ADB Family Reference

This chapter describes the **ADB family**, a collection of software pieces that allows clients to get information about and communicate with hardware devices attached to the Apple Desktop Bus (ADB). The ADB, in turn, communicates with the keyboard, the mouse, and other user-input devices.

Mac OS 8 contains standard keyboard and mouse-handling functions that automatically take care of all required ADB access operations. Applications typically receive keyboard and mouse input by calling the Apple Event Manager, not by calling the ADB family. For complete information about receiving and interpreting keyboard and mouse input, see "Apple Events in Mac OS 8."

The **Apple Desktop Bus** (**ADB**) is an open-collector, low-speed serial bus that connects devices such as keyboards, mice, graphics tablets, and joysticks to the **ADB hardware**, the microcontroller that resides in a central processing unit or in other hardware equipment. Macintosh computers come equipped with one or two ADB connectors. Although a particular model might include two ADB connectors, all models come with only one Apple Desktop Bus. The ADB is Apple Computer's standard interface for input devices such as keyboards and mouse devices.

The ADB family presents ADB services to **ADB clients** (for instance, the keyboard and pointing families as well as applications that run in user space) and to **ADB plug-ins** (software modules, such as drivers for specific microprocessor types, such as the 6100, 7100, or 8100, that understand and maintain information about the relationships of ADB devices to various central processing units).

Note
In subsequent releases of the Mac OS 8, the keyboard and pointing families will become part of the input devices family. ◆

An **ADB device** is any device that can connect to the ADB and meets the design requirements described in the *Apple Desktop Bus Specification*. If you are planning on implementing a device you should read the *Apple Desktop Bus Specification* and the technical note *Macintosh Tech Note ADB–The Untold Story: Space Aliens Ate My Mouse*.

IMPORTANT

Apple Computer, Inc. owns patents on the Apple Desktop Bus (ADB). If you want to manufacture a device that works with the ADB software, you must obtain a license and device handler ID from Apple Computer, Inc. Write to this address:

Apple Software Licensing
Apple Computer, Inc.
1 Infinite Loop
Cupertino, CA 95014

A license includes a copy of the *Apple Desktop Bus Specification*. ▲

Figure 1-1 illustrates how the ADB programming interface, the ADB server, and the ADB plug-ins allows clients to get information about ADB devices.

Figure 1-1 The ADB Family, Its Clients, and Plug-ins

If you are an ADB client application or a pointing family plug-in, you may call either the pointing family or the ADB family programming interface.

If you are implementing a new computer that has the Apple Desktop Bus, you may call the ADB plug-in functions.

ADB Client Constants and Data Types

This section describes the data types and constants in the ADB family client programming interface. A client uses the services of the ADB family and its plug-ins to manage data generated by ADB devices.

ADB Connection ID

An **ADB connection** is a logical path to an ADB device and serves to control access to the device. Clients can obtain an ID for an ADB connection by calling the ADBOpen function (page 1-18).

Clients pass an ADB connection ID to close an ADB connection with the ADBClose function, to retrieve the next autopoll event, obtain and set

- the contents of an ADB register with the ADBGetRegister (page 1-23) and ADBSetRegister (page 1-24) functions respectively
- a handler ID for an ADB connection with the ADBGetHandlerID (page 1-27) and ADBSetHandlerID (page 1-28) functions respectively
- the status bits for ADB register 3 with the ADBGetStatusBits (page 1-30) and ADBSetStatusBits (page 1-31) functions respectively

Furthermore, clients pass the ADB connection to flush an ADB device using the ADBFlush function (page 1-33).

The ADB family defines the ADBConnectionID data type, which is an unsigned 32-bit integer that identifies an ADB connection ID.

Note
The ADB connection type will most likely change in future developer releases of the Mac OS 8. ◆

```
typedef UInt32 ADBConnectionID;
```

ADB Register Contents

Each device connected to the Apple Desktop Bus may provide up to four registers for storing data. These registers are referred to as **ADB device registers**. An ADB device can implement these registers as it chooses; that is, an ADB register does not have to correspond to an actual hardware register on the ADB device. An ADB device is accessed over the ADB by reading from or writing to these registers. Each ADB device register may store between 2 and 8 bytes of data.

The ADB family defines the ADBRegisterContents data type to provide information about the contents of an ADB register.

```
typedef struct ADBRegisterContents ADBRegisterContents;
```

To set the contents of an ADB register, you use the ADBSetRegister function (page 1-24), initializing the length field of the ADB register structure to the number of bytes of valid data. To retrieve the contents of an ADB register, you

can use the ADBGetRegister function (page 1-23), obtaining the ADB register contents structure in the contents parameter.

To retrieve the contents of the next autopoll operation after a specified one, you use the ADBGetNextAutopoll function (page 1-20), obtaining the ADB register contents structure in the contents parameter.

```
struct ADBRegisterContents {
    ByteCount       length;         /* ADB register length */
    Byte            data[8];        /* ADB register data */
};
```

Field descriptions

length — On output, a byte count that specifies the length of an ADB register (that is, the number of bytes of the data field that are valid).

data — On output, up to 8 bytes of data contained in the ADB register. This value specifies the size of the information in the ADB register.

I/O Common Information

The I/O common information structure contains information that is common for all families in modular I/O. Modular I/O defines the IOCommonInfo data type to provide information that is common to all the families in modular I/O.

```
struct IOCommonInfo
{
    IODeviceRef             ref;
    IteratorDescVersion     versionNumber; /* version number of family
                                              specific I/O iterator
                                              data */
};

typedef struct IOCommonInfo IOCommonInfo;
```

Field descriptions

ref — An I/O device reference defined by the IODeviceRef type. Clients use a device reference contained in the ref field to

identify an ADB device whose connection they wish to open in the ADBOpen function (page 1-18).

versionNumber — The version number of ADB-specific iterator data.

I/O Device Reference

The I/O device reference is an ADB family-unique reference number for returned ADB devices. The I/O device reference is unique within a family, not unique across the entire I/O name space. The reference is found in the ref field of the I/O Common Information structure, defined by the IOCommonInfo data type, which is an unsigned 32-bit integer.

```
struct IODeviceRef
{
    UInt32 contents[4];
};

typedef struct IODeviceRef IODeviceRef;
```

Field descriptions

contents — A 4-byte field that describes an ADB-family specific I/O device reference. Clients specify the I/O device reference in the ref field of the ADBOpen function (page 1-18).

ADB I/O Iterator Data

The ADB I/O iterator data provides common information of data for all modular I/O as well as a default address and default handler ID for **<<what?>>**

```
struct ADBIOIteratorData {
    IOCommonInfo    IOCI;               /* common data for all
                                           families */
    Byte            defaultAddress;     /* default address */
    Byte            defaultHandlerID;   /* default handler ID *?
};
```

Field descriptions

IOCI

defaultAddress

defaultHandlerID

ADB Current Iterator Description Version

```
enum {
        kADBCurrentIteratorDescVersion = 1
};
```

ADB Plug-in Defined Data Types

This section describes the ADB plug-in defined constants and data types.

ADB Plug-In Dispatch Table

Each ADB family plug-in must export an ADB plug-in dispatch table, so the ADB family can find the functions it contains. The ADB family calls the Driver and Family Matching Software (DFM) to load each plug-in. Subsequently, the DFM returns a pointer to the plug-in dispatch table.

