4 ASPI for Win32 Specification

About This Chapter

Read this chapter to find out about

- How to program ASPI for Win32
- The syntax of ASPI for Win32 commands
- How to handle polling and posting of ASPI for Win32 SCSI requests
- How to migrate to ASPI for Win32

▼▼▼▼▼ 4

Introduction

The Advanced SCSI Programming Interface (ASPI) has become an industry standard SCSI software interface in the DOS, Windows, OS/2, and Netware environments. ASPI allows developers to write device drivers and applications which are independent of the specific host adapter platform used to connect SCSI peripherals. With ASPI, there is a single driver managing the host bus adapter, eliminating conflicts when multiple applications try to access the SCSI bus at the same time.

With the arrival of Windows NT and Windows 95, ASPI must be extended to allow applications to take full advantage of these new, 32-bit operating systems. This specification describes the Advanced SCSI Programming Interface (ASPI) for the Win32 environment.

Note: This specification defines the protocol between ASPI client applications and the ASPI Manager *wnaspi32.dll*. The specification does not describe the protocol between *wnaspi32.dll* and the target Win32 environment, since this implementation may vary between operating systems.

Using ASPI for Win32 - An Overview

The ASPI manager for Win32 is implemented as a Dynamic Link Library (DLL) called *wnaspi32.dll*. ASPI function calls can be used to retrieve information about installed SCSI host adapters and devices, and to execute SCSI I/O requests. The ASPI for Win32 layer is fully re-entrant, supporting multitasking and multithreaded operations. To use ASPI for Win32, the following two functions must be imported from *wnaspi32.dll* into the client's Win32 application.

Function	Description
GetASPI32SupportInfo	This function returns the number of SCSI host adapters installed and ensures that the ASPI manager is initialized properly. Clients must call this function once at initialization time before the SendASPI32Command function can be accessed.
SendASPI32Command	This function handles requests for SCSI I/O.

The following SCSI I/O command codes are defined within the **SendASPI32Command**:

Value	Meaning
SC_HA_INQUIRY	Get information about installed host adapters, including the number of installed adapters.
SC_GET_DEV_TYPE	Get information about installed SCSI devices.
SC_EXEC_SCSI_CMD	Execute SCSI I/O.
SC_ABORT_SRB	Abort an outstanding I/O request.
SC_RESET_DEV	Reset an individual SCSI target.
SC_GET_DISK_INFO	Get information on disk type SCSI devices. (Not used with Windows NT)

GetASPI32SupportInfo Function

Prototype: DWORD GetASPI32SupportInfo(VOID)

The **GetASPI32SupportInfo** function returns the number of host adapters installed and ensures that the ASPI manager is initialized properly. This function must be called once at initialization time, before **SendASPI32Command** is accessed. The number of host adapters returned represents the logical bus count, not the true physical adapter count. For host adapters with a single bus, the host adapter count and logical bus count are identical.

This function has no parameters.

The DWORD return value specifies the result of the ASPI request. The DWORD is encoded as follows:

Table 4-1. Return Values from GetASPI32SupportInfo Function

Length	Bit	Meaning	
WORD	31-16	Reserved = 0	
BYTE	15-8	**Status**	
		SS_COMP	ASPI request completed without error
		SS_FAILED_INIT	ASPI manager unable to initialize; ASPI services not available
BYTE	7-0	If Status = SS_COMP	Number of host adapters

Example

This example returns the current status of ASPI for Win32.

```
DWORD ASPIStatus;
BYTE NumAdapters;
HWND hwnd;
            .
            .
ASPIStatus = GetASPI32SupportInfo();
switch ( HIBYTE(LOWORD(ASPIStatus)) )
    {
    case SS_COMP:
        /*
         * ASPI for Win32 is properly initialized
         */
        NumAdapters = LOBYTE(LOWORD(ASPIStatus));
        break;
    default:
        MessageBox( hwnd,
                    "ASPI for Win32 is not initialized!!",
                    NULL,
                    MB_ICONSTOP );
        return FALSE;
    }
        .
        .
```

SendASPI32Command Function

Prototype: DWORD SendASPI32Command(LPSRB)

Declaration: LPSRB *lpSRB*;

The **SendASPI32Command** function handles all SCSI I/O requests. A command code is used to specify the type of I/O requested. **SendASPI32Command** is called with a pointer to a SCSI Request Block (SRB) structure. While SRB definition can vary depending on the ASPI command code, all SRBs include a standard SRB header:

```
typedef struct {

        BYTE    SRB_Cmd;            // ASPI command code
        BYTE    SRB_Status;         // ASPI command status byte
        BYTE    SRB_HaId;           // ASPI host adapter number
        BYTE    SRB_Flags;          // ASPI request flags
        DWORD   SRB_Hdr_Rsvd;       // Reserved, MUST = 0

} SRB_Header;
```

The SRB_Cmd field must contain the command code for the desired SCSI I/O operation. Each of these commands is described in the following sections. Possible values for SRB_Cmd include the following:

SC_HA_INQUIRY
SC_GET_DEV_TYPE
SC_EXEC_SCSI_CMD
SC_ABORT_SRB
SC_RESET_DEV
SC_GET_DISK_INFO

The SC_GET_DISK_INFO command is not used with Windows NT.

Win32 ASPI Commands

Note: In the following sections, fields marked with **[OUT]** are returned by the ASPI manager. Fields marked with **[IN]** are set by the ASPI client before they are sent to the ASPI manager. Fields marked as **Reserved** must be zeroed before the SRB is sent to the ASPI manager.

