3 ASPI for Win16 Specification

About This Chapter

Read this chapter to find out about

- An overview of the steps involved in programming ASPI for Win16
- ASPI for Win16 commands
- Polling and posting of ASPI for Win16 SCSI requests
- Important information for ASPI for Win16 developers
- The *scanscsi.exe* application, which scans the SCSI bus and displays the device name of each target
- A summary of the error codes and messages returned by ASPI routines

▼▼▼▼▼ 3

Using ASPI for Win16 – An Overview

ASPI for Win16 is implemented as a Dynamic Link Library (DLL). The name of this file is *winaspi.dll*. ASPI function calls are used to return information about the ASPI manager, host adapter, and SCSI devices, but are mainly used to execute SCSI I/O requests. The ASPI for Win16 layer is fully multitasking and can accept function calls before previous calls have completed. Two functions must be imported from *winaspi.dll* into your Windows application:

Function	Description
GetASPISupportInfo	This function returns the number of host adapters installed and other miscellaneous information. You should call this function to make sure that ASPI is properly initialized before calling the SendASPICommand function.
SendASPICommand	This function allows you to send an ASPI for Win16 command.

All SRBs and data buffers *must* be in locked memory before they are passed to ASPI. This is discussed in greater detail on page 3-21.

ASPI Managers For Windows

This document does not define the protocol between *winaspi.dll* and any DOS ASPI managers that may be loaded. There are many reasons for this, including:

- Some hardware companies may write ASPI for Win16 managers that do not include concurrent ASPI for DOS support.
- Some hardware companies may decide to have *winaspi.dll* communicate with a Windows 386 Enhanced Mode Virtual Device Driver (VxD).
- Some hardware companies may decide to only support Windows 3.1, which may or may not have improved hardware support.

It is also not the intent of this specification to define which modes of Windows should be supported. We anticipate that most hardware companies will support ASPI for Win16 in Standard and 386 Enhanced Modes and will not support Real Mode.

GetASPISupportInfo Function

Prototype: WORD GetASPISupportInfo(VOID)

The **GetASPISupportInfo** function returns the number of host adapters installed and other miscellaneous information. This function should be called first before issuing an ASPI command to ensure that ASPI has been properly initialized. This function call does not perform any initialization itself; instead, it confirms that everything is ready for processing.

This function has no parameters.

The return value specifies the outcome of the function. The LOBYTE returns the number of host adapters installed if the HIBYTE value equals SS_COMP. The HIBYTE returns whether ASPI for Win16 is ready to accept ASPI commands. Refer to the example code. The HIBYTE is defined as follows:

Table 3-1. Return Values from GetASPISupportInfo Function

Value	Meaning
SS_COMP	SCSI/ASPI request has completed without error.
SS_OLD_MANAGER	One or more ASPI for DOS managers are loaded which do not support ASPI for Win16.
SS_ILLEGAL_MODE	This ASPI manager does *not* support this mode of Windows. You typically see this error code when running Windows in Real Mode.
SS_NO_ASPI	No ASPI managers are loaded. This typically occurs when a DOS ASPI manager is not resident in memory.
SS_FAILED_INIT	For some reason, other than SS_OLD_MANAGER, SS_ILLEGAL_MODE or SS_NO_ASPI, ASPI for Win16 could not properly initialize itself. This may be caused by a lack of system resources.

Example

The following example returns the current status of ASPI for Win16:

```
WORD ASPIStatus;
BYTE NumAdapters;
HWND hwnd;

           .
           .

ASPIStatus = GetASPISupportInfo();
switch ( HIBYTE(ASPIStatus) )
    {
    case SS_COMP:
        //ASPI for Windows is properly initialized
        NumAdapters = LOBYTE(ASPIStatus);
        break;
    case SS_NO_ASPI:
        MessageBox( hwnd,
                    "No ASPI managers were found!!",
                    NULL,
                    MB_ICONSTOP );
        return 0;
    case SS_ILLEGAL_MODE:
        MessageBox( hwnd,
                    "ASPI for Windows does not support this mode!!",
                    NULL,
                    MB_ICONSTOP );
        return 0;
    case SS_OLD_MANAGER:
        MessageBox( hwnd,
                    "An ASPI manager which does not support Windows
is resident!!",
                    NULL,
                    MB_ICONSTOP );
        return 0;
    default:
        MessageBox( hwnd,
                "     ASPI for Windows is not initialized!!",
                    NULL,
                    MB_ICONSTOP );
        return 0;
    }
            .
            .
```

Host Adapter Inquiry Command

Prototype: WORD SendASPICommand(lpSRB)

Declaration: LPSRB *lpSRB*;

The **SendASPICommand** function with command code SC_HA_INQUIRY is used to get information on the installed host adapter hardware, including number of host adapters installed.

