Podmo's internal Bluetooth technology

Version 1.0

Author: Greg Payne

Free-Zone server and Bluetooth code

This document addresses how to handle Bluetooth features across all platforms, and even third-party stacks within the same platform (eg Widcomm, etc). The design and implementation of this technology constitutes well over two thirds of the total development cost of the Free-Zone server application and represents a highly desirable developmental tool with better features and greater multi-platform and multi-Bluetooth Stack support than all of the several commercially available development tools put together. Given that these tools distributors charge their customers tens of thousands of dollars up-front as well as various royalty costs, this technology forms much of the int

Kukan's *wxBluetooth* C++ class heirarchy

{Conceived with the Podmo Free-Zone server project in mind}.
Note: *The class interface for this API can be found within the "wxBluetooth" folder in the "node-free-version" SVN project. The file structure found here is intended to mirror that of any of the official and/or third-party wxWidget library offerings from the wxCommunity developers, in that the potential is to either use the library independantly, or to optionally drag directly into the root wxWidgets source tree in keeping with traditional wxWidget open-source methodology.*

Primary design goals for wxBluetooth API

1). All Bluetooth functions (device discovery, service publishing, connections in and out, sending/receiving data) must work, regardless of which platform or stack is used.

2). The process of providing support for a completely new platform/stack must be as painless as possible. A small, core set of API's which perform distinct, well-defined bluetooth-related tasks, are all that must be implemented to create a new wxBluetoothProvider class.

3). A custom Bluetooth service with unique GUID must be simple to create, publish, and connections processed with a minimum of fuss and maximum re-use of common code.
(a) examples of classes derived from wxBluetoothService are
- PodmoBluetoothService (ie. encapsulates all Ping/Web/OBEXRequestHandling for the Podmo client protocol)
- OBEXBluetoothService (an OBEX server, for receiving connections and handling requests to send/receive files)

(b) Code which is typically common to all services includes functions such as
- publishing the service in the SDP table
- waiting for and accepting multiple incoming connections
- updating the RF Channel within the SDP where applicable
- basic I/O buffering, data stats monitoring, etc.

(c) The ONLY code that should be unique to each service, should relate solely to:
- The Service GUID and other SDP parameters.
- The protocol itself.

4). Use of Bluetooth Events to make the implementation of dynamically updating GUI displays, as simple and uniform as possible.

(a). To avoid having to duplicate code to display device details and connection stats for every different platform and stack, a simple wxBluetoothEvent interface is designed according to the multipipelined, asynchronous wxEventHandler paradigm.

(b). Typical Bluetooth Events at this time include events such as:
- device discovery
- dropouts
- connections
- Data I/O and statistics

(c). Event Handlers may choose to handle one or more of these events via the standard, easy to use Event Tables, which are very neatly done using powerful macros (like BEGIN/END_EVENT_TABLE() macros which appear before and after a series of EVENT_HANDLER(EVENT_TYPE, Class::FunctionName) which automatically associates incoming events of certain types with specific member functions to handle, skip, or even subvert future processing of the event.

(d). Within the Free-Zone server, the main window's event handler (which handles all other events (eg. user input, GUI-related, and scheduled timer events, also has handlers for many of the Bluetooth Events, which reflect the event details and bluetooth status within the "Visible Devices" and "Connected Devices" windows (which contain simple list control reports which reflect the presence and status for each relevant device). Another event-handler simply logs bluetooth event details to the standard log output. Other event-handlers could conceivably be implemented that display Bluetooth status in different ways (eg. flashing lights in the system tray, or some big modem-rack with 105409850425 blinking lights and rotating Bluetooth icons in 3D with radio waves bouncing around... etc).

(e) Base bluetooth classes (consisting of platform-independant/common code) are typically responsible for posting these events.

(f) Derived bluetooth classes may override or modify how these events are sent if there was some extra detail or other reason to. In practice, it's doubtful to happen very much beyond minor platform tweaks.

(g) If a new or different way of displaying and interacting with Bluetooth stats is required, implementing a new event-handler is a
With these design goals in mind, the skeleton interface that is now used within the Free-Zone server is attached.

