User Manual

BCM1000-BTW

Bluetooth® Software

BTW SDK Sample Applications

1000-BTW-UM205-R

5300 California Avenue • P.O. Box 57013 • Irvine, California 92617 • Phone: 949-450-8700 • Fax: 949-926-5203

06/13/07

REVISION HISTORY

Revision	*Date*	*Change Description*

1000-BTW-UM205-D2	06/13/07	**Updated:**
		• "Overview" on page 1
		• "Using the DK Classes in an Application" on page 1
		• BluePrint—A CPrintClient Application:
		  • "BluePrint Functionality" on page 31
		  • "Printing" on page 32
		• Figure 8: "BlueHeadphone - Primary Class Relationships," on page 35 (graphic modification only, no technical modification)

1000-BTW-UM204-R	12/05/06	**Updated:** • "Using the DK Classes in an Application" on page 1. • BlueTime—An L2CAP Time Monitor. "Building the Application" on page 3. • BlueChat—An RFCOMM Chat Application. "Building the Application" on page 9. • BlueClient—An FTP OPP Application. "Building the Application" on page 15. • BlueComChat—A Com Port Chat Application. "Building the Application" on page 19. • BlueObex—An OBEX Exerciser. "Building the Application" on page 22. • BlueAudio—An Audio Sample Application. "Building the Application" on page 27. • BluePrint—A CPrintClient Application. "Building the Application" on page 32. • BlueHeadphone - A CHeadphoneClient Application. "Building the Application" on page 34.
1000-BTW-UM203-R	08/04/06	**Updated:** • "Using the DK Classes in an Application" on page 1. • "OBEX Connection Function" on page 20
1000-BTW-UM202-R	09/12/05	**Modified** "BlueComChat—a Com Port Chat Application" on page 15 to reflect UI enhancements.

Broadcom Corporation
P.O. Box 57013
5300 California Avenue
Irvine, California 92617

Revision	*Date*	*Change Description*
1000-BTW-UM201-R	09/02/05	**Modified** “BlueAudio—an Audio Sample Application” on page 23: updated the screenshots and elaborated on the procedures to create and use a monaural audio connection. **Added** “BlueHeadphone - a CHeadphoneClient Application” on page 30: BlueHeadphone is a sample UI that establishes a stereo audio connection using the Advanced Audio Distribution Profile (A2DP).
1000-BTW-UM200-R	10/14/04	Initial release

TABLE OF CONTENTS

LIST OF FIGURES

OVERVIEW

This document describes applications that can be built using the WIDCOMM® BTW Software Development Kit.

All of the sample applications provided with the DK can be deployed against the latest version of Broadcom's Bluetooth® Software for Windows® stack software (BTW 5).

Most of the sample applications run against Broadcom's Vista Profile Pack Bluetooth software for the Microsoft® Bluetooth stack (BTW 6). See the DK Programmer's Guide and Release Notes for more information on which DK classes and demonstration sample applications are supported in the latest Profile Pack software version.

USING THE DK CLASSES IN AN APPLICATION

Sample applications are provided as Microsoft Visual C++ 6.0 projects (.dsp files) and Microsoft Visual Studio 2005 projects (.vcproj files).

Support for Windows platforms is provided in 32 bit and AMD64 development environments. For Visual Studio 2005, use the x64 Platform configuration to build the 64 bit samples. Use the Release AMD64 and Debug AMD64 project configurations under Visual C++ 6.0 to build the samples in an AMD64 bit development environment.

Note: To use Visual C++ 6.0 to build AMD64 applications requires the Microsoft October 2003 update to the February 2003 Platform SDK. When building the Release AMD64 or Debug AMD64 targets, use the Windows Server 2003 AMD64 Build Environment window to launch Visual C++ 6.0 and to open the samples.

Note: Refer to the SDK Programmer's Guide for information on redistribution requirements (btwapi.dll and redistributable Microsoft library modules) when deploying VS2005 built applications.

Two Bluetooth-enabled PCs are required to run the sample applications.

The sample applications are built using MFC dialog classes. The primary application class is based on the CDialog class.

In addition, the CRfCommPort, CL2CapConn, CFtpClient, COppClient, CLapClient, CSppClient, CSppServer, CObexClient, and CObexServer DK classes are base classes containing virtual methods provided by the applications. These virtual methods serve as event handlers for the application, managing events such as file transfer complete and progress.

Caution! Virtual functions are called on an independent thread. Therefore, operations performed within the virtual functions must be made thread safe.

Sample application dialog classes (CBlueChatDlg, CBlueTimeDlg, and CBlueClientDlg, etc.) use multiple inheritance from CDialog and whichever DK base classes needed for the application. This approach makes it convenient for the application to define derived methods for the DK virtual methods. These derived methods have access to the dialog class services and can easily share data with the main sample dialog object.

OTHER APPROACHES

Other approaches are possible for using the DK classes, such as when an application developer wants to use the DK base class `CRfCommPort`, but does not want the application's main dialog module to be based on `CRfCommPort`. The application may need to support two RFCOMM connections, for example.

In this case, the developer could define a derived class, `CRfCommPortDerived`, which supplies all the virtual methods for using the DK base class `CRFCommPort`. Then the application's main program, or main dialog module, would define two instance members, such as:

`CRfCommPortDerived` connection_a;

`CRfCommPortDerived` connection_b;

The developer would need to supply methods for `CRfCommPortDerived` that allow the main module to exchange data and status with connection_a and connection_b. How this is done depends on the application requirements.

Class `CRfCommPortDerived` could be defined with a public method that returned current status to the caller. This would allow the main module to poll the `CRfCommPortDerived` object, passing commands and receiving status.

If polling by the main module is not desirable, the `CRfCommPortDerived` constructor can be defined to allow the main module to pass a "this" pointer. Then the `CRfCommPortDerived` object can call functions in the main module in an event-driven manner.

BLUETIME—AN L2CAP TIME MONITOR

The BlueTime application creates a client-server L2CAP connection. Every second, each side sends its local time to the other side.

BlueTime is installed on two platforms. When they are executed, one is configured as the client and the other as the server by the user.

Both sides display the current local time and the latest time sent from the other side.

The user establishes the connection; the server creates a new service and then waits for a client to connect. The client performs a device inquiry and service discovery, and then the two sides connect and communicate.

BUILDING THE APPLICATION

To run the BlueTime sample application, complete these steps:

1. Start Microsoft Visual C++.
2. Open the BlueTime.dsp project (VC++ 6.0) or the BlueTime.vcproj project (VS2005) located in the \Sdk\Samples\BlueTime directory.
3. Build the application.
4. Run BlueTime.exe.

BLUETIME FUNCTIONALITY

At startup, the user must choose to run the application as either the server or the client.

Running in Server Mode

To run in server mode, complete these steps:

1. Click the **Act as Time Server** check box.
2. Click **Start Session**.

   BlueTime starts a session and then performs the following tasks:

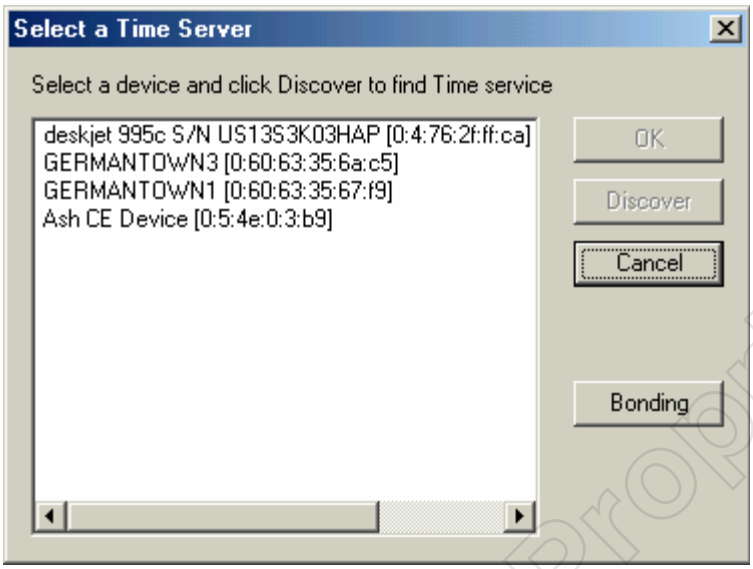
   - Adds a Service Record using SDP.
   - Opens an L2CAP server connection.
   - Waits for a client connection.

Running in Client Mode

To run in client mode, complete these steps:

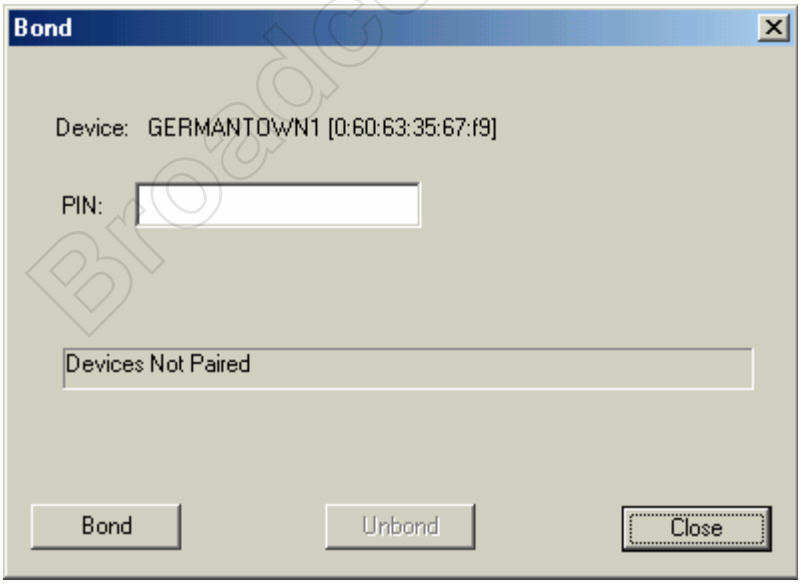
1. Click **Start Session**.

   The Select a Time Server dialog box appears allowing you to choose the Time service you want to use. The server list is populated automatically with the Bluetooth devices found using device inquiry.



- To select a device to perform a bonding operation, click **Bonding**.

  The Bond dialog box appears informing the user if the selected device is currently paired with the local device.



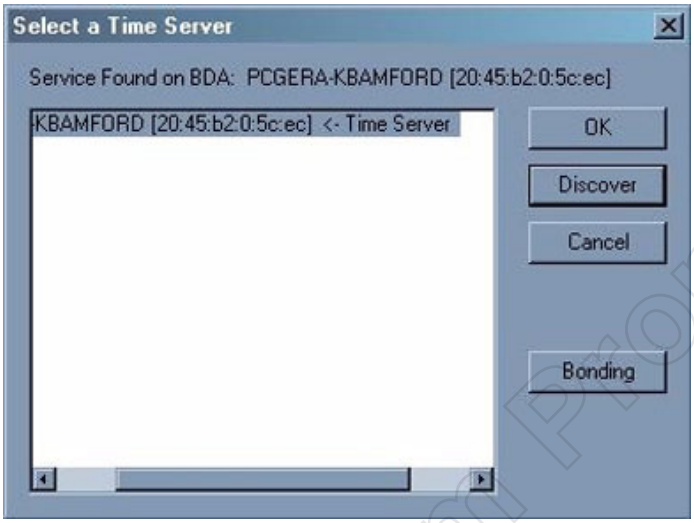
- To bond the devices (optional), enter a PIN code, and then press **Bond** to initiate the pairing process.

Note: If the devices are currently paired, clicking **Unbond** severs the bond.

