Compaq Computer Corporation

Intel Corporation

Phoenix Technologies, Ltd.

EXTENDED SYSTEM CONFIGURATION DATA SPECIFICATION

Version 1.02A

May 31, 1994

Part Number 485547-001

Revision History

Issue Date	Comments
October 5, 1993	Preliminary - version 1.00
December 28, 1993	Changed reserved field in the ECD_FREEBRDHDR to indicate if the PnP ISA or PCI device is re-configurable by the PnP BIOS. Version 1.01
February 14, 1994	Clarification of CannotConfig bit - Version 1.02
May 31, 1994	Clarification of System Device Node interface - Version 1.02A

Table of Contents

1. Introduction

In order to support the automatic configuration of plug and play devices (e.g., PCI, Plug and Play ISA) on platforms that include a standard expansion bus (e.g., ISA, EISA), non-volatile storage is required to store information about the system resources (i.e., IRQ, I/O port, memory window, DMA channel) used by non-plug and play devices. The Plug and Play BIOS Specification [P&PBIOS] describes two techniques for storing this information. For low end systems that have very little available non-volatile storage, the information maintained in non-volatile storage is limited to describing the combined set of resources allocated to non-plug and play devices. This information can be stored in compact coded bit strings. The Plug and Play BIOS Specification describes two interfaces to read and write this information, functions 09h and 0Ah (Set and Get Statically Allocated Resource Information). The information provided through these interfaces is sufficient to allow for the full autoconfiguration of plug and play devices with platforms running plug and play operating systems. Platforms with non-plug and play operating systems or employing add-on plug and play support (e.g., Plug and Play Kit for MS-DOS* and Windows*), may not be able to automatically configure all plug and play devices in all cases. For support of these systems, it is recommended that the Extended System Configuration Data (ESCD) format be employed to store configuration information in non-volatile storage.

The ESCD structure format allows the storage of detailed configuration information on a per device basis rather than the combined configuration storage described in the preceding paragraph. In addition, the ESCD format accommodates storage of configuration information for plug and play devices. The storage of the detailed configuration information allows the BIOS configuration software to work together with configuration utilities to provide robust support for non-plug and play as well as plug and play devices. The detailed configuration information for non-plug and play devices can be used by configuration utilities (e.g., ISA Configuration Utility in the Plug and Play Kit for MS-DOS and Windows) to provide users with a detailed display of system configuration as well as the ability to perform robust resource balancing when adding new cards. The configuration information for plug and play devices is used to either store information about the last working configuration for the device or indicate that the configuration for a plug and play device should be locked, allowing the card to always be configured to the same settings. The former capability allows the robust resource balancing performed by configuration utilities to be employed in the configuration of plug and play devices. The latter capability allows plug and play devices to be automatically configured for systems running non-plug and play devices.

1.1 Purpose

This document is intended to provide sufficient information to system software developers to enable them to utilize the configuration information about Dynamically Configurable Devices (DCDs) for successful configuration of a system. Two varieties of DCDs are considered in this document: PCI devices and Plug and Play ISA devices.

* MS-DOS is a registered trademark and Windows is a trademark of Microsoft Corporation.

This document describes the format of non-volatile storage for configuration (and re-configuration) of DCDs. The following sections address these topics:

- concepts and terminology
- configuration of Plug and Play resources in the Extended Configuration Data (ECD)
- mechanism for locking a single function in a DCD
- mechanism for disabling a single function in a DCD
- primitives to indicate failures while configuring a single function in a DCD
- data formats that are used to store the Plug and Play device status information.

The first design goal was to use, whenever possible, the existing definition of EISA data format as defined in the EISA Specification [EISA]. By adhering to this definition and providing only the necessary extensions, we are assured of upward compatibility with the current implementations of the EISA Configuration Utility (ECU) and device drivers. The second goal was to extend this design to ISA systems so that a common Plug and Play implementation can be provided.

1.2 Related Documents

The following is a list of references containing information that is relevant to the discussion in this document:

[EISA] EISA Specification, Version 3.12, BCPR Services.

[PCI] PCI Local Bus Specification, Revision 2.1, PCI Special Interest Group.

[P&PBIOS] Plug and Play BIOS Specification, Version 1.0a, Compaq Corp., Intel Corp., and Phoenix Technologies, Ltd.

[PnPISA] Plug and Play ISA Specification, Version 1.0a, Intel Corp. and Microsoft Corp.

[DDI] Plug and Play Device Driver Interface for Microsoft Windows 3.1 and MS-DOS, Version 1.0c, Microsoft Corporation.

[ACFG] Plug and Play BIOS Extensions Design Guide, Revision 1.2, Intel Corp.

1.3 Terms and Abbreviations

Acfg BIOS — Auto-Configuration BIOS. System BIOS that contains Intel's Plug and Play BIOS extensions to configure DCDs as required.

CM — Configuration Manager. DOS driver that is responsible for configuring DCDs that are not configured by the Acfg BIOS. It provides access to the configuration space (as defined in [DDI]) for devices present in the system.

DCD	Dynamically Configurable Device. A device whose configuration can be changed (i.e., its resources can be relocated) dynamically. In this document we consider two instances of DCDs: Plug and Play ISA devices and PCI devices.
Disabled DCD	A DCD is disabled when the resources that are currently in use by the DCD are released and its functionality is no longer available to the user.
ECD	Extended Configuration Data. ECD is a data structure that is used to store information about DCDs that could not be stored in the EISA structures.
Enabled DCD	A DCD is enabled when it is using system resources.
ESCD	Extended System Configuration Data. ESCD is the data format for storing resource information describing (E)ISA devices and DCDs.
Locked DCD	A locked DCD is a DCD whose resources cannot be dynamically re-configured. The DCD is bound to its current resources until the lock is released.
NVS	Non-Volatile Storage. NVS is the place where the ESCD is stored. It could be either an NVRAM (in EISA systems and in many ISA systems) or a data file (in legacy ISA systems). The storage media is implementation dependent.
Slot	A slot is a position in the system where a board is inserted. ESCD defines a slot record as a data structure that stores resource information for the device occupying that slot. In ISA systems a slot is merely an identifier used to provide the abstraction of physical slot. Slot zero is reserved for the Motherboard and slots 16 through 64 are reserved for virtual devices.
Virtual Device	A device that is not associated with a specific physical slot. Information about virtual devices is kept in virtual slots.

2. Concepts

2.1 Assumptions

This document assumes that the reader is familiar with the EISA structures and definitions as specified in the EISA Specification [EISA]. Unless otherwise stated, this document conforms to, and complies with the EISA specification. This document further assumes that the reader is familiar with the configuration of DCDs as described in [PCI] and [PnPISA].

This document frequently uses the terms function and device. When the term device is used, it represents the physical hardware. A single device and its configuration resources are translated into a single function. This document also uses the terminology card and board. Both of these terms are synonymous with the term device. On PCI systems, the terms device and function are used to identify the location of a specific PCI device such as IDE, SCSI, etc. A function as defined in the EISA specification [EISA] and ESCD is an entity that describes resources and information about devices. A device can have several functions. For example, a Plug and Play ISA device can have two functions, with each function using resources independently. It is also possible that a device has a single function.

2.2 Slot Records Without 'cfg' Files

The EISA specification [EISA], specifies that board and slot id configuration information is kept in bytes zero and one of the *ID and slot information* field. Byte one of this field contains four reserved bits numbered three to six. The ESCD specification redefines the *bit six* of byte one in the ID and slot information field from *reserved bit* to *bIdSlotNoCfgFile bit*. This bit will be set to *one* when the slot board record is created by the Configuration Manager or the system ACFG BIOS. When set, this bit should be interpreted to mean that this slot record was auto-configured and therefore doesn't require any configuration files for determining the device configuration possibilities. This new *bIdSlotNoCfgFile bit* is defined in the EISAIDSLOTINFO structure in the escdfmt.h (see attached Appendix C).

It is the responsibility of the Configuration Utilities to correctly interpret this bit and when set, not make requests for configuration files.

3. Slot Assignment

This section describes the locations where resource information for different device types will be stored in ESCD.

