# Pico Stack PSS1

# Windows Development Version

# Users Manual

# Table of Contents

# 1 Introduction

The PICO Stack Bluetooth Software for Windows developer kit allows application programmers to access the configuration functions of the PICO Stack directly by using a standard windows DLL, thus replacing the Bluetooth Monitor user interface.

# 2 System Requirements

Supported operating systems are Windows NT, Windows 2000 and Windows XP. The Bluetooth Hardware (PicoCard, PicoPCI or any USB Bluetooth Adapter supported by G&W Instruments PICO USB Stack) and the appropriate drivers must be installed before using this software.
To use the DLL interface the Bluetooth Monitor must be stopped, because the configuration interface of the Bluetooth stack can no be multiplexed between applications. For information on how to stop and restart the Bluetooth Monitor see section Installation.

# 3 Installation

The developers kit consists of a standard Windows DLL with an accompanying export library, C language header files and a C command line demonstration program. To use the DLL it must be copied into a directory which is included in the search path (e.g. c:\windows\system32) or the applications installation directory.

To test the DLL interface copy all files provided to a separate directory. It is assumed that the PICO Bluetooth software is already installed and the Bluetooth Monitor program is running. Than open a command line window and change to the installation directory. At the command prompt type
*c:\PicoLib\>net stop btmonitor*
to stop the Bluetooth monitor. Than run the test program
*c:\PicoLib\>PicoConfigTest*
This will give you a list of available options. You can do inquiries, name request, service requests and more.
To restart the Bluetooth monitor program type
*c:\PicoLib\>net start btmonitor*

## 3.1 List of Files

| | |
|---|---|
| PicoConfig.dll | The Interface DLL |
| PicoConfig.lib | DLL Export Library |
| | |
| PicoIf.h | Main Include File, defines structures and functions used by the DLL |
| Meta.h | Basic Type Definitions |
| ErrorCodes.h | Definitions for error codes |
| ServiceDc.h | Structures and definitions for service requests |
| | |
| PicoConfigTest.exe | DLL test program (Windows command line application) |
| PicoConfigTest.c | Source code of test program |
| MakeTestProg.bat | Script to compile test program (requires MS C-compiler) |

More programming examples can be found in the subdirectory *examples*.

# 4 Connecting, Disconnecting and Data Transfer

To connect to or disconnect from a remote device simply open / close the associated COM port from your application. When opening a Bluetooth virtual Com port, the Bluetooth stack will automatically connect to the installed remote device (see *SetBTPeer()*).

If the connection fails (e.g. because the remote device is out of range) the open() call will fail. However, this standard behaviour may be changed by setting the special registry key

*HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\BtDriver\Devices\COMM\X\ CreateAlwaysSucceeds*

where "X" is to be replaced with the port number minus 1 (e.g. for COM3 X=2).If this binary (1 Byte) key exists and is set to one the port open will succeed even if the connection could not be made. The Bluetooth stack will try to create the connection each time it receives a write request to the virtual port.

To create a virtual COM port you must use the Bluetooth Monitor and either install a remote device or a local service.

If you use a port installed as a local service and do not set its peer address, no connection is made when the port is opened. Instead, once you have opened the port, the service will be visible to other Bluetooth devices and the Bluetooth stack will wait for a connection from a remote device.

If you have a port with a valid peer address and want to use it as a local service, you can set the peer address to 0xFF-0xFF-0xFF-0xFF-0xFF-0xFF. This address is interpreted by the Bluetooth stack as an invalid address and it will not try to connect to this address. Instead, it will wait for a connection from a remote device.

Data is transferred by simple read / write operations on the virtual COM port. It is strongly recommended that you use time outs for read and write requests to prevent blocking of your application in case a connection attempt has failed or the connection broke down.

# 5 DLL Functions

## 5.1 DLL Overview

Because most of the functions provided by the DLL do not immediately return a result the application has to provide callback functions for asynchronous events generated by the Bluetooth stack. The call-back functions are organised in a single structure that is passed to the BtInit() function. The structure members must be initialised before BtInit() is called. Unused callbacks shall be set to NULL. For more information please refer to the example programs.

## 5.2 Naming Conventions

| | |
|---|---|
| BtAskFor… | Functions that provide their data in a callback function |
| Ci… | Bluetoth management functions |
| Hc… | HCI related functions |

## 5.3 Argument handling in callbacks

All arguments that are passed to callback functions as pointers are only valid during the call of the callback function. The memory of the argument is freed or reused by the PicoConfig DLL after the callback function returns. The user has to copy the data if it will be used later.