The BlueTime application waits until bonding is resolved by approval from the selected server, rejection by the server, or a timeout.

The result is reported in the bonding dialog window.

In the Select A Time Server dialog box, discovered devices offering BlueTime services are identified by the suffix “Time Server.”



2. Select a host from the list.
3. Click **Discover** to send a discovery request to determine if the host is a Time Server.
4. Click **OK** if the Time Service was found.

The dialog window closes and an L2CAP client connection is established with the Time Server host.

BlueTime is ready to communicate.

Enabling Security

Security is supported in Client and Server mode. To enable security, click **Security**, and then select one or more of the available security options.

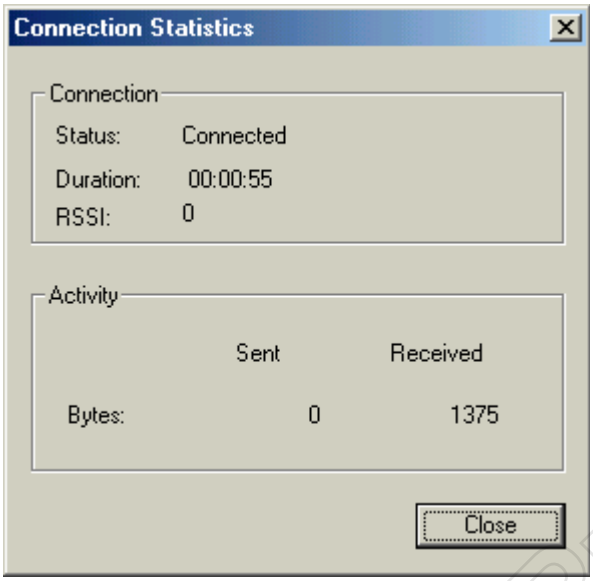
The following security options are available:

- No Security
- Authorize Incoming
- Authenticate Incoming
- Encrypt Incoming
- Authenticate Outgoing
- Encrypt Outgoing

Viewing Connection Statistics

To view current connection statistics, in the BlueTime main window, click **Conn Stats**.

The modeless Connection Statistics window appears, displaying the current connection statistics. The values in the window are updated every second. See *CL2CapConn::GetConnectionStats()* for details.



Exiting BlueTime

To exit the application, click **Close**.

KEY CLASS DESCRIPTIONS

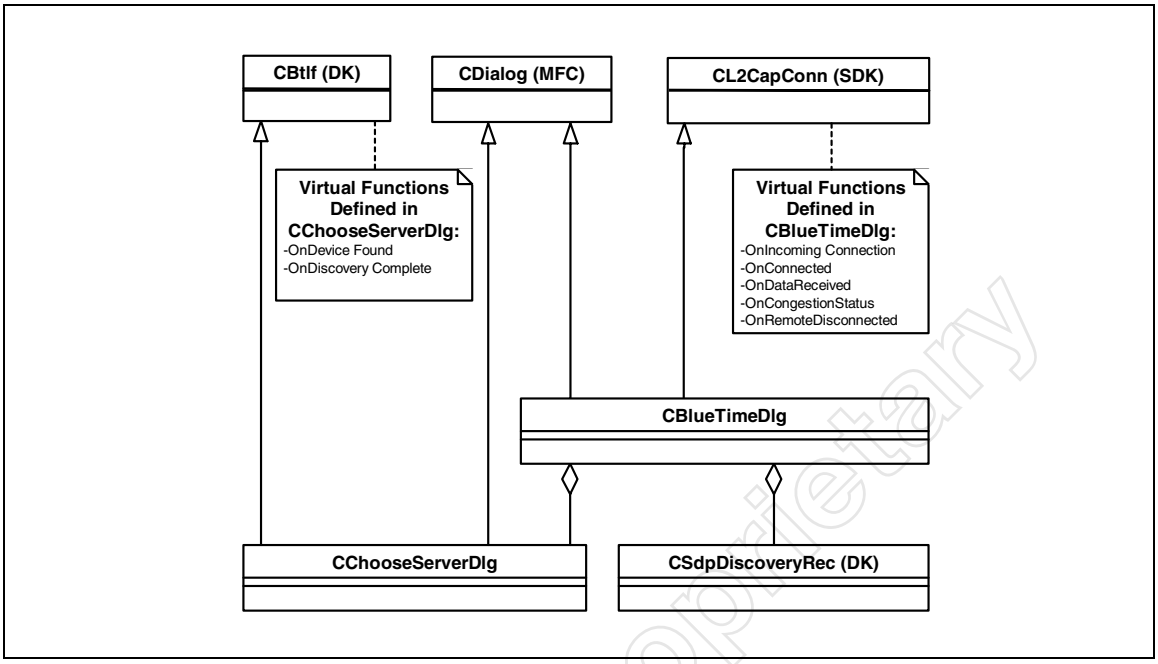


Figure 1: BlueTime—Primary Class Relationships

`CBlueTimeDlg` implements dialog functionality for the main chat window. This module demonstrates the following features of the DK:

- Creating SDP Service records
- Opening a client or server L2CAP connection
- Setting the security level
- Checking the connection
- Writing data to the port
- Closing the connection

This is a typical MFC dialog class, derived on the `CDialog` base class.

For this application this class also inherits from the DK class `CL2CapConn`. The virtual methods from `CL2CapConn` can then be incorporated within `CBlueTimeDlg` for convenient access to common class interfaces.

`CChooseServerDlg` implements the dialog functionality for choosing a Time server. This module demonstrates the following features of the DK:

- Using the BT API class
- Performing a device inquiry
- Performing a service discovery
- Reading discovery records

BLUECHAT—AN RFCOMM CHAT APPLICATION

The BlueChat application creates an RFCOMM connection and permits the client and server to chat. BlueChat is installed on two platforms. When they are executed, one is configured as the client, and the other is configured as the server by the user.

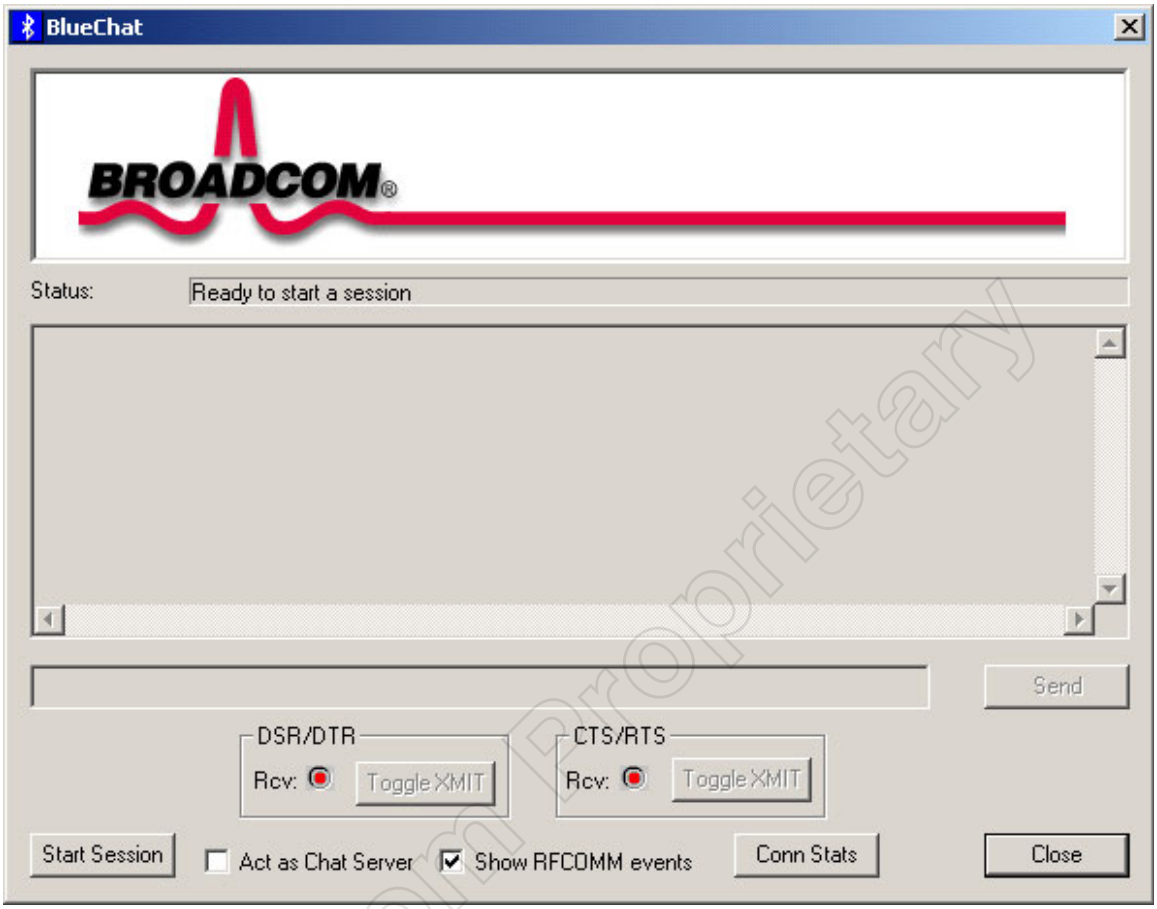
- The GUI allows modem control signals to be set/read and error conditions to be sent to the other side.
- The user establishes the connection (with one PC as the server and the other as the client). The server PC creates a new service, and then waits for the client to connect.
- The client performs a device inquiry and service discovery.
- The two sides connect and communicate.

BUILDING THE APPLICATION

To run the BlueChat sample application, complete these steps:

1. Start Microsoft Visual C++.
2. Open the BlueChat.dsp project (VC++ 6.0) or the BlueChat.vcproj project (VS2005) located in the \Sdk\Samples\BlueChat directory.
3. Build the application.
4. Run BlueChat.exe.

BLUECHAT FUNCTIONALITY



At startup, the user must choose to run the application as either the server or the client.

Running in Server Mode

To run in server mode, complete these steps:

1. Click the **Act as Chat Server** check box.
2. Click **Start Session**.

   BlueChat starts a session, and then performs the following tasks:

   - Adds a service record using DK class CSdpService in function `CBlueChatDlg::DoCreateServiceRecord()`.
   - Opens an RFCOMM server port using `OpenServer()` (a method inherited from DK class `CRfCommPort`).
   - Waits for a client connection.

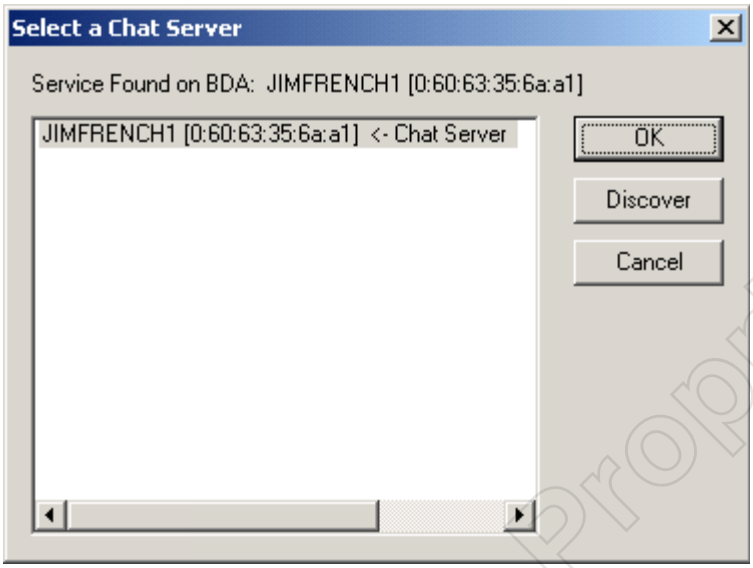
Running in Client Mode

To run in client mode, complete these steps:

1. Click **Start Session**.

   BlueChat starts a client session. The list of servers is populated automatically with the Bluetooth devices found using device inquiry.

