External Accessory Framework Reference

Data Management: Device Information

2010-05-11

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Contents

Introduction

Framework	/System/Library/Frameworks/ExternalAccessory.framework
Header file directories	/System/Library/Frameworks/ExternalAccessory.framework/Headers
Companion guide	External Accessory Programming Topics
Declared in	EAAccessory.h EAAccessoryManager.h EASession.h

The External Accessory framework provides support for communicating with external hardware connected to an iOS-based device through the 30-pin dock connector or wirelessly using Bluetooth. Applications that support external accessories must be sure to configure their `Info.plist` file correctly. Specifically, you must include the `UISupportedExternalAccessoryProtocols` key to declare the specific hardware protocols your application supports. For more information about this framework, see *External Accessory Programming Topics*.

Introduction

Classes

PART I

Classes

EAAccessory Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Availability	Available in iOS 3.0 and later.
Declared in	ExternalAccessory/EAAccessory.h

Overview

The `EAAccessory` class provides your application with information about a single connected hardware accessory. You can use the information in this class to determine if your application is able to open a session to a given accessory. After you have an open session, you can also associate a custom delegate with the accessory object to be notified to changes in the accessory state. Your delegate must adopt the `EAAccessoryDelegate` protocol.

You use an accessory object to create an `EASession` object, which itself provides the communications channel to and from the accessory hardware. The accessory object provides information about the communications protocols the accessory supports, along with information about current hardware and firmware revisions.

When deciding whether to connect to an accessory, you should use the accessory's declared protocols to make your determination. The protocols associated with an accessory indicate the types of data the accessory is capable of processing. You may use other properties to help you decide whether or not to connect to an accessory but the list of protocols should be the key factor you consider.

Accessories can be physically connected to the device through the 30-pin dock connector or wirelessly using Bluetooth.

Tasks

Getting Connection Information

`connected` (page 10) *property*
A Boolean value indicating whether the accessory is currently connected to the iOS-based device. (read-only)

`connectionID` (page 10) *property*
The accessory's unique connection ID to the iOS-based device. (read-only)

Getting the Manufacturer-Supplied Attributes

`name` (page 12) *property*
The display name of the accessory. (read-only)

`manufacturer` (page 12) *property*
The name of the accessory's manufacturer. (read-only)

`modelNumber` (page 12) *property*
The model information for the accessory. (read-only)

`serialNumber` (page 13) *property*
The serial number of the accessory. (read-only)

`firmwareRevision` (page 11) *property*
The current firmware version for the accessory. (read-only)

`hardwareRevision` (page 11) *property*
The hardware version of the accessory. (read-only)

`protocolStrings` (page 12) *property*
The communication protocols supported by the accessory. (read-only)

Accessing the Delegate

`delegate` (page 11) *property*
The object that acts as the delegate of the accessory.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

connected

A Boolean value indicating whether the accessory is currently connected to the iOS-based device. (read-only)

```
@property(nonatomic, readonly, getter=isConnected) BOOL connected
```

Availability
Available in iOS 3.0 and later.

Declared In
`EAAccessory.h`

connectionID

The accessory's unique connection ID to the iOS-based device. (read-only)

```
@property(nonatomic, readonly) NSUInteger connectionID
```

Discussion
The connection ID uniquely identifies this accessory to the device. If multiple accessories of the same type are connected to the device, you can use this information to distinguish between them.

The connection ID for an accessory persists only for the duration of the current connection. If the accessory is disconnected and reconnected, a new connection ID is assigned.

Availability
Available in iOS 3.0 and later.

Declared In
`EAAccessory.h`

delegate

The object that acts as the delegate of the accessory.

```
@property(nonatomic, assign) id<EAAccessoryDelegate> delegate
```

Discussion
The delegate receives notifications about changes to the status of the accessory object. The delegate must adopt the `EAAccessoryDelegate` protocol.

Availability
Available in iOS 3.0 and later.

Declared In
`EAAccessory.h`

firmwareRevision

The current firmware version for the accessory. (read-only)

```
@property(nonatomic, readonly) NSString *firmwareRevision
```

Discussion
The format of this string is determined by the accessory manufacturer.

