



Bluetooth's Next Challenge

Connecting Everything to Everything

by Greg Burns, Open Interface

Bluetooth chips are now shipping at a rate of over 3 million per week, more than all other short range RF wireless chips combined (November 2004). Bluetooth Wireless Technology defines an amazing collection of interoperability standards, known as profiles, for application scenarios ranging from file transfer, to imaging, printing, network access, remote control, and streaming audio and video. Products currently incorporating Bluetooth include cellphones, headsets, automobiles, consumer electronics, printers, cameras, and medical equipment from manufacturers around the world.

Like most emerging technologies, Bluetooth took longer to become mainstream than originally forecasted. But also like most successful emerging technologies, Bluetooth is having far greater impact on the market and reaching into many more products than originally anticipated.

Bluetooth was originally positioned as a neat, high-end feature on cellphones, eliminating the tangle of wires between phones and headsets, phones and hands-free units and phones and PCs. One supposes this usage scenario is to be expected since it was a mobile phone handset company (Ericsson) that invented Bluetooth.

But from the beginning, the visionaries at Ericsson along with IBM, Intel, Microsoft, Motorola, Nokia, Toshiba and others recognized that Bluetooth was a technology that could be applied to many more applications as a wirelessly connection of "everything to everything."

As Bluetooth software experts, Open Interface well knows the barriers to widespread adoption of Bluetooth in devices outside the mobile phone space are complexity and cost – not so much the complexity of adding the Bluetooth radio, since the hardware interfaces are standardized, but the complexity of integrating the required Bluetooth protocol software stacks and in developing Bluetooth applications.

In a mobile phone platform the Bluetooth software development cost can be easily amortized across hundreds of millions of handsets. This is not the case for the lower volumes of consumer electronics devices, audio/video receivers for example, or in vertical markets, like medical device products. Further, the software integration requires extensive expertise in embedded systems and a deep familiarity with Bluetooth specifications. Finally, the product must be tested for interoperability and compliance with the appropriate Bluetooth specifications to obtain Bluetooth Qualification Board (BQB) certification.

Overcoming these barriers requires a new set of powerful software tools that enables rapid integration of Bluetooth into new and existing designs that up to this point haven't been offered to the marketplace. These tools need to eliminate the need for in-depth familiarity with the Bluetooth specification, the mysteries of radio frequency (RF) design as well as provide guarantees for reliability and compliance. These would need to be tools that enable the lay engineer to integrate Bluetooth rapidly and affordably into virtually any device, truly empowering the vision of "everything connected to everything."

Glossary of Acronyms

API – *Application Programming Interface*
AVRCP – *Audio/Video Remote Control Profile*
BHAPI – *BLUEmagic Host API*
BHCLI – *BHAPI Client*
BHSRV – *BHAPI Server*
BIP – *Basic Imaging Profile*
BLST – *BHAPI Lightweight Serial Transport*
BQB – *Bluetooth Qualification Board*
CE – *Consumer Electronics*
GUI – *Graphic User Interface*
HCI – *Host Controller Interface*
HID – *Human Interface Device*
OISP – *Open Interface Service Provision*
OPP – *Object Push Profile*
SDP – *Service Discover Protocol*
UART – *Universal Asynchronous Receive Transmit*

The State of the Market in Software – Bluetooth Protocol Stack and Profiles

Open Interfaces' most popular product to date has been a software stack called BLUEmagic 3.0. It is indicative of the market that BLUEmagic 3.0 is currently integrated into many leading Bluetooth products including over 33 cellphone handsets worldwide.

But feedback from handset manufacturers and other types of customers, bolstered by Open Interfaces' experience handling technical support calls, convinced the company that many customers needed an easier way to integrate Bluetooth technology into their products. To address this need Open Interface defined a Bluetooth Development Platform that Open Interface named BLUEmagic Host Application Programming Interface (BHAPI).

A Bluetooth protocol stack is a collection of layered software components that interface with a Bluetooth radio module. A Bluetooth radio module is one or more chips that implement what is commonly referred to as a Bluetooth baseband. The baseband provides low-level functionality for managing radio communication links. The Bluetooth protocol stack typically communicates with the baseband over a standard interface named the Host Controller Interface (*see sidebar*). The protocol stack implements the protocols required for determining the capabilities of other Bluetooth devices and establishing communication links between those devices. Bluetooth profiles define application specific protocols that are layered on top of the Bluetooth protocol stack. The protocol stack and profiles have Application Programming Interfaces (APIs) that are used to develop Bluetooth applications.