The ADB plug-in dispatch table is defined by the ADBPluginDispatchTable data type.

```
struct ADBPluginDispatchTable {
    ADBPluginHeader                 header;             /* header */
    ADBPluginValidateHardwarePtr    ValidateHardware    /* validate hardware */
    ADBPluginInitProc               Init;               /* initialize function*/
    ADBPluginSetAutopollDelayProc   SetAutopollDelay;   /* set autopoll delay */
    ADBPluginGetAutopollDelayProc   GetAutopollDelay;   /* get autopoll delay */
    ADBPluginSetAutopollListProc    SetAutopollList;    /* set autopoll list */
    ADBPluginGetAutopollListProc    GetAutopollList;    /* get autopoll list */
    ADBPluginAutopollEnableProc     AutopollEnable;     /* autopoll enable function */
    ADBPluginAutopollDisableProc    AutopollDisable;    /* autopoll disable function */
    ADBPluginResetBusProc           ResetBus;           /* reset bus function */
```

```
    ADBPluginFlushProc              Flush;              /* flush function */
    ADBPluginSetRegisterProc        SetRegister;        /* set register function */
    ADBPluginGetRegisterProc        GetRegister;        /* get register function */
    ADBPluginSetKeyboardListProc    SetKeyboardList;    /* get register function */
};

typedef struct ADBPluginDispatchTable ADBPluginDispatchTable;
```

Field descriptions

header — The ADB plug-in header, defined by the ADBPluginHeader data type (page 1-10).

ValidateHardware — The ADB plug-in defined validate hardware function (page 1-37).

Init — The ADB plug-in defined initialization function (page 1-39).

SetAutopollDelay — The ADB plug-in defined set autopoll delay function (page 1-41).

GetAutopollDelay — The ADB plug-in defined get autopoll delay function (page 1-42).

SetAutopollList — The ADB plug-in defined set autopoll list function (page 1-43).

GetAutopollList — The ADB plug-in defined get autopoll list function (page 1-44).

AutopollEnable — The ADB plug-in defined autopoll enable function (page 1-45).

AutopollDisable — The ADB plug-in defined autopoll disable function (page 1-46).

ResetBus — The ADB plug-in defined reset bus function (page 1-48).

Flush — The ADB plug-in defined flush function (page 1-49).

SetRegister — The ADB plug-in defined set register function (page 1-50).

GetRegister — The ADB plug-in defined get register function (page 1-51).

SetKeyboardList — The ADB plug-in defined set keyboard list function (page 1-53).

ADB Plug-In Header

The ADB family provides an ADB plug-in header, defined by the ADBPluginHeader data type. Plug-ins use the ADB plug-in header in the header field of the ADB plug-in dispatch table (page 1-8).

```
struct ADBPluginHeader {
    UInt32      pluginVersion;  /* version number formatted like a number
                                   version */
    UInt32      reserved1;      /* reserved for use by Apple */
    UInt32      reserved2;      /* reserved for use by Apple */
    UInt32      reserved3;      /* reserved for use by Apple */
};

typedef struct ADBPluginHeader ADBPluginHeader;
```

Field descriptions

version — An unsigned 32-bit integer that specifies the version of the ADB plug-in header. This version number is not up to plug-in developers. It is part of the ADB family plug-in programming interface.

reserved1 — An unsigned 32-bit integer that is reserved for future use by Apple.

reserved2 — An unsigned 32-bit integer that is reserved for future use by Apple.

reserved3 — An unsigned 32-bit integer that is reserved for future use by Apple.

ADB Plug-in Version

The ADB plug-in version enumerator describes the version number of a specific ADB plug-in. The version number appears in the version field of the ADB plug-in header data structure, which is defined by the ADBPluginHeader data type (page 1-10).

```
enum {
    kADBPluginCurrentVersion        /* ADB plug-in version enumerator*/
};
```

ADB Plug-in Function Types

ADBPluginValidateHardwareProc

Before the ADB family calls the initialization function, the ADB family gives a plug-in a registered entry reference **<<open issue: This may become an I/O device reference>>** to a device and calls the validate hardware function provided by the plug-in. The plug-in then determines whether the registered name entry reference is the device expected by the plug-in software.

The function pointer is defined by the ADB family as follows:

<<This function pointer is adapted from Suzanne's and needs to be confirmed by Mike next Monday.>>

```
typedef OSStatus (*ADBValidateHardwareProc) (RegEntryRef *device, Boolean *isMyDevice);
```

For information about creating your own validate hardware function, see the description of the MyADBValidateHardwareProc function (page 1-37).

ADBPluginInitProc

When the ADB family selects a ADB plug-in, it calls the initialization function provided by the plug-in. The plug-in then performs appropriate initialization.

The function pointer is defined by the ADB family as follows:

```
typedef OSStatus (*ADBPluginInitProc) (void);
```

For information about creating your own initialization function, see the description of the MyADBPluginInitProc function (page 1-39).

ADBPluginSetAutopollDelayProc

When the ADB family wants to send a command to a chip to set the interval between autopoll operations, it calls the set autopoll delay function provided by the plug-in. The plug-in sets the delay in a device-specific fashion.

The function pointer is defined by the ADB family as follows:

```
typedef OSStatus (*ADBPluginSetAutopollDelayProc)(Duration delay);
```

For information about creating your own set autopoll delay function, see the description of the MyADBPluginSetAutopollDelayProc (page 1-41).

ADBPluginGetAutopollDelayProc

When the ADB family wants to find out the current interval between autopoll operations, it calls the get autopoll delay function provided by the plug-in. The plug-in then retrieves the autopoll delay in a device-specific fashion.

The function pointer is defined by the ADB family as follows:

```
typedef OSStatus (*ADBPluginGetAutopollDelayProc)(Duration *delay);
```

For information about creating your own set autopoll delay function, see the description of the MyADBPluginGetAutopollDelayProc (page 1-42).

ADBPluginSetAutopollListProc

An **autopoll list** is the set of addresses with registered listeners for autopoll data. When the ADB family wants to set the autopoll list for a specific central processing unit, it calls the set autopoll list function provided by the plug-in. The plug-in then sets the autopoll list in a device-specific manner.

The function pointer is defined by the ADB family as follows:

```
typedef OSStatus (*ADBPluginSetAutopollListProc) (UInt16 addressMask);
```

For information about creating your own set autopoll list function, see the description of the MyADBPluginSetAutopollListProc (page 1-43).

ADBPluginGetAutopollListProc

When the ADB family wants to obtain the set of addresses with registered listeners for autopoll data (called the autopoll list), it calls the get autopoll list function provided by the plug-in. The plug-in then retrieves the autopoll list in a device-specific fashion.

The function pointer is defined by the ADB family as follows:

```
typedef OSStatus (*ADBPluginGetAutopollListProc)(UInt16 *addressMask);
```

For information about creating your own get autopoll list function, see the description of the MyADBPluginGetAutopollListProc function (page 1-44).

ADBPluginAutopollEnableProc

When the ADB family wants to turn on the hardware interrupt on autopoll operations, it calls the autopoll enable function provided by the plug-in. The plug-in then translates this autopoll enabling request in a device-specific fashion.

The function pointer is defined by the ADB family as follows:

```
typedef OSStatus (*ADBPluginAutopollEnableProc)(void);
```

For information about creating your own autopoll enable function, see the description of the MyADBPluginAutopollEnableProc function (page 1-45).

ADBPluginAutopollDisableProc

When the ADB family wants to turn off the hardware interrupt on autopoll operations, it calls the autopoll disable function provided by the plug-in. The

plug-in then translates this autopoll disabling request in a device-specific fashion.

The function pointer is defined by the ADB family as follows:

```
typedef OSStatus (*ADBPluginAutopollDisableProc)(void);
```

For information about creating your own autopoll disable function, see the description of the MyADBPluginAutopollDisableProc function (page 1-46).

ADBPluginResetBusProc

When the ADB family wants to send a reset signal over the bus, it calls the reset bus function provided by the plug-in. The plug-in then translates the reset signal request in a device-specific fashion.

The function pointer is defined by the ADB family as follows:

```
typedef OSStatus (*ADBPluginResetBusProc)(void);
```

For information about creating your own reset bus function, see the description of the MyADBPluginResetBusProc function (page 1-48).

ADBPluginFlushProc

When the ADB family wants to send a flush command, it calls the flush function provided by the plug-in. The plug-in then translates the command in a device-specific fashion.

The function pointer is defined by the ADB family as follows:

```
typedef OSStatus (*ADBPluginFlushProc)(Byte deviceAddress);
```

For information about creating your own flush function, see the description of the MyADBPluginFlushProc function (page 1-49).

ADBPluginSetRegisterProc

When the ADB family expert wants to set the contents of ADB register 0, 1, or 2, it calls the set register function provided by the plug-in. The plug-in then translates the request in a device-specific manner.

