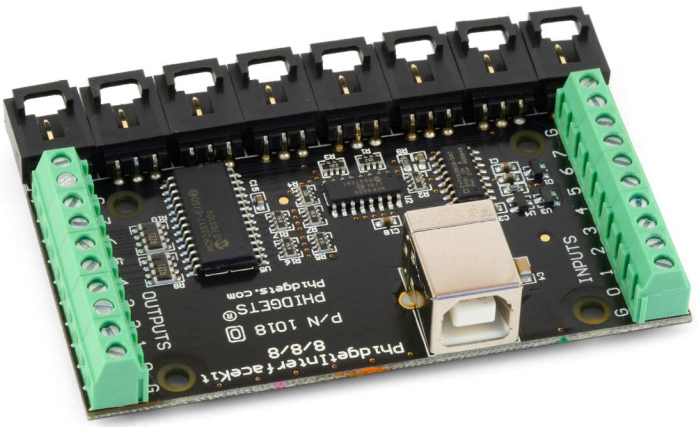
Phidgets

Programming Manual



Phidgets Programming Manual
Version 2.1.4
© Phidgets Inc. 2008

Last updated: May 15, 2008

Contents

Introduction

5 Overview
5 Hardware Model
5 Software Model
6 Operating System Support
6 Platform and Language Support

Phidgets API Concepts

8 Opening Phidgets
8 Initialization
9 Closing Phidgets
9 Multiple Phidgets
9 Exceptions
9 Value Ranges
9 Resolution
10 Threads
10 Phidget Events
10 Change Triggers

Common API

11 Functions
15 Properties
17 Events

Phidget Manager

18 Overview
18 Functions
20 Properties
20 Events

Phidget Dictionary

22 Overview
22 Functions
25 Events

Phidget Webservice

27 Overview
27 Bonjour (mDNS)

28 Properties
29 Events

Appendix I: Programming Concepts

30 Overview
30 Event Based Programming
30 Threading

Appendix II: Regular Expressions

31 Overview
31 Definitions
32 Extended Regular Expressions

Appendix III: Error Codes

Appendix IV: Logging

35 Overview
35 Constants
36 Functions

Introduction

Overview

Phidgets are an easy to use set of building blocks for low cost sensing and control from your PC. Using the Universal Serial Bus (USB) as the basis for all Phidgets, the complexity is managed behind this easy to use and robust Application Program Interface (API) library.

This manual documents the Phidgets software programming model in a language and device unspecific way, providing a general overview of the Phidgets API as a whole.

This should be the first manual to be read for someone beginning to use Phidgets. After the concepts described here are understood, the manual specific to your device should be used for device specific API call reference and device documentation in general. The language API manual for your language of choice should be referenced for language specific API call documentation.

The API calls specified in this manual most closely resemble the .NET library calls. However, all APIs are very similar, and where there are differences it is explicitly mentioned.

Note that the language specific API manuals are light on function documentation - generally only containing language specific reference information and basic API call information.

Hardware Model

All Phidgets are connected to the computer using USB. Most computers support up to 127 USB devices (or more), so it is easy to connect as many Phidgets as are required for almost any project. Phidgets can be connected either directly to a computer or through Hubs, but there are some limitations.

The maximum cable length for USB is 15 feet. This is a maximum distance between device and computer, even if there are one or more Hubs in between. There are cable extenders available on the market, but these can be unreliable and are not endorsed by Phidgets Inc. You should never try to run USB over anything other then a certified USB cable, and you should never try to run it longer than the spec.

Phidgets run as USB 1.1 low speed or full speed devices, and are supported by both USB 1.1 and USB 2.0 hosts.

Software Model

Our software model is to support as many languages and operating systems as are useful for our users, and to make our APIs as simple to use and consistent as possible.

The bottom level of our API is the C library - phidget21. This is a cross-platform library, which implements the low-level protocols necessary to communicate with the Phidgets, and exports a unified interface to the software programmer.

Built upon this low level library are higher level libraries that simplify using Phidgets for many more languages. These higher level libraries contain only glue logic for interfacing with the C library, thus making maintenance much easier.

Available libraries include: COM, .NET, AS3, Java, Python, Max/MSP and MSRS. With these libraries, many languages can be used for interfacing with Phidgets in a variety of programming environments. Specifics are detailed under Platform and Language Support.

Also provided is the Phidget Web Service, which allows Phidgets to be controlled over a network.

It should be noted our libraries employ threading and events extensively. See Appendix I - Programming Concepts for more information.

Operating System Support

Windows

Microsoft Windows 2000 and later are supported, including 64-bit editions. The Windows libraries are installed using an MSI installer that can be found on the Phidgets web site. This installs the C library, the .NET library, the COM library, the Java library, the Phidget Web Service and the Phidget Control Panel.

The Phidget Control Panel is represented by a "Ph" icon that runs in the system tray (usually on the right end of the Windows task bar). This program can be used to list and control any Phidgets attached to the system, and to control the Web Service.

Mac OS X

Mac OS X 10.3.9 and newer on Intel and PPC are supported. The Mac libraries are distributed in a .dmg and are installed using a standard Mac package installer. This installs the C library, the Kernel driver, the Java library, the Phidget Web Service and the Phidget Preference Pane.

The Phidget Preference Pane is a preference pane which resides in System Preferences. This program can be used to list and control any Phidgets attached to the system, and to control the Web Service.

Linux

Linux version 2.4 is supported, but 2.6.7 or newer is recommended. The Linux libraries are distributed as source. The source for the C library, with optional JNI (Java support) extensions and the source for the Phidget Web Service are available as a .tar.gz. The included Makefile makes it easy to build and install the libraries on most Linux distribution.

Windows CE

Windows CE 5.0 and higher are supported. This includes Windows Mobile 5.0 and higher. The .NET compact framework 2.0 is supported as well. Architecture support includes ARMV4 and x86, which covers most Windows CE machines. The libraries are installed using .CAB installers. This installs the C library, the .NET library, the Phidget Web Service, as well as the kernel driver.

Other

Other Operating System support is not currently available.

