BlueTServer.cpp

```
#include "stdafx.h"
#include "BlueTServer.h"
#include "BlueTServerDlg.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

BEGIN_MESSAGE_MAP(CBlueTServerApp, CWinApp)
ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

CBlueTServerApp::CBlueTServerApp()
CBlueTServerApp theApp;

BOOL CBlueTServerApp::InitInstance()
{       if (!AfxSocketInit())
        {       AfxMessageBox(IDP_SOCKETS_INIT_FAILED);
                return FALSE;
        }
        if (!AfxOleInit())
        {       AfxMessageBox(IDP_OLE_INIT_FAILED);
                return FALSE;
        }
        AfxEnableControlContainer();

#ifdef _AFXDLL
        Enable3dControls();
#else
        Enable3dControlsStatic();
#endif
        if (RunEmbedded() || RunAutomated())
        {   COleTemplateServer::RegisterAll();    }
        else
        {   COleObjectFactory::UpdateRegistryAll();   }
        CBlueTServerDlg dlg;
        m_pMainWnd = &dlg;
        int nResponse = dlg.DoModal();
        if (nResponse == IDOK) {}
        Else if (nResponse == IDCANCEL) {}
        return FALSE;}
```

BlueTServerDlg.cpp

```
#include "stdafx.h"
#include "BlueTServer.h"
#include "BlueTServerDlg.h"
#include "DlgProxy.h"
#include "WinampHandler.h"
#include "PowerPointHandler.h"
#include "MediaPlayerHandler.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
#define ID_TASKBARICON 100
#define WM_ICONNOTIFY (WM_USER + 101)
const char* KIniFileName = "\\BlueTServer.ini";
const char* KIniFileSection = "BlueTServer";
const char* KDefaultExtensions[] = { "mp3;mp2;wav;wma","avi;mpeg","ppt"};
```

```
#define DEBUG_LOG(x)	fwrite(x, strlen(x), 1, file); \
			fflush(file);

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();
	enum { IDD = IDD_ABOUTBOX };
protected:
	virtual void DoDataExchange(CDataExchange* pDX);
protected:
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()

CDirTreeNode::CDirTreeNode()
	{
	iFirstChild = NULL;
	iNextSibling = NULL;
	iParent = NULL;
	iFileName = NULL;
	iNodeType = iFileType = 0x0;
	iWrittenToBuffer = FALSE;
	}

CDirTreeNode::~CDirTreeNode()
	{
	delete iFirstChild;
	delete iNextSibling;
	delete iFileName;
	}

IMPLEMENT_DYNAMIC(CBlueTServerDlg, CDialog);

CBlueTServerDlg::CBlueTServerDlg(CWnd* pParent /*=NULL*/) : CDialog(CBlueTServerDlg::IDD, pParent)
{	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
	m_pAutoProxy = NULL;
	iDirTreeRoot = iDirTreeCurrentNode = NULL;
	iBluetoothHandle = INVALID_HANDLE_VALUE;
	iBluetoothThread = NULL;
	iSocketHandle = -1;
	iSocketThread = NULL;
	iErrorCount = 0;
}

CBlueTServerDlg::~CBlueTServerDlg()
{	if (m_pAutoProxy != NULL)
	m_pAutoProxy->m_pDialog = NULL;
	for (int i = 0; i < NUM_HANDLERS; i++)
	delete iAppHandlers[i];
	delete iBluetoothThread;
	delete iSocketThread;
	delete iDirTreeRoot;
}
```

```
BOOL CBlueTServerDlg::OnInitDialog()
{	CDialog::OnInitDialog();
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{	CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{	pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
		}
	}
	SetIcon(m_hIcon, TRUE);
	SetIcon(m_hIcon, FALSE);

	CoInitialize(NULL);

	Log("BlueT Initializing...\r\n");

	iNotifyIconData.cbSize = sizeof(NOTIFYICONDATA);
	iNotifyIconData.hWnd = m_hWnd;
	iNotifyIconData.uID = ID_TASKBARICON;
	iNotifyIconData.uFlags = NIF_ICON | NIF_MESSAGE | NIF_TIP;
	iNotifyIconData.uCallbackMessage = WM_ICONNOTIFY;
	iNotifyIconData.hIcon = (HICON) LoadImage(AfxGetInstanceHandle(),
	MAKEINTRESOURCE(IDI_TRAYICON),IMAGE_ICON, 16, 16, 0);
	strcpy(iNotifyIconData.szTip, "BlueT Server");
	Shell_NotifyIcon(NIM_ADD, &iNotifyIconData);
	GetCurrentDirectory(sizeof(iStartDirectory), iStartDirectory);

	char iniFileName[256];
	sprintf(iniFileName, "%s\\BlueTServer.ini", iStartDirectory);
	FILE* iniFile;
	if (!(iniFile = fopen(iniFileName, "r")))
	{	fclose(iniFile);
	}
	if (iniFile)
		fclose(iniFile);
	ReadIniFile();

	if (strcmp(iCommPortName, ""))
		BluetoothConnect(iCommPortName);
	if (iBluetoothHandle != INVALID_HANDLE_VALUE)
		iBluetoothThread = AfxBeginThread(BluetoothThreadFunction, this);
	if (iListenOnSocket)
		ListenOnSocket();
	iAppHandlers[0] = new CWinampHandler;
	iAppHandlers[1] = new CMediaPlayerHandler;
	iAppHandlers[2] = new CPowerPointHandler;
	iAppHandler = iAppHandlers[iDefaultAppIndex];
	iCurrentAppIndex = iDefaultAppIndex;
	SetTimer(TIMER_ID, 1000, NULL);
	return TRUE;
}

void CBlueTServerDlg::StopListeningOnSocket()
{	if (iSocketThread)
	{	TerminateThread(iSocketThread->m_hThread, 0);
		delete iSocketThread;
		iSocketThread = NULL;
	}
	if (iSocketHandle != -1)
		closesocket(iSocketHandle);
	iSocketHandle = -1;
}
```

```
void CBlueTServerDlg::ListenOnSocket()
{	StopListeningOnSocket();
	WSADATA wsaData;
	WSAStartup(MAKEWORD(1,1), &wsaData);
	iSocketHandle = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
	SOCKADDR_IN sockAddr;
	sockAddr.sin_family = AF_INET;
	sockAddr.sin_addr.s_addr = INADDR_ANY;
	sockAddr.sin_port = htons(7654);
	int nRet;
	nRet = bind(iSocketHandle, (LPSOCKADDR) &sockAddr, sizeof(struct sockaddr));
	if (nRet == SOCKET_ERROR)
	{	Log("Gagal memilih socket %d", nRet);
		closesocket(iSocketHandle);
		iSocketHandle = -1;
		return;
	}

	nRet = listen(iSocketHandle, 1);
	if (nRet == SOCKET_ERROR)
	{	Log("Gagal membaca socket %d", nRet);
		closesocket(iSocketHandle);
		iSocketHandle = -1;
		return;
	}
	if (iSocketHandle != -1)
		iSocketThread = AfxBeginThread(SocketThreadFunction, this);
}

LRESULT CBlueTServerDlg::WindowProc(UINT message, WPARAM wParam, LPARAM lParam)
{	switch (message)
	{
	case WM_ICONNOTIFY:
	{	if (lParam == WM_LBUTTONUP)
		{	if (!iHidden)
			{	ShowWindow(SW_HIDE);
				iHidden = TRUE;
			}
			else
			{	ShowWindow(SW_SHOW);
				WINDOWPLACEMENT placement;
				placement.length = sizeof(placement);
				placement.flags = 0;
				placement.showCmd = SW_RESTORE;
				RECT pos;
				pos.top = 100;
				pos.left = 100;
				pos.right = 550;
				pos.bottom = 645;
				placement.rcNormalPosition = pos;
				SetWindowPlacement(&placement);
				iHidden = FALSE;
			}
			return 0;
		}
	}
		break;
	case WM_TIMER:
		iAppHandler->OneSecondTick();
		return 0;
		break;
	default:
		break;
	}
	return CWnd::WindowProc(message, wParam, lParam);
}
```

```
void CBlueTServerDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CBlueTServerDlg, CDialog)
	//{{AFX_MSG_MAP(CBlueTServerDlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_WM_CLOSE()
	ON_BN_CLICKED(IDC_CONNECT, OnConnect)
	ON_BN_CLICKED(IDC_EXIT, OnExit)
	ON_WM_SIZE()
	ON_BN_CLICKED(IDC_ADDPATH, OnAddpath)
	ON_BN_CLICKED(IDC_BROWSEPATH, OnBrowsepath)
	ON_BN_CLICKED(IDC_CLEARPATH, OnClearpath)
	ON_CBN_SELCHANGE(IDC_APPCOMBO, OnSelchangeAppcombo)
	ON_BN_CLICKED(IDC_STOPWHENSWITCHING, OnStopwhenswitching)
	ON_BN_CLICKED(IDC_LISTENONSOCKET, OnListenonsocket)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

void CBlueTServerDlg::WriteIniFile()
{	char path[MAX_PATH + 20];
	strcpy(path, iStartDirectory);
	strcat(path, KIniFileName);
	char buffer[256];
	WritePrivateProfileString(KIniFileSection, "CommPort", iCommPortName, path);
	SendDlgItemMessage(IDC_EDITPATH, WM_GETTEXT, sizeof(iRootPath), (LPARAM)iRootPath);
	WritePrivateProfileString(KIniFileSection, "Path", iRootPath, path);
	int startAutomatically = SendDlgItemMessage(IDC_STARTAPP, BM_GETCHECK, 0, 0);
	sprintf(buffer, "%d", startAutomatically);
	WritePrivateProfileString(KIniFileSection, "StartApplicationAutomatically", buffer, path);
	iDefaultAppIndex = SendDlgItemMessage(IDC_APPCOMBO, CB_GETCURSEL, 0, 0);
	sprintf(buffer, "%d", iDefaultAppIndex);
	WritePrivateProfileString(KIniFileSection, "DefaultApplication", buffer, path);

	for (int i = 0; i < NUM_HANDLERS; i++)
	{	sprintf(buffer, "PathToApplication%d", i);
		if (i == iDefaultAppIndex)
		{	char currentPath[MAX_PATH];
			SendDlgItemMessage(IDC_APPPATH, WM_GETTEXT, sizeof(currentPath),
			(LPARAM) currentPath);
			WritePrivateProfileString(KIniFileSection, buffer, currentPath, path);
			sprintf(buffer, "ExtensionsHandled%d", i);
			SendDlgItemMessage(IDC_FILETYPE, WM_GETTEXT, sizeof(currentPath),
			(LPARAM) currentPath);
			WritePrivateProfileString(KIniFileSection, buffer, currentPath, path);