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
        WORD    SRB_55AASignature;    // Extended signature bytes
        WORD    SRB_ExtBufferSize;    // Extended buffer length
        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
        BYTE    HA_ExtBuffer[4];      // Extended buffer area

} SRB_HAInquiry;
```

Table 3-2. Host Adapter Inquiry Command

Member	Description	R/W*
SRB_Cmd	This field must contain SC_HA_INQUIRY.	W
SRB_Status	On return, this field is the same as the return status defined below.	R
SRB_HaId	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.	W
SRB_Flags	Reserved = 0	—
SRB_55AASignature	Place a value of AA55h at this location if you want to receive back extended host adapter information (returned in the HA_ExtBuffer field). If the ASPI manager supports this command, it switches the value of this field from AA55h to 55AAh. If the ASPI manager does not support the extended host adapter information, the value of this field remains unchanged.	R/W
SRB_ExtBufferSize	Place the number of bytes available in the HA_ExtBuffer field. (A typical value is 4.)	R/W
HA_Count	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 GetASPISupportInfo can be used.	R
HA_SCSI_ID	The ASPI manager sets this field with the SCSI ID of the given host adapter.	R
HA_ManagerId[..]	The ASPI manager fills this 16-character buffer with an ASCII string describing the ASPI manager.	R
HA_Identifier[..]	The ASPI manager fills this 16-character buffer with an ASPI string describing the SCSI host adapter.	R

* Indicates whether the field is sent to ASPI (W), returned from ASPI (R), or reserved (—).

Table 3-2. Host Adapter Inquiry Command *(Continued)*

Member	Description		R/W*
HA_Unique[..]	The definition of this field is left to implementation notes specific to a particular host adapter.		R
HA_ExtBuffer	The bits in this field are defined as follows:		R
	Bit 15-4	Reserved	
	Bit 3	0 = Not a Wide SCSI 32-bit host adapter 1 = Wide SCSI 32-bit host adapter	
	Bit 2	0 = Not a Wide SCSI 16-bit host adapter 1 = Wide SCSI 16-bit host adapter	
	Bit 1	0 = Residual byte length not reported 1 = Residual byte length reported. See *Residual Byte Length* below for more information.	
	Bit 0	Reserved	

* Indicates whether the field is sent to ASPI (W), returned from ASPI (R), or reserved (—).

The return value specifies the outcome of the function. One of the following values is returned by ASPI for Win16:

Table 3-3. 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.
SS_INVALID_SRB	One or more parameters in the SCSI Request Block (SRB) are set incorrectly.

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. Another example: 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

The following example retrieves host adapter hardware information from adapter #0:

```
SRB_HAInquiry MySRB;
WORD ASPI_Status;
            .
            .
MySRB.SRB_Cmd          = SC_HA_INQUIRY;
MySRB.SRB_HaId         = 0;
MySRB.SRB_Flags        = 0;
MySRB.SRB_Hdr_Rsvd     = 0;
ASPI_Status = SendASPICommand ( (LPSRB) &MySRB );
            .
            .
```

Get Device Type Command

Prototype: WORD SendASPICommand(lpSRB)

Declaration: LPSRB *lpSRB*;

The **SendASPICommand** function with command code SC_GET_DEV_TYPE is used by Windows applications to identify the targets which they need to support. For example, a Windows tape backup package can scan each Target/LUN on each installed host adapter looking for the device type corresponding to sequential access devices. This eliminates the need for each Windows application to duplicate the effort of scanning the SCSI bus for devices.

Note: Some Windows applications may scan the SCSI bus themselves instead of using this command. This allows them to detect SCSI devices that were powered up after ASPI initialization.

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;          // ASPI request flags
        DWORD   SRB_Hdr_Rsvd;       // Reserved, MUST = 0
        BYTE    SRB_Target;         // Target's SCSI ID
        BYTE    SRB_Lun;            // Target's LUN number
        BYTE    SRB_DeviceType;     // Target's peripheral device type

} SRB_GDEVBlock;
```