Platform and Language Support

C/C++

C/C++ is supported on all platforms. Writing code in C requires the C library. See the C API manual for more information

C#

C# is supported on Windows and Windows CE, and can unofficially be used in Linux and Mac OS via Mono. C# requires the .NET library as well as the C library. On Windows, the .NET Framework Version 1.1 and 2.0 are supported. On Windows CE, the .NET Compact Framework 2.0 is supported. See the .NET API Manual for more details.

VB.NET

VB.NET is supported on Windows and Windows CE. VB.NET requires the .NET library. On Windows, the .NET Framework Version 1.1 and 2.0 are supported. On Windows CE, the

.NET Compact Framework 2.0 is supported. See the .NET API Manual for more details.

Visual Basic

Visual Basic 6.0 is supported on Windows. Visual Basic requires the COM library and the C library. See the COM API manual for more details.

Cocoa

Cocoa is supported on Mac OS X via the C library. There is no Cocoa Phidgets library available. See the C API manual and the Cocoa examples for more details.

Java

Java 1.4.2 and higher is supported on Windows, Linux and Mac OS X. Java requires the C library, with JNI extensions compiled in, and the Phidget21.jar Java library file. See the Java API manual for more details.

Python

Python 2.5 is supported on Windows, Linux and Mac OS X. Python requires the Python library and the C library. See the Python API manual for more details.

Delphi

Delphi 7.0 and 2005 are supported on Windows, with other usable versions. Delphi requires the COM library and the C library. See the Delphi examples and Readme as well as the COM API manual for more details.

Actionscript 3.0

Actionscript 3.0 is supported on Windows and Mac OS X via Flex 2.0 and higher, and Flash CS3. Actionscript requires the C library and the Phidget Web Service. See the AS3 API manual for more details.

Max/MSP

Max/MSP 4.5 and higher is supported on Windows and Max OS X. Max/MSP requires the C library and the Max/MSP externals provided by Phidgets Inc. These are available on the website. See the Max/MSP Readme and the provided .help files for more information.

Note the Max/MSP support has not been extended to every Phidget, however the most common Phidgets are supported.

MSRS (Microsoft Robotics Studio)

MSRS 1.5 is supported on Windows. MSRS requires the C library and the .NET library, as well as the MSRS extensions (which are installed using a .MSI installer). Only some Phidgets are supported by MSRS, and support is limited. Download the examples for more information.

LabView

LabView is supported on Windows. LabView requires the C library and the COM library. See the LabView example and the COM API manual for more details.

Other

Other platforms will be supported as they are requested. In the meantime, any system that can call into C libraries should be fairly easy to support on the user's end. All of our libraries and system support are built on top of the C library, as it contains all of the actual logic for talking to Phidgets.

Phidgets API Concepts

Opening Phidgets

The first step in controlling a Phidget is calling `Open()` on it. This will signal the library that you would like to use this Phidget and register it for usage.

Open will return immediately once called, because it can be called even if the Phidget to be used is not attached to the system. This is known as an asynchronous call. It's important to understand that most calls on a Phidget will fail if they are calls when the Phidget is not attached - in fact the only calls that are allowed on a detached Phidget are `Close()`, `waitForAttachment()` and `Attached [get]`.

Once open has been called, there are two options: You can call `waitForAttachment(timeout)`, which will block until either the Phidget is available or until the time out has passed, or you can wait until the Attach event fires. The event based method is recommended, and generally most useful for GUI based applications. The waitForAttachment() method is useful for simple command line applications.

If you decide to use events, you need to register the event handlers before calling `Open()`, or you will miss events.

Once the Phidget is attached, the full API can be used on it.

Open is also pervasive. This means that once open has been called, it will constantly try to stay attached to a Phidget. Even if the Phidget is unplugged from the computer and then plugged back in, you will simply get a Detach event, and then an Attach event. It's a good idea to handle the Detach event in order to avoid calling the Phidget after it has detached.

Phidgets can either be opened with or without using their unique serial number. In the event a serial number is not specified, the first available device will be opened. If there are more than one of the same type of Phidget attached to a computer, there is no way of knowing which of these will be opened. Once a Phidget is opened by an application, it cannot be opened again in another application until closed by the first.

Phidgets can be opened both locally (on the same computer) and remotely (over a network). Opening Phidgets remotely carries several small differences from opening them locally. See the Phidget Web Service section for more information.

Initialization

When a Phidget is opened, its initial state will be read before it is marked as attached. This allows polling of many properties even during the Attach event, and anytime afterwords.

Many properties are filled in automatically when the device is attached, however some are not. Some properties are not returned from the Phidget itself, and so they cannot be recovered. Trying to read any of these uninitialized properties will result in a PhidgetException being thrown. These properties should be set explicitly once a Phidget has attached. The Attach handler is a good place to do this. Set the Product manual for more information on which properties will be initialized at Attach.

Some properties have default values, but these should not be trusted. Remember: always set, don't rely on defaults.

Often Phidgets will retain their last state unless power is lost. This can give surprising results as the previous state may not always be what you expect. For example, if you open an InterfaceKit and set an output, this output may stay set even after the Phidget is closed. If it is opened afterwards, it will report this output as engaged on Attach.

Closing Phidgets

Before your application exits, you should call Close() on all Phidgets, Managers and Dictionaries that you called Open() on. This cleans things up, closes the USB handles, and shuts down all threads properly. Any outstanding writes will block close until they complete, because writes are guaranteed to complete (unless a device is detached).

Also note that a device should be put into a known state before calling close. For example, if a motor controller is driving a motor and close is called, it will continue to drive the motor even though the application has exited. This may or may not be what you want.

Multiple Phidgets

Every Phidget has a serial number, which is unique across all Phidgets. This serial number can be specified to the Open call in order to open a specific Device. This allows any number of the same type of Phidget to be used at the same time on one computer - simply open each device using its unique serial number.

This serial number can be read from an opened device, or you can use the Phidget manager to get a listing of all devices (and their serial numbers). Windows and Mac OS X provide a listing of all attached Phidgets in their Phidget Utilities.

