# BlueCode+

## DesignGuide

# Content

# 1 Purpose

This DesignGuide documents the integration of Stollmann´s Bluetooth Upper Layer Stack BlueCode+ into applications. It addresses developers of hardware and software environments for BlueCode+.

This documentation functions as a recommendation to the best of our knowledge. Stollmann does not assume any liability for the accuracy of information included herein and will not be liable for any damages related to or caused by the use of this DesignGuide.

# 2 Product Description

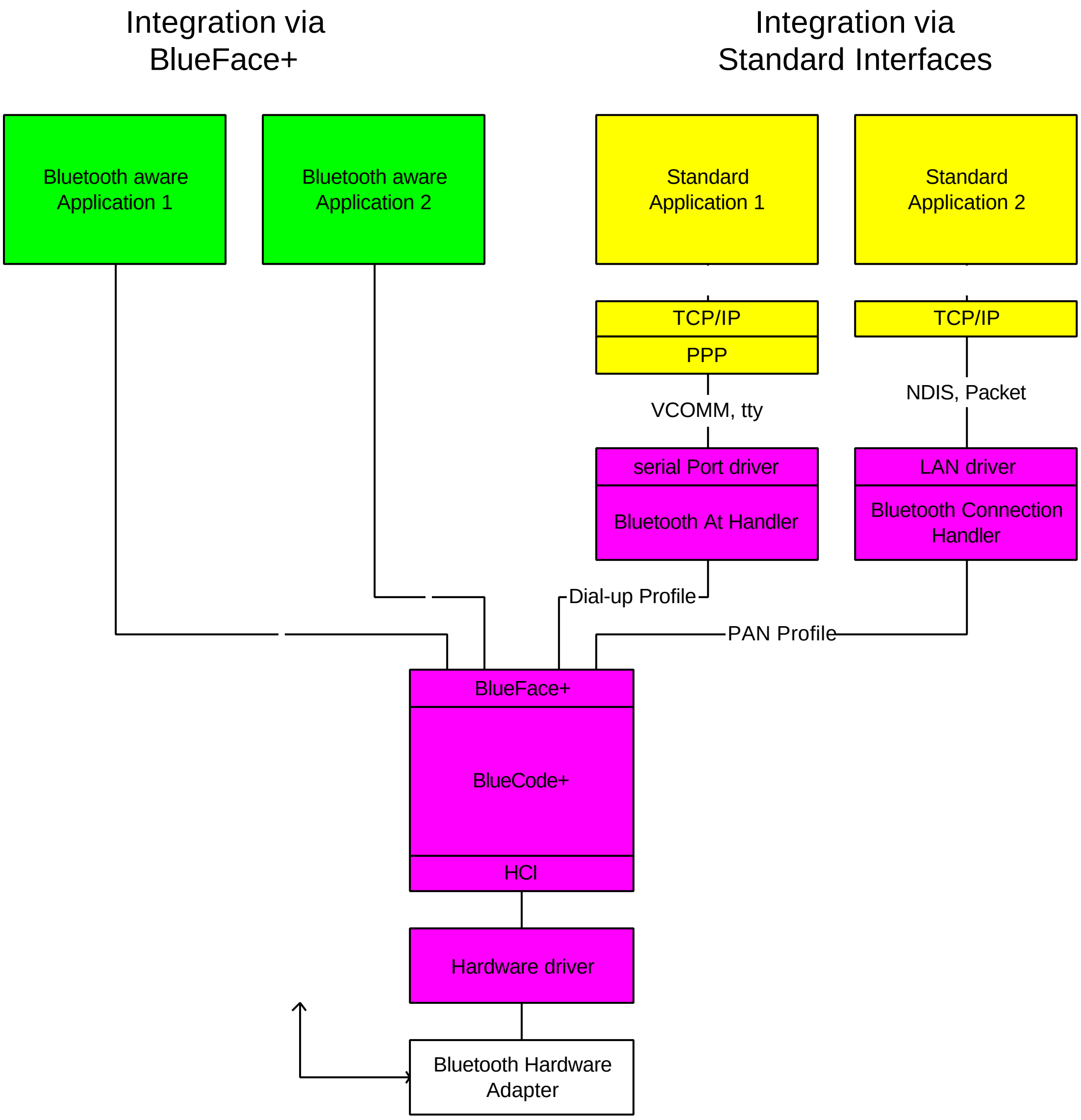
BlueCode+ is an Bluetooth Upper Layer Stack, comprising the Layers L2CAP, SDP, RFCOMM, BNEP, AVDTP, TCS.BIN and other protocol modules. It supports a great variety of Bluetooth profiles.

With regard to the hardware, BlueCode+ supports the HCI interface that is standardized in the Bluetooth specification.

BlueCode+ features the OSIF interface to the operating system. It allows the integration into various embedded and PC-based systems. In case there is no operating system OSIF can also be used stand-alone.

There are two possible strategies to integrate BlueCode+ into the application software:

- BlueCode+ features the BlueFace+ API that allows direct access to the Bluetooth functions and profiles as well as direct access to the layers below.
- On top of BlueFace+ interface drivers are built, emulating standard system drivers like virtual serial ports. They contain control functions that i.e. can establish the Bluetooth link.



Stollmann provides the BlueFace+ specification and a Development Kit for the programming of BlueFace+API.

Standard interfaces for the most important Bluetooth and operating system platforms are either available from Stollmann or may be developed on basis of BlueFace+ API with  the development kit.

