PCMCIA

SDK

PC Card Software Development Kit

7.00

.apsoft

WE HELP COMPUTERS WORK

Release 7.00

Proprietary Notice and Disclaimer

APSoft.
Sonnenstrasse 26b
85622 Feldkirchen
Germany

Tel:	+49 (0) 89 900 479 0
Fax:	+49 (0) 89 900 479 11
Internet:	http://www.tssc.de

Table of Contents

C H A P T E R 1

PC Card Basics

Standard Add-In Installation

Traditionally, installing personal computer add-in devices was at best tedious and at worst a nightmare. The simplest way of installing a device was to plug it into an external I/O port and then install or configure the system software to recognize the device. Modems, mice, and printers, which use either serial or parallel ports, are normally installed this way. In such an installation, the system usually must be reset before it can recognize the device.

Internally mounted bus devices have traditionally required that the power be turned off, the system case opened, and a board of some sort be inserted in an I/O bus slot. Prior to insertion, such a device often must be configured by jumpers and/or DIP switches. The process of configuration itself could often turn into a trial-and-error process, because vital (and limited) system resources must be selected and allocated to the device. These resources are I/O and memory address space, IRQs, and DMA channels. Conflicts are common, especially as more devices are added. Coupled with the frustrations of working in the cramped interior of a computer case and dropping small screws into obscure places on the motherboard, the entire process is daunting.

Adding memory falls somewhere between connecting to an external port and installing a configurable board. Still, the system must be powered off, the case opened, and delicate memory chips or SIMMs handled.

Mobile Computing, a Driving Force

In recent years, mobile computing has been the fastest growing sector of the personal computer industry. Given the reduced dimensions of the portables, laptops, and notebooks, manufacturers elected to handle I/O expansion by the inclusion of proprietary expansion slots, called *sockets*. The most common peripherals used in such slots were modems and FAX devices. Unfortunately, proprietary designs meant that one manufacturer’s card simply would not work in another manufacturer’s socket.

The first efforts at standardization began by addressing memory expansion. The growth of the portable market had made the marketing of nonvolatile memory IC expansion cards possible. In September 1990, the Personal Computer Memory Card International Association (PCMCIA) introduced Release 1.0 of the PC Card Standard, which defined a standard for the current 68-pin memory IC interface.

Even at that time, it was still very apparent that the need for a standard I/O bus was needed. In August 1991, Release 2.0 of the standard added this I/O interface plus execute-in-place and a definition of a low-level software interface called Socket Services.

In 1992, Release 2.1 greatly expanded the definition of compliant I/O devices as well as defining a high-level software interface called Card Services.

In 1995, PCMCIA and JEIDA (Japan Electronic Industry Development Association) decided to co-publish the PC Card Standard as a 12-volume set. This release (tagged by date only) represented a breakthrough in cooperation between the two largest groups that developed standards related to removable IC cards. The February 1995 release defined a new 32-bit CardBus PC Card interface in addition to other technology enhancements.

The new Release 7.0 of the PC Card Standard has added a variety of new features including a new Small PC Card form factor and PCI-style Power Management specification for CardBus.

In sum, the PC Card Standard addresses devices which have no jumpers or dip switches, are entirely software configured and can be inserted and removed with the system fully powered (hot swapping). The standards themselves only assert minimum requirements and are open-ended enough for individual product differentiation. At the minimum, the standards allow “plug and play” interoperability such that a PC Card can be used in a number of different sockets. Product differentiation, therefore, is a function of special support through a specific client device driver.

PC Card Technology

Hardware

PC Card hardware comprises the socket and its controller, on the system side, and the card itself, with its identifying Card Information Structure (CIS).

Socket and Controller

The lowest layer of the PC Card hardware architecture is the PC Card socket. There can be any number of sockets, but one, two and four are the most common configurations. Two or more sockets are usually joined to a controller on the system I/O bus (AT or PCI). Unlike common adapters such as hard disks and modems, this controller uses no standard I/O addresses nor are there any standard mappings for its registers.

It is the function of the *software supporting the controller* to map the card's memory and I/O resources into the system.

CardBus (32-Bit) PC Cards

CardBus is a new PC Card technology based on the Peripheral Component Interconnect (PCI) bus. CardBus technology comprises cards, sockets, connectors, and the bus. The CardBus connector is quite similar to the original PC Card connector. However, there are quite a few operational and electrical differences between the CardBus design and the standard PC Card technology design. All sockets designed for CardBus PC Cards ***MUST*** also support 16-bit PC Cards, but the reverse is not true.

R2 (16-Bit) PC Cards

The pre-CardBus, or classic, PC Card, is commonly referred to as an **R2** or 16-bit PC Card. R2 indicates Release 2.x of the PC Card Standard.

Card Types

The PC Card Standard defines three PC Card Types. These types are really only differences of thickness:

Type I.	The thinnest cards, mainly used for memory cards. Thickness: 3.3 mm.
Type II.	Slightly thicker than type I, used mostly for fax/modem and LAN cards. Thickness: 5.0 mm.
Type III.	Thickest; used by rotating hard drives. Thickness: 10.0 mm.

Some PC Card manufacturers offer cards thicker than type III cards. They are sometimes called type IV cards, although there is no type IV specification in the PC Card Standard.

Card Information Structure

Every PC Card has some identification data that the operating system reads. The data are in the Card Information Structure (CIS). Typically, the CIS contains information about the card manufacturer, its function(s), and how to set up the card for use. Chapter 6 describes the CIS format and architecture.

Software

The PC Card software specification consists of two separate but interrelated services.

Socket Services

Socket Services is the lowest software layer and, in some respects, tends to resemble a BIOS. Each socket services driver is written specifically for a single type of PC Card controller and is intended to have a small memory footprint — small enough to be ROM-able. It is seldom advisable for a programmer to directly access Socket Services. If it is necessary to do so, great care must be taken.

Card Services

Card Services provides a high level API to programmers. Card Services uses Socket Services and insulates the programmer from the specific details of communicating directly with a given controller. In some respects, Card Services can be viewed as an operating system extension, mediating access to the BIOS (in this case Socket Services) while providing more detailed services to the programmer. Again, though still possible to directly address Socket Services, programmers are advised that this could be very risky.

Limitations

There are two fundamental limitations to PC Cards. The first, Resource Management, relates to reliably configuring PC Cards while the second, Compatibility, poses the greatest limitation to today's standards.

Resource Management

I/O devices require system resources to move data and to communicate with the system. This means that such a device, to be accessible, needs to occupy some address space within system I/O or memory space. It also typically requires an unused Interrupt Request (IRQ) line to signal its status and, frequently, an unused DMA channel to move the data to and from the device.

Even today, jumpers and dip switches are the common features of most I/O devices. Even if a device is configured by software, the needed resource must be available or it simply cannot function correctly.

With PC Cards and "hot swapping", the ordinarily static pool of available system resources becomes truly dynamic. It has become easy to envision a scenario where a user plugs in a fax/modem PC Card, has IRQ 4 allocated to it, downloads a file from a remote location then unplugs the modem and inserts a

LAN card, also allocated IRQ 4, and moves the file to a server, all while never turning the system off.

The problem becomes a matter of somehow managing system resources intelligently so that a given resource, such as IRQ 4, is available to different devices at different times without recourse to powering down.

Compatibility

Even in the event that all current and next generation PC Cards conform to the standards, what is to become of older cards and controllers? Is it realistic to ignore them? The answer obviously is no.

Even more difficult to imagine is that all current and future devices will be entirely compatible. Manufacturers simply cannot redesign from the ground up whenever there is a change in the standards. Moreover, the standards themselves have been intentionally open-ended to allow product differentiation. Occasionally, the most minor deviation may mean a given card may not be detectable on a particular platform and by certain software.

APSoft PC Card Solutions

APSoft has approached the above PC Card limitations along two related paths:

1. CardWare® Card and Socket Services
2. TheAPSoft PC Card Software Development Kit (SDK)

CardWare

CardWare is APSoft's PC Card software. Offering functionality for DOS, Microsoft Windows 3.x, Windows NT 4.0, 98/ME, 2000/XP operating systems, CardWare dynamically allocates and de-allocates system resources while supporting virtually all major controller types and any PC Card. While distributed commercially, CardWare is included with the SDK exclusively as a test and development tool and is not herein being licensed for distribution.

Software Development Kit

APSoft has an extensive and comprehensive understanding of the PC Card Standard and its goals, as a natural outgrowth of APSoft's ongoing participation in and contribution to the PCMCIA Committee's efforts at arriving at viable PC Card Standard. APSoft representatives chaired the groups that defined Socket Services and Card Services and actively participated in several working groups and sub-committees.