Class interface

Applications wishing to use the functionality for this library should add:
`#include <wx/bluetooth.h>`

Please see Fig 1 (below) and the `wxBluetooth/include/wx/bluetooth.h` API interface file (in the SVN project) for the latest interface specification as used by current Free-Zone server releases.

Fig 1. wxBluetooth class interface specification [continues overleaf]

```
//============================================================
//
// wxBluetooth
//
// File:
//   wx/bluetooth.h
// Description:
//   Multiplatform/Multistack bluetooth support provision.
//   Designed to make any bluetooth application as simple and
//   intuitive as the rest of wxWidgets.
// Author:
//   Greg Payne <mailto:greg@dodgychemes.com>
//
// These objects encapsulate all runtime client/server info
// and provide a multiplatform, multienvironment, interface
// so that all Bluetooth behaviour can be controlled via this
// consistent interface.
//
// The intention is, that derived classes of wxBluetoothServer
// will also derive their own internal client class, from
// wxBluetoothServer::iClient.
//
// Thus, anyone wishing to provide new support for a new
// platform or bluetooth stack framework, need only to derive
// a new class from one of the wxBluetoothServer derivatives.
//
// All server-side override functionality*, can be provided by
// overriding one or more of the wxBluetoothServer virtual
// methods. Similarly, client-side functions**, can be provided
// in the form of overridden wxBluetoothServer::iClient methods.
//
//  * An example : SDP Table publishing &/or searching.
// ** An example : Connect/Disconnect/Input/Ouput for a remote
//                 client device.
//
// For further information, read the wxBluetooth documentation.
//
// For any other support or Podmo enquires, please contact me
// at my Podmo mailing address below.
//                          --Greggy P. <mailto:greg@podmo.com>
//                                                 May/June, 2007
#pragma once
#ifndef __WXBLUETOOTH_H__
#define __WXBLUETOOTH_H__

// ------------------------------------------------------------------
// wxBluetooth
// Preprocessor Definitions and Constants.
//
#define wxBLUETOOTH_DEFAULTS                                   (-1)
#define wxBLUETOOTH TICK DURATION                            (1.20f)
#define wxBLUETOOTH INQUIRY TIMEOUT TICKS                    (45.0f)
#define wxBLUETOOTH INQUIRY TIMEOUT                                 \
       (wxBLUETOOTH_INQUIRY_TIMEOUT_TICKS*                          \
        wxBLUETOOTH_TICK_DURATION)
#define wxBLUETOOTH GUID INVALID                               (-1)

// ------------------------------------------------------------------
// Include Files
//
#include <wx/wx.h>
#include <wx/datetime.h>

// ------------------------------------------------------------------
// wxBluetooth classes.
// Predeclarations that can illustrate the basic class heirarhcy.
//
//
// +==============+
// | wxBluetooth |
// +==============+
// |
/* +--*/class wxBluetoothAddress;
/* +--*/class wxBluetoothGUID;
```