# 3 Applications

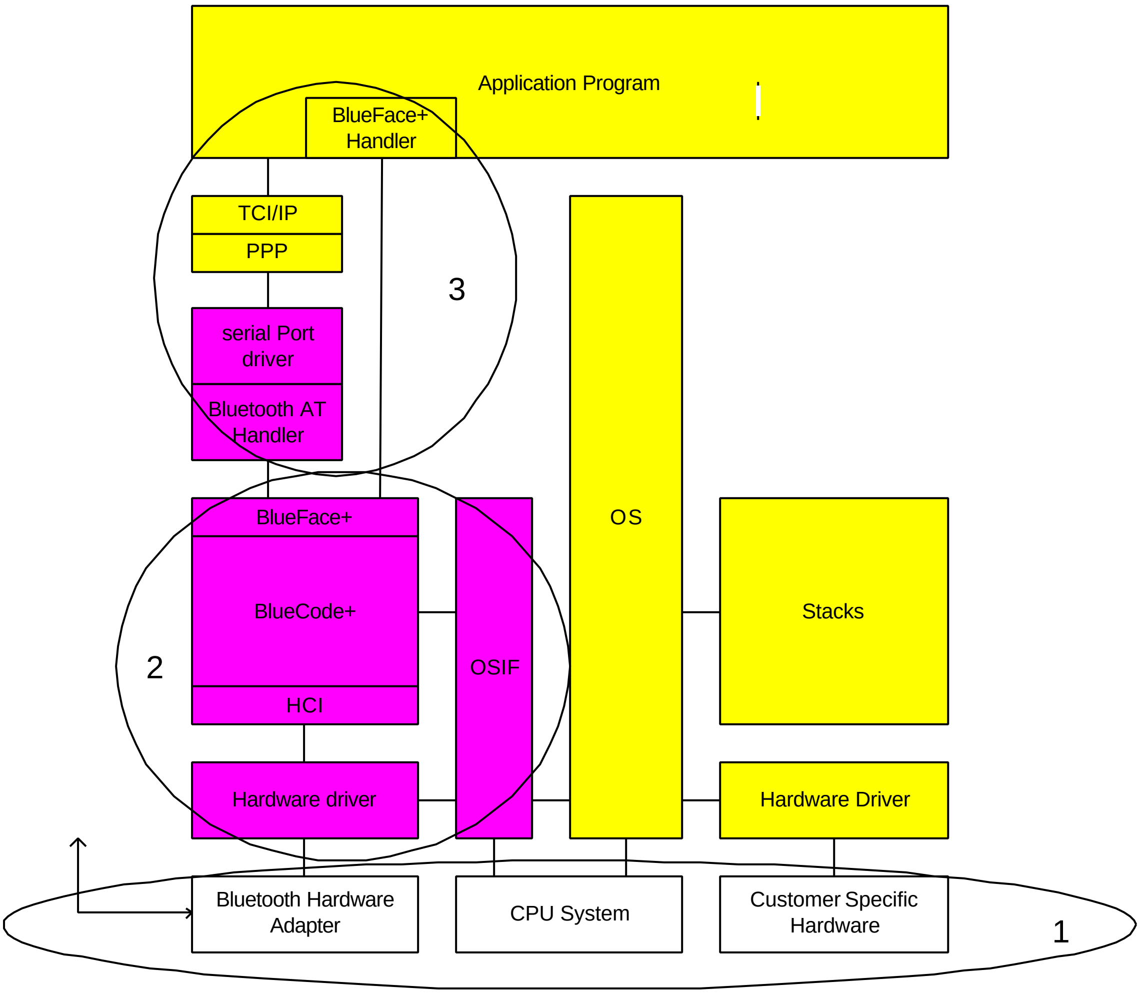
BlueCode+ can be implemented in different applications. Some typical are described in this chapter:

## 3.1 Embedded System Integration

BlueCode+ runs on a hardware platform from 16-bit CPUs onward with at least 90 kByte Flash and 10 kByte RAM. A typical integration cycle is listed as follows:

- Analysis of requirements, specification
- Development of hardware platforms (drawing, step 1)
- Porting of OSIF on hardware platform and operating system of target system (drawing, step 2)
- Adaption of the low level hardware driver to the HCI hardware interface
- Testing of BlueCode+ on target system with a test program
- Development of interface between BlueCode+ and application program respectively adaption of application program to BlueFace+ (drawing, step 3)
- Integration, test and acceptance test of the entire system

Embedded System Integration



Stollmann supports all development steps respectively performs the development on request.

## 3.2 PC Integration Windows

BlueCode+ is integrated into PC drivers for Windows XP, 2000, 98 and ME.

Hardware drivers for the serial, USB and PCMCIA interface are available. The hardware drivers can easily be adapted to support different hardware interfaces.

The protocol drivers are completely integrated into the kernel mode of the operating system. These are .SYS drivers of WDM for Windows XP/2000/NT and .VXD drivers for Windows ME/98. The implemented drivers are independent of a specific driver´s environment.

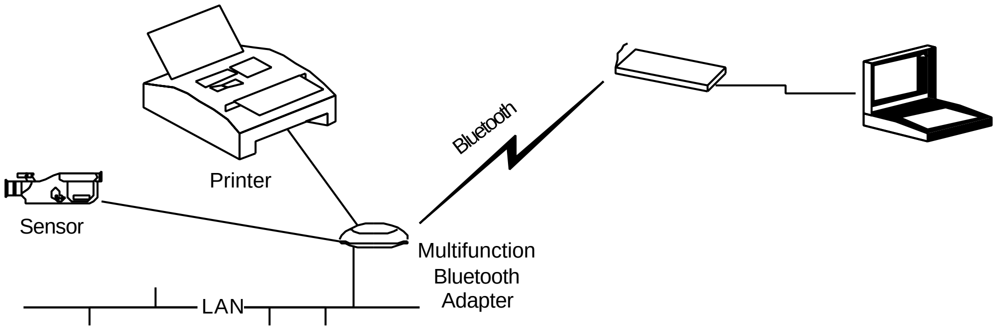
Please refer to the BlueWinDriver+ Design Guide for more details.