Host Adapter Inquiry Command

Prototype: DWORD SendASPI32Command(LPSRB)

Declaration: LPSRB *lpSRB*;

The **SendASPI32Command** function with command code SC_HA_INQUIRY is used to get information on the installed host adapter hardware, including the number of host adapters installed. As previously stated, the number of host adapters returned represents the logical bus count instead of the true physical adapter count. For host adapters that support single bus only, the host adapter count and logical bus count are identical. For host adapters that support multiple buses, the host adapter count represents the total logical bus count.

Parameter	Description
lpSRB	Points to the following SCSI request block

```
typedef struct {

        BYTE    SRB_Cmd;              // ASPI command code = SC_HA_INQUIRY
        BYTE    SRB_Status;           // ASPI command status byte
        BYTE    SRB_HaId;             // ASPI host adapter number
        BYTE    SRB_Flags;            // ASPI request flags
        DWORD   SRB_Hdr_Rsvd;         // Reserved, MUST = 0
        BYTE    HA_Count;             // Number of host adapters present
        BYTE    HA_SCSI_ID;           // SCSI ID of host adapter
        BYTE    HA_ManagerId[16];     // String describing the manager
        BYTE    HA_Identifier[16];    // String describing the host adapter
        BYTE    HA_Unique[16];        // Host Adapter Unique parameters
        WORD    HA_Rsvd1;

} SRB_HAInquiry, *PSRB_HAInquiry;
```

Table 4-2. Host Adapter Inquiry Command

Member	Description
SRB_Cmd	[IN] This field must contain SC_HA_INQUIRY.
SRB_Status	[OUT] On return, this field is the same as the return status defined below.
SRB_HaId	[IN] This field specifies which installed host adapter the request is intended for. Host adapter numbers are always assigned by the SCSI manager layer, beginning with zero.
SRB_Flags	[IN] Reserved = 0
SRB_Hdr_Rsvd	Reserved = 0

Table 4-2. Host Adapter Inquiry Command *(Continued)*

Member	Description
HA_Count	[OUT] The ASPI manager sets this field with the number of host adapters installed under ASPI. For example, a return value of 2 indicates that host adapters #0 and #1 are valid. To determine the total number of host adapters in the system, the SRB_HaId field should be set to zero, or GetASPI32SupportInfo can be used. The number of host adapters returned represents the logical bus count instead of the true physical adapter count. For host adapters that support single bus only, the host adapter count and logical bus count are identical.
HA_SCSI_ID	[OUT] The ASPI manager sets this field with the SCSI ID of the given host adapter.
HA_ManagerId[..]	[OUT] The ASPI manager fills this 16-character buffer with an ASCII string describing the ASPI manager. For ASPI for WIN32, the string “ASPI for WIN32“ is returned.
HA_Identifier[..]	[OUT] The ASPI manager fills this 16-character buffer with a string describing the host adapter.
HA_Unique[..]	[OUT] The ASPI manager fills this 16-byte buffer with host adapter unique parameters. The returned buffer is encoded as follows: Byte 15-8 Reserved = 0 Byte 7-4 Maximum Transfer Length. Byte 3 Maximum SCSI Targets. Indicates the maximum number of targets (SCSI IDs) the adapter supports. If this value is not set, it is assumed that there are 8 targets. (SCSI IDs 0-7). Byte 2 Adapter Unique Flags: Bit 7-2 Reserved Bit 1 1 = Residual byte count reporting is supported 0 = Residual byte count reporting is not supported. See *Residual Byte Length* on page 4-7 for more information. Bit 0 Reserved Byte 1-0 Buffer Alignment Mask. The host adapter requires data buffer alignment specified by the 16-bit value. A value of 0x0000 indicates no boundary requirements (e.g. byte), 0x0001 word alignment, 0x0003 double-word, 0x0007 8-byte alignment, etc. The 16-bit value allows data buffer alignments of up to 65536-byte boundaries. Byte 1 is the most significant byte.

Table 4-3. Return Values from Host Adapter Inquiry Command

Value	Meaning
SS_COMP	SCSI/ASPI request has completed without error.
SS_INVALID_HA	Invalid host adapter number.

Residual Byte Length

Residual byte length is the number of bytes not transferred to, or received from, the target SCSI device. For example, if the ASPI buffer length for a SCSI Inquiry command is set for 100 bytes, but the target only returns 36 bytes; the residual length is 64 bytes. If the ASPI buffer length for a SCSI Write command is set for 514 bytes but the target only takes 512 bytes, the residual length is 2 bytes. ASPI modules can determine if the loaded ASPI manager supports residual byte length by issuing an Extended Host Adapter Inquiry command, as described in the previous section.

Example

This example of the Host Adapter Inquiry command gets host adapter hardware information from adapter #0.

```
SRB_HAInquiry MySRB;
DWORD ASPIStatus;
    .
    .
MySRB.SRB_Cmd      = SC_HA_INQUIRY;
MySRB.SRB_HaId     = 0;
MySRB.SRB_Flags    = 0;
MySRB.SRB_Hdr_Rsvd = 0;
ASPIStatus         = SendASPI32Command ( (LPSRB) &MySRB );
    .
    .
```

Get Device Type Command

Prototype: DWORD SendASPI32Command(LPSRB)

Declaration: LPSRB *lpSRB*;

The **SendASPI32Command** function with command code SC_GET_DEV_TYPE enables you to identify the devices available on the SCSI bus. A Win32 tape backup package, for example, can scan each target/LUN on each installed host adapter looking for a device type corresponding to sequential access devices. This eliminates the need for each Win32 application to duplicate the effort of scanning the SCSI bus for devices.