The function pointer is defined by the ADB family as follows:

```
typedef OSStatus (*ADBPluginSetRegisterProc) (Byte deviceAddress,
                                              Byte registerNumber,
                                              const ADBRegisterContents *contents);
```

For information about creating your own set register function, see the description of the MyADBPluginSetRegisterProc function (page 1-50).

ADBPluginGetRegisterProc

When the ADB family wants to obtain the contents of any of the ADB registers, it calls the get register function provided by the plug-in. The plug-in then translates the command in a device-specific fashion.

The function pointer is defined by the ADB family as follows:

```
typedef OSStatus (*ADBPluginGetRegisterProc) (Byte deviceAddress,
                                              Byte registerNumber,
                                              ADBRegisterContents *contents);
```

For information about creating your own get register function, see the description of the MyADBPluginGetRegisterProc function (page 1-51).

ADBPluginSetKeyboardListProc

When the ADB family wants to create a list of the ADB addresses that are keyboards (called the **keyboard lis**t), it calls the set keyboard list function provided by the plug-in. The plug-in then translates the command in a device-specific fashion.

The function pointer is defined by the ADB family as follows:

```
typedef OSStatus (*ADBPluginSetKeyboardListProc) (UInt16 addressMask);
```

For information about creating your own set keyboard list function, see the description of the MyADBPluginSetKeyboardListProc function (page 1-53).

ADB Client Functions

Getting Information about ADB Devices

Before opening an ADB connection to a device, clients need to obtain data about devices including data size, default address, and default handler ID. The **device address is a** 4-bit bus address that uniquely identifies devices of the same type.The **handler ID** is an ADB device-specific 8-bit value. Taken together, a device's default address and handler ID uniquely identify the particular data protocol the device uses for communication.

ADBGetDeviceData

Obtains data that is common for all families associated including the data size, its default address, and default handler ID.

```
OSStatus ADBGetDeviceData(
                    ItemCount requestCount,
                    ItemCount *totalCount,
                    ADBIOIteratorData *deviceData );
```

requestCount — On input, an item count that indicates the size requested for an ADB device.

totalCount — On output, a pointer to an item count that indicates how many ADB devices there are.

deviceData — A pointer to an ADB I/O iterator data structure, defined by the ADBIOIteratorData type (page 1-7) which consists of information common to all families and default addresses and handler IDs.

function result — An operating system status code. Your request to obtain data about ADB devices can fail if **<<what kinds of result codes would be returned here?>>** See "ADB Result Codes" (page 1-54) for a list of result codes the ADB family can return.

DISCUSSION

ADBGetDeviceData usually returns an array of 16 bytes, the maximum size expected, so clients do not have to call the function twice. Clients can then peruse the data returned in the deviceData parameter to determine which default address and handler ID they want to use. At that point, they can call the ADBOpen function (page 1-18).

EXECUTION ENVIRONMENT

Reentrant?	Call at secondary interrupt level?	Call at hardware interrupt level?
Yes	No	No

CALLING RESTRICTIONS

The ADBGetDeviceData function cannot be called by hardware interrupt handlers or secondary interrupt handlers.

SEE ALSO

For a discussion of default device addresses and default handler IDs, see "Getting Information about ADB Devices" (page 1-16).

Opening and Closing an ADB Connection

Once clients have

ADBOpen

Opens an ADB connection.

```
OSStatus ADBOpen (const IODeviceRef *ref,
                  ADBConnectionID *connection);
```

ref	On input, a pointer to the I/O device reference of the device whose connection you want to open. Clients obtain this reference in the IOCI field of the ADB I/O Iterator Data structure, which is defined by the ADBIOIteratorData type (page 1-7).
connection	On output, a pointer to an ADB connection ID. This value of type ADBConnectionID (page 1-4) identifies the ADB connection that the ADB family has opened.
function result	An operating system status code. Your request to open an ADB connection can fail if the connection is already open or if the I/O device ID is invalid. If the connection is already open, ADBOpen returns the kADBDeviceBusyErr result code. If the I/O device ID is incorrectly specified, ADBOpen returns the paramErr result code. See "ADB Result Codes" (page 1-54) for a list of result codes the ADB family can return.

DISCUSSION

Once the ADB connection has been opened, the ADB family buffers a small number of autopoll events. **<<Mike, are these standard Apple events or are is this use of the word events a more generic one?>>**

EXECUTION ENVIRONMENT

Reentrant?	Call at secondary interrupt level?	Call at hardware interrupt level?
Yes	No	No

CALLING RESTRICTIONS

The ADBOpen function cannot be called by hardware interrupt handlers or secondary interrupt handlers.

SEE ALSO

To close an ADB connection, you use the ADBClose function (page 1-19).

ADBClose

Closes an ADB connection.

```
OSStatus ADBClose (ADBConnectionID connection);
```

connection	On input, a pointer to an ADB connection ID. This value of type ADBConnectionID (page 1-4) identifies the ADB connection you want the ADB family to close.
function result	An operating system status code. Your request to close an ADB connection can fail if the connection has already been closed. If you incorrectly specify the connection ID of the ADB connection, ADBClose returns the adbInvalidConnectionIDErr result code. See "ADB Result Codes" (page 1-54) for a list of the result codes.

DISCUSSION

The ADBClose function also stops autopoll events.

EXECUTION ENVIRONMENT

Reentrant?	Call at secondary interrupt level?	Call at hardware interrupt level?
Yes	No	No

CALLING RESTRICTIONS

The ADBClose function cannot be called by hardware interrupt handlers or secondary interrupt handlers.

SEE ALSO

To open an ADB connection, you use the ADBOpen function (page 1-18).

Autopolling

Autopolling is the primary method the central processing unit uses to fetch data from ADB devices. Devices cannot initiate transactions; they can only respond to commands from ADB hardware. When a device has new data to transmit, it must wait until a command is issued to some ADB device. Then it should assert a service request (SRQ) by holding the bus low.

Note
Autopolling can be disabled. To disable the service request, clients can set this bit to 0. ◆

The central processing unit then begins polling the addresses that it knows have devices, asking each in turn to send the contents of register 0, until the SRQ is no longer asserted.

This section discusses the ADBGetNextAutopoll function (page 1-20), which client call each time they are waiting for the user to perform an action.

ADBGetNextAutopoll

Returns the next autopoll event.

```
OSStatus ADBGetNextAutopoll (
                    ADBConnectionID connection,
                    Duration timeOut
                    ADBRegisterContents *contents,
                    AbsoluteTime *timestamp, );
```

connection — On input, an ADB connection ID. This value of type ADBConnectionID (page 1-4) identifies the ADB connection for which you want the ADB family to return the next autopoll. event.

timeOut	On input, a duration value that indicates how long the ADB family should wait before an autopoll event occurs. If the duration specified expires before the autopoll event happens, ADBGetNextAutopoll returns a kernelTimeoutErr result code.
contents	On output, a pointer to the contents of the ADB Register. This structure of type ADBRegisterContents (page 1-5) describes the length and size of the data in the autopoll.
timestamp	On output, a pointer to the absolute time the data arrived taken at hardware interrupt time.
function result	An operating system status code. Your request to retrieve the next autopoll event can fail if the autopoll has already been terminated, or if an invalid connection ID has been specified. If a client called the ADBClose function (page 1-19) while ADBGetNextAutopoll is pending (for example, using another thread), the function returns the adbConnectionTerminatedErr status code. If you specify a connection ID incorrectly, ADBGetNextAutopoll returns the adbInvalidConnectionIDErr result code. See "ADB Result Codes" (page 1-54) for a list of the result codes the ADB family can return.

DISCUSSION

You call the ADBGetNextAutopoll function to wait for the user to do something, (such as press a mouse button). ADGGetNextAutopoll doesn't return the contents of the autopoll until the user performs an action. Since theoretically it may be forever until the user does something, the timeOut parameter allows you to specify limits on the time that should elapse before obtaining the next autopoll event.

