Microsoft
Hardware White Paper

Draft ACPI Driver Interface Design Notes and Reference

This paper documents the Microsoft-provided interface offered by the ACPI Driver to other kernel-mode drivers.

Important: This document is an early draft revision, distributed primarily for review comment. Changes may be made to any part of the document, especially at the detailed level.

Version 0.91
November 11, 1998

Contents

Introduction

This paper documents the interface offered by the ACPI Driver to other kernel-mode drivers. Only drivers that support IRP_MJ_PNP requests and IRP_MJ_POWER requests can use the ACPI driver interface.

In general, this interface enables drivers to use the ACPI framework (control methods, ACPI Name Space, event handling, and the Global Lock mechanism). In particular, the interface offered by the ACPI Driver enables drivers to:

- Run control methods found in the device driver's device object in the ACPI name space.
- Acquire and release the Global Lock; this enables a driver to synchronize with the system firmware.
- Register for notification of proprietary device events.
- Handle events associated with a general-purpose event (GPE) bit.

Two Ways of Using the ACPI Driver Interface

A driver uses the ACPI Driver interface in two different, but not mutually exclusive, ways. Both ways, listed below, are documented in this paper.

- Initiate an IRP with a major function code of **IRP_MJ_DEVICE_CONTROL** that contains one of the IOCTL codes documented in this paper.
- Use direct function calls.

The IOCTLs, constants, and so on that are described in this paper are defined in acpiioct.h, which can be found in the Windows 98 DDK.

Constraints on Using the ACPI Driver Interface

There are some constraints on what a driver can do with the ACPI Driver interface. These constraints are shown in the following summary table.

Using the Interface to…	**Type of Call**	**Constraint**
Run a control method.	Initiate an **IRP_MJ_DEVICE_CONTROL**	The control method must be contained in the AML Device object that represents the device controlled by the calling driver. This constraint is discussed in the text following this table.
Acquire and release the global lock.	Initiate an **IRP_MJ_DEVICE_CONTROL**	
Register for notification of device events.	Direct function call	Driver can register for notification only on the AML Device object that represents the device the calling driver controls. Further, it can handle only device notification codes of 80h or higher.
Handle events associated with a GPE bit.	Direct function call	Must use a dedicated GPE bit; cannot share a GPE bit with the operating system or the ACPI driver. Note that if a dedicated GPE bit is not available, a driver can instead register for notification of device events (as shown in the example Hot key driver scenarios below).

Notice that a driver can use IOCTLs to run only certain AML control methods in the ACPI name space. The control method must be under the AML Device object in the name space that corresponds to the device the driver controls. For example, in the hypothetical name space below, the driver associated with the HKEY

device can run the control method _SB.EIO.HKEY.INFO but cannot run the control method _SB.EIO.PS2M.VAL0.

```
\_SB
    .
    .
    .
    EIO                 // Extended I/O Bus
        .
        .
        .
        HKEY            // Hot key Device Object
            _HID        // Unique Device ID that enables the ACPI Driver
                        //   to enumerate the Hot key device
            INFO        // Returns a bitmap that contains the data that
                        //   identifies which hot key was pressed
            _HKGP       // Returns the index of the GPE bit the Hot key
                        //   device is associated with
        PS2M            // PS2 Mouse Device
            _HID        // Hardware Device ID
            _STA        // Status of the PS2 Mouse device
            _CRS        // Current Resource
            VAL0        // Value-added control method
    .
    .
    .
```

Example Using a Hypothetical Hot Key Driver

This section introduces a hypothetical device driver called MyDriver that uses the ACPI Driver interface. MyDriver controls a hot key device that can recognize the keystrokes or keystroke combinations (used particularly on a laptop) that enable the user to use the keyboard to:

- Switch between the local LCD and an external CRT.
- Increase or decrease audio volume.

Notice that this example does not show hot keys used for requesting Power Off or Suspend because, on ACPI platforms, these requests are handled by standard ACPI mechanisms (which are defined in the ACPI Specification).

Important: This example assumes the following:

- On both Windows 98 and Windows NT 5.0, all hot keys must be entirely handled through proprietary drivers and applications.
- For a hot key that switches between the local LCD and an external CTR to work requires exporting a new function from the miniport driver that is called when such events happen. This feature is not currently implemented.

The ACPI Driver interface can be used in two different ways to accomplish MyDriver's goals:

- If a dedicated GPE bit is available, MyDriver can use the **GpeConnectVector** direct function call to associate the driver with the GPE bit, then the driver can handle events associated with that bit.
- As an alternative, MyDriver can use the **RegisterForDeviceNotifications** function call to register a callback routine with the ACPI driver; this callback routine is invoked when AML code executes an AML **Notify** operation on the Hot key device.

Each of these two techniques requires a somewhat different ACPI name space, as is shown below.

Example Driver Setup and Event Handling Using a Dedicated GPE Bit

An example ACPI name space that describes a hot key device that uses a dedicated GPE bit is shown below. The two phases of this process are described in this section:

- During driver setup, MyDriver is loaded and executes the code that registers it as the handler of events on the GPE bit to which the hot key device is connected. This phase processes all the initialization code.