One of the goals of the PC Card Standard is to ensure true "hot swapping" and interoperability. A main vehicle of the Standard is the Card Information

Structure (CIS). A valid CIS can be parsed by a generic enabler, and information passed to Card Services for use in configuring the card in the system. Additionally, the specification is broad enough to allow the flexibility to differentiate products through the use of Card Services Client Device Drivers.

SDK Tools

The APSoft PC Card SDK provides a set of tools to aid PC Card hardware and software vendors in developing and testing PC Card compliant CISes and Client Device Drivers. It includes a sample CIS and a utility to read and parse any CIS compliant with the new PC Card Standard.

Also included is a sample Card Services client with source skeletons for Windows NT, DOS, and 16-bit Windows 3.x, plus an adapter detection utility to quickly determine the PC Card adapter type and model. In addition, utilities are included that assist in testing on PCI, Plug-and-Play and Power Management enabled systems.

Perhaps the most important inclusions are sample versions of APSoft's CardWare products. These are included as real-world aids for developing and testing purposes only and are NOT for distribution or re-sale.

Help Files and Additional Resources

The APSoft PC Card SDK is held together by Help files. The package includes files compatible with Microsoft Windows 3.x or with Windows NT 4.0/95/98/ME/2000/XP. The files can be viewed separately or through the Help function of your Windows-based development environment. In addition to supplying information about the latest standards, information is available at the click of a button to show differences among the various standard levels.

Lastly, it is worth noting that this kit is not intended as a PC Card developer's guide. Instead, it comprises a set of tools designed to aid the development of compliant devices and support software based on the latest PC Card Standard. For more information about PC Card development, we recommend the following:

PC Card Standard. Personal Computer Memory Card International Association.. http://www.pc-card.com

Power Management Coordinator API Specification Revision 1.00. Intel Corporation, 8 April 1994

The PCMCIA Developer's Guide Second Edition. Michael T. Mori and Dean Welder. Sycard Technology (ISBN 0-9640342-1-2)

Advanced Power Management Specification v1.1. Intel Corporation.

C H A P T E R 2

System Requirements

Running CardWare for Windows 3.x

To properly utilize CardWare in Windows 3.x, your computer must have these:

- At least an 80386 or compatible processor.
- At least one PC Card socket
- A compatible PC Card Interface Controller. In Appendix **Error! Reference source not found.** you may find the full list of supported controllers.
- At least one available IRQ level for routing status-change interrupts from the socket controller to CardWare. At least one additional available IRQ level is suggested for each PC Card socket in your system to support PC Cards that use hardware interrupts.
- 4 KB of unused memory address space in the expansion ROM area of your system for CardWare internal use. It is usually best to set aside an additional 16 KB of memory address space in the expansion ROM area for each PC Card socket in your system.
- Up to 2 MB of system RAM for use by CardWare, if all components are loaded.
- Approximately 1.5 MB of free space on your hard drive.
- Floppy disk drive (3.5-inch) for installation.
- EGA/VGA video subsystem.
- Microsoft DOS version 3.2 or newer. PCDISK.EXE requires MS-DOS 5.0 or newer.

Check the CardWare README file for the latest information about system requirements.

Running CardWare for Windows NT 4.0

To properly utilize CardWare in Windows NT 4.0, your computer must have these:

- At least a Pentium or compatible processor
- At least one PC Card socket
- A compatible PC Card Interface Controller. In Appendix **Error! Reference source not found.** you may find the full list of supported controllers.
- At least one available IRQ level for routing status-change interrupts from the socket controller to CardWare. At least one additional available IRQ level is suggested for each PC Card socket in your system to support PC Cards that use hardware interrupts.
- Approximately 1.5 MB of free space on your hard drive.
- Floppy disk drive (3.5-inch) for installation.
- EGA/VGA video subsystem.
- Microsoft Windows NT 4.0

Check the CardWare README file for the latest information about system requirements.

Using the SDK

These are system requirements for the CardWare Software Development Kit:

- A C++ compiler. The source code included has been designed for Microsoft Visual C++ 4.3 and the Microsoft Foundation Class (MFC) library.
- A debugger or other utility with the ability to read/write binary file data.
- The Windows help file display utility (WINHELP32.EXE).
- Microsoft WinWord 97 or newer
- CD-ROM drive to load the SDK software.

C H A P T E R 3

CardWare for Windows 3.x

This chapter describes how CardWare manages system resources and how PC Cards are configured under Windows 3.x.

System Resources

Because of the lack of a System-Resource-Manager in today's PC systems, the PC Card Standard has defined a separate resource-management handled by Card Services.

In future systems a System-Resource-Manager (called Configuration Manager) will be implemented who manages the resources for the whole system. For such systems, Card Services must be modified so they pass all resource requests to the Configuration Manager.

In environments that have no resource management (such as DOS/WIN 3.x), Card Services manages a pool of resources for allocation to PC Cards. The developer must take care to avoid the possibility of resources being assigned to Card Services that are in use by non-PC Card devices, or a conflict can occur that may crash the system. In environments that have resource management (such as Windows NT 4.0), Card Services works cooperatively with the resource manager to request and assign resources to PC Cards.

During boot time the Resources Pool for Card Services has to be initialized. This is done by the PC Card Resource Manager (`PCRM.EXE`). In a standard configuration PCRM first detects missing COM- and LPT-Ports and adds the appropriate resources to Card Services. In the next step PCRM adds all unused IRQs to Card Services. Then PCRM adds/removes all resources which are listed in the [Resources] section of the `CARDWARE.INI` file to the resources pool of Card Services.

Remember that PC Cards can only be configured successfully if the required resources are available to Card Services. That means the resource must be in the pool and may not be used by another PCMCIA device.

For more information about PCRM, check the CardWare User Manual.

Card Configuration

When you insert a PC Card into your system, CardWare reads the Card Information Structure (CIS) from the Card. The CIS normally supplies information about the type of card (e.g. LAN or modem), and how the card can be configured.

In the next step CardWare searches its internal database for a specific record for this card. If such an entry exists, CardWare tries to configure the card with one of the associated Device Definitions.

If no record exists, CardWare checks whether the card is Generic. If yes, CardWare tries to configure the card with one of the associated Device Definitions. If this fails CardWare tries to Auto Configure the card.

If the card is neither Specific nor Generic, CardWare tries to Auto-Configure the card. In this case CardWare tries to configure the card with one of the Device Definitions defined (but not associated). If this fails, CardWare tries to create a Device Definition on the fly and to configure the card with this device.

If a PC Card can not be configured it may be due to the following reasons:

1. The required resources are not available to Card Services.
2. The required resources are in use by another PC Card.
3. The PC Card does not support the configuration which you are trying to use.

How CardWare Identifies a PC Card

CardWare uses the following information for identifying a PC Card:

Tuple	Description
CISTPL_VERS_1	Product information string - Name of Manufacturer - Name of Product
CISTPL_MANFID (if available)	Manufacturer Identification tuple
CISTPL_FUNCID (if available)	Function Identification tuple

If the CIS contains at least one valid chain, any subset of the above tuples is enough for card identification.

If the CIS is invalid, CISTPL_VERS_1 is mandatory.

This page intentionally blank.

C H A P T E R 4

CardWare for Windows NT 4.0

This chapter describes how CardWare manages the system resources and how PC Cards are configured in the Microsoft Windows NT 4.0 operating system.

System Resources

Windows NT 4.0 has some rudimentary system resource management that is not strictly enforced. When a device driver is using a particular resource (memory range, I/O range, or interrupt) there is a mechanism by which it claims ownership of that resource. Part of that claiming is also describing whether the driver can share the resource with other drivers.

In previous PC operating systems (e.g. DOS and Windows 3.x) there was a complete lack of resource management so the PC Card Standard defined resource-management which is handled by Card Services.

In Windows NT Card Services cooperates with the operating system's resource manager to manage the resources for the whole system.

During boot time the Resources Pool for Card Services has to be initialized. In Windows NT, APSoft's Card Services does this during boot/load. Resources already claimed by drivers are marked as in use and missing COM- and LPT ports are detected so that the appropriate resources are added to Card Services. Finally, all resources listed in the Resources sub-key of the Card Services registry entries are added to the resources pool of Card Services.

Keep in mind that PC Cards can only be configured successfully if the required resources are available to Card Services. That means the resource must be in the pool and may not be used by another PC Card device.

