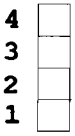


The standard notation is inadequate for describing inline jumpers. These jumper positions are represented using carets. For example, the first jumper position in an inline jumper block would be represented by 1^2. A 1 indicates that the jumper is present, a 0 indicates that the jumper is not present.

Inline Jumper:



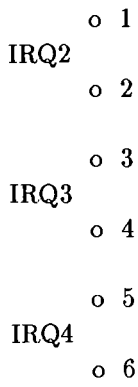
Vertical Inline Jumper:



Jumper Identification Block Example

```
JUMPER(1) = 5
    NAME = "J101"
    JTYPE = INLINE
    VERTICAL = YES
    REVERSE=YES
    LABEL = LOC(1^2 # 3^4 # 5^6) "IRQ2" "IRQ3" "IRQ4"
    FACTORY = LOC(3^4) 1
```

The example above would appear on the screen as follows. Note that the number of jumper positions (5) is not equal to the number of posts.



Software Identification Block

Software Identification Blocks are used to identify software programs that are used to initialize the board. They will display a set of instructions for adding software statements to existing drivers or programs that are executed in order to use this board or option. These drivers or programs are typically those placed in the CONFIG.SYS or AUTOEXEC.BAT files.

SOFTWARE(i) = "*information*"

In the SOFTWARE(i) statement, i is an index used to differentiate this Software Identification Block from others. The *information* may contain a maximum of 600 characters and will be displayed in a window 40 columns wide.

At configuration time, the software description comments are displayed. Often, these comments will simply inform you of a device driver that must be called from the CONFIG.SYS file, or a program that must be run under MS-DOS) to initialize a board. On ISA computers, these programs often require parameters to inform them which interrupt, DMA channel, or other global resource was selected for their board. By placing an INIT = SOFTWARE(i) statement after the resource declaration, the parameter associated with the selected resource

may be appended to the software description comment and displayed. Refer to the section "INIT Statements" in this chapter for an example of this usage.

Software drivers on EISA computers will be able to determine their resource parameters by querying EISA nonvolatile memory using a ROM BIOS call. For these programs, this feature will not be necessary.

Function Statement Block

A Function Statement Block is used to identify each distinct function of a board. Functions are what the board supplies to the computer. Parallel printer ports, serial ports, disk controllers, network interfaces, memory, and video control are some of the common functions provided by boards.

For example, most network boards have only one function. Some boards, such as serial or parallel boards, have more than one function. A separate Function Statement Block is required for each function of the board. The Function Statement Block lists all possible configurations for a function of the board, and lists the resources used for each of these configurations.

All functions of the board are considered when the board is configured by EASY CONFIG. At least one FUNCTION statement must be defined for a board.

Either choices or subfunctions can follow a Function Statement Block, but not both. Subfunctions are described in the "Subfunctions" section in this chapter. The Choice Statement Block is described in the "Choice Statement Block" section in this chapter.

NOTE: The total number of characters allowed for grouptypes, function types, subfunction types, and choice subtypes is 80 characters including separators.

FUNCTION = "*text*"
[TYPE = "*function type*"]
[CONNECTION = "*text*"]
[COMMENTS = "*information*"]
[HELP = "*text*"]
Choice Statement Block or Subfunction
.
.
[Choice Statement Block or Subfunction]

FUNCTION

This statement names a function of the board. A maximum length of 100 characters is allowed; however, 40 characters is the preferred length.

TYPE

This is an optional statement used by a board manufacturer to identify a function or device on a board. This information is available to software drivers through a ROM BIOS call. Characters in the TYPE string must be in uppercase letters, and the TYPE string must be placed in quotation marks. A list of valid types is included in appendix A.

CONNECTION

This is an optional text field that specifies the orientation and description of a board's external connectors. A maximum length of 40 characters is allowed.

COMMENTS

This is an optional text field containing additional information about the function. This text field may contain a maximum of 600 characters and will be displayed in a window at least 40 columns wide when the function is selected.

HELP

This is an optional text field containing information that will be displayed if help is requested while configuring the function. This text field may contain a maximum of 600 characters and will be displayed in a window at least 40 columns wide.

Groups of Functions

Function Statement Blocks can be grouped together using the GROUP and ENDGROUP statements:

```
GROUP = "text"
    [TYPE = "group type"]
     FUNCTION Block
     .
     .
    [FUNCTION Block]
END GROUP
```

Grouping Function Statement Blocks is an advanced feature of the configuration language that allows better presentation and organization of types and subtypes.

A group of Function Statement Blocks must contain at least one FUNCTION statement. Each FUNCTION statement in the group has one entry in CMOS, and the group's type string is appended to each of the function's type strings. The group's name is used for display purposes and can have a maximum length of 100 characters.