Table 3-4. Get Device Type Command

Member	Description	R/W*
SRB_Cmd	This field must contain SC_GET_DEV_TYPE.	W
SRB_Status	On return, this field is the same as the return status defined below.	R
SRB_HaId	This field specifies which installed host adapter the request is intended for. Host adapter numbers are assigned by the SCSI manager layer and begin with zero.	W
SRB_Flags	Reserved = 0	—
SRB_Hdr_Rsvd	This DWORD field is currently reserved for this function and must be zeroed before passed to the ASPI manager.	—
SRB_Target	SCSI ID of target device.	W
SRB_Lun	Logical Unit Number (LUN) of device.	W
SRB_DeviceType	The ASPI manager fills this field with the peripheral device type. Refer to any SCSI specification to learn more about the SCSI Inquiry command.	R

* Indicates whether the field is sent to ASPI (W), returned from ASPI (R), or reserved (—).

The return value specifies the outcome of the function. One of the following values is returned by ASPI for Win16:

Table 3-5. 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.
SS_INVALID_SRB	One or more parameters in the SCSI Request Block (SRB) are set incorrectly.

Example

The following example retrieves the peripheral device type from host adapter #0, target ID #4, and LUN #0.

```
SRB_GDEVBlock MySRB;
WORD ASPI_Status;
            .
            .
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;
ASPI_Status = SendASPICommand ( (LPSRB) &MySRB );
            .
/***********************************************/
/* If ASPI_Status == SS_COMP, MySRB.SRB_DeviceType */
/* will contain the peripheral device type.        */
/***********************************************/
            .
            .
```

Execute SCSI I/O Command

Prototype: WORD SendASPICommand(lpSRB)

Declaration: LPSRB *lpSRB*;

The **SendASPICommand** function with command code SC_EXEC_SCSI_CMD is used to execute a SCSI command: for example, sending a SCSI Test Unit Ready command to a tape drive.

Parameter	Description
lpSRB	Points to one of the following SCSI request blocks:

```
typedef struct {                        // Structure for 6-byte CDBs

     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, MUST = 0
     BYTE   SRB_Target;                 // Target's SCSI ID
     BYTE   SRB_Lun;                    // Target's LUN number
     DWORD  SRB_BufLen;                 // Data Allocation LengthPG
     BYTE   SRB_SenseLen;               // Sense Allocation Length
     BYTE   far *SRB_BufPointer;        // Data Buffer Pointer
     DWORD  SRB_Rsvd1;                  // Reserved, MUST = 0
     BYTE   SRB_CDBLen;                 // CDB Length = 6
     BYTE   SRB_HaStat;                 // Host Adapter Status
     BYTE   SRB_TargStat;               // Target Status
     FARPROC SRB_PostProc;              // Post routine

     BYTE   SRB_Rsvd2[34];              // Reserved, MUST = 0
     BYTE   CDBByte[6];                 // SCSI CDB
     BYTE   SenseArea6[SENSE_LEN];      // Request Sense buffer

} SRB_ExecSCSICmd6;

typedef struct {                        // Structure for 10-byte CDBs

     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, MUST = 0
     BYTE   SRB_Target;                 // Target's SCSI ID
     BYTE   SRB_Lun;                    // Target's LUN number
     DWORD  SRB_BufLen;                 // Data Allocation Length
     BYTE   SRB_SenseLen;               // Sense Allocation Length
     BYTE   far *SRB_BufPointer;        // Data Buffer Pointer
     DWORD  SRB_Rsvd1;                  // Reserved, MUST = 0
     BYTE   SRB_CDBLen;                 // CDB Length = 10
     BYTE   SRB_HaStat;                 // Host Adapter Status
     BYTE   SRB_TargStat;               // Target Status
     FARPROC SRB_PostProc;              // Post routine

     BYTE   SRB_Rsvd2[34];              // Reserved, MUST = 0
     BYTE   CDBByte[10];                // SCSI CDB
     BYTE   SenseArea10[SENSE_LEN];     // Request Sense buffer

} SRB_ExecSCSICmd10;
```

Note: You can easily create a new structure for non-standard CDB lengths.