3.1 Motherboard

As defined in the EISA specification, mother-board configuration information is always stored in slot zero as multiple functions that loosely correspond to the actual ISA devices that are embedded in the motherboard. The ESCD specification requires that only the ISA devices embedded in the motherboard be described as mother-board functions. As described in section 5.6, PCI device information is always stored in virtual slots. This means that for systems that contain PCI devices embedded in the motherboard (such as IDE and or SCSI) and are attached to the PCI bus, the virtual slots will be used to store the PCI-specific configuration information for the motherboard (while retaining the ISA configuration information in slot zero).

The Acfg BIOS uses *BrdConfgStat*, i.e., bit 7 in byte 1 of *ID and slot information* field for slot zero (refer to [EISA]), to indicate information inconsistency for motherboard devices. When the Acfg BIOS detects inconsistency between the Setup CMOS and NVS for motherboard data, *BrdConfgStat* will be set to *one*. It is the responsibility of the Configuration Utility to correctly interpret this bit and take the appropriate action.

3.2 Expansion Slots

Expansion slots are defined as slot numbers 1 through 15 [EISA]. The configuration information for the current generation of (E)ISA devices is stored in expansion slots. Expansion slots are also used to store information about Plug and Play ISA devices. Since the Plug and Play ISA devices are dynamically re-configurable, the ESCD uses the ECD to capture the additional information that needs to be stored. Section 4 and Appendix A describe ECD in greater detail.

3.3 Virtual Slots

Virtual slots are defined as slot numbers 16 through 64 [EISA]. These are reserved for configuration information for virtual devices. They can be safely used to store configuration information for devices other then those defined by the EISA specification [EISA]. Virtual slots are used to store the configuration information for PCI devices. In case of multi-function PCI devices, a single virtual slot will be used for multiple PCI functions that are associated with one PCI board.

Support for virtual slots is optional in the EISA specification. However, the ESCD specification requires that EISA systems support virtual slots. The number of virtual slots that need to be supported is dependent on the number of PCI devices that can be present in the system.

4. Storing of DCD Information

When compared to (E)ISA devices, DCDs need to store certain additional information. This additional information is stored in the ECD as free form data. The ECD structure is defined in escdfmt.h (see Appendix C). ECD is always the last function in the slot record for a DCD and is always disabled. The function information byte in the ECD (i.e., the *bFuncInfo* field) must be set to value *0xC0* to identify the ECD as an EISA free-format disabled function. The format for each type of DCD is now described in turn.

A single-function PCI device is described as follows:

i) a standard EISA function containing the resource usage for function number 0 on the PCI device
ii) the ECD

If the PCI device has several functions, there are several standard EISA functions in the slot record prior to the ECD, with each standard EISA function corresponding to one function in the PCI device. Since there is no correspondence between the EISA function number and the PCI function number (i.e., EISA function number one could be describing the PCI device function number three), the ECD contains the necessary information that allows correct interpretation. For more details see section 5.6. Resource information for each function of the multi-function PCI device is stored within the *same* virtual slot.

The following figure illustrates the constituents of one virtual slot record for a PCI device with two PCI functions.

ESCD Component	*Description*
Function 0	Standard EISA function to store resource information for the PCI function #0
Function 1	Standard EISA function to store resource information for the PCI function #3
Function 2	Disabled function, containing the ECD for PCI functions #0 and #3.

A single-function Plug and Play ISA card is described as follows:

i) a standard EISA function containing resource usage for one function of the device
ii) the ECD

If the Plug and Play ISA card has several functions, there are several standard EISA functions in the slot record prior to the ECD, with each standard EISA function corresponding to one function on the board.

The following figure illustrates the layout of a slot record for a three-function Plug and Play ISA card:

ESCD Component	*Description*
Function 0	Standard EISA function to store resource information for PnP ISA logical device # 0
Function 1	Standard EISA function to store resource information for PnP ISA logical device # 1
Function 2	Standard EISA function to store resource information for PnP ISA logical device # 2
Function 3	Disabled function, containing the ECD

5. Reconfiguration of DCDs

The DCDs are re-configurable at run-time, while the traditional (E)ISA devices are not. When an (E)ISA device is configured at a particular resource, either some jumper needs to be changed or a software utility needs to be run to re-configure the device (soft-settable (E)ISA devices allow for greater ease-of-use as far as re-configuration is concerned). On the other hand, PCI devices can be configured at one of several resource values depending on the availability of resources. This implies that the resources that are currently allocated to DCDs can be re-used for some static (E)ISA device. This would be desirable if, for example, the DCD resources are the only resources with which the static (E)ISA devices can be configured. At the next system boot, the DCD would then get re-configured with a different, non-conflicting resource.

Configuration utilities that understand the ESCD will be cognizant of DCDs (and the fact that they are re-configurable) and will be able to utilize the resource information of the DCDs and possibly re-allocate those resources to an add-in card when necessary. However, there are current implementations of ECUs that are not aware of DCDs and do not understand ESCD. In order to make re-configuration work even with these *legacy* ECUs, a few additional requirements in the ESCD must be specified.

There are two type of DCDs - namely those functions that need to be configured and activated by the PnP BIOS prior to OS initialization (bootable DCDs) and functions that can be reconfigured by the Configuration Manager during OS initialization(non-bootable DCDs). The ability to re-configure DCDs is dependent on the presence and capabilities of the PnP BIOS and Configuration Manager. The re-configuration information for non-bootable DCDs is always available to the Configuration Manager who can re-configure and activate any non-active DCD function. DCD functions that participate in the boot process (i.e., they have the expansion AT-BIOS) can be only re-configured by the system BIOS that contains PnP BIOS extensions. If the PnP BIOS does not have access to NVS to store the re-configuration information about bootable DCD function, then the bootable DCD function is not re-configurable. To provide the Configuration Utility with the knowledge about DCD function re-configurability, the utility can examine the *fwECDFuncsCannotConfig* bit-map field defined in the *ECD_FREEFORMBRDHRD* structure. Each bit in the bit-map field that is set to 0 represents a DCD function that is re-configurable.

5.1 Re-configurability with Legacy ECUs

The DCD configuration information will be stored in the appropriate slots as described in Chapter 4 of this document. Because the slot records (virtual slots for the PCI devices or the expansion slots for Plug and Play ISA boards) containing DCD records do not have corresponding CFG files, a legacy ECU[1] will delete these records from NVS. Since the ECU will not consider the resources contained in these records, it will be able to re-allocate the resources of the DCDs.

At the next boot, the DCDs will get re-configured at their new values (around the resources being used by (E)ISA devices). Note that this re-configuration at boot requires the presence of the Configuration Manager (CM). Thus, although the ECU is unaware of the DCDs and the CM, it is able to operate with the CM in achieving re-configurability.

New ECUs can look at the additional fields in the ESCD and achieve both the disable functionality and independently control re-configurability of DCD resources. The additional field that will allow new ECUs to support disabling of functions is the *fwECDFuncsDisabled* bit-map in the ECD.

5.2 Implementing Disabled DCD Functions

The ECD (which is present for every DCD) contains the *fwECDFuncsDisabled* field. This field is a bit-map of disabled functions for that particular device (board). Since the ECD is a free format disabled function it will be ignored by the ECU. The Acfg BIOS or CM will use the disabled bit map in the ECD to determine which functions in that device need to be configured.

The function enable field in the EISA standard function of each DCD will be used by the Acfg BIOS and the CM to determine whether the resources for that function have been *locked* (or

[1]For this sub-section only, unless otherwise specified, an ECU refers to a legacy ECU.

reserved) to their previous configuration. If the enable bit is found to be set for a particular DCD function, then that DCD function needs to come back up with the same resource configuration. If the DCD function is found to be disabled, then the Acfg BIOS or the CM does not have to configure that DCD function to the resources specified in the function.

5.3 Implementing Locked DCD Functions

The EISA specification only defines the mechanism for locking resources at the granularity of boards [EISA]. Thus, if a board is locked, all the functions within that board are also locked. Similarly, unlocking a board unlocks all the functions within that board. A board (and all of the functions associated with the board) is locked by setting the lock bit in byte one of the *ID and slot information* field of that board.

With the ESCD, a finer granularity of locking is achieved for DCDs because the user is now able to reserve the resources at the DCD function level without locking all the resources at the board level. This is achieved by the combination of turning the *lock bit in the board on* and turning *on the enable bit* for the DCD function.