		}
		else
		{	WritePrivateProfileString(KIniFileSection, buffer, iAppPaths[i], path);
			sprintf(buffer, "ExtensionsHandled%d", i);
			WritePrivateProfileString(KIniFileSection, buffer, iExtensionsHandled[i], path);
		}
	}

	int listenOnSocket = SendDlgItemMessage(IDC_LISTENONSOCKET, BM_GETCHECK, 0, 0);
	sprintf(buffer, "%d", listenOnSocket);
	WritePrivateProfileString(KIniFileSection, "ListenOnSocket", buffer, path);
	}

void CBlueTServerDlg::ReadIniFile()
{	char path[MAX_PATH + 20];
	strcpy(path, iStartDirectory);
	strcat(path, KIniFileName);
```

```
	char buffer[256];
	GetPrivateProfileString(KIniFileSection, "CommPort", "", iCommPortName, sizeof(iCommPortName), path);
	SendDlgItemMessage(IDC_COMPORT, WM_SETTEXT, 0, (LPARAM)iCommPortName);
	GetPrivateProfileString(KIniFileSection, "Path", "", iRootPath, sizeof(iRootPath), path);
	if ((strlen(iRootPath) > 0) && (iRootPath[strlen(iRootPath)-1] == '\\'))
		iRootPath[strlen(iRootPath)-1] = '\0';
	SendDlgItemMessage(IDC_EDITPATH, WM_SETTEXT, 0, (LPARAM)iRootPath);
	GetPrivateProfileString(KIniFileSection, "ListenOnSocket", "0", buffer, sizeof(buffer), path);
	sscanf(buffer, "%d", &iListenOnSocket);
	SendDlgItemMessage(IDC_LISTENONSOCKET, BM_SETCHECK, iListenOnSocket, 0);
	GetPrivateProfileString(KIniFileSection, "DefaultApplication", "0", buffer, sizeof(buffer), path);
	sscanf(buffer, "%d", &iDefaultAppIndex);
	SendDlgItemMessage(IDC_APPCOMBO, CB_SETCURSEL, iDefaultAppIndex, 0);

	for (int i = 0; i < NUM_HANDLERS; i++)
	{	sprintf(buffer, "PathToApplication%d", i);
		GetPrivateProfileString(KIniFileSection, buffer, "", iAppPaths[i],sizeof(iAppPaths[i]), path);
		if (i == iDefaultAppIndex)
			SendDlgItemMessage(IDC_APPPATH, WM_SETTEXT, 0, (LPARAM) iAppPaths[i]);
		sprintf(buffer, "ExtensionsHandled%d", i);
		GetPrivateProfileString(KIniFileSection, buffer, KDefaultExtensions[i], iExtensionsHandled[i],
			sizeof(iExtensionsHandled[i]), path);
		if (i == iDefaultAppIndex)
			SendDlgItemMessage(IDC_FILETYPE, WM_SETTEXT, 0, (LPARAM)
				iExtensionsHandled[i]);
	}
}

UINT CBlueTServerDlg::BluetoothThreadFunction(LPVOID pParam)
{	CBlueTServerDlg* self = (CBlueTServerDlg*) pParam;
	self->HandleBluetooth();
	return 0;
}

UINT CBlueTServerDlg::SocketThreadFunction(LPVOID pParam)
{	CBlueTServerDlg* self = (CBlueTServerDlg*) pParam;
	self->HandleSocket();
	return 0;
}

void CBlueTServerDlg::HandleBluetooth()
{	iExitNow = FALSE;
	do
	{	ReadCommand(iBluetoothHandle);
		if (iErrorCount >= 10)
		{	Log("Terlalu banyak error\r\n");
			break;
		}
	}
	while (!iExitNow);
	CloseHandle(iBluetoothHandle);
}

void CBlueTServerDlg::HandleSocket()
{	iExitNow = FALSE;
	SOCKET readSocket;
	do
	{	iClientClosed = FALSE;
		readSocket = accept(iSocketHandle, NULL, NULL);
		Log("Client terhubung dengan socket\r\n");
		do
		{	ReadCommand(NULL, readSocket);
			if (iErrorCount >= 10)
			{	Log("Terlalu banyak error\r\n");
				closesocket(readSocket);
				closesocket(iSocketHandle);
				return;
```

```
                        }
                }
        while (!iClientClosed);
        closesocket(readSocket);
        }
        while (!iExitNow);
        closesocket(iSocketHandle);
}

void CBlueTServerDlg::ReadCommand(HANDLE aHandle, SOCKET aSocket)
{       char buffer[5];
        unsigned long bytes_read = 0;
        memset(buffer, 0, sizeof(buffer));
        BOOL readOk = TRUE;
        if (aHandle)
        {       readOk = ReadFile(aHandle, buffer, 4, &bytes_read, NULL);
        }
        else
        {       int nRet;
                char* bufPtr = buffer;
                do
                {       nRet = recv(aSocket, bufPtr, 4 - (bufPtr - buffer), 0);
                        if (nRet == 0)
                        {       Log("Client telah terputus\r\n");
                                iClientClosed = TRUE;
                                return;
                        }
                        else if (nRet < 0)
                        {       readOk = FALSE;
                                break;
                        }
                        bufPtr += nRet;
                }
                while ((bufPtr - buffer) < 4);
        }
        if (!readOk)
        {       Log("Perintah tidak dapat dibaca : %d\r\n", GetLastError());
                iErrorCount++;
        }
        else
        {       if (strcmp(buffer, "PLAY") && strcmp(buffer, "CHCK") && strcmp(buffer, "VERS"))
                        CheckAppRunning();
                if (!strcmp(buffer, "CHCK"))
                        SendAck(aSocket);
                else if (!strcmp(buffer, "LIST"))
                        WriteListToPhone(aSocket);
                else if (!strcmp(buffer, "DLST"))
                        WriteDirectoryListToPhone(aSocket);
                else if (!strcmp(buffer, "INFO"))
                        WriteInfoToPhone(aSocket);
                else if (!strcmp(buffer, "PLAY"))
                        PlayFile(FALSE, aSocket);
                else if (!strcmp(buffer, "LADD"))
                        PlayFile(TRUE, aSocket);
                else if (!strcmp(buffer, "FINF"))
                        GetFileInfo(aSocket);
                else if (!strcmp(buffer, "DOWN"))
                        DownloadFile(aSocket);
                else if (!strcmp(buffer, "VOLM"))
                        SetVolume(aSocket);
                else if (!strcmp(buffer, "GVOL"))
                        GetVolume(aSocket);
                else if (!strcmp(buffer, "STOP"))
                        Stop(EImmediate);
                else if (!strcmp(buffer, "FADE"))
                        Stop(EFadeOut);
                else if (!strcmp(buffer, "STEN"))
```

```
                        Stop(EAtEndOfSong);
                else if (!strcmp(buffer, "PAUS"))
                        Pause();
                else if (!strcmp(buffer, "STRT"))
                        Play();
                else if (!strcmp(buffer, "PREV"))
                        PreviousTrack();
                else if (!strcmp(buffer, "NEXT"))
                        NextTrack();
                else if (!strcmp(buffer, "FFWD"))
                        FastForward();
                else if (!strcmp(buffer, "RWND"))
                        Rewind();
                else if (!strcmp(buffer, "SHFL"))
                        Shuffle(aSocket);
                else if (!strcmp(buffer, "REPT"))
                        Repeat(aSocket);
                else if (!strcmp(buffer, "PLST"))
                        WritePlaylist(aSocket);
                else if (!strcmp(buffer, "SLCT"))
                        SelectInPlaylist(aSocket);
                else if (!strcmp(buffer, "RMAL"))
                        RemoveAllFromPlaylist();
                else if (!strcmp(buffer, "DINF"))
                        GetDetailedInfo(aSocket);
                else if (!strcmp(buffer, "SHUT"))
                        ShutDown();
                else if (!strcmp(buffer, "FULL"))
                        FullScreen();
                else if (!strcmp(buffer, "SEEK"))
                        SetSeek(aSocket);
                else if (!strcmp(buffer, "PLEN"))
                        WritePlaylistLength(aSocket);
                else if (!strcmp(buffer, "VERS"))
                        WriteVersion(aSocket);
                else if (!strcmp(buffer, "EXIT"))
                        exit(0);
                else
                Log("Perintah tidak diketahui \"%s\"\r\n", buffer);
        }
}

void CBlueTServerDlg::ReadDir(char* aDir, CDirTreeNode* aNode, BOOL aRecursive)
{       if (!aDir)
        {       Log("Tidak dapat membuka direktori\r\n");
                return;
        }
        char fileNameBuf[1024];
        sprintf(fileNameBuf, "%s\\*.*", aDir);
        WIN32_FIND_DATA findData;
        int foundCount = 0;
        HANDLE findHandle = FindFirstFile(fileNameBuf, &findData);
        if (findHandle == INVALID_HANDLE_VALUE)
        {       Log("Tidak dapat membuka \"%s\"\r\n", aDir);
                return;
        }

        CDirTreeNode* currentNode = aNode;
        do
        {       if (!strcmp(findData.cFileName, ".") || !strcmp(findData.cFileName, ".."))