Exceptions

Every function can have error. In C/C++ these errors are returned as a non-zero integer return code. In languages that support exceptions (e.g., Java, .NET, Python, AS3), errors are returned using exceptions which must be caught.

It is very important to catch these exceptions because it is quite common for them to be thrown. For example, anytime a function is called on a Phidget object where the Phidget is not attached, an exception will be thrown. If the Phidget is unplugged from the computer while being written to, and exceptions are not caught, your application will terminate rather than handling the error properly.

In C, this could cause even more problems, because if return codes are never checked, errors could be occurring without anyone the wiser - everything could appear to be fine.

See the Appendix for specific error code documentation.

Value Ranges

Many properties have corresponding Min/Max properties to allow your application to programmatically determine the limitations of the Phidget it has connected to. This reduces the need for 'magic numbers'. This also makes it easier to write code that can absorb changes in functionality for future phidgets. By checking the measured value on a sensor against its Min/Max, it's possible to detect if a sensor may be saturated, and therefore the measured value may not be trusted.

Resolution

Phidgets have finite resolution, so when you write a value, your value will likely be quantized. As an example, the PhidgetLED forces a Brightness value between 0-100 into 64 values. This means the maximum discrepancy between what you wrote and what you will read back/what is used by the Phidget is ~0.75%. The resolution used for many properties will be specified in the Device Specifications section of a Phidget's Product Manual

Threads

Calling open starts a central thread. Closing everything will shut it down (before the final close returns). Each device, once attached, starts it's own read and write threads. Data events come from the context of the read thread. Attach and detach events come from the context of the central thread.

The central thread looks for device attaches and detached, keeping track of which devices are attached internally, and sending out attach and detach events to Phidgets and Managers.

Writes are performed asynchronously by the write thread. The write queue is only 1 deep so calling a write function while there is a write pending will block.

All libraries are thread safe, so you don't need to do any locking on the Phidget objects.

Phidget Events

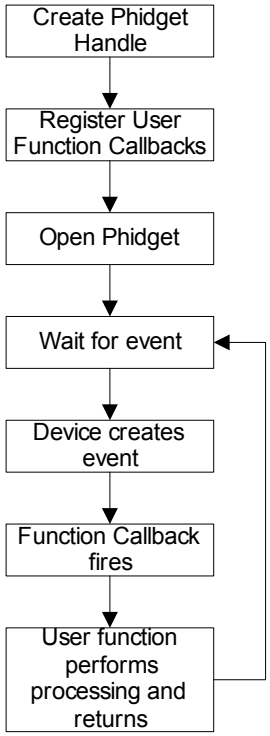
Event Handlers are used to notify your application that a noteworthy event has occurred on a Phidget. By creating a function accepting the proper parameters (an Event Handler), and registering that function with the library, the library is able to call your function whenever events occur.

Because your Event Handler can be run at anytime, it's best to register them before calling Open().

Event Handlers have to conform to a standard that the Phidget library understands. See you language API for specifics.

Every event will contain a reference to the Phidget that raised the event. This allows properties of the Phidget to be evaluated within the Callback.

The Event Type identifies the kind of event being fired, this allows the user to make a choice of how to handle the event.



Setting up an Event Handler involves:

1. Creating up a Phidget object
2. Creating a function to handle your event
3. Passing this function to the event dispatcher
4. Opening the Phidget and waiting for events.

Change Triggers

Change Triggers are used to filter the number of events that are returned to an Application, by setting a minimum amount of activity before a Change Event is sent to the Client. This is a simple hysteresis - a minimum amount of change has to occur since the last event before another event will be fired. If your application is implementing it's own filtering, setting the ChangeTrigger to zero will cause all events to fire. Change triggers are generally available only for sensor inputs events.

Common API

This API is common to all Phidgets. For the networking API, see the Phidget Web Service section. For Device specific API calls, see the Product manual for that Phidget. For Language specific Details, see the API manual for that language. The Phidget Manager and Phidget Dictionary are discussed later in this document.

Functions

create

Creates a Phidget handle/object.

```
void create();
```

Discussion:

Every Phidget has it's own create function. Typically this is the first call on a handle / object, followed by registering the event handlers, and calling the variant of open required.

In object oriented languages, this is carried out by the class constructor - there is no explicit create function.

open

Opens a Phidget attached to this computer.

```
void open(
   int SerialNumber
);
```

Parameters:

SerialNumber

Serial number of the Phidget to open. This parameter is optional, and if not specified, the first available Phidget will be opened. In some languages, there will be a version of open that doesn't take a serial number, in others, -1 can be specified to open any Phidget.

Discussion:

Open is pervasive. What this means is that you can call open on a device before it is plugged in, and keep the device opened across device dis- and re-connections.

Open is Asynchronous. What this means is that open will return immediately – before the device being opened is actually available, so you need to use either the attach event or the waitForAttachment method to determine if a device is available before using it.

The serial number is a unique number assigned to each Phidget during production and can be used to uniquely identify specific Phidgets.

openRemote

Open a Phidget remotely, using a Server ID.

```
void openRemote(
   int SerialNumber,
   string ServerID,
   string Password
);
```

Parameters:

SerialNumber

Serial number of the phidget to open. This parameter is optional, and if not specified, the first available Phidget will be opened. In some languages, there will be a version of open that does not take a serial number. In others, -1 can be specified to open any Phidget.

ServerID

Server ID of the Phidget Web Service. This will be the name of the computer by default. This can be NULL to open this Phidget (by serial number) regardless of what Web Service it is attached to.

Password

Password for the Web Service when running in secure mode. When the Web Service is running without a password, this can be set to NULL, or left out, depending on the Library used.

Discussion:

This version of open is network based.

Open is pervasive. What this means is that you can call open on a device before it is plugged in, and keep the device opened across device connections and disconnections.

Open is Asynchronous. What this means is that open will return immediately – before the device being opened is actually available, so you need to use either the attach event or the waitForAttachment method to determine if a device is available before using it.

ServerID can be set to null, if the ServerID does not matter. In this case, the specified Phidget (by serial number) will be opened whenever it is seen on the network, regardless of which server it appears on. This also applies when not specifying a serial number.