Interestingly enough the Bluetooth specification does not define APIs. Instead, the specification defines protocols and the functions that must be available to applications. Companies such as Open Interface who provide protocol stacks are free to define the APIs as they choose.

Previous to this point in time, Open Interface spent thousands of engineering man-hours individually tailoring and defining its APIs specifically for the needs of embedded systems developers. The resulting APIs provide full access to all of the underlying Bluetooth features and are optimized for embedded devices that have limited memory and CPU resources.

Bluetooth Simplified – Creating a True Bluetooth Development Platform

The easiest way to develop Bluetooth products is to use Open Interfaces' Bluetooth Development Platform, BHAPI. It partitions the Bluetooth functionality into a server and a client that provide a clean separation of the Bluetooth and application functions.

The server, which Open Interface refers to as BHAPI Server (BHSRV), encapsulates the Bluetooth protocol stack and profiles. And the client, that Open Interface refers to as BHAPI client (BHCLI), provides a high-level API that greatly simplifies the development of Bluetooth applications.

The BHAPI architecture is designed so that BHCLI and BHSRV can run on separate CPUs, although this is not required. In particular, BHSRV can be integrated directly with a Bluetooth baseband so that the baseband, Bluetooth protocol stack and profiles all run on the same chip. Other attractive features of the Bluetooth Development Platform are:

Bluetooth Host Controller Interface

The Bluetooth Wireless Technology specification defines a standard interface between the Bluetooth baseband (sometimes called a lower-stack) and the Bluetooth protocol stack. This interface known as the Host Controller Interface (HCI) uses a command/event message protocol. Messages sent from the host CPU to the baseband are termed commands; messages from the baseband to the host CPU are termed events. The HCI interface also defines how data packets are sent between the host CPU and the baseband.

Because it implements the HCI, BLUEmagic® 3.0, is designed to work with Bluetooth baseband modules from any vendor. The HCI standard also ensures that manufacturers are able to source interchangeable modules from multiple chip vendors.

The BHAPI protocol is also supported on multiple Bluetooth baseband chips. The standard transport layer for the BHAPI protocol is a three-wire Universal Asynchronous Receiver/Transmitter (UART) because that interface is supported on all the Bluetooth modules we currently work with. Other transports such as I2C, USB, or synchronous serial interface can also be supported.

- The communication protocol between BHCLI and BHSRV is a command/event protocol that can be compared to the HCI command/event protocol defined by the Bluetooth specification. The significant difference is that the BHAPI protocol operates at the level of Bluetooth profiles, whereas the HCI is a device driver type interface.
- C and Java programming versions of the BHCLI API
- APIs for managing an integrated device and service database and other “housekeeping” functions. These are functions that most application developers need and would otherwise have to implement themselves.
- There are at least half a dozen Bluetooth chip manufacturers that produce Bluetooth baseband processors that are capable of running BHSRV. Baseband processors from ST Microelectronics, RF Micro Devices, Renesas and Flextronics currently implement BHSRV. Since these devices all implement the same BHAPI protocol they are interchangeable.

A key motivator for the design of BHAPI was to enable Bluetooth to be integrated into existing products without major hardware redesigns. In such cases the product development engineers may need to bypass the BHCLI API and code directly to the BHAPI protocol. For these engineers Open Interface fully documents the BHAPI protocol and provides sample code for a very lightweight BHAPI client.

Small applications that do real work can be implemented in only a few hundred lines of code. For example, support for the Bluetooth Audio Video Remote Control Profile (AVRCP) could be added to a consumer electronics device with minor modifications to the control firmware and the incorporation of a Bluetooth module running BHSRV.

A Bluetooth Development Platform Satisfies an Urgent Need in the Marketplace

The lack of a commercially available Bluetooth Development Platform currently on the market that makes product development easier by a scale of magnitude for product applications developers and engineering generalists, lead Open Interface to develop the BHAPI Platform.