                        continue;
                if (foundCount == 0)
                {       aNode->iFirstChild = new CDirTreeNode;
                        aNode->iFirstChild->iParent = currentNode;
                        currentNode = aNode->iFirstChild;
                        currentNode->iNodeType = 0x1; // child
                }
```

```
                else
                {       currentNode->iNextSibling = new CDirTreeNode;
                        currentNode->iNextSibling->iParent = currentNode->iParent;
                        currentNode = currentNode->iNextSibling;
                        currentNode->iNodeType = 0x3; // sibling
                }
                foundCount++;
                sprintf(fileNameBuf, "%s\\%s", aDir, findData.cFileName);
                currentNode->iFileName = strdup(findData.cFileName);
                if (findData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
                {       if (aRecursive)
                        {       currentNode->iFileType = 0x1;
                                ReadDir(fileNameBuf, currentNode, TRUE);
                        }
                        else
                        {       currentNode->iFileType = 0x3;
                        }
                }
                else
                {       currentNode->iFileType = 0x2;
                }
        }
        while (FindNextFile(findHandle, &findData));
        if (foundCount > 0)
        {       if (currentNode->iNodeType == 0x1)
                        currentNode->iNodeType = 0x2;
                if (currentNode->iNodeType == 0x3)
                        currentNode->iNodeType = 0x4;
        }
        FindClose(findHandle);
}

BOOL CBlueTServerDlg::CheckAppRunning()
{       int startAutomatically = SendDlgItemMessage(IDC_STARTAPP, BM_GETCHECK, 0, 0);
        return iAppHandler->CheckRunning(iAppPaths[iCurrentAppIndex], startAutomatically);
}

void CBlueTServerDlg::WriteInfoToPhone(SOCKET aSocket)
{       unsigned char isPlaying;
        int songLength;
        int currentTime;
        BOOL shuffle;
        BOOL repeat;
        char songNameGuess[256];
        memset(songNameGuess, 0, sizeof(songNameGuess));
        iAppHandler->GetInfo(isPlaying, songLength, currentTime, shuffle, repeat, songNameGuess,
                sizeof(songNameGuess));
        char buffer[1024];
        strcpy(buffer, "INFOACK");
        buffer[7] = isPlaying;
        buffer[8] = songLength >> 24;
        buffer[9] = (songLength >> 16) & 0xFF;
        buffer[10] = (songLength >> 8) & 0xFF;
        buffer[11] = songLength & 0xFF;
        buffer[12] = currentTime >> 24;
        buffer[13] = (currentTime >> 16) & 0xFF;
        buffer[14] = (currentTime >> 8) & 0xFF;
        buffer[15] = currentTime & 0xFF;
        buffer[16] = shuffle;
        buffer[17] = repeat;
        strcpy(&buffer[18], songNameGuess);
        unsigned long bytesWritten;
        if (aSocket)
                bytesWritten = send(aSocket, buffer, 18 + strlen(songNameGuess), 0);
        else
                WriteFile(iBluetoothHandle, buffer, 18 + strlen(songNameGuess), &bytesWritten, NULL);
        if (bytesWritten < 18)
```

```
		Log("Menulis %d bytes dari INFOACK\r\n", bytesWritten);
}

void CBlueTServerDlg::WriteVersion(SOCKET aSocket)
{	char buffer[9];
	memset(buffer, 0, sizeof(buffer));
	strcpy(buffer, "VERSACK");
	buffer[7] = MAJOR_VERSION;
	buffer[8] = MINOR_VERSION;
	unsigned long bytesWritten;
	if (aSocket)
		bytesWritten = send(aSocket, buffer, 9, 0);
	else
		WriteFile(iBluetoothHandle, buffer, 9, &bytesWritten, NULL);
	Log("Menuliskan versi...\r\n");
}

void CBlueTServerDlg::PlayFile(BOOL aAddToPlaylist, SOCKET aSocket)
{	unsigned long bytes = 0;
	unsigned char buffer[1024];
	int filenameLength = 0;
	memset(buffer, 0, sizeof(buffer));
	if (!ReadFromPhone(aSocket, buffer, 2))
	{	Log("Tidak dapat membaca panjang file : %d\r\n", GetLastError());
		return;
	}
	filenameLength = (buffer[0] << 8) + buffer[1];
	buffer[0] = '\0';
	buffer[1] = '\0';
	if (filenameLength >= sizeof(buffer))
	{	Log("Nama file (%d) terlalu panjang", filenameLength);
		return;
	}
	if (!ReadFromPhone(aSocket, buffer, filenameLength))
	{	Log("Tidak dapat membaca nama file %d: %d\r\n", filenameLength, GetLastError());
		return;
	}
	CAppHandler* handler = FindHandlerForFile((char*) buffer);
	if (handler != iAppHandler)
	{	Stop(EImmediate);
	}
	iAppHandler = handler;
	CheckAppRunning();
	if (aAddToPlaylist)
	{	Log("Menambahkan \"%s\" dalam playlist\r\n", buffer);
		iAppHandler->AddSongToPlaylist((char*) buffer);
	}
	else
	{	Log("Playing... \"%s\"\r\n", buffer);
		iAppHandler->PlaySong((char*) buffer);
	}
}

void CBlueTServerDlg::GetFileInfo(SOCKET aSocket)
{	unsigned long bytes = 0;
	unsigned char buffer[1024];
	int filenameLength = 0;
	memset(buffer, 0, sizeof(buffer));
	if (!ReadFromPhone(aSocket, buffer, 2))
	{	Log("Tidak dapat membaca panjang file : %d\r\n", GetLastError());
		return;
	}
	filenameLength = (buffer[0] << 8) + buffer[1];
	buffer[0] = '\0';
	buffer[1] = '\0';
	if (filenameLength >= sizeof(buffer))
	{	Log("Nama file (%d) terlalu panjang", filenameLength);
```

```
                return;
        }
        if (!ReadFromPhone(aSocket, buffer, filenameLength))
        {       Log("Nama file tidak dapat dibaca : %d\r\n", GetLastError());
                return;
        }
        Log("Mengirim informasi \"%s\": ", buffer);
        FILE* file = fopen((char*) buffer, "rb");
        if (file)
        {       fseek(file, 0, SEEK_END);
                long fileLength = ftell(file);
                fseek(file, 0, SEEK_SET);
                Log("Panjang file %d\r\n", fileLength);
                char ackBuf[11];
                strcpy(ackBuf, "FINFACK");
                ackBuf[7] = (fileLength >> 24) & 0xFF;
                ackBuf[8] = (fileLength >> 16) & 0xFF;
                ackBuf[9] = (fileLength >> 8) & 0xFF;
                ackBuf[10] = fileLength & 0xFF;
                if (aSocket)
                        bytes = send(aSocket, ackBuf, 11, 0);
                else
                        WriteFile(iBluetoothHandle, ackBuf, 11, &bytes, NULL);
        }
        else
        {       Log("file tidak dapat dibuka\r\n");
        }
        fclose(file);
}

void CBlueTServerDlg::WritePlaylist(SOCKET aSocket)
{       unsigned long bytesWritten = 0;
        int playlistLength;
        int playlistPos;
        char* playlist;
        char* playlistPtr;
        Log("Menulis playlist...\r\n");
        iAppHandler->GetPlaylist(playlist, playlistLength, playlistPos);
        playlistPtr = playlist;
        unsigned char buffer[10];
        strcpy((char*) buffer, "PLSTACK");
        buffer[7] = (playlistPos >> 8) & 0xFF;
        buffer[8] = playlistPos & 0xFF;
        if (aSocket)
                bytesWritten = send(aSocket, (char*)buffer, 9, 0);
        else
                WriteFile(iBluetoothHandle, buffer, 9, &bytesWritten, NULL);
        bytesWritten = 0;
        if (playlist == NULL)
        {       Log("Tidak dapat menemukan playlist\r\n");
                buffer[9] = '\0';
                if (aSocket)
                        bytesWritten = send(aSocket, (char*)buffer, 10, 0);
                else
                        WriteFile(iBluetoothHandle, buffer, 10, &bytesWritten, NULL);
                return;
        }
        do
        {       if (aSocket)
                        bytesWritten = send(aSocket, playlistPtr, playlistLength, 0);
                else
                        WriteFile(iBluetoothHandle, playlistPtr, playlistLength, &bytesWritten, NULL);
                if (bytesWritten < (unsigned long) playlistLength)
                {       playlistPtr += bytesWritten;
                        playlistLength -= bytesWritten;
                        bytesWritten = 0;
                }
```

```
	}
	while (bytesWritten < (unsigned long) playlistLength);
	delete playlist;
	Log("Selesai menuliskan playlist\r\n");
}

void CBlueTServerDlg::DownloadFile(SOCKET aSocket)
{	unsigned long bytes = 0;
	unsigned char buffer[1024];
	unsigned char* bufferPtr;
	int filenameLength = 0;
	memset(buffer, 0, sizeof(buffer));
	if (!ReadFromPhone(aSocket, buffer, 2))
	{	Log("Tidak dapat membaca panjang file : %d\r\n", GetLastError());
		return;
	}
	filenameLength = (buffer[0] << 8) + buffer[1];
	buffer[0] = '\0';
	buffer[1] = '\0';
	if (filenameLength >= sizeof(buffer))
	{	Log("Nama File (%d) terlalu panjang", filenameLength);
		return;
	}
	if (!ReadFromPhone(aSocket, buffer, filenameLength))
	{	Log("Nama File tidak terbaca : %d\r\n", GetLastError());
		return;
	}
	Log("Sedang Mengupload \"%s\"... ", buffer);
	FILE* file = fopen((char*) buffer, "rb");
	unsigned long bytesRead;
	if (file)
	{	char ackBuf[8];
		strcpy(ackBuf, "DOWNACK");
		if (aSocket)
			bytes = send(aSocket, ackBuf, 7, 0);
		else
			WriteFile(iBluetoothHandle, ackBuf, 7, &bytes, NULL);
		do
		{	bytesRead = fread(buffer, 1, sizeof(buffer), file);
			bufferPtr = buffer;
			bytes = 0;
			if (bytesRead > 0)
			{
			do
			{	bytesRead -= bytes;
				bufferPtr += bytes;
				if (aSocket)
					bytes = send(aSocket, (char*)bufferPtr, bytesRead, 0);
				else
					WriteFile(iBluetoothHandle, bufferPtr, bytesRead, &bytes, NULL);
			}
			while ((bytes > 0) && (bytes < bytesRead));
			}
		}
		while (bytesRead > 0);
		Log("Selesai\r\n");
	}
	else
	{	Log("Tidak dapat membuka file\r\n");
	}
	fclose(file);
}

void CBlueTServerDlg::SetVolume(SOCKET aSocket)
{	unsigned long bytesRead = 0;
	unsigned char volume = 0;
```

```