The serial number is a unique number assigned to each Phidget during production and can be used to uniquely identify specific phidgets.

openRemoteIP

Open a Phidget remotely, using an IP Address.

```
void openRemoteIP(
   int SerialNumber,
   int Port,
   string Address,
   string Password
);
```

Parameters:

SerialNumber

Serial number of the Phidget to open. This parameter is optional, and if not specified, the first available Phidget will be opened. In some languages, there will be a version of open that doesn't take a serial number, in others, -1 can be specified to open any Phidget.

Port

Port that the Web Service is running at. By default this is 5001.

Address

Address that the Web Service is running at. This can be a hostname or and IP address.

Password

Password for the Web Service when running in secure mode. This is optional, and when the Web Service is running without a password, this can be set to NULL, or left out, depending on the Library used.

Discussion:

This version of open is network based.

Open is pervasive. What this means is that you can call open on a device before it is plugged in, and keep the device opened across device dis- and re-connections.

Open is Asynchronous. What this means is that open will return immediately – before the device being opened is actually available, so you need to use either the attach event or the waitForAttachment method to determine if a device is available before using it.

The serial number is a unique number assigned to each Phidget during production and can be used to uniquely identify specific phidgets.

close

Closes a Phidget.

```
void close();
```

Discussion:

This will shut down all threads dealing with this Phidget and no more events will be fired. If there are any outstanding writes, they will be processed before the Phidget is closed.

delete

Frees the Phidget handle / object.

```
void delete();
```

Discussion:

Once delete is called, it is no longer safe to calls any other functions on this handle, except for create.

In object oriented languages, this is carried out by the class destructor - there is no explicit delete function.

waitForAttachment

Waits for this Phidget to become available.

```
void waitForAttachment(
   int Timeout
);
```

Parameters:

Timeout
Time to wait for an attach, in milliseconds. A time out of 0 means wait forever.

Discussion:

This method can be called after open has been called to wait for the Phidget to become available. This is useful because open is asynchronous (and thus returns immediately), and most methods will throw a PhidgetException is they are called before a device is actually ready. This method is synonymous with polling the isAttached method until it returns True, or using the Attach event.

This method blocks for the length of the time out, at which point it will throw a PhidgetException. Otherwise, it returns when the phidget is attached and initialized.

Properties

Attached

Gets the attached state of a Phidgets.

```
bool Attached [get]
```

Discussion:

This method returns True or False, depending on whether the Phidget is physically plugged into the computer, initialized, and ready to use - or not. If a Phidget is not attached, many functions calls will fail with a PhidgetException, so either checking this function, or using the Attach and Detach events, is recommended, if a device is likely to be attached or detached during use.

Note that is some APIs, this function is called **`DeviceStatus()`**.

Type

Gets the device type of a Phidget.

```
string Type [get]
```

Discussion:

This is a string that describes the device as a class of devices. For example, all PhidgetInterfaceKit Phidgets will returns the String "PhidgetInterfaceKit".

Name

Gets the name of a Phidget.

```
string Name [get]
```

Discussion:

This is a string that describes the device. For example, a PhidgetInterfaceKit could be described as "Phidget InterfaceKit 8/8/8", or "Phidget InterfaceKit 0/0/4", etc., depending on the specific device.

This lets you determine the specific type of a Phidget, within the broader classes of Phidgets, such as PhidgetInterfaceKit, or PhidgetServo

Version

Gets the device version of a Phidget.

```
int Version [get]
```

Discussion:

This number is simply a way of distinguishing between different revisions of a specific type of Phidget, and is only really of use if you need to troubleshoot device problems with Phidgets Inc.

SerialNumber

Gets the unique serial number of a Phidget.

```
int SerialNumber [get]
```

Discussion:

This number is set during manufacturing, and is unique across all Phidgets. This number can be used in calls to open to specify this specific Phidget to be opened.

Label

Gets or Sets the label associated with this Phidget.

```
string Label [get,set]
```

Discussion:

This label is a String - up to ten digits - that is stored in the Flash memory of newer Phidgets. This label can be set programmatically (see setDeviceLabel), and is non-volatile - so it is remembered even if the Phidget is unplugged.

Labels can not currently be set from Windows because of driver incompatibility. Labels can be set from MacOS, Linux and Windows CE.

LibraryVersion

Returns the library version.

```
string LibraryVersion [get]
```

Discussion:

This generally either returns the version of the low level C library, or the version of a higher level library if available. See API documentation for details.

The version is returned at a string which contains the version number and build date.

ErrorDescription

Gets the description of an error code.

```
string ErrorDescription(
   int ErrorCode
) [get]
```

Parameters:

ErrorCode
The error code. See Appendix III for a list of error codes.

Discussion:

Note that some languages have a PhidgetException class, from which to call this function.

Events

Attach

Fired on device attach.

```
event Attach
```

Discussion:

Fired when this Phidget is ready to be used after being physically attached to the system and has gone through initialization. The attach event is guaranteed to be the first event that fires when a Phidget is first plugged in or opened. No other events will fire for this Phidget until the Attach event returns. The initial device state (sensor values, serial number, etc.) will be available in the Attach event, but be aware that some properties may remain uninitialized. See the Product manuals for more information.

Detach

Fired on device detach.

```
event Detach
```

Discussion:

Fired when this Phidget is physically detached from the system, and is no longer available. This is particularly useful for applications when a physical detach would be expected.

Remember that many of the methods, if called on an unattached device, will throw a PhidgetException. This Exception can be checked to see if it was caused by a device not being attached, but a better method would be to register the detach handler, which could notify the main program logic that the device is no longer available, disable GUI controls, etc.

Error

Fired when an asynchronous error occurs.

```
event Error(
   int ErrorCode,
   string ErrorDescription
)
```

Parameters:

ErrorCode

The error code for this event.

ErrorDescription

The error description string for the event.

Discussion:

Error events are currently only used with certain devices (such as the PhidgetMotorControl) to report device error conditions. Error events are also used when using the Phidget Web Service to report asynchronous network errors.

Phidget Manager

Overview

The Phidget manager is an interface that allows for monitoring of all phidgets connected to a system, without opening them. This can be useful if you don't know ahead of time how many and/or what types of Phidgets will be attached to the system.