EXECUTION ENVIRONMENT

Reentrant?	Call at secondary interrupt level?	Call at hardware interrupt level?
Yes	No	No

CALLING RESTRICTIONS

The ADBGetNextAutopoll function cannot be called by hardware interrupt handlers or secondary interrupt handlers.

SEE ALSO

For an overview of the autopolling process, see “Autopolling” (page 1-20).

Getting and Setting the ADB Registers

The ADB family has four ADB device logical registers each of which contains between 2 and 8 bytes of data. The central processing unit can get or set the contents of any one register at a time.

- Register 0, known as the data register, contains the immediate data to be sent to the central processing unit.
- The function of register 1, also known as a data register, is defined by device specification. See the *ADB Specification* for details.

Note
You only need to consult the if you are creating a new piece of ADB hardware. ◆

- Register 2 contains the talk and listen commands.
- Known as the **control register**, register 3 is two-bytes long. It stores the device address, handler ID, and four **status bits**. The ADB specification describes the protocol for changing these fields. All ADB devices must follow this protocol to ensure that dynamic address assignment and address resolution work.

The ADBGetRegister function works on all four registers (0, 1, 2, and 3). The ADBSetRegister function only works on ADB registers 0, 1, and 2.

To change the contents of register 3, you need to use the ADB family functions that get and set its individual fields. The ADBGetHandlerID and ADBSetHandlerID functions, described in "Getting and Setting Handler IDs" (page 1-26) work only on register 3 as do the ADBGetStatusBits and ADBSetStatusBits functions described in "Getting and Setting ADB Status Bits" (page 1-29).

For more detailed information on the contents of register 3, see *Inside Macintosh: Devices*.

ADBGetRegister

Retrieves the contents of an ADB register.

```
OSStatus ADBGetRegister (
                    ADBConnectionID connection,
                    Byte registerNumber,
                    ADBRegisterContents *contents,
                    AbsoluteTime *timestamp);
```

connection — On input, an ADB connection ID. This value of type ADBConnectionID (page 1-4) identifies the ADB connection whose register contents you want to retrieve.

registerNumber — A byte that describes the register (0, 1, 2, or 3) whose contents you want to obtain.

contents — On output, a pointer to the contents of the ADB Register. This structure of type ADBRegisterContents (page 1-5) describes the length and size of the data in the ADB register specified in the registerNumber parameter.

timestamp — On output, a pointer to the absolute time the contents of the register were retrieved taken at hardware interrupt time.

function result — An operating system status code. If you have specified a register number incorrectly, ADBGetRegister returns the paramErr result code. If you have specified an ADB connection ID incorrectly, the function returns the adbInvalidConnectionIDErr result

code. If the specified device doesn't respond, the function returns the adbDeviceTimeoutErr result code. See "ADB Result Codes" (page 1-54) for a list of possible result codes.

DISCUSSION

When you call the ADBGetRegister function, it is the equivalent of sending a **Talk command** over the wire. A Talk command fetches user input or other data from an ADB device. A Talk command requests that the specified device send the contents of a specified register over the bus.

EXECUTION ENVIRONMENT

Reentrant?	Call at secondary interrupt level?	Call at hardware interrupt level?
Yes	No	No

CALLING RESTRICTIONS

The ADBGetRegister function cannot be called by hardware interrupt handlers or secondary interrupt handlers.

SEE ALSO

To set an ADB register, you use the ADBSetRegister function (page 1-24). For an overview of the ADB registers, see "Getting and Setting the ADB Registers" (page 1-22).

ADBSetRegister

Sets the contents of ADB registers 0, 1, or 2.

```
OSStatus ADBSetRegister (
                    ADBConnectionID connection,
                    Byte registerNumber,
                    const ADBRegisterContents *contents);
```

connection	On input, an ADB connection ID. This value of type ADBConnectionID (page 1-4) identifies the ADB connection whose register contents you want to retrieve.
registerNumber	On input, a byte that describes the register (0, 1, or 2) whose contents you want to change.
contents	On input, a pointer to the contents of an ADB Register. This structure of type ADBRegisterContents (page 1-5) describes the length and size of the data in the ADB register specified in the registerNumber parameter.
function result	An operating system status code. If you have specified a register number incorrectly, ADBSetRegister returns the paramErr status code. If you have specified the ADB connection ID incorrectly, the function returns the adbInvalidConnectionIDErr result code. See "ADB Result Codes" (page 1-54) for a list of possible result codes.

DISCUSSION

When a client calls ADBSetRegister, it is the equivalent of sending a **Listen command** over the wire. A Listen command instructs a device to prepare to receive additional data.

Note
ADBSetRegister only sets registers 0, 1, and 2. Use the functions described in "Getting and Setting Handler IDs" (page 1-26) and "Getting and Setting ADB Status Bits" (page 1-29) to set individual fields of register 3. ◆

EXECUTION ENVIRONMENT

Reentrant?	Call at secondary interrupt level?	Call at hardware interrupt level?
Yes	No	No

CALLING RESTRICTIONS

The ADBSetRegister function cannot be called by hardware interrupt handlers or secondary interrupt handlers.

SEE ALSO

To retrieve the contents of any ADB register, you use the ADBGetRegister function (page 1-23). For an overview of the ADB registers, see "Getting and Setting the ADB Registers" (page 1-22).

To set the individual fields of the contents of register 3, you can use the ADBSetHandlerID function (page 1-28) and the ADBSetStatusBits function (page 1-31).

Getting and Setting Handler IDs

The **handler ID** is an ADB device-specific 8-bit value. Taken together, a device's default address and handler ID uniquely identify the particular data protocol the device uses for communication. Devices may support more than one protocol. The handler ID for a device occupies the second byte of ADB register 3. You can use the ADBGetHandlerID (page 1-27) and ADBSetHandlerID (page 1-28) functions, respectively, to obtain and set the handler IDs.

Note
The upper 4 bits of register 3 contain the status bits. To get and set these bits, you use the ADBGetStatusBits and ADBSetStatusBits functions described in "Getting and Setting ADB Status Bits" (page 1-29). The next 4 bits contain the device's address, which cannot be changed via a programming interface. ◆

ADBGetHandlerID

Obtains a handler ID for an ADB connection.

```
OSStatus ADBGetHandlerID (
                    ADBConnectionID connection,
                    Byte *handlerID);
```

connection	On input, an ADB connection ID. This value of type ADBConnectionID (page 1-4) identifies the ADB connection whose handler ID you wish to retrieve.
handlerID	On output, a pointer to a byte that describes the handler ID for ADB register 3.
function result	An operating system status code. If you specify an incorrect connection ID, ADBGetHandlerID returns the adbInvalidConnectionIDErr status code. See "ADB Result Codes" (page 1-54) for a list of possible result codes.

EXECUTION ENVIRONMENT

Reentrant?	Call at secondary interrupt level?	Call at hardware interrupt level?
Yes	No	No

CALLING RESTRICTIONS

The ADBGetHandlerID function cannot be called by hardware interrupt handlers or secondary interrupt handlers.

SEE ALSO

To retrieve the contents of any ADB register, you use the ADBGetRegister function (page 1-23). To set the contents of ADB registers 0, 1, and 2, you use the ADBSetRegister function (page 1-24).For an overview of the ADB registers, see "Getting and Setting the ADB Registers" (page 1-22).

To set the handler ID for an ADB register, you can use the ADBSetHandlerID function (page 1-28). To set the ADBSetStatusBits function (page 1-31). For a discussion of handler IDs, see "Getting and Setting Handler IDs" (page 1-26).

ADBSetHandlerID

Sets the handler ID for an ADB connection.

```
OSStatus ADBSetHandlerID (
                    ADBConnectionID connection,
                    Byte handlerID);
```

connection	On input, an ADB connection ID. This value of type ADBConnectionID (page 1-4) identifies the ADB connection whose handler ID you wish to retrieve.
handlerID	On input, a byte that describes the handler ID for ADB register 3.
function result	An operating system status code. Some handler IDs (for instance 00, FF, FE, and FD) are reserved by the ADB Specification for special functions. ADBSetHandlerID returns the adbReservedHandlerIDErr result code if it is unable to set the handler ID specified. If the device does not support the handler ID you have specified, ADBSetHandlerID returns the adbInvalidHandlerIDErr result code. If you have incorrectly specified the ADB connection ID, the function returns the adbInvalidConnectionIDErr result code. See "ADB Result Codes" (page 1-54) for a list of possible result codes.