	if (!ReadFromPhone(aSocket, &volume, 1))
	{	Log("Volume tidak terbaca : %d\r\n", GetLastError());
		return;
	}
	iAppHandler->SetVolume(volume);
	Log("Set volume... %d%%\r\n", (volume * 100) / 255);
}

void CBlueTServerDlg::GetVolume(SOCKET aSocket)
{	int volume = iAppHandler->GetVolume();
	char buffer[9];
	unsigned long bytesWritten;
	if (volume >= 0)
	{	strcpy(buffer, "GVOLACK");
		buffer[7] = (char) volume;
		if (aSocket)
			bytesWritten = send(aSocket, buffer, 8, 0);
		else
			WriteFile(iBluetoothHandle, buffer, 8, &bytesWritten, NULL);
		Log("Menuliskan volume... (%2.2f%%) \r\n", (volume * 100.0) / 255.0);
	}
	else
	{	strcpy(buffer, "GVOLNAK");
		if (aSocket)
			bytesWritten = send(aSocket, buffer, 7, 0);
		else
			WriteFile(iBluetoothHandle, buffer, 7, &bytesWritten, NULL);
		Log("Tidak dapat membaca volume (ret=%d)\r\n", volume);
	}
}

void CBlueTServerDlg::Play()
{	iAppHandler->Play();
	Log("Play\r\n");
}

void CBlueTServerDlg::Stop(TStopMode aStopMode)
{	iAppHandler->Stop(aStopMode);
	Log("Stop\r\n");
}

void CBlueTServerDlg::Pause()
{	iAppHandler->Pause();
	Log("Pause\r\n");
}

void CBlueTServerDlg::NextTrack()
{	iAppHandler->NextTrack();
	Log("Next\r\n");
}

void CBlueTServerDlg::PreviousTrack()
{	iAppHandler->PreviousTrack();
	Log("Previous\r\n");
}

void CBlueTServerDlg::FastForward()
{	iAppHandler->FastForward();
	Log("Fast forward\r\n");
}

void CBlueTServerDlg::Rewind()
{	iAppHandler->Rewind();
	Log("Rewind\r\n");
}
```

```
void CBlueTServerDlg::FullScreen()
{	iAppHandler->FullScreen();
	Log("Full screen\r\n");
}

void CBlueTServerDlg::Shuffle(SOCKET aSocket)
{	unsigned long bytesRead = 0;
	unsigned char shuffleOn = 0;
	if (!ReadFromPhone(aSocket, &shuffleOn, 1))
	{	Log("Tidak dapat membaca shuffle: %d\r\n", GetLastError());
		return;
	}
	iAppHandler->SetShuffle(shuffleOn);
	Log("Shuffle : %s\r\n", shuffleOn ? "on" : "off");
}

void CBlueTServerDlg::Repeat(SOCKET aSocket)
{	unsigned long bytesRead = 0;
	unsigned char repeatOn = 0;
	if (!ReadFromPhone(aSocket, &repeatOn, 1))
	{	Log("Tidak dapat membaca repeat: %d\r\n", GetLastError());
		return;
	}
	iAppHandler->SetRepeat(repeatOn);
	Log("Repeat : %s\r\n", repeatOn ? "on" : "off");
}

void CBlueTServerDlg::SendAck(SOCKET aSocket)
{	char ackBuf = 'Y';
	unsigned long bytes;
	if (aSocket)
		bytes = send(aSocket, (char*)&ackBuf, 1, 0);
	else
		WriteFile(iBluetoothHandle, &ackBuf, 1, &bytes, NULL);
	Log("Mengirimkan acknowledge...\r\n");
}

void CBlueTServerDlg::SelectInPlaylist(SOCKET aSocket)
{	unsigned long bytesRead = 0;
	unsigned char indexBytes[2];
	if (!ReadFromPhone(aSocket, indexBytes, 2))
	{	Log("Tidak dapat membaca index playlist: %d\r\n", GetLastError());
		return;
	}
	iAppHandler->SelectInPlaylist((indexBytes[0] << 8) + indexBytes[1]);
	Log("File yang dipilih %d\r\n", (indexBytes[0] << 8) + indexBytes[1]);
}

void CBlueTServerDlg::RemoveAllFromPlaylist()
{	iAppHandler->RemoveAllFromPlaylist();
	Log("Menghapus playlist\r\n");
}

void CBlueTServerDlg::GetDetailedInfo(SOCKET aSocket)
{	int bitRate;
	int sampleRate;
	int channels;
	iAppHandler->GetDetailedInfo(bitRate, sampleRate, channels);
	char infoBuf[256];
	unsigned long bytes;
	strcpy(infoBuf, "DINFACK");
	CopyIntIntoBuffer(bitRate, &infoBuf[7]);
	CopyIntIntoBuffer(sampleRate, &infoBuf[11]);
	CopyIntIntoBuffer(channels, &infoBuf[15]);
	if (aSocket)
		bytes = send(aSocket, infoBuf, 19, 0);
	else
```

```
                WriteFile(iBluetoothHandle, &infoBuf, 19, &bytes, NULL);
        Log("Mengirimkan detail...\r\n");
}

BOOL CBlueTServerDlg::ReadFromPhone(SOCKET aSocket, unsigned char* aBuffer, int aLength)
{       BOOL success = TRUE;
        int bytes;
        if (aSocket)
        {       bytes = recv(aSocket, (char*)aBuffer, aLength, 0);
                if (bytes <= 0)
                        success = FALSE;
        }
        else
        {       if (!ReadFile(iBluetoothHandle, aBuffer, aLength, (unsigned long*) &bytes, NULL))
                        success = FALSE;
        }
        return success;
}

void CBlueTServerDlg::CopyIntIntoBuffer(int aValue, char* aBuffer)
{       aBuffer[0] = (aValue >> 24) & 0xFF;
        aBuffer[1] = (aValue >> 16) & 0xFF;
        aBuffer[2] = (aValue >> 8) & 0xFF;
        aBuffer[3] = aValue & 0xFF;
}

BOOL CBlueTServerDlg::SetPrivilege(HANDLE hToken, LPCTSTR lpszPrivilege, BOOL bEnablePrivilege)
{       TOKEN_PRIVILEGES tp;
        LUID luid;
        if (!LookupPrivilegeValue(NULL, lpszPrivilege,&luid))
        {       Log("Error mencari privilegevalue : %u\n", GetLastError());
                return FALSE;
        }
        tp.PrivilegeCount = 1;
        tp.Privileges[0].Luid = luid;
        if (bEnablePrivilege)
                tp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
        else
                tp.Privileges[0].Attributes = 0;
                AdjustTokenPrivileges(hToken, FALSE, &tp, sizeof(TOKEN_PRIVILEGES),
                        (PTOKEN_PRIVILEGES) NULL, (PDWORD) NULL);
        if (GetLastError() != ERROR_SUCCESS)
        {       Log("Gagal mengubah token privileges: %u\n", GetLastError());
                return FALSE;
        }
        return TRUE;
}

void CBlueTServerDlg::ShutDown()
{       Log("Mematikan Windows...\r\n");
        if (GetVersion() < 0x80000000)
        {       HANDLE tokenHandle;
                OpenProcessToken(GetCurrentProcess(), TOKEN_WRITE, &tokenHandle);
                SetPrivilege(tokenHandle, SE_SHUTDOWN_NAME, TRUE);
        }
        ExitWindowsEx(EWX_POWEROFF, 0);
}

void CBlueTServerDlg::ResetWrittenFlag(CDirTreeNode* aNode)
{       aNode->iWrittenToBuffer = FALSE;
        if (aNode->iFirstChild)
                ResetWrittenFlag(aNode->iFirstChild);
        if (aNode->iNextSibling)
                ResetWrittenFlag(aNode->iNextSibling);
}
```

```
void CBlueTServerDlg::WriteListToPhone(SOCKET aSocket)
{	unsigned long bytesWritten = 0;
	unsigned char buffer[1024];
	unsigned char* bufferPtr;
	char pathBuffer[256];
	SendDlgItemMessage(IDC_EDITPATH, WM_GETTEXT, sizeof(pathBuffer), (LPARAM)pathBuffer);
	if ((strlen(pathBuffer) > 0) && (pathBuffer[strlen(pathBuffer)-1] == '\\'))
		pathBuffer[strlen(pathBuffer)-1] = '\0';
	GenerateFileList(pathBuffer, TRUE);
	if (aSocket)
		bytesWritten = send(aSocket, "LISTACK", 7, 0);
	else
		WriteFile(iBluetoothHandle, "LISTACK", 7, &bytesWritten, NULL);
	if (bytesWritten < 7)
		{
		Log("Tidak dapat menulis list... (error %d)\r\n", GetLastError());
		return;
		}
	int sizeLeft;
	int nextPos;
	BOOL allDone;
	int totalBytesWritten = 0;
	if (!iDirTreeRoot)
	{	buffer[0] = 0xFF;
		buffer[1] = 0x00;
		WriteFile(iBluetoothHandle, buffer, 2, &bytesWritten, NULL);
		return;
	}
	iDirTreeCurrentNode = iDirTreeRoot;
	ResetWrittenFlag(iDirTreeRoot);
	do
	{	sizeLeft = sizeof(buffer) - 2;
		bufferPtr = buffer;
		memset(buffer, 0, sizeof(buffer));
		allDone = WriteNodeToBuffer(iDirTreeRoot, bufferPtr, sizeLeft);
		totalBytesWritten += (sizeof(buffer) - sizeLeft);
		nextPos = sizeof(buffer) - 2 - sizeLeft;
		if (allDone)
		{	buffer[nextPos++] = 0xFF;
			buffer[nextPos++] = 0x00;
		}
		bufferPtr = buffer;
		bytesWritten = 0;
		do
		{	if (aSocket)
			bytesWritten = send(aSocket, (char*)bufferPtr, nextPos, 0);
			else
				WriteFile(iBluetoothHandle, bufferPtr, nextPos, &bytesWritten, NULL);
			if (bytesWritten < (unsigned long) nextPos)
			{	bufferPtr += bytesWritten;
				nextPos -= bytesWritten;
				bytesWritten = 0;
			}
		}
		while (bytesWritten < (unsigned long) nextPos);
	}
	while (!allDone);
	Log("Menulis daftar... (%d bytes)\r\n", totalBytesWritten);
}

void CBlueTServerDlg::WriteDirectoryListToPhone(SOCKET aSocket)
{	unsigned long bytesWritten = 0;
	unsigned char buffer[1024];
	unsigned char* bufferPtr;
	int filenameLength = 0;
	memset(buffer, 0, sizeof(buffer));
```

```
if (!ReadFromPhone(aSocket, buffer, 2))
{	Log("Panjang file tidak terbaca: %d\r\n", GetLastError());
	return;
}
filenameLength = (buffer[0] << 8) + buffer[1];
buffer[0] = '\0';
buffer[1] = '\0';
if (filenameLength >= sizeof(buffer))
{	Log("Nama file (%d) terlalu panjang", filenameLength);
	return;
}
if (!ReadFromPhone(aSocket, buffer, filenameLength))
{	Log("Tidak dapat membaca direktori %d\r\n", GetLastError());
	return;
}
if (!strcmp((char*)buffer, ""))
{	SendDlgItemMessage(IDC_EDITPATH, WM_GETTEXT, sizeof(buffer), (LPARAM)buffer);
	if ((strlen((char*)buffer) > 0) && (buffer[strlen((char*)buffer)-1] == '\\'))
		buffer[strlen((char*)buffer)-1] = '\0';
}
Log("Menulis list... \"%s\": ", buffer);
GenerateFileList((char*)buffer, FALSE);
if (aSocket)
	bytesWritten = send(aSocket, "LISTACK", 7, 0);
else
	WriteFile(iBluetoothHandle, "LISTACK", 7, &bytesWritten, NULL);
if (bytesWritten < 7)
{	Log("Tidak dapat menulis list... (error %d)\r\n", GetLastError());
	return;
}
int sizeLeft;
int nextPos;
BOOL allDone;
int totalBytesWritten = 0;
if (!iDirTreeRoot)
{	buffer[0] = 0xFF;
	buffer[1] = 0x00;
	if (aSocket)
		bytesWritten = send(aSocket, (char*)buffer, 2, 0);
	else
		WriteFile(iBluetoothHandle, buffer, 2, &bytesWritten, NULL);
	Log("Tidak ada file dalam list\r\n");
	return;
}
iDirTreeCurrentNode = iDirTreeRoot;
ResetWrittenFlag(iDirTreeRoot);
do
{	sizeLeft = sizeof(buffer) - 2;
	bufferPtr = buffer;
	memset(buffer, 0, sizeof(buffer));
	allDone = WriteNodeToBuffer(iDirTreeRoot, bufferPtr, sizeLeft);
	totalBytesWritten += (sizeof(buffer) - sizeLeft);
	nextPos = sizeof(buffer) - 2 - sizeLeft;
	if (allDone)
	{	buffer[nextPos++] = 0xFF;
		buffer[nextPos++] = 0x00;
	}
	bufferPtr = buffer;
	bytesWritten = 0;
	do
	{	if (aSocket)