Phidgets can be probed for basic device information (SerialNumber, Type, Name, Version and Label) without being opened, or they can be opened as they are detected for further interaction.

Multiple Phidget managers can be created at once, and they can be opened both locally for this computer, or remotely over the network.

Functions

create

Creates a Phidget Manager.

```
void create();
```

Discussion:

Typically this is the first call on a handle / object, followed by registering the event handlers, and calling the variant of open required.

In object oriented languages, this is carried out by the class constructor - there is no explicit create function.

open

Opens a Phidget manager for this computer.

```
void open();
```

Discussion:

This instructs the Phidget manager to start looking for Phidgets. If using events, they should be registered before open is called, because attach events for all currently attached Phidgets will be issued immediately.

The manager will continue to report attach and detach events until it is closed.

openRemote

Open a Phidget Manager remotely, using a Server ID.

```
void openRemote(
   string ServerID,
   string Password
);
```

Parameters:

ServerID

Server ID of the Phidget Web Service. This will be the name of the computer by default. This can be NULL to open this manager for all Phidgets on the network.

Password

Password for the Web Service when running in secure mode. This can be set to NULL when the Web Service is running without a password, or left out depending on the Library used.

Discussion:

This version of open is network based.

Open is Asynchronous. What this means is that open will return immediately – before the server being connected is actually available, and will be able to re-establish connection to a server that goes down, automatically.

ServerID can be set to NULL if the ServerID does not matter. In this case, the manager will report Phidget attach and detach events from every Web Service on the network that is setup for zeroconf.

openRemoteIP

Open a Phidget Manager remotely, using an IP Address.

```
void openRemoteIP(
   int Port,
   string Address,
   string Password
);
```

Parameters:

Port

Port that the Web Service is running at. By default this is 5001.

Address

Address that the Web Service is running at. This can be a hostname or and IP address.

Password

Password for the Web Service when running in secure mode. This is optional, and when the Web Service is running without a password, this can be set to NULL, or left out, depending on the Library used.

Discussion:

This version of open is network based.

Open is Asynchronous. What this means is that open will return immediately (before the server being connected is actually available), and will be able to re-establish connection to a server that goes down automatically.

close

Closes this Phidget manager.

```
void close();
```

Discussion:

This will shut down all threads dealing with this manager and no more events will be fired.

delete

Frees the manager handle / object.

```
void delete();
```

Discussion:

Once delete is called, it is no longer safe to calls any other functions on this handle, except for create.

In object oriented languages, this is carried out by the class destructor - there is no explicit delete function.

Properties

Devices

Gets an array of Phidget objects / handles representing all devices currently attached.

```
array Devices [get]
```

Discussion:

These objects cannot actually be used to control the devices they represent, as they have not had open called on them, but they can be used to get device info - name, type, serial number, etc.

This function is called **AttachedDevices** in some languages.

Events

Attach

Fired when any Phidget is physically attached to the system.

```
event Attach(
   phidget Device
)
```

Parameters:

Device

The Phidget that was attached. Only Common API functions can be called on this device (get SerialNumber, Type, Name, etc.). You can call open on this device within this attach handler, but must wait until the handler returns before it will attach.

Discussion:

Since this is a manager attach, the device returned can only be used for getting basic device information such as name, type, serial number, etc. In order to control this device, a new handle should be created and open called with it's serial number.

Note that blocking in Attach and Detach events will suppress all other Attach and Detach events for both Phidgets and other Managers.

Detach

Fired when any Phidget is physically detached from the system, and is no longer available.

```
event Detach(
   phidget Device
)
```

Parameters:

Device

The Phidget that was detached. Only Common API functions can be called on this device (get SerialNumber, Type, Name, etc.).

Error

Fired when an asynchronous error occurs.

```
event Error(
   int ErrorCode,
   string ErrorDescription
)
```

Parameters:

ErrorCode

The error code for this event.

ErrorDescription

The error description string for the event.

Discussion:

Error events are used when using the Phidget Web Service, to report asynchronous network errors.

Phidget Dictionary

Overview

The Phidget Dictionary is a service provided by the Phidget Web Service. The Web Service maintains a centralized dictionary of key-value pairs that can be accessed and changed from any number of clients.

Note that the Web Service uses this dictionary to control access to Phidgets through the openRemote and openRemoteIP interfaces, and as such, you should never add or modify a key that starts with /PSK/ or /PCK/, unless you want to explicitly modify Phidget specific data – and this is highly discouraged, as it's very easy to break things. Listening to these keys is fine if so desired.

The intended use for the dictionary is as a central repository for communication and persistent storage of data between several client applications. For example, it can be used as a higher level interface exposed by one application that controls the Phidgets for others to access, rather then every client talking directly to the Phidgets themselves.

The dictionary makes use of extended regular expressions for key matching.

Functions

create

Creates a Phidget Dictionary.

```
void create();
```

Discussion:

Typically this is the first call on a handle / object, followed by registering the event handlers, and calling the variant of open required.

In object oriented languages, this is carried out by the class constructor - there is no explicit create function.

openRemote

Open a Phidget Dictionary remotely, using a Server ID.

```
void openRemote(
   string ServerID,
   string Password
);
```

Parameters:

ServerID

Server ID of the Phidget Web Service. This will be the name of the computer by default. This can be NULL to open this dictionary on the first available Web Service.

Password

Password for the Web Service when running in secure mode. This can be set to NULL when the Web Service is running without a password, or left out depending on the Library

used.

Discussion:

This version of open is network based.

Open is Asynchronous. What this means is that open will return immediately (before the server being connected to is actually available) and will be able to re-establish a connection to a server that goes down, automatically.

ServerID can be set to NULL, if the ServerID does not matter. In this case, the first Phidget Web Service found will be connected to.

openRemoteIP

Open a Phidget Dictionary remotely, using an IP Address.

```
void openRemoteIP(
   int Port,
   string Address,
   string Password
);
```

Parameters:

Port

Port that the Web Service is running at. By default this is 5001.

Address

Address that the Web Service is running at. This can be a hostname or and IP address.