## 3.3 PC Integration - further Operating Systems

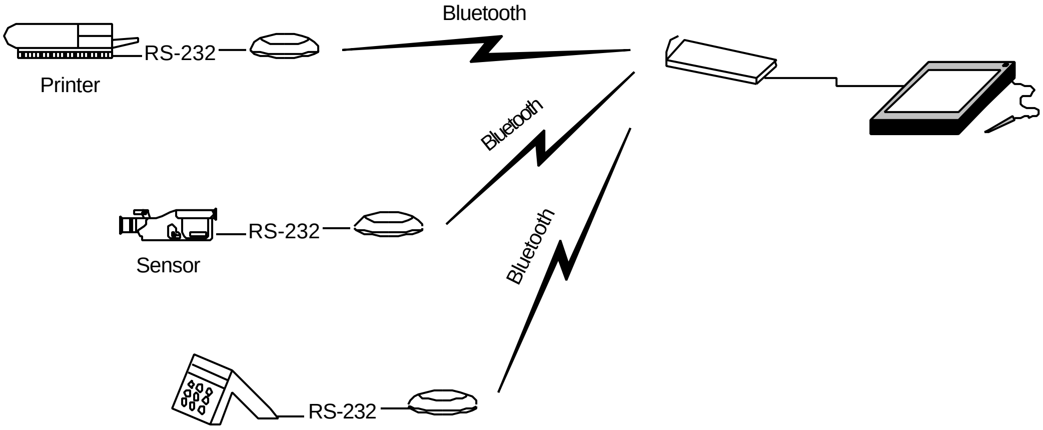
The integration of BlueCode+ into different PC systems, e.g. Linux, is made possible through the adaption of OSIF and implementation of respective system drivers. You can also use BlueFace+ for integration with the application program.

## 3.4 BlueCode+ in Multipoint Environment

BlueCode+ can establish several Bluetooth connections simultanously. These can be multiple logical connections on a physical Bluetooth Link.

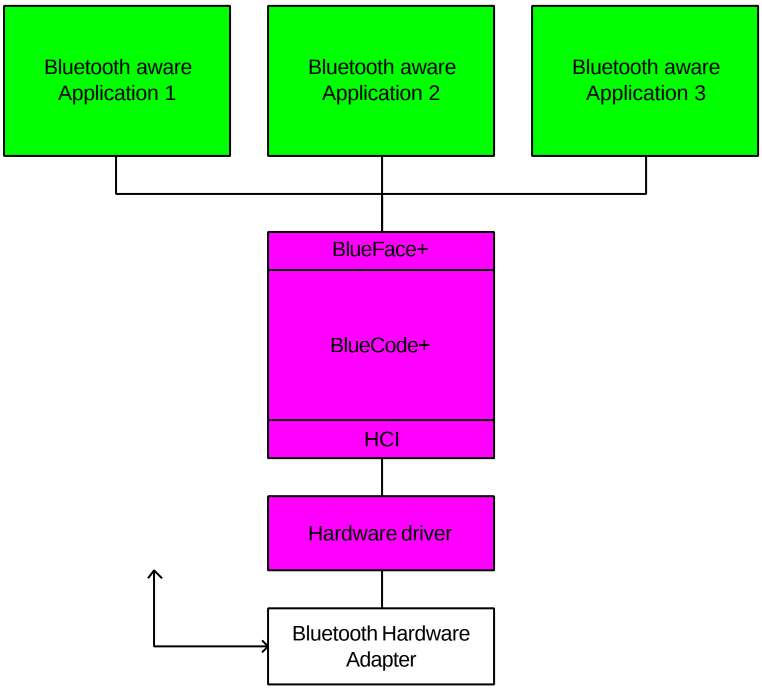


It is also possible to establish several physical links.



## 3.5 BlueFace+ in Multi Application Environments

Several applications may have access to the BlueFac+ API at the same time and thus use the Bluetooth connections in parallel. This mechanism is described in the BlueFace+ specification.


Bluetooth aware Application 1
Bluetooth aware Application 2
Bluetooth aware Application 3
BlueFace+
BlueCode+
HCI
Hardware driver
Bluetooth Hardware Adapter

# 4 BlueCode+ Ressources, System Requirements, CPU, MIPS

- BlueCode+ is V1.0b + critical errata certified
- BlueCode+ is V1.1 certified
- BlueCode+ is written in Standard ANSI C language

## 4.1 CPU

The following CPU platforms are currently supported:

- ARM 7
- Infineon C161
- Intel 8018x
- Texas DSP 320 C5X

The port to additional platforms is easily done

## 4.2 System Requirements

The requirements to the system are dependent on the requested functionality and the CPU and OS platforms. The information following are therefore supposed to be a clue for a first estimation.

Basis: Infineon C161, Serial Port Profile, Dial-up Profile, Lauterbach Compiler, Large Memory Model, ISDN Gateway with 128 kbps

- 4 MIPS for Bluetooth Stack
- 90 Kbyte ROM
- 16 Kbyte RAM

### 4.2.1 C161 on BlueTA+ (as of 4.1.2002)

| **Modul** | **Funktionen** | **ROM Debug** | **ROM Stripped** | **RAM** |
|---|---|---|---|---|
| BlueFace+ | Utils | 38.750 | 28.000 | 434 |
| SDP | Client | | | |
| | Server | 18.200 | 14.900 | |
| | Both | | | 568 |
| RFCOMM | | 21.200 | 13.350 | 536 |

| | | | | |
|---|---|---|---|---|
| L2CAP | | 30.200 | 20.400 | 836 |
| HCI | | 28.450 | 21.000 | 586 |
| OSIF | | 25.000 | 25.000 | |
| datapools | Upstream | | | 5.040 |
| | Downstream | | | 3.920 |
| | System | | | 4.608 |
| **Total** | | **161.800** | **122.650** | **16.528** |