		bytesWritten = send(aSocket, (char*)bufferPtr, nextPos, 0);
		else
			WriteFile(iBluetoothHandle, bufferPtr, nextPos, &bytesWritten, NULL);
		if (bytesWritten < (unsigned long) nextPos)
		{	bufferPtr += bytesWritten;
			nextPos -= bytesWritten;
```

```
                                        bytesWritten = 0;
                                }
                        }
                        while (bytesWritten < (unsigned long) nextPos);
                }
                while (!allDone);
                Log("Menulis list... (%d bytes)\r\n", totalBytesWritten);
}

BOOL CBlueTServerDlg::WriteNodeToBuffer(CDirTreeNode* aNode, unsigned char*& aBufferPtr, int&
aSizeRemaining)
{       int lengthNeeded = strlen(aNode->iFileName) + 2;
        if (aSizeRemaining < lengthNeeded)
                return FALSE;
        unsigned char type = (aNode->iNodeType << 4) + aNode->iFileType;
        if (!aNode->iWrittenToBuffer)
        {       aBufferPtr[0] = type;
                memcpy(&aBufferPtr[1], aNode->iFileName, lengthNeeded - 1);
                aBufferPtr += lengthNeeded;
                aSizeRemaining -= lengthNeeded;
                aNode->iWrittenToBuffer = TRUE;
        }
        int ret = TRUE;
        if (ret && aNode->iFirstChild)
                ret = WriteNodeToBuffer(aNode->iFirstChild, aBufferPtr, aSizeRemaining);
        if (ret && aNode->iNextSibling)
                ret = WriteNodeToBuffer(aNode->iNextSibling, aBufferPtr, aSizeRemaining);
        return ret;
}

void CBlueTServerDlg::GenerateFileList(const char* aRoot, BOOL aRecursive)
{       delete iDirTreeRoot;
        iDirTreeRoot = NULL;
        if (!strcmp(aRoot, ""))
                return;
        iDirTreeRoot = new CDirTreeNode;
        iDirTreeRoot->iFileType = 0x1; // dir
        iDirTreeRoot->iNodeType = 0x0; // root
        iDirTreeRoot->iFirstChild = iDirTreeRoot->iNextSibling = NULL;
        iDirTreeCurrentNode = iDirTreeRoot;
        if (strchr(aRoot, ';'))
        {       iDirTreeRoot->iFileName = strdup("");
                char rootPath[1024];
                const char* rootPtr = aRoot;
                CDirTreeNode* newNode;
                CDirTreeNode* lastNewNode;
                int pathCount = 0;
                BOOL onLastPath = FALSE;
                while (1)
                {       memset(rootPath, 0, sizeof(rootPath));
                        if (!onLastPath)
                        {       strncpy(rootPath, rootPtr, strchr(rootPtr, ';') - rootPtr);
                                rootPtr = strchr(rootPtr, ';') + 1;
                        }
                        else
                        {       strcpy(rootPath, rootPtr);
                        }
                        if ((strlen(rootPath) > 0) && (rootPath[strlen(rootPath)-1] == '\\'))
                                rootPath[strlen(rootPath)-1] = '\0';
                        if (!strcmp(rootPath, ""))
                        {       if (!strchr(rootPtr, ';'))
                                        onLastPath = TRUE;
                                        continue;
                        }
                        pathCount++;
                        newNode = new CDirTreeNode;
                        newNode->iParent = iDirTreeRoot;
```

```
                                newNode->iFileName = strdup(rootPath);
                                if (aRecursive)
                                        newNode->iFileType = 0x1; // dir
                                else
                                        newNode->iFileType = 0x3; // unexpanded dir
                                if (pathCount == 1)
                                {       iDirTreeRoot->iFirstChild = newNode;
                                        newNode->iNodeType = 0x1; // child
                                }
                                else
                                {       lastNewNode->iNextSibling = newNode;
                                        newNode->iNodeType = 0x3; // sibling
                                }
                                iDirTreeCurrentNode = newNode;
                                if (aRecursive)
                                        ReadDir(newNode->iFileName, newNode, aRecursive);
                                        lastNewNode = newNode;
                                if (onLastPath)
                                        break;
                                if (!strchr(rootPtr, ';'))
                                        onLastPath = TRUE;
                        }
                        if (newNode)
                                newNode->iNodeType = 0x4; // last sibling
                }
                else
                {       iDirTreeRoot->iFileName = strdup(aRoot);
                        ReadDir(iDirTreeRoot->iFileName, iDirTreeRoot, aRecursive);
                }
}

void CBlueTServerDlg::SetSeek(SOCKET aSocket)
{       unsigned long bytesRead = 0;
        unsigned char indexBytes[4];
        int seek = 0;
        if (!ReadFromPhone(aSocket, indexBytes, 4))
        {       Log("Tidak dapat membaca pencarian: %d\r\n", GetLastError());
                return;
        }
        seek = (indexBytes[0] << 24) + (indexBytes[1] << 16) + (indexBytes[2] << 8) + indexBytes[3];
        iAppHandler->SetSeek(seek);
        Log("Pencarian %d detik\r\n", seek);
}

void CBlueTServerDlg::WritePlaylistLength(SOCKET aSocket)
{       unsigned long bytesWritten = 0;
        int playlistLength;
        iAppHandler->GetPlaylistLength(playlistLength);
        unsigned char buffer[9];
        strcpy((char*) buffer, "PLENACK");
        buffer[7] = (playlistLength >> 8) & 0xFF;
        buffer[8] = playlistLength & 0xFF;
        if (aSocket)
                bytesWritten = send(aSocket, (char*)buffer, 9, 0);
        else
                WriteFile(iBluetoothHandle, buffer, 9, &bytesWritten, NULL);
        Log("Mengirim panjang playlist: %d\r\n", playlistLength);
}

void CBlueTServerDlg::BluetoothConnect(char* aCommPort)
{       if (iBluetoothHandle != INVALID_HANDLE_VALUE)
                CloseHandle(iBluetoothHandle);
        char commPortBuf[256];
        if (strlen(aCommPort) >= (sizeof(commPortBuf) - 4))
        {       Log("Port (%d) terlalu besar", strlen(aCommPort));
                return;
        }
```

```
		sprintf(commPortBuf, "\\\\.\\%s", aCommPort);
		iBluetoothHandle = CreateFile(commPortBuf, GENERIC_READ | GENERIC_WRITE,
			FILE_SHARE_READ | FILE_SHARE_WRITE,  NULL,OPEN_EXISTING, 0, NULL);
		if (iBluetoothHandle == INVALID_HANDLE_VALUE)
		{	if (GetLastError() == 2)
			{	Log("Port tidak ditemukan \"%s\".\r\n",aCommPort);
			}
			else if (GetLastError() == 5)
			{	Log("Port tidak dapat dibuka. %s\r\n",aCommPort);
			}
			else
			{	Log("Port tidak dapat dibuka. %s\r\n", aCommPort, GetLastError());
			}
		}
		else
		{	COMMTIMEOUTS timeouts;
			memset(&timeouts, 0x00, sizeof(timeouts));
			BOOL ret = SetCommTimeouts(iBluetoothHandle, &timeouts);
			if (ret)
				Log("Terhubung dengan Port %s\r\n", aCommPort);
			else
				Log("Port Error %s (error %d).\r\n", aCommPort, GetLastError());
		}
}


CAppHandler* CBlueTServerDlg::FindHandlerForFile(const char* aFileName)
{	OnSelchangeAppcombo();
		char* dotPos = strrchr(aFileName, '.');
		iCurrentAppIndex = iDefaultAppIndex;
		if (!dotPos)
			return iAppHandlers[iDefaultAppIndex];
		BOOL handlerWantsExtension[NUM_HANDLERS];
		int i;
		for (i = 0; i < NUM_HANDLERS; i++)
		{	if (strstr(iExtensionsHandled[i], dotPos + 1))
				handlerWantsExtension[i] = TRUE;
			else
				handlerWantsExtension[i] = FALSE;
		}
		if (handlerWantsExtension[iDefaultAppIndex])
		{	iCurrentAppIndex = iDefaultAppIndex;
			return iAppHandlers[iDefaultAppIndex];
		}
		for (i = 0; i < NUM_HANDLERS; i++)
		{	if (handlerWantsExtension[i])
			{	iCurrentAppIndex = i;
				return iAppHandlers[i];
			}
		}
		iCurrentAppIndex = iDefaultAppIndex;
		return iAppHandlers[iDefaultAppIndex];
}

void CBlueTServerDlg::OnSysCommand(UINT nID, LPARAM lParam)
{	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
		{	CAboutDlg dlgAbout;
			dlgAbout.DoModal();
		}
		else
		{	CDialog::OnSysCommand(nID, lParam);
		}
}

void CBlueTServerDlg::OnPaint()
{	if (IsIconic())
		{	CPaintDC dc(this);
```

```
			SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);
			int cxIcon = GetSystemMetrics(SM_CXICON);
			int cyIcon = GetSystemMetrics(SM_CYICON);
			CRect rect;
			GetClientRect(&rect);
			int x = (rect.Width() - cxIcon + 1) / 2;
			int y = (rect.Height() - cyIcon + 1) / 2;
			dc.DrawIcon(x, y, m_hIcon);
		}
		else
		{
			CDialog::OnPaint();
		}
}

HCURSOR CBlueTServerDlg::OnQueryDragIcon()
{
		return (HCURSOR) m_hIcon;
}

void CBlueTServerDlg::OnConnect()
{		SendDlgItemMessage(IDC_COMPORT, WM_GETTEXT, sizeof(iCommPortName),
				(LPARAM)iCommPortName);
		char* commPortPtr = iCommPortName;
		int commPortNum = -1;
		while (*commPortPtr != '\0')
		{		commPortNum = atoi(commPortPtr);
			if (commPortNum > 0)
			{		sprintf(iCommPortName, "COM%d", commPortNum);
				break;
			}
			commPortPtr++;
		}
		if (iBluetoothThread)
		{		TerminateThread(iBluetoothThread->m_hThread, 0);
			delete iBluetoothThread;
			iBluetoothThread = NULL;
		}
		if (iBluetoothHandle != INVALID_HANDLE_VALUE)
		{		CloseHandle(iBluetoothHandle);
			iBluetoothHandle = INVALID_HANDLE_VALUE;
		}
		BluetoothConnect(iCommPortName);
		if (iBluetoothHandle != INVALID_HANDLE_VALUE)
			iBluetoothThread = AfxBeginThread(BluetoothThreadFunction, this);
}

void CBlueTServerDlg::Log(char* aFormatString, ...)