Card Configuration

When you insert a PC Card into your system, CardWare reads the Card Information Structure (CIS) from the card. The CIS normally supplies information about the type of card (e.g., LAN or modem), and how the card can be configured.

In the next step CardWare searches the PCMCIA.SYS database for a specific record of this card. If such an entry exists then the card is configured to match. If no such entry exists then CardWare checks its own internal database for a specific record for this card. If such an entry exists, CardWare tries to configure the card with one of the associated Device Definitions.

If no record exists, CardWare checks whether the card is Generic. If yes, CardWare tries to configure the card with one of the associated generic Device Definitions. If this fails CardWare tries to Auto Configure the card.

If the card is neither Specific nor Generic, CardWare tries to Auto-Configure the card. In this case CardWare tries to configure the card with one of the Device Definitions defined (but not associated). If this fails, CardWare tries to create a Device Definition on the fly and to configure the card with this device.

If a PC Card can not be configured it may be due to the following reasons:

1. The required resources are not available to Card Services.
2. The required resources are in use by another PC Card.
3. The PC Card does not support the configuration which you are trying to use.

For more information about the operations of CardWare for Windows NT, refer to CardWare for Microsoft Windows NT 4.0 User's Guide and CardWare for Microsoft Windows NT 4.0 Developer's Guide.

How CardWare Identifies a PC Card

CardWare uses the following information for identifying a PC Card:

Tuple	Description
CISTPL_VERS_1	Product information string - Name of Manufacturer - Name of Product
CISTPL_MANFID (if available)	Manufacturer Identification tuple
CISTPL_FUNCID (if available)	Function Identification tuple

If the CIS contains at least one valid chain, any subset of the above tuples is enough for card identification.

If the CIS is invalid, CISTPL_VERS_1 is mandatory.

This page intentionally blank.

CHAPTER 5

File Descriptions

The files included with the APSoft PC Card SDK are of three general types:

1. Test programs: These return detailed information about your system and any installed PC Cards.

2. Windows help files: Updated to the latest PC Card Standard, these provide immediate, on-line help while developing under Windows.

3. Sample clients: The sample clients are source code components ready for building client applications and interfaces.

This chapter describes all the included files.

Test Programs

These programs test a given subsystem both for compliance with the latest industry specifications and for the detailed information available from that subsystem. Each section (following) includes checkboxes that tell at a glance the compatible operating system(s) for each program.

CIS Test: CISINFO.EXE

DOS	NT 4
√	√

This program interprets the Card Information structure (CIS). Special options /CON, /DEV and /PAR allow you to collect information about configuration, devices or partitions instead of interpretation of all tuples.

Syntax: **CISINFO [tuplename(s)] [/H|?] [/S:n|<filename>] [/P|F:<filename>] [/B:<filename>] [/L:n] [/A] [/D] [/M] [/N] [/E] [/E-] [/E+] [/CON] [/DEV] [/PAR] [/FUL]**

where:	Tuplename(s)	Interpret only specified tuple name(s). Use tuple identifiers of PCMCIA without prefix CISTPL_. Attention: if specified, the tuplename(s) must precede all other options!
	/H	Print this help text.
	/S:n	Get CIS from card in socket n (zero-based, default 0).
	/S:<filename>	Get CIS from binary file <filename>.If file extension is omitted, extension .BIN is used.
	/CF:n	Get CS of card function n (defaults to all functions)
	/P	Redirect output to printer PRN.
	/F:<filename>	Redirect output to file <filename>. If file extension is omitted, extension .CIS is used.
	/B:<filename>	Write a binary image of CIS to file <filename>. If file extension is omitted, extension .BIN is used.
	/L:n	Lines per page (default: display - 24, other - 66). Specify 0 to suppress paging.
	/A	Additional page break after every tuple (respectively after every device, if option /DEV specified or after every partition, if option /PAR specified).
	/D	Display hexadecimal dump of tuples.
	/M	Display tuple map (offset, memory type, layer).
	/N	Display only tuple names (tuple summary).
	/E	Suppress display of tuples where decoding wasn't finished successfully
	/E-	Suppress decode error/warning messages during CIS interpretation.
	/E+	Display summary of all decode error/warning messages after CIS interpretation.
	/CON	Display collection of all configuration information. Options /N, /D, /M are ignored.
	/DEV	Display collection of all information about devices. Options /N, /D, /M are ignored.
	/PAR	Display collection of all partition information. Options /N, /D, /M are ignored.
	/FUL	Display interpretation of CIS (including decode error/warning messages, tuple map and tuple dump) and configuration, devices and partition collections. Options /N, /D, /M, /E are ignored.

The first screen presented by CISINFO provides basic card status information and general socket status information. The second and subsequent screens provide tuple-by-tuple CIS information. Note that each tuple is identified by order and type on the first line. For a listing of the tuple code constants, see CSDEFS.H found in both DOSCL.ZIP and WINCL.ZIP and the Metaformat section of the PC Card Standard.

The /B: option can be used to create a binary image file of the CIS. This file can then be used with PCDIRTY (below).

CIS.BIN

This file contains the binary image of the example CIS described in Chapter "CIS" on page 35. You can use CISINFO to generate a new CIS.BIN binary file that is an image of the actual CIS of the card being tested. You can then use the CIS.BIN with the application PCDIRTY.EXE to emulate a valid CIS for a PC Card that has a damaged or non-compliant CIS (see page 19).

"Dirty" CIS Test: PCDIRTY.EXE

DOS	NT 4
√	

PCDIRTY allows the enabling of PC Cards either with no CIS or an invalid CIS. It does this by supplying Card Services with a valid CIS which it reads from the disk file passed as an argument.

Normally PCDIRTY terminates immediately after card Initialization. However, you can retain PCDIRTY in memory as a TSR. If the card is physically removed from the slot, PCDIRTY is automatically disabled and terminated, if possible.

To enable a "dirty" card, you need a binary image of the correct CIS on the disk. Insert a "dirty" card into a slot, then call PCDIRTY. If you use PCDIRTY with a card which has a valid CIS, you are prompted to insert a card anyway. The card is enabled with the binary read from disk and its own CIS is hidden from Card Services.

Syntax: **PCDIRTY [/H] [/S:n] /B:<filename> [/R]**

where:	/H	Print this text
	/S:n	Initialize card in slot *n* (zero-based, the default is 0)
	/B:<filename>	Get CIS from binary file <filename>. If file extension omitted, default extension .BIN is used.
	/R	Stay resident (default: stop after card Initialization).

CAUTION: ***Currently, PC Dirty does not work if both CS and PC Enable use DPMS. To use PC Dirty, then, disable loading DPMS.EXE in the CONFIG.SYS.***

Controller Test: ADAPTEST.EXE

DOS	NT 4
√	

Detects the PCMCIA controller used in your system and dumps/interprets the register values.

Syntax: **ADAPTEST [/H|?] [/T] [/A:n] [/B:nnnn] [/S:n] [/D] [/F] [/C]**

where:	/H	Print this text
	/L	List all adapters which ADAPTEST is able to recognize (other options are ignored).
	/T	Report type(s) of all PC Card adapter(s) installed on your machine or at address specified with /B option (options /A, /S, /D, /F and /C are ignored)
	/A:n	Adapter number (default: auto-detect). A call of ADAPTEST /T returns valid adapter numbers.
	/B:nnnn	Base I/O address (hexadecimal, default: auto-detect)
	/S:n	Socket number (default 0)
	/D	Dump socket registers without interpreting
	/F	Full information about registers (dump and interpretation)
	/C	Display type of current PC Card adapter before interpreting the respectively dump