## 5.4 Initialisation

**bool BtInit(const TPicoIfCallbacks *cb)**
This function must be called before any other DLL function can be used. The caller must provide a pointer to the array of callback functions. Unused call-backs shall be set to NULL. This function returns true if initialisation was successful, false otherwise.

Callback: None

**bool BtShutdown(void)**
Shall be called before terminating the application.

Callback: None

## 5.5 Local Configuration and Status Functions

**u32 BtGetLibraryVersion(void)**
Returns a 4 byte value with major_version.minor_version.patch_level.build_number. The MSB contains the major version number.

Callback: None

**void BtAskForStackVersion(void)**
Reads the version string of the firmware.

Callback: CiEventFwVersion(const char* Fw)

**void BtAskForLocalBD_ADDR(void)**
Reads the Bluetooth Device address of the local device.

Callback: HcEventCCReadLocalBD_ADDR(u8 HCI_ErrorCode, const u8 BdAddr[6])

**void BtAskForLocalName(void)**
Reads the friendly name of the local device. The local name provided in the callback is UTF8 encoded.

Callback: HcEventCCReadLocalName(u8 HCI_ErrorCode, const char* Name)

**void BtChangeLocalName(const char *Name)**
Sets the friendly name of local device. Name must not exceed 248 bytes in length and must be UTF8 (see www.unicode.org) encoded. (UTF8: in the simplest case use plain ASCII)

Callback: None

**bool SetBTPeer(const char * ComName, const u8* BdAddr, u8 SCN)**
Install the remote device with address BdAddr as peer in port ComName. The port must already exist. The new peer address will be used at the next open / create and stored in the system registry.
Returns true on success, false otherwise.

| | |
|---|---|
| ComName | 0 terminated string, the format is "comN" or "\\.\comN" |
| BdAddr | 6 byte binary bluetooth address as returned by an inquiry callback. |
| SCN | Service Channel Number of remote RFCOMM, this is returned by a service search in TSPPInfo. |

Callback: None

**unsigned int GetNumberOfInstalledCOMPorts(void)**
Returns the number of installed Bluetooth virtual ports.

Callback: None

**bool GetFirstInstalledCOMPort(char* buf, unsigned int* buf_len)**
**bool GetNextInstalledCOMPort (char* buf, unsigned int* buf_len)**
IN/OUT: char* buf;
char buffer that is filled with the COM name, string will be 0 terminated
COM name is in the format \\.\COMx

IN/OUT: unsigned int* buf_len
must be set to sizeof(buf) before calling.
set to actual length of COM name WITHOUT 0 byte

Returns false on failure / first illegal

Callback: None

**void BtAskForActiveStatus(void)**
Call BtAskForActiveStatus to check whether there is a Baseband connection or an Inquiry active which makes it impossible to open another Baseband connection when using SinglePoint Bluetooth hardware.

Callback: CiEventBasebandActive(bool IsActive)

## 5.6 Device Search

**void BtStartInquiry(u8 InqDuration)**
Starts an Inquiry with length of InqDuration seconds. Within this time the Bluetooth stack will send Inquiry Result Events for each remote device found. There may be multiple events for the same device. When the Inquiry is finished an InquiryComplete callback will be issued.

Callbacks: HcEventInquiryComplete(u8 HCI_ErrorCode)
HcEventInquiryResult(u8 BdAddr[6], u8 PSRM, u8 PSN, u8 CoD[3], u16 CO)

**void BtAskForRemoteName(u8 BdAddr[6])**
**void BtAskForRemoteNameEx(u8 BdAddr[6],u8 PSRM,u8 PSM,u16 CO)**
Requests the friendly name of the remote devices specified by BdAddr.
BdAddr: Bluetooth Device Address of remote device
PSRM: PageScanRepetitionMode, obtained from Inquiry Result Event, otherwise: 0
PSM: PageScanMode, obtained from Inquiry Result Event, otherwise: 0
CO: ClockOffset, obtained from Inquiry Result Event, otherwise: 0

Callback: HcEventRemoteNameRequestComplete(u8 HCI_ErrorCode, const u8 BdAddr[6], const char* Name)

## 5.7 Service Search

**void BtAskForSDPRecords(u8 BdAddr[6],u8 *uuid128)**
**void BtAskForAllSDPRecords(u8 BdAddr[6])**

Requests the services from the remote Bluetooth device with the address BdAddr. The *CiEventServiceResult* callback will be called for each service offered by the given device. If the first form is used only the services with the UUID given will be requested, if the second form is used, all services will be requested. See Bluetooth Assigned Numbers for service UUIDs.