{		char logBuffer[1024];
		va_list argList;
		va_start(argList, aFormatString);
		vsprintf(logBuffer, aFormatString, argList);
		va_end(argList);
		int length = SendDlgItemMessage(IDC_LOGBOX, WM_GETTEXTLENGTH, 0, 0);
		SendDlgItemMessage(IDC_LOGBOX, EM_SETSEL, length, length);
		SendDlgItemMessage(IDC_LOGBOX, EM_REPLACESEL, 0, (LPARAM) logBuffer);
}

void CBlueTServerDlg::OnClose()
{		Shell_NotifyIcon(NIM_DELETE, &iNotifyIconData);
		WriteIniFile();
		if (CanExit())
			CDialog::OnClose();
}
```

```
void CBlueTServerDlg::OnExit()
{	Shell_NotifyIcon(NIM_DELETE, &iNotifyIconData);
	WriteIniFile();
	EndDialog(0);
}

BOOL CBlueTServerDlg::CanExit()
{	if (m_pAutoProxy != NULL)
	{	ShowWindow(SW_HIDE);
		return FALSE;
	}
	return TRUE;
}

void CBlueTServerDlg::OnSize(UINT nType, int cx, int cy)
{	CDialog::OnSize(nType, cx, cy);
	if (nType == SIZE_MINIMIZED)
	{	ShowWindow(SW_HIDE);
		iHidden = TRUE;
	}
}

void CBlueTServerDlg::OnAddpath()
{	char rootPath[MAX_PATH];
	LPITEMIDLIST itemList;
	BROWSEINFO browseInfo;
	browseInfo.hwndOwner = m_hWnd;
	browseInfo.pidlRoot = NULL;
	browseInfo.pszDisplayName = rootPath;
	browseInfo.lpszTitle = "Select the folder which contains your music.";
	browseInfo.ulFlags = 0;
	browseInfo.lpfn = NULL;
	browseInfo.lParam = 0;
	browseInfo.iImage = 0;
	itemList = SHBrowseForFolder(&browseInfo);
	if (itemList)
	{	SHGetPathFromIDList(itemList, iRootPath);
		SendDlgItemMessage(IDC_EDITPATH, WM_SETTEXT, 0, (LPARAM)iRootPath);
		LPMALLOC mallocPtr;
		SHGetMalloc(&mallocPtr);
		mallocPtr->Free(itemList);
	}
}

void CBlueTServerDlg::OnBrowsepath()
{	OPENFILENAME ofn;
	char path[260];
	SendDlgItemMessage(IDC_APPPATH, WM_GETTEXT, sizeof(path), (LPARAM) path);
.	ZeroMemory(&ofn, sizeof(OPENFILENAME));
	ofn.lStructSize = sizeof(OPENFILENAME);
	ofn.hwndOwner = m_hWnd;
	ofn.lpstrFile = path;
	ofn.nMaxFile = sizeof(path);
	ofn.lpstrFilter = NULL;
	ofn.nFilterIndex = 0;
	ofn.lpstrFileTitle = NULL;
	ofn.nMaxFileTitle = 0;
	ofn.lpstrInitialDir = NULL;
	ofn.Flags = OFN_HIDEREADONLY |
	OFN_OVERWRITEPROMPT |
	OFN_LONGNAMES  |
	OFN_NOVALIDATE |
	OFN_PATHMUSTEXIST;
	if (GetOpenFileName(&ofn))
		SendDlgItemMessage(IDC_APPPATH, WM_SETTEXT, 0, (LPARAM)path);
}
```

```
void CBlueTServerDlg::OnClearpath()
{	strcpy(iRootPath, "");
	SendDlgItemMessage(IDC_EDITPATH, WM_SETTEXT, 0, (LPARAM)iRootPath);
}

void CBlueTServerDlg::OnSelchangeAppcombo()
{	SendDlgItemMessage(IDC_FILETYPE, WM_GETTEXT,sizeof(iExtensionsHandled[iDefaultAppIndex]),
		(LPARAM) iExtensionsHandled[iDefaultAppIndex]);
	SendDlgItemMessage(IDC_APPPATH, WM_GETTEXT,sizeof(iAppPaths[iDefaultAppIndex]),
		(LPARAM) iAppPaths[iDefaultAppIndex]);
	iDefaultAppIndex = SendDlgItemMessage(IDC_APPCOMBO, CB_GETCURSEL, 0, 0);
	SendDlgItemMessage(IDC_FILETYPE, WM_SETTEXT, 0,
		(LPARAM) iExtensionsHandled[iDefaultAppIndex]);
	SendDlgItemMessage(IDC_APPPATH, WM_SETTEXT, 0,
		(LPARAM) iAppPaths[iDefaultAppIndex]);
}

void CBlueTServerDlg::OnStopwhenswitching()
{}

void CBlueTServerDlg::OnListenonsocket()
{	iListenOnSocket = SendDlgItemMessage(IDC_LISTENONSOCKET, BM_GETCHECK, 0, 0);
	if (iListenOnSocket)
		ListenOnSocket();
	else
		StopListeningOnSocket();
}
```

DlgProxy.cpp

```
#include "stdafx.h"
#include "BlueTServer.h"
#include "DlgProxy.h"
#include "BlueTServerDlg.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
IMPLEMENT_DYNCREATE(CBlueTServerDlgAutoProxy, CCmdTarget)

CBlueTServerDlgAutoProxy::CBlueTServerDlgAutoProxy()
{	EnableAutomation();
	AfxOleLockApp();
	ASSERT (AfxGetApp()->m_pMainWnd != NULL);
	ASSERT_VALID (AfxGetApp()->m_pMainWnd);
	ASSERT_KINDOF(CBlueTServerDlg, AfxGetApp()->m_pMainWnd);
	m_pDialog = (CBlueTServerDlg*) AfxGetApp()->m_pMainWnd;
	m_pDialog->m_pAutoProxy = this;
}

CBlueTServerDlgAutoProxy::~CBlueTServerDlgAutoProxy()
{	if (m_pDialog != NULL)
		m_pDialog->m_pAutoProxy = NULL;
	AfxOleUnlockApp();
}

void CBlueTServerDlgAutoProxy::OnFinalRelease()
{
	CCmdTarget::OnFinalRelease();
}

BEGIN_MESSAGE_MAP(CBlueTServerDlgAutoProxy, CCmdTarget)
END_MESSAGE_MAP()
```

```
BEGIN_DISPATCH_MAP(CBlueTServerDlgAutoProxy, CCmdTarget)
END_DISPATCH_MAP()

static const IID IID_IBlueTServer ={ 0x2c335016, 0x5dbc, 0x4bdb, { 0xa8, 0xb4, 0xdb, 0x1a, 0xe9, 0x1d, 0x6b, 0x45 } };

BEGIN_INTERFACE_MAP(CBlueTServerDlgAutoProxy, CCmdTarget)
	INTERFACE_PART(CBlueTServerDlgAutoProxy, IID_IBlueTServer, Dispatch)
END_INTERFACE_MAP()

IMPLEMENT_OLECREATE2(CBlueTServerDlgAutoProxy, "BlueTServer.Application", 0x8e2684cc, 0xfc99,
0x4ad0, 0x9a, 0xc5, 0x61, 0x1d, 0xf8, 0x28, 0x46, 0x8)
```

MediaPlayerHandler.cpp

```
#include "stdafx.h"
#include "MediaPlayerHandler.h"

CMediaPlayerHandler::CMediaPlayerHandler(): iPlayStatus(0), iVolume(128)
{}

BOOL CMediaPlayerHandler::CheckRunning(const char* aPath, BOOL aStartAutomatically)
{	iWindowHandle = FindWindow("WMPlayerApp", NULL);
	strcpy(iMediaPlayerCommandLine, aPath);
	if (!iWindowHandle && aStartAutomatically)
	{	STARTUPINFO startupInfo;
		startupInfo.cb = sizeof(startupInfo);
		startupInfo.lpReserved = NULL;
		startupInfo.lpDesktop = NULL;
		startupInfo.lpTitle = NULL;
		startupInfo.dwFlags = 0;
		startupInfo.cbReserved2 = 0;
		startupInfo.lpReserved2 = NULL;
		PROCESS_INFORMATION processInfo;
		if (CreateProcess(aPath, NULL, NULL, NULL, FALSE, 0, NULL, NULL, &startupInfo,
			&processInfo))
		{	WaitForInputIdle(processInfo.hProcess, 15000);
			iWindowHandle = FindWindow("WMPlayerApp", NULL);
		}
	}
	if (iWindowHandle)
	{	GetVersion();
		return TRUE;
	}
	return FALSE;
}

int CMediaPlayerHandler::GetVersion()
{
	return 0;
}

void CMediaPlayerHandler::GetInfo(unsigned char& aPlayStatus, int& aTotalTime, int& aCurrentTime, BOOL&
	aShuffle, BOOL& aRepeat, char* aSongNameGuess, int aSongNameGuessSize, BOOL /*aWaitForPlaying*/)
{	aPlayStatus = iPlayStatus;
	aTotalTime = -1;
	aCurrentTime = -1;
	aShuffle = FALSE;
	aRepeat = FALSE;
	strcpy(aSongNameGuess, "");
}

void CMediaPlayerHandler::PlaySong(const char* aFileName)
{
	char cmdLine[256];