When the status of the standard format EISA function that describes the corresponding DCD is changed from disabled to enabled, and the lock bit for the board is enabled, the configuration resources for the target device become locked. The Acfg BIOS or CM will check both the board lock bit and the function enable bit before attempting to configure the device.

To summarize, the board lock bit indicates that there is one or more functions with resources that are locked. The enabled functions in that device are locked and their resources are fixed while the resources of disabled functions can be re-configured as required. See Appendix B for a more detailed explanation on the interpretation of configuration states.

It is the responsibility of the CM or the Configuration Utility to correctly update both the board lock bit and the standard EISA function disable bit when processing the lock and unlock requests for the DCDs.

5.4 DCD Configuration Errors

Depending on the requirement of device availability at pre-boot or post-boot time, either the Acfg BIOS or CM will configure the DCDs. Because device re-configuration may not always succeed, the agent attempting to configure the DCD needs a mechanism for reporting configuration errors.

The EISA specification only defines the mechanism for signaling configuration failure at the granularity of boards [EISA]. Thus, there is no way to distinguish between configuration failures occurring on more than one function. Board configuration error is indicated by setting the configuration status bit in byte one of the *ID and slot information* field of that board to *one*.

With the ESCD, a finer granularity of configuration error reporting is possible for DCDs by utilizing the *fwECDFuncsCfgErrors bit-map* field in the ECD. This is achieved by turning the *configuration bit status for the board off* and turning *the function bit in the fwECDFuncsCfgErrors* on for the DCD.

The Acfg BIOS, CM, and Configuration Utility will check both the board configuration status bit and the ECD configuration errors bit map when resolving configuration errors.

When the Acfg BIOS or the CM is unable to correctly configure one or more of the DCD functions, it needs to report the failures with this mechanism.

5.5 Plug and Play ISA devices

The configuration of the Plug and Play ISA cards is reflected in the system by storing relevant information in the expansion slots one through fifteen. The slot will contain one or more standard EISA functions that are followed by one ECD function. The ECD contains both generic and device type specific information. The Plug and Play ISA card specific information is kept in the *ECD_PNPBRDID structure*. This structure contains the *vendor id and the device serial number* that are needed for device identification. This structure is part of the *PNPFREEFORMFUNC* structure included in the ECD for the Plug and Play ISA board.

5.6 PCI Devices

The configuration of PCI devices is reflected in the system by storing relevant information in the virtual slots numbered 16 through 64. One virtual slot will contain a variable number (one to eight) of standard EISA functions (describing the resources used by the individual PCI functions) followed by the ECD function. The design of the Acfg BIOS and the CM requires that the location of specific PCI devices be determined. This in turn necessitates the use of *the bus number, the device/function number and the device id, vendor id*. This PCI specific information is kept in the *ECD_PCIBRDID* structure. Several instances of ECD_PCIBRDID structure will be present (one to eight according to the PCI specification [PCI]), depending on the number of the individual functions that constitute the multi-function PCI device.

6. ESCD Description

6.1 Difference Between EISA and ISA Systems

The EISA specification defines the layout of the NVRAM for storing the configuration of mother-board and (E)ISA devices [EISA]. The ESCD extensions in this environment are limited to the following items for DCDs only:

- ECD function for DCDs
- handling of disabled functions
- re-definition of locking
- handling of configuration errors

For EISA systems, the *ESCD_BRDHDR and the ESCD_CFGHDR* structures are not required and will not be present. The clients that require access to the EISA NVRAM can use the mechanism described in the EISA specification [EISA].

In an ISA environment the layout of ESCD conforms to the format described in section 6.2. It requires that configuration information for each slot be preceded by the ESCD_BRDHDR structure. The NVS will always contain the ESCD_CFGHDR and the checksum. The ESCD functions, described in section 8, provide uniform access to the NVS. All ESCD extensions are applicable and required in the ISA environment.

6.2 The ESCD format

The ESCD file format closely resembles the NVS as described by the EISA specification. It contains the combination of control and configuration data. The resource limitations of EISA configuration data is also applicable to the ESCD format. The following table is the graphical representation of the individual data structures that collectively describe the ESCD for a system that contains a combination of (E)ISA and DCDs:

Data Structure	*Description*
ESCD_CFGHDR	File configuration header
ESCD_BRDHDR	Header for slot 1, (E)ISA board
EISA_PackedData	Packed data for the (E)ISA-board
ESCD_BRDHDR	Header for slot 3, Plug and Play ISA board
EISA_PackedData	Packed data for the PnP ISA-card that contains two functions: 2 std EISA function plus ECD disabled function plus slot checksum word.
•	•
•	•
ESCD_BRDHDR	Header for Virtual slot 16, single function #0 PCI device
EISA_PackedData	Packed data for the PCI device, function #0: one disabled std EISA function plus ECD disabled function plus slot checksum word.
ESCD_BRDHDR	Header for Virtual slot 17, multi-function PCI board with three non-contiguous functions
EISA_PackedData	Packed data for the PCI board, functions #0, #3 and #5 -- three disabled std EISA format functions plus ECD disabled function plus slot checksum WORD.
ESCD_BRDHDR	Header for slot 0, mother-board
EISA_PackedData	Packed data for the mother-board plus slot checksum word.
CheckSum	Two bytes check sum for ESCD file.

The order of the slots in the above table is for illustration only. The program that uses the data can determine the slot position by examining the content of the *bSlotNum* field in the ESCD_BRDHDR. The ECD disabled function for PCI is described by the PCIFREEFORMFUNC structure. The ESD disabled function for PnP is described by the PNPFREEFORMFUNC structure. For detailed descriptions of the individual data structures shown in the above table, refer to the definitions in Appendix C.

Slot checksum is a 16-bit logical (modulo 64K) sum of ASCII values of the EISA_PackedData. ESCD file checksum is a 16-bit logical (modulo 64K) sum of ASCII values in the ESCD file. Slot checksum is optionally calculated by the caller. ESCD file checksum must be calculated by the BIOS on a write to ESCD.

7. Using the CM

A system configuration utility can use the CM interface in performing the services offered by the Configuration Manager. However, caution should be exercised so that lock (or unlock) configuration requests to CM are not mixed with configuration utility updating of the NVS. The lock/unlock request to CM results in immediate update of NVS. If the Configuration Utility has a copy of the NVS before requesting the CM to lock device configuration and then decides to update the NVS itself, the effect of a CM lock request will be lost.

8. PnP BIOS ESCD Access Interfaces

The Plug and Play BIOS specification defines BIOS interfaces to access system configuration information. The Plug and Play BIOS will implement three interfaces to obtain information about the ESCD and read/write of the data. These interfaces follow the function calling prototype of the form

int FAR (*entryPoint)(int Function, ...);

that is fully described in the Plug and Play BIOS specification. System software will interface with all of the ESCD functions described in this specification by making a far call to this entry point. As noted above the caller will pass a function number and a set of arguments based on the function being called.

The extended configuration services are a mechanism whereby the system software may lock the system resource configuration for specific devices by explicitly assigning a configuration to the device. This will allow the Plug and Play system BIOS to fully configure the system at power up and allocate the system resources assigned to the device by the system software. The system resource configuration information for the devices in the system must follow the specified format. The format for maintaining the configuration information in non-volatile storage is the standard EISA packed configuration data block structure with some necessary extensions. The EISA packed configuration data block is defined in the EISA Specification [EISA]. The necessary extensions to the EISA format are encompassed in a data structure referred to as Extended System Configuration Data (ESCD). ESCD is a data structure that is used to store information about Plug and Play devices that could not be stored in the EISA structures. It is assumed that adhering to this definition will ensure upward compatibility with current EISA implementations and device drivers. ***Refer to Appendix A for more information on the Extended System Configuration Data (ESCD) format.***

The primary purpose of these interfaces is to provide a mechanism for system software to specify the system resources assigned to the devices in the system, which will enable the system BIOS to fully configure the system at power up.

If necessary, the system BIOS can determine the size of the data in the ESCD from the *ESCD Configuration Header Structure* when required to update and/or modify the contents of the ESCD. Refer to Appendix A for more information on the format of the ESCD data. Note that the information passed in the *Read* and *Write Extended System Configuration Data* function calls need not be stored in the ESCD format. The ESCD structure only refers to the format in which the data is passed between the Operating System and the System BIOS. The data may be stored in any format the system vendor chooses.