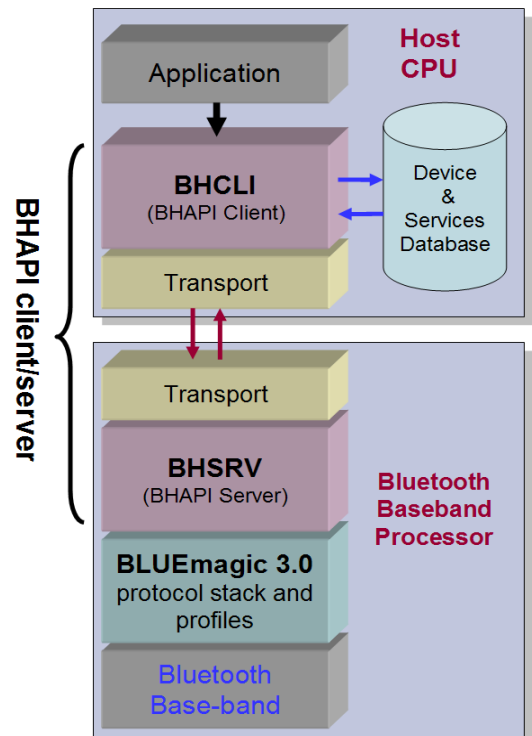
This type of Platform makes life significantly easier for developers of new products to incorporate Bluetooth into new product designs.

The current alternative is to integrate a Bluetooth protocol stack into the software design. A complete Bluetooth protocol stack with profiles, around 100,000 lines of code, will require somewhere between 64- to 128-kilobytes of Flash and between 4- to 32-kilobytes of RAM depending on the specific profiles and application scenarios. The engineering effort to integrate a Bluetooth protocol stack is significant and requires a high level of familiarity with Bluetooth specifications as well as some experience in the nuances of RF engineering. In low volume applications the engineering costs alone can easily exceed the cost of the Bluetooth hardware.

By contrast, the complete Bluetooth Development Platform client is much thinner, approximately 10,000 lines of code that includes APIs for all the profiles, and support for a device and service database. This smaller code base and added functionality greatly simplifies application level integration.

Another key advantage to the product developer is that a Bluetooth Development Platform server implemented directly on a Bluetooth module can be treated as an engineered black box – plug them in and they work immediately.

Interoperability testing as well as qualified protocol stack and profile implementations will have already been performed on that module eliminates a major testing burden. The product developer



is also insulated from hardware and software changes to the module, even from updates to the Bluetooth specifications. With a Development Platform server supported on multiple vendors' Bluetooth modules, the product developer is also not tied to one hardware supplier.

What does BHAPI do for CE Engineers?

A BHSRV module is an integrated component that includes a Bluetooth radio, baseband, protocol stack and profiles, BHSRV and support for one or more transports. The standard transport interface between a BHCLI and a BHSRV module is a Three-wire UART interface. The BHAPI Platform protocol has no real-time performance requirements so the UART interface can be configured to operate at anywhere from 9600bps to 1Mbps. For Bluetooth profiles like file transfer or basic imaging, higher data rates are required to achieve maximum throughput, while other profiles can run well with a low speed link. Streaming audio applications may impose some real-time constraints of their own.

For the product engineer tasked with quickly adding Bluetooth capabilities to an existing product design, and since the BHSRV module is a black box that requires no expertise in Bluetooth radios, protocols or profiles, the only downside is that black boxes can be difficult to debug. With the BHAPI Platform many applications can be prototyped and debugged on a Windows or Linux PC before being integrated in to the product. This can be done using a BHSRV module or a PC implementation of BHSRV. A PC connected via serial cable to the product development platform can operate as a virtual BHSRV module. On the PC, debug and trace tools are included in the Platform and give the developer full visibility into the BHAPI protocol and Bluetooth protocols using an integrated Bluetooth packet sniffer.

The BHAPI Platform supports all of the currently specified Bluetooth profiles, so the same BHSRV module can be used in many different product applications. This also allows Bluetooth capabilities to be added to a product incrementally without requiring a complete hardware redesign.

As an example, the Audio Video Remote Control Profile (AVRCP) might be integrated into a set-top box to enable remote control via Bluetooth. Later, the Human Interface Device (HID) profile might be added to allow keyboard input from a Bluetooth keyboard. Then, support for Basic Imaging Profile (BIP) might be added to enable the set top box to receive images and display slideshows via Bluetooth. All of this can be accomplished using the BHAPI Platform with only minimal software or firmware changes.

In an example like this adding Bluetooth to a product has until now meant integrating a protocol stack and profiles before even starting on application development. Bringing 100,000 lines of foreign code into a product is daunting, even for companies that specialize in software development. Companies whose core expertise is in consumer electronics, automotive systems, or industrial control usually deal with much smaller bodies of code. These companies do not always have the software engineering skills or budget allocated to individual products available to deal with the complexities of Bluetooth. In medical applications, such as patient monitoring, where the advantage of fewer wires seems obvious, regulatory requirements may make it difficult to incorporate external code such as a Bluetooth protocol stack and profiles into a product.