### 4.2.2 ARM7 on BlueRS+ (as of 4.1.2002)

| **Modul** | **Funktionen** | **ROM Debug** | **ROM Stripped** | **RAM** |
|---|---|---|---|---|
| BlueFace+ | Utils | | 14.400 | |
| SDP | Client/Server | | 13.000 | |
| RFCOMM | | | 8.700 | |
| L2CAP | | | 11.550 | |
| HCI | | | 15.250 | |
| **Total** | | | **62.900** | |

## 4.3 Supported Profiles

| **Profile** | **Scheduled** |
|---|---|
| GAP | available |
| SDAP | available |
| Serial Port | available |
| Dial-up Networking | available |
| LAN Access | available |
| PAN | available |
| OBEX | available |
| Fax | available |
| Headset | available |

| | |
|---|---|
| Handsfree | available |
| SIM Access | available |
| Phone Access | available |
| Cordless Telephony (CTP) | available |
| Intercom | available |
| ISDN (CIP) | available |
| HCRP | available |
| Basic Printer Profile | not scheduled yet |
| Human Interface Devices | available |
| Basic Imaging Profile | not scheduled yet |
| A2DP | available |

The implementation schedule can be adapted to customers requirements.

## 4.4 OS Support

The following operating systems are supported:

- OSIF (Stollmann Kernel)
- Windows XP/2000/98
- embedded Linux

Following ressources are supposed to be provided by an operating system in order to enable an easy porting of BlueCode+.

- Timer (about 10 ms)
- Memory Management (static or dynamic)
- The stack has to be called by the host OS periodically

## 4.5 Chipsets

### 4.5.1 Embedded Solutions

The stack has been embedded in the follwoing chipsets:

- ST Microelectronics
- Zeevo
- Philips

The customer may use the BlueFace+ API or a standard interface like AT-commands to interface to the chipset. The application may be embedded in the Bluetooth chipset, provided that the total requirements for memory or processing power are not exceeded.

### 4.5.2 HCI Support

BlueCode+ is tested with the HCI interface of the following Bluetooth Chipsets:

- Zeevo
- Silicon Wave
- Ericsson
- Atmel
- Infineon
- Broadcom
- CSR
- Philips
- WavePlus
- ST Microelectronics

The chipsets must support at least one of the interfaces below to ensure an easy adaption.

- HCI-H2 (USB)
- HCI-H4 (UART)

Additional or manufacturer specific HCI transport layers may be adapted on request.

# 5 BlueFace+ Basic Mechanism

The BlueFace+ API is a software interface to access the BlueCode+ Upper Layer Bluetooth Stack of Stollmann. The specification is available by Stollmann. The main design goals of BlueFace+ are:

- Message based
- OS independent
- Hardware independent
- Support for a wide range of Bluetooth features

## 5.1 Supported Features

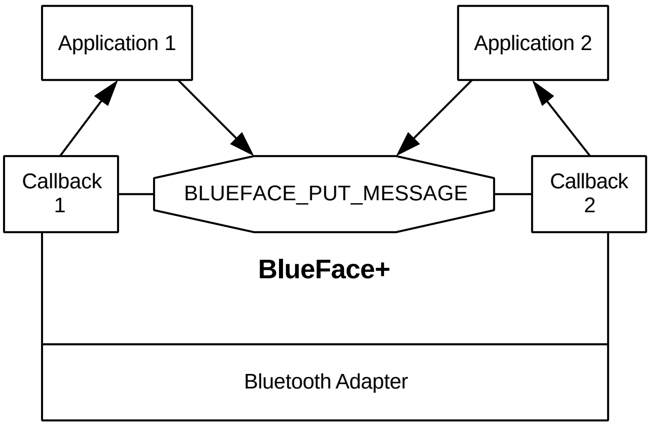
- RFCOMM, SDP, AVDTP, BNEP, L2CAP and TCS.BIN services
- Basic connection oriented services, such as connection setup and cleardown.
- Multiple concurrent applications
- Operating-system independent messages
- Operating-system dependent exchange mechanism for optimum operating system integration
- Asynchronous event-driven mechanism, resulting in high throughput
- Several ACL connections (multiple L2CAP links)
- SCO connections for voice (answ. machine, headset)
- Several RFCOMM data links within one RFCOMM session
- Multipoint
- Support for direct L2CAP access
- Support for direct HCI access
- Scatternet support (dependent on chipset)

## 5.2 Upcoming Features

- Power Management
- Connectionless channels

## 5.3 Message Exchange Mechanism

Every application program residing on BlueFace+ issues one callback function to the BlueFace+ while registering the application. BlueFace+ transmits all messages to the application using this callback function.



The BlueFace+ messages contain data and signaling information, covering all types of message exchange between the application and the Bluetooth stack. No additional API is required.

The messages are queued in BlueFace+, releasing the application from scheduling management.

The BlueFace+ API works completely asynchronous. No blocking of the system by waiting for the completion of a call can occur as it would be the case for synchronous calls.

# 6 BlueFace+ Messages

Messages are logically grouped:

- Messages concerning connection control
- Messages concerning general data transmission
- Messages concerning protocol specific data transmission
- Administrative and other messages

The format of the BlueFace+ messages is set up as follows:

| | | | | |
|---|---|---|---|---|
| Message Header | Parameter 1 | Parameter 2 | Parameter... | Parameter n |

The table below provides an overview of the BlueFace+ messages. It contains all presently defined messages. The BlueFace+ specification includes a detailed description of their functions.