Parameter	Description
lpSRB	Points to the following SCSI request block:

```
typedef struct {

        BYTE    SRB_Cmd;           // ASPI command code = SC_GET_DEV_TYPE
        BYTE    SRB_Status;        // ASPI command status byte
        BYTE    SRB_HaId;          // ASPI host adapter number
        BYTE    SRB_Flags;         // Reserved
        DWORD   SRB_Hdr_Rsvd;      // Reserved
        BYTE    SRB_Target;        // Target's SCSI ID
        BYTE    SRB_Lun;           // Target's LUN number
        BYTE    SRB_DeviceType;    // Target's peripheral device type
        BYTE    SRB_Rsvd1;         // Reserved for alignment

} SRB_GDEVBlock, *PSRB_GDEVBlock;
```

Table 4-4. Get Device Type Command

Member	Description
SRB_Cmd	[IN] This field must contain SC_GET_DEV_TYPE.
SRB_Status	[OUT] On return, this field will be the same as the return status defined below.
SRB_HaId	[IN] This field specifies which installed host adapter the request is intended for. Host adapter numbers are always assigned by the ASPI manager layer beginning with zero.
SRB_Flags	[IN] Reserved = 0
SRB_Hdr_Rsvd	Reserved = 0
SRB_Target	[IN] SCSI ID of target device.
SRB_Lun	[IN] Logical Unit Number (LUN) of device.
SRB_DeviceType	[OUT] The ASPI manager fills this field with the peripheral device type. This is derived from the Peripheral Device Type returned by the SCSI Inquiry command. Refer to the SCSI specification to learn more about the SCSI Inquiry command.

Table 4-5. Return Values from Get Device Type Command

Value	Meaning
SS_COMP	SCSI/ASPI request has completed without error.
SS_INVALID_HA	Invalid host adapter number.
SS_NO_DEVICE	SCSI device not installed.

Example

This example gets the peripheral device type from host adapter #0, target ID #4, and LUN #0.

```
SRB_GDEVBlock MySRB;
DWORD ASPIStatus;
        .
        .
MySRB.SRB_Cmd          = SC_GET_DEV_TYPE;
MySRB.SRB_HaId         = 0;
MySRB.SRB_Flags        = 0;
MySRB.SRB_Hdr_Rsvd     = 0;
MySRB.SRB_Target       = 4;
MySRB.SRB_Lun          = 0;
ASPIStatus             = SendASPI32Command ( (LPSRB) &MySRB );
        .
/*************************************************************/
/* If MySRB.SRB_Status == SS_COMP, MySRB.SRB_DeviceType
/* will contain the peripheral device type.
/*************************************************************/
        .
        .
```

Execute SCSI I/O Command

Prototype: DWORD SendASPI32Command(lpSRB)

Declaration: LPSRB *lpSRB*;

The **SendASPI32Command** function with command code SC_EXEC_SCSI_CMD is used to execute a SCSI I/O command. Once an ASPI client has initialized, virtually all I/O is performed with this command.

Parameter	Description
lpSRB	Points to the following SCSI request block:

```
typedef struct {

        BYTE    SRB_Cmd;                // ASPI command code = SC_EXEC_SCSI_CMD
        BYTE    SRB_Status;             // ASPI command status byte
        BYTE    SRB_HaId;               // ASPI host adapter number
        BYTE    SRB_Flags;              // ASPI request flags
        DWORD   SRB_Hdr_Rsvd;           // Reserved
        BYTE    SRB_Target;             // Target's SCSI ID
        BYTE    SRB_Lun;                // Target's LUN number
        WORD    SRB_Rsvd1;              // Reserved for Alignment
        DWORD   SRB_BufLen;             // Data Allocation Length
        BYTE    *SRB_BufPointer;        // Data Buffer Point
        BYTE    SRB_SenseLen;           // Sense Allocation Length
        BYTE    SRB_CDBLen;             // CDB Length
        BYTE    SRB_HaStat;             // Host Adapter Status
        BYTE    SRB_TargStat;           // Target Status
        void    (*SRB_PostProc)();      // Post routine
        void    *SRB_Rsvd2;             // Reserved
        BYTE    SRB_Rsvd3[16];          // Reserved for expansion
        BYTE    CDBByte[16];            // SCSI CDB
        BYTE    SenseArea[SENSE_LEN+2]; // Request Sense buffer

} SRB_ExecSCSICmd, *PSRB_ExecSCSICmd;
```

Table 4-6. Execute SCSI I/O Command

Member	Description	
SRB_Cmd	[IN] This field must contain SC_EXEC_SCSI_CMD.	
SRB_Status	[OUT]This field returns one of the following statuses:	
	SS_PENDING	SCSI request is in progress.
	SS_COMP	SCSI/ASPI request has completed without error.
	SS_ABORTED	SCSI command has been aborted.
	SS_ABORT_FAIL	SCSI command abort failed.
	SS_ERR	SCSI command completed with an error.
	SS_INVALID_SRB	One or more parameters in the SCSI Request Block (SRB) are set incorrectly.
	SS_INVALID_PATH_ID	SCSI ID and LUN are invalid.
	SS_BUFFER_TO_BIG	The ASPI manager cannot handle the given transfer size. See *Programming Guidelines* on page 4-23 for more information.
	SS_BUFFER_ALIGN	The ASPI manager cannot handle the alignment on this buffer. You must force the buffer into the alignment specified by the alignment mask returned as part of the SC_HA_INQUIRY command.