BdAddr: Bluetooth Device Address of remote device.
uuid128 UUID of service requested.

Callbacks: CiEventServiceResult(TServiceDescription*)
CiEventServiceComplete(const u8 BdAddr[6], u16 ErrorCode)

## 5.8 Pairing

**void BtStartRfTestCon(u8 BdAddr[6],u8 SCN, bool DoAuthenticate)**

Initiates a RFCOMM connections to the specified device and service channel. A PIN request might occur before the callback is called either if the remote device requires authentication or DoAuthenticate is true.

BdAddr: Bluetooth Device Address of remote device.
SCN Server channel number as returned by *CiEventServiceResult.*
DoAuthenticate Set to TRUE if you want authentication.

Callback: CiEventRfTestConComplete(U8 BdAddr, u8 SCN, u16 ErrorCode)
HcEventPINRequest(const u8 BdAddr[6], char PIN[16+1])

# 6 Callback Functions

**u16 Reserved(u16 p1, const u8 *p2)**
Not used, shall be set to NULL in callback structure.

**void CiEventBasebandActive(bool IsActive)**
Called as a result of calling *BtAskForActiveStatus().*
IsActive TRUE if the is any base-band activity, FALSE otherwise

**void CiEventFwVersion(const char *Fw)**
Called as a result of calling *BtAskForStackVersion().*
*Fw pointer to zero terminated string identifying the firmware version of the base-band chip used. Contents depends on base-band hardware used. May be empty.

**void CiEventError(u8 Errno)**
Called in the event any error has occurred.
Errno Error number as defined in ErrorCodes.h.

**void HcEventInquiryComplete(u8 HCI_ErrorCode)**
Called when an inquiry has been completed.
HCI_Error Code Error code as defined in ErrorCodes.h. HCI_ERROR_NO_ERROR on success.

**void HcEventInquiryResult (const u8 BdAddr[6], u8 PSRM, u8 PSPM, u8 PSM, u8 CoD[3], u16 CO)**
Called for each device found during an inquiry. Depending on base-band hardware used, this callback may be called more than once for a single device.
BdAddr Bluetooth Address of the device
PSRM Page Scan Repetition Mode
PSM Page Scan Mode
CoD Class of Device, see Bluetooth Assigned Numbers for device class description.
CO Clock Offset

**void HcEventRemoteNameReqComplete (u8 HCI_ErrorCode, const u8 BdAddr[6], const char *Name)**
Called as a result of calling *BtAskForRemoteName().*
HCI_Error Code Error code as defined in ErrorCodes.h. HCI_ERROR_NO_ERROR on success. Name and BdAddr may be invalid otherwise.
BdAddr Bluetooth Address of device.
*Name Pointer to a zero terminated UTF-8 encoded string holding the name of the remote device.

**bool HcEventPINRequest(const u8 BdAddr[6], char PIN[16+1])**
This function is called if a connection or pairing attempt is made and the remote device requires authentication (and the device is not already paired). Return true if a valid PIN code is supplied in PIN or false if not. In the latter case the pairing attempt will fail.

| | |
|---|---|
| BdAddr | Bluetooth address of remote device. |
| PIN | Pointer to a buffer where the PIN code is to be returned. The PIN code is a zero terminated string with a maximum of 16 characters. |

**void HcEventCCReadLocalBD_ADDR(u8 HCI_ErrorCode, const u8 BdAddr[6])**
Called as a result of calling *BtAskForLocalBD_ADDR (.)***.**

| | |
|---|---|
| HCI_Error Code | Error code as defined in ErrorCodes.h. HCI_ERROR_NO_ERROR on success. BdAddr may be invalid otherwise. |
| BdAddr | Bluetooth address of local device. |

**void HcEventCCReadLocalName (u8 HCI_ErrorCode, const char *Name)**
Called as a result of calling *BtAskForLocalName(.).*

| | |
|---|---|
| HCI_Error Code | Error code as defined in ErrorCodes.h. HCI_ERROR_NO_ERROR on success. Name may be invalid otherwise. |
| *Name | Pointer to a zero terminated string (UTF8 encoded). |

**void Reserved2(u8 HCI_ErrorCode, const char *Version)**
Not used, shall be set to NULL in callback structure.

**void HcEventCCSetClassOfDevice (u8 HCI_ErrorCode)**
Currently not used, shall be set to NULL in callback structure.

**void HcEventCCReadClassOfDevice(u8 HCI_ErrorCode, const u8 CoD[3])**
Currently not used, shall be set to NULL in callback structure.