When a call is made to the *Write Extended System Configuration Data(ESCD)* BIOS function, it is the responsibility of the caller to ensure that the system board device information in Slot 0 is also updated to the device nodes using the *Set System Device Node* function. Further, the system BIOS is responsible for constructing the current system board image (namely, Slot 0 record) from the current configuration of the System Device Nodes on boot. The presence of Slot 0 record in ESCD is required. Further, there is a one-to-one correspondance between the System Device Nodes and the functions in Slot 0 record. The enumeration order of the System Device Nodes is used in establishing the correspondance between the System Device Nodes and functions in Slot 0.For example, ESCD slot 0 record for a system with 5 device nodes numbered 1, 3, 5, 6, 7 should reflect the device nodes as functions 1, 2, 3, 4 and 5.

8.1. Function 41h - Get Extended System Configuration Data (ESCD) Information

Synopsis:

```
int FAR (*entryPoint)(Function, MinESCDWriteSize, ESCDSize, NVStorageBase, BiosSelector);
int Function;                           /* PnP BIOS Function 041h */
unsigned int FAR *MinESCDWriteSize; /* Minimum buffer size in bytes  for writing to NVS */
unsigned int FAR *ESCDSize;             /* Size allocated for the ESCD... */
                                        /* ...within the non-volatile storage block */
unsigned long FAR *NVStorageBase;       /* 32-bit physical base address for... */
                                        /* ...memory mapped non-volatile storage media */

unsigned BiosSelector;                  /* PnP BIOS readable/writable selector */
```

Description:

This function provides information about the non-volatile storage on the system that contains the Extended System Configuration Data (ESCD). It returns the size, in bytes, of the minimum buffer required for writing to NVS in *MinESCDWriteSize*, the maximum size, in bytes, of the block within the non-volatile storage area allocated specifically to the ESCD in *ESCDSize*, and if the nonvolatile storage is memory mapped, the 32-bit absolute physical base address will be returned in *NVStorageBase*. The physical base address of the memory mapped non-volatile storage will allow the caller to construct a 16-bit data segment descriptor with a limit of at 64K and read/write access. This will enable the Plug and Play system BIOS to read and write the memory mapped non-volatile storage in a protected mode environment. If the non-volatile storage is not memory mapped the value returned in *NVStorageBase* should be 0. It is assumed that the size of the non-volatile storage which contains the ESCD will not exceed 32K bytes.

The portion of non-volatile storage used to store the Extended System Configuration Data (ESCD) may only be a subset of the total non-volatile storage available on the system. In addition, only the system BIOS knows where the ESCD resides in the system's non-volatile storage and the proper method for accessing the non-volatile storage. Therefore, the caller should never attempt to directly access the ESCD. System software should utilize the *Read Extended System Configuration Data* and *Write Extended Configuration Data* functions described in this specification.

The *BiosSelector* parameter enables the system BIOS, if necessary, to update system variables that are contained in the system BIOS memory space. If this function is called from protected mode the caller must create a data segment descriptor using the 16-bit Protected Mode data segment base address specified in the Plug and Play Installation Check data structure, a limit of 64k, and the descriptor must be readable and writable. If this function is called from real mode *BiosSelector* should be set to the Real Mode 16-bit data segment address as specified in the Plug and Play Installation Check structure. Refer to section 4.4 in the Plug and Play BIOS specification for more information on the Plug and Play Installation Check Structure and the elements that make up the structure.

The function is available in real mode and 16-bit protected mode.

Note that this function may also be accessible through the INT 1Ah interface. Refer to *Intel PnP BIOS Extensions Design Guide* [ACFG] for details.

Returns:

0 if successful - SUCCESS
!0 if an error occurred - error code (The function return codes are described in Appendix D)

The FLAGS and registers will be preserved, except for AX which contains the return code.

Example:

The following example illustrates how the 'C' style call interface could be made from an assembly language module:

```
        .
        .
        .
push    Bios Selector
push    segment/selector of NVStorageBase       ; Pointer to 32-bit physical base address
push    offset of NVStorageBase
push    segment/selector of ESCDSize            ; Pointer to size of ESCD
push    offset of ESCDSize
push    segment/selector MinESCDWriteSize       ; Pointer to MinESCDWriteSize block size
push    offset of  MinESCDWriteSize
push    GET_ESCD_SIZE                           ; Function 041h
call    FAR PTR entryPoint
add     sp,16                                   ; Clean up stack
cmp     ax,SUCCESS                              ; Function completed successfully?
jne     error                                   ; No-handle error condition
        .
        .
        .
```

8.2. Function 42h - Read Extended System Configuration Data (ESCD)

Synopsis:

```
int FAR (*entryPoint)(Function, ESCDBuffer, ESCDSelector, BiosSelector)
int Function;                    /* PnP BIOS Function 042h */
char FAR *ESCDBuffer;            /* Address of caller's buffer for storing
                                        ESCD */
unsigned ESCDSelector;           /* ESCD readable/writable selector */
unsigned BiosSelector;           /* PnP BIOS readable/writable selector */
```

Description:

This function is used to read the ESCD data from nonvolatile storage on the system into the buffer specified by *ESCDBuffer.* The entire ESCD will be placed into the buffer. It is the responsibility of the caller to ensure that the buffer is large enough to store the entire ESCD. The caller should use the output from Function 41 (the *ESCDSize* field) when calculating the size of the *ESCDBuffer.* The system BIOS will return the entire ESCD, including information about system board devices. The system board device configuration information will be contained in the slot 0 portion of the ESCD. The system BIOS can determine the size of the data in the ESCD from the *ESCD Configuration Header Structure*. Refer to Appendix A for more information on the format of the ESCD data.

The *ESCDSelector* parameter is required when the *Get Extended System Configuration Data Information* function has returned a 32-bit absolute physical base address for the non-volatile storage media and this function is going to be called from protected mode. In this case, it is the responsibility of the caller to construct a 16-bit data segment descriptor with base = *NVStorageBase,* a limit of 64K and read/write access. In real mode, the *ESCDSelector* is a segment that points to *NVStorageBase*. If the *Get Extended System Configuration Data Information* function returned 0 for the 32-bit physical base address of the non-volatile storage, this parameter should be 0.

The *BiosSelector* parameter enables the system BIOS, if necessary, to update system variables that are contained in the system BIOS memory space. If this function is called from protected mode the caller must create a data segment descriptor using the 16-bit Protected Mode data segment base address specified in the Plug and Play Installation Check data structure, a limit of 64K, and the descriptor must be readable and writable. If this function is called from real mode, *BiosSelector* should be set to the Real Mode 16-bit data segment address as specified in the Plug and Play Installation Check structure. Refer to section 4.4 in the Plug and Play BIOS specification for more information on the Plug and Play Installation Check Structure and the elements that make up the structure.

The function is available in real mode and 16-bit protected mode.

Note that this function may also be accessible through the INT 1Ah interface. Refer to *Intel PnP BIOS Extensions Design Guide* [ACFG] for details.

Returns:

0 if successful - SUCCESS
!0 if an error occurred - error code (The function return codes are described in Appendix D)

The FLAGS and registers will be preserved, except for AX which contains the return code.