Table 4-6. Execute SCSI I/O Command *(Continued)*

Member	Description	
	SS_SECURITY_VIOLATION	The application does not have the security privileges needed to perform the operation for the device.
SRB_HaId	[IN] This field specifies which installed host adapter the request is intended for. Host adapter numbers are always assigned by the SCSI manager layer beginning with zero.	
SRB_Flags	[IN] The SRB flags field is defined as follows:	
	SRB_DIR_IN	Data transfer from SCSI target to host. Mutually exclusive with SRB_DIR_OUT.
	SRB_DIR_OUT	Transfer from host to SCSI target. Mutually exclusive with SRB_DIR_IN.
	SRB_EVENT_NOTIFY	Enable ASPI command event notification. See *Request Completion/Notification* on page 4-19 for more details. This flag and the SRB_POSTING flag are mutually exclusive.
	SRB_POSTING	Enable ASPI command completion posting. See *Request Completion/Notification* on page 4-19 for more details. This flag and the SRB_EVENT_NOTIFY flag are mutually exclusive.
	SRB_ENABLE_RESIDUAL COUNT	Enables reporting of residual byte count. This bit is only significant if the host adapter reports support for residual byte count from the SC_HA_INQUIRY command. Whenever a data underrun occurs, the SRB_BufLen field is updated to reflect the remaining bytes to transfer.
SRB_Hdr_Rsvd	Reserved = 0	
SRB_Target	[IN] SCSI ID of target device.	
SRB_Lun	[IN] Logical Unit Number (LUN) of device.	
SRB_BufLen	[IN] This field indicates the number of bytes to be transferred. If the SCSI command to be executed does not transfer data (e.g., Test Unit Ready, Rewind, etc.), this field *must* be set to zero. If residual byte length is supported and selected, this field is returned with the residual number of bytes (usually 0). See *Residual Byte Length* on page 4-7.	
SRB_BufPointer	[IN] This field is a pointer to the data buffer.	
SRB_SenseLen	[IN] This field indicates the number of bytes allocated at the end of the SRB for sense data. A request sense is automatically generated if a check condition is presented at the end of a SCSI command.	
SRB_CDBLen	[IN] This field establishes the length, in bytes, of the SCSI Command Descriptor Block (CDB). This value is typically 6 or 10.	
SRB_HaStat	[OUT] Upon completion of the SCSI command, the ASPI manager sets this field with the host adapter status as follows:	
	HASTAT_OK	Host adapter did not detect an error.
	HASTAT_TIMEOUT	The time allocated for the transaction ran out.
	HASTAT_COMMAND_TIMEOUT	SRB expired while waiting to be processed.
	HASTAT_SEL_TO	Selection of target timed out.
	HASTAT_MESSAGE_REJECT	While processing SRB, the adapter received a MESSAGE REJECT.
	HASTAT_BUS_RESET	A bus reset was detected.
	HASTAT_PARITY_ERROR	A parity error was detected.

Table 4-6. Execute SCSI I/O Command *(Continued)*

Member	Description	
	HASTAT_REQUEST_SENSE_FAILED	The adapter failed in issuing a Request Sense after a check condition was reported by the target device.
	HASTAT_DO_DU	Data overrun/underrun.
	HASTAT_BUS_FREE	Unexpected Bus Free.
	HASTAT_PHASE_ERR	Target Bus phase sequence failure.
SRB_TargStat	[OUT] Upon completion of the SCSI command, the ASPI manager will record the specific status returned by the SCSI target:	
	STATUS_GOOD	No target status.
	STATUS_CHKCOND	Check status (sense data is in SenseArea).
	STATUS_BUSY	Specified Target/LUN is busy.
	STATUS_RESCONF	Reservation conflict.
SRB_PostProc	[IN] If posting is enabled (SRB_POSTING flag), this field contains a pointer to a function. ASPI for Win32 calls this function upon completion of an ASPI request. If event notification is enabled (SRB_EVENT_NOTIFY flag), this field contains a handle to an event. ASPI for Win32 signals this event upon completion of an ASPI request. See *Request Completion/Notification* on page 4-19.	
CDBByte[..]	[IN] This field contains the CDB as defined by the target's SCSI command set. The length of the SCSI CDB is specified in the SRB_CDBLen field.	
SenseArea[..]	[OUT] The SenseArea is filled with the sense data on a check condition. The maximum length of this field is specified in the SRB_SenseLen field. Note that the target can return fewer than the number of sense bytes requested.	

Table 4-7. Return Values from Execute SCSI I/O Command

Value	Meaning
SS_PENDING	SCSI request is in progress.

Example

This example sends a SCSI Inquiry command to host adapter #0, target #0, LUN #0.

```
SRB_ExecSCSICmd MySRB;
DWORD ASPIStatus;
char InquiryBuffer[32];
    .
    .
    .
MySRB.SRB_Header       = SC_EXEC_SCSI_CMD;
MySRB.SRB_HaId         = 0;
MySRB.SRB_Flags        = SRB_DIR_IN | SRB_POSTING;
MySRB.SRB_Hdr_Rsvd     = 0;
MySRB.SRB_Target       = 0;
MySRB.SRB_Lun          = 0;
MySRB.SRB_BufLen       = 32;
MySRB.SRB_SenseLen     = SENSE_LEN;
MySRB.SRB_BufPointer   = InquiryBuffer;
MySRB.SRB_CDBLen       = 6;
MySRB.RB_PostProc      = PostProcedure;
MySRB.CDBByte[0]       = SCSI_INQUIRY;
MySRB.CDBByte[1]       = 0;
MySRB.CDBByte[2]       = 0;
MySRB.CDBByte[3]       = 0;
MySRB.CDBByte[4]       = 32;
MySRB.CDBByte[5]       = 0;
    .
/***********************************************/
/* Make sure all other reserved fields are zeroed
/* before passing the SRB to ASPI for Win32
/***********************************************/
    .
ASPIStatus = SendASPI32Command ( (LPSRB) &MySRB );
    .
    .
```

Abort SCSI I/O Command

Prototype: DWORD SendASPI32Command(LPSRB)

Declaration: LPSRB *lpSRB*;

The **SendASPI32Command** function with command code SC_ABORT_SRB is used to request that a pending SRB be aborted. It should be issued on any I/O request that has not completed if the application wishes to time-out on that request. Success of the abort command is never assured.