Subfunctions

Ordinarily, FUNCTION statements are used to describe a single device or function of a board. The different alternatives for configuring the function are then listed in subsequent CHOICE statements. There may be cases in which a single function of the board can be broken down into smaller pieces, each of which may be configured separately. In such cases, you should be able to view and modify the configurations of each of these pieces individually, even though they were presented as a single function.

Individually configurable pieces of a single function can be grouped together under a FUNCTION statement using SUBFUNCTION statements. Each SUBFUNCTION statement has all of the components of a FUNCTION statement.

```
FUNCTION = "text"
    [TYPE = "function type"]
    [COMMENTS = "text"]
    [CONNECTION = "text"]
    [HELP = "text"]
    SUBFUNCTION = "text"
        [TYPE = "function type"]
        [COMMENTS = "text"]
        [CONNECTION = "text"]
        [HELP = "text"]
        Choice Statement Block
        .
        .
        [Choice Statement Block]
    .
    .
    SUBFUNCTION = "text"
        [TYPE = "function type"]
        [COMMENTS = "text"]
        [CONNECTION = "text"]
        [HELP = "text"]
        Choice Statement Block
```

A FUNCTION statement can have its own TYPE, COMMENTS, CONNECTION and HELP statements, in addition to those of the SUBFUNCTION statements. However, a FUNCTION statement may not have Choice Statement Blocks followed by SUBFUNCTION statements.

Choice Statement Block

Choice Statement Blocks are used to place different RESOURCE statements and their corresponding initialization information into logical groups. Each FUNCTION must have at least one CHOICE statement (unless SUBFUNCTIONS are used) and may have as many as necessary. A group of related resource and initialization statements is called a resource/init group and is described later in this chapter. Each of a function's choices has a different resource/initialization group. For example, if the function was "Serial Port," its choices might be "COM1," "COM2," and "Disable Port." Each choice requires a different set of resources and different initialization values. A maximum length of 90 characters is allowed for the CHOICE name, however, 30 characters is the best length for display purposes.

CHOICE = "*text*"
    [SUBTYPE = "*choice type*"]
    [DISABLE = YES | NO]
    [AMPERAGE = *value*]
    [PORTVAR(n) = *address*]
    [TOTALMEM = *list* | *range* STEP = *value*]
    [HELP = "*text*"]
    resource/init group or Subchoice
    .
    .
    [resource/init group or Subchoice]

SUBTYPE

This is an optional statement used in conjunction with the FUNCTION level TYPE statement. The purpose of this statement is to provide information to software drivers through a ROM BIOS call. Characters in the SUBTYPE string must be in uppercase letters, and the SUBTYPE string must be placed in quotation marks.

DISABLE

This is an optional statement used to inform EASY CONFIG that, if the corresponding CHOICE is selected, the function or device will be disabled. When resource conflicts occur, EASY CONFIG will select this choice only as a last resort. The default is NO.

AMPERAGE

This is an optional statement, used to specify the maximum amount of continuous 5V current (in milliamps) required by the option specified by the Choice Statement Block. This value must be given in decimal and is added to the current requirements stated in the Board Identification Block.

PORTVAR

This is an optional statement that specifies the address of a related Programmable I/O Port Initialization Block. This statement and the corresponding IOPORT(i)=PORTVAR(n) statement are linked by the index, n, which must be given in decimal.

TOTALMEM

This is an optional statement used to specify the total amount of memory on a memory board. The TOTALMEM statement allows EASY CONFIG to verify that the total amount of memory selected in multiple MEMORY statements does not exceed the amount of memory installed on the board. This statement also allows EASY CONFIG to select the proper SUBCHOICE, when SUBCHOICES are required.

For a given CHOICE, all of the potential memory quantities for the board must be given in the TOTALMEM statement. Each of these values can be listed, or a range can be given. If a range is given, the STEP statement is required to give the smallest increment by which memory can be added to the board. The memory amounts and step value can be given in decimal, hex or decimal with unit abbreviations. For example:

TOTALMEM = 1024K | 1536K | 2048K
TOTALMEM = 1M - 2M STEP = 512K

The above statements are equivalent.

The TOTALMEM statement should encompass all system and expanded memory provided by the board (MEMTYPE = SYS or EXP). Memory that is classified as VIRTUAL is not included in the TOTALMEM statement.

HELP

This is an optional text field containing information that will be displayed if help is requested for the CHOICE. This text field may contain a maximum of 600 characters and will be displayed in a window 40 columns wide.

Resource/Init Groups

RESOURCE statements and INIT statements are placed into groups within a Choice Statement Block. Each Choice Statement Block may have one or more of these groups. A resource/init group is defined as a set of resource and INIT statements that are all directly related. RESOURCE statements that are initialized independently of each other would fall into separate groups.

The start of a grouping is marked by a keyword indicating the group type. The group ends when another group or description block is encountered.

There are three types of groups in which RESOURCE and INIT statements can be placed: linked, combined, and free-form. These groups are discussed separately.

All RESOURCE and INIT statements must be defined within a group.