Example: **ADAPTEST /F /C**

```
            ADAPTEST
  Version 1.00 - Release <Date>
   Copyright (c) APSoft, 1995

Adapter 00: Toshiba ToPIC adapter
            I/O base address       : 03E0h
            Number of valid sockets: 2

                       Toshiba ToPIC registers, socket 0
                      ===================================

 Register dump:
----------------
       Name          Index Value           Name          Index Value
-----------------------------------------------------------------------
Identification         00h   83h      Memory Win2 Start Low  20h   10h
Status                 01h   EFh      Memory Win2 Start High 21h   00h
Power                  02h   95h      Memory Win2 Stop  Low  22h   10h
Interrupt & Gen.Cont   03h   63h      Memory Win2 Stop  High 23h   00h
Card status change     04h   00h      Memory Win2 Offs  Low  24h   F0h
Card Status ch. int    05h   FFh      Memory Win2 Offs  High 25h   3Fh
Window Enable          06h   60h      Reserved               26h   00h
I/O Window control     07h   88h      Reserved               27h   00h
I/O Win0 Start Low     08h   F8h      Memory Win3 Start Low  28h   10h
I/O Win0 Start High    09h   02h      Memory Win3 Start High 29h   00h
I/O Win0 Stop Low      0Ah   FFh      Memory Win3 Stop  Low  2Ah   10h
I/O Win0 Stop High     0Bh   02h      Memory Win3 Stop  High 2Bh   00h
I/O Win1 Start Low     0Ch   00h      Memory Win3 Offs  Low  2Ch   F0h
I/O Win1 Start High    0Dh   01h      Memory Win3 Offs  High 2Dh   3Fh
I/O Win1 Stop Low      0Eh   00h      Reserved               2Eh   00h
I/O Win1 Stop High     0Fh   01h      Reserved               2Fh   00h
Memory Win0 Start Low  10h   CFh      Memory Win4 Start Low  30h   10h
Memory Win0 Start High 11h   80h      Memory Win4 Start High 31h   00h
Memory Win0 Stop  Low  12h   CFh      Memory Win4 Stop  Low  32h   10h
Memory Win0 Stop  High 13h   40h      Memory Win4 Stop  High 33h   00h
Memory Win0 Offs  Low  14h   31h      Memory Win4 Offs  Low  34h   F0h
Memory Win0 Offs  High 15h   3Fh      Memory Win4 Offs  High 35h   3Fh
Additional control     16h   01h      Reserved               36h   00h
Reserved               17h   0Fh      Reserved               37h   00h
Memory Win1 Start Low  18h   10h      Reserved               38h   00h
Memory Win1 Start High 19h   00h      Reserved               39h   00h
Memory Win1 Stop  Low  1Ah   10h      Reserved               3Ah   00h
Memory Win1 Stop  High 1Bh   00h      Reserved               3Bh   00h
Memory Win1 Offs  Low  1Ch   F0h      Reserved               3Ch   00h
Memory Win1 Offs  High 1Dh   3Fh      Reserved               3Dh   00h
Global control         1Eh   00h      Reserved               3Eh   00h
Reserved               1Fh   00h      Toshiba Hidden         3Fh   00h


 Register interpretation:
--------------------------
Card Detected:            Battery Good, Card is Ready
Socket configured:        I/O Interface, +RESET Inactive (Low)
Card Power:               On (Enabled), Vcc = 5V, Vpp1 = 5V, Vpp2 = 5V
Output:                   Enabled, 5V Interface
```

```
Internal Interrupt level: 15
Card     Interrupt level: 3

MemWin 0 (0CF000-0CFFFF, 16 bits, 1  WS) map 000000 of Common memory (Disabled)
MemWin 1 (010000-010FFF, 08 bits, 0  WS) map 000000 of Common memory (Disabled)
MemWin 2 (010000-010FFF, 08 bits, 0  WS) map 000000 of Common memory (Disabled)
MemWin 3 (010000-010FFF, 08 bits, 0  WS) map 000000 of Common memory (Disabled)
MemWin 4 (010000-010FFF, 08 bits, 0  WS) map 000000 of Common memory (Disabled)
IO Win 1 (02F8-02FF, 08 bits, 1 WS, Enabled)
IO Win 2 (0100-0100, 08 bits, 1 WS, Disabled)
```

PCI Test: PCITOOL.EXE

DOS	NT 4
√	

This program attempts to detect a PCI BIOS on your system. If one is found, PCITOOL prints information about each PCI device installed on your system.

Syntax: **PCITOOL [/H|?]**

where: /H Prints this text

Power Management Test: PMTOOL.EXE

DOS	NT 4
√	√

This program broadcasts various Power Management notifications and detects the installed power management BIOS and drivers.

Syntax: **PMTOOL [/?] [/STANDBY|SUSPEND|RESUME|CRITRES|BATLOW|INFO]**
or: PMTOOL [/?] [/1|2|3|4|5|0]

where:	/?	Print this help text
	/STANDBY (1)	Broadcast Standby notification
	/SUSPEND (2)	Broadcast Suspend notification
	/RESUME (3)	Broadcast Normal Resume notification
	/CRITRES (4)	Broadcast Critical Resume notification
	/BATLOW (5)	Broadcast Battery Low notification
	/INFO (0)	Detect installed Power management BIOS and drivers

Called without parameters, PMTOOL returns Power Management information.

PCMCIA.SYS Test: PCMTEST.EXE

DOS	NT 4
	√

This program tests the Windows NT PCMCIA.SYS driver. Via command prompt calls of PCMCIA.SYS API functions can be performed.

Command ? shows list of available commands.

Syntax: **PCMTOOL [/H|?] [/F:file]**

where:	/H	Print this text (other options are ignored)
	/F:file	Duplicate output of all buffers returned by PCMCIA.SYS driver to specified file

Plug and Play Test: PNPTOOL.EXE

DOS	NT 4
√	

This program is a tool to show Plug and Play (PnP) support currently present on your system.

By default, PNPTOOL shows the following PnP information:

1. The "PnP Installation Structure"
2. Information about installed PnP ISA devices
3. PnP Docking Station information
4. PnP Configuration Manager / Configuration Access information

Using option /OD, you can carry out the "Online Docking Station test" (disabled by default). This test shows event notification and docking status.

Syntax: **PNPTOOL [/H|?] [/IS:on|off] [/IT:on|off] [/DT:on|off][/CM:on|off] [/OD]**

where:	/H	Print this text (other options are ignored)
	/IS:off	Don't print "PnP Installation Structure"
	/IT:off	Don't print information about PnP ISA devices
	/DT:off	Don't print Docking Station information
	/CM:off	Don't print Config Manager / Config Access information

	/DV:n	Print device n managed by Configuration Manager (default: print all devices managed by CM) Note: option /DV are ignored, if /CM:off is used!
	/OD	Carry out "Online Docking Station test" (output of PnP information disabled by default, but can be enabled using /IS:on, /IT:on, /DT:on and/or /CM:on)

Storage cards test script: STTEST.CMD

DOS	NT 4
	√

STTEST script is test for stability of storage cards (ATA, SRAM, FLASH) in CardWare environment. This script is for Windows NT only.

Please feel your card with some files before start SSTEST. Please place all test files in the root of the card. (Do not create subdirectories).

At beginning of the test SSTEST creates a reference copy card's data in the subdirectory .\org. Then SSTEST copies all files from the card to the hard disk subdirectory .\test and compare them with reference copy. If no there is no differences, SSTEST will invoke CHKDSK. If no problem found, SSTEST will reset the volume (in order to clean NT cache) and repeat the operation specified number of times. Script will stop at the first error.

In additional script may physically power card off and reconfigure it between tests (P Mode). Please pay attention that PMTOOL.EXE should be in the path or in the current directory for operate in P-mode.

Comparison data and timing of every iteration will be logged.

Syntax: **STTEST Socket Drive [Loop1] [Mode] [LogFile] [Loop2]**

where:	/H, /?	Print help text (other options are ignored)
	Socket	Mandatory. Socket number where card is inserted. (Note: Socket numbering is 0 based)
	Drive	Mandatory. Drive letter assigned by CardWare (w/ colon)
	Loop1	Optional. Number of iterations to execute (default 1)
	Mode	Optional. Reset mode. Can be either: **V** - Volume change (Default). In this mode STTEST forces volume change between operations. **P** - Power down. In this mode STTEST power card

		off and reconfigure it between operations. For operate in this mode, PMTOOL should be in your path or in current directory
	LogFile	Optional. Default: log.txt in current directory
	Loop2	Optional. Number of loops to execute during one iteration. Default: 10

Example 1: STTEST 2 G: 500

- Execute 500 iteration x 10 loops (5000 tests) on card in socket 2 (drive letter G:). Write result in log.txt.

Example 2: STTEST 0 H: 500 P c:\logfile.txt 7

- Execute 500 iteration x 7 loops (3500 tests) on card in socket 0 (drive letter H:). Write result in logfile c:\logfile.txt. Power card down between tests

Hints:

1. You should check timing required for one operations before start real testing. Please execute one iteration and check time in log.
2. You may want to turn beeps off (PCTRAY, right button menu) before start operations in P-mode.
3. Parameters have to be specified in order. If e.g. you specify Mode parameter you cannot omit Loop1
4. The P-mode test is about three times slower as V-mode test.
5. When you run P-Mode over SRAM or FLASH card CardWare may display error message box "CardWare was unable to retrieve status of socket state" on artificial insertion. This error is due to the fact that PCDISK executes request of file system and can be safely ignored. CardWare versions prior to 6.00.018 will display above message on virtually every cycle.