Parameter	Description
lpSRB	Points to the following SCSI request block:

```
typedef struct {

        BYTE    SRB_Cmd;        // ASPI command code = SC_ABORT_SRB
        BYTE    SRB_Status;     // ASPI command status byte
        BYTE    SRB_HaId;       // ASPI host adapter number
        BYTE    SRB_Flags;      // Reserved
        DWORD   SRB_Hdr_Rsvd;   // Reserved
        void    *SRB_ToAbort;   // Pointer to SRB to abort

} SRB_Abort, *PSRB_Abort;
```

Table 4-8. Abort SCSI I/O Command

Member	Description
SRB_Cmd	[IN] This field must contain SC_ABORT_SRB.
SRB_Status	[OUT] On return, this field will be the same as the return status defined below.
SRB_HaId	[IN] This field specifies which installed host adapter the request is intended for. Host adapter numbers are always assigned by the ASPI manager layer beginning with zero.
SRB_Flags	[IN] Reserved = 0
SRB_Hdr_Rsvd	Reserved = 0
SRB_ToAbort	[IN] This fields contains a pointer to the SRB which is to be aborted. The actual failure or success of the abort operation is indicated by the status eventually returned in this SRB.

Table 4-9. Return Values from Abort SCSI I/O Command

Value	Meaning
SS_COMP	SCSI/ASPI request has completed without error.
SS_INVALID_HA	Invalid host adapter number.
SS_INVALID_SRB	One or more parameters in the SCSI Request Block (SRB) are set incorrectly.

Example

This example shows how to abort a "stuck" SCSI I/O.

```
SRB_ExecSCSICmd StuckSRB;
SRB_Abort AbortSRB;
DWORD ASPIStatus;
    .
    .
AbortSRB.SRB_Cmd          = SC_ABORT_SRB;
AbortSRB.SRB_HaId         = 0;
AbortSRB.SRB_Flags        = 0;
AbortSRB.SRB_Hdr_Rsvd     = 0;
AbortSRB.SRB_ToAbort      = (LPSRB) &StuckSRB;
ASPIStatus                = SendASPI32Command ( (LPSRB) &AbortSRB );
    .
    .
```

Reset SCSI Device Command

Prototype: DWORD SendASPI32Command(LPSRB)

Declaration: LPSRB *lpSRB*;

The **SendASPI32Command** function with command code SC_RESET_DEV is used to send a SCSI Bus Device reset to the specified target.

Parameter	Description
lpSRB	Points to the following SCSI request block:

```
typedef struct {

        BYTE    SRB_Cmd;            // ASPI command code = SC_RESET_DEV
        BYTE    SRB_Status;         // ASPI command status byte
        BYTE    SRB_HaId;           // ASPI host adapter number
        BYTE    SRB_Flags;          // Reserved
        DWORD   SRB_Hdr_Rsvd;       // Reserved
        BYTE    SRB_Target;         // Target's SCSI ID
        BYTE    SRB_Lun;            // Target's LUN number
        BYTE    SRB_Rsvd1[12];      // Reserved for Alignment
        BYTE    SRB_HaStat;         // Host Adapter Status
        BYTE    SRB_TargStat;       // Target Status
        void    *SRB_PostProc;      // Post routine
        void    *SRB_Rsvd2;         // Reserved
        BYTE    SRB_Rsvd3[32];      // Reserved

} SRB_BusDeviceReset, *PSRB_BusDeviceReset;
```

Table 4-10. Reset SCSI Device Command

Member	Description	
SRB_Cmd	[IN] This field must contain SC_RESET_DEV.	
SRB_Status	[OUT] This field returns one of the following status:	
	SS_PENDING	SCSI request is in progress.
	SS_COMP	SCSI/ASPI request has completed without error.
	SS_ABORTED	SCSI command has been aborted.
	SS_ABORT_FAIL	SCSI command abort failed.
	SS_ERR	SCSI command completed with an error.
	SS_INVALID_SRB	One or more parameters in the SCSI Request Block (SRB) are set incorrectly.
	SS_INVALID_PATH_ID	SCSI ID and LUN are invalid.
SRB_HaId	[IN] This field specifies which installed host adapter the request is intended for. Host adapter numbers are always assigned by the SCSI manager layer beginning with zero.	
SRB_Flags	[IN] Reserved = 0	
SRB_Hdr_Rsvd	Reserved = 0	
SRB_Target	[IN] SCSI ID of target device.	
SRB_Lun	[IN] Logical Unit Number (LUN) of device. This field is ignored by ASPI for Win32, since SCSI Bus Device resets are done on a per-target basis only.	

Table 4-10. Reset SCSI Device Command *(Continued)*

Member	Description	
SRB_HaStat	[OUT] Upon completion of the SCSI command, the ASPI manager sets this field with the host adapter status to one of the following values:	
	HASTAT_OK	Host adapter did not detect an error.
	HASTAT_TIMEOUT	The time allocated for the operation expired.
	HASTAT_COMMAND_TIMEOUT	SRB expired while waiting to be processed.
	HASTAT_SEL_TO	Selection of target timed out.
	HASTAT_MESSAGE_REJECT	While processing SRB, the adapter received a MESSAGE REJECT.
	HASTAT_BUS_RESET	A bus reset was detected.
	HASTAT_PARITY_ERROR	A parity error was detected.
	HASTAT_REQUEST_SENSE_FAILED	The adapter failed in issuing a request sense after a check condition was reported by the target device.
	HASTAT_DO_DU	Data overrun/underrun.
	HASTAT_BUS_FREE	Unexpected Bus Free.
	HASTAT_PHASE_ERR	Target Bus phase sequence failure.
SRB_TargStat	[OUT] Upon completion of the SCSI command, the ASPI manager sets this field with the target status as follows:	
	STATUS_GOOD	No target status.
	STATUS_CHKCOND	Check status (sense data is in SenseArea).
	STATUS_BUSY	Specified Target/LUN is busy.
	STATUS_RESCONF	Reservation conflict.
SRB_PostProc	[IN] If posting is enabled, ASPI for Win32 will post completion of an ASPI request to this function pointer. See *Request Completion/Notification* on page 4-19.	