Linked Groupings

In a linked group, each statement must have the same number of options. These options have a one-to-one correspondence with options in the other statements. If the first option is chosen in one statement, then by definition the first option is chosen in each of the other statements. Linked groups have the following syntax:

```
LINK
    resource and/or INIT statements
```

For example:

```
LINK
    IRQ = 3 | 4
    DMA = 2 | 5
    INIT = IOPORT(1) LOC (7 6) 10 | 01
```

In the above example, the first initialization value is used if IRQ 3 and DMA 2 are chosen, and the second is used if IRQ 4 and DMA 5 are chosen.

Combined Groupings

In a combined group, every combination of options encompassing all RESOURCE statements are provided with an initialization value in the group INIT statement. This type of group is used primarily with MEMORY and ADDRESS statements. Combined groups have the following syntax:

```
COMBINE
    resource and/or INIT statements
```

For example:

```
COMBINE
    MEMORY = 1M | 2M
    ADDRESS = 1M | 2M
    INIT = IOPORT(2) LOC (5 4) 00 | 01  | 10 | 11
```

In the above example, the first initialization value is used for 1 MB of memory starting at 1 MB. The second is used for 1 MB of memory starting at 2 MB. The third is used for 2 MB of memory starting at 1 MB. Finally, the fourth initialization value is used for 2 MB of memory starting at the address 2 MB.

All INIT statements for a combined group should have a number of values equal to the product of all of the options in the preceding RESOURCE statements. The combinations always proceed with the first RESOURCE statement taking the most significant position and the last RESOURCE statement taking the least significant position.

Free-Form Groupings

In a free-form group, all RESOURCE and INIT statements are considered to be independent of each other. No correspondence between the RESOURCE and INIT statements is required. Free-form groups have the following syntax:

```
FREE
    resource and/or INIT statements
```

For example:

```
FREE
    IRQ = 2 | 3 | 4 | 5
    DMA = 4
    INIT = IOPORT(2) LOC(31)1010
```

In the above example, IRQ 2, 3, 4, or 5 may be selected, with no initialization required. It is assumed that any initialization for this statement will be done later by an independent program. This example also states that DMA channel 4 will be used, and that the initialization given in the INIT statement should always be performed. Each of these statements is independent and completely unrelated to the others.

Resource Statements

RESOURCE statements are used to enumerate the system resources used by the configuration are defined in a Choice Statement Block. The global system resources are DMA channels, I/O port address space, IRQ levels, and memory address space.

DMA Statement

DMA statements are used to define DMA channels. This statement is required only by functions that use DMA channels. DMA values can be specified in decimal or hex.

The following example indicates that DMA channel 2 or 4 can be used. The selected channel cannot be shared, and default timing will be used.

```
DMA = list
    [SHARE = YES | NO | "text"]
    [SIZE = BYTE | WORD | DWORD]
    [TIMING = DEFAULT | TYPEA | TYPEB | TYPEC]
```

For example:

```
DMA = 2 | 4
```

SHARE. This is an optional statement that indicates whether the function is willing to share this DMA channel. The default is NO. A text field identifier can also be specified in the SHARE statement to indicate that the function can only share the DMA channel with a device that has a matching identifier. A maximum length of 10 characters may be used for the identifier.

SIZE. This is an optional statement that specifies whether the DMA channel uses 8-bit, 16-bit, or 32-bit data transfer. Valid values are BYTE, WORD, and DWORD. The default values are BYTE for DMA channels 0-3 and WORD for DMA channels 4-7. When provided, this information will be used for verification purposes only.

TIMING. This is an optional statement that specifies the timing used by the DMA channel. If omitted, DEFAULT timing is assumed.

IRQ Statement

The IRQ statement is used to define IRQ addresses. This statement is required only by functions that use IRQ addresses. IRQ values may be specified in decimal or hex.

The following example indicates that IRQ 3, 4, or 5 may be used as a shared, level triggered, interrupt.

IRQ = *list*
    [SHARE = YES | NO | "*text*"]
    [TRIGGER = LEVEL | EDGE]

For example:

```
IRQ = 3 | 4 | 5
    SHARE = YES
    TRIGGER = LEVEL
```

SHARE. This is an optional statement that indicates whether the function is willing to share this interrupt. The default is NO. A text field identifier can also be specified in the SHARE statement to indicate that the function can only share the DMA channel with a device that has a matching identifier. A maximum length of 10 characters may be used for the identifier.

TRIGGER. This is an optional statement that specifies whether the interrupt is level or edge triggered. Valid values are EDGE and LEVEL. The default is EDGE.

PORT Statement

A PORT statement is used to identify I/O ports. This statement is only required by functions that use I/O ports. PORT addresses must be given in hex. STEP and COUNT values can be given in decimal or hex.