Availability
Available in iOS 3.0 and later.

Declared In
`EAAccessory.h`

hardwareRevision

The hardware version of the accessory. (read-only)

```
@property(nonatomic, readonly) NSString *hardwareRevision
```

Discussion
The format of this string is determined by the accessory manufacturer.

Availability
Available in iOS 3.0 and later.

Declared In
`EAAccessory.h`

manufacturer

The name of the accessory's manufacturer. (read-only)

```
@property(nonatomic, readonly) NSString *manufacturer
```

Availability
Available in iOS 3.0 and later.

Declared In
`EAAccessory.h`

modelNumber

The model information for the accessory. (read-only)

```
@property(nonatomic, readonly) NSString *modelNumber
```

Availability
Available in iOS 3.0 and later.

Declared In
`EAAccessory.h`

name

The display name of the accessory. (read-only)

```
@property(nonatomic, readonly) NSString *name
```

Availability
Available in iOS 3.0 and later.

Declared In
`EAAccessory.h`

protocolStrings

The communication protocols supported by the accessory. (read-only)

```
@property(nonatomic, readonly) NSArray *protocolStrings
```

Discussion
Protocol names are formatted as reverse-DNS strings. For example, the string "`com.apple.myProtocol`" might represent a custom protocol defined by Apple. Manufacturers can define custom protocols for their accessories or work with other manufacturers and organizations to define standard protocols for different accessory types.

The protocol name should be the primary factor in determining whether your application is capable of communicating with a given accessory. You may use other properties to help you decide whether or not to connect to an accessory but the protocol should still be the key factor you consider. If your application supports multiple protocols for a single accessory, your code should always choose the highest-fidelity protocol that you support.

Availability
Available in iOS 3.0 and later.

Declared In
`EAAccessory.h`

serialNumber

The serial number of the accessory. (read-only)

```
@property(nonatomic, readonly) NSString *serialNumber
```

Availability
Available in iOS 3.0 and later.

Declared In
`EAAccessory.h`

Constants

Null Connection ID

Identifies an unconnected accessory.

```
enum {
    EAConnectionIDNone = 0,
};
```

Constants
`EAConnectionIDNone`
Indicates an invalid connection.
Available in iOS 3.0 and later.
Declared in `EAAccessory.h`.

EAAccessoryManager Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Availability	Available in iOS 3.0 and later.
Declared in	ExternalAccessory/EAAccessoryManager.h

Overview

The `EAAccessoryManager` class coordinates the attached accessories for an iOS-based device. You use this class to retrieve a list of accessories to which your application might want to connect. You also use this class to start and stop the sending of accessory-related connect and disconnect notifications.

Tasks

Getting the Shared Accessory Manager

`+ sharedAccessoryManager` (page 16)
Returns the shared `EAAccessoryManager` object for the iOS-based device.

Starting and Stopping Accessory Notifications

`- registerForLocalNotifications` (page 17)
Begins the delivery of accessory-related notifications to the current application.

`- unregisterForLocalNotifications` (page 17)
Stops the delivery of accessory-related notifications to the current application.

Getting the Available Accessories

`connectedAccessories` (page 16) *property*
The accessory objects corresponding to the list of currently connected accessories. (read-only)

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

connectedAccessories

The accessory objects corresponding to the list of currently connected accessories. (read-only)

```
@property (nonatomic, readonly) NSArray *connectedAccessories;
```

Discussion
This property contains an array of `EAAccessory` objects. Each object corresponds to an accessory that is connected and available for your application to use. Because the contents of this property can change dynamically based on the connection and disconnection of accessories, you should not cache the value of this property.

Availability
Available in iOS 3.0 and later.

Declared In
`EAAccessoryManager.h`

Class Methods

sharedAccessoryManager

Returns the shared `EAAccessoryManager` object for the iOS-based device.

```
+ (EAAccessoryManager *)sharedAccessoryManager
```

Return Value
The shared accessory manager object.

Discussion
You should always use this method to obtain the accessory manager object and should not try to create instances directly.

Availability
Available in iOS 3.0 and later.

