As shown in figure 11-1, the CIS is read by PC Card client drivers during card initialization to determine the configuration options supported by the card. The PC Card client accesses the CIS via card and socket services. Once the card type and resource requirements have been read from the CIS, the PC Card client driver programs the HBA and configures the PC Card, again via card and socket services. No further access is typically made to the CIS after the card has been initialized. The memory or I/O device can now be accessed via the host expansion bus, as would any other expansion device. Note that the CIS is only accessed by programs that are PCMCIA aware. Most application programs have no knowledge that they are accessing devices implemented in PC Card packages.

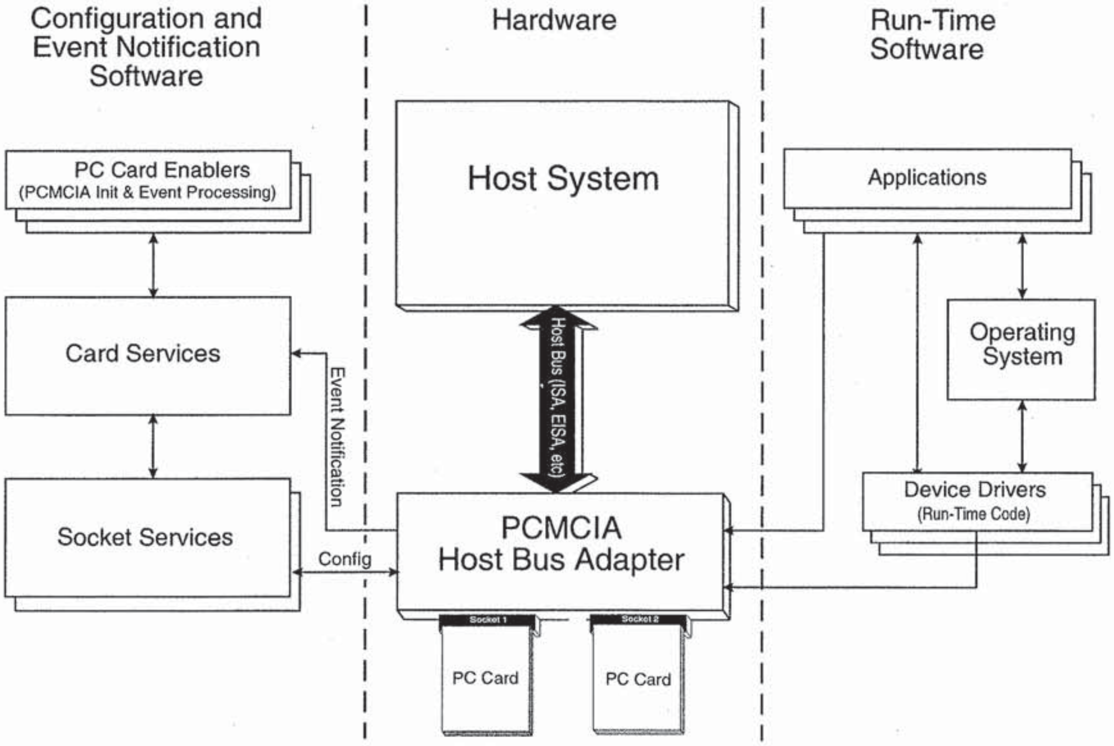

Figure 11-1. PCMCIA Software Flow

The Card Information Structure (CIS)

The CIS is mapped into the attribute memory address space starting at address zero as illustrated in figure 11-2. The CIS consists of a linked list of data blocks, or tuples, that describe the function and characteristics of a PC Card. Configuration software accesses this data to determine the characteristics and configuration requirements of a given PC Card. Tuples are identified by a unique code which in the first byte of each tuple.

Note that CIS data is mapped only to even locations within the attribute address space; thus, information is returned only on the lower data path (D7:D0). This simplifies card designs for accommodating eight-bit host systems that connect only to the lower data path.

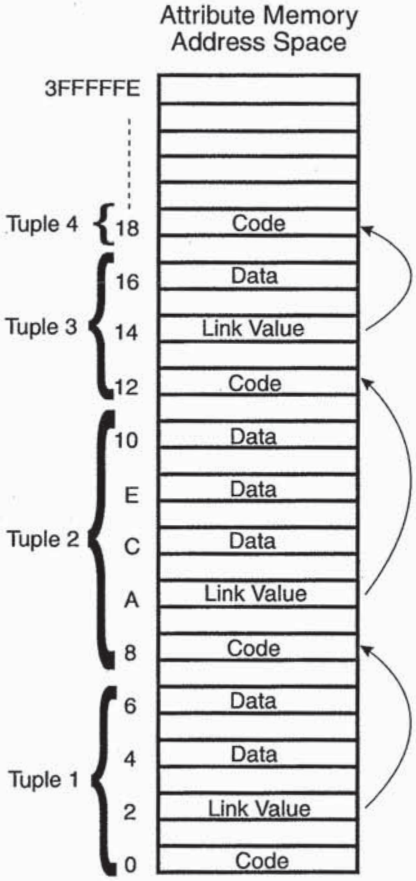

Figure 11-2. Example CIS Layout Consisting of a Linked List of Four Tuples

Tuples

A tuple is defined in Webster's Ninth New Collegiate Dictionary as a "set of elements." A tuple in PCMCIA terminology refers to a defined set of data items that characterize some facet of a PC Card. The PCMCIA standard specifies tuples intended to be used by PC Card designers for providing information about their device. Tuples provide information such as the PC Card's device speed and size. Tuple information is most often used by configuration software to determine the configuration requirements of the card. However, other tuples provide information that can be used by utility programs and applications to ascertain additional capabilities of the card.

Tuple Format

All tuples have a general format defined by PCMCIA (refer to table 11-1). The first one-byte element (entry 0) of every tuple is a tuple *type code* that defines the tuple's function. The second entry (entry 1) of every tuple is a one-byte link value (in hex) that specifies the number of additional bytes remaining in the tuple. The number and definition of these remaining bytes depends on the type of tuple.

Table 11-1. Basic Tuple Format

Byte	Standard Tuple Format	
0	TPL_CODE	Tuple type code (XXh). See table 11-7 for tuple codes.
1	TPL_LINK	Link to next tuple (number of bytes (in hex) remaining in tuple).
n	TPL_DATA	Tuple specific data block (definition, format and length defined by individual tuples).

The CIS consists of a linked list of tuples. Each tuple specifies a link value that identifies the start of the next tuple. Processing software can read the CIS entries and interpret the meaning of the tuples that contain configuration information for the PC Card.

The exact set of tuples incorporated into the CIS depends primarily on the type of card and its capabilities. For example, the Device Information Tuple may contain all the information needed to determine the resources required by a simple SRAM card, while other card types might require numerous tuples to define the configuration of the card.

A Sample Tuple

Consider the information provided by the Device Information tuple shown in table 11-2. This tuple defines a 100ns SRAM card containing 1MB of memory. The first byte within the tuple contains a value of 01h defining this tuple as a device information tuple. The second byte (03h) specifies the number of bytes remaining in the tuple. The device information tuple contains two bytes within the tuple's data area. One that defines the memory card type, speed, size, and whether the write-protect switch affects the range of memory being defined, and one that defines the size of the memory device.

The memory card device type is specified in the tuple as a hexadecimal code value. In this example, the device code is a 6h. As shown in table 11-3, a device code of 6h identifies the card as SRAM. Similarly, the SRAM's cycle time is specified with a speed code of 4h. This indicates a device speed of 100ns as shown in table 11-4. The size of the device can be determined by reading the unit size code and multiplying the unit size by the number of units specified. The unit size code of 5h, specifies memory banks of 512KB (refer to table 11-5) and the number of units field contains a 1h, indicating two memory units are implemented for a total size of 1MB. Finally, the tuple is terminated by FFh. This tuple includes a termination byte because the data within the tuple can vary in length (i.e. more than one memory device can be described by the Device Information tuple). The termination bytes make it easier for parsing software to recognize the end of variable length tuples. Tuples that do not vary in length do not define a termination byte.

Table 11-2. Example Device Information Tuple for an SRAM Card

Byte	Value	Device Information Tuple
0	01h	Tuple Code (01h)
1	03h	Link to next tuple (3h)
2	64h	*Device Type=bits 7:4 (6h); WP=bit 3 (0);Speed=bits 2:0(4h)
3	0Dh	*Device Size= # of units [bits 7:3 (1)] times unit size [bits 2:0 (5h)]
4	FFh	FFh (marks end of device info field)

* Refer to the following tables for an interpretation.

Table 11-3. Device Type Codes

Code	Name	Meaning
0	DTYPE_NULL	No memory device. Generally used to designate a hole in the address space. If used, speed field should be set to 0h.
1	DTYPE_ROM	Masked ROM
2	DTYPE_OTPROM	One-time programmable PROM
3	DTYPE_EPROM	UV EPROM
4	DTYPE_EEPROM	EEPROM
5	DTYPE_FLASH	Flash EPROM
6	DTYPE_SRAM	Static RAM (JEIDA has Nonvolatile RAM)
7	DTYPE_DRAM	Dynamic RAM (JEIDA has Volatile RAM)
8-Ch		Reserved
Dh	DTYPE_FUNCSPEC	Function-specific memory address range. Includes memory-mapped I/O registers, dual-ported memory, communication buffers, etc., which are not intended to be used as general-purpose memory.
Eh	DTYPE_EXTEND	Extended type follows.
Fh		Reserved

Table 11-4. Device Speed Codes

Code	Name	Meaning
0h	DSPEED_NULL	Use when device type = null
1h	DSPEED_250NS	250 nsec
2h	DSPEED_200NS	200 nsec
3h	DSPEED_150NS	150 nsec
4h	DSPEED_100NS	100 nsec
5h-6h		(Reserved)
7h	DSPEED_EXT	Use extended speed byte.

Table 11-5. Unit Size Codes

Code	Units
0	512 bytes
1	2 K
2	8 K
3	32 K
4	128 K
5	512 K
6	2 M
7	Reserved

The Configuration Table

I/O devices require that the CIS contain a configuration table that is not required by memory cards. This table consists of multiple entries each of which describes a set of configuration options that the PC Card needs for normal operation. A comparison can be made between each configuration table entry and each possible switch and jumper setting required when configuring an ISA card. Each configuration table entry reflects the possible resource combinations that the PC Card can be configured for.

The Configuration Entry Tuple

Figure 11-3 illustrates a CIS that contains a configuration table. Directly preceding the configuration table is the configuration tuple that specifies which configuration registers are implemented by the PC Card and where they are mapped within attribute memory address space. The configuration tuple also specifies the index number of the last entry within the configuration table. As illustrated in figure 11-3, the configuration table consists of a series of configuration table entry tuples (CFTABLE_ENTRY). Each entry contains up to seven data structures that describe operational characteristics of the PC Card. These structures include:

1. **A power description byte** — the power parameters specified within this structure may apply to Vcc only, Vcc and Vpp1 and Vpp2 (Vpp1=Vpp2), or separately to Vcc, Vpp1, and Vpp2. The specific power parameters de-

scribed by the structure are also selectable as defined by the parameter selection byte within the power description structure.

2. **Configuration timing information** — this structure defines the maximum length of time that the PC Card will keep READY deasserted and the maximum duration of the WAIT# signal.
3. **I/O address space description** — defines up to sixteen ranges of I/O address space required by the PC Card for this configuration. The structure defines the exact base I/O address and the number of address locations within the range
4. **Interrupt request description** — specifies the system interrupt request line required for this configuration. A single IRQ can be specified or a group of IRQs can be defined, any of which will satisfy the configuration requirements. Also included in the description is information that defines the deliver mode (level or pulse), whether interrupt sharing is supported, and alternative interrupt signal definitions (i.e. NMI, I/O check, bus error, vendor specific interrupt).
5. **Memory address space description** — specifies up to eight ranges of memory address space required for this configuration. Both the Host processor address and the PC Card address can be specified. When both the host and PC Card address are the same, no address translation is required since the host address is directly mapped into the common memory address space. If no host address range is specified, then any range of host address space can be used and mapped by the HBA to the specified range within common memory address space. A base address and range value are specified for each block of addresses needed for this configuration.
6. **Miscellaneous information structure** — contains information regarding support for special features required by this configuration. Two bytes are defined by the PC Card standard. The first byte identifies the PC Card's support for power down (for power management software), whether the SPKR# pin is used, and the number of identical PC Cards that are supported for the max twins cards option (e.g. support for multiple ATA drives). The second byte defines support for DMA, including the DMA transfer size and specifies which pin the PC Card uses for DREQ#.
7. **Subtuple information** — permits definition of additional information relating to this configuration. Subtuples are included as extensions to the configuration table entry tuple and may include information such as the operation system for which the configuration was intended and the physical device being implemented in this configuration.

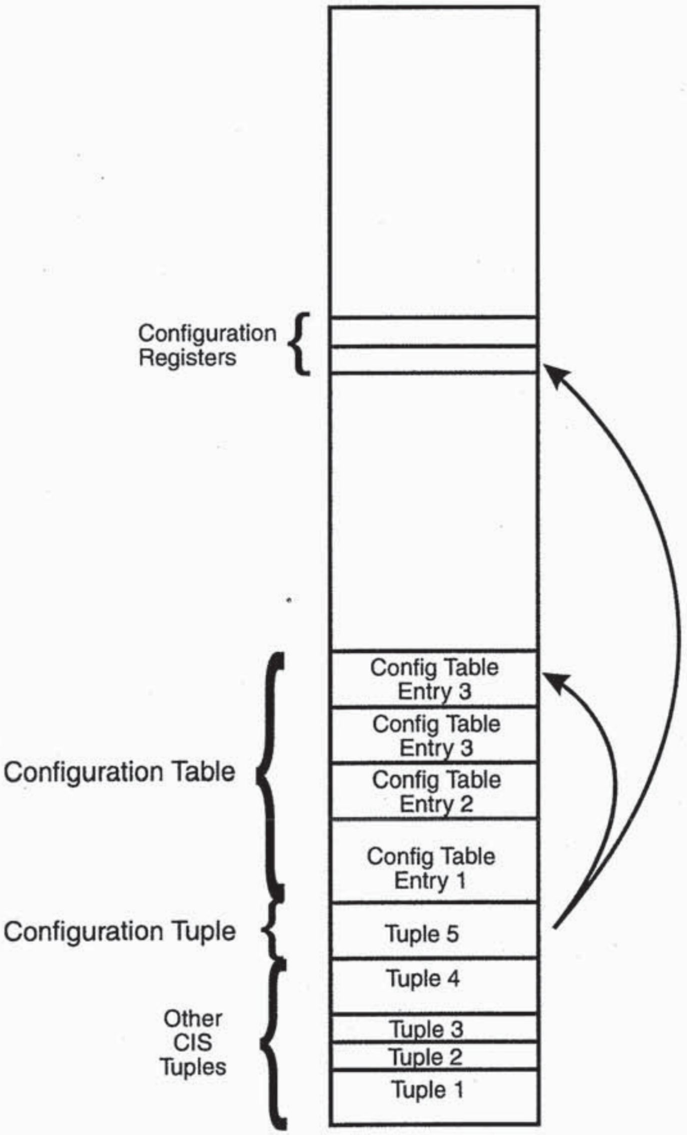

Figure 11-3. The Configuration Table Consists of a Number of Entries, Describing the Configuration Options Supported by the PC Card.

Table 11-6 shows the format of the configuration table entry tuple. The actual structures that are implemented within this tuple are specified by the feature selection byte.

Table 11-6. Format of the Configuration Table Entry Tuple

Byte	Name	Description of Entry
0	TPL_CODE	Configuration Entry tuple code (CISTPL_CFTABLE_ENTRY, 1Bh)
1	TPL_LINK	Link to next tuple (n-1, {2 minimum})
2	TPCE_INDX	Configuration table index byte — this byte contains the index number of the entry, specifies whether the interface byte will follow, and specifies whether this entry is a default entry or not.
..	TPCE_IF	Interface description byte — this field is present only when the interface bit of the Configuration-table index byte is set
..	TPCE_FS	Feature selection byte indicates the optional structures present
..	TPCE_PD	Power description structure
..	TPCE_TD	Configuration timing information structure
..	TPCE_IO	I/O address space description structure
..	TPCE_IR	Interrupt request description structure
..	TPCE_MS	Memory address space description structure
..	TPCE_MI	Miscellaneous information structure
..n	TPCE_ST	Additional information about the configuration in subtuple format

Interpreting the Configuration Table

When parsing software (usually a card services client driver) processes an entry within the configuration table, it must determine if the resources specified are available. (Refer to the chapter entitled, "Client Drivers" for a discussion of resource acquisition.) If all resources that have been requested are available then the configuration is satisfied and no additional configuration table entries need be evaluated. If however, one or more of the resources required to satisfy the configuration are not available, then parsing software must evaluate subsequent entries in an attempt to find alternative system resources that will satisfy the PC Card's configuration requirements.

The first entry within the configuration table is typically specified as a default entry. Default entries indicate that all configuration information specified within the entry should be retained even in the event that the full configuration was not satisfied. For example consider the configuration table illustrated in figure 11-4. The first entry is a default entry that specifies a power structure, a configuration timing structure, an I/O address space structure, an interrupt request structure and a miscellaneous information structure. Assume that parsing software was able to satisfy all configuration information specified by this entry except the interrupt request line. Software then proceeds in the following manner:

1. Since this is a default entry, all resources successfully acquired are retained. This eliminates the need to re-specify all the parameters that apply globally to the card's configuration regardless of which I/O address space and IRQ line is assigned to the card. In this example, since the entire configuration was not satisfied, parsing software proceeds to the next entry, attempting to find alternative resources that the PC Card can use.

2. Assume that entry 2 is not a default entry and contains only an I/O address structure and IRQ structure. Parsing software recognizing a nondefault entry knows it must successfully acquire all configuration options specified, and if unable to do so must release the partial configuration by returning the resources previously acquired. Furthermore, since a pair of resources is being requested, the parsing software recognizes that the I/O address space acquired when attempting to satisfy the previous default entry must be released in favor of the new I/O address space and IRQ lines specified by this entry. If both configuration options are acquired successfully, then the configuration is completed. If not, the incomplete configuration is released and parsing software proceeds to the next entry.

3. Assume that entry 3 is not a default entry and contains another set of I/O addresses and another IRQ line. Once again parsing software attempts to acquire both resources, and if not successful must release any resource acquired and proceed to the next entry. As before, if both are acquired the configuration is complete.

4. Entry 4 is the last configuration entry and contains the final I/O address space and IRQ options for configuring the PC Card. If these resources cannot both be acquired, then the parsing software must report to the user that the card cannot be configured.

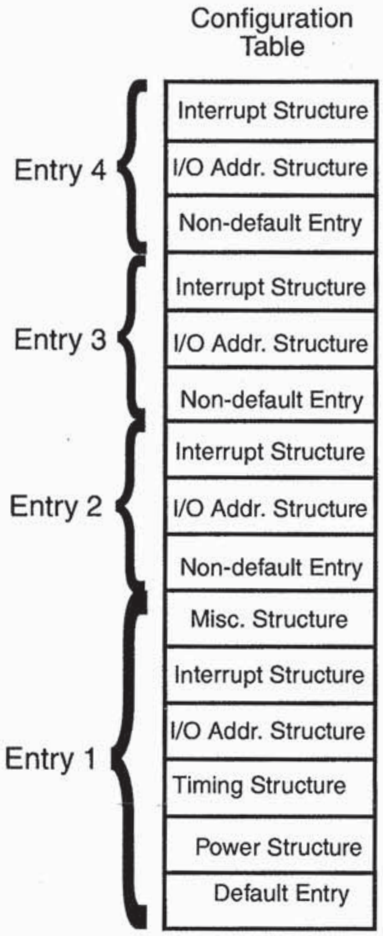

Figure 11-4. Example Configuration Table with One Default and Four Non-Default Entries

Once parsing software has obtained the configuration resources from the system it must configure the HBA and PC Card so that they respond to the resources. Parsing software uses the index number of the configuration table entry that specifies the successful configuration when configuring the PC Card. The index number is written into the PC Card's configuration option register, telling the PC Card which set of configuration options were successfully acquired.

Multiple Function PC Cards

Multi-function PC Cards require a separate CIS and configuration register set for each function within the card. As illustrated in figure 11-5, a global CIS is required when implementing a multi-function PC Card. The global CIS contains a long link multi-function tuple (LONGLINK_MFC) that lists the entry points of each function's CIS. The first entry within the target CIS must contain a LINKTARGET tuple to verify the correct start address specified by the LONGLINK_MFC tuple. Note that the configuration registers used by each function are identified by the configuration tuple within each CIS.

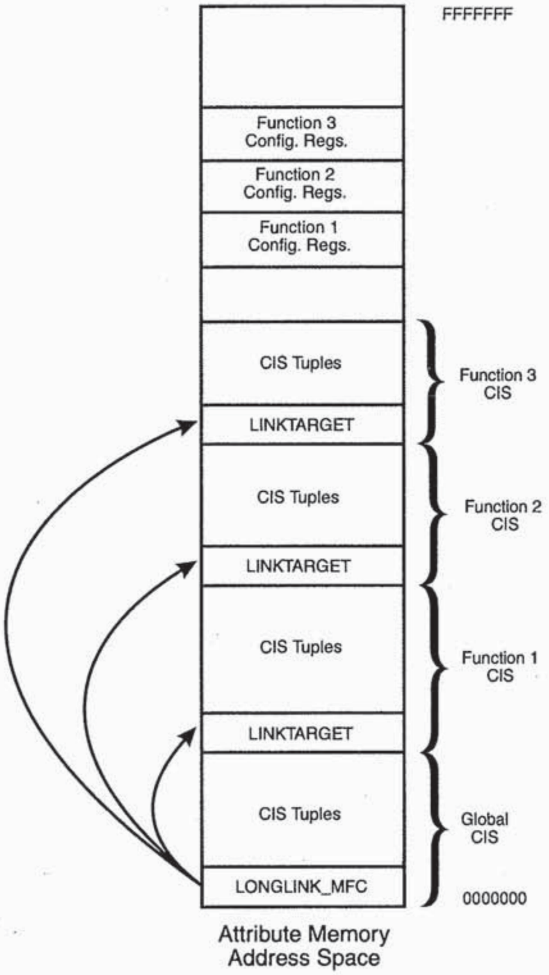

Figure 11-5. Configuration Table Structure Used by a Triple-Function PC Card

Devices Commonly Used for the CIS

Both ROM and FLASH are commonly used to implement the CIS. The clear advantage of FLASH is that the CIS can be easily updated. The CIS is quite small (usually not larger than 1 KB) and in the case of SRAM cards it can be as few as six bytes.

CIS Access Timing

Attribute memory (the CIS and configuration registers) must be accessed by card enabling software to determine the type of card installed and how it should be configured. Attribute memory is accessed by the HBA based on a default 300ns cycle time. This ensures that the CIS can be accessed regardless of the speed of other devices within the card. See the chapter entitled "The Memory-Only Interface" for details regarding attribute memory accesses.

Summary of Layer 1 Tuples

Table 11-7 lists the tuples that are currently defined by the PCMCIA specification for the CIS (layer 1 of the metaformat). Tuples are also defined for layers 2 and 3, but are not discussed here. Refer to the PCMCIA specification for details.

Table 11-7. Tuples defined for Compatibility Layer One (CIS)

Code (h)	CISTPL_NAME	Description and Purpose
00	NULL	Null Control tuple — Used as a place holder. Ignored by tuple processing software.
01	DEVICE	Device Information for Common Memory — Contains information about the card's common memory devices, including speed, type, write protect and size.
02-05	Reserved	Reserved for future versions of the device information tuple or for CardBus implementations.
06	LONGLINK_MFC	Long-Link for Multi-Function Card — Specifies the number of functions within this PC Card (i.e. sets of configuration registers) and defines the location of each function-specific CIS within the card.
07-0F	Reserved	Reserved for future versions or for CardBus tuples.

Table 11-7 Tuples Defined for Compatibility Layer One (Continued)

Code (h)	CISTPL_NAME	Description and Purpose
10	CHECKSUM	Checksum Control — Provides a means for verifying the contents of the CIS in memory. Multiple checksum control tuples can be implemented within a single CIS.
11	LONGLINK_A	Long-Link Control to Attribute Memory — Specifies the continuation of a tuple string to a location in **attribute** memory, beyond the limits of the 1 byte link field. The entry point specified must contain a Link Target tuple.
12	LONGLINK_C	Long-Link Control to Common Memory — Specifies the continuation of a tuple string to a location in **common** memory, beyond the limits of the 1 byte link field. The entry point specified must contain a Link Target tuple.
13	LINKTARGET	Link Target — Verifies the continuation of a valid tuple string. The Link Target tuple is the first tuple at the entry point specified by a Long-Link tuple.
14	NO_LINK	The No Link tuple tells processing software that when the end of the current tuple chain is reached (i.e. the Termination Tuple has been detected) that no more tuples exist in the chain to be processed. (See Termination tuple — code FFh for more information.)
15	VERS_1	Level 1 (also layer 1)Version identifies the PCMCIA compliance level of the CIS (also called the compatibility layer or metaformat layer one). Following the Version information, production information is provided in a series of ASCII strings each ended by zero (Called ASCIIZ).
16	ALTSTR	Alternate Language String — Includes additional languages for ASCII strings used in the product information tuple (code 15h). Also used for the Level 2 Version / Product Information tuple (code 40h).
17	DEVICE_A	Device Information to Attribute Memory — Contains information about the card's attribute memory devices, including speed, type, write protect and size. (optional)
18	JEDEC_C	Specifies the JEDEC (Joint Electronic Device Engineering Council) manufacturer and programming algorithm required by programmable devices listed in the device information tuple (01h) for common memory. Entries in the JEDEC identifier tuple have a one-to-one correspondence to the entries in the device information tuple.

Table 11-7 Tuples Defined for Compatibility Layer One (Continued)

Code (h)	CISTPL_NAME	Description and Purpose
19	JEDEC_A	Specifies the JEDEC (Joint Electronic Device Engineering Council) manufacturer and programming algorithm required by programmable devices listed in the device information tuple (17h). Entries in the JEDEC identifier tuple have a one-to-one correspondence to the entries in the device information tuple.
1A	CONFIG	Configuration tuple — Specifies the address of the configuration registers in attribute memory space and specifies which configuration registers are implemented in the card. Also identifies the last configuration entry within the configuration table, and provides a method of appending subtuples to the basic configuration tuple. Subtuples define additional information related to the card's configuration. Subtuple codes 80h-BFh are reserved for vendor specific items, while C0h- FEh are reserved for future PCMCIA standard definition. Currently, only the Custom Interface subtuple has been defined.
1B	CFTABLE_ENTRY	Configuration Table Entry — Provides configuration options supported by the card. Each configuration table entry provides additional configuration options. The entire set of configuration entries within the CIS is called the configuration table.
1C	DEVICE_OC	Other Conditions Device Information (common memory) — Specifies the characteristics of devices mapped in the common memory address space, when operating under conditions other than the defaults. For example, if the card is a dual voltage card (operates at both 5 volts and 3.3 volts) the characteristics of the common memory devices may be altered depending on which voltage is applied. There must be a one-to-one correspondence between the information fields listed in the Device Information tuple and the Other Conditions Device Information tuple.

Table 11-7 Tuples Defined for Compatibility Layer One (Continued)

Code (h)	CISTPL_NAME	Description and Purpose
1D	DEVICE_OA	Other Conditions Device Information (attribute memory).— Specifies the characteristics of devices mapped in the attribute memory address space, when operating under conditions other than the defaults. For example, if the card is a dual voltage card (operates at both 5 volts and 3.3 volts) the characteristics of the attribute memory devices may be altered depending on which voltage is applied. There must be a one-to-one correspondence between the information fields listed in the Device Information tuple and the Other Conditions Device Information tuple.
1E	DEVICEGEO	Device Geometry (common memory) — Device geometry provides the erase, read, and write characteristics of programmable devices. This tuple consists of multiple entries for each device identified in the device information tuple.
1F	DEVICEGEO_A	Device Geometry (attribute memory) — Device geometry provides the erase, read, and write characteristics of programmable devices. This tuple consists of multiple entries for each device identified in the device information tuple.
20	MANFID	PCMCIA Manufacturers Identification — Contains the PCMCIA manufacturer identification code and manufacturer card identifier and revision information.
21	FUNCID	Function Identification — Categorizes the card's functional type and specifies whether the card should be initialized during basic system initialization or when the operating system loads. A multi-function device may also be specified, in which case additional Function Identification tuples for each of the card's functions will follow.
Code (h)	CISTPL_NAME	Description and Purpose

Table 11-7 Tuples Defined for Compatibility Layer One (Continued)

Code (h)	CISTPL_NAME	Description and Purpose
22	FUNCE	Function Extension — Provides detailed information about a specific function previously identified by the function identification tuple. This tuple contains additional information useful to application programs or utility programs that are PCMCIA aware. Function extensions, if applicable, follow each Function Identification tuple in the tuple chain. Extensions are useful for defining the capabilities of various types of devices such as modems and network interface cards.
FF	END	Termination tuple — Indicates that this tuple is the last tuple in the string. However, by default parsing software will continue processing tuples at location zero in common memory. This implied jump to common memory occurs unless this tuple string contains either a LONGLINK OR NO_LINK tuple. If a no-link tuple has been encountered, the tuple string ends without further processing. If a valid long-link tuple has been encountered, tuple processing continues at the location specified, contingent on the presence of a LINKTARGET tuple at the target location. If there is neither a long- link nor a no-link tuple within the tuple string, tuple processing should continue at location zero in common memory.

Sample CIS implementations for SRAM, FAX/MODEM, Flash Card and ATA Hard Drive are discussed in later chapters.

Note that the CIS must start at address location zero in attribute address space or at the location specified by the LONGLINK_MFC tuple in multiple function PC Cards.