DISCUSSION

ADBSetHandlerID calls the ADBGetHandlerID function (page 1-27) to verify that the handler ID you have specified is supported.

EXECUTION ENVIRONMENT

Reentrant?	Call at secondary interrupt level?	Call at hardware interrupt level?
Yes	No	No

CALLING RESTRICTIONS

The ADBSetHandlerID function cannot be called by hardware interrupt handlers or secondary interrupt handlers.

SEE ALSO

To retrieve the contents of any ADB register, you use the ADBGetRegister function (page 1-23). To set the contents of ADB registers 0, 1, and 2, you use the ADBSetRegister function (page 1-24).For an overview of the ADB registers, see "Getting and Setting the ADB Registers" (page 1-22).

To obtain the handler ID for an ADB register, you can use the ADGetHandlerID function (page 1-27). For a detailed discussion of handler IDs, see "Getting and Setting Handler IDs" (page 1-26).

To set the status bits in register 3, you can use the ADBSetStatusBits function (page 1-31). To retrieve the status bits in register 3, you can use the ADBGetStatusBits function (page 1-30). For a detailed discussion of status bits, see "Getting and Setting ADB Status Bits" (page 1-29)

Getting and Setting ADB Status Bits

- The ADBGetStatusBits and ADBSetStatusBits functions get and set the **status bits,** which are the 4 most significant bits of the control register (ADB register 3). The upper four bits of ADB register 3 contain the **status bits**, which consist of a service request enable field, an exceptional event field, and several reserved bits. Figure 1-2 shows the status bits of the ADB register 3.

Figure 1-2 The Status Bits of the ADB Control Register

- The next four bits contain the **device address**, which you cannot change via a programming interface. The **ADB device address** is a 4-bit bus address that uniquely identifies devices of the same type.
- The second byte of register 3 contain the **device handler ID**. The **ADB device handler ID** is an 8-bit value that further identifies the specific device type or its mode of operation. For example, an Apple Standard keyboard has a device handler ID of 1, while an Apple Extended keyboard has a device handler ID of 2.

ADBGetStatusBits

Obtains the status bits of device register 3.

```
OSStatus ADBGetStatusBits (
                    ADBConnectionID connection,
                    Byte *bits);
```

connection	On input, an ADB connection ID. This value of type ADBConnectionID (page 1-4) identifies the ADB connection whose register 3 status bits you wish to retrieve.
bits	On output, a pointer to a byte that contains the 4 most significant bits of the first byte of device register 3.
function result	An operating system status code. If you specify an incorrect connection ID, ADBGetStatusBits returns the adbInvalidConnectionIDErr status code. See "ADB Result Codes" (page 1-54) for a list of possible result codes.

EXECUTION ENVIRONMENT

Reentrant?	Call at secondary interrupt level?	Call at hardware interrupt level?
Yes	No	No

CALLING RESTRICTIONS

The ADBGetStatusBits function cannot be called by hardware interrupt handlers or secondary interrupt handlers.

SEE ALSO

For a thorough discussion of status bits, see “Getting and Setting ADB Status Bits” (page 1-29).

To set the status bits in register 3, you can use the ADBSetStatusBits function (page 1-31).

To retrieve the contents of any ADB register, you use the ADBGetRegister function (page 1-23). To set the contents of ADB registers 0, 1, and 2, you use the ADBSetRegister function (page 1-24).For an overview of the ADB registers, see “Getting and Setting the ADB Registers” (page 1-22).

To set the handler ID for ADB register 3, you can use the ADBSetHandlerID function (page 1-28). To obtain the handler ID for register 3, you can use the ADBGetHandlerID function (page 1-27). For a discussion of handler IDs, see “Getting and Setting Handler IDs” (page 1-26)

ADBSetStatusBits

Sets the status bits in ADB register 3.

```
OSStatus ADBSetStatusBits (
                    ADBConnectionID connection,
                    Byte bits);
```

connection	On input, an ADB connection ID. This value of type ADBConnectionID (page 1-4) identifies the ADB connection whose register 3 status bits you wish to set.
bits	On input, a byte that contains the 4 most significant bits of the first byte of device register 3.
function result	An operating system status code. If you specify an incorrect connection ID, ADBSetStatusBits returns the adbInvalidConnectionIDErr status code. See "ADB Result Codes" (page 1-54) for a list of possible result codes.

EXECUTION ENVIRONMENT

Reentrant?	Call at secondary interrupt level?	Call at hardware interrupt level?
Yes	No	No

CALLING RESTRICTIONS

The ADBSetStatusBits function cannot be called by hardware interrupt handlers or secondary interrupt handlers.

SEE ALSO

For a thorough discussion of status bits, see "Getting and Setting ADB Status Bits" (page 1-29).

To obtain the status bits in register 3, you can use the ADBGetStatusBits function (page 1-30).

To retrieve the contents of any ADB register, you use the ADBGetRegister function (page 1-23). To set the contents of ADB registers 0, 1, and 2, you use the ADBSetRegister function (page 1-24). For an overview of the ADB registers, see "Getting and Setting the ADB Registers" (page 1-22).

To set the handler ID for ADB register 3, you can use the ADBSetHandlerID function (page 1-28). To obtain the handler ID for register 3, you can use the ADBGetHandlerID function (page 1-27). For a discussion of handler IDs, see "Getting and Setting Handler IDs" (page 1-26)

Flushing the ADB

ADBFlush

Flushes the ADB.

```
OSStatus ADBFlush (ADBConnectionID connection);
```

connection	On input, an ADB connection ID. This value of type ADBConnectionID (page 1-4) identifies the ADB connection you wish to flush.
function result	An operating system status code. If you specify an incorrect connection ID, ADBFlush returns the adbinvalidConnectionIDErr status code. See "ADB Result Codes" (page 1-54) for a list of possible result codes.

DISCUSSION

When you call ADBFlush, it is the equivalent of sending a flush command over the bus. What this Flush command does is device specific. (For details on the Flush command, see *Inside Macintosh: Devices*.)

EXECUTION ENVIRONMENT

Reentrant?	Call at secondary interrupt level?	Call at hardware interrupt level?
Yes	No	No

CALLING RESTRICTIONS

The ADBFlush function cannot be called by hardware interrupt handlers or secondary interrupt handlers.

Resetting the ADB

ADBResetBus

Sends a reset signal over the ADB bus.

```
OSStatus ADBResetBus (void);
```

function result An operating system status code. See “ADB Result Codes” (page 1-54) for a list of possible result codes.

DISCUSSION

When you call ADBResetBus, it is the equivalent of sending a SendReset command over the bus. (For details on the SendReset command, see *Inside Macintosh: Devices*.)

Clients can’t always detect a device being added to the ADB, so if a user adds one, the ADB family performs a bus reset by calling ADBResetBus. The effect of a device reset is the destruction of all ADB connections. The ADB family performs the following sequence of actions:

1. sends a device-removed message for all known devices
2. sends a reset signal over the ADB bus
3. scans for all ADB devices and performs address resolution so that each address has at most one device.
4. posts device added notifications for each ADB device found.

Note
ADBResetBus occurs automatically at start-up. You don’t typically need to use ADBResetBus. ◆

EXECUTION ENVIRONMENT

Reentrant?	Call at secondary interrupt level?	Call at hardware interrupt level?
Yes	No	No

CALLING RESTRICTIONS

The ADBResetBus function cannot be called by hardware interrupt handlers or secondary interrupt handlers.

ADB Plug-In Functions

The functions in this section are those that the ADB family provides to its plug-ins and those that ADB plug-ins make available to the ADB family.

Functions Exported by ADB Family

ADB plug-ins call the ADB family when an I/O operation has been completed in one of the following situations:

- The ADB family calls one of plug-in functions requesting some information.
- A plug-in starts an operation and returns.
- Later, when an operation is complete, a plug-in calls one of these functions to notify the family.