Fig 1. wxBluetooth class interface specification [continues overleaf]

```
/* +--*/class wxBluetoothDevice;
/* +--*/class wxBluetoothConnection;
/* +--*/class wxBluetoothDevice;
/* +--*/class wxBluetoothService;
/* +--*/class wxBluetoothProvider;
/* |    +--*/class wxBerkeleyBluetoothProvider;
/* |    |    +--*/class wxBluetoothProviderMSW;
/* |    |    +--*/class wxBluetoothProviderBLUEZ;
/* |    +--*/class wxBluetoothProviderWIDCOMM;
/* |    +--*/class wxBluetoothProviderSOLEIL;
/* |    +--*/class wxBluetoothProviderMACOSX;
// |
// +
// ------------------------------------------------------------------

#pragma pack(1)
    // ------------------------------------------------------------------
    // wxBluetoothAddress
    // Encapsulates the well-known AA:BB:CC:DD:EE:FF style of universal
    // Bluetooth 48-Bit Address space.
    //
    class wxBluetoothAddress
    {
      public:
        wxBluetoothAddress();
        /*
        union
        {
            struct
            {
                wxUChar         m_uchAddress[6];
                wxUChar         m_uchPadding[2];
            }
            UCH;

            struct
            {
                wxUint32Long     m_ullAddress;
            }
            ULL;
        };
        */
        wxUChar m_uchAddress[6];
    };
    //
    // ------------------------------------------------------------------
#pragma pack()

// ------------------------------------------------------------------

#pragma pack(1)
    // ------------------------------------------------------------------
    // wxBluetoothGUID
    // Encapsulates the 128-bit GUID values given to common Bluetooth
    // Services. Published in the SDP tables.
    //
    // [ TODO: Make available some common GUID's from the official Bluetooth Spec ].
    //
    class wxBluetoothGUID
    {
    public:
        wxUint32    m_Data1;
        wxUint16    m Data2;
        wxUint16    m Data3;
        wxUChar     m Data4[8];

        wxBluetoothGUID();
        wxBluetoothGUID(wxUChar* pucDataForGUID);
        wxBluetoothGUID(wxUint32 d1, wxUint16 d2, wxUint16 d3, wxUChar d4a, wxUChar d4b, wxUChar
d4c, wxUChar d4d, wxUChar d4e, wxUChar d4f, wxUChar d4g, wxUChar d4h);
    };
    //
    // ------------------------------------------------------------------
#pragma pack()
```

Fig 1. wxBluetooth class interface specification [continues overleaf]

```
//
// ----------------------------------------------------------------

// ----------------------------------------------------------------
// wxBluetoothDevice
//
class wxBluetoothDevice
{
 public:

   enum FLAGS {
        DEVICE_REMOTE        = 0x0000,
        DEVICE_LOCAL         = 0x0001,
        DEVICE_CONNECTED     = 0x0002,
        DEVICE_INBOUND       = 0x0004,
        DEVICE_REMEMBERED    = 0x0008,
        DEVICE_AUTHENTICATED = 0x0010,
        DEVICE_ENCRYPTED     = 0x0020,
        DEVICE_DISCONNECTED  = 0x0040,
        DEVICE_BANNED        = 0x0080,
        DEVICE_SUSPECT       = 0x0100,
        DEVICE_CRIMINAL      = 0x0200,
        DEVICE_STOLEN        = 0x0400,
        DEVICE_MISSING       = 0x0800,
        DEVICE_PODMOUSER     = 0x1000,
        };
   enum CLASS {
        DEVICE_HANDSET       = 0x0001,
        DEVICE_DESKTOP       = 0x0002,
        DEVICE_MP3PLYR       = 0x0004,
        DEVICE_HEADSET       = 0x0008,
        DEVICE_MIC           = 0x0010,
        DEVICE_VIDEO         = 0x0020,
        DEVICE_CAMERA        = 0x0040,
        DEVICE_MOBILE        = 0x0080,
        DEVICE_GIZMO         = 0x0100,
        CLASS_DEFAULT        = DEVICE_HANDSET + DEVICE_MOBILE
        };
   enum CONFIG {
        CONFIG_RECVBUF       =   8192,
        CONFIG_SENDBUF       =  57344,
        };

   // --------------------[ Public Methods ]----------------------
   const wxBluetoothAddress& GetAddress()   { return m_btAddress;   }
   const wxString& GetName()                { return m_strName;     }
   void SetName(const wxString& strName)    { m_strName=strName;    }
   void SetId(int nId)                      { m_nId = nId;          }
   int GetId()                              { return m_nId;         }

   //-------------------------------------------------------------------------
   // PROTECTED MEMBERS
   // - INTERNAL MEMBER DATA FOLLOWS
 public:
   //-------------------------------------------------------------------------
   //
   // wxBluetoothDevice
   //
   // Note that default constructors and destructors are PROTECTED,
   // meaning that only a friendly wxBluetoothServer object is able
   // to create and destroy wxBluetoothDevice objects.
   //
   // The user of the wxBluetooth API only generally interacts with
   // wxBluetoothDevice objects in response to events and/or callbacks
   // generated by various wxBluetoothServer functions.
   //
           wxBluetoothDevice(wxBluetoothAddress& addr, const wxString& strName, int nId, CLASS c =
CLASS_DEFAULT);
   virtual ~wxBluetoothDevice() {}

   // -----------------[ Vitals ]------------------
   wxString                            m_strName;
   wxBluetoothAddress                  m_btAddress;
   int                                 m_nId;
   wxBluetoothDevice*                  m_pdevDetectedBy;
   void*                               m_pvPrivate;
   int                                 m_cbPrivate;
```