Chapter 12

The Previous Chapter

The previous chapter discussed the CIS and its role in the PC Card configuration process. Tuples were introduced and their format and structure were described. The basic structure of the CIS's configuration table required by I/O cards was also described.

This Chapter

This chapter discusses the configuration registers and provides a complete description of each register specified by the PC Card standard. Configuration register implementations for both single and multiple function cards are covered.

The Next Chapter

The next chapter describes a sample SRAM card implementation, including a functional block diagram of the SRAM card along with a sample CIS.

Configuration Registers

Each PC Card's I/O function must implement configuration registers. The PC Card standard defines the following configuration registers:

- Configuration Option Register — mandatory for all I/O functions
- Configuration and Status Register — optional
- Pin Replacement Register — optional
- Socket and Copy Register — optional
- Extended Status Register — optional
- I/O Base Address Register(s) — mandatory for multi-function PC Cards
- I/O Limit Register — optional

The format of each register is listed in table 12-1. These configuration registers are mapped into the attribute memory space at the location specified within the CONFIG tuple. Note that each function of a multiple function PC Card will have a dedicated set of configuration registers.

Table 12-1. Format of the Function Configuration Registers

Offset	7	6	5	4	3	2	1	0
0	Configuration Option Register							
	SRESET	LevlREQ	Function Configuration Index					
2	Configuration and Status Register							
	Changed	SigChg	IOIS8	RFU	Audio	PwrDwn	Intr	IntrAck
4	Pin Replacement Register							
	CBVD1	CBVD2	CREADY	CWProt	RBVD1	RBVD2	RREADY	RWProt
6	Socket and Copy Register							
	RFU	Copy Number			Socket Number			
8	Extended Status Register							
	Event3	Event2	Event1	Req Attn	Enable3	Enable2	Enable1	Req Attn Enable
10	I/O Base 0							
12	I/O Base 1							
14	I/O Base 2							
16	I/O Base 3							
18	I/O Limit							

Each of these registers have read/write capability and are mapped at even locations, consistent with the design of attribute memory. The definition of each configuration register is detailed below.

Configuration Option Register

The configuration option register (COR) configures PC Cards that have programmable address decoders. Once a card's client driver successfully parses the CIS and obtains the system resources required by the card, it assigns the resources to the card via the COR.

As discussed earlier in this chapter, the configuration table within the CIS specifies the configuration options that a given card supports. Each entry within the CIS contains a different combination of resources that satisfies a card's resource requirements. When the configuration options described by a particular configuration entry are found to be available, the index number of that configuration entry is written to the COR (refer to table 12-2). The index number programs the card to utilize the resources specified within the associated configuration table entry.

As shown in table 12-2, the COR also specifies whether the card should use level or pulse mode interrupts and provides a means for software to reset the card. Note that some memory cards may implement this register to support software reset as shown in the flash example. (See the chapter entitled, "A FLASH Card Example.")

Table 12-2. Configuration Option Register format and Definition

7	6	5	4	3	2	1	0
SRESET	LevlReq	Configuration Index					

SRESET	**Software Reset**. Setting this bit to one (1) places the card in the reset state. This is equivalent to assertion of the RESET signal except that this bit is not cleared. Returning this bit to zero (0), leaves the card in the same state that follows a hardware reset. This bit is set to zero by power up and hardware reset.
LevlReq	**Level Mode IREQ#**. Level Mode Interrupts are selected when this bit is one (1). Pulse Mode Interrupts are selected when this bit is zero (0).
Conf Index	**Configuration Index.** This field is written with the index number of the entry in the card's Configuration Table that corresponds to the configuration option chosen for the card. When the Configuration Index is 0, the card's I/O is disabled and will not respond to any I/O cycles and will use the memory-only interface. **Multi-function Card Index definition.** The PC Card standard specifically defines the use of each bit within the configuration index. **Bit 0** — Enables/disables specific function. 1=enabled; 0=disabled **Bit 1** — Specifies I/O addressing used. 1=I/O addresses specified by the base and limit registers are passed to function; 0=all host I/O address are passed to the function. (This bit is valid only when function is enable via bit 0.) **Bit 2** — Enables IREQ# routing. 1=This function will deliver interrupts to the PC Card's IREQ# line; 0=interrupts disabled for this function. (This bit is valid only when function is enabled.) **Bits 3-5** — vendor specific

Card Configuration and Status Register

This register contains a variety of functions used to control the card and report status, as shown in table 12-3. These functions include:

- Status change indication and reporting (bits 6 and 7)
- PCMCIA host expansion bus interface size (bit 5)
- Audio enable (bit 3)
- Power down control for power conservation (bit 2)
- Interrupt pending status (bit 1)

Status Change

Prior to being configured, an I/O card interfaces to the HBA as a memory only device. While in this state, any status change event must be reported directly over the appropriate status change pin. However, when the card is configured, (the COR is written) the card switches to the I/O interface and status change events are now reported via the pin replacement register (PRR) and the card configuration and status register (CSR).

The status changed bit (bit 7) and the signal change bit (bit 6) of the CSR determine whether a status change has occurred when the card is configured for the I/O interface and whether it should be reported over the I/O interface's STSCHG# pin. When a status change event occurs, the appropriate bit is set in the PRR and the status changed bit (Chng) is set in the CSR. When a status change occurs, the card asserts the STSCHG# pin to notify the HBA of the event. The Chng bit remains set until the PRR bit is reset indicating that the status change event has been processed.

The signal change bit (SigChg) is used by the HBA to disable the card from asserting the STSCHG# pin again until the current status change event has been processed. Software must clear this bit when processing a status change interrupt for the card. This permits the next status change event to be reported once the previous event has been processed.

Table 12-3. Card Configuration and Status Register and Definition

7	6	5	4	3	2	1	0
Chng	SigChg	IOis8	Resrv (0)	Audio	PwrDn	Intr	IntrAck

Chng	**Status Change Detected**. This bit indicates that one or more of the Pin Replácement Register bits (CBVD1, CBVD2, CRDY, or CWProt) is set to one, normally causing the STSCHG# signal to be asserted. However, if the SigChg bit (see below) is 1 and the card is configured for an I/O interface, the STSCHG# pin is asserted when this bit is set.
SigChg	**Signal Change Enable/Disable**. This bit is set and reset by the host to enable and disable a status-change signal from the status register. When this bit is set and the card is configured for the I/O interface, the Chng bit controls pin 63 (STSCHG#). If no status change signal is desired, this bit should be set to zero and the STSCHG# signal will be held deasserted when the card is configured for I/O.
IOis8	**I/O Cycles Occur Only as 8-bit Transfers.** When the host can provide I/O cycles only using the D7:D0 data path, the PCMCIA software will set this bit to a 1. The card is guaranteed that accesses to 16-bit registers will occur as two byte accesses rather than a single 16-bit access. This information is useful when 16-bit and 8-bit registers overlap.
Resrv	Reserved bits must be 0.
Audio	**Audio Enable.** This bit enables audio information to be sent to the HBA via the speaker pin when configured for an I/O interface.
PwrDn	**Power Down.** This bit is set to one to request that the card enter a power-down state. PCMCIA software must not place the card into a power-down state while the card's READY pin is in the low (Busy) state.
Intr	**Interrupt Request Pending.** This bit represents the internal state of the interrupt request. This value is available whether or not interrupts have been configured. How the Intr bit is cleared is dependent of how the IntrAck bit is configured. **IntrAck=0** — Intr reflects the function's interrupt request status. If the interrupt is cleared within the function, then Intr is reset by the function. **IntrAck=1** — Intr remains set even though the interrupt condition has been cleared. It is reset by system software to indicate it is ready to receive another interrupt (implemented to support interrupt sharing).

Table 12-3. Card Configuration and Status Register and Definition(Continued)

7	6	5	4	3	2	1	0
Chng	SigChg	IOis8	Resrv (0)	Audio	PwrDn	Intr	IntrAck
IntrAck	**Interrupt Acknowledge.** This bit determines the response of the Intr bit. The functionality associated with the IntrAck bit permits two or more functions to share the PC Card's IREQ# pin. **IntrAck=0** — when IntrAck is reset Intr functions as described above to support a single interrupt implementation. **IntrAck=1** — This causes the Intr bit to remain set even though the interrupt service routine has already serviced the interrupt. Normally, the interrupt service routine clears the interrupt pending bit in a function specific register, causing the Intr also to be cleared. However, to support interrupt sharing the Intr bit is not cleared until PCMCIA specific software is ready to handle the next interrupt request. When cleared by the PCMCIA software, other interrupt requests that are pending can now be asserted over the PC Card's IREQ# pin. (Refer to the chapter entitled, "Multiple Function PC Cards."						

Size of Host Expansion Bus

The IOis8 bit reflects the size of the expansion bus that the HBA connects to. When this bit is set, I/O cycles will always occurs as individual 8-bit transfers over the lower data path (D7:D0). When the bit is reset, accesses to 16-bit registers will occur in a single cycle.

Audio Enable

The Audio bit is set to enable audio information to be sent over the I/O interface's SPKR pin. Whether or not the I/O card has audio capability is specified within the miscellaneous information structure within the configuration table entry.

Power Conservation Mode

Some cards support a low power mode that can be used for power conservation. Power management software can set the power down (PwrDn) bit, placing the card in a low power state, if supported. Note that this bit should not be set if the card is in the busy state as indicated by the PRR.

Interrupt Pending

The Intr bit is set by the card when its interrupt request (IREQ#) pin is asserted. If the PC Card implements a single I/O function, the Intr bit remains set until the interrupt service routine is executed, at which time the Intr bit is reset.

Pin Replacement Register

Cards using a memory only interface report status change directly to the HBA via the status change pins. However, when a card uses the I/O interface, the status change pins are replaced by other I/O specific interface signals. As a result, the HBA has no visibility of status change events that may occur on the I/O card. The pin replacement register (PRR) replaces the HBA functions that are normally used to indicate the status of change events for the memory interface.

Refer to table 12-4. The PRR specifies the current state of the status change events (bits 3:0) and whether a change has occurred for a particular event (bits 7:4). The current state of the events (RWP, RREADY, RBVD2, and RBVD1) can be read directly from the lower four bits of the PRR register. When a change occurs for any of these items, its corresponding changed bit is set in the upper group of bits. In this way, processing software can read the upper four bits to determine which event(s) has occurred and therefore, the one needing to be processed. When a given event is processed, the lower portion of the register can be read to check the new state of the event that signaled the change. When the event is processed, software should reset the changed bit, thus permitting another event to be reported.

Table 12-4. Pin Replacement Register

<table>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>CBVD1</td><td>CBVD2</td><td>CRdy</td><td>CWP</td><td>RBVD1</td><td>RBVD2</td><td>RREADY</td><td>RWP</td></tr>
<tr><td colspan="2">CBVD1, CBVD2</td><td colspan="6"><b>Changed BVD1 and BVD2.</b> These bits are set to one when the corresponding bit (RBVD1 and/or RBVD2) changes from one state to another. These bits may also be cleared by the host.</td></tr>
<tr><td colspan="2">CREADY</td><td colspan="6"><b>Changed READY.</b> This bit is set to one when the bit RREADY changes state. This bit may also be cleared by the host.</td></tr>
</table>

Table 12-4. Pin Replacement Register (Continued)

CWProt	**Changed Write Protect.** This bit is set to one when the bit RRWProt changes state. This bit may also be cleared by the host.
RBVD1, RBVD2	**Current State of BVD1 and BVD2.** These bits represent the internal state of the Battery Voltage Detect circuits on cards that contain a battery. They correspond to the values that would be on pins 63 and 62, BVD1 and BVD2 respectively. When this bit is set, the corresponding changed bit is also set. When this bit is cleared, the corresponding changed bit is unaffected.
RRdy	**Current State of Ready.** This bit represents the internal state of the READY signal. This bit reflects the state of READY (since the READY pin has been reallocated for use as Interrupt Request on IO Cards). When this bit is set, the corresponding changed bit is also set. When cleared, the corresponding changed bit is unaffected.
RWProt	**Current State of Write-Protect Switch.** This bit represents the current state of the Write-Protect switch. This bit reflects the state of the Write Protect switch when pin 24 is being used for IOIS16#. When this bit is set, the corresponding changed bit is also set. When cleared, the corresponding changed bit is unaffected.

Socket and Copy Register

Refer to table 12-5. This register is used for I/O cards that can coexist with one or more identical cards within the system and respond to the same I/O address ranges. This capability can be used for ATA (IDE) drives that are designated as drive 0 and drive 1. Each responds to the same I/O address space but can be uniquely identified with the socket and copy register. The first card configured will be assigned as copy zero and each card configured thereafter receives the next sequential copy number. The socket number identifies the socket that a given copy occupies.

Table 12-5. Socket and Copy Register

<table>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="2">Reserved (0)</td><td colspan="3">Copy Number</td><td colspan="3">Socket Number</td></tr>
<tr><td colspan="2">Reserved</td><td colspan="6">This bit is reserved for future standardization. This bit must be set to zero (0) by software when the register is written.</td></tr>
<tr><td colspan="2">Copy Number</td><td colspan="6">Cards that can coexist with other cards (twin cards) that are configured identically, should have a copy number identifying this particular copy of the card. (0 to MAX twin cards, MAX = n-1) This field indicates to the card that it is "nth" copy of the card installed in the system that is identically configured. The first card installed receives the value 0. This permits identical cards designed to do so to share a common set of I/O ports while remaining uniquely identifiable and consecutively ordered.</td></tr>
<tr><td colspan="2">Socket Number</td><td colspan="6">This field indicates to the card that it is located in the nth socket. The first socket is numbered 0. This permits any cards designed to do so to share a common set of I/O ports while remaining uniquely identifiable.</td></tr>
</table>

Extended Status Register

This register has been added to the PC Card standard to extend the number of events that can be reported via the STSCHG# pin and to give software the ability to detect and clear the event. The extended status register is organized as an upper nibble (whose bits are set when the corresponding function event occurs) and a lower nibble (that enables and disables setting the "Changed" bit in the CSR). When a status change interrupt occurs PC Card software can read the extended status register to determine if an associated bit has caused an the interrupt.

Table 12-6 illustrates the format of the extended status register. Notice that only the "Requires Attention" and "Requires Attention Enable" bits are defined.

Table 12-6. Format and definition of the Extended Status Register

D7	D6	D5	D4	D3	D2	D1	D0
Event3	Event2	Event1	Req Attn	Enable3	Enable2	Enable1	Req Attn Enable

Field	Description
Event3	Reserved for future expansion/definition, must be reset (0)
Event2	Reserved for future expansion/definition, must be reset (0)
Event1	Reserved for future expansion/definition, must be reset (0)
Req Attn	This bit is latched within one (1) ms of an event occurring on the PC Card, (such as the start of each cycle of the ring frequency to indicate the presence of ringing on the phone line in the case of a modem card). When this bit is set to a one (1), and the *Req Attn Enable* bit is set to a one (1), the *Changed* bit in the Configuration and Status register will also be set to a one (1), and if the *SigChg* bit in the Configuration and Status register has also been set by the host, the STSCHG# pin (63) will be asserted. The host writing a one (1) to this bit will reset it to zero (0). Writing a zero (0) to this bit will not have any effect.
Enable3	Reserved for future expansion/definition, must be reset (0)
Enable2	Reserved for future expansion/definition, must be reset (0)
Enable1	Reserved for future expansion/definition, must be reset (0)
Req Attn Enable	Setting this bit to a one (1) enables the setting of the *Changed* bit in the Configuration and Status register when the *Req Attn* bit is set. When this bit is reset to a zero (0), this feature is disabled. The state of the *Req Attn* bit is not affected by the *Req Attn Enable* bit.

I/O Base Registers

The PC Card standard defines these I/O base registers for use by multiple function cards, but they can also be used by single function cards. These registers define the base I/O address to which the function's I/O registers will be mapped into the host processor's address space. The number of registers used depends on the address space supported by the host processor. Since Intel compatible x86 processors have only 64KB of address space, only the first two registers are needed to specify a base address anywhere within the entire 64KB space.

I/O Limit Register

This register relates to the I/O base registers by specifying the maximum range of I/O addresses that can be mapped beginning at the base address. This register is bit mapped such that the most significant bit that is set determines the number of address lines used to decode the address and therefore the maximum block of address space supported. The most significant bit and all bits of lesser significance must be set within the register. This results in the possible number of address lines as listed in table 12-7. Note that the largest block of I/O address space that can be defined is 256 bytes.

This register is optional and need not be implemented for each function if all functions within the PC Card use the same number of I/O registers.

Table 12-7. Address Limit Associated with Function Base Address Register

Bit Position								Maximum
7	6	5	4	3	2	1	0	Number of
# of Address Lines Defined by Bit position								Address
8	7	6	5	4	3	2	1	Locations
0	0	0	0	0	0	0	0	Not defined
0	0	0	0	0	0	0	1	2
0	0	0	0	0	0	1	1	4
0	0	0	0	0	1	1	1	8
0	0	0	0	1	1	1	1	16
0	0	0	1	1	1	1	1	32
0	0	1	1	1	1	1	1	64
0	1	1	1	1	1	1	1	128
1	1	1	1	1	1	1	1	256

Chapter 13

The Previous Chapter

The previous chapter discussed the configuration registers and provided a complete description of each register specified by the PC Card standard. Configuration register implementations for both single and multiple function cards were covered.

This Chapter

This chapter describes a sample SRAM card implementation, including a functional block diagram of the SRAM card along with a sample CIS.

The Next Chapter

The next chapter describes a sample flash card implementation, including a functional block diagram of the card, a sample CIS, and configuration registers implemented by the card.

An SRAM Card Example

Figure 13-1 illustrates the functional blocks associated with an SRAM memory card. Note that this is an example implementation of a 2MB SRAM card. The contents of the CIS are illustrated and discussed in the next section.

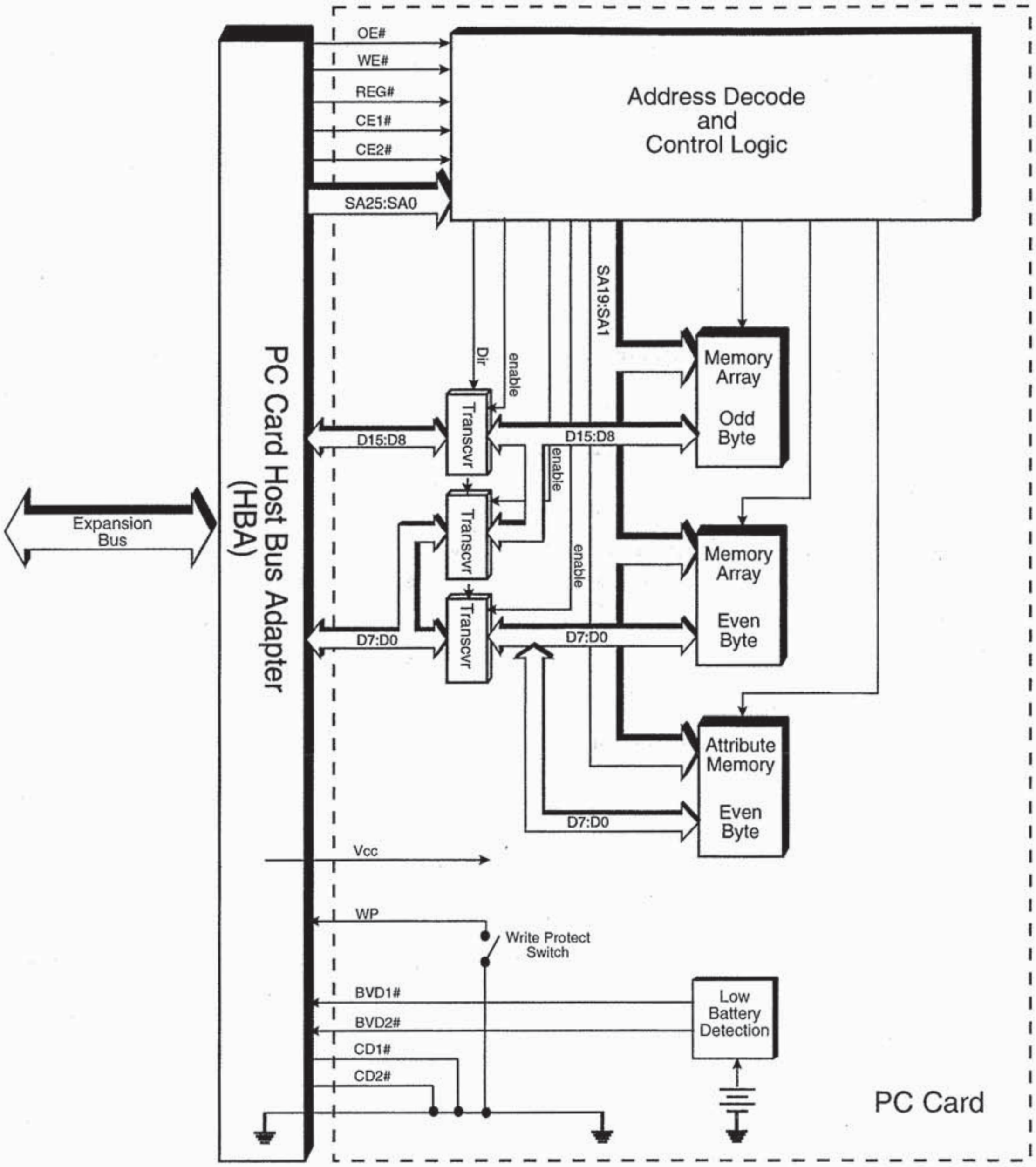

Figure 13-1. Block Diagram of 2MB SRAM PC Card

The SRAM CIS

The following example illustrates the CIS implemented within a typical SRAM card. SRAM PC Card design is relatively simple when compared to I/O cards. As shown in figure 13-2, a typical SRAM CIS may consist of four tuples. The sections following figure 13-2 describe the purpose and contents of each tuple in the SRAM example. Refer to appendix A for a detailed listing and analysis of the tuples contained in this SRAM example.

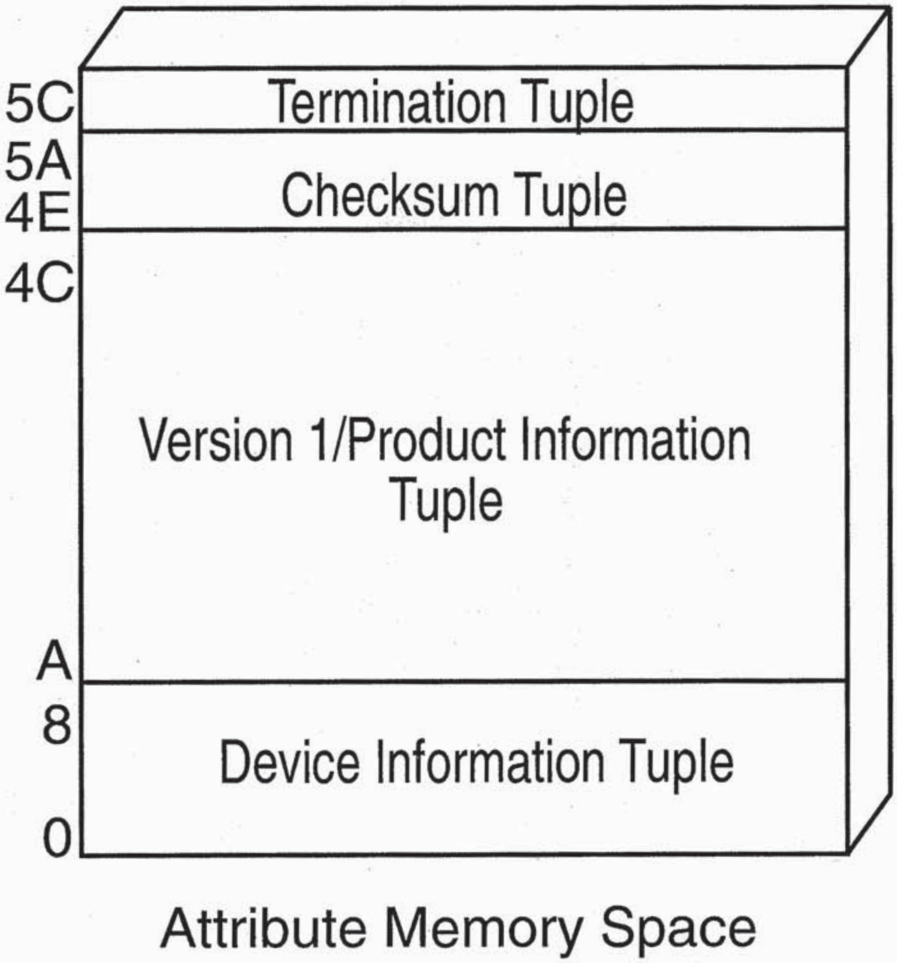

Figure 13-2. Map of Attribute Memory Addresses on Example SRAM Card

Device Information Tuple

As described earlier, the Device Information tuple defines all the information needed to characterize an SRAM memory card. The device information tuple defines the following operational characteristics:

- Device Type (SRAM in this example).
- Device Speed (250ns is this example).
- Write-Protect switch (WPS) definition. Whether the memory defined within the tuple is affected by the write-protect switch (WPS is used).
- Size of the memory array (2MB in this example).

Since no configuration table exists, the memory array described is mapped by default at base address zero within common memory address space.

Level 1 Version / Product Information Tuple

This tuple contains the PCMCIA version of the CIS and ASCII characters describing the product. The data area within the SRAM level 1 version/production information tuple consists specifically of:

- Major version 4 (relates to JEIDA release 4.0).
- Minor version 1 (relates to PCMCIA release 1.0) A major version number of 4 and a minor version number of 1 indicates 2.x compliant CIS.
- ASCII string indicating manufacturer and card description.
- ASCII string indicating model number of card.
- ASCII string indicating serial number card.

The ASCII character strings contained within the product information portion of the tuple are defined by the PC Card manufacturer. The manufacturer and card description information within this tuple are typically read and displayed by PCMCIA configuration software when a card is configured. This notifies the user that the card has been recognized and identified.

Checksum Tuple

The checksum tuple provides a way for processing software to verify that the data read from the CIS is correct. The checksum data block information includes:

- Offset from checksum tuple to the start address of the range to be checked.
- Number of locations to be checksummed from the start address.
- Checksum value.

More than one checksum tuple can be used within a CIS. This example contains a single checksum tuple used to check the CIS from location zero to location 4Ch.

Termination Tuple

The termination tuple consists only of the tuple code FFh. In this example, when processing software encounters the termination tuple, it will continue tuple processing by going to location zero in common memory. Common memory may contain additional tuple information written there by PCMCIA aware software that formats the SRAM memory for use as a virtual drive.

This capability stems from 1.0 compliant cards that did not require that a CIS be implemented. When processing software attempts to read the CIS, a value of FFh will be returned when no CIS is implemented. This is interpreted by software as a termination tuple. Software then reads from location zero in common memory where a link-target tuple will be found. The software then looks for a BIOS Parameter Block (BPB) that characterizes the size of the SRAM to be used as a virtual drive.

Chapter 14

The Previous Chapter

The previous chapter described a simple SRAM card implementation, including a functional block diagram of the SRAM card along with a sample CIS.

This Chapter

This chapter describes a flash card implementation, including a functional block diagram of the card, a sample CIS, and configuration registers implemented by the card.

The Next Chapter

The next chapter describes an example FAX/Modem implementation, including a functional block diagram, a sample CIS, and configuration registers implemented by the card.

An Example Flash Card Implementation

Figure 14-1 illustrates the functions associated with an Intel series II Flash-Card. This example is based on a 10MB flash memory array and includes a CIS contained within the flash control ASIC. This card also incorporates flash memory that implements a ready/busy (RDY/BSY#) pin and takes advantage of the memory socket's READY pin.

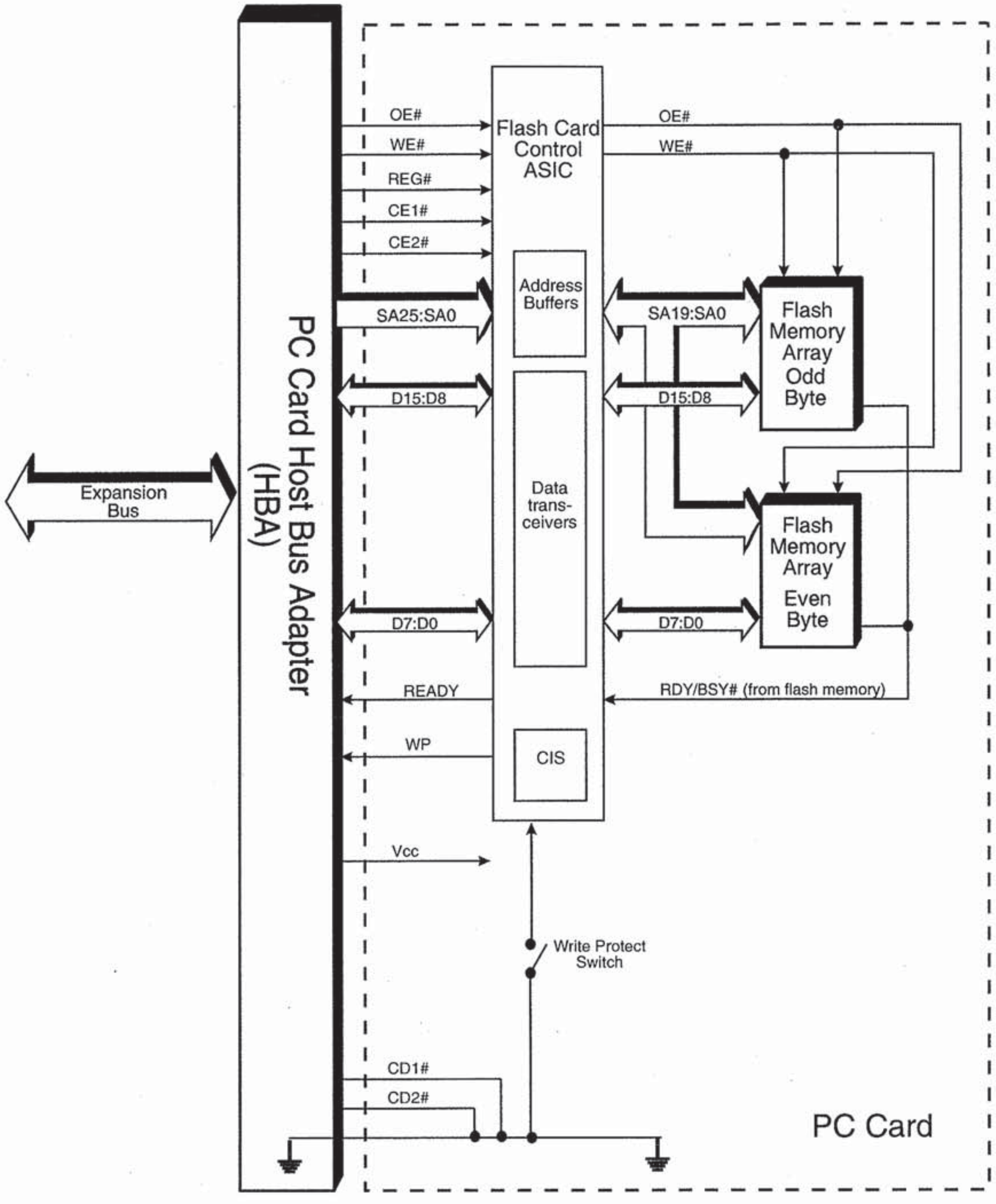

Figure 14-1. 20MB Flash Card Functional Diagram

A Flash Memory CIS Example

Following is an example of a flash memory card's attribute memory address space. As shown in figure 14-2, this flash card implements both a CIS and configuration registers. The sections following figure 14-2 describe the purpose and contents of each tuple used by the flash card in this example. Appendix B contains a detailed listing and explanation of the tuples in this flash memory card example.