Table 3-6. Execute SCSI I/O Command

Member	Description		R/W*
SRB_Cmd	This field must contain SC_EXEC_SCSI_CMD.		W
SRB_Status	On return, this field is the same as the return status defined below.		R
SRB_HaId	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.		W
SRB_Flags	The SRB Flags field is defined as follows:		W
	SRB_DIR_SCSI	Direction determined by SCSI command. Length not checked.	
	SRB_DIR_IN	Transfer from SCSI target to host. Length checked.	
	SRB_DIR_OUT	Transfer from host to SCSI target. Length checked.	
	SRB_POSTING	If this value is OR'ed in with one of the previous three values, posting is enabled. Refer to *ASPI Posting* on page 3-19.	
	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	This DWORD field is currently reserved for this function and must be zeroed before passed to the ASPI manager.		—
SRB_Target	SCSI ID of target device.		W
SRB_Lun	Logical Unit Number (LUN) of device.		W
SRB_BufLen	This field indicates the number of bytes to be transferred. If the SCSI command does not transfer data (i.e., 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 3-7.		R/W
SRB_SenseLen	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.		W
SRB_BufPointer	This field is a pointer to the data buffer.		W
SRB_CDBLen	This field establishes the length, in bytes, of the SCSI Command Descriptor Block (CDB). This value is typically 6 or 10.		W
SRB_HaStat	Upon completion of the SCSI command, the ASPI manager sets this field with the host adapter status as follows:		R
	HASTAT_OK	Host adapter did not detect an error	
	HASTAT_SEL_TO	Selection time-out	
	HASTAT_DO_DU	Data overrun/underrun	
	HASTAT_BUS_FREE	Unexpected Bus Free	
	HASTAT_PHASE_ERR	Target Bus phase sequence failure	

* Indicates whether the field is sent to ASPI (W), returned from ASPI (R), or reserved (—).

Table 3-6. Execute SCSI I/O Command *(Continued)*

Member	Description		R/W*
SRB_TargStat	Upon completion of the SCSI command, the ASPI manager sets this field with the target status as follows:		R
	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	If posting is enabled, ASPI for Win16 posts completion of an ASPI request to this function pointer. Refer to *ASPI Posting* on page 3-19.		W
CDBByte[..]	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.		W
SenseArea[..]	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. The target can return fewer than the number of sense bytes requested.		R

* Indicates whether the field is sent to ASPI (W), returned from ASPI (R), or reserved (—).

The return value specifies the outcome of the function. ASPI for Win16 returns one of the following values:

Table 3-7. Return Values from Execute SCSI I/O Command

Value	Meaning
SS_PENDING	SCSI request is in progress.
SS_COMP	SCSI/ASPI request has completed without error.
SS_ABORTED	SCSI command has been aborted.
SS_ERR	SCSI command has completed with an error.
SS_INVALID_SRB	The SCSI Request Block (SRB) has one or more parameters set incorrectly.
SS_ASPI_IS_BUSY	The ASPI manager cannot handle the request at this time. This error generally occurs if the ASPI manager is already using up all of his resources to execute other requests. Try resending the command later.
SS_BUFFER_TO_BIG	The ASPI manager cannot handle the given transfer size. Refer to *Miscellaneous* on page 3-21 for more information.

Example

The following example sends a SCSI Inquiry command to host adapter #0, target #0, and LUN #0:

```
SRB_ExecSCSICmd6 MySRB;
char InquiryBuffer[32];
FARPROC lpfnPostProcedure;
            .
            .
lpfnPostProcedure      = MakeProcInstance (PostProcedure, hInstance);
            .
            .
MySRB.SRB_Cmd          = SC_EXEC_SCSI_CMD;
MySRB.SRB_HaId         = 0;
MySRB.SRB_Flags        = SRB_DIR_SCSI | 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.SRB_PostProc     = lpfnPostProcedure;
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 Windows     */
/***********************************************/
            .
SendASPICommand ( (LPSRB) &MySRB );
            .
            .
```

Abort SCSI I/O Command

Prototype: WORD SendASPICommand(lpSRB)

Declaration: LPSRB *lpSRB*;

The **SendASPICommand** function with command code SC_ABORT_SRB is used to request that an 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;          // ASPI request flags
        DWORD   SRB_Hdr_Rsvd;       // Reserved, MUST = 0
        LPSRB   SRB_ToAbort;        // Pointer to SRB to abort