2. In the Select a Chat Server dialog box that appears, select a host from the list.



3. Click **Discover** to send a discovery request to determine if the host is a Chat Server.
4. Click **OK**.

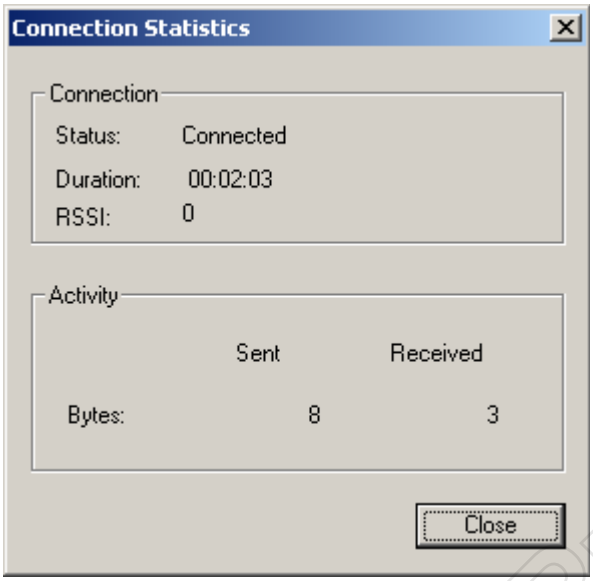
   The dialog window closes and an RFCOMM client connection is opened with the Chat Server. BlueChat is ready to communicate.

5. To send a message, type the message in the edit field, and then click **Send**.

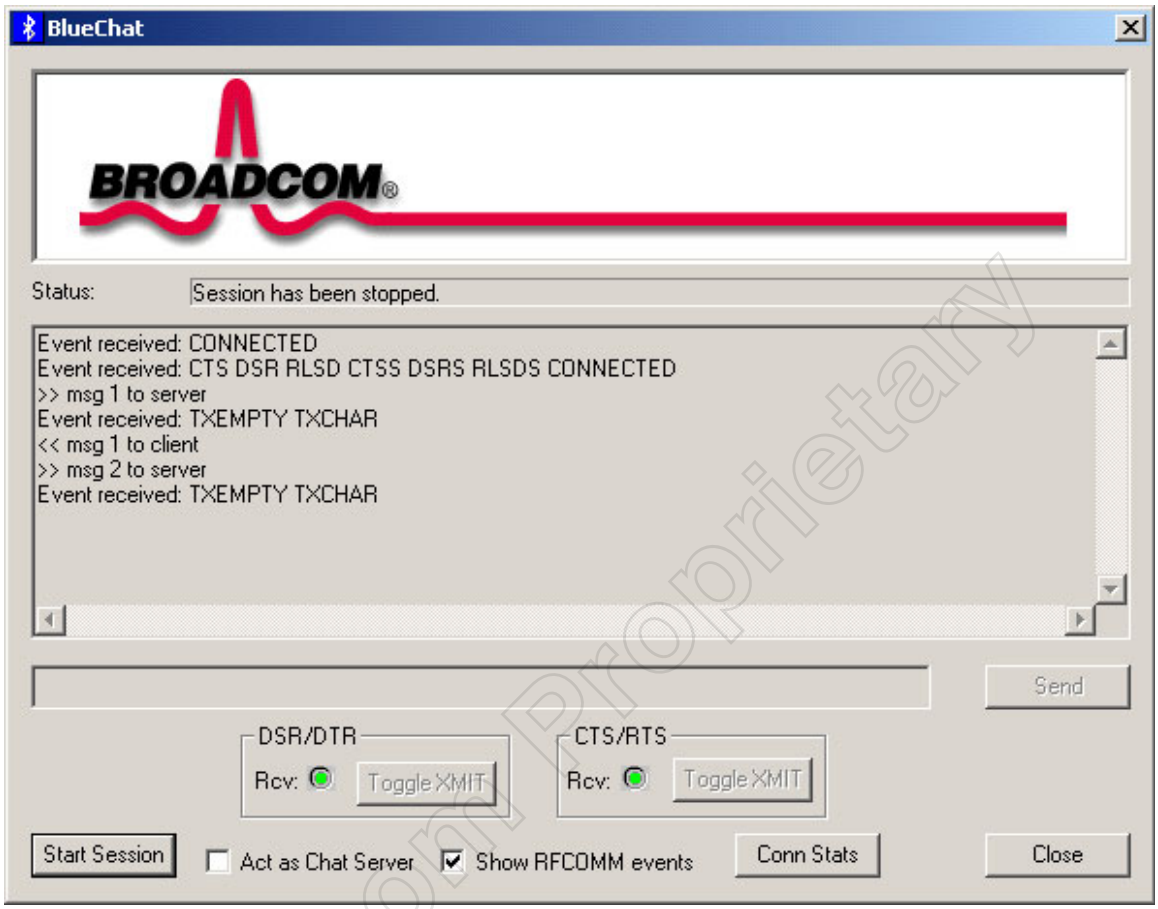
Viewing Connection Statistics

To view current connection statistics, in the BlueChat main window, click **Conn Stats**.

The modeless Connection Statistics window appears, displaying the current connection statistics. The values in the window are updated every second. See `CRfCommPort::GetConnectionStats()` for details.



After a brief chat session with a server, the BlueChat main window displays an event history similar to the history shown in the following screen.



The BlueChat main window displays the following information for Client and Server modes:

- Messages sent and received are echoed in the log window.
- RFCOMM events display in the log window.

Note: To prevent RFCOMM events from being displayed, clear the **Show RFCOMM Events** check box.

- Status lights identify control signals sent by the remote chat application.

Note: The user can transmit a control signal by clicking the appropriate button.

Exiting BlueChat

To exit the application, click **Close**.

KEY CLASS DESCRIPTIONS

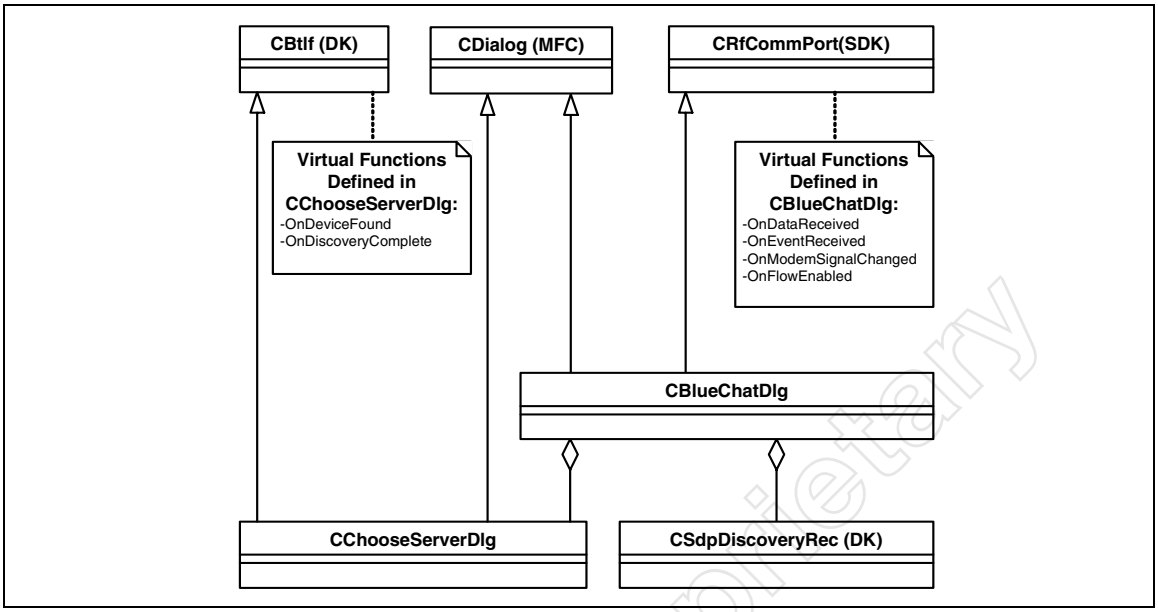


Figure 2: BlueChat—Primary Class Relationships

`CBlueChatDlg` implements the dialog functionality for the main chat window. This module demonstrates the following features of the DK:

- Creating SDP Service records
- Opening a client or server RFCOMM port
- Checking the port connection
- Writing data to the port
- Getting modem status
- Transmitting modem signals
- Closing the RFCOMM port

This is a typical MFC dialog class, derived from the `CDialog` base class.

For this application, this class also inherits from the DK class `CRfCommPort`. The virtual methods from `CRfCommPort` may then be incorporated within `CBlueChatDlg` for convenient access to common class interfaces.

`CChooseServerDlg` implements the dialog functionality for choosing a Chat server. This module demonstrates the following features of the DK:

- Using the BT API class.
- Performing a device inquiry.
- Performing a service discovery.
- Reading discovery records.

BLUECLIENT—AN FTP OPP APPLICATION

The BlueClient application provides client access to the FTP and OPP servers in the Bluetooth neighborhood. Connections are set up as needed, and client functions are under direct control of the user.

BlueClient performs an inquiry at startup to determine the available Bluetooth devices in the neighborhood. The user chooses a server, and then requests discovery of FTP and OPP services, such as file get, file put, object push, etc.

BlueClient has the look and feel of the WIDCOMM BT Neighborhood UI component of Broadcom's BCM1000-BTW. Bluetooth devices, services, folder names, and file names appear in a tree format.

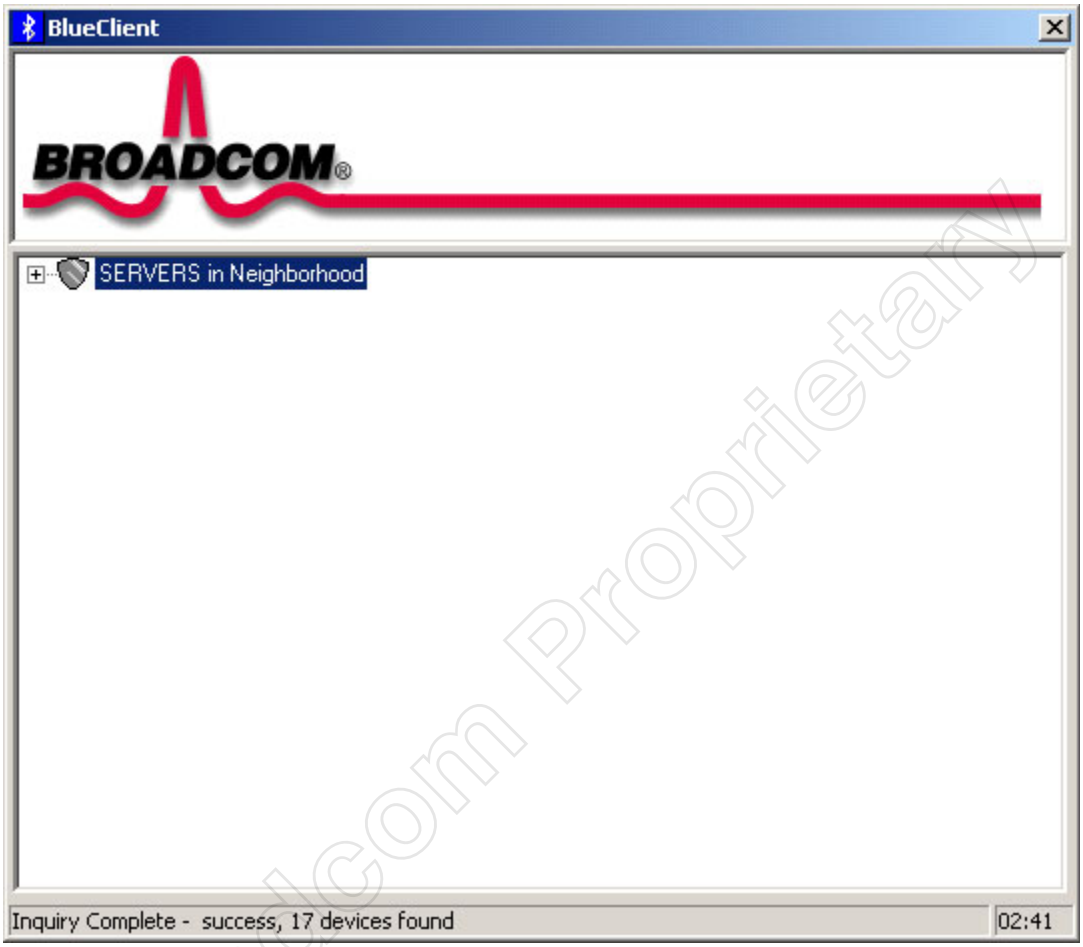
BUILDING THE APPLICATION

To run the BlueClient sample application, complete these steps:

1. Start Microsoft Visual C++.
2. Open the BlueClient.dsp project (VC++ 6.0) or the BlueClient.vcproj project (VS2005) located in the \Sdk\Samples\BlueClient directory.
3. Build the application.
4. Run BlueClient.exe.

BLUECLIENT FUNCTIONALITY

BlueClient only runs as a client application. Another platform is needed to provide the standard profile FTP and OPP services requested by BlueClient.

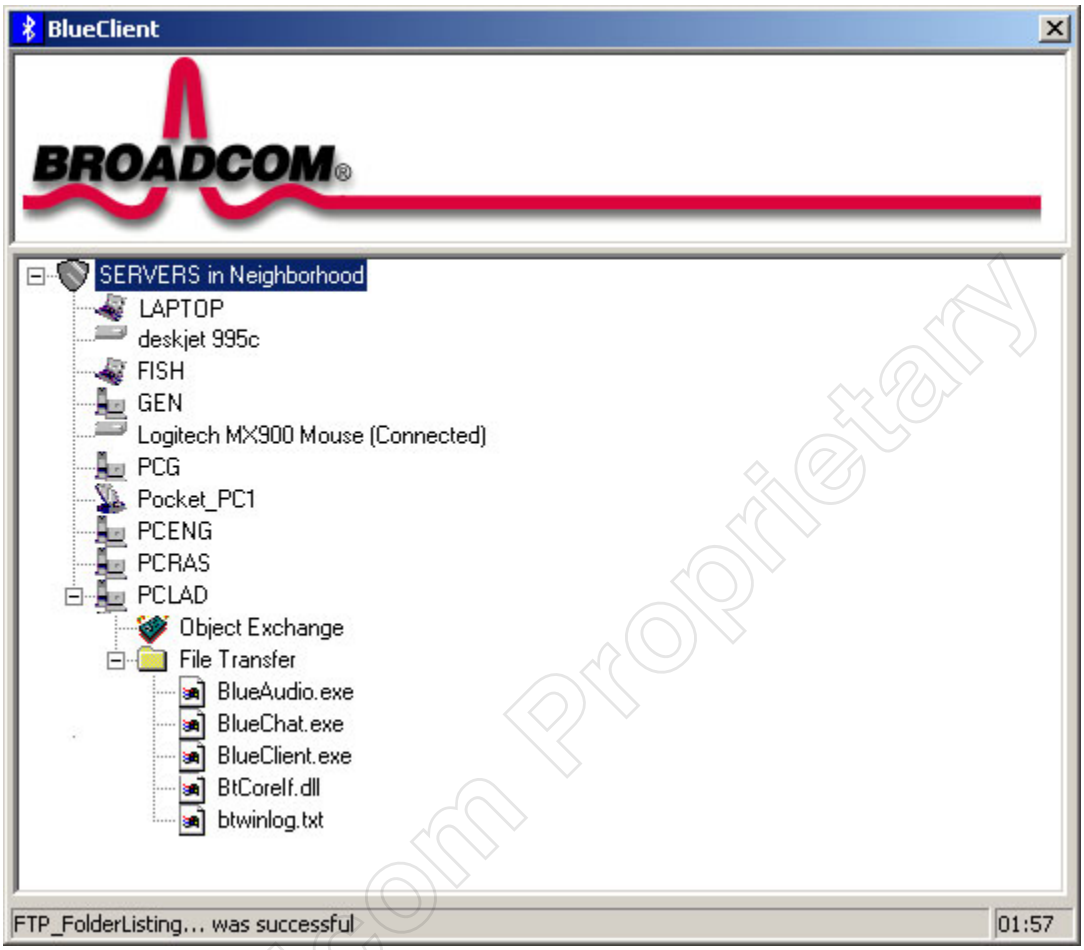


Most actions are taken in the main dialog window. Right-click the **Servers in Neighborhood** entry at the root of the tree to refresh the list of Bluetooth server devices.

Under the root, there is a branch for each server device. Right-click a device to request a discovery for it or to show the device properties.

Services (FTP and OPP) are subbranches of the tree under the device.

Right-click a service to view the available functions or expand the branch to view a sublist of files, for FTP for example. Right-click a specific file to get the file from the server.



The status bar at the bottom of the main window displays the operation in progress and the result. For file transfers, the status bar displays the progress of the transfer as *x* of *y* bytes transferred.

Subsidiary dialog windows are used as necessary. For example, selecting files for transfer, showing server properties, selecting a file to delete, etc.

KEY CLASS DESCRIPTIONS

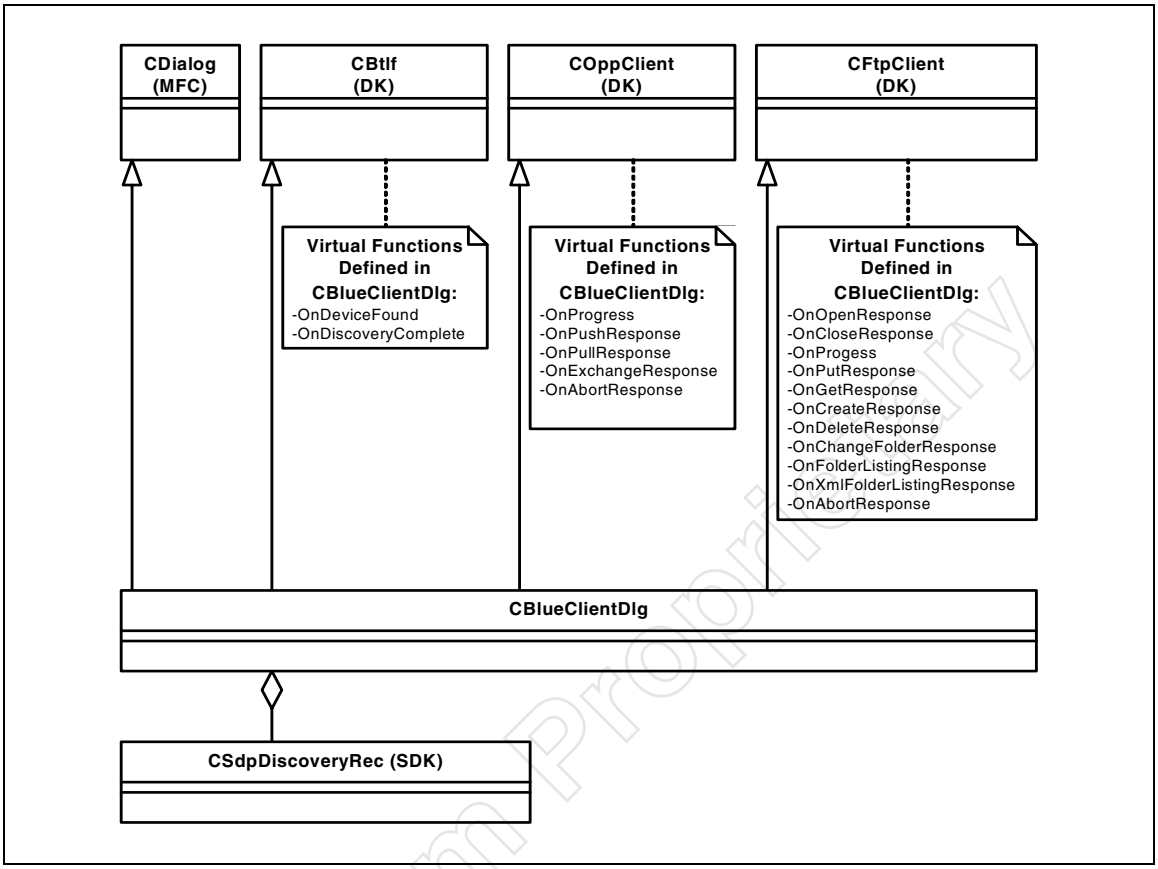


Figure 3: BlueClient—Primary Class Relationships

`CBlueClientDlg` implements the dialog functionality for the main BlueClient window. This module demonstrates the following features of the DK:

- Starting device inquiry and present the available servers in the tree.
- Allowing right-click menus to refresh and expand the device services.
- Allowing right-click menus to perform specific FTP and OPP operations.
- Presenting progress and status for the operations requested.

This is a typical MFC dialog class inheriting from the `CDialog` base class.

For this application, this class also inherits from the DK classes `CFtpClient` and `COppClient`. The virtual methods from these base classes can then be incorporated into `CBlueClientDlg` for convenient access to common class interfaces.

`FTPCreate` implements the dialog functionality to obtain a name for a new file under FTP.

`CBtDeviceProps` displays the properties of a selected server device, such as the user-friendly name, the device's Bluetooth address, and the connection state.

BLUECOMCHAT—A COM PORT CHAT APPLICATION

BlueComChat creates a serial connection over a Windows COM port that has been pre-assigned to a Bluetooth service. The client and server can then chat.

BlueComChat is installed on two platforms. When they are executed, one is configured as the client and the other is configured as the server by the user.

BUILDING THE APPLICATION

To build the BlueComChat sample application, complete these steps:

1. Start Microsoft Visual C++.
2. Open the BlueComChat.dsp project (VC++ 6.0) or the BlueComChat.vcproj project (VS2005) located in the \Sdk\Samples\BlueComChat directory.
3. Build the application.

BLUECOMCHAT FUNCTIONALITY

BlueComChat expects to use a new service, WIDCOMM DK COM Serial Port, which must be manually defined on the server.

Defining the WIDCOMM DK COM Serial Port

On the server, perform the following:

1. From the Bluetooth Configuration panel, Local Services tab, click **Add Serial Service**.
2. In the Service Properties dialog box, enter `WIDCOMM DK COM Serial Port` as the name of the service.
3. Set the following options, as desired:
   - The default COM port assignment can be accepted, or a different one may be entered.
   - Select or clear the **Startup** check box. Clearing (deselecting) the Startup check box prevents the service from automatically starting during system startup. As a result, the server side application controls when the service is started and stopped. Manual intervention is not required after the one-time setup.