File Compare/Copy Utility: CMPTOOL.EXE

DOS	NT 4
	√

This is simplified version of file copy/comparison utility especially designed for usage in test scripts. In copy mode the utility copies source file to destination. In comparison mode the utility compares two files and exits after the first mismatched byte or if the length of files mismatch.

Main advantage of this utility compare to system utilities is clear definition of returned ErrorLevel for batch mode as well as possibility to time execution and collect data for following statistical analyse.

Attention! Wildcards are not supported!

Syntax: **CMPTOOL [/H] [/L] [/C|M] [/T] [/E] [/D:file] [/N] [/S] file1 file2**

where:	/H, /?	Print help text (other options are ignored)
	/L	Print list of defined ErrorLevels
	/C	Copy mode
	/M	Compare mode (default)
	/T	Timed mode: Print start time, end time and duration
	/E	Show errors & warnings from System EventLog during operation
	/D:file	Output in comma-delimited format to file
	/N	No logo output
	/S	Silent execution: no screen output
	file1	First file to compare or copy mode source
	file2	Second file to compare or copy mode destination

Note: Comma-delimited data are added to file. I.e. file is never overriden. Comma-delimited data are printed in following format:

Operat, ErrorLevel, Duration, StartTime, EndTime, file1, file2, Errs

where:	Operat	0, for copy; 1 for compare
	Duration	in seconds
	Errs	Error and Warning events from system Event Log

Network communication test script: NETTEST.CMD

Network communication script tests file copy operation from and to network.

This script is for Windows NT only.

Test will copy files from network source subdirectory to local drive and back to another location of the network. In addition test will compare files in destination directory after copying and log timing of every iteration.

Syntax: **NETTEST [Source] [Local] [Remote] [LogFile] [Loops]**

where:	Source	Source folder on network
	Local	Local folder on test computer
	Remote	Remote folder on network
	LogFile	Name of Logfile (w/o extension)
	Loops	Number of test loops

Hints:

1. Since only root directory files are copied and compared. You should avoid copying of subdirectories.

2. Parameters have to be specified in order. If e.g. you specify LogFile parameter you cannot omit Remote.

3. By default following values are used:

Source	= t:\test1
Local	= d:\aap\Temp
Remote	= t:\Test
LogFile	= nettest
Loops	= 20

Windows Help Files

You can activate SDK HELP files in at least four different ways during any Windows session:

1. Run from Start | Programs | CardWare SDK | Help group.
2. Double-click on the file name.
3. Use the File | Run menu in the Program Manager. Use Browse to locate the desired help file and simply run it. Windows automatically associates it with the Windows help utility, WINHELP.EXE.
4. From within any Windows application with a standard Help function. Simply start Help (typically by pressing F1) and select Contents. Next, select File | Open and locate the help file you wish to use.

ADAPTERS.HLP	Online reference for most popular host socket adapters. Lists registers and offers programming tips.
ATAHELP.HLP	Help for ATA flashdisk and harddisks.
AIMS.HLP	Special help file for Auto Indexing Mass storage devices.
CSHELP.HLP	Help for Card Services. It can be used as a Card Services reference. Each service has a description of its purpose along with its parameters and return values.
MFHELP.HLP	Help for the card Metaformat along with a description of the purpose behind the Metaformat itself.
SSHELP.HLP	Help for Socket Services. It can be used as a Socket Services reference.
XIPHELP.HLP	Help file for Execute-In-Place.

Sample Card Services Clients

The SDK contains five complete sample application/driver skeletons for Card Services clients: one for DOS, one for Microsoft Windows 3.x (Win31), and three for Microsoft Windows NT 4.0.

Each client skeleton was created using Microsoft Visual C++ 4.3 and the Microsoft Foundation Class (MFC) library in the medium memory model. All build files are included.

DOS Driver Sample

The DOS directory contains a sample Card-Services Client with source code for DOS, including the following files:

1. DOSCL.CPP
2. DOSCL.EXE
3. DOSCL.MAK
4. DOSCLCS.CPP
5. DOSCLCS.H
6. MAKEFILE
7. README.TXT

WIN31 Sample

The WIN31 directory contains a sample Card Services Client with source code for Windows 3.x (Win31), including the following files:

1. RES\WINCL.ICO
2. RES\WINCL.RC2
3. MAINFRM.CPP
4. MAINFRM.H
5. MAKEFILE
6. README.TXT
7. RESOURCE.H
8. STDAFX.CPP
9. STDAFX.H
10. WINCL.CPP
11. WINCL.DEF
12. WINCL.EXE
13. WINCL.H
14. WINCL.MAK
15. WINCL.RC
16. WINCLCS.CPP
17. WINCLCS.H
18. WINCLDOC.CPP
19. WINCLDOC.H
20. WINCLVW.CPP

21. WINCLVW.H

Windows NT 4.0 Samples

There are three complete sample application/driver skeletons for Windows NT Card Services clients: one each for console mode utilities, kernel mode device drivers, and UI applications.

NT Kernel Mode Driver Sample

The NT\Driver directory contains a sample Card-Services Client with source code for a kernel mode NT driver, including the following files:

1. BLD.CMD
2. MAKEFILE
3. NTDEMOCL.C
4. NTDEMOCL.INI
5. NTDEMOCL.SYS
6. README.TXT
7. SOURCES

NT Kernel Mode Driver2 Sample

This is an extended version of the Driver sample.

In addition to the Driver sample, this sample checks for a specific card and, if inserted, demonstrates how to retrieve resources assigned to the card. The driver also properly process GetClientInfo request returning the valid info.

For adapt code of the driver to your card, please modify manufacturer name and product name in head of the source file

The NT\Driver2 directory contains the following files:

1. BLD.CMD
2. MAKEFILE
3. NTDEMOCL.C
4. NTDEMOCL.INI
5. NTDEMOCL.SYS
6. README.TXT
7. SOURCES

NT UI Mode Utility Sample

The NT\GUI directory contains a sample Card-Services Client with source code for an NT user-interface (i.e., graphic) mode utility/application, including the following files:

1. RES\CS_GUI.ICO
2. RES\CS_GUI.RC2

3. CALLBACK.CPP
4. CALLBACK.H
5. CS_GUI.CPP
6. CS_GUI.EXE
7. CS_GUI.H
8. CS_GUI.MAK
9. CS_GUI.RC
10. CS_GUIDLG.CPP
11. CS_GUIDLG.H
12. MAKEFILE
13. README.TXT
14. RESOURCE.H
15. STDAFX.CPP

STDAFX.H

Windows 98/ME, NT 4.0, 2000/XP Samples

Console Mode Utility Sample

The Common\Console directory contains a sample Card-Services Client with source code for console mode (i.e., command line) utility/application, including the following files:

1. CS_CON.C
2. CS_CON.EXE
3. CS_CON.MAK
4. MAKEFILE

Bulk Sample

This is future development of the Console sample.

Additionally to the Console sample this application watch for insertion of memory cards and display first 512 bytes of the first memory region. Please notice, that term "Memory card" in this context correspond to the card with physical or virtual memory region (i.e. includes ATA, FLASH or SRAM).

Please also notice, that application will only work if the MTD driver for corresponding memory technology is load.

The application demonstrates techniques for region enumeration and access of Bulk memory services. The Common\BULK directory contains the following files:

1. CS_CON.C
2. CS_CON.EXE
3. CS_CON.MAK
4. MAKEFILE

Bulk2 Sample

This is future development of the Bulk sample.

Additionally to the Bulk sample this application writes small pattern to the offset 0 of the memory (including ATA, FLASH or SRAM) card, read and display modified region, write the original contents back, read back and compare. Although the whole test is non-destructive, please use with care. If any of functions fail, your data may be destroyed. The Common\BULK2 directory contains the following files:

1. BULK2.CPP
2. BULK2.EXE
3. BULKDLG.CPP
4. CLIENT.CPP
5. BULK2.H
6. BULKDLG.H
7. RESOURCE.H
8. STDAFX.H
9. BULK2.RC
10. BULK2.ICO
11. MAKEFILE

Windows DLL

The CWDLL16.DLL in the WIN16 sample directory allows calls to Card Services from Windows 3.x applications.

CWDLL16.DLL contains the code which transfers calls to Card Services from WINDOWS applications, running in Standard or Enhanced mode, to the Card Services Driver running in V86 mode.

The DLL consists of one function called DoCardServices.

Syntax: **DoCardServices(Service, Handle, Pointer, ArgLength, ArgPointer)**

The service names are listed in CSDEFS.H.