However The BHAPI Bluetooth Development Platform is the catalyst needed for widespread adoption of Bluetooth in product categories outside of the mobile phones and PCs. A lightweight BHAPI client can be implemented in a few hundred lines of code. In many cases, BHAPI can be integrated with a products' existing firmware and without any serious design modifications. By eliminating the technical barriers, the cost to add Bluetooth becomes essentially the cost of the BHSRV module. The economics of this model works even for very low volume products where the engineering overhead is not amortized across millions of units.

In highly competitive markets such as consumer electronics, the need to keep products fresh and deliver the latest features limits product lifetimes. Aggregate volume for a product category may be high, in the tens of millions, but volume for a given product design may be relatively small. The key to success is rapid design cycles based on standard platforms and components.

BHAPI in Action: Problem/Solution

Example 1 – Medical

Medical patient monitoring device: In this market volumes are typically low relative to other products that utilize Bluetooth technology. There are also regulatory issues that complicate the product development process. A standard BHSRV module that can be taken through the approval process and reused for multiple applications is very attractive to medical product companies.

Profiles that might be incorporated into a medical monitoring device are:

File Transfer Profile	<i>enables download of data files from the device to a PDA</i>
Human Interface Device Host	<i>enables keyboard input of patient information</i>
Basic Printing Profile	<i>enables wireless connection to a printer</i>
Basic Imaging Profile	<i>enables download of graphs or images</i>
Serial Port	<i>enables connection to Bluetooth thermometers, blood pressure cuffs, pulse oximeters etc.</i>

If the monitoring device is for use in the home the Dialup Network Profile might be added to enable data to be sent via a mobile phone modem.

Example 2 – Consumer Electronics

DVD Player: In this application scenario, the initial requirement is to add Bluetooth remote control capabilities and roadmap additional functions for the future. The initial profiles package:

Audio/Video Remote Control	<i>enables wireless Bluetooth remote control functions.</i>
Human Interface Device Host	<i>enables mouse and keyboard input</i>

The desire is to add these functions with minimum impact to the existing design through firmware changes to the DVD's microcontroller. In this case, the BTLS serial transport is used and the application is modified to send only the required BHAPI commands to enable the two supported profiles.

Support will be added to a DVD player to upload images from the Bluetooth device into a memory device and display a slide show, or use BIP to control a slide show from a remote device such as mobile phone.

In this case, a future version of BHAPI for DVD will be added with support for:

Basic Imaging Profile	<i>enables image upload and slide shows</i>
Advanced Audio Distribution Profile	<i>enables wireless surround speakers</i>

Example 3 – Convergence Products

Smart-phones: Smart-phones merge the mobile phone, MP3 player, PDA, and digital camera into a single device. BHAPI provides all of the Bluetooth features needed for wirelessly enabling all of these capabilities. In this application scenario the key profiles provided by BHAPI:

Headset Audio Gateway	<i>enables use of Bluetooth headset</i>
Handsfree Audio Gateway	<i>enables use with an automotive handsfree kit</i>
Object Push Profile	<i>enables business card exchange</i>
File Transfer Profile	<i>enables music and image file transfer</i>
Dialup Networking Profile	<i>enables laptop/PDA to use phone as a modem</i>
Serial Port Profile	<i>enables all for "other" applications</i>
Basic Imaging Profile	<i>enables image capture, upload, download, print</i>
Advance Audio Distribution Profile	<i>enables streaming to stereo headset</i>

The Smart-phone has a capable host processor and via Linux or other powerful Operating System standards would be capable of hosting a full BHAPI C or Java client. The BHAPI server could run directly on the Start-phone CPU or could run embedded on a Bluetooth baseband processor.

Conclusion: BHAPI's Innovation

Open Interface, in its new Bluetooth Development Platform, BHAPI, defines a high-level protocol that sits above, and is integrated with, a certified Bluetooth software stack and profiles. Used as a component, BHAPI inherits the Bluetooth qualification status of the underlying the protocol stack and in many cases will not require any further certification from the BQB.

The high level BHAPI protocol provides an abstraction layer on top of the Bluetooth stack and profiles and permits the application and protocol stack to run in separate memory spaces or on separate CPUs.