} SRB_Abort;
```

Table 3-8. Abort SCSI I/O Command

Member	Description	R/W*
SRB_Cmd	This field must contain SC_ABORT_SRB.	W
SRB_Status	On return, this field will be the same as the return status defined below.	R
SRB_HaId	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.	W
SRB_Flags	Reserved = 0	—
SRB_Hdr_Rsvd	This DWORD field is currently reserved for this function and must be zeroed before passed to the ASPI manager.	—
SRB_ToAbort	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.	W

* Indicates whether the field is sent to ASPI (W), returned from ASPI (R), or reserved (—).

The return value specifies the outcome of the function. One of the following values is returned by ASPI for Win16:

Table 3-9. 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

The following example shows how to abort a *stuck* SCSI I/O:

```
SRB_ExecSCSICmd6 StuckSRB;
SRB_Abort AbortSRB;
WORD ASPI_Status;
           .
           .
AbortSRB.SRB_Cmd      = SC_ABORT_SRB;
AbortSRB.SRB_HaId     = 0;
AbortSRB.SRB_Flags    = 0;
AbortSRB.SRB_Hdr_Rsvd = 0;
AbortSRB.SRB_ToAbort  = (LPSRB) &StuckSRB;
ASPI_Status = SendASPICommand ( (LPSRB) &AbortSRB );
           .
           .
while (StuckSRB.SRB_Status==SS_PENDING);
           .
           .
/*********************************************/
/* This sample code has no error handling, time-  */
/* out code, nor does it free up the processor.   */
/* Your application should be more robust.        */
/*********************************************/
```

Reset SCSI Device Command

Prototype: WORD SendASPICommand(lpSRB)

Declaration: LPSRB *lpSRB*;

The **SendASPICommand** 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;            // ASPI request flags
        DWORD   SRB_Hdr_Rsvd;         // Reserved, MUST = 0
        BYTE    SRB_Target;           // Target's SCSI ID
        BYTE    SRB_Lun;              // Target's LUN number
        BYTE    SRB_ResetRsvd1[14];   // Reserved, MUST = 0
        BYTE    SRB_HaStat;           // Host Adapter Status
        BYTE    SRB_TargStat;         // Target Status
        FARPROC SRB_PostProc;         // Post routine
        BYTE    SRB_ResetRsvd2[34];   // Reserved, MUST = 0

} SRB_BusDeviceReset
```

Table 3-10. Reset SCSI Device Command

Member	Description		R/W*
SRB_Cmd	This field must contain SC_RESET_DEV.		W
SRB_Status	On return, this field is the same as the return status defined below.		R
SRB_HaId	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.		W
SRB_Flags	Reserved = 0		—
SRB_Hdr_Rsvd	This DWORD field is currently reserved for this function and must be zeroed before passed to the ASPI manager.		—
SRB_Target	SCSI ID of target device.		W
SRB_Lun	Logical Unit Number (LUN) of device. This field is ignored by ASPI for Win16 since SCSI Bus Device resets are done on a per-target basis only.		W
SRB_HaStat	Upon completion of the SCSI command, the ASPI manager sets this field with the host adapter status as follows:		R
	HASTAT_OK	Host adapter did not detect an error	
	HASTAT_SEL_TO	Selection time-out	
	HASTAT_DO_DU	Data overrun/underrun	
	HASTAT_BUS_FREE	Unexpected Bus Free	
	HASTAT_PHASE_ERR	Target Bus phase sequence failure	

* Indicates whether the field is sent to ASPI (W), returned from ASPI (R), or reserved (—).

Table 3-10. Reset SCSI Device Command *(Continued)*

Member	Description		R/W*
SRB_TargStat	Upon completion of the SCSI command, the ASPI manager sets this field with the target status as follows:		R
	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	If posting is enabled, ASPI for Win16 posts completion of an ASPI request to this function pointer. Refer to *ASPI Posting* on page 3-19.		W

* Indicates whether the field is sent to ASPI (W), returned from ASPI (R), or reserved (—).

The return value specifies the outcome of the function. One of the following values is returned by ASPI for Win16. See the ASPI command code definitions to learn which ASPI commands return which errors.

Table 3-11. Return Values from Reset SCSI Device 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.
SS_ASPI_IS_BUSY	The ASPI manager cannot handle the request at this time. This error generally occurs if the ASPI manager is already using up all of his resources to execute other requests. Try resending the command later.