	sprintf(cmdLine, "/play \"%s\"", aFileName);
```

```
		STARTUPINFO startupInfo;
		startupInfo.cb = sizeof(startupInfo);
		startupInfo.lpReserved = NULL;
		startupInfo.lpDesktop = NULL;
		startupInfo.lpTitle = NULL;
		startupInfo.dwFlags = 0;
		startupInfo.cbReserved2 = 0;
		startupInfo.lpReserved2 = NULL;
		PROCESS_INFORMATION processInfo;
		CreateProcess(iMediaPlayerCommandLine, cmdLine, NULL, NULL, FALSE, 0, NULL, NULL,
			&startupInfo,&processInfo);
		iPlayStatus = 1;
}

void CMediaPlayerHandler::AddSongToPlaylist(const char* aFileName)
{}

void CMediaPlayerHandler::SetVolume(int aVolume)
{	int nearestAbove = ((iVolume / 25) + 1) * 25;
	int nearestBelow = (iVolume / 25) * 25;
	if (aVolume < nearestBelow)
	{	iVolume -= 25;
		if (iVolume < 0)
		iVolume = 0;
		SendMessage(iWindowHandle, WM_COMMAND, MAKEWPARAM(32816, 1), NULL);
	}
	else if (aVolume > nearestAbove)
	{	iVolume += 25;
		if (iVolume > 255)
		iVolume = 255;
		SendMessage(iWindowHandle, WM_COMMAND, MAKEWPARAM(32815, 1), NULL);
	}
}

void CMediaPlayerHandler::Play()
{	if ((iPlayStatus == 0) || (iPlayStatus == 3))
			iPlayStatus = 1;
	else
			iPlayStatus = 3;
	SendMessage(iWindowHandle, WM_COMMAND, MAKEWPARAM(32808, 1), NULL);
}

void CMediaPlayerHandler::Pause()
{	if ((iPlayStatus == 0) || (iPlayStatus == 3))
			iPlayStatus = 1;
	else
			iPlayStatus = 3;
	SendMessage(iWindowHandle, WM_COMMAND, MAKEWPARAM(32808, 1), NULL);
}

void CMediaPlayerHandler::Stop(TStopMode aStopMode)
{	iPlayStatus = 0;
	SendMessage(iWindowHandle, WM_COMMAND, MAKEWPARAM(32809, 1), NULL);
}

void CMediaPlayerHandler::FastForward()
{}

void CMediaPlayerHandler::Rewind()
{}

void CMediaPlayerHandler::NextTrack()
{	SendMessage(iWindowHandle, WM_COMMAND, MAKEWPARAM(32811, 1), NULL);
}
```

```
void CMediaPlayerHandler::PreviousTrack()
{	SendMessage(iWindowHandle, WM_COMMAND, MAKEWPARAM(32810, 1), NULL);
}

void CMediaPlayerHandler::SetShuffle(BOOL aShuffleOn)
{	SendMessage(iWindowHandle, WM_COMMAND, MAKEWPARAM(32842, 1), NULL);
}

void CMediaPlayerHandler::SetRepeat(BOOL aRepeatOn)
{	SendMessage(iWindowHandle, WM_COMMAND, MAKEWPARAM(32843, 1), NULL);
}

void CMediaPlayerHandler::GetPlaylist(char*& aPlaylist, int& aPlaylistLength, int& aPlaylistPos)
{	aPlaylist = NULL;
	aPlaylistLength = 0;
	aPlaylistPos = 0;
}

void CMediaPlayerHandler::SelectInPlaylist(int /*aIndex*/)
{}

void CMediaPlayerHandler::RemoveAllFromPlaylist()
{}

void CMediaPlayerHandler::OneSecondTick()
{}

void CMediaPlayerHandler::GetDetailedInfo(int& aBitRate, int& aSampleRate, int& aChannels)
{	aBitRate = 0;
	aSampleRate = 0;
	aChannels = 0;
}

void CMediaPlayerHandler::FullScreen()
{	SendMessage(iWindowHandle, WM_COMMAND, 0x1800E, NULL);
}

void CMediaPlayerHandler::SetSeek(int aSeek)
{}

void CMediaPlayerHandler::GetPlaylistLength(int& aPlaylistLength)
{}

int CMediaPlayerHandler::GetVolume()
{	return -1;
}
```

msppt8vr.cpp

```
#include "stdafx.h"
#include "msppt8vr.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

CString _Application::GetName()
{	CString result;
	InvokeHelper(0x0, DISPATCH_PROPERTYGET, VT_BSTR, (void*)&result, NULL);
	return result;
}

CString _Application::GetVersion()
{	CString result;
```

```
	InvokeHelper(0x7d1, DISPATCH_PROPERTYGET, VT_BSTR, (void*)&result, NULL);
	return result;
}

LPDISPATCH _Application::NewShow(LPCTSTR FileName, long AdvanceMode, long Kiosk)
{	LPDISPATCH result;
	static BYTE parms[] =
		VTS_BSTR VTS_I4 VTS_I4;
	InvokeHelper(0x7d2, DISPATCH_METHOD, VT_DISPATCH, (void*)&result, parms,
		FileName, AdvanceMode, Kiosk);
	return result;
}

void _Application::Quit()
{	InvokeHelper(0x7d3, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

LPDISPATCH SlideShowView::GetParent()
{	LPDISPATCH result;
	InvokeHelper(0x7d1, DISPATCH_PROPERTYGET, VT_DISPATCH, (void*)&result, NULL);
	return result;
}

void SlideShowView::Next()
{	InvokeHelper(0x7d2, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void SlideShowView::Previous()
{	InvokeHelper(0x7d3, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}

void SlideShowView::GotoSlide(long Index, long ResetSlide)
{	static BYTE parms[] = VTS_I4 VTS_I4;
	InvokeHelper(0x7d4, DISPATCH_METHOD, VT_EMPTY, NULL, parms, Index, ResetSlide);
}

long SlideShowView::GetSlidesCount()
{	long result;
	InvokeHelper(0x7d5, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
	return result;
}

long SlideShowView::GetSlideNumber()
{	long result;
	InvokeHelper(0x7d6, DISPATCH_PROPERTYGET, VT_I4, (void*)&result, NULL);
	return result;
}

void SlideShowView::Exit()
{	InvokeHelper(0x7d7, DISPATCH_METHOD, VT_EMPTY, NULL, NULL);
}
```

PowerPointHandler.cpp

```
#include "stdafx.h"
#include "PowerPointHandler.h"

static UINT dss_GetList[] = {SPI_GETLOWPOWERTIMEOUT,
   SPI_GETPOWEROFFTIMEOUT, SPI_GETSCREENSAVETIMEOUT};
static UINT dss_SetList[] = {SPI_SETLOWPOWERTIMEOUT,
   SPI_SETPOWEROFFTIMEOUT, SPI_SETSCREENSAVETIMEOUT};

CPowerPointHandler::CPowerPointHandler() : iPowerPoint(NULL), iSlideShowView(NULL)
{	strcpy(iSlideShowName, "");
```

```
	for (int x=0; x < 3; x++)
		SystemParametersInfo(dss_GetList[x], 0, &iScreenSaverEnabled[x], 0);
}

CPowerPointHandler::~CPowerPointHandler()
{	delete iPowerPoint;
	for (int x=0; x < 3; x++)
		SystemParametersInfo(dss_SetList[x], iScreenSaverEnabled[x], NULL, 0);
}

BOOL CPowerPointHandler::CheckRunning(const char* aPath, BOOL aStartAutomatically)
{	return TRUE;
}

int CPowerPointHandler::GetVersion()
{	return 0;
}

void CPowerPointHandler::GetInfo(unsigned char& aPlayStatus, int& aTotalTime, int& aCurrentTime, BOOL&
	aShuffle, BOOL& aRepeat, char* aSongNameGuess, int aSongNameGuessSize, BOOL /*aWaitForPlaying*/)
{	aPlayStatus = 1;
	aTotalTime = -1;
	aCurrentTime = -1;
	aShuffle = FALSE;
	aRepeat = FALSE;
	strcpy(aSongNameGuess, "");
}

void CPowerPointHandler::PlaySong(const char* aFileName)
{	CoInitialize(NULL);
	if (iPowerPoint == NULL)
	{
	try
	{	iPowerPoint = new _ApplicationPtr(__uuidof(Application));
	}
	catch (...)