Declared In
`EAAccessoryManager.h`

Instance Methods

registerForLocalNotifications

Begins the delivery of accessory-related notifications to the current application.

```
- (void)registerForLocalNotifications
```

Discussion
You must call this method if you want to be notified when accessories become connected or disconnected. The system does not send these notifications automatically, so calling this method lets the system know that your application is interested in them. Typically, you would call this method only once early in your application, either before or after configuring your notification observers. When you no longer need to monitor these notifications, you should call the matching `unregisterForLocalNotifications` method.

You can configure your notification observers either before or after calling this method. Because the shared accessory manager is the only object that sends accessory-related notifications, specifying that object or `nil` for the notification sender has the same outcome.

Availability
Available in iOS 3.0 and later.

See Also
- `unregisterForLocalNotifications` (page 17)
- `addObserver:selector:name:object:` (NSNotificationCenter)

Declared In
`EAAccessoryManager.h`

unregisterForLocalNotifications

Stops the delivery of accessory-related notifications to the current application.

```
- (void)unregisterForLocalNotifications
```

Discussion
Typically, you would call this method either when your application exits or when you no longer want to receive accessory-related notifications. Calls to this method must be balanced with a preceding call to the `registerForLocalNotifications` method.

Availability
Available in iOS 3.0 and later.

Declared In
`EAAccessoryManager.h`

Constants

Notification User Info Keys

Keys associated with the userInfo dictionary of accessory notifications.

```
NSString *const EAAccessoryKey;
```

Constants

`EAAccessoryKey`

The value assigned to this key is the `EAAccessory` object whose status changed.

Available in iOS 3.0 and later.

Declared in `EAAccessoryManager.h`.

Notifications

EAAccessoryDidConnectNotification

Posted when an accessory becomes connected and available for your application to use.

The notification object is the shared accessory manager. The `userInfo` dictionary contains an `EAAccessoryKey`, whose value is an `EAAccessory` object representing the accessory that is now connected. Before delivery of this notification can occur, you must call the `registerForLocalNotifications` (page 17) method to let the system know you are interested in receiving this notification.

Availability
Available in iOS 3.0 and later.

Declared In
`EAAccessoryManager.h`

EAAccessoryDidDisconnectNotification

Posted when an accessory is disconnected and no longer available for your application to use.

The notification object is the shared accessory manager. The `userInfo` dictionary contains an `EAAccessoryKey`, whose value is the `EAAccessory` object representing the accessory that was disconnected. Before delivery of this notification can occur, you must call the `registerForLocalNotifications` (page 17) method to let the system know you are interested in receiving this notification.

If your accessory manager has a delegate, the delegate can use the `accessoryDidDisconnect:` (page 25) method to receive this notification instead.

Availability
Available in iOS 3.0 and later.

Declared In
`EAAccessoryManager.h`

EASession Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Availability	Available in iOS 3.0 and later.
Declared in	ExternalAccessory/EASession.h

Overview

The `EASession` class is used to create a communications channel between your application and a connected hardware accessory. When creating a session, you must specify the protocol you wish to use to communicate with the accessory. After initializing an instance of this class, you use the provided output and input streams to transfer data to and from the accessory using that protocol.

After creating a session object, you should immediately retrieve and configure the stream objects provided by the session. Streams send events to their associated delegate to notify it of changes in the stream status. For example, streams notify the delegate when data is waiting to be read or when more space is available for writing data. For more information about how to use stream objects, see *Stream Programming Guide for Cocoa*.

When sending and receiving data using the provided streams, it is your responsibility to ensure the data is formatted according to the specified protocol. The `EASession` class has no knowledge of specific accessory protocols and does not attempt to format the data in any way before or after transferring it.

Tasks

Initializing the Session

`- initWithAccessory:forProtocol:` (page 21)
Initializes the session for the specified accessory and protocol.

Getting Session Information

`accessory` (page 20) *property*
The accessory attached to this session. (read-only)

`protocolString` (page 21) *property*
The protocol being used for communication with the accessory. (read-only)

Getting the Communication Streams

`inputStream` (page 20) *property*
The stream to use for receiving data from the accessory. (read-only)

`outputStream` (page 21) *property*
The stream to use for sending data to the accessory. (read-only)

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

accessory

The accessory attached to this session. (read-only)

```
@property (nonatomic, readonly) EAAccessory *accessory;
```

Availability
Available in iOS 3.0 and later.