The syntax is the same as described in the “Programming Interface” section of the PC Card Card Services Specification R5.0 with the following extension to the Card Services function ‘RequestWindow’ (21H):

The corresponding CWDLL function name for ‘RequestWindow’ is CS_REQUESTMEM.

The ArgPointer (above) points to the argument list. The ‘Request Window” service includes an ‘Attribute’ argument of which Bit 7 is normally reserved as binding specific. The ‘CS_REQUESTMEM’ now uses this bit to distinguish between **physical to logical** address translation and **no** address translation. If Bit 7 of the attribute byte is NOT set (reset to 0) we return the segment without Translation.

We use this extension because of the nature of switching between protected and real memory modes. Card Services runs in real mode and any address is interpreted as *Segment:Offset*. To transfer data from an application running in protected mode (here addresses are interpreted as *Selector:Offset*) to Card Services, it is necessary to allocate some common memory in the address range below 1 MB. Then data from application memory is copied to this buffer on entry to Card Services and finally data is copied back to the application memory on return.

This allow the necessary translations to be performed for you transparently within the DLL. On entry of CS_REQUESTMEM you may pass a base address (given in bytes). On successful exit you receive the allocated base address already converted into a protected mode address.

When Card Services returns this base address as a real mode address to the DLL, it has to be mapped to a selector which points to the memory location. This is done within the DLL so that you can treat the bits 4..19 of the return value in `CWARGS.dwBase` as the selector.

NOTE: ***Because the segment to which a selector points is limited to 64 KB, it is not possible to allocate a memory window larger than 64KB! If you specify a larger value in `dwSize` this call returns `BAD_SIZE`.***

The PC Card specification contains no definition how to map segments to selectors.

Example: You intend to allocate a memory window at the system base address D0000h.

On entry you pass to Card Services:	`dwBase =  0xD0000l`
On exit you may expect to receive:	`dwBase == 0xD0000l`

But you receive something completely different, for example something like `dwBase == 0x10470l`

To create a pointer which in fact points to the memory window allocated at the system memory location D000:0000 you have to convert `dwBase` to a far pointer variable: `FPTR lpBase = MK_FP((WORD)(dwBase >> 4), 0)`; Now `lpBase` points to the start of your allocated memory window

All this low level operations are done automatically within the library on a call to `DoCardServices` by default, and thus are hidden for you.

The only thing you have to keep in mind is that addresses supplied by Card Services are real mode addresses *(Segment:Offset)* and have to be mapped to protected mode addresses *(Selector:Offset)* in your application as described for `CS_REQUESTMEM` whenever you intend to deal with Card Services without the

DLL. See earlier in this chapter for information about how to enable/disable the address translation of CWDLLS.

You may look into the DPMI specification for more information concerning allocation of selectors which map to a given segment.

C H A P T E R 6

An Example CIS

Structure of the CIS

The APSoft PC Card SDK and the CardWare products strongly emphasize the importance of the Card Information Structure (CIS). The CIS should contain the data from which software can identify the function and capabilities of a PC Card. In fact, the PC Card Standard requires all cards to have a CIS and includes a definition of the *Card Metaformat*, a set of rules determining the format of a CIS as a whole as well as the individual data structures that comprise a CIS.

A CIS is comprised of a variable-length chain (or linked list) of data blocks, called *tuples* (pronounced *too'pulls*). They contain information about the type of card, its resource needs, and manufacturer-specific data. It is the design and use of the CIS that opens a path to interoperability and true Plug and Play.

Memory Architecture

PC Cards have two memory address spaces, *Attribute Memory* and *Common Memory*, and two physical entities, *Attribute Memory Plane* and *Common Memory Plane*.

Common Memory Plane. A *physical* entity on a PC Card.

Common Memory Space. A *logical* term to identify the physical Common Memory Plane.

Attribute Memory Plane. A *physical* entity on a PC Card that can be accessed by asserting the -REG pin on the PCMCIA interface.

Attribute Memory Space. A *logical* entity that can exist in either the Attribute Memory Plane or the Common Memory Plane. The CIS starts at address zero (0) of the Attribute Memory Space.

It is important to distinguish between Attribute-Memory *Space* and Attribute Memory *Plane*. **All** cards have an Attribute Memory Space (logical), while **some** cards do not have an Attribute Memory Plane (physical).

The PC Card Standard allows storage of card attribute information or descriptions in both Attribute-Memory space and Common-Memory space.

PC Cards that do ***NOT*** have a physical Attribute Memory Plane are often called *aliased* (because the attribute memory space is aliased in the Common Memory Plane). In such cards, the Attribute Memory Space (logical) resides in the Common Memory Plane.

Remember that the CIS starts at address zero (0) of the Attribute Memory Space . This address could be in the Attribute Memory Plane or in the Common Memory Plane. Because there is no way for software to know if a card is aliased or not, only even bytes can be used to store tuple data.

If, for space reasons, a card manufacturer wants to store portions of the CIS in the Common Memory Space (packed into ascending bytes), the CIS must include a Long-Link-to-Common-Memory tuple. On an aliased card this technique can allow much of the CIS to be packed into consecutive bytes, instead of even bytes only. Note that for aliased cards, the target address of the long-link tuple is non-zero, and the Common-Memory CIS is stored immediately following the Attribute-Memory CIS.

For further information see the *Card Metaformat* volume of the PC Card Standard.

Types of Tuples

The PC Card Standard defines approximately 30 different tuples, which can be grouped in four layers:

1. Basic Compatibility Layer. This layer defines the minimal requirements for compatibility by including the hardware's functions and organization.
2. Data Recording Format Layer. This describes a memory card's format.
3. Data Organization Layer. This is a layer which describes the type of information residing on the memory card.
4. System Specific Layer. This layer describes system-specific data.

When a PC Card is inserted, an interrupt is generated. With CardWare, Card Services detects the interrupt. It parses the CIS tuple structure and dispatches a card insertion notification for all registered clients. One of the clients is PCEnable. It performs needed operations for card and system configuration. An enabler such as PCEnable depends on Card Services to manage the dynamic pool of system resources so it can correctly configure the card with available resources.

But, before any resources are allocated, the CIS must be parsed. The remainder of this chapter presents an example Basic Compatibility Layer CIS, based on the following Fax/Modem specifications:

1. 9600 baud Fax
2. 2400 baud data modem
3. 16450 UART
4. All possible character, stop and start bit configurations
5. Support for US data communications
6. V.22bis, Bell 212A, V.22A&B, V.21, Bell 103, V.29, MNP1-5, V.42, V.42bis, V.27ter

An example CIS binary file, CIS.BIN, comes on the SDK distribution disk. See page 19 for information about CIS.BIN.

Basic Compatibility Layer Tuples

This example covers the following tuples for a generic 9600 Fax/2400 Modem.

Data Code	Tuple Name	Description
01h	CISTPL_DEVICE	Device Information
15h	CISTPL_VERS_1	Level 1 Ver / Product Info
20h	CISTPL_MANFID	Manufacturer ID
21h	CISTPL_FUNCID	Function ID
22h	CISTPL_FUNCE	Function ID Extensions
1Ah	CISTPL_CONFIG	Configurable Table Start
1Bh	CISTPL_CFTABLE_ENTRY	Configuration Table Entry
14h	CISTPL_NO_LINK	No Link
FFh	CISTPL_END	End of Chain

Table 1 Tuple Codes for Fax/Modem Card Example

Device Information Tuple (01h)

The device-information tuple for Common-Memory must be the first tuple in Attribute-Memory. Besides being the first tuple, the Device ID tuple is also the most confusing of the required tuples for I/O cards.

Data Code	Tuple Name	Description
01h	CISTPL_DEVICE	Device information tuple
03h	TPL_LINK	Link to next tuple
00h	DEVICE INFO1 (00 = DTYPE_NULL)	No device in common memory (optional)
00h		Empty block
FFh		End of device info field

Table 2 Device Information Tuple

As shown in Table 2, the device information tuple contains a minimum of five bytes. The first two bytes contain the tuple code and the link to the next tuple in the list. The link is calculated starting with the next byte of the tuple after the link and ending at the last byte of the tuple. So for this example the value placed in the TPL_LINK field is 03h.

The next two bytes contain the common memory device information for the first memory device. In the fax/modem card there are no memory devices.

Each memory device requires 2 bytes of device information. The two bytes contain the device type code, if a Write protect switch is included, what speed the device is, and how much address space it needs. A value of 00h is placed in the first device ID byte because there is no device in common memory (in this fax/modem example), so a null device type is selected.