The abstraction layer simplifies the application interface to the Bluetooth profiles by defining consistent representations of devices, services and connections. Where practical, operations that are similar are unified, for instance, many profiles share the BHAPI "write" command even though the actual implementations of the write function are very profile-specific. Generic "start" and "stop" commands register and de-register profiles so that resources are only allocated when needed.

Running the application and protocol stack in separate memory spaces improves robustness and reliability while helping to ensure adherence to the Bluetooth specification. Running on separate CPUs allows the execution of the Bluetooth profile execution to be offloaded, reducing resource requirements on the host CPU. This advantage is most prominent when the BHAPI server is integrated directly onto a Bluetooth baseband processor.

In this scenario, commands originate at the application and events originate from the protocol stack and profiles. For this reason, the application, the sender of commands and recipient of events, is considered to be the "client," and the protocol stack, the recipient of commands and sender of events, the "server."

- The BHAPI server responds to all commands with an acknowledgment, some data or an error status.
- The BHAPI server also generates asynchronous events such as connect requests or PIN code requests.
- For streaming data, the BHAPI protocol uses a credit-based flow control method. This is mapped by BHSRV to whatever flow control mechanism is provided by the profile.
- Commands and events carry a "connection handle," which identifies a virtual connection between a local and remote Bluetooth service.

This provides a clean unification of all the different types of Bluetooth profile connections. The application can use the connection handle to de-multiplex command responses, events, and data associated with different profiles. BHAPI allows multiple simultaneous connections using different profiles to different Bluetooth devices. The only constraints are the resources available to the specific BHAPI server implementation.

BHAPI is easy to use but still provides access to all the important Bluetooth functions. BHAPI has commands for putting Bluetooth links into low power modes, enabling and disabling device discoverability and connectivity, as well as fully supporting all three Bluetooth security modes, and supporting name string localization.

For documentation purposes, the BHAPI protocol is described in XML format. The same format is generated in trace logs during testing and debugging. The actual protocol is a compact binary representation based on the "data element" format defined in the Bluetooth specification of the Service Discover Protocol (SDP).

- When the BHAPI server is running on a Bluetooth baseband processor, the BHAPI protocol runs over a serial transport, currently on a UART hardware interface.
- The BHAPI protocol assumes it has a reliable transport; Open Interface currently has two implementations, the Open Interface Serial Protocol (OISP) optimized for performance, and the BHAPI Lightweight Serial Transport (BLST) that trades off small code size for lower data throughput.

- Both of these are three-wire serial protocols designed specifically for BHAPI that provide a reliable transport and manage their own flow control.
- When the BHAPI server and client are both hosted on the same processor the BHAPI protocol uses an interprocess communications (IPC) mechanism provided by the underlying operating system. For example, on Windows or Linux BHAPI uses “pipes”.

Although the emphasis with BHAPI has been on providing a high level Bluetooth Development Platform, the devil is in the details as Open Interface provides C and Java APIs that enable rapid application development on many capable host platforms. In many cases, functions in the BHAPI client API are thin layers on top of the BHAPI protocol, but the API includes support for common housekeeping functions such as device and service discovery and management of a discovery database.

The BHAPI client API is a synchronous API typically used in a multi-threaded application environment. The API supports C and Java application development.

With a Bluetooth Development Platform now newly available to product designers and CE engineers, Bluetooth is becoming a mainstream technology ready and waiting for active development in a wide range of vertical applications as well as mass market CE devices.

Bluetooth chips are affordable, reliable and readily available from multiple suppliers. Unfortunately the cost and complexity of Bluetooth application development to this point has prevented the true potential of Bluetooth from being realized outside of the single, well-funded handset market. Open Interface's BHAPI Bluetooth development platform makes Bluetooth application development available to the masses.

And as they say, if you build it, they will come...

About the Author

Greg Burns is Chief Technology Officer at Open Interface, the principle architect of BLUEmagic 3.0 and BHAPI and brings more than 25 years of software development experience to the company. Greg was previously group program manager at Microsoft, during which time he defined mobility features for Windows XP, prototyped applications for Windows-based embedded devices, drove key networking initiatives in Windows 98 and Windows 2000 and was an early pioneer in interactive TV, home networking and advanced consumer technology. Greg can be reached at greg.burns@openinterface.com.