| Message | Value | Description |
|---|---|---|
| BT_CON_REQ | 0x04 | Initiates outgoing connection |
| BT_CON_CONF | 0x05 | Local confirmation of request |
| BT_CON_IND | 0x06 | Indication of incoming connection request |
| BT_CON_RESP | 0x07 | Response to incoming connection request |
| BT_CON_ACT_IND | 0x08 | Indicates activation of connection |
| BT_CON_ACT_RESP | 0x09 | Response to indication |
| BT_DISC_REQ | 0x0A | Initiates clearing down of connection |
| BT_DISC_CONF | 0x0B | Local confirmation of request |
| BT_DISC_IND | 0x0C | Indicates clearing of connection |
| BT_DISC_RESP | 0x0D | Response to indication |
| | | |
| BT_DATA_REQ | 0x20 | Initiates sending data over connection |
| BT_DATA_CONF | 0x21 | Local confirmation of request |
| BT_DATA_IND | 0x22 | Indicates incoming data on connection |

| | | |
|---|---|---|
| BT_DATA_RESP | 0x23 | Response to indication |
| | | |
| BT_RFC_MSC_REQ | 0x40 | Request flow control command |
| BT_RFC_MSC_CONF | 0x41 | Local confirmation of request |
| BT_RFC_MSC_IND | 0x42 | Flow control indication |
| BT_RFC_MSC_RESP | 0x43 | Response to indication |
| BT_RFC_RLS_REQ | 0x44 | Request Remote Line Status |
| BT_RFC_RLS_CONF | 0x45 | Local confirmation of request |
| BT_RFC_RLS_IND | 0x46 | Indicates Remote Line status |
| BT_RFC_RLS_RESP | 0x47 | Response to indication |
| BT_RFC_RPN_REQ | 0x48 | Request Remote Line Parameters |
| BT_RFC_RPN_CONF | 0x49 | Local confirmation of request |
| BT_RFC_RPN_IND | 0x4A | Indicates Remote Line Parameters |
| BT_RFC_RPN_RESP | 0x4B | Response to indication |
| | | |
| BT_SDP_SEARCH_REQ | 0x60 | Search remote SDP server for service |
| BT_SDP_SEARCH_CONF | 0x61 | Local confirmation of request |
| BT_SDP_ATTRIBUTE_REQ | 0x62 | Search remote SDP server for attributes |
| BT_SDP_ATTRIBUTE_CONF | 0x63 | Confirmation of request |
| BT_SDP_SEARCH_ATTRIBUTE_REQ | 0x64 | Search remote SDP server for service/attribute combined |
| BT_SDP_SEARCH_ATTRIBUTE_CONF | 0x65 | Local confirmation of request |
| BT_SDP_REGISTER_REQ | 0x66 | Register service with local SDP server |
| BT_SDP_REGISTER_CONF | 0x67 | Local confirmation of request |
| BT_SDP_RELEASE_REQ | 0x68 | Release service from local SDP server |
| BT_SDP_RELEASE_CONF | 0x69 | Local confirmation of request |
| | | |
| BT_HCI_INQUIRY_REQ | 0x80 | Initiates Bluetooth inquiry scan |

| | | |
|---|---|---|
| | | procedure |
| BT_HCI_INQUIRY_CONF | 0x81 | Indicates inquiry scan result |
| | | |
| BT_HCI_NAME_REQ | 0x82 | Requests name from remote Bluetooth adapter |
| BT_HCI_NAME_CONF | 0x83 | Indicates name request results |
| BT_HCI_PAGEMODE_REQ | 0x84 | Request to set Pagescan / Inquiry Mode |
| BT_HCI_PAGEMODE_CONF | 0x85 | Local confirmation of request |
| BT_HCI_LNAME_REQ | 0x86 | Request to set local adapter name |
| BT_HCI_LNAME_CONF | 0x87 | Confirmation of adapter name request |
| BT_HCI_CLASS_REQ | 0x88 | Request to set device class |
| BT_HCI_CLASS_CONF | 0x89 | Confirmation of class request |
| | | |
| BT_HCI_AUTH_REQ | 0xA0 | Request to authenticate link |
| BT_HCI_AUTH_CONF | 0xA1 | Confirmation of authentication request |
| BT_HCI_ENCRYPTION_IND | 0xA2 | Indication of link encryption status |
| BT_HCI_KEY_IND | 0xA3 | Indication for PIN or Linkkey |
| BT_HCI_KEY_RESP | 0xA4 | Response to indication |
| BT_HCI_NEWKEY_IND | 0xA5 | Indication for new linkkey |
| BT_HCI_NEWKEY_RESP | 0xA6 | Response to indication |
| | | |
| BT_HCI_TUNNEL_REQ | 0xC0 | Request to Send HCI Command |
| BT_HCI_TUNNEL_CONF | 0xC1 | Confirmation of request |
| BT_HCI_TUNNEL_IND | 0xC2 | Indication of HCI event data |
| BT_HCI_TUNNEL_RESP | 0xC3 | Response to indication |
| | | |
| BT_COM_LINK_REQ | 0x0100 | Request to redirect connection to secondary application |
| BT_COM_LINK_CONF | 0x0101 | Local confirmation for request |

| | | |
|---|---|---|
| BT_COM_LINK_IND | 0x0102 | Indication that connection is redirected to secondary application |
| BT_COM_LINK_RESP | 0x0103 | Response to indication |
| BT_COM_STAT_REQ | 0x0104 | Request for link statistics |
| BT_COM_STAT_CONF | 0x0105 | Delivery of link statistics to application |
| | | |
| BT_APP_BROADCAST_REQ | 0x0120 | Request to broadcast message |
| BT_APP_BROADCAST_CONF | 0x0121 | Confirmation of broadcast |
| BT_APP_BROADCAST_IND | 0x0122 | Indication of broadcast message |
| | | |
| BT_PNP_REMOVE_IND | 0x0140 | Indication that Bluetooth adapter is removed |
| | | |
| BT_ACT_IND | 0x7F00 | Indication of BlueFace+ stack activation |

# 7 BlueFace+ Function Calls

The BlueFace+ specification defines the basic functions with respect to the exchange of messages.