ADBFamRequestComplete

Indicates that an ADB family I/O operation has been completed.

```
void ADBFamRequestComplete (OSStatus theStatus);
```

theStatus — On input, an operating system status message that indicates the status of the request that is completing. ADBFamRequestComplete returns the noErr result code if the request has been successfully completed or the ioErr result code if there has been an I/O problem. See "ADB Result Codes" (page 1-54) for a list of possible result codes.

DISCUSSION

The plug-in's interrupt service routine (ISR) calls the ADBFamRequestComplete function when I/O is complete. If the work is done immediately (that is synchronously), the plug-in must call the ADBFamRequestComplete function (page 1-35) before returning.

EXECUTION ENVIRONMENT

Reentrant?	Call at secondary interrupt level?	Call at hardware interrupt level?
Yes	Yes	Yes

CALLING RESTRICTIONS

This function is only called by ADB family plug-ins.

ADBFamAutopollArrived

Indicates that an ADB family autopoll event has arrived.

```
void ADBFamAutopollArrived (
                    Byte deviceAddress,
                    const ADBRegisterContents *contents);
```

deviceAddress — On input, the address of the device from which the autopoll event has arrived.

contents	On input, a pointer to the contents of the ADB register. This information is described in a data structure defined by type ADBRegisterContents (page 1-5).

DISCUSSION

A plug-in uses the ADBFamAutopollArrived function to inform the ADB family that an autopoll event has arrived.

EXECUTION ENVIRONMENT

Reentrant?	Call at secondary interrupt level?	Call at hardware interrupt level?
Yes	Yes	Yes

CALLING RESTRICTIONS

This function is only called by ADB family plug-ins.

ADB Plug-in Defined Functions

Validating Hardware

MyADBPluginValidateHardwareProc

Instructs the plug-in to verify that the ADB device specified by the registry entry reference is the piece of hardware expected.

<<Mike will provide this interface next week>>

```
OSStatus MyADBPluginValidateHardwareProc
                    (RegEntryRef *device,
                    Boolean *isMyDevice);
```

device — On input, a pointer to a registry entry reference, defined by the RegEntryRef data type. This reference identifies the plug-in to be validated. For more on registry entry references, see "About the I/O Architecture."

isMyDevice — On output, a pointer to a Boolean value. If set to true, the expected ADB device identified by the registry entry reference in the device parameter is validated. Otherwise, this parameter is set to false.

function result — An operating system status code. **<<Will there be a no error result code?>>** Your plug-in should return the result code noErr to indicate that its initialization function has been successfully activated. If an error occurs, it should return an appropriate result code, for instance, the paramErr result code if an invalid registry entry reference has been specified. See "ADB Result Codes" (page 1-54) for a list of the result codes the pointing family can return.

DISCUSSION

The ADB family calls the MyPTPluginValidateHardwareProc function before initialization.

The ADBPluginValidateHardwareProc type (page 1-11) defines a pointing family plug-in's validate hardware function.

EXECUTION ENVIRONMENT

Reentrant?	Call at secondary interrupt level?	Call at hardware interrupt level?
No	No	No

CALLING RESTRICTIONS

Only the pointing family calls the MyADBPluginValidateHardwareProc function. This function cannot be called by hardware interrupt handlers or secondary interrupt handlers.

SEE ALSO

To terminate a pointer to a plug-in, you can use the MyADBPluginValidateHardwareProc function (page 1-37).

For general information about implementing your own plug-in defined functions, see "ADB Plug-in Defined Functions" (page 1-37).

Initializing ADB Plug-ins

MyADBPluginInitProc

Instructs the plug-in code to initialize both its software structures and its ADB hardware.

```
OSStatus MyADBPluginInitProc (void);
```

function result An operating system status code. Your plug-in should return the result code noErr to indicate that the initialization function has been successfully activated. See "ADB Result Codes" (page 1-54) for a list of possible result codes.

DISCUSSION

The ADB family calls MyADBPluginValidateHardwareProc before it makes any other calls to the plug-in. At initialization, the plug-in should initialize any autopolling delay to approximately 1/100 second. Then the ADB family calls MyADBPluginInitProc before any of the other ADB plug-in defined functions. At initialization, the plug-in should also disables autopolling and empty the autopoll list.

EXECUTION ENVIRONMENT

Reentrant?	Call at secondary interrupt level?	Call at hardware interrupt level?
No	No	No

CALLING RESTRICTIONS

Only the ADB family calls the MyADBPluginInitializeProc function. This function cannot be called by hardware interrupt handlers or secondary interrupt handlers.

SEE ALSO

The ADBPluginInitProc type (page 1-11) defines the MyADBPluginInitializeProc function.

Delaying Autopolling

The **autopoll delay** is the interval between autopoll commands performed by the microcontroller. When the ADB family sets autopolling delay using the MyADBPluginSetAutopollDelayProc function (page 1-41), the plug-in uses the closest value supported by the hardware, never greater than 16.625 milliseconds. The ADB family calls the MyADBPluginGetAutopollDelayProc function (page 1-42) to return the actual value.

MyADBPluginSetAutopollDelayProc

Instructs the plug-in code to set the autopoll delay.

```
OSStatus MyADBPluginSetAutopollDelayProc (Duration delay);
```

delay	On input, a duration that specifies the autopoll interval, that is, the closest value for delay supported by the hardware.
function result	An operating system status code. Your plug-in should return the result code noErr to indicate that the set autopoll delay function has been successfully activated. If an error occurs, it should return an appropriate result code, for instance, the paramErr result code if the ADB family has specified an invalid duration. See "ADB Result Codes" (page 1-54) for a list of possible result codes.

DISCUSSION

The ADBPluginSetAutopollDelayProc type (page 1-12) defines the ADB plug-in set autopoll delay function.

EXECUTION ENVIRONMENT

Reentrant?	Call at secondary interrupt level?	Call at hardware interrupt level?
No	No	No

CALLING RESTRICTIONS

Only the ADB family calls the MyADBPluginSetAutopollDelayProc function. This function cannot be called by hardware interrupt handlers or secondary interrupt handlers.

SEE ALSO

To instruct a plug-in to obtain an autopoll delay, the ADB family can use the MyADBPluginGetAutopollDelayProc function (page 1-42). For details on autopoll delay, see "Delaying Autopolling" (page 1-40).

MyADBPluginGetAutopollDelayProc

Instructs the plug-in to obtain an actual autopoll delay.

```
OSStatus MyADBPluginGetAutopollDelayProc (Duration *delay);
```

delay	On output, a pointer to a duration that specifies the autopoll interval. The plug-in uses the closest value for delay supported by the hardware.
function result	An operating system status code. Your plug-in should return the result code noErr to indicate that its get autopoll delay function has been successfully activated. If an error occurs, it should return an appropriate result code, for instance, the paramErr result code if the ADB family has specified an invalid duration. See "ADB Result Codes" (page 1-54) for a list of possible result codes.

DISCUSSION

The ADBPluginGetAutopollDelayProc type (page 1-12) defines the ADB plug-in get autopoll delay function.

EXECUTION ENVIRONMENT

Reentrant?	Call at secondary interrupt level?	Call at hardware interrupt level?
No	No	No

CALLING RESTRICTIONS

Only the ADB family calls the MyADBPluginGetAutopollDelayProc function. This function cannot be called by hardware interrupt handlers or secondary interrupt handlers.

SEE ALSO

To instruct a plug-in to set an autopoll delay, the ADB family can use the MyADBPluginSetAutopollDelayProc function (page 1-41). For details on autopoll delay, see "Delaying Autopolling" (page 1-40).

Setting and Getting the Autopoll List

An **autopoll list** is the set of addresses with devices opened. When trying to clear a Service Request on the bus, the microcontroller first polls the addresses with their bits set to 1, which will be addresses with which a client has called the ADBOpen function (page 1-18).