Product Overview

BHAPI Performance Features/Benefits:

- Packages a full suite of Bluetooth profiles into a single easily integrated component. Lowers hardware cost as components can be reused in multiple designs even if different profiles are required.
- Enables offload of Bluetooth profiles processing onto separate CPU. Reduces time to market and cost of software integration since it makes better use of the resources available on the Bluetooth baseband processor.
- Protects against badly behaving devices and malicious attacks. Improves reliability by running protocol stack in separate memory space.
- Close adherence to the Bluetooth specification. Enables product interoperability by providing more complete profile implementations.

BHAPI Developer Benefits:

- Ready-to-use, plug-in solution simplifies development and ensures quicker time to market. No need to port or maintain a Bluetooth protocol stack.
- Bluetooth operation shifts weight of code from application host to system on chip solution. Very little code required for simple applications, only slightly more for more involved applications.
- Flexible development environment enables use of multiple Bluetooth profiles simultaneously.
- Allows for incremental Bluetooth adoption with minimal cost to CE manufacturer. New functions can be added with software/firmware changes only.

BHAPI Protocol Example

The example below shows a short transaction where a Bluetooth device registers an Object Push Profile (OPP) server, receives a connect request followed by a request to pull a business card. Finally the remote device disconnects. This is the actual trace output captured by running this scenario. Every command has a response that may include data.

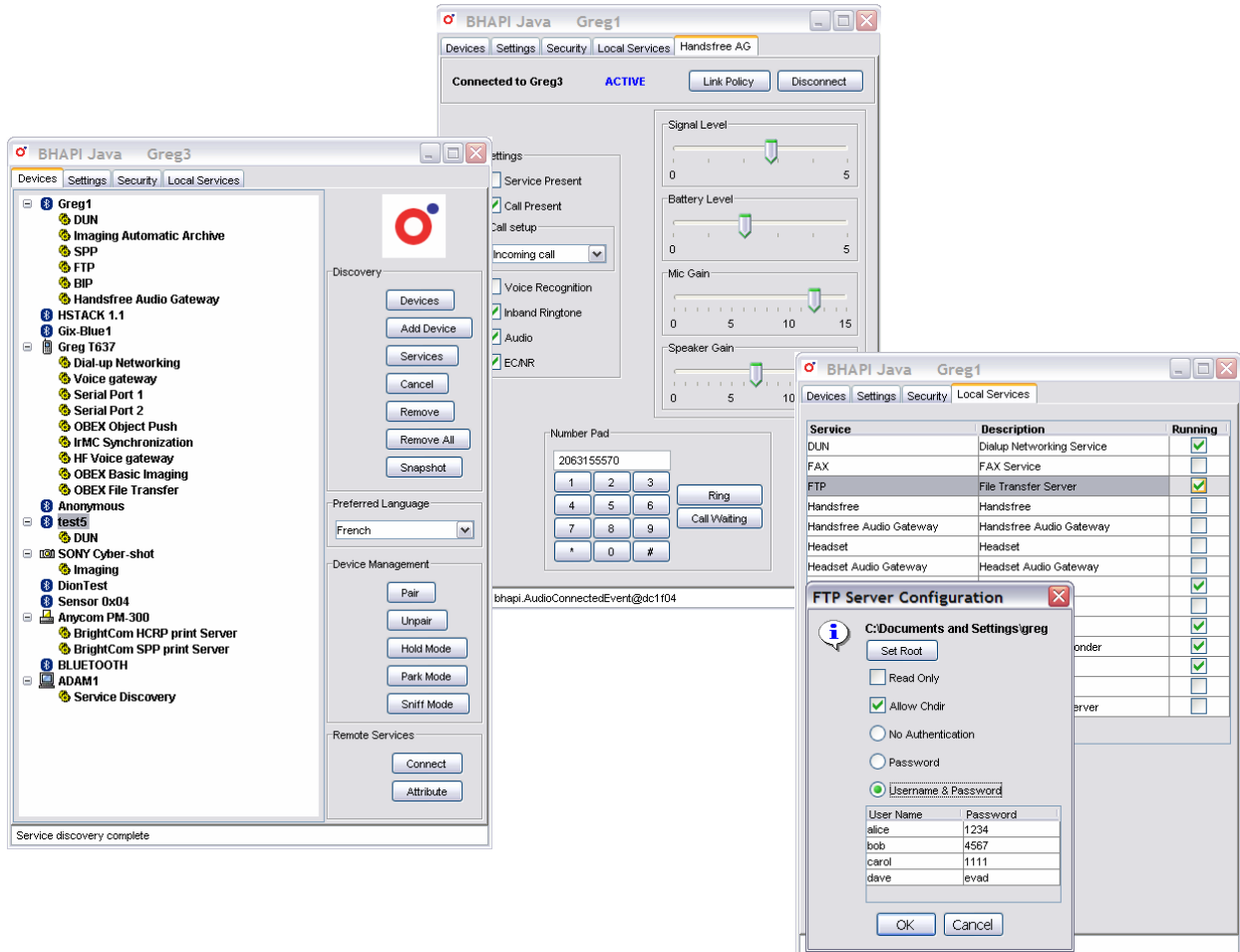
Although not apparent in this simple example, multiple commands for different profiles may be in progress in parallel. The number following the command (0000) or (0001) is the connection handle that represents a service-level connection between two Bluetooth devices, or in the case of 0000, is used for management commands between the BHAPI client and server. That connection handle is then used by the BHAPI client to de-multiplex the responses and events.