Example:

The following example illustrates how the 'C' style call interface could be made from an assembly language module:

```
        .
        .
        .
push    Bios Selector
push    ESCD Selector                       ; ESCD selector if protected mode and NVS is
                                                ; memory mapped, otherwise 0
push    segment/selector of ESCDBuffer      ; Pointer to caller's ESCD memory buffer
push    offset of ESCDBuffer
push    READ_ESCD                           ; Function 042h
call    FAR PTR entryPoint
add     sp,10                               ; Clean up stack
cmp     ax,SUCCESS                          ; Function completed successfully?
jne     error                               ; No-handle error condition
        .
        .
        .
```

8.3. Function 43h - Write Extended System Configuration Data (ESCD)

Synopsis:

```
int FAR (*entryPoint)(Function, ESCDBuffer, ESCDSelector, BiosSelector);
int Function;                   /* PnP BIOS Function 043h */
char FAR *ESCDBuffer;           /* Buffer containing complete ESCD to write... */
                                /* ...to non-volatile storage */
unsigned ESCDSelector;          /* ESCD readable/writable selector */
unsigned BiosSelector;          /* PnP BIOS readable/writable selector */
```

Description:

This function will write the Extended Static Configuration Data (ESCD) contained in the *ESCDBuffer* to non-volatile storage on the system. The data contained in the caller's buffer must contain a complete block of ESCD structures describing the configuration information for devices on the system. The caller should use the output from Function 41 (the *MinESCDWriteSize* field) when calculating the size of the *ESCDBuffer*. Reconfiguration of the system board devices must be handled through *Get System Device Node* and *Set System Device Node* functions as described in Plug and Play BIOS. The system BIOS can determine the size of the data in the ESCD from the *ESCD Configuration Header Structure* within the caller's ESCD buffer. Refer to Appendix A for more information on the format of the ESCD data.

The *ESCDSelector* parameter is required when the *Get Extended System Configuration Data Information* function has returned a 32-bit absolute physical base address for the non-volatile storage media and this function is going to be called from protected mode. It is the responsibility of the caller to construct a 16-bit data segment descriptor with base = *NVStorageBase*,a limit of 64K and read/write access. In real mode, the *ESCDSelector* is a segment that points to

NVStorageBase. If the *Get Extended System Configuration Data Information* function returned 0 for the 32-bit physical base address of the non-volatile storage, this parameter should be 0.

The *BiosSelector* parameter enables the system BIOS, if necessary, to update system variables that are contained in the system BIOS memory space. If this function is called from protected mode, the caller must create a data segment descriptor using the 16-bit Protected Mode data segment base address specified in the Plug and Play Installation Check data structure, a limit of 64K, and the descriptor must be readable and writable. If this function is called from real mode, *BiosSelector* should be set to the Real Mode 16-bit data segment address as specified in the Plug and Play Installation Check structure. Refer to Plug and Play BIOS specification (section 4.4) for more information on the Plug and Play Installation Check Structure and the elements that make up the structure.

The function is available in real mode and 16-bit protected mode.

Note that this function may also be accessible through the INT 1Ah interface. Refer to *Intel PnP BIOS Extensions Design Guide* [ACFG] for details.

Returns:

0 if successful - SUCCESS
!0 if an error occurred - error code (The function return codes are described in Appendix D)

The FLAGS and registers will be preserved, except for AX which contains the return code.

Example:

The following example illustrates how the 'C' style call interface could be made from an assembly language module:

```
        .
        .
        .
push    Bios Selector
push    ESCD Selector                       ; ESCD selector if protected mode and NVS is
                                            ; memory mapped, otherwise 0
push    segment/selector of ESCDBuffer      ; pointer to ESCD Buffer
push    offset of ESCDBuffer
push    WRITE_ESCD                          ; Function 043h
call    FAR PTR entryPoint
add     sp,10                               ; Clean up stack
cmp     ax,SUCCESS                          ; Function completed successfully?
jne     error                               ; No-handle error condition
        .
        .
        .
```

Appendix A: Extended System Configuration Data (ESCD)

This appendix describes the format of non-volatile storage for storing configuration information about the devices installed in the system and assumes that the reader is familiar with the EISA data structures and definitions as specified in the EISA Specification Version 3.12 from BCPR Services, Inc. Unless otherwise stated, this document conforms to, and complies with the EISA specification.

ESCD Configuration Header (ESCD_CFGHDR):

Field	Offset	Length	Value
Size	00h	WORD	Varies
Signature	02h	DWORD	"ACFG" (ASCII)
Minor version number	06h	BYTE	Varies
Major version number	07h	BYTE	02h
Board count	08h	BYTE	Varies
Reserved	09h	3 BYTES	0's

Size: Specifies the size of the ESCD data in non-volatile storage.

Signature: The ASCII string "ACFG" identifies the data in the non-volatile storage as Extended Configuration Data.

Minor and Major version: Current version support. The minor version number should be greater than or equal to 0. The major version number should be set to 2 to indicate Version 2 of the ESCD specification.

Board count: Specifies the number of boards in the Extended Configuration Data block.

ESCD Board Header (ESCD_BRDHDR):

Field	Offset	Length	Value
Size	00h	WORD	Varies
Slot number	02h	BYTE	Varies
Reserved	03h	BYTE	0's

Size: Size of the ESCD board header structure.

Slot number: Identifies the slot the board is plugged into on the system.

Extended Configuration Data Freeform Board Header (ECD_FREEFORMBRDHDR):

Field	Offset	Length	Value
Signature	00h	DWORD	"ACFG" (ASCII)
Minor version number	04h	BYTE	Varies
Major version number	05h	BYTE	02h
Board Type	06h	BYTE	Varies
Reserved	07h	BYTE	0
Disabled Functions	08h	WORD	Varies
Configuration Error Functions	0Ah	WORD	Varies
Functions are re-configurable	0Ch	WORD	Varies

Signature: Identifies the start of the ECD freeform header and should be initialized to "ACFG".

Minor version number: Provides current version information and should be greater than or equal to 0.

Major version number: Provides current version information and should be set to 2 indicate Version 2 of the ESCD specification.

Board type: Identifies the type of board and should be one of the following: ISA=01h, EISA=02h, PCI=04h, PCMCIA=08h, PNPISA=10h, MCA=20h.

Disabled Functions: Bitmap that specifies the functions that are disabled on the device. For instance, bit 4 is set to a one indicates that function 4 on the device is disabled. Note that EISA function numbering scheme(i. e., starting with function number 1) is in effect.

Configuration Error Functions: Bitmap that indicates the function on the device has a configuration error.

Functions are re-configurable: Bitmap that indicates which of the functions on the device can be re-configured by either the PnP BIOS or the Configuration Manager.

Freeform PCI Device Identifier and Data (ECD_PCIBRDID):

Field	Offset	Length	Value
Bus number	00h	BYTE	Varies
PCI device and Function number	01h	BYTE	Varies
PCI device identifier	02h	WORD	Varies
PCI vendor identifier	04h	WORD	Varies
Reserved	06h	2 BYTES	0's

Bus number: Represents the PCI bus number (0-255).

PCI device and Function number: Specifies the PCI device and function numbers.
Bits 7:3 - Device number (0-31).
Bits 2:0 - Function number (0-7).

PCI device identifier and PCI vendor identification: Provide the device and vendor identification for the PCI hardware. Refer to the PCI Specification for more information about these identifiers.

Freeform Plug and Play ISA Board Identifier (ECD_PNPBRDID):

Field	Offset	Length	Value
Vendor identifier	00h	DWORD	Varies
Serial number	04h	DWORD	Varies

Vendor identifier: Unique 32-bit EISA identifier.

Serial number: Differentiates between multiple cards that have the same vendor identifier when they are plugged into the system.

Refer to the Plug and Play ISA Specification for more information on the Vendor Identifier and Serial number.

Plug and Play ISA Extended Configuration Data (ECD) function (PNPFREEFORMFUNC): Used as the last function for a specific Plug and Play ISA board.

Field	Offset	Length	Value
Function size	00h	WORD	28 (1Ch)
Selection size	02h	BYTE	01h
Selection data	03h	BYTE	00h
Function information byte: Identifies free format configuration data block(bit 6) and disabled (bit 7)	04h	BYTE	0Ch
Free format data size	05h	BYTE	18h
ECD_FREEFORMBRDHDR (see structure definition above)	06h	16 BYTES	Varies
ECD_PNPBRDID (see structure definition above)	16h	8 BYTES	Varies

Function size: Specifies the size of the structure.

Selection size: Represents the length or number of selection bytes that follow. This field should be initialized to 01h.

Selection data: Identifies the functions selected on the board. This fields should be initialized to 00h.

Function information byte: Identifies this function as an Extended Configuration Data structure. This value must be set to 0Ch, bit 6 and bit 7 set, which indicates free form data follows and the function is disabled.

Free format data size: Size of the free format data that follows. This byte is not included in the size value.

ECD_FREEFORMBRDHDR: Extended Configuration Data freeform board header. This data structure is defined above.

ECD_PNPBRDID: This data structure specifies the Plug and Play ISA board identifier and serial number. This data structure is defined above.