## 7.1 Registering a BlueFace+ Application Program

The function BLUEFACE_REGISTER registers an application program to the BlueFace+ API. This is necessary before any data can be exchanged. The following parameters are set:

- select a specific controller
- register the application provided callback function
- register the application provided callback context
- check for version compatibility between application and BlueFace+ implementation

## 7.2 Messages from the Application to BlueFace+

Messages from an application are queued by the BLUEFACE_PUT_MESSAGE function call. The application handle is used to identify the application context for the message.

A message queue overrun returns a  BLUEFACE_PUT_MESSAGE error.

## 7.3 Messages from BlueFace+ to the Application

Messages from BlueFace+ to the application are sent with an application provided callback function. Therefore no polling is required.

## 7.4 Releasing a BlueFace+ Application

To release the link between BlueFace+ and the application, the application issues an BLUEFACE_RELEASE operation. This also frees all resources associated with the application.

## 7.5 Link Handle from BlueFace+ to Application

To identify the different links between BlueFace+ and one or more applications a unique link handle is issued by BlueFace+ for each logical connection. The link handle for incoming calls is set in BT_CON_IND message. The link handle for outgoing calls is set in BT_CON_CONF message. The application must use the link handle in every downstream message (requests and responses) related to this logical connection.

## 7.6 Link Context from Application to BlueFace+

An application may associate a unique link context with each logical connection to simplify the internal dispatch of messages coming from BlueFace+. The link context for outgoing calls can be defined to BlueFace+ in BT_CON_REQ message. The link context for incoming calls can be set in BT_CON_RESP message. BlueFace+ will provide the link context to the application in each upstream message (indication and confirmation) related to this logical connection.

# 8 Principal Application Flow of a BlueFace+ Connection

The typical flow of a connection setup and data transmission via BlueFace+ is demonstrated in the following table. A bluetooth connection is set up, data transmitted and received, and the connection closed again.

| Dir | Message | Parameters | Comment |
|---|---|---|---|
| ↓ | BT_CON_REQ | context, bd, psm=RFCOMM, serverChannel | Active connection setup |
| ↑ | BT_CON_CONF | handle,context | confirmed |
| ↑ | BT_CON_ACT_IND | context | connection is active |
| ↓ | BT_CON_ACT_RESP | handle | confirmed |
| ↑ | BT_RFC_MSC_IND | context, status | Initial command to set local line status |
| ↓ | BT_RFC_MSC_RESP | handle | confirmed |
| | | | |
| ↓ | BT_DATA_REQ | handle, packet | Data transmission: transmit |
| ↑ | BT_DATA_CONF | context, cause=0 | confirmed |
| | | | |
| ↑ | BT_DATA_IND | context, packet | Data transmission: receive |
| ↓ | BT_DATA_RESP | handle | confirmed |
| | | | |
| ↓ | BT_DISC_REQ | handle | Active connection closure |
| ↑ | BT_DISC_CONF | context | confirmed |
| ↑ | BT_DISC_IND | context, cause | channel closed |
| ↓ | BT_DISC_RESP | handle | confirmed |

# 9 BlueFace+ Sample Code Fragments

The following sample code fragments illustrate the use of the BlueFace interface.

They show the use of BlueFace functions and the generation / analysis of BlueFace messages. They do however not constitute a full functional program.

## 9.1 Common Declarations

In order to use the BlueFace+ interface, the headerfile blueface.h must be included:

```
#include <blueface.h>
```

## 9.2 Obtain Blueface+ function adresses (MS Windows environment)

Under MS Windows environment, dynamic linkage to the DLL that contains the BlueFace functions is performed:

```
HINSTANCE               blueFaceDll          = NULL;
PblueFaceRegister       pblueFaceRegister    = NULL;
PblueFaceRelease        pblueFaceRelease     = NULL;
PblueFacePutMessage     pblueFacePutMessage  = NULL;

blueFaceDll = LoadLibrary(„blueface.dll“);
pblueFaceRegister = (PblueFaceRegister)
         GetProcAddress(blueFaceDll,"blueFaceRegister");
pblueFaceRelease  = (PblueFaceRelease)
         GetProcAddress(blueFaceDll,"blueFaceRelease");
pblueFacePutMessage = (PblueFacePutMessage)
         GetProcAddress(blueFaceDll,"blueFacePutMessage");
```

Now all BlueFace entry functions are available for call.

## 9.3 Register to Blueface+ interface

```
TblueFaceReg            reg;
blueFaceStatus          res;
BAPPHANDLE              app                  = NULL;
```

```
reg.appContext = myContext;           // any local useful
                                      // context might be used here
reg.callBack   = blueFaceCallBack;    // adress of local message
                                      // displatch function
reg.version    = BLUEFACE_VERSION;    // Set version indication

res = pblueFaceRegister(&reg,sizeof(reg));
// if res == blueFaceNoError, the registration process started, a
// BT_REGISTER_CONF message will arrive
```

## 9.4 Send BlueFace message

The following code fragment illustrates how a BT_CON_REQ message (create an outgoing message, here for RFCOMM) is constructed and send:

```
TblueFaceMsg   msg;
blueFaceStatus status;
msg.length = blueFaceMsgSize + sizeof(msg.p.connectRequest);
msg.command                              = BT_CON_REQ;
msg.p.connectRequest.bLinkContext        = myLinkContext
msg.p.connectRequest.frameSize           = 0; // use default
msg.p.connectRequest.psm                 = BLUEFACE_PSM_RFCOMM;
msg.p.connectRequest.p.rfc.creditBased   = TRUE;
msg.p.connectRequest.p.rfc.serverChannel = 1; // use channel 1
msg.p.connectRequest.p.rfc.mscState      = 0; //
msg.p.connectRequest.p.rfc.encrypted     = TRUE;
memcpy(msg.p.connectRequest.bd,myBD, BD_ADDR_SIZE); // set target address
status = pBlueFacePutMessage(app,&msg);
// if status == blueFaceNoError, the message was correctly send
```

Note: it is important to set the correct message length in all messages, otherwise the message contents will be corrupted.