Password

Password for the Web Service when running in secure mode. This can be set to NULL when the Web Service is running without a password, or left out depending on the Library used.

Discussion:

This version of open is network based.

Open is Asynchronous. What this means is that open will return immediately (before the server being connected is actually available), and will be able to re-establish connection to a server that goes down automatically.

close

Closes this Phidget Dictionary.

```
void close();
```

Discussion:

This will shut down all threads dealing with this Dictionary and no more events will be fired. The connection to the server will be closed.

delete

Frees the Phidget Dictionary handle / object.

```
void delete();
```

Discussion:

Once delete is called, it is no longer safe to calls any other functions on this handle, except for create.

In object oriented languages, this is carried out by the class destructor - there is no explicit delete function.

add

Adds a new key to the Dictionary, or modifies the value of an existing key.

```
void add(
   string Key,
   string Value,
   bool Persistent
);
```

Parameters:

Key
The key value. Note valid keys are restricted to a specific character set.

Value
The Value value.

Persistent
Whether the key will remain in the dictionary after disconnecting form the server.

Discussion:

The key can only contain numbers, letters, "/", ".", "-", "_", and must begin with a letter, "_" or "/".

The value can contain any value.

The persistent value controls whether a key will stay in the dictionary after the client that created it disconnects. If persistent == 0, the key is removed when the connection closes. Otherwise the key remains in the dictionary until it is explicitly removed. This value is optional and if not specified is true.

remove

Removes a key, or set of keys, from the Dictionary.

```
void remove(
   string KeyPattern
);
```

Parameters:

KeyPattern
A regular expression string representing a subset of keys to remove

Discussion:

The key name is a regular expressions pattern, and so care must be taken to only have it match the specific keys you want to remove.

get

Gets a key value from the dictionary.

```
string get(
   string Key
);
```

Parameters:

Key
The key to get the value of.

Discussion:

This method is not yet implemented and will always throw a PhidgetException if called.

Currently the only way to get keys is to listen for them with the KeyListener.

Events

The event model for the Phidget Dictionary is more complicated then with Phidgets or the Phidget manager. This is because we can specify any number of specific key patterns to listen for, and each of these has it's own set of events for key change and key removal.

In object oriented languages, there is a DictionaryListener class that takes a dictionary object and a key matching pattern string. You create as many of these objects as needed for each pattern you wish to listen for and start each using it's `start()` method.

In C/C++ you use the `set_OnKeyChange_Handler` and `remove_OnKeyChange_Handler` functions to add and remove listeners for specific key patterns. Also, in C/C++ there is only one type of key change event for both change and removal events.

Key matching patterns are specified using regular expressions. See the appendix for more information.

Consult the API manual for your language for more information.

KeyChange

Fired when a matching key is added to the dictionary, or when it's value changes.

```
event KeyChange(
    string Key,
    string Value
)
```

Parameters:

Key
The Key value.

Value
The Value value.

KeyRemoval

Fired when a matching key is removed from the dictionary.

```
event KeyRemoval(
    string Key,
    string Value
)
```

Parameters:

Key
The Key value.

Value
The Value value.

Error

Fired when an asynchronous error occurs.

```
event Error(
    int ErrorCode,
    string ErrorDescription
)
```

Parameters:

ErrorCode
The error code for this event.

ErrorDescription
The error description string for the event.

Discussion:

Error events are used when using the Phidget Web Service, to report asynchronous network errors.

Phidget Webservice

Overview

The Phidget Web Service is a socket based server which allow Phidgets to be opened remotely, over a network. The Web Service also serves as an interface for Phidgets to be used in Adobe Flash and Flex with Actionscript 3.0 and allows a single Phidget to be opened by multiple applications - something that cannot be done with the regular interface.

Once the Web Service is running, all of the Phidgets attached to that computer will be available for remote access. To open a Phidget remotely, use the OpenRemote or OpenRemoteIP function to open the Phidget. All other API calls and events are identical to a Phidget opened locally. A remotely opened Phidget gains extra functions as seen below. These will throw exceptions if called on a Phidget that was not opened remotely.

Phidget Managers can also be opened remotely, and the Phidget Dictionary also becomes available when the Web Service is run (See Phidget Dictionary section).

The Web Service is provided as a standalone executable - phidgetwebservice21 - which is included as part of the standard install.

Windows and Mac OS X provide a Web Service Tab in their Phidget Utilities (The Phidget Control Panel on Windows and the Phidget Preference Pane on Mac), which allows the Web Service to be configured, and set to start at boot.

The Web Service can also be started from the command line. Run with the -h switch to see a list of command line options.

See the Phidget networking manual for more information and recommended coding conventions.

Bonjour (mDNS)

As of version 2.1.3, Phidget21 support mDNS. This allows for Phidgets to be found and opened on the network without having to know the IP address or Port that the Web Service is running at.

When the Phidget Web Service is started, it checks to see if mDNS is available. If so, it automatically broadcasts Phidget attach and detach events across the network. Clients wishing to open one of these Phidgets (or a Manager or Dictionary), just need to call the `OpenRemote()` method with the Web Services Server ID.

The Server ID of a Web Service can be specified when the Web Service is started. If it is not specified, this defaults to the computer name.

Windows and Mac OS X provide a Bonjour Tab in their Phidget Utilities (The Phidget Control Panel on Windows and the Phidget Preference Pane on Mac), which lists all available Phidgets on the network.

Note: All open methods that specify a ServerID rather then an IP Address and Port require that both the client and host sides of the connection be running an implementation of zeroconf:

- On Windows, this means installing Apple's Bonjour - available here: http://www.apple.com/support/downloads/bonjourforwindows.html
- On Linux, this means Avahi, which is usually either installed by default or available as a package install.
- On Mac OS X, Bonjour is already integrated into the operating system.

Properties

These properties apply to all Phidgets, Managers, and Dictionaries.

ServerID

Gets the ServerID string.

```
string ServerID [get]
```

Discussion:

This is an arbitrary server identifier, independent of IP address and Port. Note that this is only available if this device was opened with openRemote.