ADBPluginSetAutopollList

Instructs the plug-in to set the autopoll list.

```
OSStatus ADBPluginSetAutopollList(UInt16 addressMask);
```

addressMask	On input, an unsigned 16-bit integer that represents an address mask. Each bit of the address describes an ADB address with the least significant bit at address 0 and the most significant bit at address 15.
function result	An operating system status code. Your plug-in should return the result code noErr to indicate that its set autopoll list function has been successfully activated. See "ADB Result Codes" (page 1-54) for a list of possible result codes. **<<Mike to check low-numbered errors for next week.>>**

DISCUSSION

The ADBPluginSetAutopollListProc type (page 1-12) defines the ADB plug-in set autopoll list function.

EXECUTION ENVIRONMENT

Reentrant?	Call at secondary interrupt level?	Call at hardware interrupt level?
No	No	No

CALLING RESTRICTIONS

Only the ADB family calls the MyADBPluginSetAutopollListProc function. This function cannot be called by hardware interrupt handlers or secondary interrupt handlers.

SEE ALSO

To instruct a plug-in to obtain an autopoll list, the ADB family can use the MyADBPluginGetAutopollListProc function (page 1-44). For details on autopoll lists, see "Setting and Getting the Autopoll List" (page 1-43).

MyADBPluginGetAutopollListProc

Instructs the plug-in to obtain the autopoll list.

```
OSStatus MyADBPluginGetAutopollListProc (UInt16 *addressMask);
```

addressMask — On output, a pointer to an unsigned 16-bit integer that represents an address mask. Each bit of the address describes an ADB address with the least significant bit at address 0 and the most significant bit at address 15.

function result — An operating system status code. Your plug-in should return the result code noErr to indicate that its get autopoll list function has been successfully activated. If an error occurs, it should return an appropriate result code, for instance, paramErr. See "ADB Result Codes" (page 1-54) for a list of possible result codes.

DISCUSSION

The ADBPluginGetAutopollListProc type (page 1-13) defines the ADB plug-in get autopoll list function.

EXECUTION ENVIRONMENT

Reentrant?	Call at secondary interrupt level?	Call at hardware interrupt level?
No	No	No

CALLING RESTRICTIONS

Only the ADB family calls the MyADBPluginGetAutopollListProc function. This function cannot be called by hardware interrupt handlers or secondary interrupt handlers.

SEE ALSO

To instruct a plug-in to set an autopoll list, the ADB family can use the MyADBPluginSetAutopollListProc function (page 1-43). For details on autopoll lists, see "Setting and Getting the Autopoll List" (page 1-43).

Enabling and Disabling Autopolling

The ADB family calls the MyADBPluginAutopollEnableProc and MyADBPluginAutopollDisableProc functions to instruct the plug-in to turn autopolling on and off respectively.

MyADBPluginAutopollEnableProc

Instructs the plug-in to turn on autopolling

```
OSStatus MyADBPluginAutopollEnableProc (void);
```

function result An operating system status code. Your plug-in should return the result code noErr to indicate that its autopoll enable function has been successfully activated. See "ADB Result Codes" (page 1-54) for a list of possible result codes.

DISCUSSION

The ADBPluginAutopollEnableProc type (page 1-13) defines the ADB plug-in enable autopolling function.

EXECUTION ENVIRONMENT

Reentrant?	Call at secondary interrupt level?	Call at hardware interrupt level?
No	No	No

CALLING RESTRICTIONS

Only the ADB family calls the MyADBPluginAutopollEnableProc function. This function cannot be called by hardware interrupt handlers or secondary interrupt handlers.

SEE ALSO

To instruct a plug-in to turnoff autopolling, the ADB family can use the MyADBPluginAutopollDisableProc function (page 1-46).

For details on enabling and disabling autopolling, see "Enabling and Disabling Autopolling" (page 1-45).

MyADBPluginAutopollDisableProc

Instructs the plug-in to turn off autopolling.

```
OSStatus MyADBPluginAutopollDisableProc (void);
```

function result An operating system status code. Your plug-in should return the result code noErr to indicate that its autopoll disable function has been successfully activated. See "ADB Result Codes" (page 1-54) for a list of possible result codes.

DISCUSSION

The ADBPluginAutopollDisableProc type (page 1-13) defines the ADB plug-in disable autopolling function.

EXECUTION ENVIRONMENT

Reentrant?	Call at secondary interrupt level?	Call at hardware interrupt level?
No	No	No

CALLING RESTRICTIONS

Only the ADB family calls the MyADBPluginAutopollDisableProc function. This function cannot be called by hardware interrupt handlers or secondary interrupt handlers.

SEE ALSO

To instruct a plug-in to turn on autopolling, the ADB family can use the MyADBPluginAutopollEnable function (page 1-45).

For details on enabling and disabling autopolling, see "Enabling and Disabling Autopolling" (page 1-45).

Resetting the ADB Bus

MyADBPluginResetBusProc

Instructs the plug-in to reset the ADB bus.

```
OSStatus ADBPluginResetBus (void);
```

function result An operating system status code. An operating system status code. Your plug-in should return the result code noErr to indicate that its flush function has been successfully activated. See "ADB Result Codes" (page 1-54) for a list of possible result codes.

DISCUSSION

The ADBPluginResetBusProc type (page 1-14) defines the ADB plug-in reset bus function.

EXECUTION ENVIRONMENT

Reentrant?	Call at secondary interrupt level?	Call at hardware interrupt level?
No	No	No

CALLING RESTRICTIONS

Only the ADB family calls the MyADBPluginResetBusProc function. This function cannot be called by hardware interrupt handlers or secondary interrupt handlers.

SEE ALSO

For a description of resetting the ADB, see "Resetting the ADB" (page 1-34).

Flushing ADB Devices

MyADBPluginFlushProc

Instructs a plug-in to flush an ADB device.

```
OSStatus MyADBPluginFlushProc (Byte deviceAddress);
```

deviceAddress	On input, a byte that indicates the address of the device the ADB family wants to flush.
function result	An operating system status code. Your plug-in should return the result code noErr to indicate that its flush function has been successfully activated. If an error occurs, it should return an appropriate result code, for instance, the paramErr result code if the ADB family has specified an invalid device address. See "ADB Result Codes" (page 1-54) for a list of possible result codes.

DISCUSSION

The ADBPluginFlushProc type (page 1-14) defines the ADB plug-in flush function.

EXECUTION ENVIRONMENT

Reentrant?	Call at secondary interrupt level?	Call at hardware interrupt level?
No	No	No

CALLING RESTRICTIONS

Only the ADB family calls the MyADBPluginFlushProc function. This function cannot be called by hardware interrupt handlers or secondary interrupt handlers.

Setting and Getting the ADB Plug-in Register

The ADB family calls the MyADBPluginSetRegisterProc and MyADBPluginGetRegisterProc functions to instruct a plug-in to set and get the contents of a register on an ADB device.

MyADBPluginSetRegisterProc

Instructs the plug-in to set the contents of an ADB register.

```
OSStatus MyADBPluginSetRegisterProc (
                    Byte deviceAddress,
                    Byte registerNumber,
                    const ADBRegisterContents *contents);
```

deviceAddress — On input, a byte that indicates the address of the device whose register the ADB family wants to set.

registerNumber — On input, a byte that describes the register whose contents the ADB family wants to set.

contents — On input, a pointer to the contents of the ADB Register. This structure of type ADBRegisterContents (page 1-5) describes the length and size of the data in the ADB register specified in the registerNumber parameter.

function result — An operating system status code. Your plug-in should return the result code noErr to indicate that its set register function has been successfully activated. If an error occurs, it should return an appropriate result code, for instance, the paramErr result code if the ADB family has specified an invalid device address or register number. See "ADB Result Codes" (page 1-54) for a list of possible result codes.

DISCUSSION

The ADBPluginSetRegisterProc type (page 1-15) defines the ADB plug-in set register function.

EXECUTION ENVIRONMENT

Reentrant?	Call at secondary interrupt level?	Call at hardware interrupt level?
No	No	No

CALLING RESTRICTIONS

Only the ADB family calls the MyADBPluginSetRegisterProc function. This function cannot be called by hardware interrupt handlers or secondary interrupt handlers.