## 9.5 Receive and analyse BlueFace message

Upstream messages will be transferred by activating the regitration-provided callback function, the context value provided will be the context value of the registration (myContext).

```
void APIENTRY blueFaceCallBack(BAPPCONTEXT context, LPblueFaceMsg pmsg)
{
switch (pmsg->command)
    {
    case BT_REGISTER_CONF:
      // confirmation for registration arrived
      if (pmsg->p.registerConfirmation.status != 0)
        // registration failed
      else
        // save app value for later use
        app = pmsg->registerConfirmation.app;
     break;

    case BT_CON_CONF:
      // Confirmation for BT_CON_REQ
      if (pmsg->p.connectConfirmation.status != 0)
         // error during connection setup, handle the error
        else
         // connection setup accepted, will proceed, store the link handle
         bLinkHandle = pmsg->p.connectConfirmation.bLinkHandle
      break;

    case BT_CON_ACT_IND:
       //  Connection is active now, data exchange is possible now
       // issue BT_CON_ACT_RESP message
       msg.command                      = BT_CON_ACT_RESP;
       msg.length                       = blueFaceMsgSize +
                                          sizeof(msg.p.conActResponse);
       msg.p.conActResponse.bLinkHandle = bLinkHandle;
       blueFacePutMessage(app, &msg);
       break;

   case BT_DATA_IND:
      // data from remote peer arrived, consume the data and
      // respond the message
      msg.command                     = BT_DATA_RESP;
      msg.length                      = blueFaceMsgSize +
                                        sizeof(msg.p.dataResponse);
      msg.p.dataResponse.bLinkHandle = bLinkHandle;
      status=blueFacePutMessage(appHandle,&msg);
      break;
   case BT_DISC_IND:
    // connection to remote peer is lost
    // ACK the message
    msg.command                          = BT_DISC_RESP;
    msg.length                           = blueFaceMsgSize +
                                           sizeof(msg.p.disconnectResponse);
    msg.p.disconnectResponse.bLinkHandle = bLinkHandle;
    blueFacePutMessage(appHandle,&msg);
    bLinkHandle = NULL; // link handle is no more valid
    break;


   case BT_RELEASE_CONF:
    app = NULL;  // app handle no longer valid
    break;
  } /* switch */
} /* callback */
```

### 9.6 Release from BlueFace+ interface

In order to release the application registration from the interface, the blueFaceRelease function is called:

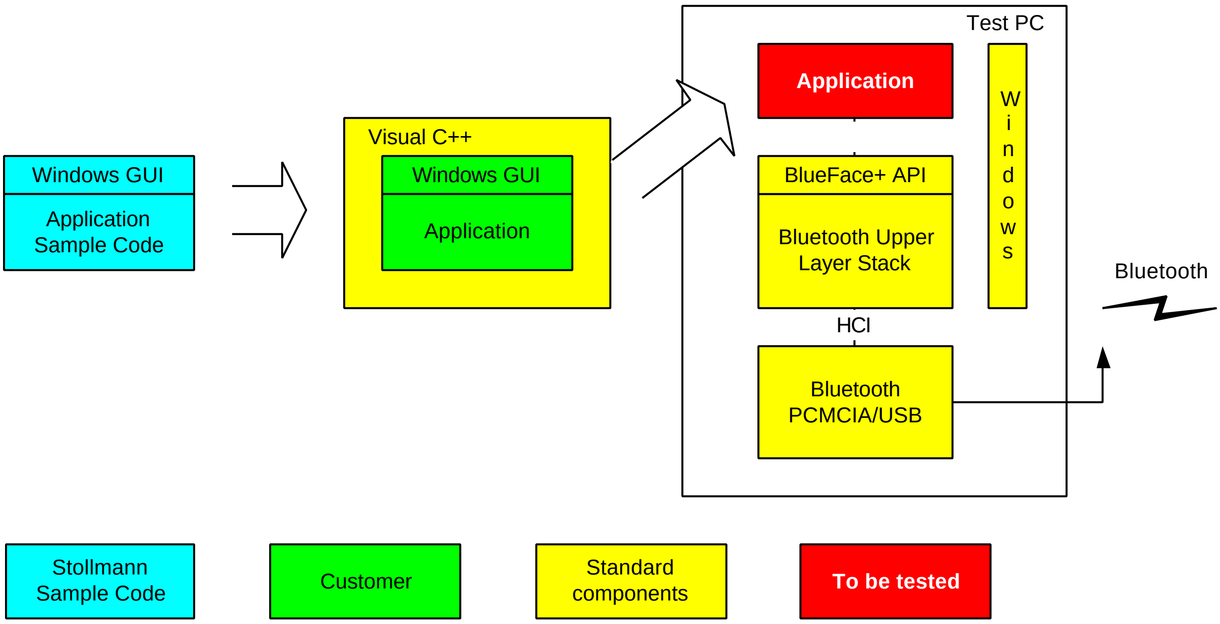
```
pblueFaceRelease(app);
// wait for BT_RELEASE_CONF message
```

## 10 Application Development Process

There are several ways to develop BlueFace+ based applications. For all platforms the sample code of Stollmann is the starting point.