Port

Gets the port that the Web Service is running at.

```
string Port [get]
```

Address

Gets the address that the Web Service is running at.

```
string Address [get]
```

Discussion:

This will be a fully qualified hostname if available, or an IP address otherwise.

AttachedToServer

Gets the network attached status for remotely opened Phidgets, managers and dictionaries.

```
bool AttachedToServer [get]
```

Discussion:

This method returns True or False, depending on whether a connection to the Phidget Web Service is open or not. If this is false for a remote Phidget, then the connection is not active (either because a connection has not yet been established, or because the connection was terminated).

Note that is some APIs, this function is called ServerStatus().

Events

These events apply to Phidgets, Managers and Dictionaries

ServerConnect

Fired when a connection to a server is made.

```
event ServerConnect
```

ServerDisconnect

Fired when a connection to a server is lost.

```
event ServerConnect
```

Appendix

Appendix I: Programming Concepts

Overview

The Phidgets software API relies on knowledge of several programming concepts, in order to be used properly. These concepts will be discussed briefly here, but note that this is not a manual for programming in general, and further study may be required.

Concepts that are specific to a particular language will be covered in that language's API manual.

Event Based Programming

Events are used extensively throughout the Phidgets API. Although the libraries can be used without events, this is discouraged - there are many advantages to event based programming (and it is a valuable skill to learn).

Threading

The Phidgets library is threaded. This means that at least a basic understanding of threads is highly recommended. There are many implications to a threaded library, many of which are not obvious.

Appendix II: Regular Expressions

Overview

Regular Expressions are used with the Phidget Dictionary Object to perform Key Matching. They are a powerful method for performing searches that can vary from extremely specific to extremely general matches. Please see the Examples for additional information on using the Dictionary object and Regular Expressions (available on the Downloads page from http://www.Phidgets.com)

Definitions

. Matches any single character. Into [ ] this character has its habitual meaning.

[ ] Matches a single character that is contained within the brackets. For example, [abc] matches “a”, “b”, or “c”.

[a-z] matches any lowercase letter. These can be mixed: [abcq-z] matches a, b, c, q, r, s, t, u, v, w, x, y, z, and so does [a-cq-z].

The ‘-’ character should be literal only if it is the last or the first character within the brackets: [abc-] or [-abc].

To match an ‘[‘ or ‘]’ character, the easiest way is to make sure the closing bracket is first in the enclosing square brackets: [][ab] matches ‘]’, ‘[‘, ‘a’ or ‘b’.

[^] Matches a single character that is not contained within the brackets. For example, [^abc] matches any character other than “a”, “b”, or “c”. [^a-z] matches any single character that is not a lowercase letter. As above, these can be mixed.

^ Matches the start of the line (or any line, when applied in multiline mode)

$ Matches the end of the line (or any line, when applied in multiline mode)

( ) Defines a “marked sub-expression”. What the enclosed expression matched can be recalled later. See the next entry, \n. Note that a “marked sub-expression” is also a “block”. Note that this is not found in some instances of regex.

\n Where n is a digit from 1 to 9; matches what the nth marked sub-expression matched. This construct is theoretically irregular and has not been adopted in the extended regular expression syntax.

* A single character expression followed by “*” matches zero or more copies of the expression. For example, “[xyz]*” matches “”, “x”, “y”, “zx”, “zyx”, and so on.

\n*, where n is a digit from 1 to 9, matches zero or more iterations of what the nth marked subexpression matched. For example, “(a.)c\1*” matches “abcab” and “abcabab” but not “abcac”.

An expression enclosed in “\(“ and “\)” followed by “*” is deemed to be invalid. In some cases (e.g. /usr/bin/xpg4/grep of SunOS 5.8), it matches zero or more iterations of the string that the enclosed expression matches. In other cases (e.g. /usr/bin/grep of SunOS 5.8), it matches what the enclosed expression matches, followed by a literal “*”.

+ A single character expression followed by “+” matches one or more copies of the expression. For example, “[xyz]+” matches “x”, “y”, “zx”, “zyx”, and so on.

\n+, where n is a digit from 1 to 9, matches one or more iterations of what the nth marked sub-expression matched.

An expression enclosed in “\(“ and “\)” followed by “+” is deemed to be invalid.

{x,y} Match the last “block” at least x and not more than y times. For example, “a\{3,5\}” matches “aaa”, “aaaa” or “aaaaa”. Note that this is not found in some instances of regex.

Extended Regular Expressions

+ Match the last “block” one or more times - “ba+” matches “ba”, “baa”, “baaa” and so on

? Match the last “block” zero or one times - “ba?” matches “b” or “ba”

| The choice (or set union) operator: match either the expression before or the expression after the operator - “abc|def” matches “abc” or “def”.

Also, backslashes are removed: \{…\} becomes {…} and \(…\) becomes (…). Examples:

“[hc]+at” matches with “hat”, “cat”, “hhat”, “chat”, “hcat”, “ccchat” etc.

“[hc]?at” matches “hat”, “cat” and “at”

“([cC]at)|([dD]og)” matches “cat”, “Cat”, “dog” and “Dog”

Since the characters ‘(‘, ‘)’, ‘[‘, ‘]’, ‘.’, ‘*’, ‘?’, ‘+’, ‘^’ and ‘$’ are used as special symbols they have to be escaped if they are meant literally. This is done by preceding them with ‘\’ which therefore also has to be escaped this way if meant literally. Examples:

“a\.(\(|\))” matches with the string “a.)” or “a.(“

Appendix III: Error Codes

These codes are returned by Phidget API calls. See the API manual for your language to see which codes can be returned by which functions.

EPHIDGET_NOTFOUND = 1

Phidget not found exception. “A Phidget matching the type and or serial number could not be found.”

This exception is not currently used externally.

EPHIDGET_NOMEMORY = 2

No memory exception. “Memory could not be allocated.”

This exception is thrown when a memory allocation (malloc) call fails in the c library.

EPHIDGET_UNEXPECTED = 3

Unexpected exception. “Unexpected Error. Contact Phidgets Inc. for support.”

This exception is thrown when something unexpected happens (more unexpected than another exception). This generally points to a bug or bad code in the C library, and hopefully won’t even be seen.