- Event handling is initiated when the user presses a hot key on the keyboard.

This example is based on the following example ACPI name space:

```
\_SB
    .
    .
    .
    EIO                 // Extended I/O Bus
        .
        .
        .
        HKEY            // Hot key Device Object
            _HID        // Unique Device ID that enables the ACPI Driver
                        //   to enumerate the Hot Key device
            INFO        // Returns a bitmap that contains the data that
                        //   identifies which hot key was pressed
            HKGP        // Returns the index of the GPE bit the hot key
                        //   device is associated with
    .
    .
    .
```

Driver Setup

The steps in driver setup are:

1. The unique ID in the _HID object causes the ACPI Driver, in its role as an enumerator, to load MyDriver. (For information about the _HID control method, see section 6.1.3 of the *ACPI Specification*.)

When MyDriver is started, it initiates an **IRP_MJ_DEVICE_CONTROL** IRP with the IOCTL code of **IOCTL_ACPI_EVAL_METHOD** that runs the HKGP control method. (For more information about the control method evaluation IOCTLs, see "Using the ACPI Interface IOCTLs.") Running the HKGP control method returns the index of the GPE bit to which the Hot Key device is connected.

2. To get pointers to the direct call functions offered by the ACPI driver, MyDriver calls **Query_Interface**. This returns a pointer to the **GpeConnectVector** function. ACPI's GUID must be used: IRP_MJ_PNP, IRP_MN_QUERY_INTERFACE
3. MyDriver calls **GpeConnectVector**, using the bit index returned by evaluating the HKGP method as an argument. When this call returns, MyDriver's callback routine is registered with the ACPI Driver to handle all events on that particular GPE bit.

Event Handling

The steps in the event handling are:

1. The callback routine initiates an **IRP_MJ_DEVICE_CONTROL** request, with the IOCTL code The AsyncEval version must be used here.]This runs the INFO control method in the ACPI name space under the HKEY Device object. The INFO control method returns a bitmap that contains the information that identifies the hot key that was pressed.
2. The callback routine interprets the bitmap and, based on that information, does one of the following:
    - Switches between the local LCD and external CRT.
    - Increases or decreases audio volume.

Example Driver Setup and Event Handling Using Event Notification

An example ACPI name space that describes a Hot key device that uses event notification is shown below. The two phases of this process are described in this section:

- During driver setup, MyDriver is loaded and executes the code that registers the HKEY device for a device notification.
- Event handling is initiated when the user presses a hot key on the keyboard.

This example is based on the following example ACPI name space:

```
\_GPE
    _L06                    // Control method run to handle event on
                            //   bit 0x06 of the GPE register. This control
                            //   method contains the following ASL statement:
                            //     Notify (HKEY, x'80')
    .
    .
    .

\_SB
    .
    .
    .
    EIO                     // Extended I/O Bus
        .
        .
        .
        HKEY                // Hot Key Device Object
            _HID            // Unique Device ID that enables the ACPI Driver
                            //   to enumerate the Hot Key device
            INFO            // Returns a bitmap that contains the data that
                            //   identifies which hot key was pressed
        .
        .
        .
```

Driver Setup

The steps in driver setup are:

1. The unique ID in the _HID object causes the ACPI Driver, in its role as an enumerator, to load MyDriver. (For information about the _HID control method, see section 6.1.3 of the *ACPI Specification*.)
2. To get pointers to the direct call functions offered by the ACPI driver, MyDriver calls **Query_Interface**. This returns a pointer to the **RegisterForDeviceNotifications** function.
3. MyDriver calls **RegisterForDeviceNotifications**, passing in a pointer to PDO of the HKEY device and a pointer to the entry-point of the MyDriver's notification call back routine. When this call returns, MyDriver's notification call back routine has been registered with the ACPI Driver to handle device notifications for the HKEY device.

Event Handling

The steps in event handling are:

1. When an event happens at GPE bit 6, the ACPI Driver runs the _L06 control method, which results in the HKEY driver's registered device notification callback being called with a Notify value of X'80'. This routine can be called at DPC.
2. The AsyncEval version must be used since callback can run at DPC.
3. The callback routine interprets the bitmap and, based on that information, does one of the following:
    - Switches between the local LCD and external CRT.
    - Increases or decreases audio volume.

Using the ACPI Interface IOCTLs

A kernel-mode driver sends an IOCTL to the ACPI driver by initiating an IRP_MJ_DEVICE_CONTROL request. The IOCTL codes recognized by the ACPI driver are listed in the following table.

Notice that for the IOCTL_ACPI_EVAL_METHOD and IOCTL_ACPI_ASYNC_EVAL_METHOD IRPs, which are used to run a control method, that control method must be under the AML Device object in the ACPI name space that represents the device the calling driver controls. For more information about these two IOCTLs, see the topic "Using IOCTLs That Run Control Methods."

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.