The device size which is the second byte of the device information field is a 00h because that is the smallest amount allowed. Since the previous byte was DTYPE_NULL, this meaning is undefined for this byte. Conceptually, any value could be placed here -- in practice a zero is the only value which makes any sense. With the value of 00h, a window of 1 block of 512 bytes is required. Unfortunately, the byte does not allow for no memory, so you should specify the smallest one possible. The last byte is an FFh which marks the end of the device information tuple.

Level 1 Version/Product Information Tuple (15h)

The Level 1 version/product information tuple is used to indicate the level of the card and also the card's manufacturer. The PC Card specification describes two version levels, however, this tuple is used for Level 1 tuples only.

Data Code	Tuple Name	Description
15h	CISTPL_VERS_1	Level 1 Version / Product Information Tuple
1Eh	TPL_LINK	Link to Next Tuple

04h	TPLLV1_MAJOR	Major Revision Number
01h	TPLLV1_MINOR	Minor Revision Number
50h	TPLLV1_INFO	P
43h		C
4Dh		M
43h		C
49h		I
41h		A
00h		Terminator
46h		F
61h		a
78h		x
20h		space
43h		C
61h		a
72h		r
64h		d
00h		Terminator
46h		F
58h		X
33h		3
32h		2
31h		1
30h		0
00h		Terminator
41h		A
2Dh		-
30h		0
00h		Terminator
FFh		End of Tuple

Table 3 Version Product Information Tuple

The TPLLV1_MAJOR and _MINOR fields are defined in the PCMCIA specification as having a value of 04h for _MAJOR and 01h for _MINOR in release 2.0. This value should not be changed for your configuration.

The TPLLV1_INFO field is used to input the manufacturer, the product name and revision number. The specification is not specific about the exact format for this, however, the example above should be followed including the use of terminators and including the end of list (FFh) field at the end. Incidentally, the end of list field is not a requirement for all tuples, but it is required for this one. The values placed in this field are the ASCII hex values of the letters indicating the product name, revision number (A-0) and manufacturer (PC Card).

Manufacturer's ID Tuple (20h)

This tuple was added to provide the ability for software to determine the origin and manufacturer of the PC Card as well as the card type without needing to compare ASCII strings.

Data Code	Tuple Name	Description
20h	CISTPL_MANFID	Manufacturing ID Tuple Device Information Tuple
04h	TPL_LINK	Link to Next Tuple
xxh	TPLMID_MANF	PCMCIA code (LSB)
xxh		PCMCIA code (MSB)
xxh	TPLMID_CARD	Company Specific Info
xxh		Company Specific Info

Table 4 Manufacturer's ID Tuple

The TPLMID_MANF field is the unique PC Card code that is developed from the JEDEC Device ID code indicating the manufacturer. If a manufacturer has a JEDEC code, then the 16 bit PC Card ID has a 00h in the upper byte, and the code in the lower byte. If a PCMCIA member manufacturer does not have a JEDEC code, PCMCIA can assign a company specific PC Card ID.

The TPLMID_CARD field designates the card type. These two bytes are manufacturer specific identifiers. Each manufacturer develops its own coding scheme for its card family.

Function ID Tuple (21h)

The purpose of this tuple is to provide information regarding the function of the card and system Initialization information.

Data Code	Tuple Name	Description
21h	CISTPL_FUNCID	Function ID Tuple
02h	TPL_LINK	Link to Next Tuple
02h	TPLDFID_FUNCTION	Serial I/O Fax/Modem
01h	TPLFID_SYSINIT	Configure at Post

Table 5 Function ID Tuple

The TPLDFID_FUNCTION field specifies the function, or multiple functions, of the card. The example fax/modem card performs as a serial device for both modem and fax functions and thus has the value of 02h.

These are the current values:

0	More-than-one-function (Vend. Spec. Combo)
1	Memory
2	Serial Port (both modem and fax cards)
3	Parallel Port (parallel printer port both uni and bi directional)
4	Fixed Disk
5	Video Adapter
6	LAN Adapter
7	AIMs
8-ffh	Reserved

The TPLFID_SYSINIT field of the tuple indicates to the host when the card should be initialized. Only two values are defined today; a 01h indicates that the card can be initialized during POST, and a value of 02h indicates that the card contains expansion ROM which should be installed during system installation. Our example card does not contain ROM and consequently should have the value of 01h.

Modem Function Extension Tuple (22h)

The Modem Function Extension tuples document the capabilities of the fax or modem before the communications package tries to use it.

Data Code	Tuple Name	Description
22h	CISTPL_FUNCE	Serial Port ID
04h	TPL_LINK	Link To Next Tuple
00h	TPLFE_TYPE	Serial Port
01h	TPLFE_UA	16450 UART
0Fh	TPLFE_UC	All Parity Supported
7Fh		All Stops and Characters Supported
22h	CISTPL_FUNCE	Modem Function Extension
09h	TPL_LINK	Link to Next Tuple
01h	TPLFE_TYPE	Modem as Discrete Device
1Fh	TPLFE_FC	All Flow Control Methods
09h	TPLFE_CB	40 Character DCE Command Buffer
C8h	TPLFE_EB	200 Character DCE to DCE Buffer, LSB of LSW
00h		DCE to DCE Buffer MSB of LSW
00h		DCE to DCE LSB of MSW

Data Code	Tuple Name	Description
C8h	TPLFE_TB	200 Character DTE to DCE Buffer, LSB of LSW
00h		DTE to DCE MSB of LSW
00h		DTE to DCE LSB of MSW
22h	CISTPL_FUNCE	Data Modem Extension
0Ch	TPL_LINK	Link to Next Tuple
02h	TPLFE_TYPE	Modem
00h	TPLFE_UD	DTE to UART Max Data Rate MSB
80h		DTE to UART Max Data LSB (9600)
3Bh	TPLFE_MS	Modulation V.22bis,Bell 212A, V.22A and B, V.21, Bell 103
00h		Not Supported
03h	TPLFE_EM	Error Correction using MNP 2-4 or V.42/LAPM
03h	TPLFE_DC	Data Compression using MNP5 or V.42bis
08h	TPLFE_CM	Command Protocol is MNP AT
07h	TPLFE_EX	Escape Mechanism is User Defined, +++, Break
00h	TPLFE_DY	No Data Encryption
00h	TPLFE_EF	No Caller ID
B5h	TPLFE_CD	CCITT Country Code (USA)
22h	CISTPL_FUNCE	Fax Extension
08h	TPL_LINK	Link to Next Tuple
13h	TPLFE_TYPE	Fax Class 1
00h	TPLFE_UD	DTE to UART Max Data Rate MSB
80h		DTE to UART Max Data Rate LSB (9600)
06h	TPLFE_FM	CCITT Modulation Supported V.29, V.27ter
00h	TPLFE_FY	Reserved for Future Encryption Definition
22h	TPLFE_FS	Fax features Polling and T.4
00h		Reserved

Data Code	Tuple Name	Description
B5h	TPLFE_CF	CCITT Country Code USA

Table 6 Function Extension Tuples

Using these tuples allows the host and the application software, which is CIS aware, to do away with the current set of questions that must be asked when configuring the operating environment.

The fax/modem tuple list contains three different function extension tuples. These are used to describe the three distinct areas of functionality of fax/modem products. The current extension tuple has been designed to accommodate one more function (not included in this example card), Voice Encoding. The tuples start with the simplest function, a UART or Universal Asynchronous Receiver Transmitter. The serial port ID tuple is a 5 byte tuple. The functions that are included are the determination of the UART type, the start and stop characters supported, and the parity types.

There are three types of UART’s used for serial communication, they are the 8250, 16450 and the 16550. The fourth and fifth fields deal with the actual UART capabilities. The standard 16450 UART is capable of handling all data sizes (5, 6, 7, 8), stop characters (1, 1.5, 2) and parity types (even odd, mark, or space). The 16450 is the UART used in this example.

The Modem Interface tuple, which starts with the second CISTPL_FUNCE code, details the functionality of the modem . This tuple string describes the modem characteristics. This is a nine byte tuple, starting with the standard code and link bytes. The next byte deals with the flow control methods of the modem. Flow control is the handshaking between two modems and is used to decide if the connection can support the sending/receiving of new information. A byte value of 1Fh corresponds to the support of all flow control methods including hardware and software flow control. This tuple also describes the chip set feature of a 40 character DCE Command Buffer, and 2 character buffers. The 40 character command buffer gives the tuple value of 09h because the formula used for calculation is to take the buffer size, divide by 4, then subtract a 1 (size = n, n/4-1). On our example card, the two character buffers are 200 character DCE to DCE and DTE to DCE buffers.