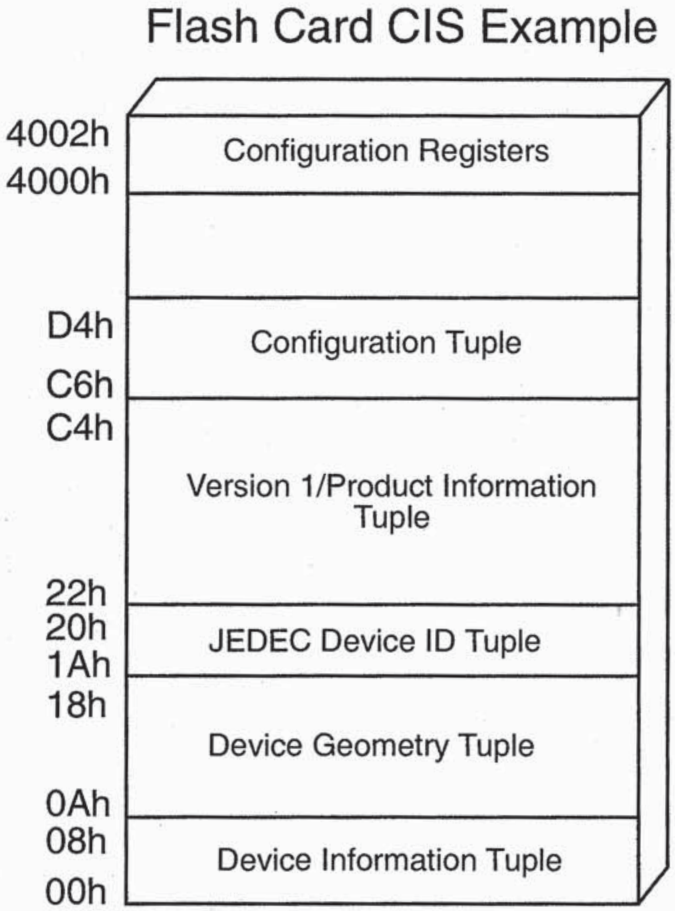

Figure 14-2. Example Contents of a Flash Card's Attribute Address Space

Device Information Tuple

The device information tuple identifies the basic characteristics of the card. The device information tuple determines the following parameters:

- Device Type (flash memory).
- Device Speed (150ns).
- Write-Protect switch (WPS) definition. Whether the memory defined within the tuple is affected by the write-protect switch (WPS used).
- Size of the memory array (10MB).

Unlike the previous SRAM example, the contents of the Device Information tuple does not completely characterize a flash memory array. Flash cards require an additional Device Geometry tuple to specify the block size for erasing and writing to the flash memory array.

Since no configuration table exists, the memory array described is not programmable and responds only to location 0 to 10MB in common memory address space.

Device Geometry Tuple

Flash memory cards are block oriented devices when writing to or erasing their memory arrays. As a result, the Memory Technology driver must know the block size in order to access the device correctly. The Device Geometry tuple contains the block size that is implemented by the memory array for erasing, writing and reading the flash card. Information described by the tuple includes:

- Internal bus width (always 2 bytes for release 1.0 - 2.x cards).
- Erase geometry block size.
- Read geometry block size.
- Write geometry block size.
- Partition size (indicates partition size, if the memory array is partitioned).
- Interleave size (describes whether hardware interleaving is incorporated to enhance read performance, and if so, what the interleaving size is).

JEDEC Device Identifier (ID) Tuple

Many memory devices contain the JEDEC Device ID tuple within their CIS. As its name suggests, this tuple contains the card manufacturer's JEDEC ID and incorporates device type information that specifies a corresponding programming algorithm. The Joint Electronics Device Engineering Council (JEDEC) assigns an ID to manufacturers designing programmable memory devices. All programmable memory devices should have a corresponding JEDEC identifier.

Note that for each entry in a device information (DEVICE) tuple a corresponding entry must be made in the JEDEC device identifier tuple. If a DEVICE tuple contains both programmable and non-programmable memory devices, then the JEDEC tuple entries for the non-programmable device will contain null values.

Level 1 Version / Product Information Tuple

This tuple contains the PCMCIA compliance level of the CIS (level 1 version) and ASCII characters describing the product. The data area within the flash level 1 version/production information tuple consists specifically of:

- Major version 5 (relates to PC Card February, 1995 release).
- Minor version 0 (relates to PC Card February, 1995 release).
  Note: A major version number of 5 and a minor version number of 0 indicates compliance with the PC Card 95 release.
- ASCII string indicating manufacturer and card description.
- ASCII string indicating model number of card.
- ASCII string indicating serial number of card.

The ASCII character strings contained within the product information portion of the tuple are defined by the manufacturer. The manufacturer name and card description is sometimes read and displayed by PCMCIA utilities when a card is configured. This tuple is also commonly used by PC Card enablers that are designed to identify and configure a specific card.

Configuration Tuple

The Configuration tuple identifies the type of the configuration register(s) used by the PC Card, along with their location within attribute memory space. Data entries within the Configuration tuple contain the following:

- Size of address fields—This entry defines the number of bytes used by this tuple to identify the location of the configuration registers. Since these registers can be located anywhere within attribute memory address space (0 to 64MB), the number of bytes needed to define their location depends on where they reside in the address space. In this example, the registers are mapped to location 4000h, therefore only two bytes are needed to specify their location.
- Size of configuration register mask field— Specifies the number of bytes needed by the configuration register mask field to identify the configuration registers implemented by this function. PCMCIA currently defines ten configuration registers of the 128 configuration registers that can be specified. To specify all 128 registers the configuration register mask field would require sixteen 8-bit mask registers. This example implementation uses the first two registers, therefore a single mask register is implemented. Refer to the section entitled, "Flash Card Configuration Registers" later in this chapter for details.
- Index number of the last entry in the configuration table—Since this example flash card has no configuration table this entry is zero.
- Starting (base) address of the configuration registers—In this example, a two byte field identifies the location of the configuration registers in attribute memory (location 4000h).
- Configuration register mask — A bit map that corresponds to the configuration register implemented by the PC Card function. The mask value in this example specifies that only registers corresponding to bit 0 (the Configuration Option Register) and bit 1 (the Status Register) are implemented.

Termination Tuple

The termination tuple consists only of the tuple code FFh. In this example, when processing software encounters the termination tuple, it will continue tuple processing by going to location zero in common memory. Common memory may contain additional tuple information written there by PCMCIA aware software that formats the flash memory for use as a virtual drive.

This capability stems from 1.0 compliant cards that did not require that a CIS be implemented. When processing software attempts to read the CIS, a value of FFh will be returned since no CIS is implemented. This is interpreted by software as a termination tuple. Software then reads from location zero in attribute memory where a link-target tuple will be found. The software then looks for a BIOS Parameter Block (BPB) that characterizes the size of the memory used as a virtual drive.

Flash Card Configuration Registers

The flash card in this example uses two of the configuration registers that are defined by the PCMCIA standard. These two registers are the configuration option register and the configuration status register. As implemented, these register use only a small portion of the associated functions defined by PCMCIA.

Configuration Option Register

The flash card in this example uses the configuration option register (bit 7) to permit software reset capability at the card level. The other functions associated with the configuration option register are not used.

Configuration Status Register

The flash card in this example also uses the configuration status register (bit 2) for placing the card into the power down state for power conservation. All other functions associated with the configuration status register are not used.

Chapter 15

The Previous Chapter

The previous chapter described a flash card implementation, including a functional block diagram of the card, a sample CIS, and configuration registers implemented by the card.

This Chapter

This chapter describes an example FAX/Modem implementation, including a functional block diagram, sample CIS, and related configuration registers.

The Next Chapter

The next chapter describes an PC Card ATA drive implementation, including a functional block diagram, a sample CIS, and configuration registers implemented by the card.

An Example FAX/Modem Card

Figure 15-1 illustrates the functions incorporated into a FAX/Modem PC Card. The socket interface is configured as a memory-only interface when the PC Card is first installed and reconfigured as a memory or I/O socket during the configuration process. Note that all the registers in this PC Card implementation are 8-bit registers; therefore, this PC Card does not assert the IOIS16# pin.

The modem consists of the UART (Universal Asynchronous Receiver/Transmitter), the modem controller, the modem data pump and the DAA (Data Access Arrangement).

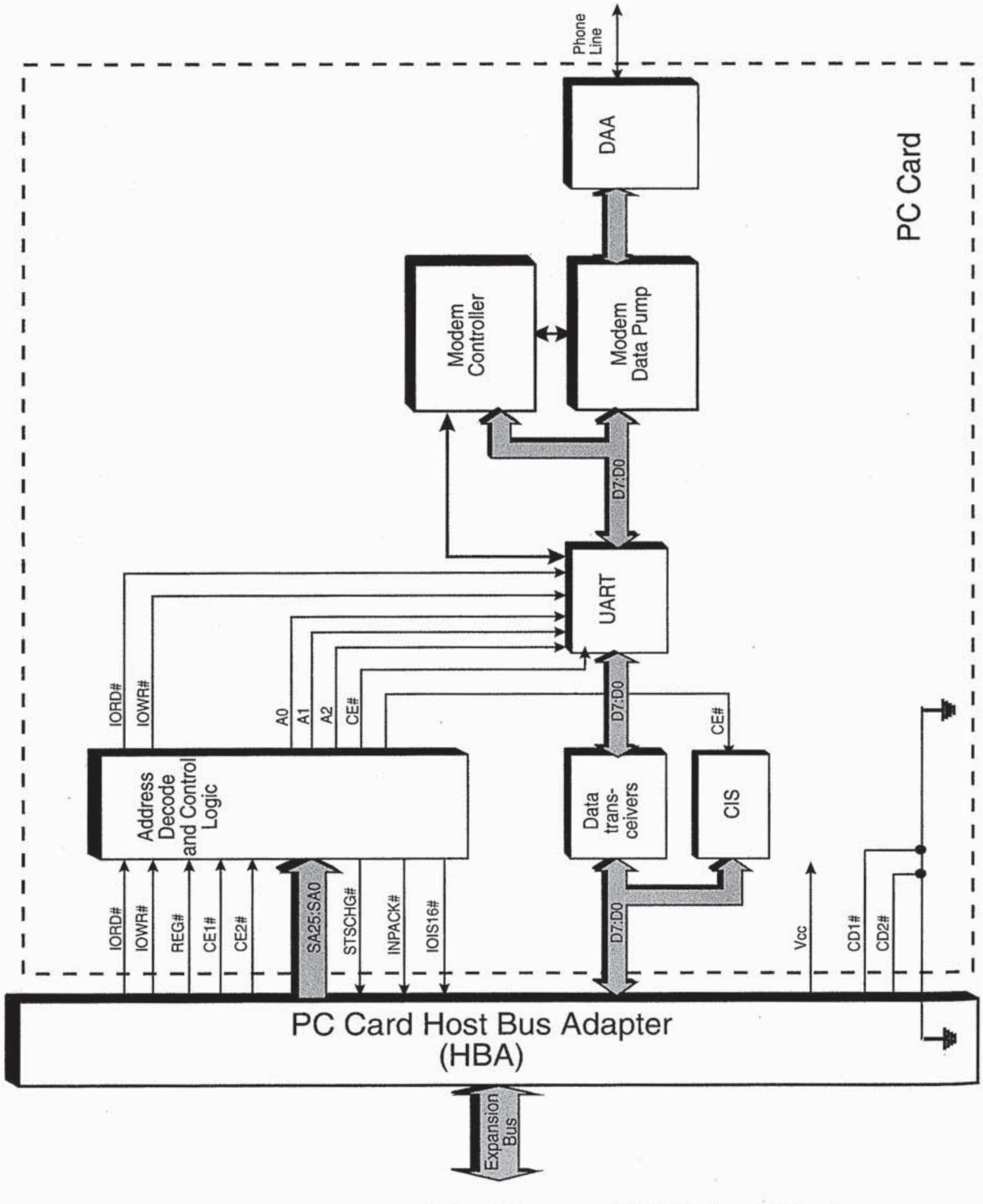

Figure 15-1. Functional Block Diagram of FAX/Modem PC Card

FAX/Modem Resource Requirements

FAX/Modems require I/O address space and a system IRQ line to allow the application software to communicate via a standard serial interface. In MS-DOS and Windows implementations, the serial interface has been mapped to a standardized range of addresses and associated IRQ lines. These conventional resource locations are needed because communications software typically accesses FAX/Modem hardware registers directly. Software typically expects the serial interface to be mapped to the conventional resources that are frequently referred to by the DOS device names: communications ports one through four (COM 1, COM 2, COM 3, and COM 4).

The convention location for these communications interface are:

- COM 1 = I/O addresses 3F8h-3FFh and IRQ 4
- COM 2 = I/O addresses 2F8h-2FFh and IRQ 3
- COM 3 = I/O addresses 3E8h-2EFh and IRQ 4
- COM 4 = I/O addresses 2E8h-3EFh and IRQ 3

Note that some communications software may be able to access the serial interface at other non-conventional address locations and IRQs. Specifically, PC Card aware application programs can gain access to the PC Card configuration information and determine how the PC Card has been configured by the enabler. Once the application knows how the PC Card has been configured, it can gain access to the card via the specified I/O address locations and IRQ lines without having to rely upon the conventional configurations specified above.

A FAX/Modem CIS Example

Figure 15-2 illustrates the contents of attribute memory address space for a FAX/modem. Notice that the CIS contains a configuration table. A configuration table is used by PC Cards having functions that can be configured using a variety of different system resources. The configuration table consists of entries that define different resources combinations that can be assigned to the PC Card. If one of the resource combinations are available for the PC Card's use, then it can be successfully configured. If the resource combinations required by the FAX/Modem are not available for use then the card cannot be configured.

Device Information Tuple

The Device Information tuple identifies the basic characteristics of memory cards. Since the FAX/Modem is an I/O device, the device information tuple contains no relevant information. The data portion of this tuple is zero, indicating that this card is not a memory card.

Level 1 Version / Product Information Tuple

This tuple contains the PCMCIA compliance level of the CIS (i.e. the version of CIS, recall that level 1, or layer 1 of the metaformat defines the CIS) and ASCII characters describing the product. The data area within the FAX/Modem level 1 version/production information tuple consists specifically of:

- Major version 5 (relates to PC Card February, 1995 release).
- Minor version 0 (relates to PC Card February, 1995 release).
  Note: A major version number of 4 and a minor version number of 1 indicates 2.x compliant CIS.
- ASCII string indicating manufacturer and card description.
- ASCII string indicating model number of the card.
- ASCII string indicating serial number of the card.

The ASCII character strings contained within the product information portion of the tuple are defined by the manufacturer. The manufacturer name and card description is sometimes read and displayed by PCMCIA utilities when a card is configured. This tuple is also used by client device drivers that are designed to identify a specific card.

Card Manufacturer Identification (ID) Tuple

As its name suggests, this tuple contains the PCMCIA card manufacturer's ID number. The PCMCIA organization assigns an ID to the manufacturers designing PCMCIA compliant cards.

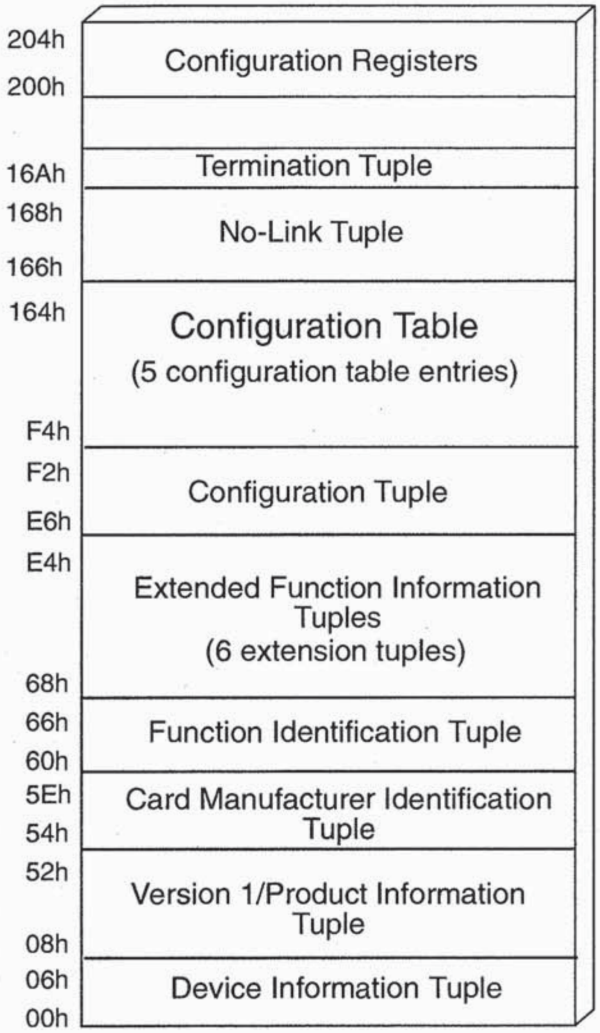

Figure 15-2. Example of Attribute Memory Address Contents for FAX/Modem

Function Identification Tuple

The Function Identification tuple determines the type of functional device that is implemented in the PC Card. Memory cards can be specified through the Device Information tuple, whereas, I/O devices must use the Function Identification tuple. This tuple defines the following items:

- Function type code — consists of a code representing the type of device implemented in the PC Card. The function type associated with the FAX/modem is a serial port.
- Initialization byte — specifies whether this device should be configured during system initialization (also called Power-On Self Test or POST) and whether the card has a ROM containing configuration software. This is used by devices when loading the operating system. Not used by the FAX/modem.

Function Extension Tuples

Function Extension tuples are defined by PCMCIA for some types of devices, including modems. Function Extension tuples must immediately follow the Function Identification tuple to which they apply. This example consists of six different function extension tuples. Within each tuple is a code identifying it as a particular type of function extension. These extensions fall into three basic categories for serial devices:

- Data modem extensions
- FAX modem extensions
- Voice modem extensions (not used by the FAX/modem)

Each Function Extension tuple provides information related to the capabilities of the modem. This information includes items such as communications protocols, error correction protocols, and other communications parameters. This information can be used by PCMCIA aware applications to automatically configure the application based on the card's capabilities.

Configuration Tuple

The Configuration tuple identifies the type of the configuration register(s) implemented in the PC Card, along with their location within attribute memory space. This tuple also specifies the index number of the last entry within the CIS. Data entries within the Configuration tuple contain the following:

- Size of address fields — This entry defines the number of bytes used later within this tuple to identify the location of the configuration registers. Since the configuration registers can be located anywhere within attribute memory address space (0 to 64MB), the number of bytes needed to define their location depends on where they reside in the address space. In this example, the registers are mapped starting at location 200h; therefore, only two bytes are needed to specify their location.

- Size of configuration register mask field — Specifies the number of bytes needed by the configuration register mask field to identify the configuration registers implemented by this function. PCMCIA currently defines ten configuration registers out of the 128 configuration registers that can be identified. To specify all 128 registers the configuration register mask field would require sixteen 8-bit mask registers. This example implementation uses registers 0, 1, and 2 therefore a single mask register is implemented. Refer to the section entitled, "FAX/Modem Card Configuration Registers" later in this chapter for details.
- Index number of the last entry in the configuration table — This value indicates to processing software when the last entry within the card's configuration tuple has been reached.
- Starting (base) address of the configuration registers — In this example, a two byte field identifies the location of the configuration registers in attribute memory (location 0200h).
- Configuration register mask — Specifies that configuration registers zero (Configuration Option Register), one (Status Register) and two (Pin Replacement Register) are implemented.

Configuration Table

The configuration table contains the configuration option supported by the FAX/modem card. The card in this particular example contains five entries within the configuration table, each defining a different combination of system resources required to support its functions. The serial port used by the modem requires an eight byte block of contiguous I/O addresses and a system interrupt line. This device supports standard resources defined by convention in the DOS environment. The following list shows the 8-byte I/O range and IRQ line specified by each entry within the configuration table.

- COM 1—I/O base address 3F8h and IRQ 4 (entry 1)
- COM 2—I/O base address 2F8h and IRQ 3 (entry 2)
- COM 3—I/O base address 3E8h and IRQ 4 (entry 3)
- COM 4—I/O base address 2E8h and IRQ 3 (entry 4)
- Any 8-byte range of I/O addresses and any one of the IRQs: 2, 3, 4, 5, 7, 9, 10, or 15 (entry 5)

The first resource combination that can be allocated by the system will be assigned to the HBA and PC Card for its use. The index number of the configuration table entry that satisfied the resource requirements is programmed into the configuration option register. This configures the PC Card to respond to the resources specified within the selected configuration table entry.

No-Link Tuple

The no-link tuple tells processing software to terminate tuple processing when the termination tuple is reached. This prevents the implied jump to location zero of common memory.

Termination Tuple

The termination tuple consists only of the tuple code FFh. In this example, when processing software encounters the termination tuple, it will end tuple processing since the no-link tuple exists in the tuple listing.

FAX/Modem Configuration Registers

The FAX/Modem card in this example implements three of the ten configuration registers defined by the PCMCIA standard. These registers include the configuration option register, status register and pin replacement register. Their use in the fax/modem card is defined in the following sections.

Configuration Option Register

The configuration option register performs several functions related to the FAX/modem card's operation:

- Configuration Index — selects the entry within the configuration table that satisfied the card's resource requirements. This value programs the I/O address decoders on the card to respond to the correct address range.
- Interrupt Request Level — selects whether level or pulse mode interrupts should be delivered over the IREQ# pin by the PC Card.

- Software Reset — provides the ability for software to reset the PC Card. Setting this bit has the same affect on the hardware as asserting the RESET pin.

Configuration Status Register

This register performs the following functions as they relate to the FAX/modem card:

- Audio Supported — set by software to enable the PC Card to output audio information to the HBA via the speaker pin.
- Interrupt Pending — set by the PC Card to indicate that an interrupt has been asserted to the HBA and has not yet been serviced.
- Status Change — set by the PC Card to indicate that a pin replacement register has been implemented and should be checked to see if a status change has occurred.

Pin Replacement Register

The pin replacement register is used to report status change events that are supported by the PC Card. This is done in lieu of socket interface pins that are not available when the socket is configured as an I/O interface. The FAX/modem in this example implements the READY status change function and therefore implements the pin replacement register.

Chapter 16

The Previous Chapter

The previous chapter described an example FAX/Modem implementation, including a functional block diagram, sample CIS, and related configuration registers.

This Chapter

This chapter describes an example PC Card ATA drive implementation, including a functional block diagram, a sample CIS, and configuration registers implemented by the card.

The Next Chapter

The next chapter describes a multi-function PC Card design, including a functional block diagram, a multi-function CIS, and related configuration registers.

An ATA PC Card Example

Figure 16-1 illustrates the functions contained within an ATA PC Card based on rotating magnetic media. Other ATA PC Card designs are based on flash memory technology implemented as virtual disk drives that provide the same programming interface employed by standard ATA disk drives.

As with any PC Card, the initial socket interface is automatically configured as a memory-only interface when the PC Card is first installed. After the CIS is read and the ATA PC Card's enabler has detected the ATA card's presence, the enabler initiates the configuration process. As discussed in the following section an ATA PC Card can be configured to operate with the memory interface (i.e. the registers are mapped into the processor's memory address space) or with the memory or I/O interface (using standard I/O mapping).

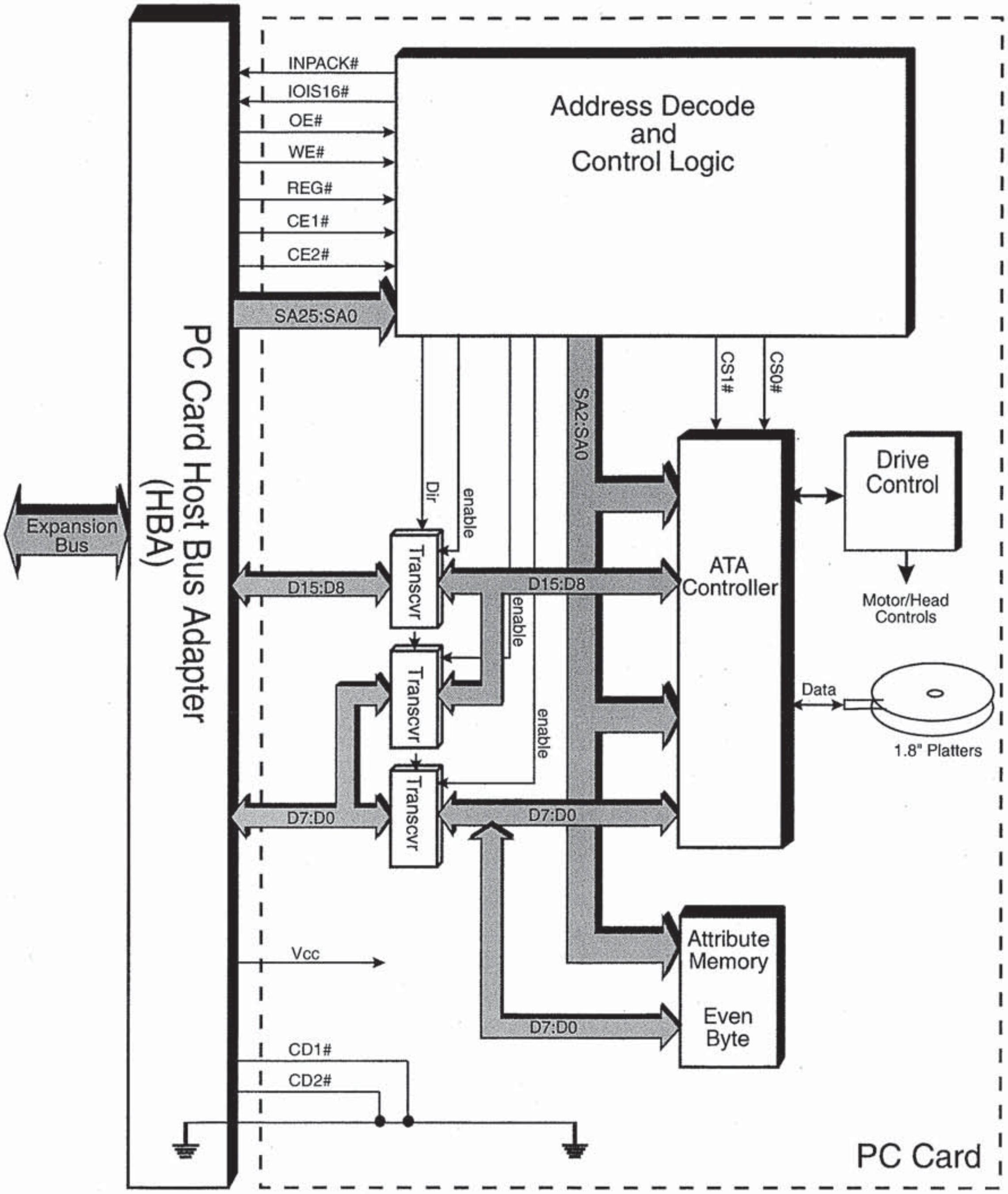

Figure 16-1. Functional Block Diagram of an ATA Disk Drive PC Card

ATA System Resource Requirements

ATA devices contain two register blocks called the command register block and control register block. Each of these register blocks must be assessable by the system. PC Card ATA devices support either I/O or memory-mapping using one of four addressing modes listed in table 16-1.

Standard mapping in the ISA environment includes the assignment of two separate I/O address ranges to map ATA drive registers into. If these ranges are not available, another range of I/O addresses can be used. If neither of the standard I/O address ranges are available, then a contiguous block of 16 I/O locations is requested for mapping the command and control block registers into.

Alternatively, the registers can be mapped into memory locations. When memory-mapping is chosen, a contiguous 2KB block of memory locations is used. The command and control registers are mapped into the first 16 bytes of the 2KB memory block, while the last 1KB of the block is used as a high speed buffer to transfer data to and from the PC card.