### 10.1 Development of Windows Applications

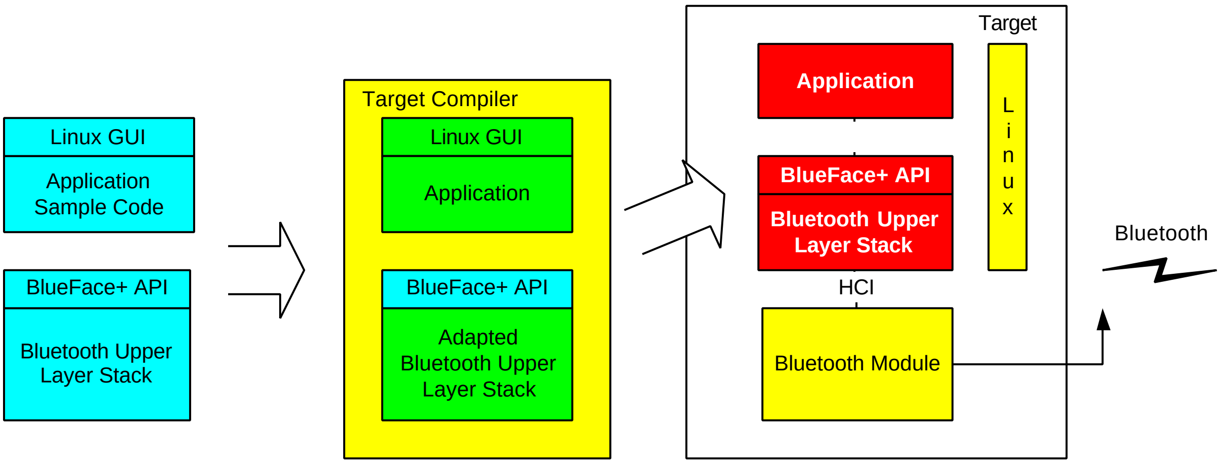
For the development of PC applications under Windows you may use the application sample code contained in the Bluetooth development kit of Stollmann.



The sample code may be compiled to test it on a Windows system. It can then be modified for customer specific applications.

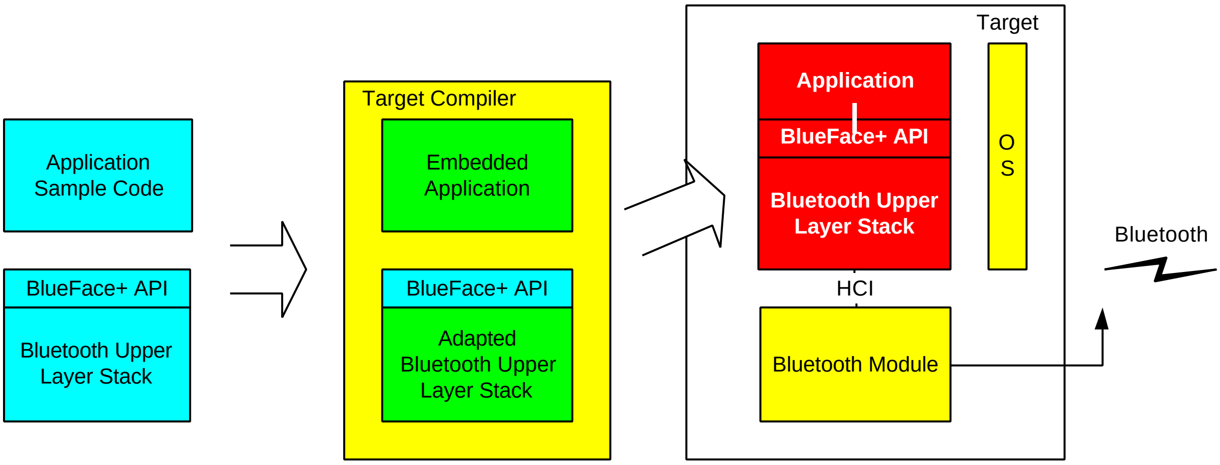
## 10.2 Other PC operating systems

For other PC operatings systems the sample application may be modified and compiled for the respective hardware and software platform. The Upper Layer Stack can be compiled to a standard driver and installed in the system.



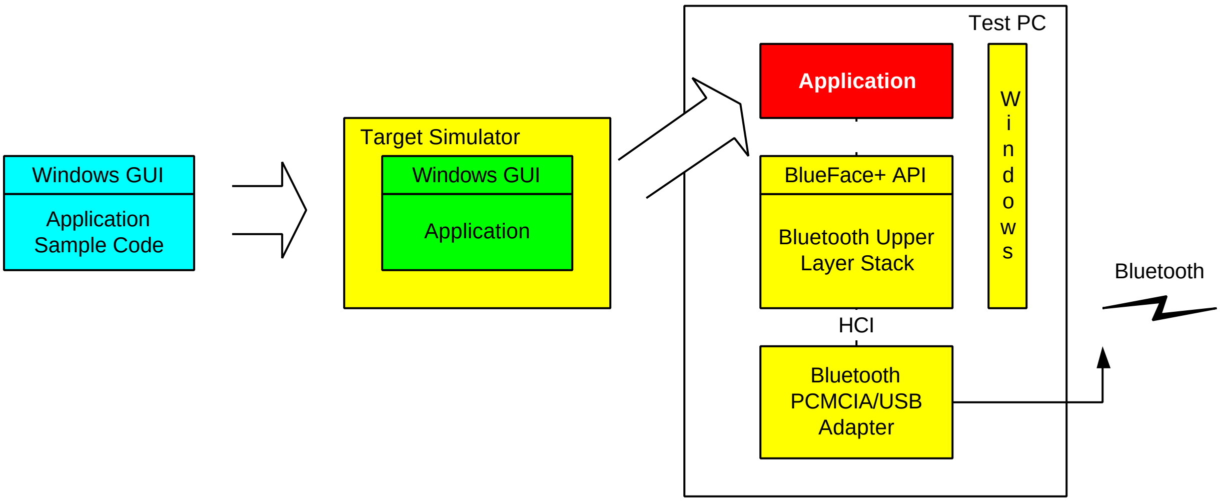
## 10.3 Embedded System development

In case of an embedded system the sample application will be modifed and compiled with the Upper Layer Stack and further modules for the embedded system.



## 10.4 Embedded System Simulation

To test the application on a PC without porting the Bluetooth Upper Layer Stack to the embedded system first some embedded development environments allow to test the „embedded“ code in a standard PC environment.



In a second step the application and the Bluetooth Upper Layer Stack will be integrated on the embedded system according to the previous chapter.