The Data Modem tuple is the third of the five CISTPL_FUNCE code tuples. This tuple describes the data rates, modulation schemes, error correction and detection, command protocols, and the CCITT Country code. This tuple is 13 bytes long, starting with the standard tuple code and link. The values for this string of data comes straight off of the modem chip set (internal to the card) data sheets, consequently when developing the data for this tuple refer to the appropriate data sheets.

The Fax tuple is the final function extension tuple in this section. This tuple is a minimum of nine bytes, but must be replicated for each supported Service class. As with the Data modem tuple, the features described are the maximum data rate, modulation schemes, and country code.

Configuration Tuple (1Ah)

The Configuration tuple, Tuple 1Ah, describes the general configuration characteristics of PCMCIA cards, for example, where the first configuration register is located and how many there are in total.

Data Code	Tuple Name	Description
1Ah	CISTPL_CONFIG	Configurable-Card Tuple
05h	TPL_LINK	Link to Next Tuple
01h	TPCC_SZ	Two bytes of Address
23h	TPCC_LAST	23h (Com4) is Last entry in Configuration Table
00h	TPCC_RADR	Configuration Register Base Address LSB
02h	TPCC_RADR	Configuration Register Base Address MSB
03h	TPCC_RMSK	Two Configuration Registers

Table 7 Configurable Card Tuple

The TPCC_SZ field indicates the number of bytes in the configuration registers base address, and the configuration register presence mask. The values of these registers are detailed in the bytes following the size field.

The TPCC_LAST field gives the value that corresponds to the final entry in the card configuration table.

The TPCC_RADR field indicates the address of the configuration registers in Attribute memory. At least one byte of address is required. If more than one byte is needed, the LSB should be presented first.

The TPCC_RMSK field gives the amount of registers that start at the base address in Attribute memory. Each bit in the field corresponds to the presence or absence of a register. The host would use this information along with the value in the RADR field to determine the location of the card's configuration register set.

Configuration-Entry Tuple (1Bh)

The Configuration Entry tuple, Tuple 1Bh, defines the default configuration and its description. If the fax/modem is expected to be used in the PC Market, the Configuration tuple would need to be repeated for the other three COM ports of the PC.

Data Code	Tuple Name	Description
1Bh	CISTPL_CFTABLE_ENTRY	Configuration Table Entry Tuple
15h	TPL_LINK	Link to Next Tuple
E0h	TPCE_INDX	Config 32 (COM1) Provides defaults for succeeding entries
01h	TPCE_IF	I/O, Wait Supported
9Dh	TPCE_FS	Feature Selection Byte
71h	TPCE_PD	Power Description
55h		V nominal = 5 v
86h		I Avg. = 138mA
26h		Extension
86h		I Peak = 197mA
61h		Extension
64h		I Power Down = 6mA
FCh	TPCE_TD	Wait Scale Only
0Ch		10 uS Max Wait Time
AAh	TPCE_IO	8 Bit I/O 10 Bit I/O Addr
60h		1 Range, I/O Addr = 2 bytes
F8h		Start of I/O Addr (LSB)
03h		Start of Addr (MSB)
07h		Length of Range =8
30h	TPCE_IR	Level Mode, Mask for INT’s
BCh		IRQ7, IRQ5, IRQ4, IRQ3, IRQ2
86h		IRQ15, IRQ10, IRQ9
28h	TPCE_MI	Supports Power Down and Audio

Table 8 Configuration Entry Tuple

As with all tuples, the configuration tuple begins with the tuple code and the link bytes. After the link is the Table Index Byte. A value of E0h defines that this entry is the default entry, that a set of interface descriptions follows, and that a value of 20h must be written into the configuration register to enable the configuration of the card.

The interface description field, feature selection byte field, and power selection byte are straightforward. They describe that the card is an I/O card, that it supports the wait line of the 68 pin interface, and that the following bytes describe the voltage, current, wait scale, address port, power down features and audio features of the card.

The most confusing bytes of information in this section are the Power Description Structure parameter definitions. These are not required, but help the system to get a full description of the card. Since this is confusing, we will step through the description in detail. Tables 9 through 11 are the tables given in the specification that detail the power description structure.

Table 9 shows the power extension fields:

Byte	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	Ext	Mantissa				Exponent		
1	Ext	Extension						

Table 9 Power Extension Fields for Tuple 1Bh

Table 10 shows the exponent of the current and voltage values:

Exponent	**Current Scale**	**Voltage Scale**
0	100nA	10uV
1	1uA	100uV
2	10uA	1mV
3	100uA	10mV
4	1mA	100mV
5	10mA	1V
6	100mA	10V
7	1A	100V

Table 10 Current/Voltage Scales

Table 11 gives the mantissa of the value:

Mantissa	**Value**
0h	1
1h	1.2
2h	1.3

Mantissa	Value
3h	1.5
4h	2
5h	2.5
6h	3
7h	3.5
8h	4
9h	4.5
Ah	5
Bh	5.5
Ch	6
Dh	7
Eh	8
Fh	9

Table 11 Mantissa Values

The nominal voltage is an example of a value not needing an extension. Looking at byte 0, the extension bit is 0, since no extension is needed because the value can be defined with only one byte. The mantissa, which occupies bits 6-3, is an 0Ah. This is seen from Table 11, where a 0Ah represents the numerical value of 5. The exponent, which occupies bits 2-0, is a 5, since the voltage scale is 1V. Grouping all of the bits together gives a byte value of 55h.

The peak current of 197 mA needs an extension byte, byte 1, because the value to be described has three significant digits. As with the voltage, the designer must first look at byte 0. In this case bit 7 is a 1, thus signifying that an extension byte follows. The mantissa is 0 because the value to be described is 1 (see Table 11). Since we are looking at the 100 mA range, the current scale for the exponent is 6. When put together, the byte 0 value is 86h . This takes care of the 100's portion of the value. To get the remaining 97 mA defined, the extension byte is used. The value shown of 61h is the hex value for 97.

Configuration Entry Tuples COM2-COM4

The following table provides the configuration entry tuples for the I/O definition of COM2, COM3 and COM4.

Data Code	Tuple Name	Description
1Bh	CISTPL_CFTABLE_ENTRY	Configuration Table Entry Tuple
07h	TPL_LINK	Link to Next Tuple
21h	TPCE_INDX	Config 33 (COM2)

Data Code	Tuple Name	Description
08h	TPCE_FS	Feature Selection Byte only I/O space description follows
AAh	TPCE_IO	8 Bit I/O 10 Bit I/O Addr
60h		1 Range, I/O Addr = 2 Bytes
F8h		Start Addr (LSB)
02h		Start Addr (MSB)
07h		Length of range
1Bh	CISTPL_CFTABLE_ENTRY	Configuration Table Entry Tuple
07h	TPL_LINK	Link to Next Tuple
22h	TPCE_INDX	Config 34 (COM§)
08h	TPCE_FS	Feature Selection Byte only I/O space description follows
AAh	TPCE_IO	8 Bit I/O 10 Bit I/O Addr
60h		1 Range, I/O Addr = 2 Bytes
E8h		Start Addr (LSB)
03h		Start Addr (MSB)
07h		Length of range
1Bh	CISTPL_CFTABLE_ENTRY	Configuration Table Entry Tuple
07h	TPL_LINK	Link to Next Tuple
23h	TPCE_INDX	Config 35 (COM4)
08h	TPCE_FS	Feature Selection Byte only I/O space description follows
AAh	TPCE_IO	8 Bit I/O 10 Bit I/O Addr
60h		1 Range, I/O Addr = 2 Bytes
E8h		Start Addr (LSB)
02h		Start Addr (MSB)
07h		Length of range

Table 12 Configuration Entry Tuple

No Link Tuple (14h)

The NO_LINK tuple is an optional tuple that helps speed up tuple processing by indicating to the software that no long link tuples are in this chain. Long links often a tuple chain stored in Common, rather than Attribute, memory. Additionally, there are special long links for multifunction cards.

Data Code	Tuple Name	Description
14h	CISTPL_NO_LINK	No Link Tuple
00h	TPL_LINK	Link to Next Tuple

Table 13 No Link Tuple

End of Tuple Chain (FFh)

The End of Tuple Chain is used to indicated the end of the tuple chain and has a value of FFh. This tuple appears at the end of the CIS.

Data Code	Tuple Name	Description
FFh	CISTPL_END	End of Tuple Chain

Table 14 End of Tuple Chain

This page intentionally blank.

APPENDIX A

List of supported PC Card adapters

Please refer to List of Supported Controller Chips.pdf document.