NAME

Net::Bluetooth − Perl Bluetooth Interface

SYNOPSIS

```
use Net::Bluetooth;

#### list all remote devices in the area
my $device_ref = get_remote_devices();
foreach $addr (keys %$device_ref) {
      print "Address: $addr Name: $device_ref->{$addr}\n";
}


#### search for a specific service (0x1101) on a remote device
my @sdp_array = sdp_search($addr, "1101", "");

#### foreach service record
foreach $rec_ref (@sdp_array) {
      #### Print all available information for service
      foreach $attr (keys %$rec_ref) {
              print "Attribute: $attr Value: $rec_ref->{$attr}\n";
      }
}


#### Create a RFCOMM client
$obj = Net::Bluetooth->newsocket("RFCOMM");
die "socket error $!\n" unless(defined($obj));
if($obj->connect($addr, $port) != 0) {
      die "connect error: $!\n";
}

#### create a Perl filehandle for reading and writing
*SERVER = $obj->perlfh();
$amount = read(SERVER, $buf, 256);
close(SERVER);



#### create a RFCOMM server
$obj = Net::Bluetooth->newsocket("RFCOMM");
#### bind to port 1
if($obj->bind(1) != 0) {
      die "bind error: $!\n";
}

#### listen with a backlog of 2
if($obj->listen(2) != 0) {
      die "listen error: $!\n";
}

#### register a service
#### $obj must be a open and bound socket
my $service_obj = Net::Bluetooth->newservice($obj, "1101", "GPS", "GPS Receiver
unless(defined($service_obj)) {
      #### couldn't register service
}

#### accept a client connection
```

```
$client_obj = $obj->accept();
unless(defined($client_obj)) {
      die "client accept failed: $!\n";
}

#### get client information
my ($caddr, $port) = $client_obj->getpeername();

#### create a Perl filehandle for reading and writing
*CLIENT = $client_obj->perlfh();
print CLIENT "stuff";

#### close client connection
close(CLIENT);
#### stop advertising service
$service_obj->stopservice();
#### close server connection
$obj->close();
```

DESCRIPTION

This module creates a Bluetooth interface for Perl.

Net::Bluetooth works with the BlueZ libs as well as with Microsoft Windows.

If you are going to be using a Unix system, the Bluez libs can be obtained at www.bluez.org. Please make sure these are installed and working properly before you install the module. Depending on your system BlueZ maybe already installed, or you may have to build it yourself and do some configuration. You can verify that BlueZ can detect devices and services with the utilities that are included with it (hciconfig, sdptool, hcitool, etc).

If you are using Windows, please make sure you have Service Pack 2 installed and the Microsoft Platform SDK. Also please make sure the ‘‘$win_include’’ variable at the top of Makfile.PL is set properly. This needs to point to the SDK include directory for SP2. This is where the module will look for all the Bluetooth header files (ws2bth.h, etc).

Please check out the samples included in the samples directory for more general information.

FUNCTIONS

get_remote_devices()

Searches for remote Bluetooth devices. The search will take approximately 5 − 10 seconds (This will be a configurable value in the future.). When finished, it will return a hash reference that contains the device address and name. The address is the key and the name is the value. Name will be set to ‘‘[unknown]’’ if the name could not be resolved. See the NOTES section of this document for more information about this.

sdp_search($addr, `$uuid`, `$name`)

This searches a specific device for service records. The first argument is the device address which is not optional. The uuid argument can be a valid uuid or ‘‘0’’. The name argument can be a valid service name or "". It will return services that match the uuid or service name if supplied, otherwise it will return all public service records for the device.

The return value is a list which contains a hash reference for each service record found. The key/values for the hash are as follows:

`SERVICE_NAME`: Service Name

`SERVICE_DESC`: Service Description

`SERVICE_PROV`: Service Provider

`RFCOMM`: RFCOMM Port

`L2CAP`: L2CAP Port

`UNKNOWN`: Unknown Protocol Port

If any of the values are unavailable, the keys will not exist.

If `$addr` is ‘‘localhost’’ the call will use the local SDP server.

SOCKET OBJECT

The bluetooth socket object is used to create bluetooth sockets and interface with them. There are two types of sockets supported, RFCOMM and L2CAP. The methods are listed below.

newsocket(‘‘RFCOMM’’)

This constructs a socket object for a RFCOMM socket or L2CAP socket.

connect($addr, `$port`)

This calls the *connect()* system call with the address and port you supply. You can use this to connect to a server. Returns 0 on success.

bind($port)

This calls the *bind()* system call with the port you provide. You can use this to bind to a port if you are creating a server. Returns 0 on success. As a side note, RFCOMM ports can only range from 1 − 31.

listen($backlog)

This calls the *listen()* system call with the backlog you provide. Returns 0 on success.

accept()

This calls the *accept()* system call and creates a new bluetooth socket object which is returned. On failure it will return undef.

perlfh()

This call returns a Perl filehandle for a open socket. You can use the Perl filehandle as you would any other filehandle, except with Perl functions that use the socket address structure. This provides a easy way to do socket IO instead of doing it through the socket object. Currently this is the only way to do socket IO, although soon I will provide read/write calls through the object interface.

close()

This closes the socket object. This can also be done through the Perl *close()* call on a created Perl filehandle.

getpeername()

This returns the address and name for a open bluetooth socket. (BlueZ only for now)

SERVICE OBJECT

The service object allows you to register a service with your local SDP server. The methods are as follows:

newservice($obj, `$service_uuid`, `$service_name`, `$service_desc`)

This registers a service with your local SDP server. The first argument is a open and bound socket that you created with *newsocket()*. The second argument is the service uuid. The third argument is the service name. The fourth argument is the service description.

The return value is a new service object. This will be undefined if there was an error.

stop_service()

This unregisters your service with the local SDP server. The service will be unregistered without this call when the application exits.

NOTES

All uuids used with this module can either be 128 bit values: ‘‘00000000−0000−0000−0000−000000000000’’ or 16 bit values: ‘‘0000’’. All values must be represented as strings (enclosed in quotes), and must be hexadecimal values.

Windows will not immediately return the device name if it is not already cached. Also there is no mechinism to alert the system when it has acquired the device name. Therefore you may have to call *get_remote_devices()* twice before the name shows up. I’ll see if this can be handled better in the future.

Currently on Windows the service name and description returned by *sdp_search()* are not setting their terminating NULL character properly. This can result in some garbage characters at the end of the string. I am looking at parsing the raw record to fix this problem.

REQUIREMENTS

You need BlueZ or Microsoft Service Pack 2 installed and the Microsoft Platform SDK. Windows needs at least Perl 5.8.

AUTHOR

Ian Guthrie IGuthrie AT aol DOT com

SEE ALSO

perl (1).