Table 4-11. Return Values from Reset SCSI Device Command

Value	Meaning
SS_PENDING	SCSI request is in progress.

Example

This example issues a SCSI bus device reset to host adapter #0, target #5.

```
SRB_BusDeviceReset MySRB;
DWORD ASPIStatus;
    .
    .
MySRB.SRB_Header      = SC_RESET_DEV;
MySRB.SRB_HaId        = 0;
MySRB.SRB_Flags       = 0;
MySRB.SRB_Hdr_Rsvd    = 0;
MySRB.SRB_Target      = 5;
MySRB.SRB_Lun         = 0;
ASPIStatus            = SendASPI32Command ( (LPSRB) &MySRB );
    .
    .
```

Get SCSI Disk Information Command

Prototype: DWORD SendASPI32Command(LPSRB)

Declaration: LPSRB *lpSRB*;

The **SendASPI32Command** function with command code SC_GET_DISK_INFO is used to obtain information about a disk type SCSI device. The information returned includes BIOS Int 13h control and accessibility of the device, the drive's Int 13h physical drive number, and the geometry used by the Int 13h services for the drive.

Note: This command is not valid for Windows NT, which does not use the Int 13 interface.

Parameter	Description
lpSRB	Points to the following SCSI request block:

```
typedef struct {

        BYTE    SRB_Cmd;              // ASPI command code = SC_EXEC_SCSI_CMD
        BYTE    SRB_Status;           // ASPI command status byte
        BYTE    SRB_HaId;             // ASPI host adapter number
        BYTE    SRB_Flags;            // Reserved
        DWORD   SRB_Hdr_Rsvd;         // Reserved
        BYTE    SRB_Target;           // Target's SCSI ID
        BYTE    SRB_Lun;              // Target's LUN number
        BYTE    SRB_DriveFlags;       // Driver flags
        BYTE    SRB_Int13HDriveInfo;  // Host Adapter Status
        BYTE    SRB_Heads;            // Preferred number of heads translation
        BYTE    SRB_Sectors;          // Preferred number of sectors translation
        BYTE    SRB_Rsvd1[10];        // Reserved

} SRB_GetDiskInfo, *PSRB_GetDiskInfo;
```

Table 4-12. Get SCSI Disk Information Command

Member	Description
SRB_Cmd	[IN] This field must contain SC_GET_DISK_INFO.
SRB_Status	[OUT] On return, this field will be the same as the return status defined below.
SRB_HaId	[IN] This field specifies which installed host adapter the request is intended for. Host adapter numbers are always assigned by the ASPI manager layer beginning with zero.
SRB_Flags	[IN] Reserved = 0
SRB_Hdr_Rsvd	Reserved = 0
SRB_Target	[IN] SCSI ID of target device.
SRB_Lun	[IN] Logical Unit Number (LUN) of device. This field is ignored by ASPI for Win32, since SCSI Bus Device resets are done on a per-target basis only.
SRB_DriveFlags	[OUT] Upon completion of the SCSI command, the ASPI manager sets this field as follows: Bit 7-2 Reserved Bit 1-0 These bits are encoded to reflect Int 13h control and accessibility to the device. Bit 1 is the most significant bit. DISK_NOT_INT13 Device is not controlled by Int 13h services DISK_INT13_AND_DOS Device is under Int 13h control and is claimed by DOS DISK_INT1 Device is under Int 13h control but not claimed by DOS

Table 4-12. Get SCSI Disk Information Command *(Continued)*

Member	Description
SRB_Int13DriveInfo	[OUT] Upon completion of the SCSI command, the ASPI manager sets this field with the physical drive number that Int 13h services assigned to the device. The valid drive numbers are 0x00 to 0xFF. This field is only valid if bits 1-0 of the SRB_DriveFlags were set to DISK_INT13_AND_DOS or DISK_INT13.
SRB_Heads	[OUT] Upon completion of the SCSI command, the ASPI manager sets this field to the number of heads the Int13h services is using for this device's geometry. The valid drive numbers are 0x00 to 0xFF. This field is only valid if bits 1-0 of the SRB_DriveFlags were set to DISK_INT13_AND_DOS or DISK_INT13.
SRB_Sectors	[OUT] Upon completion of the SCSI command, the ASPI manager sets this field to the number of sectors the Int 13h services is using for this device's geometry. The valid drive numbers are 0x00 to 0xFF. This field is only valid if bits 1-0 of the SRB_DriveFlags were set to DISK_INT13_AND_DOS or DISK_INT13.

Table 4-13. Return Values from Get SCSI Disk Information Command

Value	Meaning
SS_COMP	SCSI/ASPI request has completed without error.
SS_INVALID_HA	Invalid host adapter number.
SS_INVALID_SRB	One or more parameters in the SCSI Request Block (SRB) are set incorrectly.

Example

This example obtains disk information from device LUN 0, SCSI ID 2, attached to host adapter #0.

```
SRB_GetDiskInfo MySRB;
DWORD ASPIStatus;
    .
    .
MySRB.SRB_Header.SRB_Cmd  = SC_GET_DISK_INFO;
MySRB.SRB_HaId            = 0;
MySRB.SRB_Flags           = 0;
MySRB.SRB_Hdr_Rsvd        = 0;
MySRB.SRB_Target          = 2;
MySRB.SRB_Lun             = 0;
ASPIStatus                = SendASPI32Command ( (LPSRB) &MySRB );
    .
    .
```

Request Completion/Notification

Once an ASPI for Win32 SCSI request has been sent to the ASPI manager, there are three ways of being notified that the SCSI request has completed. The first and recommended method uses event notification. The second method uses callback (also referred to as *posting*). The third method uses polling.