Table 16-1. ATA Addressing Options Supported by PCMCIA

Address Mode	Command Block	Control Block
I/O - Primary ATA drive address	1F0h - 1F7h	3F6h - 3F7h
I/O - Secondary ATA drive address	170h - 177h	376h - 377h
I/O - Any 16-byte contiguous range	XXX0h - XXXFh	
Memory - Any 2KB address range	Card must respond to locations 0h - Fh and 400h - 7FFh within the 2KB range	

In addition to mapping the registers, an interrupt request line must also be supported for I/O addressing. Normally IRQ 14 is used by ATA drives. When configured for memory-mapped registers, the socket interface does not define an interrupt line, therefore software polling must be used.

Supporting Two Drives

It is possible for two ATA drives to be simultaneously installed into PCMCIA sockets of the same HBA. When accessing these drives, some method must be used to individually select these drives as either drive 0 or drive 1. This is accomplished in a standard ATA environment via the daisy-chained cable with the cable-select signal or by jumpers (switches) on the drive. In the PCMCIA environment, the Socket and Copy Register, defined as one of the configuration registers, can be used to identify two ATA PC cards mapped to the same address space. The copy number programmed into the Socket and Copy Registers is used by the HBA to differentiate between drive 0 from drive 1.

The ATA Card's CIS

When an ATA card is installed, the normal process of calling client drivers that have registered with card services occurs. These client drivers attempt to identify the card installed to determine if it should be configured by them. ATA client drivers typically identify their card by interpreting one or more of the following CIS tuples:

- The JEDEC ID tuple.
- The Manufacturers ID tuple.
- The Function ID and Disk Device Function Extension tuples.

Once a PC Card has been detected as an ATA disk, the CIS can be further processed to determine the configuration options supported by the card.

The PC Card ATA specification defines Function Identification Extension tuples that are used to identify the disk as an ATA interface and to specify features supported by the ATA card. The Interface Function Extension tuple must immediately follow the Function Identification tuple that identifies the card as a disk device.

Disk Device Function Extensions

This tuple specifies additional information for disk devices. Two function extension types are currently defined. As shown in table 16-2, the first disk function extension tuple (type 01h) defines the type of interface used by the disk. An interface type of 01h indicates an ATA drive interface.

Table 16-2. Disk Function Extension Tuple Format (Type 1)

Offset	Disk Function Extension Tuple Format	
0	TPL_CODE	CISTPL_FUNCE (22H)
1	TPL_LINK	Link to next tuple
2	TPL_TYPE	Interface type extension (01h)
3	TPLFE_DATA	Interface type code (01h = ATA Interface)

A second Disk Function Extension defines additional ATA Card features as shown in table 16-3.

Table 16-3. PC Card ATA Function Extension Tuple

Offset	CIS	Tuple	Comments	Bit Fields						
00h	22h	cistpl_funce	ATA Function Extension tuple	Tuple Code						
02h	03h	link	This tuple has 3 info bytes	Link Length						
04h	02h	tplfe_type	Basic PC Card Extension tuple	PC Card ATA Basic Features						
06h	xxh	tplfe_data	PC Card ATA Features Byte 1	R	R	R	R	U	S	V
08h	xxh	tplfe_data	PC Card ATA Features Byte 2	R	I	E	N	P		

The bit fields illustrated in table 16-3 are defined in table 16-4 for normal operation and table 16-5 for low power modes.

Table 16-4. Bit Definition for Normal Operation

Name	Description	Values
V	Vpp Power	0 Not Required 1 Required for Media Modification Accesses 2 Required for all Media Accesses 3 Required Continuously
S	Silicon	0 Rotating Device 1 Silicon Device
U	Unique Drive Identifier	0 Identify Drive Model/Serial Number may not be unique 1 Identify Drive Model/Serial Number is guaranteed unique
R	Reserved	This field is reserved for future standardization. Must be 0.

Table 16-5. Bit Definition for Low Power Operation

P	**Low Power Modes** (Idle, Standby, Sleep)	Bit 3: 0 Low Power Mode Use Required to Minimize Power Bit 3: 1 Drive Automatically Minimizes Power. No need for host to actively power manage. Bit 2: 0 Idle Mode Not Supported Bit 2: 1 Idle Mode Supported Bit 1: 0 Standby Mode Not Supported Bit 1: 1 Standby Mode Supported Bit 0: 0 Sleep Mode Not Supported Bit 0: 1 Sleep Mode Supported
N	3F7/377 Register Inhibit Available	0 = All Primary and Secondary I/O Addressing Modes include ports 3F7 or 377. 1 = Some Primary or Secondary I/O Addressing Modes exclude 3F7 and/or 377 for floppy interference avoidance.
E	Index Emulated	0 = Index Bit is Not Emulated 1 = Index Bit is Supported or Emulated
I	IOIS16# on Twin Card	0 = IOIS16# use is Unspecified on Twin-Card Configurations 1 = IOIS16# is asserted only for Data Register on Twin-Card Configurations
R	Reserved	This field is reserved for future standardization. Must be 0.

IPL from a PCMCIA ATA Drive

To load the operating system from a PCMCIA ATA drive, the drive must be configured during main system initialization, commonly referred to as POST (power-on self test). The initialization byte within the Function Identification table specifies if a PC Card should be configured during POST.

Since in many systems PC Cards are not installed until the operating system loads, the system designer must provide PCMCIA initialization software. This software must read the CIS of all cards installed in sockets to determine if they should be configured before the operating system loads. Many of the vendors that supply socket services have a solution (i.e. ROM-based PCMCIA initialization code) that permits PC ATA cards and others requiring early configuration to be initialized during POST.

An Example ATA Card CIS

Figure 16-2 illustrates a memory map of the attribute memory address space used by a sample ATA card that implements rotating media. This example CIS supports all the addressing modes specified in table 16-1. Appendix D contains a detailed listing of this CIS.

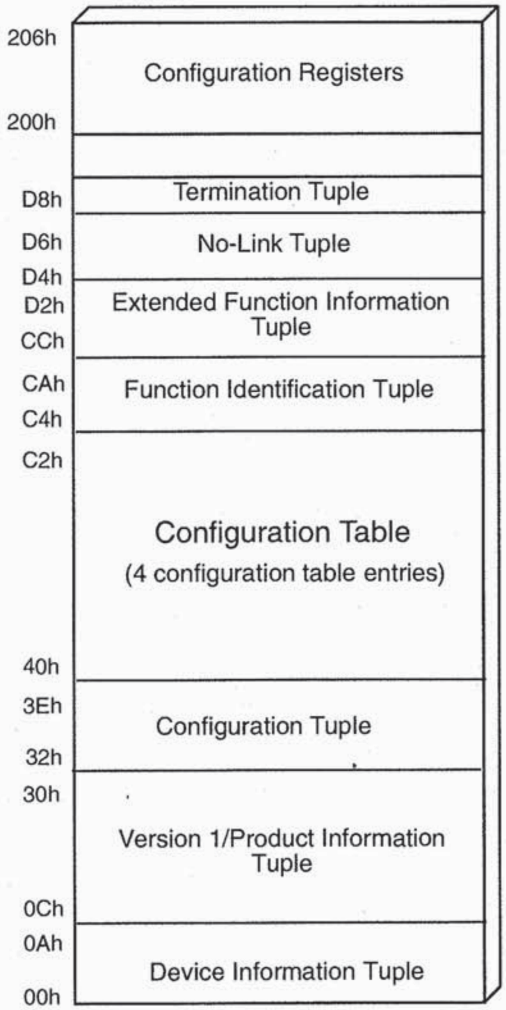

Figure 16-2. Sample ATA CIS and Configuration Register Map

Device Information Tuple

Since the ATA card contains a memory-mapped options for it registers, the Device Information tuple contains a valid memory device entry. The information described in this tuple includes:

- Memory type (specified as *function specific* memory)
- Extended memory speed defined (400ns)
- Memory size (2KB)

Level 1 Version / Product Information Tuple

This tuple contains the PCMCIA compliance level of the CIS (level 1 version) and ASCII characters describing the product. The data area within the ATA level 1 version/production information tuple consists specifically of:

- Major version 4 (indicates 2.x compliant CIS)
- Minor version 1 (indicates 2.x compliant CIS)
- ASCII string indicating manufacturer
- ASCII string indicating model information

The ASCII character strings contained within the product information portion of the tuple are left for the manufacturer to define. The manufacturer name and card description is sometimes read and displayed by PCMCIA utilities when a card is configured. This tuple is also used by client device drivers that are designed to identify a specific card.

Configuration Tuple

The Configuration tuple identifies the type of the configuration register(s) used by the PC Card, along with their location within attribute memory space. Data entries within the Configuration tuple contain the following:

- Size of address fields — this entry defines the number of bytes used by this tuple to identify the location of the configuration registers. Since these registers can be located anywhere within attribute memory address space (0 - 64MB), the number of bytes needed to define their location depends on where they reside in the address space. In this example, the registers are mapped to location 200h, therefore only two bytes are needed to specify their location.

- Size of configuration register mask field — specifies the number and mix of configuration registers implemented by the PC Card. A bit map of the configuration register identifies how many registers are implemented. PCMCIA currently defines four configuration registers, but provides expandability up to 32 configuration registers (requiring four 8-bit mask registers). This example implementation uses all four registers, therefore a signal mask register is implemented.
- Index number of the last entry in the configuration table. Since this example has four configuration entries, the index number of entry four is specified.
- Starting (base) address of the configuration registers — In this example, a two byte field identifies the location of the configuration registers in attribute memory (location 200h).
- Configuration register mask — Specifies that configuration registers zero (Configuration Option Register), one (Status Register) two (Pin Replacement Register) and three (Socket and Copy Register) are implemented.

Configuration Table

The configuration table contains the configuration option supported by the ATA card. This card in this particular example contains four entries within the configuration table, each defining a different combination of system resources required to support its functions. This card supports all four configuration options defined by the PCMCIA and ATA standards as listed in table 16-1.

Function Identification Tuple

The Function Identification tuple determines the type of functional device that is employed by the PC Card. This tuple defines the following items:

- Function type code — consists of a code representing the type of device employed by the PC Card. The function type associated with the ATA card is fixed disk.
- Initialization byte — specifies whether this device should be configured during system initialization (also called Power-On Self Test or POST) and whether the card has a ROM containing configuration

software. Since the ATA drive may need to load the operating system, the POST bit is set. This indicates that the system should configure this card during POST. Refer to the section entitled, "IPL from a PCMCIA ATA Drive", discussed earlier in this chapter.

Function Extension Tuples

Two Function Extension tuples are defined by PCMCIA for ATA drives. This sample CIS includes only the type 1 disk function extension that identifies the fixed disk interface type as ATA.

No-Link Tuple

This No-Link tuple indicates that when the Termination tuple is reached that no more tuples exist within the string.

Termination Tuple

The termination tuple consists only of the tuple code FFh. In this example, when processing software encounters the checksum tuple, it terminates tuple processing since the No-Link tuple was previously encountered in this tuple string.

Configuration Registers

The ATA card in this example implements all four configuration registers defined by the PCMCIA standard. These registers include the Configuration Option Register, Status Register, Pin Replacement Register and Socket and Copy Register.

Chapter 17

The Previous Chapter

The previous chapter described an example PC Card ATA drive implementation, including a functional block diagram, a sample CIS, and configuration registers implemented by the card.

This Chapter

This chapter discusses the multiple function PC Card strategy and the mechanisms for achieving it. It also includes a functional block diagram of a multiple function PC Card, a sample multi-function CIS, related configuration registers, and multi-function interrupt handling.

The Next Chapter

The next chapter provides an overview of the PCMCIA software environment and the configuration process.

Overview

Since most systems implement a limited number of PC Card sockets (usually one or two), it is advantageous to implement cards containing multiple functions. However, prior to release of the PC Card standard PCMCIA did not offer full support for multiple function PC Cards. Only one CIS structure and only one set of configuration registers were specified for a PC Card. This meant that each function had to somehow share the single CIS and configuration registers. Several multiple function cards have been designed, but these implementations are typically vendor-specific/proprietary solutions and require vendor-specific client drivers that have been designed with knowledge of the implementation.

The PC Card standard now incorporates a multiple function card strategy that specifies how multiple functions must be implemented. This permits software solutions that are aware of the multiple function implementation to recognize and configure multiple function PC Cards. An important part of this implementation is the definition of a separate CIS and configuration registers for each function implemented within the PC Card. This chapter discusses the multiple function PC Card strategy and the mechanisms for achieving it.

An Example Multiple Function PC Card

Figure 17-1 illustrates a functional block diagram associated with a multiple function PC Card. Each function has its own CIS mapped into the PC Card's attribute memory address space, along with its own set of configuration registers. Note in this PC Card example that both functions require use of interrupts. Since a PC Card memory or I/O socket interface defines only one IREQ# pin, it is necessary to share the IREQ# pin between functions. The interrupt requests from the functions are labeled IREQ0# and IREQ1# respectively, which are inputs to the interrupt routing logic illustrated in figure 17-1. The interrupt routing logic also includes inputs named INTR0 and INTR1 from the configuration registers. These inputs represent the state of the INTR bit in the configuration status register. When the INTR bit is cleared, the interrupt routing logic knows that the corresponding interrupt request has been serviced, and that it is free to generate another IREQ# to the HBA. Interrupt sharing is discussed in the section entitled, "Shared Interrupt Handling" later in this chapter.

An Example CIS

Each function within a multiple function PC Card must have its own CIS. However, some information specified within a CIS is common to the PC Card itself (i.e. the information applies to all functions implemented by the PC Card). For this reason multiple function PC Cards contain a global CIS along with separate CISs for each function implemented. Since each function has its own CIS, it can specify the location of the configuration registers needed to support its function. Figure 17-2 illustrates a multi-function CIS structure that includes two functions.

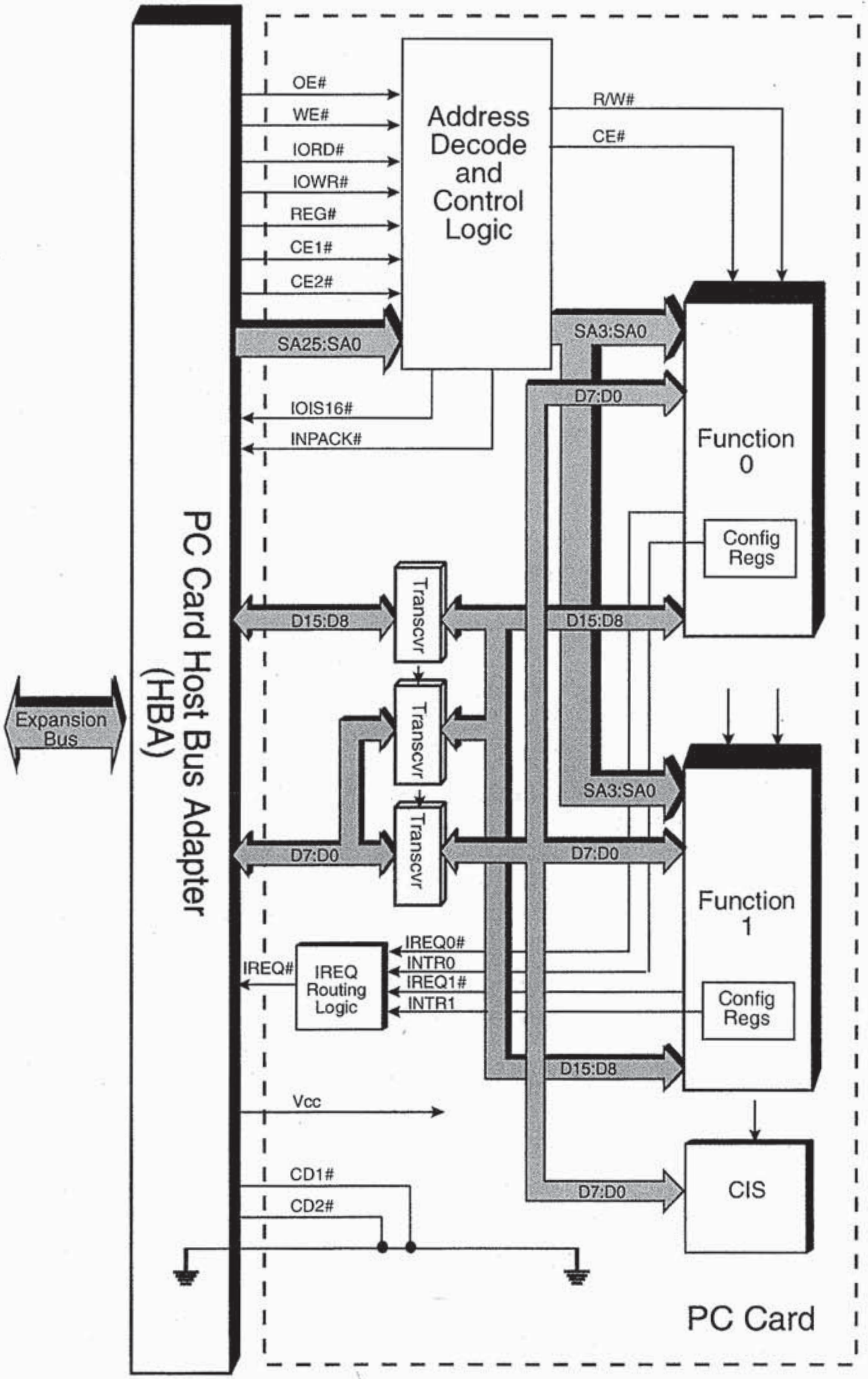

Figure 17-1. Functional Diagram of a Multiple Function PC Card

Every multiple function PC Card that is compliant with the standard must include a primary CIS that contains a LongLink_MFC tuple. This tuple specifies the location within attribute memory of the function-specific CISs that are required for each function implemented within the PC Card. Each function-specific CIS must begin with a LINKTARGET tuple to verify the start of the CIS. The standard specifies the tuples that must be included within the primary CIS, which ones are optional, and their exact order within the CIS. Table 17-1 lists these tuples in the required order.

Table 17-1. Tuples Defined for the Primary CIS (Listed in the Order)

Tuple Name	Required/Optional	Description
CISTPL_DEVICE	Required	Specifies whether memory is implemented within PC Card's common memory address space. If common memory is not used, the type code must be NULL.
CISTPL_MANFID	Optional	Only one manufacturer's ID tuple can be implemented.
CISTPL_VERS_1	Optional	May be used by enabling software to identify the PC Card.
CISTPL_LONGLINK_MFC	Required	Specifies the number of functions (i.e. the number of configuration register sets) within the PC Card, and the starting address of each function-specific CIS within attribute memory space.

The standard also specifies the order and combination of tuples required for each secondary CIS. These tuples are listed in table 17-2.

Table 17-2. Tuples Defined for each Secondary CIS (Listed in the Order)

Tuple Name	Required/Optional	Description
CISTPL_LINKTARGET	Required	Used to validate the beginning of a function-specific CIS.
CISTPL_FUNCID	Required	Must be used to identify the function.
CISTPL_FUNCE	Optional	Some functions have extensions that specify additional information about the function.
CISTPL_CONFIG	Required	Describes presence and location of Function Configuration Registers for this function.
CISTPL_ENTRY	Required	Specifies the configuration requirements of this function.

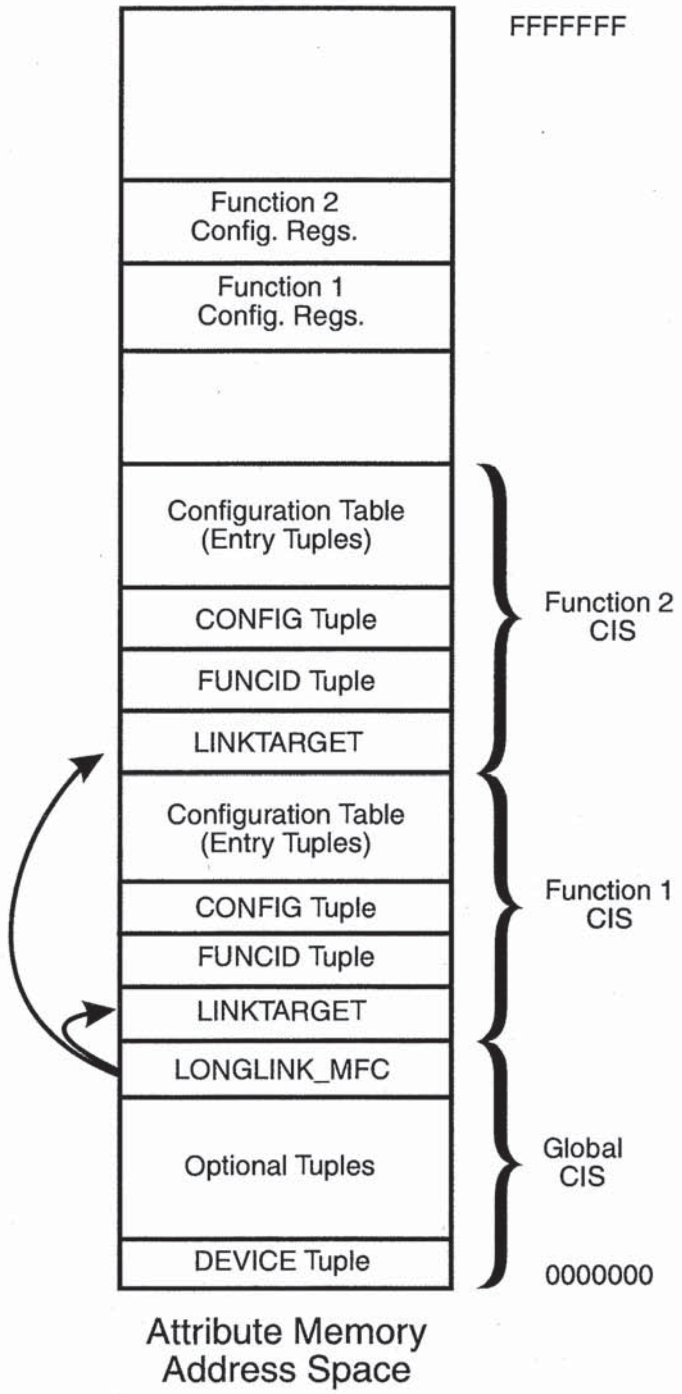

Figure 17-2. An Example CIS Structure Supporting Two Functions.

Configuration Registers

Each function contains its own set of configuration registers and may include the registers illustrated in table 17-3. The exact set of registers employed by each function depends on the requirements of the particular function being implemented. Each function that uses the I/O interface must include the Configuration Option Register, the I/O Base and typically the I/O Limit registers (the I/O Size register may be eliminated as discussed below in the section entitled "I/O Limit Register"), all other registers are optional.

Once the PC Card's enable has correctly identified the functions within the card, it must configure the HBA and PC Card. Configuring the PC Card means writing the appropriate values into the configuration registers that have been implemented. Refer to the chapter entitled "The Configuration Registers" for a detailed explanation of each register.

Table 17-3. The Configuration Registers Defined by the PC Card Standard

Offset	7	6	5	4	3	2	1	0
0	**Configuration Option Register**							
	SRESET	LevlREQ	Function Configuration Index					
2	**Configuration and Status Register**							
	Changed	SigChg	IOIS8	RFU	Audio	PwrDwn	Intr	IntrAck
4	**Pin Replacement Register**							
	CBVD1	CBVD2	CREADY	CWProt	RBVD1	RBVD2	RREADY	RWProt
6	**Socket and Copy Register**							
	RFU	Copy Number			Socket Number			
8	**Extended Status Register**							
	Event3	Event2	Event1	Req Attn	Enable3	Enable2	Enable1	Req Attn Enable
10	**I/O Base 0**							
12	**I/O Base 1**							
14	**I/O Base 2**							
16	**I/O Base 3**							
18	**I/O Limit**							

Configuration Option Register

The configuration option register (COR) has a specific definition (different from single function PC Cards) when employed within multiple function PC Cards. Specifically, the configuration index field is different from the single function implementation. Recall that in a single function PC Card the configuration index field can be defined in any fashion that the single function card designer chooses, which specifies a given configuration for the card (i.e. a value corresponding to the index number of the configuration table entry that specifies the configuration chosen by the enabler). However, the multi-function PC Card must implement the configuration index field as specifically defined in table 17-4.

Note that the definition of the SRESET and LevlReq bits are the same as for single function cards. Each bit is defined in table 17-4.

Table 17-4. Configuration Option Register format and Definition

7	6	5	4	3	2	1	0
SRESET	LevlReq	Configuration Index					

SRESET	**Software Reset**. Setting this bit to one (1) places the card in the reset state. This is equivalent to assertion of the RESET signal except that this bit is not cleared. Returning this bit to zero (0), leaves the card in the same state that follows a hardware reset. This bit is set to zero by power up and hardware reset.
LevlReq	**Level Mode IREQ#**. Level Mode Interrupts are selected when this bit is one (1). Pulse Mode Interrupts are selected when bit is zero.
Conf Index	**Multi-function Card Index definition.** The PC Card standard specifically defines use of each bit within the configuration index. **Bit 0** — Enables/disables this function. 1=enabled; 0=disabled **Bit 1** — Specifies the number of I/O addresses used. 1=I/O function uses the number of address lines specified by the base and limit registers; 0=all host I/O address are passed to the function. (This bit is valid only when function is enable via bit 0.) **Bit 2** — Enables IREQ# routing. 1=the function will deliver interrupts to the PC Card's IREQ# line; 0=interrupts disabled for this function. (This bit is valid only when function is enabled.) **Bits 3-5** — vendor specific

Card Configuration and Status Register

Portions of the Card Configuration and Status Register (CSR) have also been redefined to support interrupt sharing on multi-function PC Cards. A new bit named IntrAck (interrupt acknowledge) specifies how the Intr bit is implemented. Refer to table 17-5

- Single function PC Cards with IntrAck reset (0) — the Intr bit remains set until the interrupt service routine is executed, at which time the Intr bit is reset.
- Multiple function PC Cards with INTRack set (1) — the Intr bit remains set even though the interrupt service routine has already serviced the interrupt request. Normally, the interrupt service routine clears an interrupt pending bit within a function specific register, causing the Intr within the CSR also to be cleared. However, to support interrupt sharing the Intr bit is not cleared until card services is ready to handle the next interrupt request. When cleared by card services, other interrupt requests that are pending can now be generated via the PC Card's IREQ# pin.

Table 17-5. Card Configuration and Status Register and Definition

7	6	5	4	3	2	1	0
Chng	SigChg	IOis8	Resrv (0)	Audio	PwrDn	Intr	IntrAck
Chng	**Status Change Detected**. This bit indicates that one or more of the Pin Replacement Register bits (CBVD1, CBVD2, CRDY, or CWProt) is set to one, normally causing the STSCHG# signal to be asserted. However, if the SigChg bit (see below) is 1 and the card is configured for an I/O interface, the STSCHG# pin is asserted when this bit is set.						
SigChg	**Signal Change Enable/Disable**. This bit is set and reset by the host to enable and disable a status-change signal from the status register. When this bit is set and the card is configured for the I/O interface, the Chng bit controls pin 63 (STSCHG#). If no status change signal is desired, this bit should be set to zero and the STSCHG# signal will be held deasserted when the card is configured for I/O.						
IOis8	**I/O Cycles Occur Only as 8-bit Transfers.** When the host can provide I/O cycles only using the D7:D0 data path, the PCMCIA software will set this bit to a 1. The card is guaranteed that accesses to 16-bit registers will occur as two byte accesses rather than a single 16-bit access. This information is useful when 16-bit and 8-bit registers overlap.						
Resrv	Reserved bits must be 0.						

Table 17-5. Card Configuration and Status Register and Definition (Continued)

<table>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>Chng</td><td>SigChg</td><td>IOis8</td><td>Resrv
(0)</td><td>Audio</td><td>PwrDn</td><td>Intr</td><td>IntrAck</td></tr>
<tr><td>Audio</td><td colspan="7"><b>Audio Enable.</b> This bit enables audio information to be sent to the HBA via the speaker pin when configured for an I/O interface.</td></tr>
<tr><td>PwrDn</td><td colspan="7"><b>Power Down.</b> This bit is set to one to request that the card enter a power-down state. PCMCIA software must not place the card into a power-down state while the card's READY pin is in the low (Busy) state.</td></tr>
<tr><td>Intr</td><td colspan="7"><b>Interrupt Request Pending.</b> This bit represents the internal state of the interrupt request. This value is available whether or not interrupts have been configured. How the Intr bit is cleared is dependent of how the IntrAck bit is configured.
<b>IntrAck=0</b> — Intr reflects the function's interrupt request status. If the interrupt is cleared within the function, then Intr is reset by the function.
<b>IntrAck=1</b> — Intr remains set even though the interrupt condition has been cleared. It is reset by system software to indicate it is ready to receive another interrupt (implemented to support interrupt sharing).</td></tr>
<tr><td>IntrAck</td><td colspan="7"><b>Interrupt Acknowledge.</b> This bit determines the response of the Intr bit. The functionality associated with the IntrAck bit permits two or more functions to share the PC Card's IREQ# pin.
<b>IntrAck=0</b> — when IntrAck is reset Intr functions as described above to support a single interrupt implementation.
<b>IntrAck=1</b> — This causes the Intr bit to remain set even though the interrupt service routine has already serviced the interrupt. The Intr bit is not cleared until Card Services is ready to handle the next interrupt request. When the Intr bit is cleared, the PC Card generates another interrupt request (if another interrupt request is pending from another function).</td></tr>
</table>

I/O Base Registers

The PC Card standard requires use of the I/O base registers by multiple function cards, and they can also be used by single function cards. These registers define the base I/O address at which the function's I/O registers will be mapped into the host processor's address space. The number of registers used depends on the address space supported by the host processor. Since Intel compatible x86 processors have 64KB of address space only the first two registers are needed to specify a base address anywhere within the entire 64KB space.

Note that in a typical single function PC Card the I/O address range is specified by the configuration index value within the configuration option register. This value identifies the configuration table entry that specifies the I/O address range that the PC Card has been configured to use.