PORT = *list* | *range* STEP = *value* [COUNT = *value*]
[SHARE = YES | NO | "*text*"]
[SIZE = BYTE | WORD | DWORD]

SHARE. This is an optional statement that indicates whether the function is willing to share the requested ports. The default is NO. A text field identifier can also be specified in the SHARE statement to indicate that the function can only share the DMA channel with a device that has a matching identifier. A maximum length of 10 characters may be used for the identifier.

SIZE. This is an optional statement that is used to indicate the size of the port. Valid values are BYTE, WORD, and DWORD. There is no default value. When provided, this information will be used for verification purposes only. For example:

SIZE = BYTE

Hints

If a list is given in the PORT statement, the elements of the list can be ranges or single port addresses. The STEP and COUNT statements may not be used with a list of ranges. In this case, the entire range will be allocated.

If a single range is given in the PORT statement, the STEP statement is required to indicate the number of ports requested and the increment used in searching for available I/O space. If the number of ports requested is different from the increment value, the COUNT statement may be used to supply the number of ports.

Slot-specific EISA port addresses must be prefixed with 0Z.

The COUNT statement may not be used without the STEP statement. Also, the COUNT statement must follow the STEP statement. If the STEP and COUNT statements are not supplied it is assumed that every port in the range is being requested. Some examples of PORT statements are shown below.

The following example indicates that the entire range of 16 port addresses from 300h to 30Fh will be used.

PORT = 300h - 30Fh

The following example indicates that any of the following ranges may be used: 300h-303h, 304h-307h, 308h-30Bh, 30Ch-30Fh.

PORT = 300h - 30Fh STEP = 4

The following example is acceptable, although it calls for 17 ports taken 4 at a time. The extra port address is ignored, and the example becomes identical to the previous one.

PORT = 300h - 310h STEP = 4

The following example indicates that any of the following ranges may be used: 300h-301h, 304h-305h, 308h-309h, 30Ch-30Dh.

PORT = 300h - 30Fh STEP = 4 COUNT = 2

MEMORY and ADDRESS Statements

MEMORY statements are used to define memory addresses. This statement is required only by functions that use memory addresses. If a linked, combined, or free-form resource group contains a MEMORY statement, it can contain no other RESOURCE statements. Memory amounts, addresses and STEP values may be given in decimal, hex, or decimal with a unit abbreviation.

MEMORY = *list* | *range* STEP = *value*
    [INIT Statements]
[ADDRESS = *list* | *range* STEP = *value*]
    [WRITABLE = YES | NO]
    [MEMTYPE = SYS | EXP | OTH | VIR]
    [SIZE = BYTE | WORD | DWORD]
    [DECODE = 20 | 24 | 32]
    [CACHE = YES | NO]
    [SHARE = YES | NO | "*text*"]

MEMORY. This statement is used to define a range of memory on the board. Memory boards can usually contain several different amounts of memory. Each of these values can be listed or a range can be given. If a range is given, the STEP statement is required to give the smallest increment by which memory can be added to the board.

INIT. If the amount of memory is initialized independently of its address, one or more INIT statements can be placed after the MEMORY statement. Refer to the "INIT Statement" section in this chapter for more information.

ADDRESS. This statement specifies the starting address of the memory. If a single address is given, the memory must be located there. If a list or range of addresses is given, the memory may be located at any of those addresses. If an address range is given, the STEP statement is required to give the increment used in searching for an available slot. The ADDRESS statement should only be used for memory that resides in the computer's physical address space. The ADDRESS statement may not be used with expanded memory.

WRITABLE. This is an optional statement that indicates whether you can write to the memory. The default is YES.

MEMTYPE. This is an optional statement that specifies whether the memory is system, expanded, other, or virtual. System memory includes memory in the physical address space that is managed by the operating system. Expanded memory is memory that does not use physical address space and is managed by an expanded memory manager. Expanded memory should not have an ADDRESS statement. OTHER includes memory that is not managed by the operating system, such as expanded memory page frames, memory mapped I/O, and bank-switched memory. OTHER memory should have an ADDRESS statement only if it resides in the physical address space. The virtual memory type is used to reserve address space when no physical memory actually exists. Valid values are SYS, EXP, OTHER, and VIR. The default for the MEMTYPE statement is SYS.

SIZE. This is an optional statement used to identify the memory as 8-bit, 16-bit, or 32-bit. Valid values are BYTE, WORD, and DWORD. There is no default value. When provided, this information will be used for verification purposes only.

DECODE. This is an optional statement that specifies the number of address lines decoded by the memory board. Valid values are 20, 24, and 32. The default value is 32.

CACHE. This is an optional statement that indicates whether or not this memory can be cached. The default is NO.

SHARE. This is an optional statement that indicates whether this memory address space may be shared with another board. The default is NO. A text field identifier can also be specified in the SHARE statement to indicate that the function can only share the DMA channel with a device that has a matching identifier. A maximum length of 10 characters may be used for the identifier.