Running BlueComChat in Server Mode

To run BlueComChat in Server Mode, complete these steps:

1. Run BlueComChat.
2. Select the **Act as Com Chat Server** check box.

3. Click **Start Session**.

   BlueComChat starts a chat session, and then performs the following tasks:

   - Initiates an SPP server connection for the service name defined in m_serviceNameForServer in the application source file BlueComChatDlg.cpp (the DK is installed with the service name “WIDCOMM DK COM Serial Port”).
   - Waits for a client connection.

Running BlueComChat in Client Mode

To run BlueComChat in Client Mode, complete these steps:

1. Run BlueComChat
2. Click Start Session

   The client performs a device inquiry and then service discovery. Servers that offer the serial port service class (GUID equal to CBtIf::guid_SERVCLASS_SERIAL_PORT) are displayed on the client user interface. For each discovered service, the server device name and the service name are displayed.

3. Select a service to initiate the SPP connection.

   After a connection is established, the text field and the Send button on both the server and the client are enabled and the users can exchange messages.

4. Click **Stop Session** to close the connection.

   The SetSniffMode and CancelSniffMode buttons control the link’s sniff mode.

The ReadLinkMode button reads and displays the current link mode:

```
typedef enum
{
   LINK_MODE_NORMAL,
   LINK_MODE_HOLD,
   LINK_MODE_SNIFF,
   LINK_MODE_PARK
} LINK_MODE;
```

Exiting BlueComChat

To exit the application, click **Close**.

KEY CLASS DESCRIPTIONS

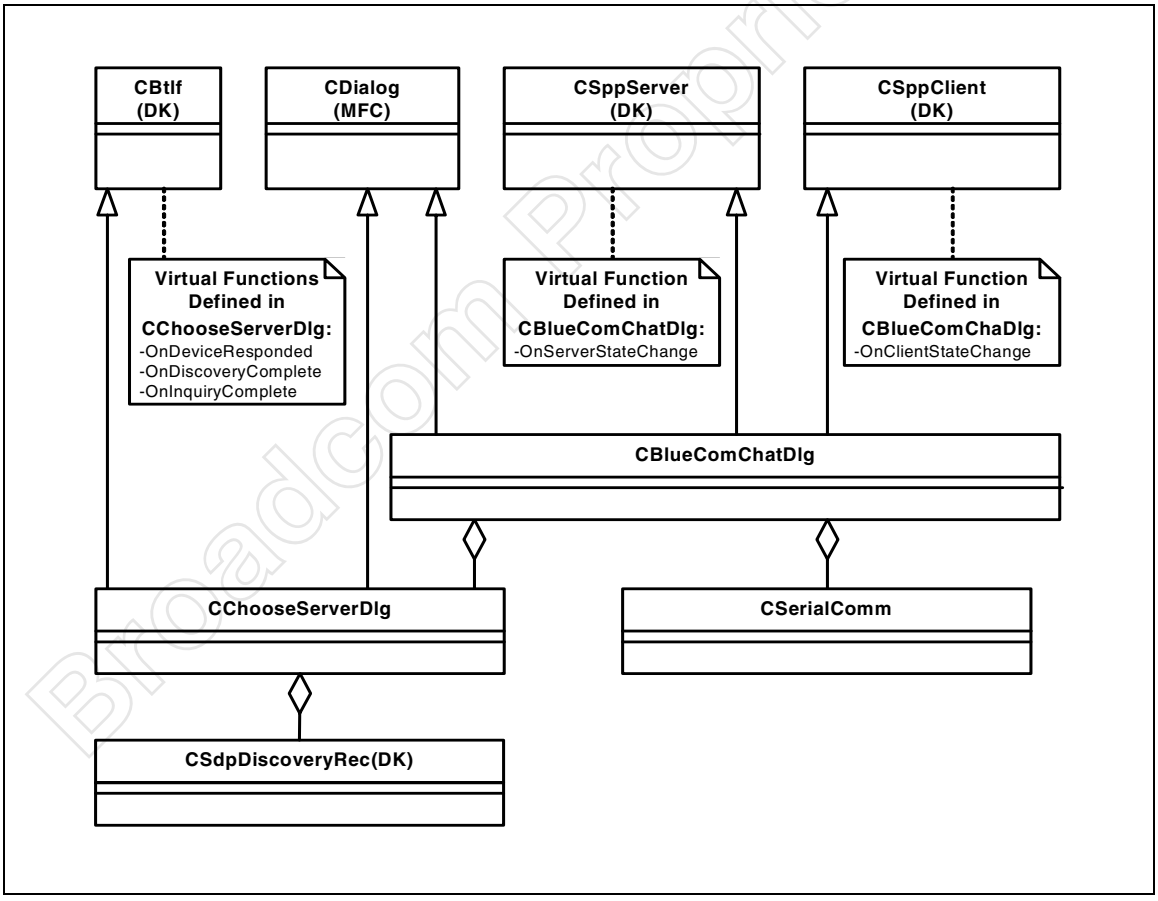


Figure 4: BlueComChat—Primary Class Relationships

CBlueComChatDlg implements the dialog functionality for the main chat window. This module demonstrates the following features of the DK:

- Opening a client or server Bluetooth SPP connection.
- Checking the connection status.
- Opening and closing the associated Windows COM serial port.
- Reading messages entered by the user and writing them to the COM port.
- Reading messages received from the COM port and writing them to the user interface.

This is a typical MFC dialog class, derived on the CDialog base class. For this application, this class also inherits from the DK classes CSppClient and CSppServer. The virtual methods from these classes can then be incorporated into CBlueComChatDlg for convenient access to common class interfaces.

CChooseServerDlg implements the dialog functionality for choosing a chat server. This module demonstrates the following features of the DK:

- Performing a device inquiry.
- Performing service discovery.
- Reading discovery records and displaying only those for the service class serial port.
- Passing the name of the selected service back to CBlueComChatDlg.

CSerialComm implements an interface to the Windows COM serial ports. Methods are provided to Open, Close, Read and Write to the selected COM port.

BLUEOBEX—AN OBEX EXERCISER

The BlueObex sample application provides a user interface to run a sequence of object transfers between an OBEX client and an OBEX server.

BlueObex is installed on two platforms. When they are executed, one is configured as the client, and the other acts is configured as the server by the user.

The application also includes a checkheaders.cpp module, which runs a test sequence validating the CObexHeaders and CObexUserDefined OBEX support classes.

BUILDING THE APPLICATION

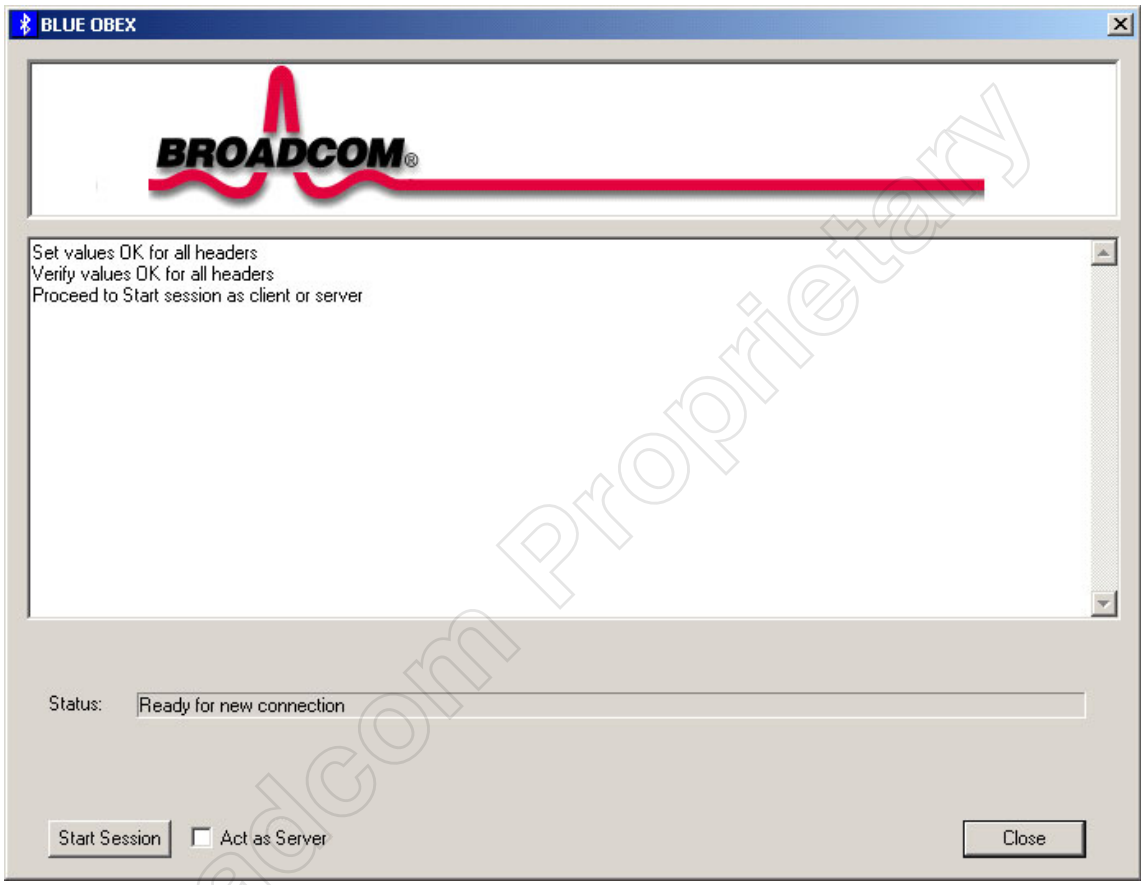
To build the sample, complete these steps:

1. Start Microsoft Visual C++.
2. Open the BlueObex.dsp project (VC++ 6.0) or the BlueObex.vcproj project (VS2005) located in the \Sdk\Samples\BlueObex directory.
3. Build the application.
4. Run BlueObex.exe.

BLUEOBEX FUNCTIONALITY

At startup, the application automatically runs the OBEX headers test functionality from the checkheaders.cpp module. A pass/fail result is logged to the user interface.

Then the user can request an OBEX connection, either as a client or as a server. If server mode is desired, click the **Act as Server** check box, and then click **Start Session** to start an OBEX server session.



checkheaders.cpp Module

The main purpose of the checkheaders.cpp module is to demonstrate the use of all methods offered by the CObexHeaders and CObexUserDefined DK classes.

This module also validates that the methods are consistent. The checkheaders.cpp module defines a class ObexCheckHeaders. This class has two methods, *Fill()* and V*erify()*. These methods are called from the ConnectionMgrDlg module as an automatic part of the startup sequence for the BlueObex application.

The *Fill()* method populates every headers structure in a CObexHeaders object. The return codes from the CObexHeaders methods are checked for errors. The first time an error is detected, *Fill()* returns with an error message indicating the header and method reporting the problem. If all headers are filled successfully, an OK message is returned. The message returned from *Fill()* is reported by CConnectionMgrDlg on the log window.

The *Verify()* method calls the CObexHeaders methods that read out the header values. These are compared to the original input used in *Fill()*. As with *Fill()*, the first error condition causes *Verify()* to stop and return a descriptive error message. If all headers are verified, *Verify()* proceeds to delete each header and verify that the header is no longer accessible. Any error condition is reported as a return error message. If all headers are verified successfully, an OK message is returned and reported on the user's log window.

Test values are defined as static variables at the beginning of the checkheaders.cpp module. The BlueObex user can easily modify them and rebuild the sample application to check for special cases or variations.