I/O Limit Register

This register corresponds to the I/O base registers and specifies the maximum number of I/O addresses that can be mapped beginning at the base address. This register is bit mapped such that the most significant bit set within the register determines the number of address lines used to decode the address and therefore the maximum block of address space supported. The most significant bit and all bits of lesser significance must be set within the register. This results in the possible number of address lines as listed in table 17-6. Note that the largest block of I/O address space that can be defined is 256 bytes.

This register is optional and need not be implemented for each function if all functions within the PC Card use the same number of I/O address lines.

Table 17-6. Address Limit Associated with Function Base Address Register

Bit Position								Maximum Number of Address Locations
7	6	5	4	3	2	1	0	
# of Address Lines Defined by Bit position								
8	7	6	5	4	3	2	1	
0	0	0	0	0	0	0	0	Not defined
0	0	0	0	0	0	0	1	2
0	0	0	0	0	0	1	1	4
0	0	0	0	0	1	1	1	8
0	0	0	0	1	1	1	1	16
0	0	0	1	1	1	1	1	32
0	0	1	1	1	1	1	1	64
0	1	1	1	1	1	1	1	128
1	1	1	1	1	1	1	1	256

Shared Interrupt Handling

The PC Card standard defines an interrupt sharing mechanism that allows multiple I/O functions to share the PC Card's single IREQ# pin. This mechanism requires specific hardware and software support beyond that required for single function PC Cards. The changes required are:

- Multiple Function PC Card — interrupt sharing logic required.
- HBA — no changes required.
- Socket Services — no changes required.
- Card Services — provides ISR registration, must detect IRQ, determine which PC Card function generated the interrupt, and route the request to the interrupting function's enabler.
- PC Card Enabler — must support sharing protocol.

Review of Single Function Interrupt Handling

The following discussion reviews the interrupt handling procedures typically employed in single function PC Card implementations. This discussion is based on an x86-based system operating in real mode.

IRQ Initialization

The PC Card's enabler, after having determined the configuration requirements of the PC Card, requests a specific IRQ line from card services by making the RequestIRQ function call. Card services then verifies that the IRQ line is available by successfully completing the function call. The enabler now knows that it has acquired the IRQ that it wanted and must "hook" the interrupt (i.e. place the starting address of its interrupt service routine into the interrupt table entry that corresponds to the IRQ line that it has been assigned) so that interrupt requests are directed to its interrupt service routine (ISR).

Next, the enabler requests that card services configure the HBA so that it steers the PC Card's IREQ# line to the specified IRQ line on the expansion bus (using the RequestConfiguration function call). Card services in turn makes the appropriate calls to socket services, directing it to load the appropriate registers within the HBA; thereby, setting it up to steer the PC Card's interrupt requests over the specified IRQ line.

Handling the Interrupt Request

A summary of the events that take place when a PC Card generates an IREQ# are detailed in the following paragraphs.

When a PC Card generates an interrupt request, it sets its interrupt pending bit in the CSR register and asserts the IREQ# line. The HBA steers the PC Card's IREQ# to the selected IRQ line and on to the interrupt controller. The interrupt controller responds by asserting the processor's interrupt request input (INTR). This causes the processor to cease normal program execution and to interrogate the interrupt controller to find out which interrupt has occurred. The interrupt controller responds by sending the interrupt table entry number corresponding to the IRQ line that generated the interrupt request. The processor receives the entry number (aka vector) and performs a memory read to get the starting address of the interrupt service routine from the interrupt table.

The processor temporarily stores the ISRs starting address in a special register (not named) and saves the current status of the program that was being executed when the interrupt occurred (i.e. pushes the flags, CS, and IP registers to the stack). This is done so the processor can return to the original program after the interrupt has been serviced. Once the processor saves its place, it then moves the ISRs starting address into the CS and IP registers, causing it to begin fetching and executing instructions from the PC Card's interrupt service routine.

The ISR reads the Configuration Status Register (CSR) to verify that an interrupt request is pending (i.e. the Intr bit is set). If the Intr bit is set, the ISR recognizes that an interrupt is pending and clears the Intr bit since the interrupt is now being serviced.

After clearing the interrupt within the PC Card, the ISR continues execution. Before the ISR completes it must also clear the interrupt at the interrupt controller to prevent the interrupt from being serviced again (i.e. the interrupt controller will send the same vector to the processor, causing the same ISR to be executed again). The interrupt is cleared by issuing an End Of Interrupt (EOI) command to the interrupt controller. After the EOI command has been issued and the interrupt has been serviced, the ISR executes an Interrupt Return instruction (IRET). The IRET causes the processor to restore the flags, CS, and IP registers previously saved, returning it to normal program flow.

Note: for a more in-depth discussion of x86 interrupt handling refer to the MindShare book entitled *ISA System Architecture*, published by Addison-Wesley.

Multiple Function Interrupt Handling

Each function within a multiple function PC Card has its own enabler that includes an interrupt service routine designed specifically for that function. The following sections detail the interrupt handling procedures for multi-function PC Cards.

IRQ Initialization

Multiple function IRQ initialization must be handled differently than single PC Card initialization. When a given enabler detects that its function is implemented within a multiple function PC Card it reads the function specific CIS, determines the configuration requirements of its function and initiates the configuration of the HBA and PC Card. Since a PC Card has a single IREQ# pin, all functions within the PC Card must share the same interrupt line.

Interrupt sharing is managed by card services. The interrupt sharing mechanism requires that the ISR for each function be registered with card services. The following describes the actions that would typically be taken by each function enabler during IRQ initialization.

Function Zero

When an enabler detects the presence of its function within a multi-function PC Card and determines that an interrupt is required, it must request an interrupt from card services. The multiple function enabler passes the starting address of its ISR to card services when it makes the RequestIRQ function call. It also identifies the location of its function by passing card services the logical socket number and logical function number (zero in this example) for its function.

Card services then provides a first level interrupt handler (FLIH) by hooking the interrupt table entry corresponding to the interrupt requested by the enabler. Note that multiple function enablers register their ISR with card services and do not directly hook the interrupt. When the interrupt is generated card services FLIH will be executed.

Function One

When function one's enabler detects its function within the multiple function PC Card, it must also request an interrupt (via the RequestIO service) from card services. When making the service call, the enabler passes the starting address of its interrupt service routine to card services and specifies the logical socket and function number (one in this example) of the PC Card. After the HBA and PC Card are configured, an interrupt generated by function one will cause the FLIH within card services to execute.

Handling the Interrupt Request

A summary of the events that take place when a multiple function PC Card generates an IREQ# are detailed in the following paragraphs. The example is based on an ISA platform. Refer to Figure 17-3.

When a single function within a multiple function PC Card generates an interrupt request it sets the Intr bit in its CSR register and signals the PC Card's interrupt routing logic. The routing logic in turn asserts the PC Card's IREQ# line. The HBA steers the IREQ# signal to the selected IRQ line and on to the interrupt controller. The interrupt controller responds by asserting the processor's interrupt request input. This causes the processor to cease normal program execution and to interrogate the interrupt controller to find out which interrupt has occurred. The interrupt controller responds by sending the processor an 8-bit interrupt table entry number corresponding to the IRQ line that generated the interrupt request. The processor receives the entry number (a.k.a. the vector) and performs a memory read to get the starting address of the card services FLIH from the interrupt table.

The processor temporarily stores the FLIH's starting address in a special register (not named) and saves the current status of the program that was being executed when the interrupt occurred (i.e. pushes the flags, CS, and IP registers to the stack). This is done so the processor can return to the original program after the interrupt has been serviced. Once the processor saves its place, it then moves the FLIH's starting address into the CS and IP registers, causing the processor to begin fetching and executing instructions from card services FLIH.

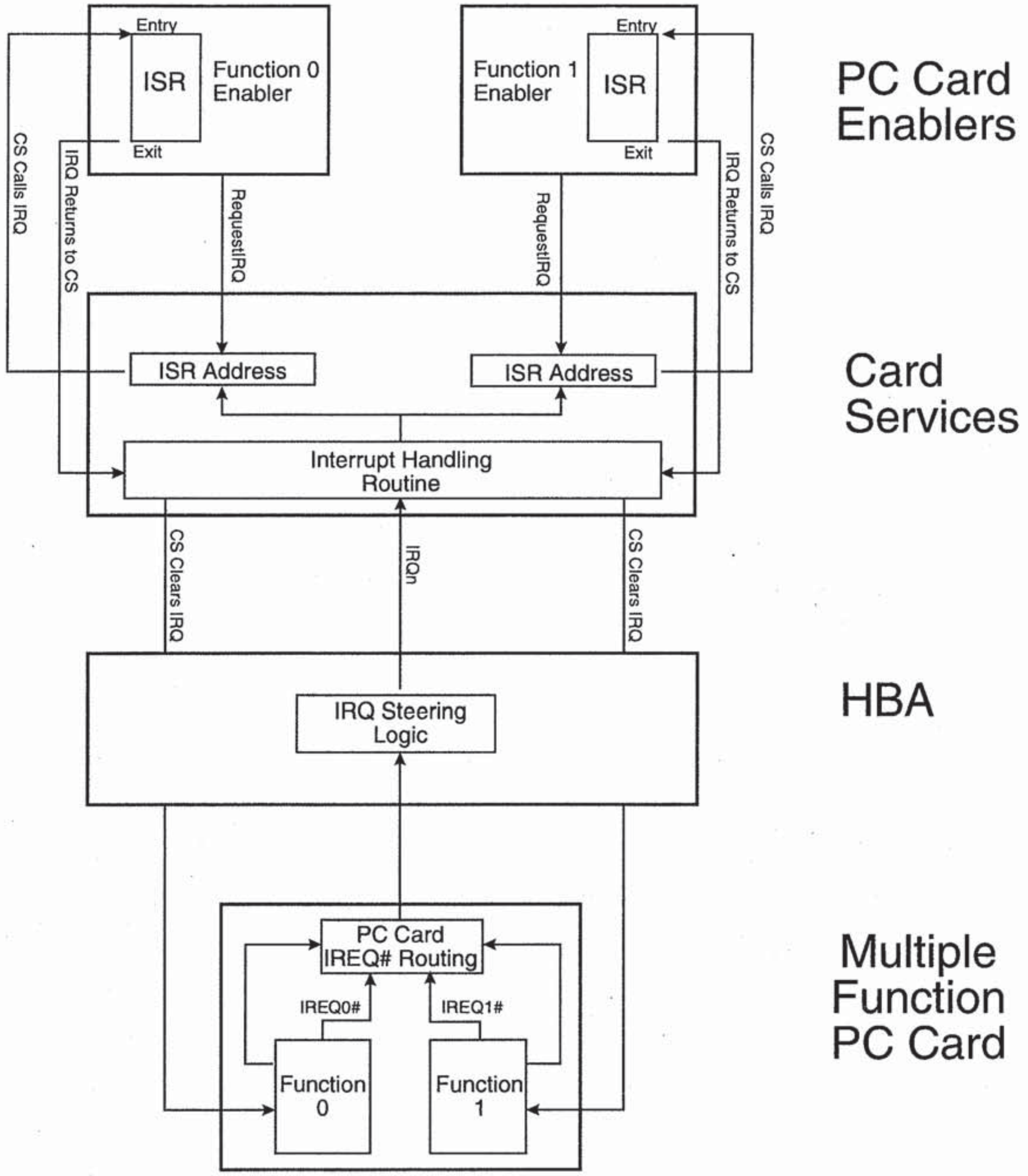

Figure 17-3. Multiple Function IRQ Sharing Procedure.

The FLIH reads the function Configuration Status Registers (CSRs) to determine which function currently has an interrupt request pending (i.e. the function whose Intr bit is set). If the Intr bit is set for one of the functions, the FLIH calls the ISR for that function, using the starting address that the enable passed to card services when the RequestIRQ function was performed.

The function's ISR does not clear the interrupt at the function's CSR, nor at the interrupt controller as single function ISRs do. When the ISR completes execution, it returns to the FLIH. Before the FLIH completes, it issues an EOI command to the interrupt controller, preventing it from servicing the same interrupt again. The FLIH also clears the Intr bit within the CSR, indicating that card services is ready to handle another interrupt request. This prompts the interrupt routing logic to issue another IREQ#, if another function within the card has signaled that it has an interrupt request pending. After the EOI command has been issued, the FLIH executes the IRET instruction, returning the processor to normal program flow.

Applications Unaware of Multiple Function Protocol

The Problem

Generic enablers for some functions (e.g. modems) request specific resources that common application program expect the function to use (e.g. many communications programs expect the modem to use the convention I/O address space and IRQ lines associated with COM1 or COM2). If two or more functions within a single PC Card require specific IRQ lines, then the interrupt sharing mechanism will not work. However, the PC Card Standard permits one of the functions within a multiple function card to request a specific IRQ that it requires to maintain compatibility with application programs. The enabler for functions that require a specific IRQ does not participate in the interrupt sharing protocol. Note however, that all other functions within the multiple function PC Card must support the interrupt sharing protocol.

An Example Solution

As an example, a generic modem enabler, being unaware that multiple function support exists, will not register its ISR with card services. Therefore, when the enabler calls the RequestIRQ function the ISR address field will be zero. (Note that card services permits only one enabler per socket to specify an ISR address field of zero.) Card services assigns the specific IRQn to the modem enabler to satisfy its configuration. The modem enabler then "hooks" the interrupt (places the starting address of its ISR into the interrupt table entry corresponding to the IRQn line that it has been assigned). Next, card services "hooks" the same interrupt by reading and saving the starting address of the modem's ISR and replacing it with the starting address of the FLIH.

Enablers for other functions within the PC Card must register their ISRs with card services. When any of the functions within a PC Card generate an interrupt request, the FLIH will be executed first (because the processor will obtain the starting address of the FLIH when it obtains the starting address of the ISR in the interrupt table). The FLIH checks the interrupt pending bits within each function to detect which has an interrupt pending.

If the modem has an interrupt pending, the FLIH jumps to the entry point of the modem's ISR (recall that card services previously read and saved the starting address of the modem's ISR when it installed the FLIH in the interrupt table). The modem's ISR executes normally by clearing the PC Card's interrupt request (for level interrupts) and performing the EOI command, and executing IRET.

Changes to Card Services Functions

To support multiple function PC Cards, many of the card services functions have been modified. For example, when accessing a single function PC Card, the function could be identified by merely specifying the logical socket number in which the PC Card resided. However, when a PC Card contains more than one function each function within the PC Card is identified by an additional logical function number. Table 17-7 lists of services that have added support for multiple function implementations.

Table 17-7. Card Services Modified for Multiple Function Support

Service Name	Code
AccessConfigurationRegister	36h
GetCardServicesInfo	0Bh
GetConfigurationInfo	04h
GetEventMask	2Eh
GetFirstClient	0Eh
GetFirstTuple	07h
GetNextClient	2Ah
GetNextTuple	0Ah
GetStatus	0Ch
GetTupleData	0Dh
ModifyConfiguration	27h
RegisterMTD	1Ah
ReleaseConfiguration	1Eh
ReleaseExclusive	2Dh
ReleaseIO	1Bh
ReleaseIRQ	1Ch
ReleaseSocketMask	2Fh
RequestConfiguration	30h
RequestExclusive	2Ch
RequestIO	1Fh
RequestIRQ	20h
RequestSocketMask	22h
ResetFunction	11h
SetEventMask	31h

Part Four

PCMCIA Software

Chapter 18

The Previous Chapter

The previous chapter discussed the multiple function PC Card strategy and the mechanisms for achieving it. It also included a functional block diagram of a multiple function PC Card, a sample multi-function CIS, related configuration registers, and multi-function interrupt handling.

This Chapter

This chapter provides an overview of the PCMCIA software environment and the configuration process. The primary role and interaction between each piece of software is established. This chapter also introduces the common software solutions provided along with the most popular suppliers.

The Next Chapter

The next chapter discusses the role of socket services and the initialization process. It also defines each function and details the calling interface.

Overview of the Configuration Process

Each PC Card must have an enabler that recognizes it, reads the CIS to determine the PC Card's resource requirements, programs the host bus adapter (HBA) and configures the card. Figure 18-1 illustrates the most common form of PC Card enabler known as the client driver. Client drivers interface directly to Card Services, which services requests from the client drivers. Client drivers call a variety of services within card services to assist it in configuring and controller accesses to its PC Card. Using card services greatly simplifies the job of enabling the PC Card, monitoring status change events, and controlling access to the card.

As illustrated in figure 18-1, card services interfaces directly to socket services to gain access to the HBA and PC Card. Socket services is designed with specific knowledge of the HBA hardware design and contains software routines that card services can call to gain access to the registers within the HBA without having to know the low-level details of the hardware interface.

Configuring a PC Card may take place when the system powers up (if the PC Card is already installed in a socket), or when a PC Card is inserted into a socket (after the system is powered up and fully operational). In either case, the PC Card must be detected by an enabler and configured. Without an enabler, a PC Card would never be recognized by the system. Once a PC Card is configured, it then responds like any other device residing on the host bus.

This configuration processor involves interaction between a client driver, card services, socket services, and the PC Card's CIS. The Role of each of these items is reviewed below.

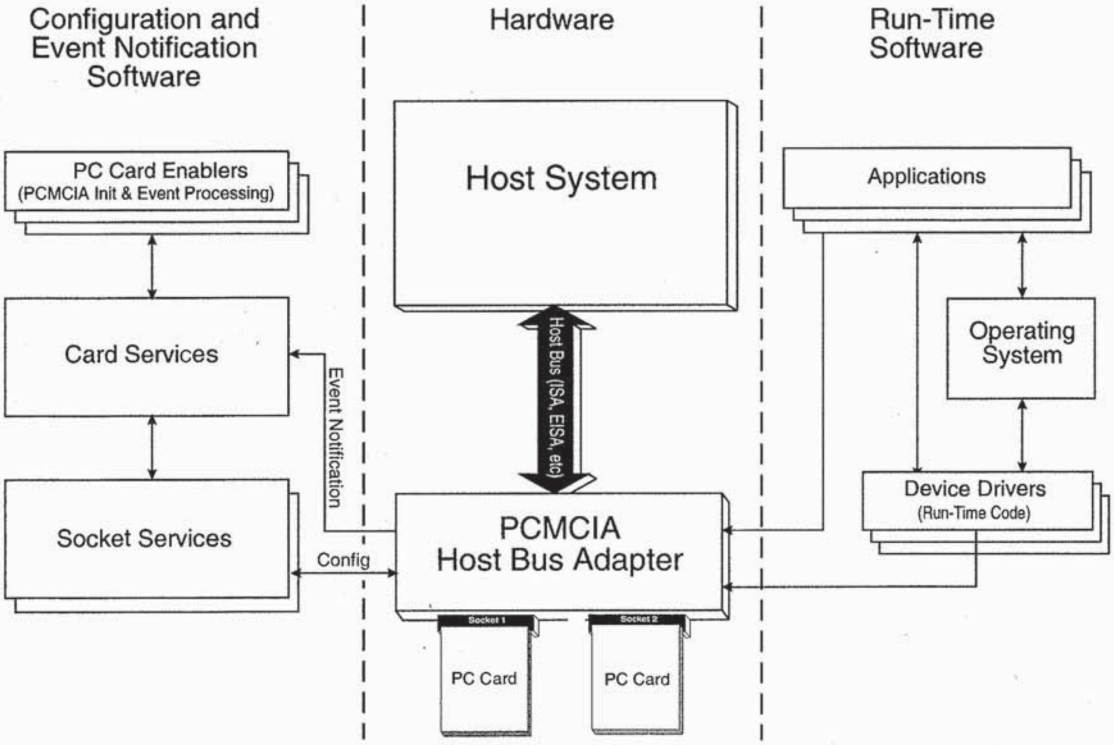

Figure 18-1. PCMCIA Software Flow

The Role of the CIS

Each PC Card is required to have a Card Information Structure, or CIS to be compliant with 2.x or PC Card implementations. The CIS is a data structure that is stored in non-volatile memory, which provides a method for software to determine what kind of PC Card is installed, along with its speed, size, system resources required by the card, and other pertinent information. The CIS is mapped within the attribute memory space or alternatively can be located in common memory address space.

As illustrated in figure 18-1, the CIS is read by PC Card client drivers (via card and socket services) during card initialization to determine the configuration options supported by the card. Once the card type and resource requirements have been read from the CIS, the PC Card client driver programs the HBA and configures the PC Card, again via card and socket services. No further access is typically made to the CIS after the card has been initialized. The PC Card can now be accessed via the host expansion bus, just like any other expansion device. Note that the CIS is only accessed by programs that are PCMCIA aware. Most application programs have no knowledge that they are accessing devices implemented in PC Card packages.

The Role of the Socket Service Functions

Socket services provides a set of software routines written specifically to access the registers within a given HBA. Socket services eliminates the need for special knowledge of the HBA hardware programming interface. These routines or functions are comparable to the BIOS routines that are used in the PC environment. In practice, most client drivers seldom, if ever, directly access socket service functions, because properly designed enablers access the HBA via card services. Card services, makes calls to socket services at the HBA request.

The Role of Card Services

Card services provides a central resource available to all client drivers. Specifically, card services is a collection of service routines designed for use by programmers writing enablers for PC Cards. These services provide a software interface that permits the programmer to simplify code and helps to reduce conflicts with other client drivers and with allocating system resources for PC Cards.

A major function of card services is to provide call-back services to notify the enablers that a particular event has occurred. Each enabler must register with card services and specify which PC Card events that it wishes to be notified of. When card services detects a given event (e.g. a card has been inserted or removed) it then calls each enabler that previously registered to receive notification of card insertion or removal.

The Role of the PC Card Enabler

PC Card enablers must recognize that a PC Card has been installed and access the card's CIS to determine if it should attempt to configure the card.

Three basic types of enablers exist:

- Dedicated enablers — designed for a particular PC Card.
- Generic (Super) enablers — designed for a wide range of PC Card types.
- Point enablers — designed to configure and enable the PC Card without using card and socket services.

Note that dedicated enablers and generic enablers both interface to card services as illustrated in figure 18-1. These enablers all register with card services when they first install. The registration process permits access to card services and allows the enabler to specify the events that it wishes to be notified of. Enablers that use card services are also referred to as client drivers.

Dedicated Enablers

Dedicated enablers are typically supplied by the PC Card manufacturer to increase the probability that the card will install correctly in the absence of a generic driver. Dedicated enablers identify a specific PC Card and will typically not recognize and enabler other PC Cards of the same type. These enablers may also manage functions that are unique to a given PC Card implementation.

Generic Enablers

Generic enablers are designed to handle PC Cards of a particular functional type. For example, the system manufacturer may include generic drivers for card types such as SRAM, flash ROM, Modems, and ATA drives. These enablers attempt to identify and enable cards based on a generic type without regard to the manufacturer or special features that may be incorporated into the PC Card's design.

Another class of generic enablers are the super I/O enablers. These enablers are designed to recognize and configure a wide range of I/O devices such as, modem, fax/modems, LAN controllers, etc. These enablers reduce the number of enablers that must be installed to detect the possible PC Cards that might be installed in a socket. The exact mechanism employed by these super enablers varies, but all have the same goal of enabling the most common I/O cards. Most system manufacturers supply super I/O client solutions as a part of the PCMCIA software shipped with the PC.

Point Enablers

Point enablers are dedicated enablers that bypass card and socket services. These enablers are popular in environments such as DOS where limited memory address space is available for application programs. Card and socket services take a considerable amount of memory when they install. Added to this is the space required by the enabler(s) and any TSR (terminate and stay resident programs) that might be used. As a result, too little memory is left for many application programs to run. One solution is to eliminate the PCMCIA specific software, thereby freeing up memory space that is needed to run the application programs. Point enablers are needed to configure the PC Cards that the user want to access. In the absence of card and socket services, point enablers must load the appropriate registers within the HBA to recognize and configure their PC Card.

For more information regarding enablers refer to the Chapter entitled, "PC Card Enablers."

PCMCIA Software Solutions

The entire PCMCIA software environment is typically provided by a single vendor. This software includes generic enablers, card services, a resource detection utility that builds the resource table (used by card services), and socket services. PC manufacturers license these software solutions for use in their products. PCMCIA software is available from several different vendors. The major vendor and the name of their PCMCIA software is listed in table 18-1.

Table 18-1. Major Vendors of PCMCIA Software Solutions

PCMCIA Software Vendor	Product Name
American Megatrends (AMI)	AMICARDZ
Award	CardWare
IBM	PlayAtWill (DOS & OS/2)
Microsoft	Windows95
Phoenix Technology	PCM3+
SystemSoft	CardSoft

While most PCMCIA software solutions provide the same basic functionality, many differences have existed. Some of the differences are inconsequential, such as, differences in logical drive letter assignments for various types of PC Cards, the visual and/or audible feedback provided when cards are inserted or removed, etc. However, some differences have been potentially more critical, including:

- HBAs supported
- Power management support
- Flash card support (i.e. Flash file systems and MTDs)
- Abridged versions of card services (Note that the functionality not included in card services is typically integrated directly into the enablers.)
- Resource Allocation (PC Cards mapped to different system resources)
- Generic enabler support (Types of PC Cards supported)

As the PCMCIA software has matured, the problematic differences between vendor solutions have diminished. Further, the PC Card 95 release has defined specific support for several areas that were previously the source of significant differences between vendor solutions.

Chapter 19

The Previous Chapter

The previous chapter provided an overview of the PCMCIA software environment and the configuration process. The primary role and interaction between each piece of software was established. The chapter also introduced the common software solutions provided along with the most popular suppliers.

This Chapter

This chapter discusses the role of socket services. It also describes the initialization of socket services and explains the basic purpose of the functions commonly supported in the PC environment.

The Next Chapter

The next chapter focuses on the role of card services in the PCMCIA environment. It reviews each of the functions defined by the PC Card specification that apply to 16-bit PC Cards, along with related return codes. The call back mechanism is also described and the event and call back codes are defined.

The Role of Socket Services—Making Life Easier

Before the development of socket services, a PC Card's client driver was responsible for ensuring that its card satisfied the requirements of the PC Card plug and play environment. Plug and play means that the PC Card can be automatically configured after being installed in a system, without requiring user intervention. In the PC Card environment this responsibility includes:

- Accessing registers within the HBA to open an attribute memory window, allowing access to the card's CIS.
- Interpreting the CIS to determine the configuration requirements of the card.

- Determining if the resources needed by the card are available (not already in use by other system devices).
- Loading HBA registers with the specified configuration values that permit host software to access the PC Card.
- Polling HBA registers to monitor socket status change events (e.g. card removal).
- Releasing system resources by clearing registers in the HBA when a card removal event occurs.
- Providing the ability to perform these functions regardless of the HBA design.

These requirements make it clear that developing PC Card client driver prior to the introduction of socket services required detailed knowledge of the particular HBA's hardware interface. Furthermore, HBA design changes could lead to heavy revision and update of the client driver.

As shown in figure 19-1, today's client drivers can configure a PC Card with relative ease by accessing the PCMCIA configuration software that is comprised of card and socket services. This chapter focuses on the role of socket services, which eliminates the need for client drivers to know the details of the HBA hardware.

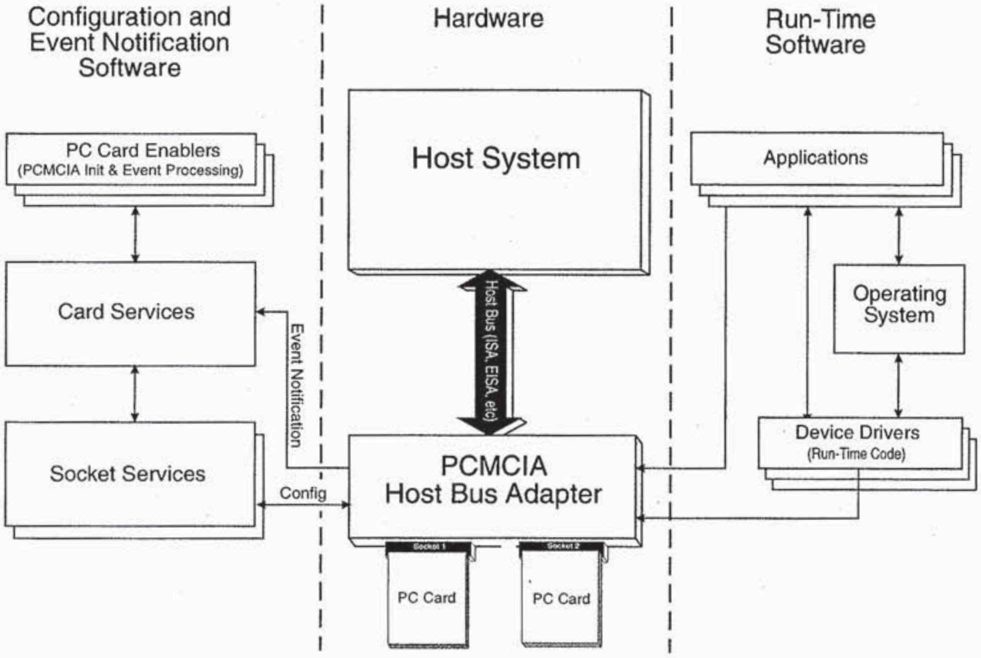

Figure 19-1. Relationship of Socket Services to the Rest of the System.

Chapter 19: Socket Services

Socket services provides a set of functions that can be called by client drivers (typically card services), eliminating the need for special knowledge of the HBA hardware programming interface. These functions can be compared to the BIOS routines that are used in the PC environment. In practice, most client drivers seldom, if ever, directly access socket service functions, because client drivers typically access the HBA via card services. Card services, makes calls to socket services at the client drivers request. In fact, card services blocks access attempts to socket services that are made by client drivers.

Installation and Initialization

Socket services can be contained in ROM, can be loaded into system memory via an installable device driver, or can be incorporated as extensions to the operating system. In the PC environment socket services are typically installed via a device driver and must be loaded into the system before card services and other client drivers (i.e. any software that requires socket services). Without socket services being present card services and PC Card client drivers will not install.