Hints

The following example describes a 128 KB system ROM located at address 0E0000h.

```
MEMORY = 128K
ADDRESS = 0E0000h
    WRITABLE = NO
    MEMTYPE = OTH
```

The following example describes a memory board that can add from 1 to 4 MB of extended memory. Memory can be added to this board in 512 KB increments. This memory must be located in the address space above 1 MB and below 15 MB. Its starting address can be on any 256 KB boundary within this range.

```
MEMORY = 1M - 4M  STEP = 512K
ADDRESS = 1M - 15M  STEP = 256K
    WRITABLE = YES
    MEMTYPE = SYS
```

INIT Statements

INIT statements provide the values needed to initialize programmable boards, or the settings used on switch- and jumper-programmable boards. INIT statements may also be used to display information needed to properly install software drivers. INIT statements may follow any system resource statement, including DMA, IRQ, PORT, MEMORY, and ADDRESS.

These RESOURCE statements may have multiple INIT statements if necessary. The relationship between INIT statements and their corresponding RESOURCE statements is defined by the type of group into which these statements are placed. RESOURCE and INIT statement groups are discussed in the "Resource/INIT Statement Groupings" section in this chapter.

INIT Statement for Programmable I/O Ports

IOPORT(i) indicates which port is to be used. The index, "i", must be given in decimal. The variable list gives a list of values to be output to the port for each resource option. These values must be in binary. Ranges can be used.

INIT = IOPORT(i) [LOC(*bitlist*)] *list*

The LOC clause can be used in the INIT statement to specify the bits being defined. These bit positions are specified in descending order from most to least significant. If the LOC clause is not used, the INIT statement must contain 8, 16, or 32 bits, depending on the stated size of the port. The LOC clause is zero-based and must be given in decimal. The following is an example of a valid INIT statement:

INIT = IOPORT(1) LOC(3-0) 1010

In the following example, bits 1 and 0 would be set to 01 for IRQ 3 or 10 for IRQ 4.

IRQ 3 | 4 INIT = IOPORT(1) LOC (4 3) 01 | 10

Shorthand PORT Statement

A shorthand form of the INIT statement for I/O ports is provided for boards that must initialize a large number of ports. This form does not require a corresponding IOPORT statement, as the port address is contained within the INIT statement. The syntax for the shorthand form is as follows:

INIT = PORTADR(*address*) [BYTE | WORD | DWORD] *value*

The variable *address* specifies the address of the I/O port. Slot-specific addresses must be prefixed with 0Z. The address must be specified in hex.

Indicate the size of the port with BYTE, WORD, or DWORD. The default is BYTE.

The variable *value* gives the value to be output to the port for each resource option. The values are binary only; it is not necessary to specify the radix. A bit position may also be marked with an "r" to indicate that the bit value is read from the port. This value will be either "R," "0," or "1." The length of the value must be the same as the data width of the port: 8, 16, or 32 bits. Ranges may not be used.

INIT Statement for Switches and Jumpers

INIT Statements for Switches and Jumpers are used to specify switch and jumper settings.

INIT = SWITCH(i) LOC(*switchlist*) *valuelist*
INIT = JUMPER(i) LOC(*jumperlist*) *valuelist*

The index, "i", in SWITCH and JUMPER statements indicates which switch or jumper block is to be used. The index must be given in decimal.

The LOC(*switchlist*) and LOC(*jumperlist*) clauses list the location of the individual switches or jumpers used to configure the current resource. Switch and jumper positions must be given in decimal.

A *switchlist* is defined as a list of switch numbers, delimited with blanks. This list may be either ascending or descending, but not mixed. For example:

LOC(1 # 3 # 5)

The following mixed list is an example of *illegal* syntax.

LOC(1 # 5 # 3)

A range may also be used in place of *switchlist*. For example:

LOC(4-1) or LOC(1-4).

A *jumperlist* is defined as a list of jumper numbers, delimited with blanks. This list may be either ascending or descending, but not mixed. For example:

LOC(1 # 3 # 5)

The following mixed list is an example of *illegal* syntax.

LOC(1 # 5 # 3)

A range may also be used in place of *jumperlist*. For example:

LOC(4-1) or LOC(1-4)

Paired and tripole jumpers are specified by their numbers, while inline jumpers are specified by the two or more posts that they connect. A caret is used to show this connection on an inline jumper. For example:

LOC(2^1)

Ranges may not be used for inline jumper *jumperlists*.

List gives a list of the settings needed for each resource option. A 1 represents a switch in the ON position, a 0 represents a switch in the OFF position. Ranges may be used. Settings for jumpers depend on the jumper type, and are discussed in "Jumper Examples" in this chapter.

```
PORT = 300h - 30Fh  STEP = 4
INIT = SWITCH(1) LOC(1 # 2) 00 | 01 | 10 | 11
```

The example above shows a PORT statement that has four possible configurations. DIP switch locations 1 and 2 of SWITCH(1) are used to select a configuration, as shown in the following table:

Port Range	Switch #1	Switch #2
300h-303h	OFF	OFF
304h-307h	OFF	ON
308h-30Bh	ON	OFF
30Ch-30Fh	ON	ON

INIT Statement for Software

INIT statements for software are used to pass parameters to the Software Identification Block for display.