OBEX Connection Function

Running in Server Mode

The server initiates an OBEX server connection for the service name defined in m_serviceNameForServer in application source file ConnectionMgrDlg.cpp. The DK is installed with the WIDCOMM DK OBEX Service service name. After registering the service, the server waits for clients to connect.

Running in Client Mode

From the client side, when the user clicks the Start Session button, the application performs device inquiry and service discovery. This process is the same as that used for BlueChat and BlueTime applications, when run as clients. The only difference is that the GUID and service name are different.

A GUID has been defined for this sample application. See variable service_guid in the ConnectionMgrDlg.cpp source file. All servers offering service for this GUID value are displayed on the client user interface. For each discovered service, the server device name and the service name are displayed. When the user selects one of the services an OBEX connection to that server is attempted.

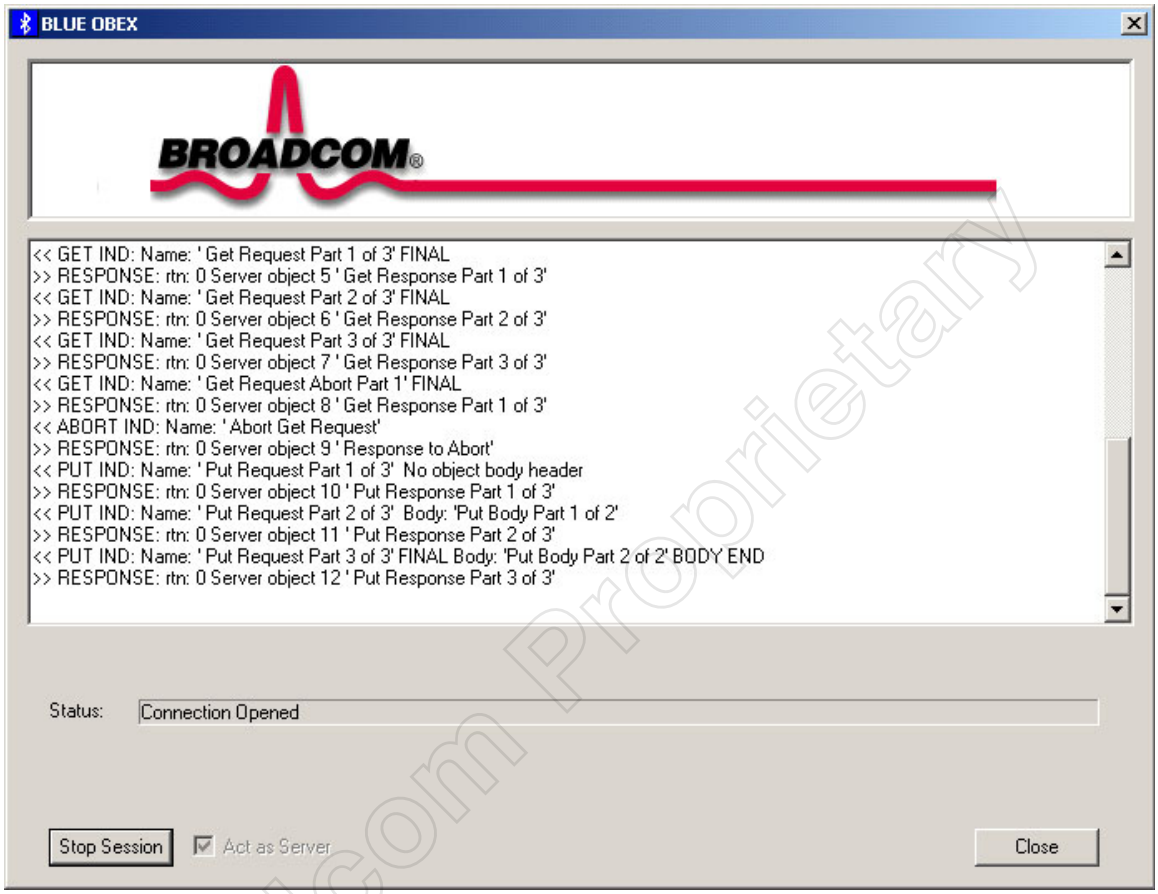
When the connection is established, the client sends a predefined sequence of OBEX objects to the server and monitors the responses.

Viewing Status and Events Information

The user interface shows a status line and a scrolling event log for the client and the server side. The status line reports major events such as connection, close, abort. The event log reports each object sent as `>>...` and each object received as `<<...`

After a connection is established, the Start Session button changes to Stop Session. The user can then stop the session from the client or the server side.

The predefined set of objects to exchange is defined in the ConnectionMgrDlg.cpp module, under labels m_client_list and m_server_list. The list contains a simple Get, a simple Put, a SetPath request, a Create Empty File request, a Delete File request, a multipart Get request, a multipart Get which is aborted, and a multipart Put request.



This list can be easily modified for exercising other OBEX sequences. Each object sent by the client must have a corresponding object to which the server responds.

Exiting BlueObex

To exit the application, click **Close**.

KEY CLASS DESCRIPTIONS

The following diagram shows the relationships between the DK classes and the primary application class, CConnectionMgrDlg.

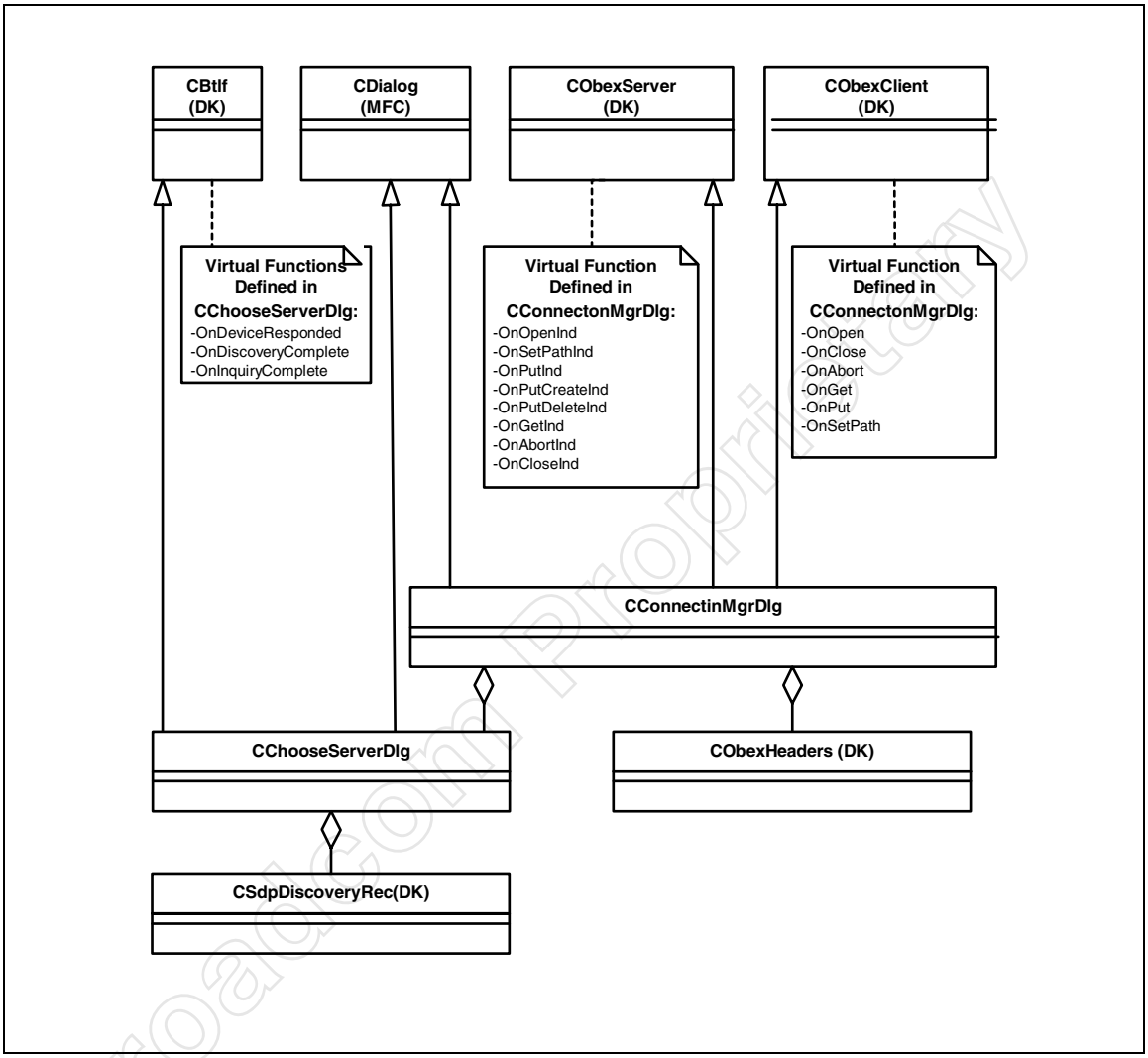


Figure 5: BlueObex—Primary Class Relationships

CConnectionMgrDlg implements the dialog functionality for the Connection Manager window. It is derived from the CObexClient and CObexServer classes defined in the DK.

CObexClient provides the client-side functions for an OBEX connection. CObexServer provides the server-side functions for OBEX connections.

CObexHeaders provides a container for the various OBEX header structures that may be needed to control an OBEX transfer.

BLUEAUDIO—AN AUDIO SAMPLE APPLICATION

This BlueAudio application creates an RFCOMM connection, and then an audio connection on top of RFCOMM, permitting a client and server to play audio files. BlueAudio is installed on two platforms. When they are executed, one is configured as the client, and the other is configured as the server by the user.

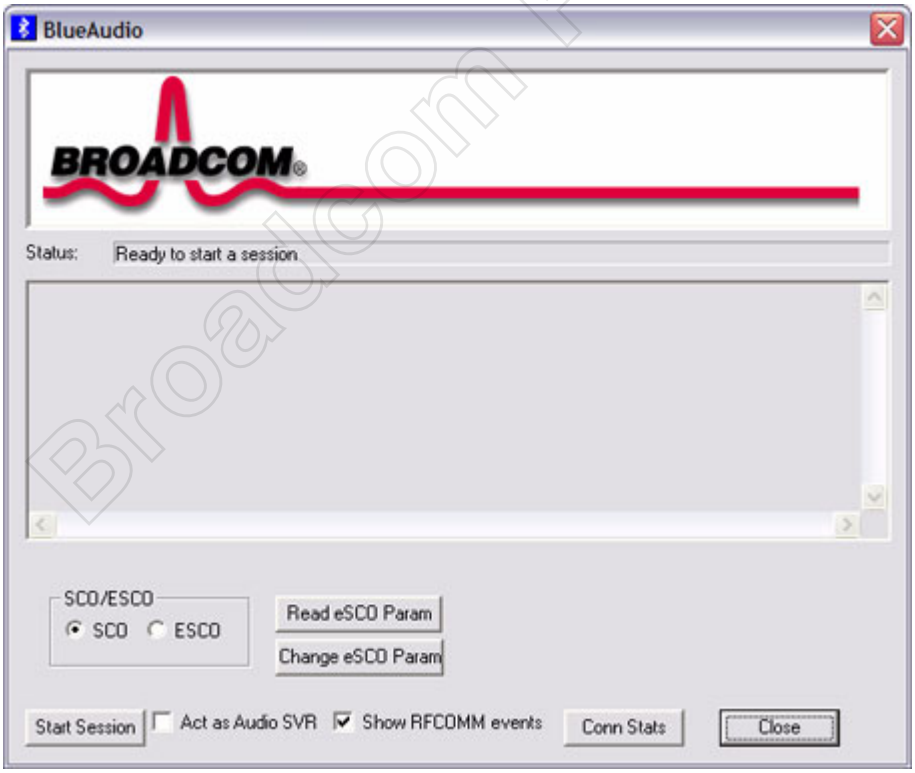
The user establishes the connection (with one PC as the server and the other as the client). The server PC creates a new service, and then waits for the client to connect. The client performs a device inquiry and service discovery. Once an RFCOMM connection is established, the client calls *CreateAudioConnection()* to create an audio connection.