PCI Extended Configuration Data (ECD) function (PCIFREEFORMFUNC): Used as the last function for a specific PCI board.

Field	Offset	Length	Value
Function size	00h	WORD	Varies
Selection size	02h	BYTE	01h
Selection data	03h	BYTE	00h
Function information byte: Identifies free format configuration data block(bit 6) and disabled (bit 7)	04h	BYTE	0Ch
Free format data size	05h	BYTE	Varies
ECD_FREEFORMBRDHDR (see structure definition above)	06h	16 BYTES	Varies
ECD_PCIBRDID (see structure definition above). Array of 1 to 8 structures for multi-function PCI boards.	16h	8 to 64 BYTES	Varies

Function size: Specifies the size of the structure.

Selection size: Represents the length or number of selection bytes that follow. This field should be initialized to 1.

Selection data: Identifies the functions selected on the board. This fields should be initialized to 0.

Function information byte: Identifies this function as an Extended Configuration Data structure. This value must be set to 0Ch, bit 6 and bit 7 set, which indicates free form data follows and the function is disabled.

Free format data size: Size of the free format data that follows. This byte is not included in the size value. Depending on the number of PCI board identifier (ECD_PCIBRDID) data structures, the free format data size can be from 24 bytes long up to a maximum value of 80 bytes.

ECD_FREEFORMBRDHDR: Extended Configuration Data freeform board header. This data structure is defined above.

ECD_PCIBRDID: This data structure specifies the PCI board identifier. This field is specified as an array of structures in which there will be only one entry for each function on a multi-function PCI board. This data structure is defined above. There can be from 1 to 8 structures specified here.

Appendix B State Table for DCD Configuration

The purpose of this Appendix is to explain the correlation of all the possible states of a DCD function with the states of the following ESCD fields: EISA device lock bit, EISA function disable bit, and the ECD *fwECDFuncsDisabled* bit.

From a user's perspective a DCD function can be in one of only three states:

S1. The DCD function is active and is fully re-configurable.

S2. The DCD function is disabled.

S3. The DCD function is active and its configuration resources are locked.

The [EISA] specification defines a disable bit for each function and a lock bit for the entire device. As described in section 5 of this document, the EISA function disable bit is used to control re-configuration of DCDs in legacy ECUs. In order to provide the disable functionality (that was originally provided by the EISA function disable bit), the ECD defines the *fwECDFuncsDisabled* bit-map to indicate which DCD functions are disabled. The combinations of these three fields result in eight possible states not all of which are valid.

Before describing the eight states, we must ensure that the three fields are consistent with each other. If the *fwECDFuncsDisabled* bit is 1 (i.e., the function is truly disabled) the states of the device lock bit and the EISA function enable bits can be ignored in new ECUs. However, since the legacy ECUs do not have access to the *fwECDFuncsDisabled* bit, the CM or the ACFG BIOS needs to set the states of the device lock bit and the EISA function disable bit such that the legacy ECU interprets the fields correctly.

The correctness criteria are defined by the following rules:

1) *If the* fwECDFuncsDisabled *bit is 1, then the corresponding EISA function disable bit should be forced to 1.* If the EISA function disable bit is 0 (i.e., the legacy ECU interprets the function to be enabled), the legacy ECU will not allocate the resources for that function to other devices. This is an incorrect scenario because the function is truly disabled and is not using any resources that are described in its EISA function.

2) *If all the functions for a device do not have their EISA function disable bit set, then the device lock bit should not be changed from its current state.* The rule mentioned above changes the state of the EISA function disable bit for one function. This rule specifies the follow-up action for the device lock bit. It says that if there are any enabled functions in the device, the status of the lock bit should remain unchanged. In other words, if by disabling a function (as a result of rule 1 above), it turns out that all the functions of the device are disabled, then the lock bit can be reset. It is possible that some legacy ECUs ignore the lock bit if all the functions in that device are disabled, but this specification now requires that this bit be turned off to ensure consistency across all implementations. The resulting interpretation of the device lock bit is that the lock bit is set if at least one function in that device is not re-configurable.

3) *The device lock bit in conjunction with the EISA function disable bit defines the locked status of the function.* If the device lock bit is set, then the lock status of the function is derived from the status of the EISA function disable bit, i.e., if the EISA function disable bit is 0 then the function is locked (its resources cannot be re-configured) and if the bit is 1 then the function is unlocked (its resources can be re-configured). If the device lock bit is reset, the only valid state is for the function to be fully re-configurable (i.e., the EISA function disable bit should be 1).

The CM or the ACFG BIOS will always modify the states of the three fields if they do not conform to these correctness criteria. We thus arrive at the following state table:

Devic e lock bit	**EISA function disable bit**	**ECD disable bit**	**Description**
0	0	0	Invalid state because of rule 3). The CM or the ACFG BIOS will correct this by changing the device lock bit to 1.
0	0	1	Invalid state because of rule 1). The CM or the ACFG BIOS will correct this by changing the EISA function disable to 1.
0	1	0	**S1**; This is the normal state for DCD. This device can be fully re-configured as needed.
0	1	1	**S2;** The DCD function is disabled and will not be configured by ACFG BIOS or the CM.
1	0	0	**S3;** This DCD has locked resources.
1	0	1	Invalid state because of rule 1). The CM or the ACFG BIOS will correct this by changing the EISA function disable bit to 1. Following rule 2) the device lock bit may be changed to 0 depending on the status of other functions in the device.
1	1	0	**S1;** The status of the device lock bit is valid only if there are other functions in this device that have their EISA function disable bit set to 0. If this is not the case then this state is invalid and will be corrected by the ACFG BIOS or the CM according to rule 2). The function whose EISA disable bit is shown in the state table here is fully re-configurable.
1	1	1	**S2;** The status of the device lock bit is valid only if there is at least one other function in this device that has its EISA function disable bit set to 0. If that is not the case then this state is invalid and will be corrected according to rule 2) by ACFG BIOS or the CM.