INIT = SOFTWARE(i) *list*

The index, "i", indicates which SOFTWARE statement is displayed during computer configuration. The index must be given in decimal. *List* gives a list of parameters to be passed to the SOFTWARE statement for each resource option. Each parameter should be text enclosed in quotation marks. The SOFTWARE statement will display a block of comments followed by the parameters associated with the resources chosen.

```
BOARD
    ID = ...
    NAME = "Network Adapter"
    .
    .
SOFTWARE(1) =
    "The Network Adapter is initialized using the program \n
    NET.EXE.  The following command should be placed in your \n
    AUTOEXEC.BAT file:  NET /I=x /D=y where "
    .
    .
    IRQ = 3 | 4 | 5
```

```
INIT = SWITCH(1) LOC(1 # 2) 00 | 01 | 10
    INIT = SOFTWARE(1) "x is 3" | "x is 4" | "x is 5"

    DMA = 2 | 5
INIT = SWITCH(1) LOC(3) 0 | 1
    INIT = SOFTWARE(1) "y is 2" | "y is 5"
```

In the above example, the IRQ and DMA statements are followed by two INIT statements. The first INIT statement is used to inform you of the correct switch setting for the chosen value. The next INIT statement provides the correct parameter to the SOFTWARE statement. This parameter is displayed after the SOFTWARE statement. For example, if IRQ 3 and DMA channel 2 were chosen above, the following would be displayed:

```
The Network Adapter is initialized using the program
NET.EXE. The following command should be placed in your
AUTOEXEC.BAT file: NET /I=x /D=y where
x is 3
y is 2
```

Subchoices

There are many examples of existing boards in which the number of possible configurations is very large. Memory boards that allow you to allocate memory as conventional, extended, or expanded are common examples. In these cases, it is desirable to let EASY CONFIG do most of the work rather than forcing you to select from a long and confusing list of configuration choices. SUBCHOICES may be used to insulate you from part of this decision making process.

The syntax for the SUBCHOICE statement is shown below:

```
CHOICE = "text"
    [global resource/INIT groups]
    SUBCHOICE
        subchoice resource/INIT groups
    .
    .
    SUBCHOICE
    subchoice resource/INIT groups
```

A CHOICE may have as many SUBCHOICES as necessary.

The resources listed before the SUBCHOICES in the global resource description block are allocated regardless of the SUBCHOICE selected.

Each SUBCHOICE contains the RESOURCE and INIT statements required for that configuration.

SUBCHOICES are mutually exclusive. Only one SUBCHOICE will be selected per CHOICE by EASY CONFIG.

SUBCHOICES are used for complex statements that cannot be made with a single CHOICE statement. They should only be used in complex situations, such as handling certain memory configurations. SUBCHOICES are configuration alternatives that are not seen. You may still scroll through the possible configurations, including those present in SUBCHOICES, but a list of these alternatives is never displayed as is the case with separate, named choices.

File INCLUDE Statements

Configuration files may be included within a configuration file by using the following statement:

INCLUDE = "!*AAAPPPr*.CFG" [,*n*]

The first character of the configuration file name must be an exclamation point. The next three characters (*AAA*) of each file name should match the manufacturer's three-character ID. The remaining four characters (*PPPr*) are the three-digit product number and one-digit revision number. The file extension must be .CFG. The *n* variable is optional, and can be used to recommend a slot number for the device supported by the configuration file.

The INCLUDE statement may be placed before or after an Initialization Information Block or Function Statement Block.

Note

Customized interaction with EASY CONFIG can be done with the additional capability of configuration file extensions (.OVL, or overlay, files). The general format is:

INCLUDE = *"filename.OVL"*

Refer to the *CFG File Extension Functional Specification* available from Micro Computer Systems, Inc., 2300 Valley View Lane, Suite 800, Irving, Texas, 75062, (214) 659-1514.

A

Using Types and Subtypes

The TYPE and SUBTYPE identifiers are used by product-independent device drivers to identify, initialize, and operate an installed device that is compatible with the device driver. System board and other board manufacturers must specify consistent and expandable TYPE and SUBTYPE identifiers for their products.

When used properly, system board manufacturer configuration routines will be able to interrogate these strings to determine specific information that may be needed to update ISA nonvolatile memory as well.

TYPE Strings

The first segment of the TYPE string should identify the most general device characters, such as video or communications port, followed by TYPE string segments that identify more detailed device characteristics (such as VGA video adapter or asynchronous communications port). For example, the TYPE string for a VGA video adapter is "VID,VGA", where "VID" identifies a video board and "VGA" indicates VGA compatibility. The TYPE string for the asynchronous communications port is "COM,ASY", where "COM" identifies a communications board and "ASY" indicates compatibility with the PC-AT asynchronous port. A list of valid types is included later in this section.