The two sides connect and are ready to send sound files using Windows Sound Recorder or Media Player.

BUILDING THE APPLICATION

1. Start Microsoft Visual C++.
2. Open the BlueAudio.dsp project (VC++ 6.0) or the BlueAudio.vcproj project (VS2005) in the \Sdk\Samples\BlueAudio directory.
3. Build the application.
4. Run BlueAudio.exe.

BLUEAUDIO FUNCTIONALITY



At startup, the user must choose to run the application as either the server or the client.

Running in Server Mode

1. Select the **Act as Audio Server** check box.
2. In the SCO/ESCO pane, select **SCO** or **ESCO** to specify the type of audio connection and the BTW default audio connection type.

   Selecting **ESCO displays** the eSCO Settings window, allowing the user to set the eSCO parameters.

3. Click **Start Session**.

   BlueAudio starts a session, and then performs the following tasks:

   - Adds a record using the CSdpService DK class in the *CBlueAudioDlg::DoCreateServiceRecord()* function.
   - Opens an RFCOMM server port using *OpenServer()* (a method inherited from DK class CRfCommPort).
   - Calls *CreateAudioConnection()*.
   - Waits for a client connection.

   After an eSCO connection is established, the user can do the following:

- Click **Read eSCO Param** to display the eSCO connection parameters.
- Click **Change eSCO Param** to change the following eSCO connection parameters:
  - max_latency
  - pkt types
  - retrans

Running in Client Mode

1. Make sure the **Act as Audio Server** check box is not selected.
2. In the SCO/ESCO pane, select **SCO** or **ESCO** to set the audio connection type and the BTW default audio connection type.

   Selecting **ESCO** displays the eSCO Settings window, allowing the user to set the eSCO parameters.

3. Click **Start Session**.

   BlueAudio starts a client session, and then the Select an Audio Server window appears, listing the Bluetooth devices found using device inquiry.

4. Select an Audio server, and then click **Discover** to send a discovery request to the Audio server.
5. Click **OK** to close the Select an Audio Server window, open an RFCOMM client connection and then an Audio connection with the Audio server.

After the eSCO connection is established, the user can do the following:

- Click **Read eSCO Param** to display the eSCO connection parameters.
- Click **Change eSCO Param** to change the following eSCO connection parameters:
  - max_latency
  - pkt types
  - retrans

BlueAudio is ready to play sound files.

Playing Sound Files

Windows Setup

1. Open the **Start** menu, and then click **Control Panel**.
2. In Control Panel, double-click **Sounds and Audio Devices**.
3. Click on the **Audio** tab.
4. Perform one of the following steps:
   - Server: In the Sound *Recording* pane, set **Default device** to Bluetooth Audio.
   - Client: In the Sound *Playback* pane, set **Default device** to Bluetooth Audio.

Playing and Recording Audio

To play and record an audio track, complete these steps:

- Server: Open the Windows Sound Recorder and create a recording.
- On the Client, open the Windows Media Player and play the recorded sound.

Verifying Operation

To verify that playback is operating properly, play the recorded sound on the server.

Exiting BlueAudio

To exit BlueAudio, click **Close**.

KEY CLASS DESCRIPTIONS

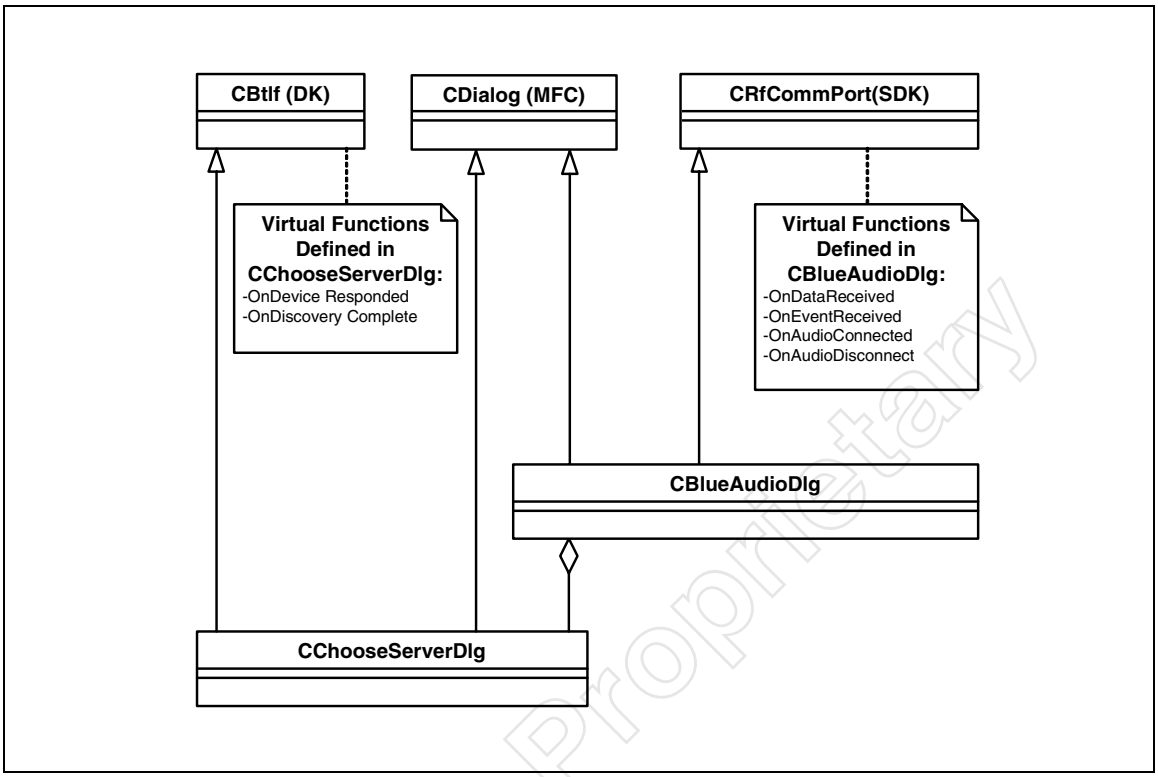


Figure 6: BlueAudio—Primary Class Relationships

CBlueAudioDlg implements the dialog functionality for the main audio window. This module demonstrates the following features of the DK:

- Creating SDP Service records.
- Opening a client or server RFCOMM port.
- Opening a client or server audio connection over the RFCOMM port.
- Writing audio data to the port.
- Reading audio data from the port.
- Closing the audio connection and RFCOMM port.

This is a typical MFC dialog class, derived from the CDialog base class. For this application, this class also inherits from the DK class CRfCommPort. The virtual methods from CRfCommPort can then be incorporated into CBlueAudioDlg for convenient access to common class interfaces.

CChooseServerDlg implements the dialog functionality for choosing an Audio server. This module demonstrates the following features of the DK:

- Using the BT API class
- Performing a device inquiry
- Performing a service discovery
- Reading discovery records

BLUEPRINT—A CPRINTCLIENT APPLICATION

This application provides a user interface for printing a document to a Bluetooth enabled printer.

BUILDING THE APPLICATION

To run the BluePrint sample application, complete these steps:

1. Start Microsoft Visual C++.
2. Open the BluePrint.dsp project (VC++ 6.0) or the BluePrint.vcproj project (VS2005) located in the \Sdk\Samples\BluePrint directory.
3. Build the application.
4. Run BluePrint.exe.

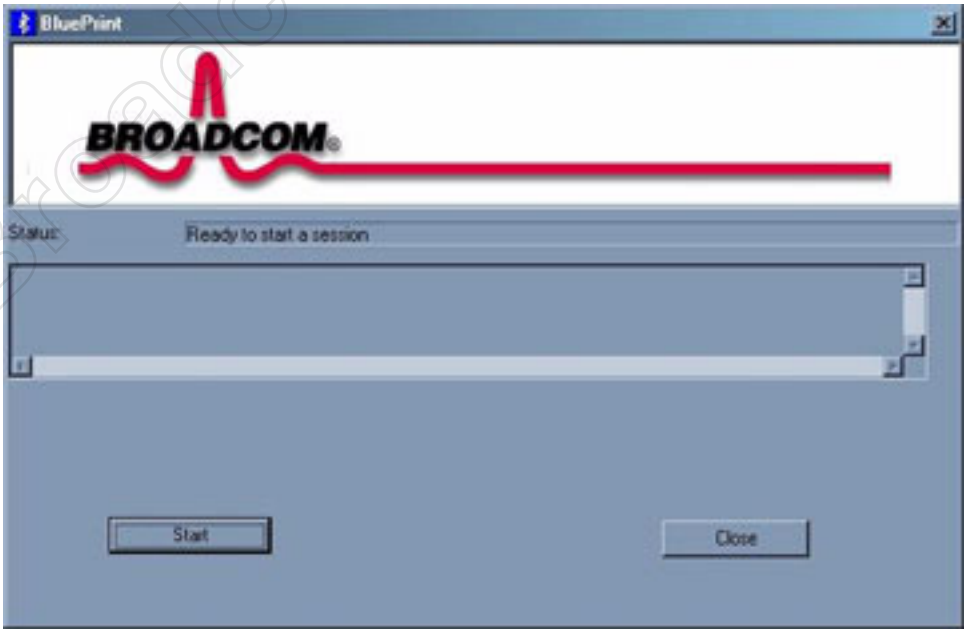
BLUEPRINT FUNCTIONALITY

The BluePrint application performs an inquiry to determine the available Bluetooth printers in the range. After user selects the printer server, discovery is performed on the selected printer server for all supported profiles.

BluePrint supports the HCRP, SPP, and BPP printing profiles. When a print server is selected, BluePrint displays a checkbox for each profile; by default, each profile supported by the server is enabled. Clear the checkboxes of undesired profiles to specify which profile will be used.

If more than one profile is enabled, the SDK uses the first enabled profile that it encounters, in the following order:

1. PRINT_PROFILE_HCRP
2. PRINT_PROFILE_SPP
3. PRINT_PROFILE_BPP



Printing

To issue a print request, complete the following steps:

1. In the BluePrint main window, click **Start Session**.

   A dialog box appears allowing the user to select printer server. The server list is populated automatically with the Bluetooth printer devices using device inquiry.

2. Select the Printer Server from the list.

   A discovery request for supported profiles is sent to the printer.
   Optionally, clear any supported profiles from the checkbox list.

3. Click **Print** to send a temporary file to the printer.

   A temp file is created in the system temp directory.

Exiting BluePrint

To exit the application, click **Close**.