<pre>COMMAND (0000): BHAPI_CMD_START_SERVICE <seq> <uuid32 id="OBEXObjectPush">0x00001105</uuid32> <seq> <text>OPP</text> </seq> <seq> <text>Object Push Server</text> </seq> <seq> <uint32>0x00000001</uint32> </seq> </seq></pre>	<p>The start service command tells the BHSRV to register and start the Object Push Profile Server.</p>
<pre>RESPONSE (0000): BHAPI_CMD_START_SERVICE <seq> <uint32>0x00000001</uint32> </seq></pre>	<p>BHSRV responds by returning the service handle (0x00000001) for the started OPP service. The OPP server can now be discovered by other Bluetooth devices.</p>
<pre>EVENT (0000): BHAPI_EVENT_CONNECT_INDICATION <seq> <uint16>0x0001</uint16> <uuid16 id="OBEXObjectPush">0x1105</uuid16> <bdaddr>00:08:1B:02:97:93</bdaddr> <uint32>0x00000001</uint32> <null/> </seq></pre>	<p>A connect indication event informs the BHAPI client that a remote Bluetooth device wants to connect to the OPP server. The connection indication includes a connection handle (0x0001)</p>
<pre>COMMAND (0001): BHAPI_CMD_ACCEPT <seq> <true/> <null/> </seq></pre>	<p>The BHAPI client application accepts or reject the connect request.</p>
<pre>RESPONSE (0001): BHAPI_CMD_ACCEPT <null/></pre>	
<pre>EVENT (0001): BHAPI_EVENT_OPP_SERVER_PULL <null/></pre>	<p>On receiving a business card pull request from the remote device BHSRV generates an OPP pull event.</p>
<pre>COMMAND (0001): BHAPI_CMD_OPP_SERVER_ACCEPT_PULL <seq> <text16> 006F 0069 006E 0061 002E 0076 0063 0066 0000 </text16> <text>text/x-vCard</text> <uint32>0x0000019A</uint32> </seq></pre>	<p>The BHAPI client application accepts the pull request and returns basic information about the business card.</p>
<pre>RESPONSE (0001): BHAPI_CMD_OPP_SERVER_ACCEPT_PULL <null/></pre>	
<pre>COMMAND (0001): BHAPI_CMD_WRITE <rawbytes> 42 45 47 49 4E 3A 56 43 41 52 44 0D 0A 56 45 52 53 49 4F 4E ... </rawbytes></pre>	<p>The client issues one of more write commands to send the business card data.</p>
<pre>RESPONSE (0001): BHAPI_CMD_WRITE <null/></pre>	
<pre>COMMAND (0001): BHAPI_CMD_WRITE <seq> <true/> </seq></pre>	<p>The final write command indicates if the write was completed successfully</p>
<pre>RESPONSE (0001): BHAPI_CMD_WRITE <null/></pre>	
<pre>EVENT (0001): BHAPI_EVENT_DISCONNECTION <seq> <status id="OI_OK">0</status> </seq></pre>	<p>The remote device terminates the connection which causes BHSRV to issue a disconnection event.</p>

BHAPI Java GUI Screenshots

Open Interface uses the BHAPI Java API to implement a graphical user interface in the below example. This is mainly used for testing and evaluation but can also be used as a basis for developing or prototyping Java applications based on the BHAPI Java API.

The left screen shot shows the device and service discovery panel. This is a graphical representation of the device and service cache that is maintained by the BHAPI client. The right screen shot shows the local service panel. This panel is used to start, stop and configure the services on the BHAPI server. The center screen shot is a control pane for the headset audio gateway end of the headset profile. This is purely a test tool; a real audio gateway application would not have such an elaborate GUI.