**void CiEventRfTestConComplete(const u8 BdAddr[6], u8 SCN, u16 ErrorCode)**
Called as a result of a pairing request *BtStartRfTestCon()*

| | |
|---|---|
| BdAddr | Bluetooth address of remote device. |
| SCN | Remote device server channel number. |
| ErrorCode | ErrorCode as defined in ErrorCodes.h. BT_NO_ERROR on success. |

**void CiEventServiceResult(TServiceDescription *sd)**
Called for every service descriptor found on a remote device

| | |
|---|---|
| *sd | Pointer to a data structure of type *TServiceDescription.* |

**void CiEventServiceComplete(const u8 BdAddr[6], u16 ErrorCode)**
Called after all services from a remote device have been requested.

| | |
|---|---|
| BdAddr | Bluetooth address of remote device. |
| ErrorCode | ErrorCode as defined in ErrorCodes.h. BT_NO_ERROR on success. |

# 7 Structures

### TPicoIfCallbacks

A structure of this type must be allocated and initialised by the application before the BtInit() function is called. It holds the pointers to the callback functions used by the DLL to inform the application about asynchronous events. Unused callback functions shall be set to NULL. The structure member *CallbackLength* shall be set to the size of the structure (in bytes).

```
typedef struct {
        u32     CallbackLength;
        u16     (*Reserved)(u16 p1, const u8 *p2);
        void    (*CiEventBasebandActive)(bool IsActive);
        void    (*CiEventFwVersion)(const char *Fw);
        void    (*CiEventError)(u8 Errno);
        void    (*HcEventInquiryComplete)(u8 HCI_ErrorCode);
        void    (*HcEventInquiryResult)(const u8 BdAddr[6], u8 PSRM, u8 PSPM, u8 PSM,
                u8 CoD[3], u16 CO);
        void    (*HcEventRemoteNameReqComplete)(u8 HCI_ErrorCode, const u8
                BdAddr[6], const char *Name);
        bool    (*HcEventPINRequest)(const u8 BdAddr[6], char PIN[16+1]);
        void    (*HcEventCCReadLocalBD_ADDR) (u8 HCI_ErrorCode, const u8
                BdAddr[6]);
        void    (*HcEventCCReadLocalName) (u8 HCI_ErrorCode, const char *Name);
        void    (*Reserved2) (u8 HCI_ErrorCode, const char *Version);
        void    (*HcEventCCSetClassOfDevice) (u8 HCI_ErrorCode);
        void    (*HcEventCCReadClassOfDevice)(u8 HCI_ErrorCode, const u8 CoD[3]);
        void    (*CiEventRfTestConComplete)(const u8 BdAddr[6], u8 SCN, u16
                ErrorCode);
        void    (*CiEventServiceResult)(TServiceDescription *sd);
        void    (*CiEventServiceComplete)(const u8 BdAddr[6], u16 ErrorCode);
} TPicoIfCallbacks;
```

**TServiceDescription**

This structure is returned by the CiEventServiceResult callback and holds all information about a single service record.

```
typedef struct {
        u32     Version;                filled with SDP_SERVICE_DESCRIPTION_VERSION
        u8      BdAddr[6];              the Bluetooth address of the remote device
        u8      UUID[16];               UUID128 of the service
        u16     PSM;                    L2CAP Protocol Service Multiplexer Number.
        char*   ServiceName;            Name of the service
        char*   ServiceDescription;     Verbose descritpion of service
        char*   ProviderName;

        TSDPServiceType ServiceType;    selects union element, can be one of the
                                        following enum types:
                                        ST_unknown
                                        ST_SPP              serial port profile
                                        ST_HCRP             HCRP
                                        ST_ObjPush,         OBEX Objext Push
                                        ST_ObexFileTransfer OBEX file transfer

        union {
                TSPPInfo                spp;
                THCRPInfo               hcrp;
                TobjPushInfo            obpush;
                TObjexFileTransferInfo  obexft;
        } u;
}
```

Note that services based on the Bluetooth profiles Dial Up Networking, Lan Access or Fax will return a Serial Port Profile type service description but different UUIDs.

For more information see ServiceDC.h.

# 8 Revision History

| Revision | Date | Description |
|---|---|---|
| 1.0 | Oct. 1, 2003 | Initial Version |