KEY CLASS DESCRIPTIONS

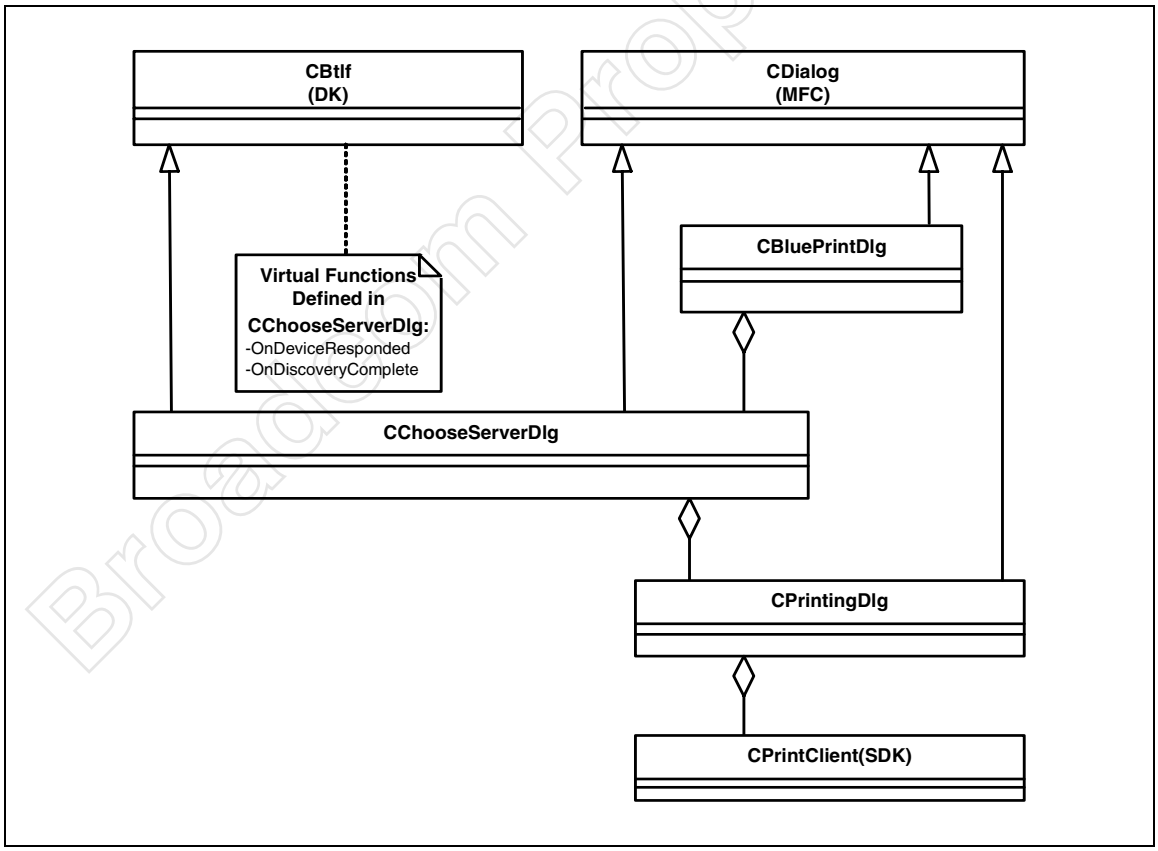


Figure 7: BluePrint—Primary Class Relationships

CBluePrintDlg implements the dialog functionality for the main window.

CChooseServerDlg implements the dialog functionality for choosing the print server. This module demonstrates the following features of the DK:

- Using of the BT API class.
- Performing a device inquiry.
- Performing service discovery.
- Reading discovery records.
- Determining supported print profiles.

CPrintingDlg implements the dialog functionality during a print job. This module demonstrates the following features of the DK:

- Starting a printing job.
- Monitoring a printing job.
- Cancelling a printing job.

The CPrintingDlg class instantiates an object of the CPrintClient class, providing access to the printing methods available in the SDK.

BLUEHEADPHONE - A CHEADPHONECLIENT APPLICATION

BlueHeadphone provides a UI for establishing a stereo audio connection to a device that offers the Bluetooth Advanced Audio Distribution Profile (A2DP) service, such as a Bluetooth stereo headphone. This allows use of the remote device as the audio output device for the computer.

BlueHeadphone performs an inquiry to find all available Bluetooth headphones in range and uses the CHeadphoneClient class to establish a connection.

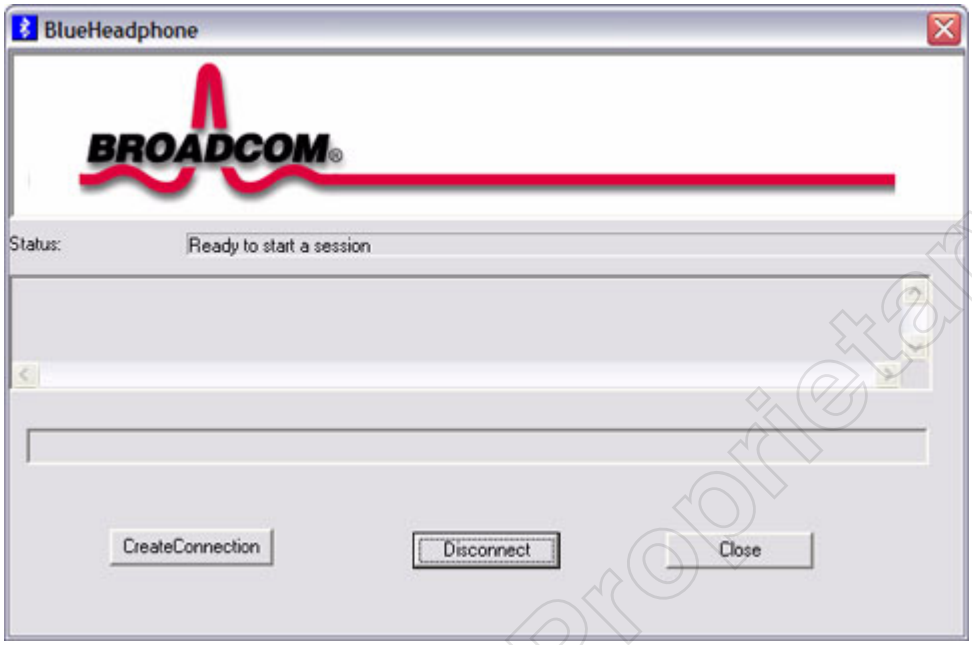
BUILDING THE APPLICATION

To build and run the BlueHeadphone sample application, complete these steps:

1. Start Microsoft Visual C++.
2. Open the BlueHeadphone.dsp project (VC++ 6.0) or the BlueHeadphone.vcproj project (VS2005) located in the \Sdk\Samples\BlueHeadphone directory.
3. Build the Application.
4. Run BlueHeadphone.exe.

BLUEHEADPHONE FUNCTIONALITY

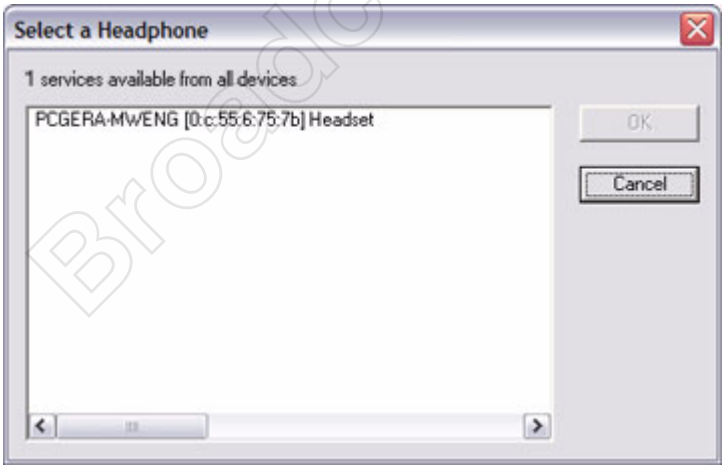
This application performs an inquiry to find available Bluetooth headphones. The application then connects to the Bluetooth headphone selected by the user.



USING BLUEHEADPHONE

1. In the BlueHeadphone main window, click **CreateConnection**.

   The Select a Headphone window appears, allowing the user to select an available headphone.



2. Select a headphone, and then click **OK** to close the window and connect to the headphone.

   Sounds played on the computer can be heard on the remote device.

Closing the Connection

To close the connection, in the BlueHeadphone main window, click **Disconnect**.

Exiting BlueHeadphone

To exit the application, in the BlueHeadphone main window, click **Close**.

KEY CLASS DESCRIPTIONS

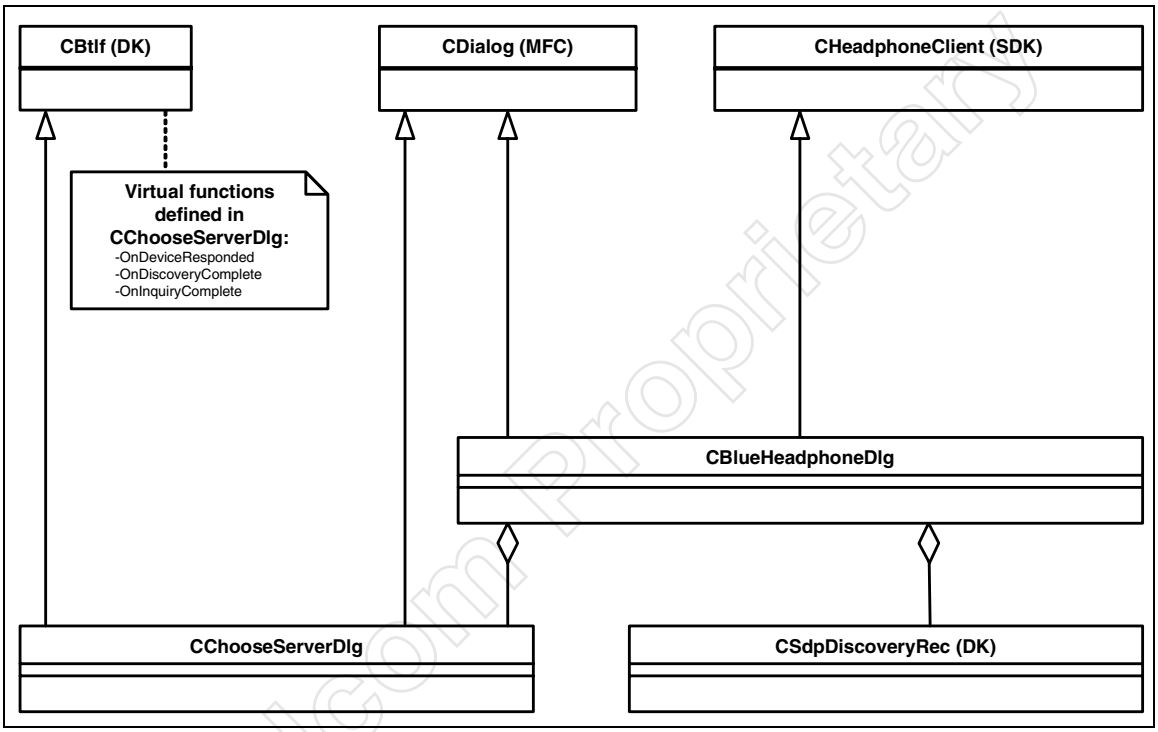


Figure 8: BlueHeadphone - Primary Class Relationships

CBlueHeadphoneDlg implements the dialog functionality for the main windows. This module demonstrates the following features of the DK:

- Registering CHeadphoneClient callback function.
- Connecting to headphone.
- Disconnecting from headphone.

CChooseServerDlg implements the dialog functionality for choosing the headphone. This module demonstrates the following features of the DK.

- Using the CBtIf and CSdpDiscoveryRec classes.
- Performing a device inquiry.
- Performing service discovery.
- Reading discovery records.

Broadcom Corporation

5300 California Avenue
P.O. Box 57013
Irvine, California 92617
Phone: 949-450-8700
Fax: 949-926-5203