Declared In
`EASession.h`

inputStream

The stream to use for receiving data from the accessory. (read-only)

```
@property (nonatomic, readonly) NSInputStream *inputStream;
```

Discussion
This stream is provided for you automatically by the session object but you must configure it if you want to receive any associated stream events. You do this by assigning a delegate to the stream that implements the `stream:handleEvent:` delegate method. This stream handles data transfers asynchronously but delivers stream events on your application's main thread.

For more information on how to receive data using an input stream, see *Stream Programming Guide for Cocoa*.

Availability
Available in iOS 3.0 and later.

Declared In
`EASession.h`

outputStream

The stream to use for sending data to the accessory. (read-only)

```
@property (nonatomic, readonly) NSOutputStream *outputStream;
```

Discussion
This stream is provided for you automatically by the session object but you must configure it if you want to receive any associated stream events. You do this by assigning a delegate to the stream that implements the `stream:handleEvent:` delegate method. This stream handles data transfers asynchronously but always delivers stream events on your application's main thread.

For more information on how to send data using an output stream, see *Stream Programming Guide for Cocoa*.

Availability
Available in iOS 3.0 and later.

Declared In
`EASession.h`

protocolString

The protocol being used for communication with the accessory. (read-only)

```
@property (nonatomic, readonly) NSString *protocolString;
```

Availability
Available in iOS 3.0 and later.

Declared In
`EASession.h`

Instance Methods

initWithAccessory:forProtocol:

Initializes the session for the specified accessory and protocol.

```
- (id)initWithAccessory:(EAAccessory *)accessory forProtocol:(NSString
    *)protocolString
```

Parameters

accessory
The accessory with which you want to communicate. You can get a list of accessory objects from the `EAAccessoryManager` object.

protocolString
The protocol to use when communicating with the accessory. This protocol must be one that the accessory understands. All communications with the accessory are expected to use this protocol.

Return Value
The initialized session object. This method may return `nil` if the accessory does not recognize the specified protocol or there was an error communicating with the accessory.

Discussion
There can be only one session object at a time for a given accessory and protocol combination.

Availability
Available in iOS 3.0 and later.

Declared In
`EASession.h`

Protocols

PART II

Protocols

EAAccessoryDelegate Protocol Reference

Conforms to	NSObject
Availability	Available in iOS 3.0 and later.
Declared in	ExternalAccessory/EAAcessory.h

Overview

The `EAAccessoryDelegate` protocol defines a single method for receiving notifications when the associated `EAAccessory` object is disconnected. Implementation of this method is optional.

Tasks

Responding to Disconnection Events

`- accessoryDidDisconnect:` (page 25)
Tells the delegate that the specified accessory was disconnected from the device.

Instance Methods

accessoryDidDisconnect:

Tells the delegate that the specified accessory was disconnected from the device.

```
- (void)accessoryDidDisconnect:(EAAccessory *)accessory;
```

Parameters

accessory
The accessory that was disconnected.

Discussion
The accessory manager calls this method as a convenience whenever it receives an `EAAccessoryDidDisconnectNotification` (page 18) notification. You can use this method to remove any references to the specified accessory object and to stop any services currently using the accessory.

Because this is a convenience method, your delegate does not also need to register as an observer of the `EAAccessoryDidDisconnectNotification` (page 18) notification. However, if you want your delegate to be notified of newly connected accessories, you should configure it as an observer of the `EAAccessoryDidConnectNotification` (page 18) notification.

Availability
Available in iOS 3.0 and later.

Declared In
`EAAccessory.h`

Document Revision History

This table describes the changes to *External Accessory Framework Reference*.

Date	Notes
2010-05-11	Added a link to the External Accessory Programming Topic document.
2009-07-15	Added an introduction to the framework collection.
2009-02-22	New document describing the classes and methods of the External Accessory framework.

Document Revision History