The method used to install socket services and the protocol used to call the functions is platform dependent. The PCMCIA standard currently defines the socket services function call interface only for the Intel x86 platform. Refer to the section entitled, "Socket Services Calling Convention" later in this chapter.

Socket Services Functions

As discussed in the chapter entitled, "The Host Bus Adapter", the HBA must be programmed to allow system access to the PC Card and to manage a variety of HBA functions including:

- Specifying the socket interface type (memory or I/O).
- Programming memory address windows.
- Programming I/O address windows.
- Steering each PC Card's IREQ# signal to the selected system interrupt line.
- Steering the HBA's status change interrupt to the selected system interrupt line.
- Controlling socket power switching.
- Enabling power conservation features.
- Controlling EDC generators.

Socket services controls these functions through a defined set of function calls, each related to objects managed by the adapter. Table 19-1 lists the functions according to the object-based grouping defined below:

- **Adapter Functions** — Those functions that affect all sockets that are controlled by the HBA (i.e., setting Vcc to 3.3 volts for one socket causes all other sockets to also receive 3.3 volts). Adapter functions also pertain to items such as the single status change interrupt, which reports status changes for all sockets.
- **Socket Functions** — Those functions controlled individually at the socket level (i.e., setting Vcc to 3.3 volts for a given socket affects only that socket).
- **Window Functions** — Those functions that control the memory and I/O address windows.
- **Error Detection and Correction (EDC) Functions** — Those functions used to interact with the EDC generators.

Table 19-1. Socket Services Functions

ADAPTER Functions	**WINDOW Functions**
AcknowledgeInterrupt	GetPage
GetAccessOffsets	GetWindow
GetAdapter	InquireWindow
GetAdapterCount	SetPage
GetSetPriorHandler	SetWindow
GetSetSSAddr	
GetSSInfo	**EDC Functions**
GetVendorInfo	GetEDC
InquireAdapter	InquireEDC
SetAdapter	PauseEDC
VendorSpecific	ReadEDC
	ResumeEDC
SOCKET Functions	SetEDC
GetSocket	StartEDC
GetStatus	StopEDC
InquireSocket	
ResetSocket	
SetSocket	

Note that three new socket service functions were added to the PC Card 95 Standard. These functions support CardBus bridge implementations and are not included in this book. For information regarding CardBus, see MindShare's *CardBus System Architecture* book, published by Addison-Wesley.

Socket services has also been designed to permit ease of use. Within each functional group shown in table 19-1, there is are "inquire," "get," and "set" functions, defined below:

- **Inquire functions** — used to report the capabilities of each object defined.
- **Get functions** — used to report the current parameter settings associated with the object.
- **Set functions** — used to set the parameters associated with the object.

The "get" and "set" functions for a specific item have the same basic data structure format, allowing easy modification of parameters. For example, if some parameter within the adapter must be modified, the GetAdapter function can be called to obtain the current adapter settings. This adapter setting information can be written back to the adapter registers using the SetAdapter function once the specific parameter has been changed. This technique permits easy read/modify/write operations to modify individual parameters without having to build the entire data structure that must be passed to socket services when the function is called.

Socket Services Calling Convention

The method used for calling functions depends on the specific platform. Currently, the PCMCIA specification details the programming interface or socket services binding, for Intel x86-compatible systems. The binding specifies use of software interrupt 1Ah to call socket service functions (real mode). This interrupt is typically used by the real-time clock BIOS functions. Therefore socket services shares entry 1Ah in the interrupt table with the real-time clock.

When in protected mode the method of calling socket service functions is operating system specific.

When socket services installs it hooks interrupt 1Ah. This is done by reading and saving the current value of entry 1Ah within the interrupt table (the real time clock BIOS entry pointer) and replacing it with an entry point for its own functions. Socket services functions can then be called using the INT 1Ah instruction. The function numbers are defined in table 19-2 and the general register usage is defined as follows. The exact register content defined for each function can be found in the PCMCIA specification:

Entry:	[AH]	Function number desired in hex
	[AL]	HBA number
	[BH]	Window number
	[BL]	Page or Socket (depending on function)
	[CX]	Counts
	[DX]	Attributes
	[DS]:[(E)SI]	Reserved in ROM BIOS Int 1Ah interface
	[ES]:[(E)DI]	Pointer to socket services client buffer
	[DI]	Offset in 4 KB units
Exit:	[CF]	Status (carry set = error, reset = success)
	[AH]	Success or failure return code depending on Carry Flag value.

If the value in the AH register does not match one of the socket services functions, socket services will pass the call on to the Real Time Clock function, whose entry point was saved during initialization of socket services.

Note that the last function number within socket services is for card services (function 0AFh). Card services also installs into entry 1Ah in the interrupt table and therefore will be called before socket services. Card service functions are called using the 0AFh value in the AH register, allowing definition of the call as a CS function. CS then checks the AL register to identify which CS function is being called. If, however, a socket services function is called, then the AH register contains a value other than 0AFh and CS will not pass the function to socket services. This prevents client drivers from accessing socket services directly and changing HBA settings without CS being notified. See the chapter entitled "PC Card Configuration: Card Services and Client Drivers" for additional information.

Upon exit from a socket services routine, a return (or completion) code is placed in the AH register. The state of the carry flag determines whether the socket service function incurred an error or executed successfully. Table 19-3 lists the return codes.

Table 19-2. Socket Services Function Code Listing

SS Functions Arranged Alphabetically	
Function	**Code**
ACCESS_OFFSETS	0A1h
ACK_INTERRUPT	9Eh
CARD_SERVICES	0AFh
GET_ADAPTER	85h
GET_ADP_CNT	80h
GET_EDC	96h
GET_PAGE	8Ah
GET_SOCKET	8Dh
GET_SS_INFO	83h
GET_STATUS	8Fh
GET_VENDOR_INFO	9Dh
GET_WINDOW	88h
INQ_ADAPTER	84h
INQ_EDC	95h
INQ_SOCKET	8Ch
INQ_WINDOW	87h
PAUSE_EDC	99h
PRIOR_HANDLER	9Fh
READ_EDC	9Ch
Reserved	81h and 82h
Reserved	91h thru 94h
Reserved for expansion	A2h thru ADh
RESET_SOCKET	90h
RESUME_EDC	9Ah
SET_ADAPTER	86h
SET_EDC	97h
SET_PAGE	8Bh
SET_SOCKET	8Eh
SET_WINDOW	89h
SS_ADDR	0A0h
START_EDC	98h
STOP_EDC	9Bh
VEND_SPECIFIC	0AEh

SS Functions Arranged Numerically	
Code	**Function**
80h	GET_ADP_CNT
81h and 82h	Reserved
83h	GET_SS_INFO
84h	INQ_ADAPTER
85h	GET_ADAPTER
86h	SET_ADAPTER
87h	INQ_WINDOW
88h	GET_WINDOW
89h	SET_WINDOW
8Ah	GET_PAGE
8Bh	SET_PAGE
8Ch	INQ_SOCKET
8Dh	GET_SOCKET
8Eh	SET_SOCKET
8Fh	GET_STATUS
90h	RESET_SOCKET
91h thru 94h	Reserved
95h	INQ_EDC
96h	GET_EDC
97h	SET_EDC
98h	START_EDC
99h	PAUSE_EDC
9Ah	RESUME_EDC
9Bh	STOP_EDC
9Ch	READ_EDC
9Dh	GET_VENDOR_INFO
9Eh	ACK_INTERRUPT
9Fh	PRIOR_HANDLER
0A0h	SS_ADDR
0A1h	ACCESS_OFFSETS
A2h thru ADh	Reserved for expansion
0AEh	VEND_SPECIFIC
0AFh	CARD_SERVICES

Table 19-3. Socket Services Return Codes

Return Codes listed alphabetically	
Name of Return Code	Code
BAD_ADAPTER	01h
BAD_ATTRIBUTE	02h
BAD_BASE	03h
BAD_EDC	04h
BAD_IRQ	06h
BAD_MODE	16h
BAD_OFFSET	07h
BAD_PAGE	08h
BAD_SERVICE	15h
BAD_SIZE	0Ah
BAD_SOCKET	0Bh
BAD_SPEED	17h
BAD_TYPE	0Dh
BAD_VCC	0Eh
BAD_VPP	0Fh
BAD_WINDOW	11h
BUSY	18h
NO_CARD	14h
READ_FAILURE	09h
reserved	05h
reserved	0Ch
reserved	10h
reserved	13h
reserved	19h-FFh
Success	00h
WRITE_FAILURE	12h

Return Codes listed numerically	
Code	Name of Return Code
00h	Success
01h	BAD_ADAPTER
02h	BAD_ATTRIBUTE
03h	BAD_BASE
04h	BAD_EDC
05h	reserved
06h	BAD_IRQ
07h	BAD_OFFSET
08h	BAD_PAGE
09h	READ_FAILURE
0Ah	BAD_SIZE
0Bh	BAD_SOCKET
0Ch	reserved
0Dh	BAD_TYPE
0Eh	BAD_VCC
0Fh	BAD_VPP
10h	reserved
11h	BAD_WINDOW
12h	WRITE_FAILURE
13h	reserved
14h	NO_CARD
15h	BAD_SERVICE
16h	BAD_MODE
17h	BAD_SPEED
18h	BUSY
19h-FFh	reserved

Adapter Functions

The adapter functions can be categorized into four classes:

- Functions used to identify the number of adapters within the system and to assign socket services to a specific adapter or adapters. Note that in some cases multiple adapters having different hardware interfaces may be installed in the system. This would require multiple versions of socket services be installed to handle the various adapters.
- Functions that control adapter parameters via the inquire, get and set functions.
- A function used to support status change interrupt processing.
- Low-level access and protected-mode support functions.

Verifying SS is installed (GetAdapterCount)

The Get Adapter Count (GetAdapterCount) function is used by the socket service client (typically Card Services) to determine if socket services is installed and to determine the number of HBAs in the system. This function is typically the first function called and returns the following information to the client.

- the number of adapters that are detected by socket services
- the ASCII string "SS" that verifies that socket services is installed.

Once the client detects that one or more adapters are installed, socket services must be assigned to a given adapter or adapters.

Getting Information from Socket Services (GetSSInfo)

A socket service client calls Get Socket Services Information (GetSSInfo) to determine among other things the number of adapters discovered and controlled by a given set of socket services. When making the GetSSInfo call, the client passes a logical HBA number to socket services as an input. This logical number will be used by the client when it wants to access the HBA in the future. Socket services must remember the logical HBA number and use it to identify accesses to an HBA. Socket services will assign the logical HBA number to the first HBA that it discovers. If socket services discovers more than one HBA, it will assign the next logical number to the second HBA it discovers, etc. Socket services returns the total number of adapters that it has discovered, telling the client the number of adapters this particular set of socket services controls, and therefore the range of logical adapters that it will respond to in the future.

Note that the first adapter detected by the first set of socket services installed is assigned as adapter "0". The client will continue making GetSSInfo calls until all HBAs have been located. This means that one GetSSInfo call will be make by the client for each set of socket services installed. Normally only one set of socket services will be installed.

The following information is returned by the GetSSInfo function:

- Compliance level of adapter. Returned as BCD (Binary Coded Decimal) value. (i.e. 0500h = PC Card Standard, February 1995).
- Number of adapters supported or found by this set of socket services. If socket services recognizes more than one adapter in the system, it returns the total number that it finds and therefore the number it can control.
- First adapter number supported. Note that the first socket services installed always controls adapter zero. The adapter numbers are assigned sequentially starting with zero.

The GetSSInfo function must be run once for each set of socket services installed, thereby assigning logical adapter numbers to all adapters controlled by a particular copy of socket services.

When Two or More Socket Services Are Needed (GetSetPriorHandle)

Some users may want to add more PCMCIA sockets to their system, resulting in two or more different HBA implementations. For example, consider a notebook system with two sockets. When the system is installed in a docking station, more sockets can be added via an additional HBA inserted in an expansion card slot. The additional HBA may have a different hardware interface, requiring its own set of socket services.

PCMCIA can accommodate multiple sets of socket services to support a variety of different HBA implementations. During the initialization process, a socket services client (the SS initialization routine) detects existing HBAs and identifies those that it is compatible with, using the GetAdapterCount and GetSSInfo (as discussed earlier). When installed, additional socket services will also initialize and attempt to identify HBAs that they are compatible with.

When an additional copy of socket services is installed, the client must determine which adapter numbers have already been assigned by previous copies of socket services. The new socket services initialization code can then call the GetSSInfo, to ascertain the first adapter that this new socket services will control.

When a socket services client (card services) makes a call, it specifies a target adapter number or a target socket residing within a particular adapter. The socket services copy receiving the call will be the last installed. If the target adapter or socket is not controlled by this set of socket services, it must pass the call to the next socket services in the chain. This means that some method of linking copies of socket services must be employed. The exact method used to link all copies of socket services together depends on the implementation used by a given architecture.

Some architectures may use the socket services function GetSetPriorHandle to link together multiple copies of socket services. The GetSetPriorHandle function retrieves the handle (entry point address) at which the previous socket services resides. In this way, a linked list of entry points can be maintained such that each socket services passes the call to the next until the target adapter is located. The socket services chain can also be modified (set), allowing a new socket services to supersede or replace an existing copy.

The Intel X86 architecture uses a software interrupt, 1Ah, to call socket services. When the first set of socket services installs, it first reads and saves the existing value in entry 1Ah of the interrupt table and then replaces it with its entry point. If another socket services installs, it also uses entry 1A, by reading and saving the previous pointer (belonging to the current socket services) and replacing it with its own pointer. In this way, each subsequent socket services that installs obtains the pointer to the previous socket services, creating a linked list. Calls to a particular adapter will then be passed from one socket services to the next until the target adapter is located.

Controlling HBA Parameters (InquireAdapter, GetAdapter, SetAdapter)

Before configuring the HBA, the programmer must first determine a specific HBA's capabilities using the InquireAdapter function. Once its capabilities are determined, the HBA configuration parameters can be set using the SetAdapter function. If necessary, the client can check the current adapter settings using the GetAdapter function.

InquireAdapter Function. This function requires the following input parameters be specified: the target adapter number and the location of a memory buffer. The function returns information to the processor's registers and to the specified memory buffer. Parameters returned to the processor's registers include:

- Number of sockets
- Number of address windows
- Number of EDC generators

Parameters returned in memory buffer provide additional information regarding the capabilities of the adapter. The memory buffer format is defined by the socket services specification and can be categorized into two separate parts as shown in table 19-4.

Table 19-4. Adapter Information Structure Definition

Adapter Information Structure	
Adapter Characteristics Structure	**Indicators** — If indicator bit is set, indicators for write protect, battery status, busy status and XIP (Execute-in-place) status are shared for all sockets on the adapter. If reset, indicators exist for individual sockets.
	Power Level — If power level bit is set, the adapter applies the same power level to all sockets. When a SetSocket function is used to set the power for a specific socket, that setting is reflected at all sockets. If power level bit is reset, the adapter can apply power to sockets individually in response to the SetSocket function.
	Data Bus Width — When data bus width bit is set, all adapter address windows use the same data width. If data bus width bit is reset, data width can be assigned to individual windows within the adapter.
	Status Interrupts (High Level) — Bit map of system interrupts to which status interrupts can be steered using an active high state. **Status Interrupts (Low Level)** — Bit map of system interrupts to which status interrupts can be steered using an active low state.
Power Entry Structure	**Number of Power Entries**
	Power Entries — Each entry specifies a voltage level supported and the socket pins (Vcc, Vpp1 and Vpp2) to which the voltage level applies. The voltage level is specified as a DC voltage in tenth of a volt increments. Flag bits are set to indicate the voltage is valid for the specified supply.

GetAdapter Function. The GetAdapter function returns the current status of the HBA settings. When the GetAdapter function is called, the socket services client must pass the physical adapter number to socket services. Adapter parameter states returned by this function include:

- **Powerdown state** — If the bit is set, the adapter is in power conservation state and the SetAdapter function should be used to restore full power before using the adapter. If the bit is reset, full power is applied and the adapter is fully functional.
- **Maintain state** — If this bit is set, configuration information is retained by the adapter hardware during power conservation mode. If reset, the client is responsible for maintaining adapter configuration information during power conservation.
- **Status Change Interrupt Steering** — Returns the system interrupt line, to which status change interrupts are directed.
- **Status Change Interrupt Level** — If set, the status change interrupt is active high. If reset, the interrupt is active low.
- **Status Change Interrupt Enable/Disable State** — The status change interrupt is enabled when set and disabled when reset.

SetAdapter Function. HBA parameters can be set using the SetAdapter function. The exact same parameter mapping is used for the SetAdapter function as for the GetAdapter function. This allows for easy read-modify-write operations when a specific parameter must be changed.

For example, to place the adapter into power conservation mode, the GetAdapter function can be called and the powerdown bit can be toggled. Next the SetAdapter call can be made, causing the powerdown bit to be set within the adapter.

Vendor Functions (GetVendorInfo, VendorSpecific)

The GetVendorInfo function returns information about the vendor that implemented the socket services for a particular adapter in the system. Input parameters to socket services for this call include:

- HBA number
- Type of vendor information requested — a code type of zero indicates that the programmer is requesting the vendor information as an ASCIIZ string (only code currently defined).

- Pointer to the buffer where the ASCIIZ string is to be returned. The buffer format is specified in the socket services standard.

The function returns the ASCIIZ string to the buffer specified, indicating the version number of this particular release of socket services. The vendor's first release of socket services must use a version number of 0100h (release 1.00).

The optional VendorSpecific function is left up to the vendor to implement. The adapter number is specified as an input parameter to socket services when the call is made. The functions supported and the function identification numbers are defined by the vendor to support capabilities beyond the scope of the specification.

Indirect Access to PC Card Memory (GetAccessOffsets)

Some HBAs may access memory cards via I/O registers rather than via memory-mapped address ranges. This eliminates memory address conflicts that might otherwise occur when mapping a PC Card into the system memory address space. These HBAs define a command set that is used when accessing the cards. The client driver uses the GetAccessOffsets function to locate the code that performs these commands. These memory client drivers are HBA specific.

Determining What Card Caused a Status Change Interrupt (AcknowledgeInterrupt)

When a status change event occurs at one of the HBAs sockets, an interrupt request is generated by the HBA. The socket services client (typically card services) is notified of the event via a system interrupt. When the client receives the interrupt, it has no knowledge of which socket encountered the status change event or what the specific event was. The client must determine which socket has experienced a status change event by calling the AcknowledgeInterrupt function. Once the socket (or sockets) that has experienced a status change has been determined, then the GetStatus function is called to determine which event caused the interrupt.

The AcknowledgeInterrupt function must be called once for each HBA in the system. The client supplies the HBA number to socket services when the AcknowledgeInterrupt function is called and socket services returns a bit map of the sockets within the HBA that have experienced a status change. When obtaining status information from the HBA, socket services also prepares the adapter to generate another status change interrupt when one occurs.

In the interrupt service routine, the sockets that have experienced a status change are determined using the AcknowledgeInterrupt function. After the interrupt service routine completes, the client then calls the GetStatus function, specifying the socket that experienced the status change. Most HBAs preserve the state of the status change so that the status change event that caused the interrupt can be determined using the GetStatus function. If the HBA does not preserve this state information, then socket services must.

Note that the AcknowledgeInterrupt function is called by the status change interrupt service routine. Socket services must not re-enable interrupts while processing a status change interrupt service routine. This could cause nesting of status change interrupts to itself, a situation that socket services is unprepared to manage.

Socket Functions

Socket functions deal with parameters that can be controlled on a socket-by-socket basis. These calls require that a particular socket number be specified, whereas adapter functions require an HBA number. The following sections discuss each function in the socket group.

Controlling Individual Sockets (InquireSocket, SetSocket, GetSocket)

Functions used to control a socket are similar to the adapter functions that are used to control HBA functionality. The adapter functions control parameters that apply to all sockets supported by a specific HBA, whereas the socket functions control parameters that apply individually to each of the HBA's sockets.

InquireSocket. This function requires that a target socket number be specified along with the address of a memory buffer. This function returns the following information:

- Events that can trigger a status change interrupt. These events can be a combination of the following items:
    - PC Card write-protect (WP) signal.
    - A signal (from card interlock logic) indicating the state of a card lock mechanism.
    - A signal (from the card interlock logic) indicating a request to eject a PC Card from the socket.
    - A signal indicating a request to insert a card into the socket.
    - PC Card BVD1 signal indicating that the card's battery is completely discharged.
    - PC Card BVD2 signal indicating the card's battery is weak.
    - PC Card READY signal, indicating a change in the card's ready state.
    - PC Card Detect Signals.
- Bit map of status change events that are reported via the GetStatus function. This bit map includes all the items that can generate a status change interrupt (listed above), plus other events that do not generate an interrupt but whose status is returned to the socket services client driver by the GetStatus function.
- Bit map of items for which there is a control or an indicator supported at the socket level. Indicators are items such as LED indicators that the HBA provides which shows the status of given events. These items may include:
    - Indicator for WP signal.
    - Indicator for state of card lock mechanism.
    - Control for motor to eject card from socket.
    - Control for motor to insert card into socket.
    - Control to establish a card lock.
    - Indicator for BVD1 and BVD2 state.
    - Indicator showing when card is in use.
    - Indicator for execute-in-place (XIP) application is progress.
- The Socket Information Structure is returned to a memory buffer supplied by the socket services client. The memory buffer format is defined by the socket services as shown in table 19-5.

Table 19-5. Socket Information Structure Definition

Socket Information Structure	
Socket Characteristics Structure	**Socket Interface Type**—If interface bit is set, the socket is a memory only interface. If reset, the socket interface is memory or I/O.
	PC Card Interrupts (High Level)—Bit map of system interrupts to which PC Card interrupts can be steered using an active high state.
	PC Card Interrupts (Low Level)—Bit map of system interrupts to which PC Card interrupts can be steered using an active low state.

GetSocket Function. The GetSocket function returns the current status of the HBA socket settings. When the GetSocket function is called, the socket services client must pass the adapter and socket number to socket services. The parameter's returned by this function are:

- **Status Change Mask** — Returns the current setting of the events that cause a status change interrupt from the socket.
- **Vcc Level** — Returns the current supply voltage applied to the socket on the Vcc pin.
- **Vpp Levels** — Returns the current supply voltage applied to the socket on the Vpp pins. Separate values are returned for Vpp1 and Vpp2.
- **State Change**. Returns the latched values of the status change events that have occurred at the socket.
- **Socket Controls and Indicators** — Returns a bit map of socket controls and indicators that are in use. Bits that are set indicate the control or indicator is activated.
- **IREQ Routing** — Returns the system interrupt line to which the card's IREQ# signal is directed. Optionally, an additional bit can specify whether the IREQ# signal should be inverted or not, and another optional bit can enable or disable interrupt routing.
- **Interface Type** — Returns the interface setting. Only one of the following selections can be set; a "Memory-Only" interface and a "Memory or I/O" interface

SetSocket Function. Socket parameters are set using the SetSocket function. The exact same parameter mapping is used for the SetSocket function as for the GetSocket function. That is, the data structure format for the SetSocket function mirrors the definition of the GetSocket function's data structure format listed earlier. This simplifies read-modify-write operations when a specific parameter must be changed.

Determining the Current Status of the Socket and PC Card (GetStatus)

This function is intended to be called by the socket services client to determine what event(s) have caused a status change interrupt. This call should not be made during hardware interrupt processing of the status change interrupt, but rather after the interrupt has been processed and the socket(s) experiencing a status change event has been identified. The socket services client can then call the GetStatus function to determine which event caused the status change interrupt.

The information returned reflects the current state of the parameters set within the HBA:

- Returns the current state of the events that can cause a status change interrupt (as defined by the InquireSocket function) and the current state of the socket control and indicators (also defined in the InquireSocket function).
- Returns the current bit map of parameters or events that cause a status change interrupt. These events are defined in the GetSocket function's status change mask.
- Returns a bit map of Socket control and indicator bits supported by HBA.
- Returns the current settings of the IREQ Routing parameters.
- Returns the current Interface Type setting.

Resetting the Socket Under Software Control (ResetSocket)

This function provides a software reset to the PC Card and resets the socket hardware interface to its power-on default condition as follows:

- Socket interface is reset to memory only.
- IREQ routing is disabled.
- All socket supplies (Vcc, Vpp1, and Vpp2) are set to 5vdc.
- All address windows are disabled.
- All EDC Generators are disabled.

Window Functions

Window functions, like the adapter and socket functions, include the inquire, get, and set functions, as well as page functions that allow memory windows to be divided into multiple pages. Memory locations within a window can be segmented into 16KB pages.

Controlling Windows (InquireWindow, GetWindow and Set-Window)

The window functions are designed for flexibility, such that they can be used for common memory, attribute memory, or I/O. Despite this flexibility provided by socket services, a given hardware implementation of the HBA may be more restrictive. The capabilities for each window is obtained when the socket services client calls the InquireWindow function for each window detected by the InquireAdapter function.

The characteristics of a given window extend far beyond whether they can be used for memory, I/O or both. Many other parameters such as the base address, window size, fastest and slowest devices supported, etc., must be characterized for each window. Once the characteristics of the window is determined then it can be programmed by the socket services client at the request of the PC Card's driver.

InquireWindow Function. When the InquireWindow function is called, the HBA number and window number are passed to socket services, along with a pointer to a memory buffer supplied by the socket services client. Information is returned to the processor's registers and to the specified memory buffer. The total set of information returned to the socket services client includes the following:

- **Window Type** — Returns the characteristics of the window selected with the HBA and window parameters. A single window may be designed to provide support for any or all of the following:
  - A window can be used as a common memory window.
  - A window can be used as an attribute memory window.
  - A window can be used as an I/O window.
  - A window can specify that the WAIT# signal from the PC Card to is used to generate additional wait states during a socket data transfer.

- Note that even though socket services allows a window to be used as both an I/O window and a memory window, this usually is not the case. More typically, hardware designs restrict a given window to either I/O addresses or memory addresses, but not both.

- **Socket Assignment** — Returns a bit map of sockets that a window can be assigned to. Bit zero refers to socket zero and bit N refers to the maximum socket number. The size of this bit map restricts the number of sockets that can be supported by a given HBA. In the x86 environment, socket services has a 16-bit field, permitting a maximum of 16 sockets per HBA.

- **Window Characteristics Structure** — Returns a variety of windows parameters to a memory buffer supplied by the socket services client. Two types of window characteristics structures are defined: one for memory windows and one for I/O windows. As mentioned earlier, a given adapter may be designed to permit a given window to support memory addresses only, I/O addresses only, or both memory or I/O. A window characteristics structure is returned for each window type supported by the target address window.

Table 19-6 lists the parameters defined within a memory window characteristics structure, and table 19-7 lists parameters defined within a I/O window structure. The parameter definition for many of the entries within both structures are identical; however, some important differences exist. Parameters that differ are highlighted in tables 19-6 and 19-7.

Table 19-6. Memory Window Characteristics Structure Definition

Memory Characteristics Structure		
Mem Window Capabilities		Consists of flag bits that specify any of the parameters listed below.
	Base Address	Determines if the base address is programmable (bit is set) or is fixed (bit is reset) in the host's address space. If programmable, the base address must be within the range specified by the FirstByte and LastByte entries, and if fixed, the base address location is specified by the value of the FirstByte entry and the LastByte entry has no meaning.
	Window Size	Determines if the memory window size is programmable (bit is set) or is fixed (bit is reset). If programmable, the size can be any value within the range specified by the Minimum Size and Maximum Size entries. If fixed, the window size is determined by the value of the Minimum Size entry and the Maximum Size entry should be set to the same value and the Minimum Size.
	Window Enable	Determines if the HBA will preserve window state information when the window is disabled (bit is set), or whether software must be responsible for preserving the state information (bit is reset). This means that when the window is re-enabled, it must be reprogrammed by the client if the HBA does not preserve the information.
	8-Bit Data Width	Determines whether the memory window supports 8-bit data transfers to the socket required (8-bit hosts). If set, 8-bit transfers are supported and if reset, they are not supported.
	16-Bit Data Width	Determines whether the memory window supports 16-bit data transfers to the socket required (16-bit hosts). If set, 16-bit transfers are supported and if reset, they are not supported.
	Base Address Alignment	If set, the base address must be programmed to start at an address aligned on the size of the window. If reset, the base address can be programmed to start anywhere within the window's address range, consistent with the "Base Address Alignment" value (defined later).
	Window Size Increments	Determines if windows supporting a programmable size must be sized in "powers of two" increments consistent with the "Window Size Granularity" value defined later (bit is set). If the granularity is 4KB, then the window size can be 4KB, 8KB, 16KB, 32KB, ... up to the maximum size of the window. If bit is reset, window sizes can be any multiple of the "Window Size Granularity" value -- 4KB, 8KB, 12KB, 16KB, 20KB .... up to the maximum window size.
	Window Page Boundaries.	Specifies whether offsets specified to Set Page must be on boundaries equal to the size of the window (bit is set), or if page offset can be set without relation to the window size (bit is reset).

Table 19-6. Memory Window Characteristics Structure Definition(Continued)

Memory Characteristics Structure		
Mem Window Capabilities		Consists of flag bits that specify any of the parameters listed below.
	Window Page Support	Determines if window hardware supports dividing a window into multiple pages (bit set), or does not support window paging (bit is reset).
	Page Sharing	If set, the window paging hardware is shared with another window and care must be taken to ensure that no conflicts arise due to resource sharing. If reset, paging hardware is dedicated to the window.
	Page Enable.	If set, the HBA preserves the paging characteristics when the page is disabled. If reset, the software must preserve the settings and reprogram the paging hardware when the page is enabled again.
	Write-Protect.	Determines if the window can be write-protected (bit is set) or not (bit is reset).
FirstByte		The first byte in the host system's addressable memory space that can be programmed for the window's base address. Note that if the base address register is not programmable, the value is the fixed address for the window's base address.
LastByte		The last byte in the host system's memory address space that the window can be programmed to.
Minimum Window Size		Defines minimum size that the window can be programmed to.
Maximum Window Size		Maximum size that window can be programmed to.
Window Size Granularity		Window size granularity determines the minimum size that a window can be programmed to based on the hardware implementation. For example, if lower address lines A11:A0 go directly to the PC Card socket, then the window size that can be programmed is based on 4KB intervals.
Base Address Alignment		Specifies the base address alignment value for the window.
Window Offset Alignment		Specifies the alignment boundaries that the window offset can be programmed to for remapping the system address to PCMCIA memory.
Selected Access Speed		Specifies the slowest access speed supported for devices accessed through this window.
Fastest Access Speed		Specifies the fastest access speed supported for devices accessed through this window.