Appendix C Detailed ESCD Data Structure Specification

```
/***
  NAME
    escdfmt.h (Extended Static Configuration Data)
  PURPOSE
    Defines the structures needed to access the ESCD
  NOTES
    1. The definitions here reflect additions to the packed Eisa format strucures. ESCD will be used to store
       configuration information both on ISA and EISA systems.
    2. ESCD definition differs slightly for ISA and EISA systems (refer to the ESCD documentation). ISA
       systems do not have a notion of slots. The byte reserved for slot number in the ESCD (for ISA systems) is
       used to create the abstraction of slots.
    3. Slot numbers 16-64 are referred to as Virtual Slots. Any peripheral, device or software that needs a
       configuration file and is not covered by other device types can be specified as a virtual device.
    4. Configuration information for PCI devices in an (E)ISA system,  is stored in virtual slots.
    5. Configuration information for PnP ISA devices in an (E)ISA system, is stored in slots one thru 15.
    6. The slot zero has special meaning and is reserved for motherboard configuration.
    7. It is a goal to use DCDs with old ECUs and achieve some amount of reconfigurability. This is achieved by
       using the function enable/disable bit for slightly different purpose. Refer to the ESCD documentation for
       more detailed description.
    8. PnP ISA and PCI devices have device specific information that cannot be completly represented by the
       EISA structures. For these devices only, this additional information will be stored in an EISA FreeFormat
       function that is alway disabled and is always the last function in the slot record.

    9. ESCD file checksum is a 16-bit logical (modulo 64K) sum of ASCII values in the ESCD file. ESCD file
           checksum must be calculated by the BIOS on a write to ESCD.

  HISTORY:
       Version 0.15, created June 1993.
       Version 0.16, updated July 15 1993.
       Version 0.99, updated July 22 1993.
       Version 0.99a,updated August 10 1993.
       Version 1.00, updated October 1 1993.
       Version 1.02, updated February 14 1994.
       Version 1.02A, updated May 1994.

***/

#ifndef   _ESCD
#define   _ESCD

#ifndef   BYTE
typedef unsigned char       BYTE;
typedef unsigned short      WORD;
typedef unsigned long       DWORD;
typedef long                LONG;
#endif
/* end of data types */

/***********************************************************************
                    Standard EISA format definitions
***********************************************************************/
                    /* Bytes #0 and #1 of ID and Slot Information */
typedef struct
{
        BYTE    bDupCFGNumId :4;        /*Byte # 0:  Bits 0-3  Numeric id for duplicate CFG filenames
                                          0000 - No duplicate CFG filenames
```

```
                                                0001 - 1st duplicate(1ACE0105)
                                                .....
                                                1111 - 15th duplicate(FACE0105) */
        BYTE    bSlotType          :2;        /* Byte # 0:  Bits 4-5
                                                 00 - Expansion Slot
                                                 01 - embedded slot
                                                 10 - virtual slot
                                                 11 - reserved(0)          */
        BYTE    bIDReadable        :1;        /* Byte # 0:  Bit  6
                                                 0 - ID reabable
                                                 1 - ID not reabable       */
         BYTE   bDupIDPresent      :1;        /* Byte # 0:  Bit 7
                                                 0 - no duplicate ID present
                                                 1 - duplicate ID present  */

        BYTE bBrdEISAEnableSupp: 1;    /* Byte # 1: Bit 0 - board can be disabled = 1 */
        BYTE bBrdIochker        : 1;   /* Byte # 1: Bit 1 - IOCHKERR supported = 1 */
        BYTE bBrdOrEntryLck     : 1;   /* Byte # 1: Bit 2 - board or entries locked = 1 */
        BYTE bIdSlotResvrd      : 3;   /* Byte # 1: Bit 3-5 - reserved */
        BYTE bIdSlotNoCfgFile   : 1;   /* Byte # 1: Bit 6 - Board doesn't have/need cfg file = 1 */
        BYTE bBrdConfgStat      : 1;   /* Byte # 1: Bit 7 - config is completed = 0 */
                                       /*                 - config is not completed = 1 */
} EISAIDSLOTINFO;

                    /* Function Information Byte #0 */
typedef struct
{
  BYTE bTypeSubTypeEntry      : 1;     /* Bit 0 - type subtype data = 1 */
  BYTE bMemoryEntry           : 1;     /* Bit 1 - mem entry data = 1 */
  BYTE bIrqEntry              : 1;     /* Bit 2 - IRQ data = 1 */
  BYTE bDmaEntry              : 1;     /* Bit 3 - DMA entry data = 1 */
  BYTE bPortRangeEntry        : 1;     /* Bit 4 - port range data = 1 */
  BYTE bPortInitEntry         : 1;     /* Bit 5 - port init data = 1 */
  BYTE bFreeFormEntry         : 1;     /* Bit 6 - free form data = 1 */
  BYTE bEISAFuncDisabled      : 1;    /* Bit 7 - enabled = 0, disabled = 1 */
} EISAFUNCENTRYINFO;

                    /* Memory Info struct Bytes #0-6 */
typedef struct
{
  BYTE bMemRdWr               : 1;   /* Bit 0 - 0 = ROM, 1 = RAM */
  BYTE bMemCached             : 1;   /* Bit 1 - 0 = not cached */
  BYTE bMemChType             : 1;   /* Bit 2 - 1 = WB cache, 0=WT cache */
  BYTE bMemType               : 2;   /* Bits3-4 -00=sys, 01=exp, 10=vir,11=oth */
  BYTE bMemShared             : 1;   /* Bit 5 - 0 not=shared */
  BYTE bMemReserved1          : 1;   /* Bit 6 - 0 = reserved */
  BYTE bMemMoreEntries        : 1;   /* Bit 7 - last entry = 0, more = 1 */
         /* Mem data size byte */
  BYTE bMemDataSize           : 2;   /* Bit 0-1 -00=byte,01=word,10=dwrd,11=rsv*/
  BYTE bMemDecodeSize         : 2;   /* Bit 2-3 -00=20,01=24,10=32,11=rsv*/
  BYTE bMemReserved2          : 4;   /* Bit 4-7 -0 = reserved */
         /* memory start addr */
  BYTE bMemStartAddr0;             /* LSByte (divided by 0x100) mem start */
  BYTE bMemStartAddr1;             /* Middle Byte memory start */
  BYTE bMemStartAddr2;             /* MSByte memory start */
         /* memory size */
  BYTE bMemSize0;                   /* LSByte (divided by 0x400) mem size */
  BYTE bMemSize1;                   /* LSByte=MSByte=0 means 64MB */
```

```
} EISAMEMORYINFO;

                    /* IRQ Info struct Bytes #0-1 */
typedef struct
{
 BYTE bIrqNumber          :4;    /* Bit 0-3 - IRQ Number */
 BYTE bIrqRsvrd           :1;    /* Bit 4 - must be 0 */
 BYTE bIrqTrigger         :1;    /* Bit 5 - 0=Edge , 1=Level */
 BYTE bIrqType            :1;    /* Bit 6 - 0=Non-shared, 1=Sharable */
 BYTE bIrqMoreEntries     :1;    /* Bit 7 - 0=Last Entry, 1=More entires follow */

 BYTE bIrqReserved;       /* Reserved (set to 0) */
} EISAIRQINFO;

                    /* DMA Info struct Bytes #0-1 */
typedef struct
{
 BYTE bDmaNumber          :3;    /* Bits 0-2 - DMA Number(0-7) */
 BYTE bDmaReserved1       :3;    /* Bits 3-5 Reserved (set to 0) */
 BYTE bDmaType            :1;    /* Bit 6 - 0=Non-Sharable, 1=Sharable */
 BYTE bDmaMoreEntries     :1;    /* Bit 7 - 0=Last Entry, 1=more entires follow */

 BYTE bDmaReserved2       :2;    /* Bit 0-1 Reserved (set to 0) */
 BYTE bDmaTransferSize    :2;    /* Bits 2-3
                                      00 = 8bit  transfer
                                      01 = 16bit transfer
                                      10 = 32bit transfer
                                      11 = 16bit transfer with byte count */
 BYTE bDmaTiming          :2;    /* Bits 4-5
                                      00 = Isa Compatible timing
                                      01 = Type "A"
                                      10 = Type "B"
                                      11 = Type "C"(Burst) */
 BYTE bDmaReserved3       :2;    /* Bits 6-7 Reserved (set to 0) */
} EISADMAINFO;

                    /* I/O ports Info struct Bytes #0-2 */
typedef struct
{
 BYTE bPortCount          :5;    /* Bit 0-4 Number of Ports
                                    0000 = 1Port
                                    0001 = 2Sequential Ports (and so on)
                                    1111 = 32Sequential Ports */
 BYTE bPortRsvrd          :1;    /* Bit 5 Reserved (set to 0) */
 BYTE bPortShared         :1;    /* Bit 6 0=Non-shared, 1=Sharable */
 BYTE bPortMoreEntries    :1;    /* Bit 7 - 0=Last Entry, 1=More entires follow */

 WORD wPortAddr;          /* I/O Port Address */
} EISAPORTINFO;


                    /* Init ports Info struct Bytes #0-2 */
typedef struct
{
 BYTE bAccessType         :2;    /* Bit 0-1
                                    00 - Byte address(8-bit)
                                    01 - Word address(16-bit)
                                    10 - Dword address(32-bit)
```

```
                                        11 - Reserved(0) */
 BYTE bPortMaskSet              :1;   /* Bit 2
                                        0 - Write value to Port(no mask)
                                        1 - Use mask and value */
 BYTE bInitReserved             :4;   /* Reserved(0) */
 BYTE bMoreEntries              :1;   /* 0 = Last Entry
                                        1 = More entries follow */
} EISAINITDATA;

                /* EISA free format data definition */
typedef struct
{
 BYTE bDataSize;        /* Length of following data block */
 BYTE abData[203];      /* 203 bytes */
} EISAFREEFORMDATA;

/* eisa slot function config 320 bytes structure layout definition   */
typedef struct
{
 BYTE                  bCompBrdID1;          /* first byte of compressed board ID  */
 BYTE                  bCompBrdID2;          /* second byte of compressed board ID */
 BYTE                  bCompBrdID3;          /* third byte of compressed board ID  */
 BYTE                  bCompBrdID4;          /* forth byte of compressed board ID  */
 EISAIDSLOTINFO        sIDSlotInfo;          /* bit specific slot ID and slot info */
 BYTE                  bCFGMinorRevNum;      /* minor revision of CFG file extension*/
 BYTE                  bCFGMajorRevNum;      /* major revision of CFG file extension*/
 BYTE                  abSelections[26];     /* 26 bytes of selection information */
 EISAFUNCENTRYINFO     sFuncEntryInfo;       /* Func status and resources stat */
 BYTE                  abTypeSubType[80];    /* 80 character type/subtype field   */
 union
  {
   struct
        {
         EISAMEMORYINFO    asMemData[9];     /* 63 bytes mem cfg data */
         EISAIRQINFO       asIrqData[7];     /* 14 bytes IRQ config data */
         EISADMAINFO       asDmaData[4];     /* 8 bytes DMA channel info */
         EISAPORTINFO      asPortData[20];   /* 60 bytes I/O port info */
         BYTE              abInitData[60];   /* 60 bytes init. data */
        } sResData;
   EISAFREEFORMDATA        sFFData;          /* Free format data */
  } uFuncData;
} EISAFUNCCFGINFO;

/***********************************************************************
                End of Standard EISA format definitions
***********************************************************************/


/***********************************************************************/
/***********************************************************************/
/* Layout of the whole storage for the ESCD.img file              */
/*
```