SEE ALSO

To instruct a plug-in to obtain the contents of a specified register on a particular ADB device, the ADB family calls MyADBPluginGetRegisterProc (page 1-51).

MyADBPluginGetRegisterProc

Instructs the plug-in to retrieve the contents of an ADB register for a specified device.

```
OSStatus MyADBPluginGetRegisterProc (
                    Byte deviceAddress,
                    Byte registerNumber,
                    ADBRegisterContents *contents);
```

deviceAddress — On input, a byte that indicates the address of the device whose register the ADB family wants to get.

registerNumber — On input, a byte that describes the register whose contents the ADB family wants to get.

contents — On output, a pointer to the contents of the ADB Register. This structure of type ADBRegisterContents (page 1-5) describes the length and size of the data in the ADB register specified in the registerNumber parameter.

function result An operating system status code. Your plug-in should return the result code noErr to indicate that its get register function has been successfully activated. If an error occurs, it should return an appropriate result code, for instance, the paramErr result code if the ADB family has specified an invalid device address or register number. If there is no response from the specified device address, your plug-in should return the kADBTimeout result code. See "ADB Result Codes" (page 1-54) for a list of possible result codes.

DISCUSSION

The ADBPluginGetRegisterProc type (page 1-15) defines the ADB plug-in get register function.

EXECUTION ENVIRONMENT

Reentrant?	Call at secondary interrupt level?	Call at hardware interrupt level?
No	No	No

CALLING RESTRICTIONS

Only the ADB family calls the MyADBPluginAGetRegisterProc function. This function cannot be called by hardware interrupt handlers or secondary interrupt handlers.

SEE ALSO

To instruct a plug-in to set the contents of a specified register on a particular ADB device, the ADB family calls MyADBPluginSetRegisterProc (page 1-50).

Setting the Keyboard List

So that the user can reset the central processing unit, the ADB microcontroller scans the power key on keyboards. The MyADBPluginSetKeyboardList function instructs the plug-in to tell the microcontroller which device addresses are keyboards. The **keyboard list** is a set of device addresses that are keyboards.

MyADBPluginSetKeyboardList

Instructs the plug-in to create a device-specific list of the ADB addresses that are keyboards.

```
OSStatus MyADBPluginSetKeyboardList (UInt16 addressMask);
```

addressMask	On input, an unsigned 16-bit integer that represents an address mask. Each bit of the address describes an ADB address with the least significant bit at address 0 and the most significant bit at address 15.
function result	An operating system status code. Your plug-in should return the result code noErr to indicate that its set keyboard list function has been successfully activated. If an error occurs, it should return an appropriate result code, for instance, the paramErr result code if the ADB family has specified an invalid address mask. See "ADB Result Codes" (page 1-54) for a list of possible result codes.

DISCUSSION

When the ADB family wants to create a list of the ADB addresses that are keyboards, it calls MyADBPluginSetKeybordListProc.

The ADBPluginSetKeyboardListProc type (page 1-15) defines the ADB plug-in set keyboard list function.

EXECUTION ENVIRONMENT

Reentrant?	Call at secondary interrupt level?	Call at hardware interrupt level?
No	No	No

CALLING RESTRICTIONS

Only the ADB family calls the MyADBPluginASetKeyboardListProc function. This function cannot be called by hardware interrupt handlers or secondary interrupt handlers.

SEE ALSO

See “Setting the Keyboard List” (page 1-52) for a definition of the keyboard list and details on its use.

ADB Result Codes

<<more to come and to be renumbered next week>>

Many ADB family functions return result codes. The various result codes specific to the ADB family are listed here.

adbDeviceBusyErr	-30279	ADB device has already been opened.
adbInvalidConnectionIDErr	-30278	ADB connection ID has been incorrectly specified.
adbConnectionTerminatedErr	-30277	ADB device has been closed or unplugged during the call.
adbDeviceTimeoutErr	-30276	ADB device has timed out.
adbReservedHandlerIDErr	-30275	Specified ADB handler ID has already been reserved.
adbInvalidHandlerIDErr	-30274	ADB handler ID has been incorrectly specified.

Glossary

ADB connection A logical path to an ADB device and serves to control access to the device. Clients can obtain an ID for an ADB connection by calling the ADBOpen function (page 1-18).

ADB control register The name for ADB register 3. Two-bytes in length, the control register stores the device address, handler ID, and four status bits. The ADB specification describes the protocol for changing these fields. See also **ADB device address**, **ADB handler ID**, and **ADB status bits**.

ADB device Any device that can connect to the ADB and meets the design requirements described in the *Apple Desktop Bus Specification*.

ADB device address The second two bits of the ADB control register. You cannot change the ADB device address via a programming interface. This 4-bit

bus address uniquely identifies devices of the same type. See also **ADB control register**.

ADB device registers Up to four registers for storing data provided to each device connected to the Apple Desktop Bus. An ADB device can implement these registers as it chooses; that is, an ADB register does not have to correspond to an actual hardware register on the ADB device. An ADB device is accessed over the ADB by reading from or writing to these registers. Each ADB device register may store between 2 and 8 bytes of data. The ADB family defines the ADBRegisterContents data type (page 1-5) to provide information about the contents of an ADB register.

ADB family The collection of software pieces in the Mac OS 8 modular I/O that presents ADB services to its clients (for instance, the keyboard and pointing families as well as applications that run in user space) and its plug-ins.

ADB hardware The microcontroller that actually performs the communication of the bus to the ADB devices.

ADB handler ID An ADB device-specific 8-bit value in the control register. Taken together, a device's default address and handler ID uniquely identify the particular data protocol the device uses for communication. Devices may support more than one protocol. The handler ID for a device occupies the second byte of the control register (ADB register 3). You can use the ADBGetHandlerID (page 1-27) and ADBSetHandlerID (page 1-28) functions, respectively, to obtain and set the handler IDs. See also ADB **device address** and **ADB control register**.

ADB plug-ins Software modules, such as generic drivers for specific microprocessor types, such as the 6100, 7100, or 8100, that understand and maintain information about the relationships of ADB devices to various central processing units.

ADB status bits The 4 most significant bits of the control register. The upper four bits of ADB register 3 contain the status bits, which consist of a service request enable field, an exceptional event field, and several reserved bits. The ADBGetStatusBits (page 1-30) and ADBSetStatusBits (page 1-31) functions get and set the status bits.

Apple Desktop Bus (ADB) An open-collector, low-speed serial bus that connects user input peripherals such as keyboards, mice, graphics tablets, joysticks, and copy-protection dongles to a central processing unit or to other hardware equipment. Macintosh computers come equipped with one or two ADB connectors. Although a particular model might include two ADB

connectors, all models come with only one Apple Desktop Bus. The ADB is Apple Computer's standard interface for input devices such as keyboards and mouse devices.

autopolling The primary method the ADB hardware uses to fetch data from ADB devices. Devices cannot initiate transactions; they can only respond to commands from ADB hardware.

autopoll delay The interval between autopoll commands performed by the microcontroller. When the ADB family sets autopolling delay using the MyADBPluginSetAutopollDelayProc function (page 1-41), the plug-in uses the closest value supported by the hardware, never greater than 16.625 milliseconds. The ADB family calls the MyADBPluginGetAutopollDelayProc function (page 1-42) to return the actual value.

autopoll list A set of addresses with registered listeners for autopoll data. When the ADB family wants to set the autopoll list for a specific central processing unit, it calls the set autopoll list function (page 1-43) provided by the plug-in.

keyboard list A list of the ADB addresses that are keyboards. When the ADB family wants to create a keyboard list, it calls the set keyboard list function (page 1-53) provided by the plug-in.

Listen command A command that instructs a device to prepare to receive additional data. When a client calls the ADBSetRegister function (page 1-24), it is the equivalent of sending a Listen command over the wire.

Talk command A command that fetches user input or other data from an ADB device. A talk command requests that the specified device send the contents of a specified device register across the bus. When clients call the ADBGetRegister function (page 1-23), it is the equivalent of sending a Talk command over the wire.