Table 19-7. I/O Window Information Structure Definition

I/O Window Information Structure		
I/O Window Capabilities		Consists of flag bits that specify any combination of the parameters below.
	Base Address	Determines if the base address is programmable (bit is set) or is fixed (bit is reset) in the host's address space. If programmable, the base address must be within the range specified by the FirstByte and LastByte entries, and if fixed, the base address location is specified by the value of the FirstByte entry and the LastByte entry has no meaning.
	Window Size	Determines if the I/O window size is programmable (bit is set) or is fixed (bit is reset). If programmable, the size can be any value within the range specified by the Minimum Size and Maximum Size entries. If fixed, the window size is determined by the value of the Minimum Size entry and the Maximum Size entry should be set to the same value and the Minimum Size.
	Window Enable	Determines if the HBA will preserve window state information when the window is disabled (bit is set), or whether the client must be responsible for preserving the state information (bit is reset). This means that the window must be reprogrammed by the client when reenabled if the HBA does not preserve the information.
	8-Bit Data Width	Determines whether the I/O window supports 8-bit data transfers to the socket required by 8-bit hosts. If set, 8-bit transfers are supported and if reset, they are not supported.
	16-Bit Data Width	Determines whether the I/O window supports 16-bit data transfers to the socket (16-bit hosts). If set, 16-bit transfers are supported and if reset, they are not supported (8-bit hosts).
	Base Address Alignment	If set, the base address must be programmed to start at address locations equal to the size of the window. If reset, the base address can be programmed to start anywhere within the window's address range, consistent with the "Base Address Alignment" value defined later.
	Window Size Increments	Determines if windows supporting a programmable size must sized in "powers of two" increments consistent with the "Window Size Granularity", or if the windows size can be any multiple of the "Window Size Granularity" value.
	INPACK Signal Support	Specifies whether the adapter supports the Input Port Acknowledge (INPACK) signal or not. The INPACK signal permits an I/O window to overlap address space mapped elsewhere in the system.
	EISA Slot Specific I/O Address Support	Indicates support for EISA compatible addressing. In this case, the HBA in this case should respond to I/O addresses consistent with the slot specific addressing protocol required by EISA systems. See the MindShare publication, "EISA System Architecture" for details.

Table 19-7. I/O Window Information Structure Definition (Continued)

I/O Window Information Structure	
I/O Window Capabilities	Consists of flag bits that specify any combination of the parameters below.
Ignore EISA-Defined alias (ISA) I/O Accesses.	Determines whether accesses to ISA address alias ranges should be ignored or not when slot-specific EISA I/O addressing is used.
FirstByte	The first byte in the host system's addressable I/O space that can be programmed for the window's base address. Note that if the base address register is not programmable the value is the fixed address for the window's base address.
LastByte	The last byte in the host system's I/O address space that the window can be programmed to.
Minimum Window Size	Defines minimum size that window can be programmed to.
Maximum Window Size	Maximum size that window can be programmed to.
Window Size Granularity	Describes the size interval that the window can be programmed to.
Base Address Alignment	Specifies the base address alignment value.
Number of Address Lines Decoded	Specifies the number of address lines decoded by the window.
EISA Slot Addressing	Specifies the upper nibble (A15:A12) of an x86 I/O address when EISA addressing is supported.
Fastest Access Speed	Specifies the fastest access speed supported for devices accessed through this window.

GetWindow Function. The Get Window function returns the current setting of the window specified by the programmer. The programmer passes the HBA and window numbers to the function. The function returns the following information:

- **Socket to which window is assigned.**
- **Window size.**
- **Current State of window hardware**—Returns the current setting of other window parameters. The value can be a combination of the following:
  - **Memory or I/O mapped**. This bit specifies whether the window is mapped into the host system's memory address space or I/O address space.
  - **Enabled or disabled**. Specifies whether the window is currently enabled or disabled.

  - **Window data width**. Specifies whether the window is programmed for 16-bit data width or 8-bit data width.
  - **Memory window pages used** (memory windows only). This parameter indicates if memory window pages are in use, indicating that this window is subdivided into multiple 16KB pages and that the GetPage and SetPage functions can be used for accessing individual pages within the window.
  - **EISA I/O Mapping used**.
  - **Card access permitted during EISA I/O accesses**. If this bit is set and EISA mapping is used, accesses to standard ISA addresses result in PC Card accesses. If reset, accesses to ISA addresses are ignored.
- **Access Speed.** Indicates the current access speed programmed into the memory window.
- **Window's Base Address.**

SetWindow Function. This function uses the same mapping as the GetWindow function. The definition of the parameters are the same, allowing the GetWindow function to be called to obtain the current window settings. Parameters requiring modification can then be changed from the current settings and the SetWindow function called to update the window's settings.

EDC Functions

Error Detection/Correction Generators are optional for PCMCIA HBAs. These functions are designed to enable and control EDC generators implemented by HBAs. However, card services provides no support for EDC functions. Furthermore, to the author's knowledge no current HBA designs employ EDC generators. Based on these issues discussion of the socket services EDC functions has been omitted from this book.

Maximum Number of Sockets Per HBA

The maximum number of sockets that a single adapter can support under control of socket services is limited by the **InquireWindow** function. A bit-map of assignable sockets is returned by this function. The size of this bit-mapped socket selection field defines the maximum number of sockets supported by each adapter. The field size is not defined by PCMCIA and depends on the socket services implementation. The Intel x86 socket services definition defines a 16-bit socket selection field, permitting 16 sockets per HBA.

Maximum Number of HBAs Supported by Socket Services

The maximum number of adapters supported by socket services depends on several factors, including:

- Limitations associated with the implementation of socket services for a given platform. For example, the field size used to specify a target adapter càn vary with a particular implementation. Note that the x86 implementation uses an 8-bit field, permitting 256 adapters to be specified (clearly not a meaningful limitation).
- Constraints related to available space when implementing socket services in ROM.
- Constraints related to available memory space required by multiple sets of socket services required to support numerous adapters.

Chapter 20

The Previous Chapter

The previous chapter discussed the role of socket services. It also described the initialization of socket services and explained the basic purpose of the functions commonly supported in the PC environment.

This Chapter

This chapter focuses on the role of card services in the PCMCIA environment. It also reviews each of the functions defined by the PC Card specification that apply to 16-bit PC Cards, along with related return codes. The call-back mechanism is also described and the event and call-back codes are defined.

The Next Chapter

The next chapter discusses the three basic types of enablers: point enablers, device specific enablers, and super enablers. The chapter also discusses the specific jobs performed by several different device specific enablers including SRAM enablers, FLASH enablers, I/O device enablers, and ATA enablers.

Overview

Each PC Card must have a client driver that recognizes it, reads the CIS to determine its resource requirements, programs the host bus adapter (HBA) and configures the PC Card. As illustrated in figure 20-1, PC Card client drivers interface directly to Card Services. Card services simplifies the job of configuring a PC Card and monitoring status change events.

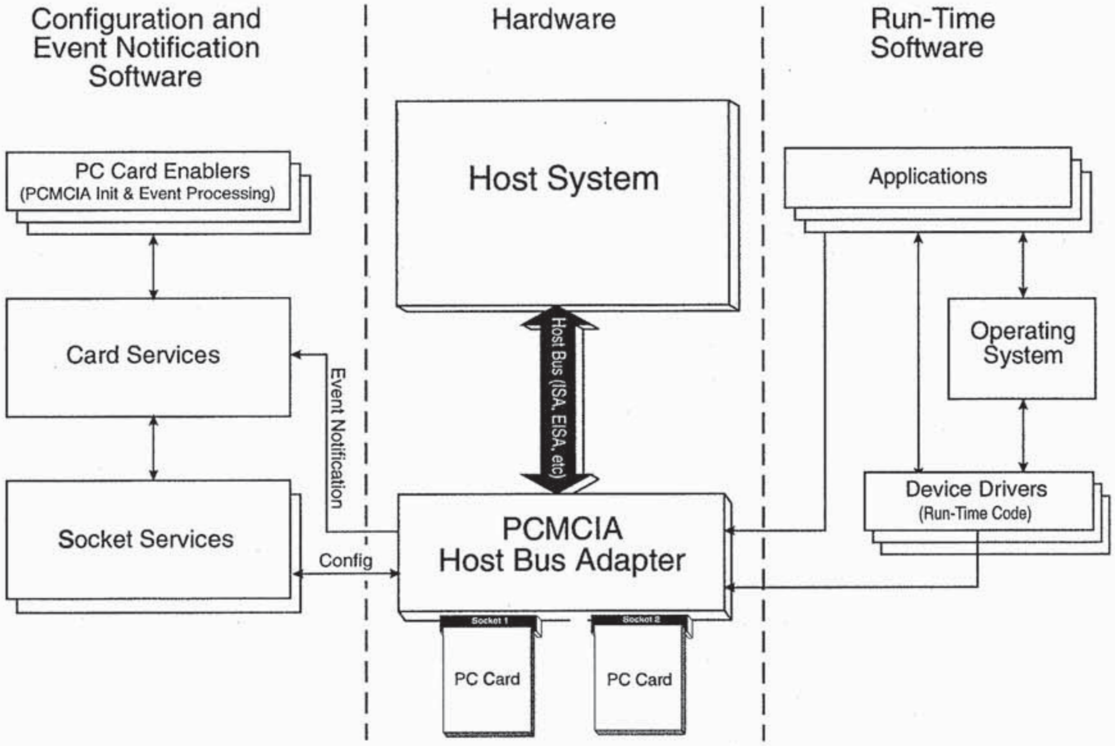

Figure 20-1. PCMCIA Software Flow

Configuring a PC Card may take place when the system powers up (if the PC Card is already installed in a socket), or when a PC Card is inserted into a socket (after the system is powered up and fully operational). In either case, the PC Card must be detected and configured by an enabler. Without an enabler, a PC Card would never be recognized by the system. However, once a PC Card is recognized and configured by the enabler, it then responds like any other device residing on the host bus.

Enablers that use card services are called card services client drivers. The term client driver is used because card services and the enablers perform their functions based on the client/server model. Card services exists to serve the needs of its clients (i.e. the enablers) as they attempt to configure and access their PC Cards. Two basic types of client drivers exist:

- Dedicated client drivers — designed for a particular PC Card.
- Generic or super client drivers — designed for a wide range of PC Cards.

Dedicated client drivers are typically supplied by the PC Card manufacturer to increase the probability that its card will be recognized and configured correctly in the absence of a generic driver. Dedicated client drivers may also manage functions that are unique to a given manufacturer's implementation.

Generic client drivers are frequently designed to handle PC Cards of a particular functional type. For example, the system manufacturer may include generic drivers for card types such as SRAM, flash ROM, Modems, and ATA drives. Ideally, a single super client driver could detect and configure all PC Cards regardless of type.

Enabling PC Cards Before Card Services

Prior to the release of card services, the enabler was burdened with recognizing when a card was inserted into a socket, reading its CIS, programming the HBA and configuring the PC Card so that it responded to a given system address range. The PC Card's enabler also had to continually monitor the socket to detect if the PC Card was removed. If removed, the enabler would deallocate the system resources the card was using by clearing registers in the HBA. In this way, the HBA would no longer respond to addresses previously assigned to the PC Card.

To configure a card, an enabler also had to determine what address ranges were available within the system (not in use by other devices) for allocation to its card. This was an almost impossible job for enablers since they had no knowledge of the other devices incorporated into the system or of other installed PC Cards. Assumptions had to be made by the programmer based on what resources were likely available so that contention with other devices was (hopefully) avoided.

It is also possible that other software applications or utility programs written by other programmers may want to share access to a given PC Card. These various programs will not be aware of each other and, as a result, conflicts may occur.

In summary, PC Card enablers that are compliant with PCMCIA releases prior to 2.0 each act independently, unaware of the existence of each other. Furthermore, they have no knowledge of the resources available within the system that could safely be allocated to their associated PC Card.

The Role of Card Services

Card services provides a central resource available to all client drivers. Specifically, card services is a collection of functions designed for use by programmers writing client drivers for PC Cards. These functions provide a software interface that permits the programmer to simplify code and helps to reduce conflicts with other client drivers and system resources. Card services is divided into five functional groupings by the PCMCIA specification:

- **Client Services**—Provides a registration facility that permits client drivers to register and be notified by card services when specific socket events occur (such as card insertion or removal).
- **Resource Management**—Allows client drivers to request the use of system resources required by the PC Card they are enabling. If the resources are granted, addition resource management functions can be used to assign these resources by programming the HBA (via socket services) and configuring the PC Card (by writing to the PC Card's configuration registers).
- **Client Utilities**—Provides a set of functions that allow the client driver to perform common jobs with ease. For example, the functions include accessing the PC Card's CIS, thereby simplifying tuple processing code.
- **Bulk Memory Services**—Provides block memory functions to read, write, copy and erase blocks of data within memory cards (without knowledge of the specific memory technology). These functions are passed to the appropriate Memory Technology Driver (MTD) that understands the hardware protocol necessary to erase or write to devices such as flash memory. (See the next chapter "PC Card Enablers" for details regarding memory technology drivers.)
- **Advanced Client Services**—Provides specialized functions that may be needed by some client drivers.

Only one copy of card services is required (and permitted), since it controls access to all sockets (whether associated with a single adapter or multiple adapters). Once a PC Card has been configured, it responds like any other host bus device. As a result, application programs designed to access a particular function need not even be aware of the existence of card and socket services. Card services and socket services are employed by enablers during:

- PC Card initialization and configuration (client driver makes calls).
- PC Card event notification (interrupt driven calls).

- Block transfers to/from memory card (memory client driver makes calls during run-time).

During other times, card and socket services remain in memory, but are not used. The following sections discuss typical uses of the card services functions. The information included in this chapter is not intended for reference purposes. The function descriptions provide only a basic description of the function's purpose. Refer to the PCMCIA Card Services standard for the exact calling parameters, format, field sizes, etc., of each function.

Initialization of Card Services

Card Services is designed as an operating system dependent extension that provides client services for the PC Card environment. Card services may come with the operation system as a built-in extension (e.g. OS/2 and Windows 95). In the MS-DOS environment, card services is typically implemented as an installable device driver.

In the DOS environment card services are called using an INT 1A instruction, requiring that card services "hook" entry 1Ah within the interrupt table. Also card services hooks the hardware interrupt used by the HBA to report status change events. This allows card services to be notified when a status change event occurs at the socket level.

Verifying the Presence of Socket Services

Since card services utilizes socket services to fulfill client driver requests, it must install after socket services installs. Socket services may reside in ROM on the system, or may be installed as a loadable device driver when the operating system loads. If socket services installs as a device driver, card services must be placed in the config.sys so that it installs after socket services.

Before card services installs it must verify that socket services are resident. This is done by calling the GetAdapterCount function within socket services. This function returns the total number of HBAs detected in the system and which returns the ASCII string "SS" verifying that socket services is present. If "SS" is detected, then card services proceeds with its installation.

After card services installs, it blocks access to socket services. If a client driver attempts to call socket services directly, card services will not pass the call on to socket services, but will return failure to the client driver. This prevents a client driver from using socket services to access the adapter hardware directly and perhaps allocating resources or modifying the HBA's programming without the knowledge of card services. Since this would result in card services becoming desynchronized with regard to the actual adapter hardware, attempts to access socket services without going through card services are prevented.

Note that card services does include a function (ReturnSSEntry) that can be called by a client driver that returns the entry point of socket services. This allows a client driver to gain direct access to socket services, but it must not perform any socket service function that causes card services to become desynchronized with the HBA.

Verifying that Card Services Installed

Initialization code used to install card services also includes code that actually calls card services to validate that the installation of card services was successful. This is accomplished by calling the GetCSInfo service, which returns information about this version of card services and the ASCII values "CS" to verify that card services are present. If card services installed correctly, the initialization code can make additional service call to prepare card services for access by a PC Card client driver.

Determining Availability of System Resources

One of the major functions performed by card services is to allocate available system resources to PC Cards. Resource management services are called by a PC Card's client driver in an attempt to acquire the resources (i.e. the I/O address space, memory address space, IRQ line, and DMA channels) that will satisfy the card's configuration requirements. Card services must check the available system resources to verify that the requested resources are not already used by the system.

Since card services is an extension to the operation system, in many operating environments it will have no specific knowledge of the resources that are already being used by other devices installed into the system. As a result, some method must be employed by card services (or by other platform specific software) that can detect free resources that can be allocated and assigned to PC Cards. The exact method used is operating system and hardware platform specific.

In x86 DOS compatible systems, a utility program is typically used to scan the host system in an attempt to detect the presence of devices that use system resources. The utility program builds a table of system resources that are not in use and passes the table to card services. Card services then manages the resources table as resources are requested and released by the client drivers as PC Cards are inserted and removed from sockets. This program is either embedded within card services initialization code or is implemented as a separate installable device driver that executes immediately after card services has installed (e.g. listed in the config.sys file immediately following the card services device driver) and before the PC Card enablers.

Power Management Support

Power management support was added to the PC Card standard (95 release). Card service defines power management (PM) support via power management call-back events. Card services can be designed to detect the presence of a power management facility within the PC platform and register to receive notification of power management events. When card services receives the power management notification, it calls-back all client drivers that registered to receive the PM events.

Card Services Calling Conventions

When a client calls card services, the binding used in a given environment will differ. The PC Card specification specifies a card services programming interface (binding) for x86 real mode (DOS), Intel 80286 Protected Mode (Windows), Intel 80286 Protected Mode (OS/2), and Intel 80386 Flat Address Model (Windows VxD Clients). Each binding specifies the register usage for calling card services functions and the register usage employed when the callback handler is invoked. An example of the binding specified for the Intel X86 Real Mode environment follows. Refer to the card services specification for additional information.

Input:

[AH]	AFh (specifies card services function)
[AL]	Service Desired (service code number)
[DX]	Handle
[DI]:[SI]	Pointer argument [DI]=16-bit segment, [SI]=16-bit offset
[CX]	Argument Length (total length of argument packet)
[ES]:[BX]	Pointer to argument packet (used when additional address space is required to pass parameters and data) [ES]=16-bit segment, [BX]=16-bit offset

Output:

[AX]	Return Code
[CF]	Success when clear, failure when set

Specifying the Service

The AH register must contain a value of AFh to specify that this card is meant for card services. The AL register then specifies the service code number of the service being requested.

Table 20-1 lists all of the services (listed in alphabetical order) defined by the PC Card Standard along with their associated service number. Table 20-2 lists the services and their service number in ascending numerical order. Note that the services in shaded boxes were added by the PC Card 95 release.

The value placed in the AH register permits card services to block access to socket services functions made by enablers. Note that AFh is the last function number within socket services (function 0AFh) and is defined for use by card services.

When card services initializes, it hooks entry 1Ah in the interrupt table. Card services saves the current value of entry 1Ah (pointing to socket services) before installing its own. As a result, card services knows the entry point for socket services. INT 1Ah calls now access card services, which verifies that the call is a card services call by checking for the value AFh in the AH register. If verified, the card services function call specified in the AL register is then processed.

If card services finds a value other than 0AFh, it then checks to determine if the value represents a valid socket services function. If it is a valid socket services function, card services blocks access and returns failure to the calling program. This prevents client drivers from accessing socket services directly and changing HBA settings without card services being notified.

If the value in the AH register is for neither card services nor socket services, then card services passes the call to socket services, knowing it will not recognize the call. Socket services then passes the call to the previous interrupt service routine in the chain. Interrupt table entry 1Ah is used by the real-time clock functions in DOS compatible machines, therefore, card and socket services shares INT 1Ah with the real-time clock functions.

The Handle

A handle may specify the client making the service call or a particular resource that is being targeted by the function. The client handle is returned to the client during the registration process. This handle is used by the client when requesting many services. For example, a memory client may choose to Open a region of memory within a memory card for use with other memory services (i.e. read, write, or erase services). The client must specify its client handle in the DX register as an input and card services returns a memory handle (to identify the region of memory) to the DX register. The client later uses memory handle as an input when calling the read, write, or erase memory services.

Table 20-1. Card Services Listed in Alphabetical Order

Function	Code
AccessConfigReg	36h
AddSocketServices	32h
AdjustResourceInfo	35h
CheckEraseQueue	26h
CloseMemory	00h
CopyMemory	01h
DeregisterClient	02h
DeregisterEraseQueue	25h
GetCardServicesInfo	0Bh
GetClientInfo	03h
GetConfigurationInfo	04h
GetEvenMask	2Eh
GetFirstClient	0Eh
GetFirstPartition	05h
GetFirstRegion	06h
GetFirstTuple	07h
GetFirstWindow	37h
GetMemPage	39h
GetNextClient	2Ah
GetNextPartition	08h
GetNextRegion	09h
GetNextTuple	0Ah
GetNextWindow	38h
GetStatus	0Ch
GetTupleData	0Dh
MapLogSocket	12h
MapLogWindow	13h
MapMemPage	14h
MapPhySocket	15h
MapPhyWindow	16h
ModifyConfiguration	27h
ModifyWindow	17h
OpenMemory	18h
ReadMemory	19h
RegisterClient	10h
RegisterEraseQueue	0Fh
RegisterMTD	1Ah
RegisterTimer	28h
ReleaseConfiguration	1Eh
ReleaseDMA	3Bh
ReleaseExclusive	2Dh
ReleaseIO	1Bh
ReleaseIRQ	1Ch
ReleaseSocketMask	2Fh
ReleaseWindow	1Dh
ReplaceSocket Services	33h
RequestConfiguration	30h
RequestDMA	3Ah
RequestExclusive	2Ch
RequestIO	1Fh
RequestIRQ	20h
RequestSocketMask	22h
RequestWindow	21h
ResetCard	11h
ReturnSSEntry	23h
SetEvenMask	31h
SetRegion	29h
ValidateCIS	2Bh
VendorSpecific	34h
WriteMemory	24h

Table 20-2. Card Services Function Codes Listed in Numerical Order

Code	Function
00h	CloseMemory
01h	CopyMemory
02h	DeregisterClient
03h	GetClientInfo
04h	GetConfigurationInfo
05h	GetFirstPartition
06h	GetFirstRegion
07h	GetFirstTuple
08h	GetNextPartition
09h	GetNextRegion
0Ah	GetNextTuple
0Bh	GetCardServicesInfo
0Ch	GetStatus
0Dh	GetTupleData
0Eh	GetFirstClient
0Fh	RegisterEraseQueue
10h	RegisterClient
11h	ResetFunction
12h	MapLogSocket
13h	MapLogWindow
14h	MapMemPage
15h	MapPhySocket
16h	MapPhyWindow
17h	ModifyWindow
18h	OpenMemory
19h	ReadMemory
1Ah	RegisterMTD
1Bh	ReleaseIO
1Ch	ReleaseIRQ
1Dh	ReleaseWindow
1Eh	ReleaseConfiguration
1Fh	RequestIO
20h	RequestIRQ
21h	RequestWindow
22h	RequestSocketMask
23h	ReturnSSEntry
24h	WriteMemory
25h	DeregisterEraseQueue
26h	CheckEraseQueue
27h	ModifyConfiguration
28h	RegisterTimer
29h	SetRegion
2Ah	GetNextClient
2Bh	ValidateCIS
2Ch	RequestExclusive
2Dh	ReleaseExclusive
2Eh	GetEvenMask
2Fh	ReleaseSocketMask
30h	RequestConfiguration
31h	SetEvenMask
32h	AddSocketServices
33h	ReplaceSocket Services
34h	VendorSpecific
35h	AdjustResourceInfo
36h	AccessConfigReg
37h	GetFirstWindow
38h	GetNextWindow
39h	GetMemPage
3Ah	RequestDMA
3Bh	ReleaseDMA

The Argument Packet

Some services require that the client provide a memory buffer to pass parameters. Functions requiring a large data area for passing parameters use an argument packet. The pointer to the argument packet specifies the start memory location of the buffer, while the argument length specifies the size of the buffer (i.e. length of argument packet). The size and format of the argument packet is typically depends of the individual function.

Not all of the generic arguments just defined are used when calling a given service. Many functions require only a function code, handle and the pointer argument to pass all of the required parameters. Some service require the pointer argument, while other require use of the argument packet.

Return Codes

A variety of codes may be returned by card services into the processor's AX register. The return codes specify the results of the service. Table 20-3 lists and defines each of the return codes in alphabetical order. Table 20-4 lists the return codes in numerical order.

The Pointer Argument

Some services require a read/write buffer to pass input and output information between the client and card services. The pointer argument value placed in the DI and SI registers specifies the location of the memory buffer. These same buffer is used by card services to return data to the client. DI:SI are also used to specify the memory location the call-back buffer.

Table 20-3. Card Services Return Codes Listed in Alphabetical Order

Return Code	Value	Description
BAD_ADAPTER	01h	Specified adapter is invalid
BAD_ARG_LENGTH	1Bh	ArgLength argument is invalid
BAD_ARGS	1Ch	Values in Argument Packet are invalid
BAD_ATTRIBUTE	02h	Value specified for attributes field is invalid
BAD_BASE	03h	Specified base system memory address is invalid
BAD_EDC	04h	Specified EDC generator is invalid
BAD_HANDLE	21h	ClientHandle is invalid
BAD_IRQ	06h	Specified IRQ level is invalid
BAD_OFFSET	07h	Specified PC Card memory array offset is invalid
BAD_PAGE	08h	Specified page is invalid
BAD_SIZE	0Ah	Specified size is invalid
BAD_SOCKET	0Bh	Specified socket is invalid (logical or physical)
BAD_SPEED	17h	Specified speed is unavailable
BAD_TYPE	0Dh	Window or interface type specified is invalid
BAD_VCC	0Eh	Specified Vcc power level index is invalid
BAD_VERSION	22h	Client version is unsupported
BAD_VPP	0Fh	Specified VPP1 or VPP2 power level index is invalid
BAD_WINDOW	11h	Specified window is invalid
BUSY	18h	Unable to process request at this time - retry later
CONFIGURATION_LOCKED	1Dh	A configuration has already been locked
GENERAL_FAILURE	19h	An undefined error has occurred
IN_USE	1Eh	Requested resource is being used by a client
NO_CARD	14h	No PC Card in socket
NO_MORE_ITEMS	1Fh	There are no more of the requested item
OUT_OF_RESOURCE	20h	Card Services has exhausted resource
READ_FAILURE	09h	Unable to complete read request
Reserved	05, 0C, 10, 13h	Reserved for historical purposes
SUCCESS	00h	The request succeeded
UNSUPPORTED_MODE	16h	Processor mode is not supported
UNSUPPORTED_SERVICE	15h	Implementation does not support service
WRITE_FAILURE	12h	Unable to complete write request
WRITE_PROTECTED	1Ah	Media is write-protected

Table 20-4. Card Services Return Codes Listed in Numerical Order

Value	Return Code	Description
00h	SUCCESS	The request succeeded
01h	BAD_ADAPTER	Specified adapter is invalid
02h	BAD_ATTRIBUTE	Value specified for attributes field is invalid
03h	BAD_BASE	Specified base system memory address is invalid
04h	BAD_EDC	Specified EDC generator is invalid
05h	*Reserved*	Reserved for historical purposes
06h	BAD_IRQ	Specified IRQ level is invalid
07h	BAD_OFFSET	Specified PC Card memory array offset is invalid
08h	BAD_PAGE	Specified page is invalid
09h	READ_FAILURE	Unable to complete read request
0Ah	BAD_SIZE	Specified size is invalid
0Bh	BAD_SOCKET	Specified socket is invalid (logical or physical)
0Ch	*Reserved*	Reserved for historical purposes
0Dh	BAD_TYPE	Window or interface type specified is invalid
0Eh	BAD_VCC	Specified Vcc power level index is invalid
0Fh	BAD_VPP	Specified VPP1 or VPP2 power level index is invalid
10h	*Reserved*	Reserved for historical purposes
11h	BAD_WINDOW	Specified window is invalid
12h	WRITE_FAILURE	Unable to complete write request
13h	*Reserved*	Reserved for historical purposes
14h	NO_CARD	No PC Card in socket
15h	UNSUPPORTED_SERVICE	Implementation does not support service
16h	UNSUPPORTED_MODE	Processor mode is not supported
17h	BAD_SPEED	Specified speed is unavailable
18h	BUSY	Unable to process request at this time - retry later
19h	GENERAL_FAILURE	An undefined error has occurred
1Ah	WRITE_PROTECTED	Media is write-protected
1Bh	BAD_ARG_LENGTH	ArgLength argument is invalid
1Ch	BAD_ARGS	Values in Argument Packet are invalid
1Dh	CONFIGURATION_LOCKED	A configuration has already been locked
1Eh	IN_USE	Requested resource is being used by a client
1Fh	NO_MORE_ITEMS	There are no more of the requested item
20h	OUT_OF_RESOURCE	Card Services has exhausted resource
21h	BAD_HANDLE	ClientHandle is invalid
22h	BAD_VERSION	Client version is unsupported

Client Services (Client Registration and Support)

The category of card services defined as "client services functions" are those typically used when a card services client driver performs device initialization. Other services within this category provide basic card support. Table 20-5 lists the card services functions typically used during the registration process. The sections following the table discuss the registration process and discuss the use of each function listed.