EPHIDGET_INVALIDARG = 4

Invalid argument exception. “Invalid argument passed to function.”

This exception is thrown whenever a function receives an unexpected null pointer, or a value that is out of range. ie setting a motor’s speed to 101 when the maximum is 100.

EPHIDGET_NOTATTACHED = 5

Phidget not attached exception. “Phidget not physically attached.”

This exception is thrown when a method is called on a device that is not attached, and the method requires the device to be attached. ie trying to read the serial number, or the state of an output.

EPHIDGET_INTERRUPTED = 6

Interrupted exception. “Read/Write operation was interrupted.”

This exception is not currently used externally.

EPHIDGET_INVALID = 7

Invalid error exception. “The Error Code is not defined.”

This exception is thrown when trying to get the string description of an undefined error code.

EPHIDGET_NETWORK = 8

Network exception. “Network Error.”

This exception is usually only seen in the Error event. It will generally be accompanied by a specific Description of the network problem.

EPHIDGET_UNKNOWNVAL = 9

Value unknown exception. “Value is Unknown (State not yet received from device, or not yet set by user).”

This exception is thrown when a device that is set to unknown is read (e.g., trying to read the position of a servo before setting its position).

Every effort is made in the library to fill in as much of a device's state before the attach event gets thrown, however, many there are some states that cannot be filled in automatically (e.g., older interface kits do not return their output states, so these will be unknown until they are set).

This is a quite common exception for some devices, and so should always be caught

EPHIDGET_BADPASSWORD = 10

Authorization exception. "Authorization Failed."

This exception is thrown during the Error event. It means that a connection could not be authenticated because of a password miss match.

EPHIDGET_UNSUPPORTED = 11

Unsupported exception. "Not Supported."

This exception is thrown when a method is called that is not supported, either by that device, or by the system (e.g., calling setRatiometric on an interfaceKit that does not have sensors).

EPHIDGET_DUPLICATE = 12

Duplicate request exception. "Duplicated request."

This exception is thrown when open is called twice on a device, without calling close in between. The second call to open is ignored.

EPHIDGET_TIMEOUT = 13

Timeout exception. "Given timeout has been exceeded."

This exception is thrown by waitForAttachment(int) if the provided time out expires before an attach happens. This may also be thrown by a device set request, if the set times out (though this should not happen, and would generally mean a problem with the device).

EPHIDGET_OUTOFBOUNDS = 14

Out of bounds exception. "Index out of Bounds."

This exception is thrown anytime an indexed set or get method is called with an out of bounds index.

EPHIDGET_EVENT = 15

Event exception. "A non-null error code was returned from an event handler."

This exception is not currently used.

EPHIDGET_NETWORK_NOTCONNECTED = 16

Network not connected exception. "A connection to the server does not exist."

This exception is thrown when a network specific method is called on a device that was opened remotely, but there is no connection to a server (e.g., getServerID).

EPHIDGET_WRONGDEVICE = 17

Wrong device exception. "Function is not applicable for this device."

This exception is thrown when a method from device is called by another device. ie casting an InterfaceKit to a Servo and calling setPosition.

Appendix IV: Logging

Overview

Logging is provided in the Phidgets libraries for debugging purposes. Logging can be enabled and written to from any language. Currently, only the C library writes to the Log. Logging is usually used to help Phidgets Inc. debug a problem, but it can also be used by users to record log messages.

Logs can be sent to either a file or stdout.

Constants

There are 6 levels of logging. Each higher level will include all lower levels when outputting logs. These are represented either as a set of constants or an enumerator, depending on language.

PHIDGET_LOG_CRITICAL = 1

Critical error messages.

This is the lowest logging level. Errors at this level are generally non-recoverable and indicate either hardware problems, library bugs, or other serious issues.

PHIDGET_LOG_ERROR = 2

Non-critical error messages.

Errors at this level are generally automatically recoverable, but may help to track down issues.

PHIDGET_LOG_WARNING = 3

Warning messages.

Warnings are used to log behavior that is not necessarily in error, but is nevertheless odd or unexpected.

PHIDGET_LOG_DEBUG = 4

Debug messages.

Debug messages are generally used for debugging at Phidgets Inc.

Note: **PHIDGET_LOG_DEBUG** messages are only logged in the debug version of the library, regardless of logging level. Thus, these logs should never be seen outside of Phidgets Inc.

PHIDGET_LOG_INFO = 5

Informational messages.

Informational messages track key happenings within phidget21 - mostly to do with threads starting and shutting down, and the internal USB code.

PHIDGET_LOG_VERBOSE = 6

Verbose messages.

This is the highest logging level. Verbose messages are informational messages that are expected to happen so frequently that they tend to drown out other log messages.

Functions

enableLogging

Turns on logging in the native C Library.

```
void enableLogging(
   loglevel Level,
   string File
);
```

Parameters:

Level

highest level of logging that will be output. This is one of the constants from the Constants section above. This may be and int or an enumerator, depending on the language.

File

file to output log to. Specify NULL to output to stdout (console).

Discussion:

This is mostly useful for debugging purposes - when an issue needs to be resolved by Phidgets Inc. The output is mostly low-level library information, that won't be useful for most users.

Logging may be useful for users trying to debug their own problems, as logs can be inserted by the user using `log()`.

disableLogging

Turns off logging in the native C Library.

```
void disableLogging();
```

Discussion:

This only needs to be called if `enableLogging()` was called to turn logging on. This will turn logging back off.

log

Adds a log entry into the Phidget log.

```
void log(
   loglevel Level,
   string ID,
   string Message
);
```

Parameters:

Level

highest level of logging that will be output. This is one of the constants from the Constants section above. This may be and int or an enumerator, depending on the language.

ID

an arbitrary identifier for this log. This can be NULL. The C library uses this field for source filename and line number.

Message

the message to log.

Discussion:

This log is enabled by calling `enableLogging()` and this allows the entry of user logs in amongst the phidget library logs.

Note: PHIDGET_LOG_DEBUG should not be used, as these logs are only printed when using the debug library, which is not generally available.