BHAPI Client Code Example

Below is a minimal code fragment for C code that implements the business card pull operation described as XML above.

The commands are all prepackaged constant data blocks. Any command response may return an error status but for clarity we chose not to do error checking in this sample. The sample also omits the commands required to initialize the BHAPI server and make the Bluetooth device discoverable and connectable.

Finally, and most importantly, this sample does not use the high-level API.

```
static char cmdStartService[] = {
    0x00, 0x00, 0x80, 0x03, 0x35, 0x27, 0x19, 0x11, 0x05, 0x35, 0x05, 0x25, 0x03, 0x4F, 0x50, 0x50,
    0x35, 0x14, 0x25, 0x12, 0x4F, 0x62, 0x6A, 0x65, 0x63, 0x74, 0x20, 0x50, 0x75, 0x73, 0x68, 0x20,
    0x53, 0x65, 0x72, 0x76, 0x65, 0x72, 0x35, 0x05, 0x0A, 0x00, 0x00, 0x00, 0x01 };
static char cmdAccept[] = {
    0x00, 0x01, 0x80, 0x07, 0x35, 0x03, 0x28, 0x01, 0x00 };
static char cmdAcceptPull[] = {
    0x00, 0x01, 0x87, 0x03, 0x35, 0x28, 0xA5, 0x12, 0x00, 0x6F, 0x00, 0x69, 0x00, 0x6E, 0x00, 0x61,
    0x00, 0x2E, 0x00, 0x76, 0x00, 0x63, 0x00, 0x66, 0x00, 0x00, 0x25, 0x0D, 0x74, 0x65, 0x78, 0x74,
    0x2F, 0x78, 0x2D, 0x43, 0x61, 0x72, 0x64, 0x00, 0x0A, 0x00, 0x00, 0x01, 0xA3 };
static char cmdWrite[] =
    "\000\000\200\014\226\001\243"
    "BEGIN:VCARD\r\n"
    "VERSION:2.1\r\n"
    "N:Interface;Open\r\n"
    "FN:Open Interface\r\n"
    "ORG:Open Interface North America, Inc.\r\n"
    "TEL;WORK;VOICE:+1 (206) 315-5570\r\n"
    "TEL;WORK;FAX:+1 (206) 315-5580\r\n"
    "ADR;WORK;;;506 2nd Avenue, Suite 420;Seattle;WA;98104;USA\r\n"
    "LABEL;WORK;ENCODING=QUOTED-PRINTABLE:506 2nd Avenue, Suite 420=0D=0ASeattle, WA 98104=0D=0AUSA\r\n"
    "URL;WORK:http://www.openinterface.com\r\n"
    "EMAIL;PREF;INTERNET:info@openinterface.com\r\n"
    "END:VCARD\r\n";
static char cmdWriteFinal[] = {
    0x00, 0x01, 0x80, 0x0C, 0x35, 0x02, 0x28, 0x01 };

static short connection = 0;

void SendConnectionCmd(char *cmd, short cmdSize)
{
    cmd[0] = (connection & 0xff00) >> 8; // Set the connection handle in the command
    cmd[1] = (connection & 0x00ff);
    OI_MSGIO_SendCmd(cmd, cmdSize);
}

void EventInd(char *event, short eventSize)
{
    static int writeComplete = FALSE;
    short opcode = (event[2] << 8) | event[3]; // Event or command response
    switch (opcode) {
        case BHAPI_EVENT_CONNECT_INDICATION:
            if (connection == 0) {
                connection = (event[7] << 8) | event[8];
                SendConnectionCmd(cmdAccept, sizeof(cmdAccept));
            }
            break;
        case BHAPI_EVENT_OPP_SERVER_PULL:
            SendConnectionCmd(cmdAcceptPull, sizeof(cmdAcceptPull));
            break;
        case BHAPI_CMD_OPP_SERVER_ACCEPT_PULL: // response to accept pull command
            writeComplete = FALSE;
            SendConnectionCmd(cmdWrite, sizeof(cmdWrite));
            break;
        case BHAPI_CMD_WRITE: // response to write command
            if (!writeComplete) {
                SendConnectionCmd(cmdWriteFinal, sizeof(cmdWriteFinal));
                writeComplete = TRUE;
            }
            break;
        case BHAPI_EVENT_DISCONNECTION:
            connection = 0;
            break;
    }
}
```