Using Event Notification

Event notification is an ideal mechanism for notifying ASPI clients of the completion of an ASPI request. ASPI clients may block on this event until completion. Upon completion of a request, the ASPI for Win32 manager will set the event to the signaled state. The ASPI client is responsible for making sure that the event is not in a signaled state when an ASPI command request is submitted to the ASPI for WIN32 manager. The following sample code illustrates event notification. The sample code sends a SCSI Inquiry command to target #2

```
SRB_ExecSCSICmd MySRB;
HANDLE ASPICompletionEvent;
DWORD ASPIEventStatus;
DWORD ASPIStatus;
char InquiryBuffer[32];
.
/***********************************************/
/* Create event for MySRB.  Initial state      */
/* non-signaled, manual reset.                 */
/***********************************************/
if ((ASPICompletionEvent = CreateEvent(NULL,FALSE,FALSE,NULL)) == NULL)
     return FALSE;

/***********************************************/
/* Code is entered with 'MySRB' zeroed.        */
/***********************************************/

MySRB.SRB_Cmd          = SC_EXEC_SCSI_CMD;
MySRB.SRB_Flags        = SRB_DIR_SCSI|SRB_EVENT_NOTIFY;
MySRB.SRB_Target       = 2;
MySRB.SRB_BufLen       = 32;
MySRB.SRB_SenseLen     = SENSE_LEN;
MySRB.SRB_BufPointer   = InquiryBuffer;
MySRB.SRB_CDBLen       = 6;
MySRB.CDBByte[0]       = SCSI_INQUIRY;
MySRB.CDBByte[4]       = 32;
MySRB.SRB_PostProc     = ASPICompletionEvent;
.
.
ASPIStatus             = SendASPI32Command ( (LPSRB) &MySRB );

/***********************************************/
/* Block on event till signaled                */
/***********************************************/
if ( MySRB.SRB_Status == SS_PENDING )
     ASPIEventStatus == WaitForSingleObject(ASPICompletionEvent, TIMEOUT);

/***********************************************/
/* Reset event to non-signaled state.          */
/***********************************************/
if (ASPIEventStatus == WAIT_OBJECT_0)
    ResetEvent(ASPICompletionEvent);
          .
          .
```

Using Callback

Callback (or *posting*) may also be used to receive notification that a SCSI request has completed. When callback is used, ASPI for Win32 posts completion by passing control to your callback function. For example, the following code segment sends a SCSI Inquiry command to target #2 during the WM_CREATE message.

```
LRESULT CALLBACK WndProc (HWND, UINT, UPARAM, LPARAM);
void APIENTRY ASPIPostProc (PSRB_ExecSCSICmd );

HWND       PostHWND;
HANDLE     hInstance;
                    .
                    .
                    .
// ****************************************************************************
//
// Function:       ASPIPostProc
//
// Description:    If POSTING is enabled,  this function is called by ASPI
//                 for Windows when the SCSI request has completed.  This
//                 sample function simply posts a message to our Window
//                 handle to indicate that the SCSI request has completed.
//
//      DoneSRB    This parameter points to the ASPI SCSI Request Block (SRB)
//                 which has completed.
//
//      Returns    Nothing
//
//****************************************************************************
#ifdef WIN32
        void ASPIPostProc(PSRB_ExecSCSICmd DoneSRB)
        {
            PostMessage(PostHWND,WM_ASPIPOST,0,(LPARAM)DoneSRB);
            return;
        }
#else
        void _loadds __far __pascal ASPIPostProc(LPSRB DoneSRB)
        {
            PostMessage(PostHWND,WM_ASPIPOST,
                        (WORD)((SRB_ExecSCSICmd6 far *)DoneSRB)->SRB_Status,
                        (DWORD)DoneSRB);
            return;
        }
#endif

// ************************************************************************
// *
// *    Procedure: WndProc()
// *
// ************************************************************************
#ifdef WIN32
        LRESULT CALLBACK WndProc(HWND hwnd,UINT message,WPARAM wParam,
        LPARAM lParam)
#else
        long __far __pascal _export WndProc(HWND hwnd,UINT message,WPARAM
        wParam,LPARAM lParam)
#endif
{
        HDC             hdc;
        PAINTSTRUCT     ps;
        TEXTMETRIC      tm;
        HMENU           hMenu;
        static short    cxClient, cyClient;
        int             i=0;
        WORD            status;
        #ifdef WIN32
```

```
            SRB_ExecSCSICmd *SRBPtr;
        #else
            SRB_ExecSCSICmd6 far *SRBPtr;
        #endif

        switch (message)
        {
            case WM_CREATE:
                hdc = GetDC(hwnd);
                SelectObject(hdc,GetStockObject(SYSTEM_FIXED_FONT));
                GetTextMetrics(hdc,&tm);
                cxChar = tm.tmAveCharWidth;
                cyChar = tm.tmHeight;
                ReleaseDC(hwnd,hdc);
                rect.top = 0;
                return 0;

            case WM_PAINT:
                InvalidateRect(hwnd,NULL,TRUE);
                hdc = BeginPaint(hwnd,&ps) ;
                EndPaint (hwnd,&ps) ;
                return 0 ;

            case WM_SIZE:
                cxClient = LOWORD(lParam);
                cyClient = HIWORD(lParam);
                rect.right = LOWORD(lParam);
                rect.bottom = HIWORD(lParam);
                UpdateWindow(hwnd);
                return 0;

            case WM_COMMAND:
                hMenu = GetMenu(hwnd);
                switch (wParam)
                {
                    case ID_FILE_EXIT:
                        SendMessage(hwnd,WM_CLOSE,0,0L);
                        return 0;

                    default:
                        // 10 = target #0
                        // 11 = target #1
                        // 12 = target #2
                        // 13 = target #3
                        // 14 = target #4
                        // 15 = target #5
                        // 16 = target #6
                        // 17 = target #7

                        // Toggle the SCSI target scan status (ENABLED/DISABELD)
                        if ((wParam >= 10 && wParam <= 17) ||
                            (wParam >= 20 && wParam <= 27))
                        {
                            status = GetMenuState(hMenu,wParam,MF_BYCOMMAND) &
MF_CHECKED;
                            CheckMenuItem(hMenu,wParam,MF_BYCOMMAND | ((status) ?
MF_UNCHECKED:MF_CHECKED));
                        }
                        return 0;
                }

            case WM_ASPIPOST:
                #ifdef WIN32
                    SRBPtr = (PSRB_ExecSCSICmd)lParam;
                #else
                    SRBPtr = (SRB_ExecSCSICmd6 far *)lParam;
                #endif
                DisplaySCSIID(hwnd,SRBPtr);
                ScanNextTarget(hwnd);
```

```
            return 0;

        case WM_DESTROY:
            PostQuitMessage(0) ;
            return 0 ;
    }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

When the post routine is called, this sample post handler will fill the wParam field with the status of ASPI command (SRB_Status) while the lParam field will contain a pointer to the SRB which has completed.

Polling

Polling is another method of determining SCSI request completion. This method is not recommended. After the command is sent and ASPI for Win32 returns control back to the calling application, you can then poll the status byte waiting for the command to complete. For example, the following code segment will execute a SCSI Inquiry command to target #2 using polling.