New TYPE segments should be appended to the TYPE string when a device is enhanced with additional capabilities. A device driver compatible with the original product determines its ability to control the device after checking the original TYPE segments. A device driver that supports enhanced capabilities checks the appended TYPE statements to determine the level of capability supported by the device.

For example, the TYPE string for a VGA video adapter with a 1024x1024 high resolution mode might be: "VID,VGA,RES=1024X1024". Device drivers that

support VGA identify the video adapter as VGA compatible and device drivers that support 1024x1024 identify the video adapter as compatible with the 1024x1024 mode.

Another vendor may offer a compatible video adapter with a new 1280x1024 mode. The TYPE string for the 1280x1024 video adapter might be: "VID,VGA,RES=1024X1024,RES=1280X1024". Device drivers that support VGA identify the video adapter as VGA compatible, device drivers that support 1024x1024 identify the video adapter as compatible with the 1024x1024 mode, and device drivers that support 1280X1024 identify the video adapter as compatible with the 1280X1024 mode.

SUBTYPE Strings

The SUBTYPE string identifies the device options selected during configuration. A device driver can scan the TYPE string to determine if the device is compatible with the driver, then can scan the SUBTYPE string to determine the device configuration. For example, the video adapter described above might use the SUBTYPE string to indicate the power-up video display mode. A list of valid types and subtypes is included later in this section.

```
FUNCTION = "VGA Video Adapter"
    TYPE = "VID,VGA,AGA,RES=1280X1024"
    CHOICE(1) = "VGA Default Mode"
        SUBTYPE = VGA
    CHOICE(2)= "1024X768" Default Mode
        SUBTYPE = AGA
    CHOICE(3) = "1280X1024" Default Mode
        SUBTYPE = RES=1280X1024
```

The device driver can utilize the SUBTYPE string to determine the default mode set during power-up. The TYPE/SUBTYPE string for a selection of VGA as the default power-up video mode is:

"VID,VGA,AGA,RES=1280X1024;VGA"

A device driver should read the device configuration registers for configuration information that changes during device operation. A driver that needs detailed configuration information not specified in the SUBTYPE string should also read the device configuration registers.

Board Type	Sort Key	View By Type	Device Description
COM	yes	yes	Communications board
CPU		yes	Processor board
JOY		yes	Joystick board
KEY		yes	Keyboard
MEM		yes	Memory board
MFC	yes		Multi-function board
MSD	yes	yes	Mass storage device
NET	yes	yes	Network function
NPX		yes	Numeric coprocessor
OSE	yes	yes	Operating system or environment
OTH	yes	yes	Miscellaneous entry (OTHer)
PAR		yes	Parallel port controller
PTR		yes	Pointing device
SYS	yes		System board
VID	yes	yes	Video adapter

Function Type		Subtype	Device Description
COM	ASY		Standard AT (ISA) compatible serial controller
	ASY	COM1,COM2	Standard Comm ports: 3f8h and 2f8h respectively
COM	ASY,FIFO		NS16550A based serial controller with FIFO
COM	SYN		Synchronous communication board
COM	SER		Serial Communications controller
COM	TLX		Telex board
COM	FAX		Fax board
COM	MDM		Modem board
COM	ASY,LPT		Serial printer
PAR			ISA compatible parallel port controller
		LPT1, LPT2, LPT3	Std. Par. ports: 3bch, 378h, 278h respectively
PAR	BID		Supports 16550A Bidirectional mode
PAR	LPT		Parallel printer
PTR	8042		Standard 8042 pointing device
PTR	BUS		Bus mouse
NPX	287		Intel 287 numeric coprocessor
NPX	287C		Intel 287C numeric coprocessor
NPX	387		Intel 387 numeric coprocessor
NPX	387SX		Intel 387 numeric coprocessor, SX version
NPX	486		Intel 486 internal numeric coprocessor
NPX	W1167		Weitek 1167 numeric coprocessor
NPX	W3167		Weitek 3167 numeric coprocessor
		SPE=nn	Processor speed, i.e. 16,20,33 (mhz)