Fig 1. wxBluetooth class interface specification [continues overleaf]

```
};


// ----------------------------------------------------------------------
// wxBluetoothDeviceList
//
WX_DECLARE_LIST(wxBluetoothDevice, wxBluetoothDeviceList);
//
// ----------------------------------------------------------------------
//
extern wxBluetoothProvider* g pwxBluetoothProvider;
//wxBluetoothProvider& ::wxGetBluetoothProvider()
//{
//     return *g_pwxBluetoothProvider;
//}
class wxBluetoothProvider
{
public:
       wxBluetoothProvider() { g pwxBluetoothProvider = this; m bAsynchronous = false; }
       virtual ~wxBluetoothProvider() {}
    virtual int Startup();
    virtual int Shutdown();
       virtual int BeginDeviceEnumeration();
    virtual int OnDeviceDiscovered(wxBluetoothDevice& devLocal, wxBluetoothDevice& devRemote);
       virtual int PublishService(const wxBluetoothService& svc, int rfChannel);
       virtual wxBluetoothConnection& ConnectTo(const wxBluetoothAddress& addr, int rfChannel);
       virtual wxBluetoothConnection& ConnectTo(const wxBluetoothAddress& addr, GUID& guid);
       virtual wxBluetoothConnection& AcceptIncomingConnection(int rfChannel);
       virtual wxBluetoothConnection& InstallService(const wxBluetoothService& svc, int rfChannel
= -1);
       virtual int Transmit(wxBluetoothConnection& conn, void* p, int cb);
       virtual int Receive(wxBluetoothConnection& conn, void* p, int cb);
    wxBluetoothDevice& CreateRemoteDevice(const wxBluetoothAddress& addr);
    wxBluetoothDevice& GetLocal(int idx=0);
    wxBluetoothDevice& GetRemote(int idx=0);

    bool                    m_bAsynchronous;
    wxBluetoothDeviceList   m_devicesLocal;
    wxBluetoothDeviceList   m_devicesRemote;

};


//
// Proj: wxBluetooth
// File: wx/bluetooth.h
// Code: Greggy P. <mailto:greg@podmo.com>
// ----------------------------------------------------------------------
//                                                                   EOF
//







class wxBluetoothConnection
{
public:
    // Flags
    enum StateFlags {
         Uninitialized      = 0x00000001,
         WaitingForIncoming = 0x00000002,
         Attempting         = 0x00000004,
         Connected          = 0x00000008,
         Disconnected       = 0x00000010,
         Reconnected        = (Connected+Disconnected),
         Error              = 0x80000000
         }                  m flags;

    // Default Constructor
       wxBluetoothConnection(
        int cbBufferIn=65536,
        int cbBufferOut=65536
        )
       :
        m_flags(Uninitialized),
```