```
SRB_ExecSCSICmd6 MySRB;
DWORD ASPIStatus;
char InquiryBuffer[32];
    .
    .
/***********************************************/
/* Code is entered with 'MySRB' zeroed.
/***********************************************/
MySRB.SRB_Cmd         = SC_EXEC_SCSI_CMD;
MySRB.SRB_Flags       = SRB_DIR_SCSI;
MySRB.SRB_Target      = 2;
MySRB.SRB_BufLen      = 32;
MySRB.SRB_SenseLen    = SENSE_LEN;
MySRB.SRB_BufPointer  = InquiryBuffer;
MySRB.SRB_CDBLen      = 6;
MySRB.CDBByte[0]      = SCSI_INQUIRY;
MySRB.CDBByte[4]      = 32;
    .
    .
ASPIStatus            = SendASPI32Command ( (LPSRB) &MySRB );
                                              // Send Inquiry command
while ( MySRB.SRB_Status == SS_PENDING );     // Wait till it's finished
```

Migrating to ASPI for Win32

If you have written ASPI or ASPI for Windows applications in the past, here are some key points to keep in mind when you migrate the applications to ASPI for Win32:

- Although structure definitions have changed slightly to optimize alignment for 32-bit processors, structure names are consistent with those used in previous ASPI developer's kits. If you would like to use one source base for both 16-bit and 32-bit applications, make sure that you conditionally compile with the appropriate include files for each programming model. New 32-bit include files are available in the ASPI developer's kit. There are many #ifdef statements in the sample code.
- The CDB area has been fixed in length at 16. Therefore, the sense data area no longer shifts location depending on command length. If you are developing an application targeted only at Win32, you no longer need to account for the "floating" sense buffer.
- For requests requiring data transfers, the direction bits in the SRB Flags field must be set correctly. Direction bits are no longer optional for data transfers. For requests not requiring data transfers, the direction bits are ignored.
- If you are using dynamic DLL linking, use LoadLibrary (*winaspi32.dll*) instead of LoadLibrary(*winaspi.dll*) in order to access ASPI for Win32. If you are using static DLL linking, use the *wnaspi32.lib*.
- All pointers in Win32 are 32 bits (Flat model). Therefore, FAR references are not used.
- When allocating memory, page locking and global allocations are not necessary.

Programming Guidelines

The following are additional guidelines for writing ASPI for Win32 applications.

- ASPI for Win32 applications/clients should send an ASPI Abort command to terminate pending ASPI requests before exit.
- SRBs and data buffers do not need to be in page-locked memory. The ASPI manager takes care of locking buffers and SRBs.
- If an error SS_BUFFER_TO_BIG is returned by the **SendASPI32Command** routine, you should break the transfer down into 64 KByte transfers or less. For maximum compatibility, we recommend that you do not request transfer sizes larger than 64 KBytes unless your application requires it.
- If you send an ASPI request with posting or callback enabled, the callback procedure will always be called. The post or callback routine is called as a standard C function. The caller (in this case, the ASPI manager) cleans up the stack.
- ASPI for Win32 is fully re-entrant and permits overlapped, asynchronous I/O. Clients can send additional ASPI requests while others are pending completion. Be sure to use a separate SRB for each ASPI request.
- Be sure to zero out all reserved fields before passing an SRB to ASPI for Win32.
- Be sure that buffers are aligned according to the buffer alignment mask returned by the SC_HA_INQUIRY command. An alignment of at least a double word is recommended.

- All SRB structures sent to **SendASPI32Command** should be compiled packed with byte alignment.
- Some SCSI devices may have security restrictions that make them inaccessible for read/write operations. In operating systems where security is a feature, accessing a device where the application does not have sufficient security privilege results in a SS_SECURITY_VIOLATION error.
- When scanning for devices, the **SendASPI32Command** may also return the status SS_NO_DEVICE in the SRB_Status field. Check for this exception in addition to the host adapter status HASTAT_SEL_TO. As previously mentioned, the callback procedure is always called, including this condition.
- The SRB flags SRB_POSTING and SRB_EVENT_NOTIFY are mutually exclusive.