1) Escd_CFGHDR. This contains the ESCD size, signature, version#, and the number of slot entries.
2) This is followed by board records that contain a board header and board data. Board header contains the size of the board record and the slot number for the board. The board header is specific to ISA systems only.
3) The packed data for each slot is preceeded by ESCDBrdHdr that contains the size and the slot# for the slot data that immediately follows.

4) EISA format data for slot zero, the Mother-board: data for functions 0-n describing MB resources
5) EISA data for slots 1-15 describe EXP EISA and ISA boards: data in standard format for functions 0-n corresponding to devices associated with the expansion boards.
6) ESCD data for slots 1-15 describing the PnP ISA boards:
   - data for functions 0-n corresponding to devices on the expansion board. Unlocked PnP ISA devices are described as disabled functions; locked PnP devices on the board are enabled functions.
   - disbled function n+1 describing extentions specific to PnP board type. The data uses free format spec.
7) ESCD data for slots 16-64 (Virtual slots) describe the PCI devices, one PCI board (device) per one slot:
   - one or more standard EISA function(s) corresponding to the PCI function(s) 0-7 for the PCI device that is located at Bus#,Dev# and Fun#0 address in the system. If the configuration for this device is unlocked, the standard EISA format function(s) will be disabled; locked PCI function(s)will be enabled.
   - a last function, the ECD describing the PCI specific information for the PCI function(s) 0-7.
8) There is a checksum at the end of the storage.

```
*/
/**********************************************************************/

/* The following structures describe the ESCD extensions.            */

typedef struct
{
 WORD        wEScdSize;              /* Total Size of File/NVRAM */
 DWORD        dSignature;            /* Initialized to "ACFG" */
 BYTE        bVerMinor;              /* Minor #, should be >= 0 */
 BYTE        bVerMajor;              /* Major #, should be >= 2 */
 BYTE        bBrdCnt;
 BYTE        abEscdHdrReserved[3];
} ESCD_CFGHDR;

typedef struct
{
 WORD        wBrdRecSize;            /* Including this word */
 BYTE        bSlotNum;
 BYTE        bEscdBHdrReserved;
} ESCD_BRDHDR;

#define  ESCD_SIGNATURE       0x47464341       /* ACFG characters */

        /* Free format last funct Board Header ecd extensions */
typedef struct
{
                          /* Total size of 16 bytes */
 DWORD       dSignature;              /* Initialized to "ACFG" */
 BYTE        bVerMinor;               /* should be >= 00 */
 BYTE        bVerMajor;               /* Must be set to 0x02 */
 BYTE        bBrdType;                /* Board Type as in CM defintion */
                                      /* 0x01=isa, 0x02=eisa, 0x04=pci */
                                      /* 0x08=pcmcia, 0x10=PnP Isa, 0x20=mca */
 BYTE        bEcdHdrReserved1;        /* Reserved */
 WORD        fwECDFuncsDisabled;       /* 16 PnP functions disabled bit-map */
 WORD        fwECDFuncsCfgError;      /* 16 PnP functions config error status bit-map */
 /* This reserved field will now be used  BYTE          abEcdHdrReserved2[4];    Reserved */
 WORD        fwECDFuncsCannotConfig;         /* 16 PnP funct bit-map to indicate  *
                                              /*   if the device is reconfigurable*/
                                             /* For each bit 0 - Reconfigurable 1- Not reconfigurable */
  BYTE       abEcdHdrReserved[2];            /* Reserved */
} ECD_FREEFORMBRDHDR;
```

```
                    /* Free Fmt PCI device identifier and data */
typedef  struct
{
 BYTE            bBusNum;                  /* PCI Bus Number (0-255) */
 BYTE            bDevFuncNum;              /* PCI defined Device (0-31) and Func 0-7) Number */
                                           /* Device # in bits 7:3, Function # in bits 2:0 */
 WORD            wDeviceId;                /* PCI device ID */
 WORD            wVendorId;                /* PCI vendor ID */
 BYTE            abPciBrdReserved[2];      /* Reserved */
} ECD_PCIBRDID;

                    /* Free Fmt PnP ISA board identifier */
typedef  struct
{
 DWORD           dVendorId;                /* PnP ISA vendor ID, 4 char */
 DWORD           dPnPSerialNum;            /* Board/Device serial # identifier */
} ECD_PNPBRDID;



          /* PnP ISA ECD extention function, a last function per board */
typedef struct
{
 WORD            wFuncSize;                /* Size set to 28 */
 BYTE            bSelectionSize;           /* initialize to 1 */
 BYTE            bSelectionData;           /* initialize to 0 */
 BYTE            bFuncInfo;                /* FreeFormat, disabled bit set (set to 0xC0) */
 BYTE            bFreeFormSize;            /* Size of following free fmt data, excl this byte Size set to 24: sizeof
                                              ECD_FREEFORMBRDHDR + PnP specific data */
 ECD_FREEFORMBRDHDR sFFBrdHdr;             /* sizeof struct = 16 bytes */
 ECD_PNPBRDID            sPnPBrdId;        /* sizeof struct =8 bytes */

} PNPFREEFORMFUNC;

          /* PCI ECD extention function, a last function per board/device */
typedef struct
{
 WORD            wFuncSize;                /* set to min of 28 for single PCI function */
                                           /* and to 86 for eight functions PCI card */
 BYTE            bSelectionSize;           /* initialize to 1 */
 BYTE            bSelectionData;           /* initialize to 0 */
 BYTE            bFuncInfo;                /* FreeFormat, disabled bit set (set to 0xC0) */
 BYTE            bFreeFormSize;            /* Size of following free fmt data, excl this byte
                                              Size set to max of 80: sizeof ECD_FREEFORMBRDHDR
                                             + PnP specific data  allowed intry is 24 or 32 or 40 .. 80 */
 ECD_FREEFORMBRDHDR sFFBrdHdr;             /* sizeof struct = 16 bytes */
 ECD_PCIBRDID      sPCIBrdId[8];            /* sizeof struct = Maximum of 8*8 bytes */
                                           /* There will be only one sPCIBrdId entry for each function on
                                              a multi-function  PCI card */
} PCIFREEFORMFUNC;

/*                  End of the ECD extensions.                 */
/**************************************************************************/
#endif
/* end of _ESCD definition */
```

Appendix D ESCD Access Functions Return Codes

The following table defines the return codes for the Plug and Play BIOS functions dealing with ESCD access.

Return Code	Value	Description
SUCCESS	00h	Function completed successfully
FUNCTION_NOT_SUPPORTED	81h	The function is not supported on this system.
ESCD_IO_ERROR_READING	55h	The system BIOS could not read or write the Extended System Configuration Data (ESCD) from nonvolitale storage
ESCD_INVALID	56h	The system does not have a valid Extended System Configuration Data (ESCD) in nonvolitale storage.
ESCD_BUFFER_TOO_SMALL	59h	The memory buffer passed in by the caller was not large enough to hold the data to be returned by the system BIOS.
ESCD_NVRAM_TOO_SMALL	5Ah	All of the ESCD cannot be stored in the NVRAM storage available on this system.