Fig 1. wxBluetooth class interface specification [continues overleaf]

```
        m_dxIn (cbBufferIn),
        m_dxOut(cbBufferOut)
        {
        m_devLocal = NULL;
        m_devRemote = NULL;
        m_svc = NULL;
        SetWaitLen(-1);
        SetRFChannel(-1);
        SetContiguous();
    }

    wxBluetoothService* GetService()
        { return m_svc; }
    wxBluetoothDevice* GetLocalDevice()
        { return m_devLocal; }
    wxBluetoothDevice* GetRemoteDevice()
        { return m_devRemote; }
    int GetRFChannel()
        { return m_iRFChannel; }

    // Get/Set how many (incoming) bytes of data are required before this
    // service handler can perform any further processing. When the wait condition
    // is met, the virtual handler routine (override) is called with a ref to the data.
    int  GetWaitLen()               { return m_cbWait; }
    bool IsWaiting()                { return GetWaitLen() > 0; }
    bool IsWaitComplete()           { return IsWaiting() && m_dxIn.BytesAvailable() >=
GetWaitLen(); }
    bool GetContiguous()            { return m_bNeedContiguousData; }
    void SetRFChannel(int iRFChan)  { m_iRFChannel=iRFChan; }
    void SetWaitLen(int cbReqd)     { m_cbWait=cbReqd; }
    void SetContiguous(bool b=true) { m_bNeedContiguousData=b; }


protected:
    // Vital Statistix:
       wxBluetoothDevice*      m_devLocal;
       wxBluetoothDevice*      m_devRemote;
       wxBluetoothService*     m_svc;
       int                     m_iRFChannel;
    int                     m_cbWait;
    bool                    m_bNeedContiguousData;

    // Less-vital (GUI/Log/Etc) Statistix
    wxDateTime              m_tsEstablished;
       wxTimeSpan              m_tsElapsed;

public:
    class HalfDuplexStream
    {
    public:
         HalfDuplexStream(int cbBuffer=65536);
        ~HalfDuplexStream();
        bool                IsContiguous() { return m_bContiguousBuffer; }
              int                   AdvanceTip (int cbAdvanceBy=1);
              int                   AdvanceTail(int cbAdvanceBy=1);
        int                 BytesAvailable();
    protected:
              int                   m_cbBuffer;
              unsigned char*        m_pBuffer;
              int                   m_idxTip;
              int                   m_idxTail;
              int                   m_cOverflows;
              int                   m_cbXferred;
              bool                  m_bContiguousBuffer;
              wxTimeSpan            m_tsElapsed;
              wxDateTime            m_tmLastActivity;
       wxCriticalSection   m_crit;
    };

       HalfDuplexStream        m_dxIn;
       HalfDuplexStream        m_dxOut;

};

class wxBluetoothService
{
protected:
```

Fig 1. wxBluetooth class interface specification [continues overleaf]

```
        enum ServiceType {
                ServiceSimpleRF,
                ServiceCustomSDP
                }                                   m_type;
        wxBluetoothGUID                 m guid;
        int                             m iRFChannel;
        void*                           m_pvCustomSDP;

public:
        wxBluetoothService(wxBluetoothGUID& guid)
    :    m type(ServiceSimpleRF),
         m guid(guid),
         m_iRFChannel(-1),
         m_pvCustomSDP(NULL)
         {
         }
        virtual ~wxBluetoothService() {}
    virtual int OnConnectionAccepted(wxBluetoothConnection& conn)             { return 0; }
    virtual int OnBytesReceived(wxBluetoothConnection& conn, void* p, int cb) { return 0; }
};


/*
class wxBluetoothEvent : public wxEvent
{
public:
        enum BTEventType {
                BTEventStartup,
                BTEventReady,
                BTEventDeviceDiscovered,
                BTEventDeviceDisappeared,
                BTEventConnection,
                BTEventDisconnection,
                BTEventDataIO,
                BTEventShutdown,
                BTEventError
                } m_btEvtType;
    wxBluetoothEvent(wxBluetoothEvent& e);
    wxBluetoothEvent(wxEventType commandType = wxEVT_NULL, int id = 0);
    void SetProvider(wxBluetoothProvider* pprovider) { m pprovider=pprovider; }
    void SetConnection(wxBluetoothConnection* pconn) { m pconn=pconn; }
    void SetDevice(wxBluetoothDevice* pdevice) { m pdevice=pdevice; }
    void SetBuffer(void* pvoid, int cb) { m_pvoid=pvoid; m_cbvoid=cb; }
    wxBluetoothProvider* GetProvider() { return m_pprovider; }
    wxBluetoothConnection* GetProvider() { return m_pprovider; }
    wxBluetoothProvider* GetProvider() { return m pprovider; }
    wxBluetoothProvider* GetProvider() { return m pprovider; }
    wxBluetoothProvider* GetProvider() { return m pprovider; }

protected:
    wxBluetoothProvider*    m pprovider;
    wxBluetoothConnection*  m pconn;
    wxBluetoothDevice*      m pdevice;
    void*                   m_pvoid;
}
*/


#endif  //  WXBLUETOOTH H
```