Table 20-5. Client Services Functions

Client Services Functions	
Tuple Name	Description
GetCardServicesInfo	Determines if a valid copy of card services is installed and reports information regarding this copy of card services, including its revision and compliance level.
RegisterClient	Used by the client to register with card services as either a memory, MTD or I/O client. The client driver also specifies which card status events (such as card removal) it wishes to be notified of by card services. The client can also request that card services generate artificial card insertion events for all PC Cards that are currently installed, allowing the client to configure PC Cards it wishes to use.
DeregisterClient	Allows the client to notify card services that it no longer requires notification of status change events.
GetStatus	Returns the current status of the PC Card and its socket. It returns the same information obtained with the socket services GetStatus function.
ResetCard	This function resets the PC Card specified in the input argument, providing that all other clients that are using the same PC Card agree. Since more that one client may use a card, the ResetCard function will not be satisfied until all other clients agree to the reset. Card services generates Reset Request call-back events to all registered clients. Once all client drivers have responded to the call-back, card services calls the client that initiated the request via a Reset Complete call-back to inform the client whether the reset succeeded or failed.
SetEventMask	Used by the client to indicate the events it wishes to receive call-backs for. During registration, a client driver can specify which PC Card events that it wants to be notified of. This function can be used after registration to change the global event mask, originally set during RegisterClient. This function can also be used to change the SocketEvent mask originally set during Request-SocketMask (see table 20-8), but only if the RequestSocketMask function has been previously called by the client.
GetEventMask	Allows the client to obtain the current values of either the global or socket event mask.

Determining If Card Services Is Installed (GetCardServicesInfo)

The registration process begins with the card services client verifying that a valid copy of card services is installed and determining the compliance level of this particular version of card services. The GetCardServicesInfo function performs this task. When the card services client calls the GetCardServicesInfo function, it specifies a buffer size and pointer to the buffer where card services data is to be returned. Information returned by the GetCardServicesInfo function:

- **Length of data returned by card services.**
- **Card services signature**—Two consecutive bytes containing the ASCII characters "CS" verify the validity of the returned data.
- **Number of sockets**—Returns the number of sockets in the system.
- **Card services revision**—Indicates the vendor's revision level.
- **Card services compliance level**—Indicates the PCMCIA compliance level of card services. The compliance level is the PCMCIA release number upon which this socket services was based.
- **Location of vendor string**—Optional information can be provided by the card services vendor. This field specifies the start location within the buffer where the vendor information can be found. See "Vendor String" below.
- **Vendor string length**—Specifies the length of the vendor string.
- **Vendor string**—A vendor-defined string comprised of ASCIIZ characters.

Signing Up with Card Services (RegisterClient)

Once the card services client determines that an appropriate copy of card services exists, it then can register with card services using the RegisterClient function. A card services client driver registers with card services for notification of selected events generated by PC Cards. This function can also be used by the card services client to request that card services notify it of all PC Cards currently installed. This gives the card services client driver an opportunity to identify and configure the PC Cards that it requires access to.

In summary, the card services client registers with card services for the following reasons:

- To receive notification of specified PC Card status change events.
- To specify the type of client (memory, I/O or MTD) that is registering.
- To receive notification of PC Cards already installed in sockets (artificial card insertion events).

Note that card services returns a handle to the client upon return from the RegisterClient function. The client driver uses this handle to identify itself when calling other card services functions. Note that card services returns to the client drivers without having fully completed the registration process. Card services attempts to complete the registration process in the background and notifies the client driver that registration has been completed via the RegistrationComplete call-back.

Receiving Notification of Status Change Events

To receive notification of status change events occurring at the PC Card, the client driver must specify the events that it wishes to be notified of. This is accomplished by the card services calling a routine within the client when a card status change event occurs. This routine is referred to as the client's call-back routine. The card services client driver must specify the entry point of its call-back routine and the start address of a data buffer to deposit the change event into. Note that an event mask is passed to card services when the RegisterClient function is called, indicating to card services the events for which the client wants to be notified. Events that can be specified include the following (Refer to the section later in this chapter entitled "The Call-Back Process" for additional information):

- Write Protect change.
- Card Lock change (from HBAs that support a card interlock mechanism).
- Card Ejection request (HBAs supporting a card interlock mechanism).
- Card Insertion request (HBAs supporting a card interlock mechanism).
- Battery Dead.
- Ready Change.
- Card Detect Change.
- Power Management Change.
- PC Card reset request by another client.
- Socket Services Updated.

A given client determines which of the events it wants to be notified of during the registration process. For example, if the client driver so specifies, it can register with card services to receive card insertion events. This allows the client driver to be notified when a PC Card is inserted, permitting it to then check the PC Card to determine if it should configure the card.

The card insertion callback is triggered when a PC Card is inserted. Card services is notified via a status change interrupt generated by the HBA. Card services then interrogates the HBA to determine the cause of interrupt and calls back all client drivers that have registered to be notified of the card insertion event. When called-back each client driver then reads the card's CIS to determine if it should configure the card.

When call-backs occur, card services passes event information to the clients call-back buffer. The information passed typically includes an event code, logical socket number and information specific to the event. The exact information returned to the client depends on the specific event. Refer to the PCMCIA Card Services standard for details.

Note that the GetEventMask function can be used by the client driver to read the current setting of its event mask. The client passes its card services handle to identify itself, and card services returns the event mask indicating which status change events the client is currently registered to receive. Similarly, a client driver can call the card service's SetEventMask function to change the events for which it wants to be notified.

Determining the Order of Call-Backs: Client Driver Type

When a client driver registers with card services, it must also specify its driver type. For example, if a PC Card contains SRAM, flash memory, and I/O registers, the client driver that configures the card must contain a separate client driver for each group, and must register with card services three separate times as defined below:

- I/O client driver.
- Memory technology client driver (MTD) for Flash memory.
- Memory client driver.

The client driver type determines the order in which clients are called-back when a status change event occurs. I/O clients are called first on a Last In First Out (LIFO) basis; that is, the last I/O client registered is the first to be called. This is done on the premise that the last I/O client installed likely supersedes client drivers installed previously. MTD drivers are called next on a FIFO basis (the first to register is the first to be called). Finally, the memory client drivers receive the call-back last, also in a FIFO order.

Artificial Card Insertion Events

A client driver may also register with card services to have artificial card insertion events generated during the registration process. Card services can create a call-back to the client driver for each card currently installed in the system. In this way, the client driver's call-back routine can determine which of the cards already installed it should attempt to configure.

A client driver determines whether it should configure a card based on reading the CIS to determine if it recognizes the card. For example, a client driver may be designed to recognize a specific card (usually a client driver written by a manufacturer for only its card), or it may recognize any card within a given group (usually a client driver written for example to recognize all modem cards). When recognizing a card that it has been designed to configure and monitor, it then attempts to configure the card when an card insertion event occurs, providing that the card has not already been configured.

When artificial insertion notifications have been made for all PC Cards installed in sockets, card services generates a RegistrationComplete event. This event informs the client driver that the call-back process is complete. Note that when card services returns from the initial RegisterClient service, the registration process is not complete. Card services attempts to complete the registration process in the background; and therefore, the client is not fully registered until the RegistrationComplete call-back is received.

When processing the artificial card insertion events, the client driver may or may not recognize any PC Cards currently installed that it can configure. The client driver having registered with card services to receive card insertion events, will remain in memory and be called-back when a another PC Card is inserted sometime later. The client driver then checks to see if it can the configure this card.

Telling Card Services You're Leaving (DeregisterClient)

If a client driver will no longer be available at the call-back entry point (for example a driver that is transient), it must deregister with card services by passing its card services handle to card services and calling the DeregisterClient function. This tells card services that the client driver will no longer require call-backs.

Client Utility Services (Detecting a PC Card)

During the configuration process, the client driver must determine if it wishes to enable the PC Card, and if so, should attempt to configure it for operation. Once the client driver establishes that it will attempt to configure the PC Card, it may also be necessary to read additional information from the card to determine the specific resources it requires.

The GetConfigurationInfo function may be sufficient for many client drivers to determine if they should configure the PC Card. Other client drivers may need to further process the CIS to determine if it should attempt to configure the card. Card services assists with this by providing a group of utility functions that the client driver can use to obtain additional configuration information from the PC Card's CIS. These functions are listed in table 20-6.

Table 20-6. Client Utility Functions Used by the Client Driver to Access PC Card Information

Client Utility Functions	
Function Name	Description
AccessConfigRegisters	Used to access a PC Card configuration registers.
GetConfigurationInfo	Provides the client with information about a specified socket and the PC Card installed. This information can be used to determine the configuration requirements of the PC Card installed.
GetFirstTuple	Permits the client to specify a given tuple code and find the first occurrence of that tuple within the PC Card's CIS.
GetNextTuple	Requests that card services find the next occurrence of the tuple code that was previously specified for the GetFirstTuple function.
GetTupleData	Requests the contents of the specified tuple, once it has been located using GetFirst/NextTuple.

Table 20-6. Client Utility Functions Used by the Client Driver to Access PC Card Information (Continued)

Client Utility Functions	
Function Name	Description
GetFirstRegion	Used by memory technology client drivers (MTDs) to get device information for devices defined for the first region within the PC Card (as defined in the card's CIS). Information received by the client includes: location of region within the card, size of region, speed of devices within region, memory type (attribute or common), erase/write capabilities, etc.
GetNextRegion	Finds the device information for the next region within the card.
GetFirstPartition	Similar to the GetFirstRegion function, this function returns information for the first partition on the card based on information contained in the PC Card's CIS. If a PC Card has no partition information defined in its CIS, then card services may be able to determine partition information based on a given file system structure (such as the BIOS parameter block (BPB)/FAT structure used by DOS).
GetNextPartition	Finds device information for the next partition.

Client drivers can use these utility functions to obtain information regarding the configuration of the PC Card in a given socket, or to scan the CIS itself to determine the exact configuration requirements of the PC Card. If the client driver is a memory drive, the job of determining the configuration requirements can be quite simple, since it is likely that the first tuple (Device Information Tuple) within the CIS will provide the client driver with much (if not all) of the information it needs to configure the card. Tuple processing for I/O devices can be considerably more challenging due to the resource combinations that may be required.

Evaluating the PC Card and Socket (GetConfigurationInfo)

The GetConfigurationInfo service provides the enabler with information about the specified socket and card. An enabler may call this function to determine if the card installed into the socket has already been configured. If not configured the information returned to the enabler provides a general view of the card installed in the socket. Refer to table 20-7 for a list of the information returned by the GetConfigurationInfo service.

The GetConfigurationInfo service returns information from the PC Card's CIS including the device ID, function ID, and manufacturing ID. This information provides a way for the enabler to quickly determine whether or not it should attempt to configure the card.

Table 20-7. Information Returned by the GetConfigurationInfo Service

Information Returned	Description
Logical socket/function number	This field contains the logical socket and function number specified.
Attribute Bits	Indicates whether the PC Card has been previously configured and if exclusively owned. Also provides miscellaneous information regarding the configuration of the card.
Vcc setting	The values returned in these fields are those that the configuring client driver passed to card services during RequestConfiguration call. If the card/function has not been configured, these values are invalid.
Vpp1 setting	
Vpp2 setting	
Interface type	
Config. register base address	
Status register settings	The values returned in these fields are the values that were written to the configuration registers by the enabler when it called the RequestConfiguration service. These values are invalid if the card/function is not configured.
Pin replacement register settings	
Socket and copy register settings	
Config. option register settings	
Config. Registers implemented	This values is obtained from the information passed to card services during the RequestConfiguration call.
First device type	This value is taken from the DEVICE tuple.
Function code	These values are taken from the Function ID tuple.
System initialization byte	
Manufacturers code	These values are taken from the Manufacturers ID tuple.
Manufacturers Information	
Card values	This field is a bit map that indicates which configuration register were written with valid values.
Assigned IRQ	These fields contain the values specified when the RequestIRQ service was called for this function/card.
IRQ attributes	
Base ports 1	These fields are derived from the information specified when the RequestIO function was called. If the Request-IO function has not been called the number of ports fields will contain 00h.
Number of ports	
Attributes 1	
Base ports 2	
Number of ports	
Attributes 2	
I/O address lines	
Extended Status	Contains the value written to the extended status register when the RequestConfiguration call was made.
DMA Attributes	Defines the DREQ# pin assignments and DMA width.
Assigned Channel	Specifies the DMA channel requested during configuration.
Number of I/O windows	Specifies the number of I/O windows in use for this socket and function.
Number of memory windows	Specifies the number of memory windows in use for this socket and function.

Additionally, the GetConfigurationInfo function provides specific configuration information about a socket and card that has already been configured. If the card has been previously configured, then card services returns the client handle (in handle argument, DX register) of the enabler that has already configured the card, along with the primary configuration settings. If the card has not been configured then the client handle and configuration settings returned by the service are invalid.

Note that support for multiple function cards has been added. An enabler can specify the logical socket and a function within the PC Card that it wishes to get information about.

Scanning the CIS (GetFirstTuple, GetNextTuple, GetTuple Data)

When the client driver must determine the specific configuration requirements of the PC Card, it reads the configuration table within the PC Card's CIS. The client driver can use the GetFirstTuple function to specifically request the tuple containing the information it needs. For example, if a client driver wishes to find the first Configuration Table Entry within the CIS, it passes the socket number and the desired tuple code (1Bh for the configuration table entry tuple) to card services and calls the GetFirstTuple function. Card services will scan the card's CIS looking for the first instance of the tuple code that was specified in the call.

The GetTupleData function can be called next to obtain the data within the tuple. When the data is returned, the client driver interprets the data to determine the system resources required by the PC Card. The client then attempts to obtain these resources from card services and, if successful, no further tuple processing is necessary. However, if the system resources specified in the first configuration table entry are not available, then the client must continue processing the CIS by calling the GetNextTuple function, which finds the next occurrence within the CIS of the indicated tuple type. This process continues until the resources specified by a Configuration Table Entry are determined to be available. If no more tuples of the type specified exist within the CIS when the GetNextTuple is called, card services returns a code indicating that no more items are available.

Note that the GetFirstTuple, GetNextTuple, and GetTupleData functions use the same argument packet format. This simplifies calling these utility functions (since the argument packet returned by one function can be used when calling the other).

Simplifying CIS Processing for Memory and MTD Clients (GetFirstPartition, GetNextPartition, GetFirstRegion, GetNextRegion)

Some client drivers may need to obtain information describing partitions and regions within memory cards. Since obtaining the necessary information requires reading multiple tuples, the GetFirst/NextRegion and GetFirst/NextPartition functions can be used by clients to get the required information without having to process the tuples individually.

Resource Management Services (Assigning Resources)

Card Services maintains a database of resources available within the system. Client drivers can call card services to verify availability of resources needed by their PC Card. Configuring a PC Card and programming the HBA is a two step process.

1. The client driver must acquire each resource from the resource table one at a time. If any of the resources required are not all available, this particular combination of resources cannot be satisfied and another group must tried.
2. Once all resources required by the PC Card have been successfully allocated, the actual configuration (allocation of these resources to the HBA and PC Card) occurs.

The resource management functions allow the client driver to verify the availability of and to allocate resources required by the PC Card. These functions are listed in table 20-8. The services in the shaded boxes were added by the PC Card 95 standard. Refer to the card services specification for details related to these functions.

Table 20-8. Resource Management Functions

Resource Management Functions	
Function Name	Description
RequestIO	Used to request I/O address ranges for the PC Card. This function can be called only once per socket, and a maximum of two I/O address ranges can be specified per card. Input parameters request the starting or base address for each range and the number of I/O address locations requested for each range, and whether a given address range is to be shared with other devices within the system. This function, if successful, assigns the specified I/O address ranges to the client and adjusts the card services resource table to indicate that the assigned ranges are no longer available.
RequestIRQ	Used to obtain a system interrupt line for the calling client. The client specifies which interrupt line or lines will satisfy its interrupt needs. Input parameters request that an interrupt be either exclusive (not shared), time-multiplexed shared (client coordinates with other clients sharing this line, using the ModifyConfiguration function to enable and disable its connection to the interrupt line) or shared dynamically through an interrupt sharing protocol supported by the system. An input parameter also specifies whether the interrupt sent from the PC Card should be pulse or level mode. This function, if successful, assigns the specified IRQ line to the client and adjusts the card services resource table to indicate that the assigned IRQ is no longer available.
RequestWindow	Allows the client to request ownership of a block of system memory addresses. The client passes the starting (base) address and the size of the memory window along with a variety of other parameter to card services. Other parameters include: type of memory window (attribute or common), window enabled or disabled, whether window can be shared with other clients (only time-multiplexed sharing is permitted), whether paging of window is enabled, and speed of the memory devices. This function assigns the address ranges (if available) to the client and adjusts the resource table to indicate that they are no longer available. Note that this same block of addresses can be assigned to another client if the shared parameter is set. This function can be called multiple times per socket, up to the maximum number of memory windows supported by the HBA. Card services passes a window handle back to the client to be used when calling other functions pertaining to this window.
RequestDMA	Requests the use of a desired DMA channel for a given PC Card. One of 16 DMA channels can be specified (numbered zero through fifteen). Also specifies whether the DMA channel is to be shared and if so by which method, which PC Card pin is used for DREQ#, and what the DMA channel size is.
ReleaseIO	Adjusts the resource table by releasing the I/O address range(s) acquired by a client with the RequestIO service.
ReleaseIRQ	Adjusts the resource table by releasing the IRQ acquired by a client with the RequestIRQ service.
ReleaseWindow	Adjusts the resource table by releasing the block of memory address locations acquired by a client with the Request Window function. The window handle is passed to card services to specify the window to be released.

Table 20-8. Resource Management Functions (Continued)

Resource Management Functions	
ReleaseDMA	Adjusts the resource table by releasing the DMA channel acquired by a client with the RequestDMA service.
GetFirstWindow	This service returns the window handle and information associated with the first memory or I/O window (i.e. base address, size, and related attributes) of the specified socket and function.
GetNextWindow	This service is called following the GetFirstWindow call to obtain information associated with the next window used by the specified socket.
GetMemPage	This service return information for the specified page within the requested memory window. The window is specified by the window handle returned by the GetFirstWindow or GetNextWindow services.
ModifyWindow	Allows parameters assigned to a given block of memory addresses acquired with the RequestWindow function to be modified. These parameters include memory device speed, window type (attribute or memory) and window enabled or disabled. The window handle is passed to card services to specify the window to be modified.
MapMemPage	Selects a 16KB memory block within the PC Card to be mapped into a 16KB page within system memory. The 16KB memory block within the PC Card is identified by the client with an absolute offset value from the beginning of the PC Card's memory array.
RequestSocketMask	Selects the status change events that the client wishes the PC Card to generate. The client specifies which status change events it wants to be generated at the socket. A bit-map of the events masks each status change event that should not be reported by the HBA from the specified card. Note that during the RegisterClient function, the client driver indicates which status change events it wishes to be notified of, setting a global event mask.
ReleaseSocketMask	Releases the status change events mask, so that no status change events are reported by the PC Card residing in this specified socket.
ModifyConfiguration	Allows the configuration established by the Request Configuration function to be modified. Note that IRQ routing and the I/O address range assigned cannot be modified with this function. These parameters can only be changed by first releasing the configuration and then performing the requests for those resources again.
RequestConfiguration	Used to establish the configuration for an I/O interface. The I/O address ranges and system interrupt previously acquired are established at the hardware level (HBA and PC Card). Other configurable items are also specified based on the values indicated by the selected Configuration Table Entry, including: Vcc, Vpp1, Vpp2, interface type (memory only or memory/IO) and setting for the configuration registers, if present.
ReleaseConfiguration	This function releases the configuration information set previously using the RequestConfiguration function. This function returns the interface to a memory-only interface and power is removed from the socket (if no memory client indicates its use of the PC Card). The IRQ and I/O resources must be released separately to adjust the resource table.

Requesting a Resource

A client driver may use three types of request functions to determine if the resources that its PC Card requires is available. These functions include:

- RequestIO—used to request a range of I/O address locations
- RequestIRQ—used to request a system interrupt line
- RequestWindow—used to request a range of memory address locations
- RequestDMA—used to request a DMA channel.

A client driver whose PC Card requires one or more of these system resources calls card services to determine their availability. The client passes its handle to card services along with a pointer to the memory buffer containing the input argument packet. The argument packet passed to card services specifies parameters identifying the resource being requested. Card services checks the allocation table to determine if the requested resource is available. If available, card services updates its resource table, indicating that the resource is no longer available and returns "success" along with the argument packet, verifying that the resource parameters that have been granted.

Once all of the resources required by a PC Card have been acquired with the request functions, the actual task of programming the HBA and configuring the card can then occur. See the chapter entitled, "Client Drivers".

Card services has no way of knowing what resources are available for a PC Card to use. As a result, platform-specific utility programs have been written to probe the system and build a data base of available resources. This data base is passed to card services to manage.

Requesting Resource Combinations

Consider the example of a serial port that typically requires a range of I/O addresses and an IRQ. In a PC-DOS environment, a serial port is typically configured either as COM1 (I/O locations 3F8h-3FFh & IRQ4), COM2 (2F8h-2FFh & IRQ3), COM3 (3E8h-3EFh & IRQ4) or COM4 (2E8h-2EFh & IRQ3). The client driver for a serial port must ensure that both the RequestIO and the RequestIRQ functions return success before configuring the PC Card and HBA.

Assume that the client attempts to configure the serial port as COM1. If the RequestIO function returns "success", then I/O locations 3F8h-3FFh are allocated to the client driver and the resource table is updated to indicate these I/O addresses are no longer available. Next, the client driver calls RequestIRQ to obtain IRQ4, but card services returns BAD_IRQ to the client, indicating that IRQ4 is not available. If the client simply moved to the next configuration option (COM2), the I/O address range 3F8h-3FFh would remain allocated and other clients requesting an address within that range will not be successful, even though the addresses are not being used.

To avoid this problem, the client must release resources that have been granted but will not be used. The ReleaseIO function would be used in this instance before moving on to the next configuration option. Similarly, the ReleaseIRQ and ReleaseWindow are used to release interrupts and memory address ranges, respectively.

Configuring the HBA and PC Card (RequestConfiguration)

When the client driver has obtained from card services all of the resources needed by the card, then the actual configuration can take place. Prior to this time the resources have been granted to the client driver for assignment to its PC Card, but neither the HBA nor the PC Card have yet been configured to use these resources.

The card services client uses the RequestConfiguration function to complete the configuration process. When the RequestConfiguration function is called, card services makes the appropriate calls to socket services to set the specified values into the window registers and IRQ steering registers. Also, the index number of the Configuration Table Entry whose configuration options were successfully allocated is written to the card's Configuration Option Register, located in attribute memory.

The client must ensure that it is ready to perform all of the functions associated with a fully-operational card before calling the RequestConfiguration function. Once the function call completes, the PC Card and HBA are configured and the PC Card is now "on line". For example, in an x86 environment, if interrupts are used by a given PC Card, the client driver must ensure that the pointer to the device's interrupt service routine has been installed in the interrupt table prior to configuring the card. It will then be prepared to handle the card's interrupt requests.

Bulk Memory Services

Bulk memory services primarily relate to memory clients, utility programs, execute-in-place (XIP) managers, and other clients requiring access to memory cards. These clients can use bulk memory functions to access memory devices without knowing the details of the various memory technologies used by PC Cards. The functions within the bulk memory services group support RAM devices, but not devices such as flash memory.

Table 20-9. Bulk Memory Functions

Bulk Memory Services Functions	
OpenMemory	This function opens an area of common memory within a PC Card that is to be accessed some time in the future (i.e. read, write, copy or erase operation). A memory handle is returned that identifies this memory range when performing one of the operations mentioned above.
ReadMemory	This functions reads data from an area of common memory specified by a given memory handle (obtained from the OpenMemory function). The calling MTD passes a pointer during the call specifying a system memory buffer to which data is to be returned.
WriteMemory	This function writes data to a common memory area identified with a memory handle obtained via the OpenMemory function. The calling MTD passes a pointer to a system memory buffer that contains the data to be written.
CopyMemory	This function reads data from a source location and writes it to a destination within the same common memory region that is identified by a memory handle obtained via the OpenMemory function.
CloseMemory	This function closes an area of common memory that was previously opened with the OpenMemory function. The calling MTD passes the memory handle of the memory area to be closed along with the call.
RegisterEraseQueue	Establishes an erase queue where erase entries can be made.
CheckEraseQueue	Notifies card services that one or more erase request entries have been sent to the erase queue.
DeregisterEraseQueue	Eliminates an erase queue previously registered using the RegisterEraseQueue function. This function fails if erase entries within the queue are still pending completion.

Since flash memory devices require particular erase and write algorithms, PCMCIA chose not to attempt embedding the code necessary to support all potential variations into card services. Instead, a memory device that requires a specific algorithm must supply a memory technology driver (MTD) that is designed to handle access to the card. When a client such as a memory client attempts to access memory within a flash card, card services passes the request to the flash MTD, which makes the low-level access to the memory device. Table 20-9 lists the bulk memory functions and provides a brief description of each.

Advanced Client Functions

Advanced client functions include miscellaneous functions that satisfy the special needs of some client drivers. Table 20-10 lists the advanced client functions and provides a brief description of each. Refer to the PCMCIA card services specification for details.

Table 20-10. Advanced Card Services Functions

Advanced Card Services Functions	
ReturnSSEntry	Provides a means of gaining access directly to socket services. Normally, access to socket services is denied by card services to ensure that it maintains synchronization with the state of the HBA. If client drivers are allowed access to socket services, the HBA setting can be modified without card services knowledge. If absolutely required, a client driver can request access to socket services via the ReturnSSEntry call. The programmer must be certain that nothing is changed at the HBA level that will affect the operation of card services.
MapLogSocket	Determines the physical adapter and socket that is assigned to a logical socket number.
MapPhySocket	Identifies the logical socket number assigned to a physical adapter and socket.
MapLogWindow	Identifies the physical adapter and window that are mapped to a given logical window handle.
MapPhyWindow	Identifies the logical window handle assigned to a given physical adapter and window.
RegisterMTD	Assigns an MTD to a region of memory. When access to the assigned region occurs, the MTD is called to handle the memory operation.
RegisterTimer	Allows a client driver to register for callback at specified time intervals. A client may register multiple times to get notification at various time intervals. Timing is based on 1ms interval. The client specifies the call-back interval based on the number of 1ms ticks specified during registration. A timer handle is returned during registration and passed to the client when the call-back occurs. This permits the client to identify the specific timer that has expired when the call-back occurs.
SetRegion	Allows a client driver of a card that does not have a CIS to specify the characteristics of a given region within the card.
ValidateCIS	Scans the CIS by reading the tuple chain contained on the PC Card. The function returns the number of valid tuples found within the chain.

Table 20-10. Advanced Card Services Functions (Continued)

Advanced Card Services Functions	
RequestExclusive	Permits a client driver to request exclusive access to a given PC Card. Card services ensures that no other client is currently using the card before granting exclusive access to this client driver. If another client driver is currently using the card and is unwilling to release control, then function will fail.
ReleaseExclusive	Releases exclusive access to a card that was previously granted via the RequestExclusive function.
GetFirstClient	Returns the client handle of the first client to register with card services.
GetNextClient	Returns the client handle of the next client to register with card services.
GetClientInfo	Provides client driver information for the client handle specified.
AddSocketServices	Allows another socket services handler to be installed to support an additional HBA.
ReplaceSocketServices	Replaces the current version of socket services with a new version.
VendorSpecific	Defined by the vendor of card services to extend functionality.
AdjustResourceInfo	Adjusts the resource database maintained by card services. This data base contains the system resources that are available for use by PC Cards. This function allows system resources to either be added or removed from the database.

The Call-Back Process

Card service makes call-backs to clients that are triggered by a wide variety of events. The type of call-back events can be categorized as:

- Card insertion/removal events
- Registration complete event
- Status Change events
- Card insertion/ejection request events
- Exclusive request/compete events
- Reset request/complete events
- Client Information request event
- Erase Complete event
- MTD Request event
- Timer event
- New socket services event

When making call-backs card services uses the call-back entry point specified by each client during registration. The specific events supported by card services are listed in figure 20-11.

Some events must be supported by all clients. During registration, the client driver specifies the individual events that it wishes to be notified of. The events that must be supported include:

- Client_Info — a client may request information about another client when calling the GetClientInfo service. Card services calls-back the specified client using the Client_Info call-back.
- Exclusive_Request — an client that has previously configured a PC Card may receive a RequestExclusive call-back, indicating that another client wishes to gain exclusive access to the PC Card. For example, a generic client driver may have enabled a modem, but a device-specific client driver may want to gain exclusive access to the same PC Card.
- Reset_Request — request by a client to reset a socket/PC Card must be granted by other clients using the same socket/PC Card. This call-back notifies a client that a ResetRequest has been made.

Identifying a Status Change Event

When a status change event occurs at one of the PCMCIA sockets, an interrupt is generated by the HBA. Card services is notified of the event via a system interrupt (called a status change or management interrupt). When the card services receives the interrupt, it must determine which socket encountered the status change event. Card services accomplishes this by calling the socket services AcknowledgeInterrupt function which returns the socket(s) that experienced the status change event. Once the socket or sockets that have experienced a status change have been identified, then card services calls the GetStatus function to determine which event caused the interrupt.

The AcknowledgeInterrupt function must be called once for each HBA in the system. The client supplies the HBA number to socket services when the AcknowledgeInterrupt function is called, and socket services returns a bit map of the sockets within the adapter that have experienced a status change. When obtaining status information from the HBA, socket services also prepares the HBA to generate another status change interrupt if another should occur.

The AcknowledgeInterrupt function only identifies the sockets that have experienced a status change. After the AcknowledgeInterrupt routine completes, card services then calls the socket services GetStatus function, HBAs typically preserve the state of the status change so that the exact status change event that caused the interrupt can be determined using the GetStatus function. If the HBA does not preserve this state information, then socket services must.