Function Type		Subtype	Device Description
KEY	nnn		Standard keyboard where nnn is the number keys: nnn = 083, 084, 101, or 103
		KBD=xx	xx is the keyboard country code: xx = US, UK, FR, GR, IT, SP, LA, SV, SU, NL, DK, NO, PO, SF, SG, CF, or BE
		COU=yyy	yyy is the 3 digit nat. code for tel. system
KEY		TMR=nn	Keyboard typematic rate (nn=0..31)
KEY		DLY=nn	Keyboard delay (nn=0..3)
KEY		VOL=nn	Keyboard click volume (nn=0..15)
KEY		NUM	Numlock state at power-on
VID		SRV	Video shadow RAM at C0000h
VID		RLV	Relocate video to E0000h
VID	MMODE		Multimode adapter
VID	MDA		Standard mono adapter
VID	MGA		Standard Hercules monochrome graphics adapter
VID	CGA		Standard CGA adapter
VID	CGA,RTR		CGA with write sync. required during retrace
VID	EGA		Standard EGA adapter
VID	VGA		Standard VGA adapter
VID	AGA		Standard Advanced Graphics Adapter
		COL=nn	Number of colors supported with option
		RES=nnnnXnnnn	Maximum resolution allowed with option
VID	MON		Specifies that this is a monitor
	COL=nn		Number of colors supported with option
	RES=nnnnXnnnn		Maximum resolution allowed with option
	PRI		Primary monitor
	SEC		Secondary monitor
JOY			Joystick board

Function Type		Subtype	Device Description
OSE	DOS		DOS Operating environment
	OS2		OS2
	UNX		UNIX
	NOV		NOVELL NETWARE
		VER=nn.nn	Operating environment version number
OTH			Miscellaneous entry (OTHer)
NET			Network function
MEM			Memory function
MEM	OPT	SRO	Shadow option ROM at E0000h
MSD	DSKCTL	CTLn	Standard disk controller (n=1,2,..)
		CTL1;SPL=x	Hard disk drive splitting (x=0,1)
MSD	UNITn		Unit number on controller (n=0, 1 ...)
		DSKDRV	disk drive connected to controller*
		TYP=nn	drive type
		BLKSIZ=	bytes per sector or block size
		NUMBLK=	number of blocks
MSD	UNITy,DSKDRV	TYP=33	Hewlett-Packard custom hard disk
MSD	FPYCTL	CTLn	Standard 765 flexible controller
MSD	UNITn		Unit number on controller (1, 2 ...)
		FPYDRV	Flexible drive connected to controller
		TYP=nn	drive type where 1=360 KB 2=1.2 MB, 3=720 KB, 4=1.4 MB
		TAPDRV	Tape drive attached
		IRW40	Irwin 40mb tape drive connected
		IRW80	Irwin 80/120mb tape drive connected
		FIFO	Fifo based controller

Function Type		Subtype	Device Description
MSD	TAPCTL		Tape controller
	QIC02		drive interface
	QIC36		drive interface
	WAN		drive interface
	ARC		drive interface
		UNITn	Unit number on tape controller
		TAPDRV	Tape drive attached
		QIC24	Format of tape
		QIC120	Format of tape
CPU		POS=AUTO	Power-on speed
		CACHE=ON	Internal 80486 cache
	8086		8086 Processor
	80286		80286 Processor
	80386		80386 Processor
	80386SX		80386SX Processor
	80486		80486 Processor
		STEP=x	Processor step, i.e. B,C
		SPE=nn	CPU speed, ie. 16,20,33 (mhz)
		POS=AUTO	Power-on speed

Note

It is recommend that freeform data be used in conjunction with the drive type to further identify a logical drive using the following format:

Block length	2 bytes
Number of Blocks	4 bytes
Drive parameter table	16 bytes

To generate a custom hard disk configuration for HP Vectra PCs, use a drive type of 33 or 34 followed by freeform data.

The following is an incomplete sample configuration file that illustrates the use of types and subtypes.

```
GROUP = " "
    TYPE = "MSD"

    FUNCTION = "Disk Controller"
        TYPE = "DSKCTL"
        CHOICE = "Primary Controller"
            SUBTYPE = "CTL1"
            IRQ = 14
            PORT = 1F0h = 1FFH
        CHOICE = "Secondary Controller"
            SUBTYPE = "CTL2"
            IRQ = 12
            PORT = 170h - 17Fh
        CHOICE = "Controller Disabled"
            DISABLE = YES

    FUNCTION = "Disk 0"
        TYPE = "UNIT0,DSKDRV"
        CHOICE = "Drive Type 35 - 40 MB"
            SUBTYPE = "TYP=35"
            FREEFORM = 20,0FFh, 34h, 67h, 89h, ...
        CHOICE = "Drive Type 43 - 60 MB"
            SUBTYPE = "TYP=43"
            FREEFORM = 20,0FFh, 34h, 67h, 89h, ...
        CHOICE = "No Drive Present"
            SUBTYPE = "TYP=00"

    FUNCTION = "Disk 1"
        TYPE = "UNIT1,DSKDRV"
        CHOICE = "Drive Type 35 - 40 MB"
            SUBTYPE = "TYP=35"
            FREEFORM = 20,0FFh, 34h, 67h, 89h, ...
        CHOICE = "Drive Type 43 - 60 MB"
            SUBTYPE = "TYP=43"
            FREEFORM = 20,0FFh, 34h, 67h, 89h, ...
        CHOICE = "No Drive Present"
            SUBTYPE = "TYP=00"
ENDGROUP
```

Printed in Singapore 12/89
Part Number D2230-90001