Example

The following example issues a SCSI Bus Device reset to host adapter #0, target #5:

```
SRB_BusDeviceReset MySRB;
WORD ASPI_Status;
            .
            .
MySRB.SRB_Cmd          = SC_RESET_DEV;
MySRB.SRB_HaId         = 0;
MySRB.SRB_Flags        = 0;
MySRB.SRB_Hdr_Rsvd     = 0;
MySRB.SRB_Target       = 5;
MySRB.SRB_Lun          = 0;
ASPI_Status = SendASPICommand ( (LPSRB) &MySRB );
            .
/***********************************************/
/* Make sure all other reserved fields are zeroed */
/* before passing the SRB to ASPI for Windows     */
/***********************************************/
            .
while (MySRB.SRB_Status==SS_PENDING);
            .
            .
/***********************************************/
/* This sample code has no error handling, time-  */
/* out code, nor does it free up the processor.   */
/* Your application should be more robust.        */
/***********************************************/
```

ASPI Polling

Once you send an ASPI for Win16 SCSI request, you have two ways of being notified that the SCSI request has completed. The first and simplest method is called *polling*. After the command is sent, and ASPI for Win16 returns control to your program, you can poll the status byte waiting for the command to complete. For example, the following code segment sends a SCSI Inquiry command to target #2.

```
SRB_ExecSCSICmd6 MySRB;
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;
        .
        .
SendASPICommand ( (LPSRB) &MySRB );      // Send Inquiry command
while ( MySRB.SRB_Status == SS_PENDING );// Wait till it's finished

/***********************************************/
/* At this point, the SCSI command has completed  */
/* with or without an error.                   */
/***********************************************/

if ( MySRB.SRB_Status == SS_COMP )
   ;// Command completed without error
else
   ;// Command completed with error
```

Since Windows is currently a non-preemptive multitasking operating system, you should use polling with caution. The example above is not a good example of freeing up the processor, nor does it have any time-out handler. See *Source Code for scanscsi.c* on page 3-23 for sample code that does free up the processor while using polling.

ASPI Posting

Most applications use *posting*, rather than polling, to be notified that a SCSI request has completed. When posting is enabled, ASPI for Win16 posts completion by calling your callback function. For example, the following code segment sends a SCSI Inquiry command to target #2 during the WM_CREATE message.

```
long FAR PASCAL WndProc (HWND, WORD, WORD, LONG);
void FAR PASCAL ASPIPostProc ( LPSRB );

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, the sample post handler fills the wParam field and contains the status of ASPI command (SRB_Status), while the lParam field contains a far pointer to the SRB which has completed. See *Source Code for scanscsi.c* on page 3-23.

Miscellaneous

This section contains important information about using ASPI for Win16.

- Your ASPI for Win16 program should *never* exit with pending SCSI I/Os. Doing so could lead to system instability. Send an ASPI Abort command if necessary.
- Your SRBs and data buffers must be in page-locked memory. Most SCSI host adapters are bus masters. This means that the data buffer must *not* move while the transfer is taking place. We recommend that you allocate your buffers using GlobalAlloc and lock it first with GlobalLock and then with GlobalPageLock. This technique can overcome some quirks that Windows 3.x seems to have with locking down buffers.

Note: This is no longer a requirement if you are writing to the ASPI for Win32 interface.

- At a minimum, all ASPI for Win16 managers must support transfers of 64 KBytes or less. It is not possible for all SCSI host adapters to transfer data blocks larger than this. If the ASPI manager cannot support the requested transfer size, the SS_BUFFER_TOO_BIG error is returned from the **SendASPICommand** routine. No posting occurs. If this happens, 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.
- Do not forget to support the SS_ASPI_IS_BUSY return status when sending a SCSI command. Under extreme loads, some ASPI for Win16 managers may not have enough resources to service each request.
- If you send an ASPI request with posting enabled, and the return value is not equal to SS_PENDING (in other words, the request is not in progress), then ASPI for Win16 *does not* post completion to your specified window handle. (Refer to the specific return value for more information about why the request is not in progress.)
- ASPI for Win16 is fully multitasking. You can send a request to ASPI while another request is executing. Make sure you use a separate SRB for each ASPI request. We also recommend that you only send one SRB at a time per target.
- If you use posting, your post routine will most likely be called during interrupt time. Since most Windows routines are non-reentrant, you should call Windows routines with caution. You can call **PostMessage** during interrupt time, however.
- Make sure that you zero out all reserved fields before passing the SRB to ASPI for Win16.