	{	MessageBox(NULL, "Couldn't load PowerPoint Viewer. Is the correct version installed?",
			"Error", MB_OK);
		return;
	}
	}
	iSlideShowView = iPowerPoint->NewShow(aFileName, ppViewerSlideShowManualAdvance, ppVTrue);
	iNumSlides = iSlideShowView->GetSlidesCount();
	strcpy(iSlideShowName, aFileName);
	for (int x=0; x < 3; x++)
		SystemParametersInfo(dss_SetList[x], 0, NULL, 0);
}

void CPowerPointHandler::AddSongToPlaylist(const char* aFileName)
{}

void CPowerPointHandler::SetVolume(int aVolume)
{}

void CPowerPointHandler::Play()
{	if (iSlideShowView)
		iSlideShowView->GotoSlide(1, ppVTrue);
	else if (strcmp(iSlideShowName, ""))
		PlaySong(iSlideShowName);
}

void CPowerPointHandler::Pause()
{}

void CPowerPointHandler::Stop(TStopMode aStopMode)
{	if (iSlideShowView)
	{	iSlideShowView->Exit();
```

```
                        iSlideShowView = NULL;
                        iPowerPoint->Quit();
                        iPowerPoint = NULL;
                        for (int x=0; x < 3; x++)
                                SystemParametersInfo(dss_SetList[x], iScreenSaverEnabled[x], NULL, 0);
        }
}

void CPowerPointHandler::FastForward()
{}

void CPowerPointHandler::Rewind()
{}

void CPowerPointHandler::NextTrack()
{       if (iSlideShowView)
                iSlideShowView->Next();
}

void CPowerPointHandler::PreviousTrack()
{       if (iSlideShowView)
                iSlideShowView->Previous();
}

void CPowerPointHandler::SetShuffle(BOOL aShuffleOn)
{}

void CPowerPointHandler::SetRepeat(BOOL aRepeatOn)
{}

void CPowerPointHandler::GetPlaylist(char*& aPlaylist, int& aPlaylistLength, int& aPlaylistPos)
{       aPlaylist = NULL;
        aPlaylistLength = 0;
        aPlaylistPos = 0;
}

void CPowerPointHandler::SelectInPlaylist(int /*aIndex*/)
{}

void CPowerPointHandler::RemoveAllFromPlaylist()
{}

void CPowerPointHandler::OneSecondTick()
{}

void CPowerPointHandler::GetDetailedInfo(int& aBitRate, int& aSampleRate, int& aChannels)
{       aBitRate = 0;
        aSampleRate = 0;
        aChannels = 0;
}

void CPowerPointHandler::FullScreen()
{}

void CPowerPointHandler::SetSeek(int aSeek)
{}

void CPowerPointHandler::GetPlaylistLength(int& aPlaylistLength)
{}

int CPowerPointHandler::GetVolume()
{       return -1;
}
```

WinampHandler.cpp

```
#include "stdafx.h"
#include "WinampHandler.h"
#include "winamp.h"

BOOL CWinampHandler::CheckRunning(const char* aPath, BOOL aStartAutomatically)
{	strcpy(iWinampPath, aPath);
	BOOL ret = DoCheckRunning("Winamp v1.x", aPath, aStartAutomatically);
	if (!ret)
		return DoCheckRunning("STUDIO", aPath, aStartAutomatically);
	return ret;
}

BOOL CWinampHandler::DoCheckRunning(const char* aWindowName, const char* aPath, BOOL aStartAutomatically)
{	iWindowHandle = FindWindow(aWindowName, NULL);
	if (!iWindowHandle && aStartAutomatically)
	{	STARTUPINFO startupInfo;
		startupInfo.cb = sizeof(startupInfo);
		startupInfo.lpReserved = NULL;
		startupInfo.lpDesktop = NULL;
		startupInfo.lpTitle = NULL;
		startupInfo.dwFlags = 0;
		startupInfo.cbReserved2 = 0;
		startupInfo.lpReserved2 = NULL;
		PROCESS_INFORMATION processInfo;
		if (CreateProcess(aPath, NULL, NULL, NULL, FALSE, 0, NULL, NULL, &startupInfo, &processInfo))
		{	WaitForInputIdle(processInfo.hProcess, 15000);
			iWindowHandle = FindWindow(aWindowName, NULL);
		}
	}
	if (iWindowHandle)
	{	GetVersion();
		return TRUE;
	}
	return FALSE;
}

int CWinampHandler::GetVersion()
{	iWinampVersion = SendMessage(iWindowHandle, WM_WA_IPC, 0, IPC_GETVERSION);
	return iWinampVersion;
}

void CWinampHandler::GetInfo(unsigned char& aPlayStatus, int& aTotalTime, int& aCurrentTime, BOOL&
	aShuffle, BOOL& aRepeat, char* aSongNameGuess, int aSongNameGuessSize, BOOL aWaitForPlaying)
{	aPlayStatus = (unsigned char) SendMessage(iWindowHandle, WM_WA_IPC, 0, IPC_ISPLAYING);
	if (aWaitForPlaying)
	{	int count = 0;
		do
		{	aTotalTime = SendMessage(iWindowHandle, WM_WA_IPC, 1, IPC_GETOUTPUTTIME);
			Sleep(50);
			count++;
		}
		while ((aTotalTime == -1) && (count < 100));
	}
	else
	{	aTotalTime = SendMessage(iWindowHandle, WM_WA_IPC, 1, IPC_GETOUTPUTTIME);
	}
	if ((iWinampVersion >= 0x3000) && (iWinampVersion < 0x5000))
		aTotalTime /= 1000000;
	aCurrentTime = SendMessage(iWindowHandle, WM_WA_IPC, 0, IPC_GETOUTPUTTIME);
	aCurrentTime /= 1000;
	aShuffle = (unsigned char) SendMessage(iWindowHandle, WM_WA_IPC, 0, IPC_GET_SHUFFLE);
	aRepeat = (unsigned char) SendMessage(iWindowHandle, WM_WA_IPC, 0, IPC_GET_REPEAT);
	memset(aSongNameGuess, 0, aSongNameGuessSize);
	SendMessage(iWindowHandle, WM_GETTEXT, aSongNameGuessSize, (long) aSongNameGuess);
	char* winampStringPtr = strstr(aSongNameGuess, " - Winamp");
```

```
	if (winampStringPtr)
		*winampStringPtr = '\0';
}

void CWinampHandler::PlaySong(const char* aFileName)
{	SendMessage(iWindowHandle,WM_WA_IPC,0,IPC_DELETE);
	COPYDATASTRUCT cds;
	cds.dwData = IPC_PLAYFILE;
	cds.lpData = (void*) aFileName;
	cds.cbData = strlen((char*) cds.lpData) + 1;
	SendMessage(iWindowHandle, WM_COPYDATA, (WPARAM)NULL, (LPARAM)&cds);
	SendMessage(iWindowHandle, WM_WA_IPC, 0, IPC_STARTPLAY);
}

void CWinampHandler::AddSongToPlaylist(const char* aFileName)
{	COPYDATASTRUCT cds;
	cds.dwData = IPC_PLAYFILE;
	cds.lpData = (void*) aFileName;
	cds.cbData = strlen((char*) cds.lpData) + 1;
	SendMessage(iWindowHandle, WM_COPYDATA, (WPARAM)NULL, (LPARAM)&cds);
	int playlistPos = SendMessage(iWindowHandle,WM_WA_IPC,0,IPC_WRITEPLAYLIST);
}

void CWinampHandler::SetVolume(int aVolume)
{	SendMessage(iWindowHandle, WM_WA_IPC, aVolume, IPC_SETVOLUME);
}

void CWinampHandler::Play()
{	SendMessage(iWindowHandle, WM_COMMAND, WINAMP_BUTTON2, 0);
}

void CWinampHandler::Pause()
{	SendMessage(iWindowHandle, WM_COMMAND, WINAMP_BUTTON3, 0);
}

void CWinampHandler::Stop(TStopMode aStopMode)
{	if (aStopMode == EAtEndOfSong)
		SendMessage(iWindowHandle, WM_COMMAND, WINAMP_BUTTON4_CTRL, 0);
	else if ((aStopMode == EFadeOut) && (iWinampVersion < 0x3000))
		SendMessage(iWindowHandle, WM_COMMAND, WINAMP_BUTTON4_SHIFT, 0);
	else
		SendMessage(iWindowHandle, WM_COMMAND, WINAMP_BUTTON4, 0);
}

void CWinampHandler::FastForward()
{	SendMessage(iWindowHandle, WM_COMMAND, WINAMP_FFWD5S, 0);
}

void CWinampHandler::Rewind()
{	SendMessage(iWindowHandle, WM_COMMAND, WINAMP_REW5S, 0);
}

void CWinampHandler::NextTrack()
{	SendMessage(iWindowHandle, WM_COMMAND, WINAMP_BUTTON5, 0);
}

void CWinampHandler::PreviousTrack()
{	SendMessage(iWindowHandle, WM_COMMAND, WINAMP_BUTTON1, 0);
}

void CWinampHandler::SetShuffle(BOOL aShuffleOn)
{	SendMessage(iWindowHandle, WM_WA_IPC, aShuffleOn, IPC_SET_SHUFFLE);
}

void CWinampHandler::SetRepeat(BOOL aRepeatOn)
{	SendMessage(iWindowHandle, WM_WA_IPC, aRepeatOn, IPC_SET_REPEAT);
}
```

```
void CWinampHandler::GetPlaylist(char*& aPlaylist, int& aPlaylistLength, int& aPlaylistPos)
{	aPlaylistPos = SendMessage(iWindowHandle, WM_WA_IPC, 0, IPC_WRITEPLAYLIST);
	aPlaylist = NULL;
	aPlaylistLength = 0;
	char winampDir[MAX_PATH];
	strcpy(winampDir, iWinampPath);
	if (strrchr(winampDir, '\\'))
	{	*strrchr(winampDir, '\\') = '\0';
		strcat(winampDir, "\\Winamp.m3u");
		FILE* file = fopen(winampDir, "rb");
		if (file)
		{	fseek(file, 0, SEEK_END);
			aPlaylistLength = ftell(file) + 1;
			fseek(file, 0, SEEK_SET);
			aPlaylist = new char[aPlaylistLength];
			fread(aPlaylist, aPlaylistLength - 1, 1, file);
			aPlaylist[aPlaylistLength-1] = '\0';
			fclose(file);
		}
	}
}

void CWinampHandler::SelectInPlaylist(int aIndex)
{	SendMessage(iWindowHandle, WM_WA_IPC, aIndex, IPC_SETPLAYLISTPOS);
	SendMessage(iWindowHandle, WM_COMMAND, WINAMP_BUTTON2, 0);
}

void CWinampHandler::RemoveAllFromPlaylist()
{	SendMessage(iWindowHandle, WM_WA_IPC, 0, IPC_DELETE);
}

void CWinampHandler::OneSecondTick()
{}

void CWinampHandler::GetDetailedInfo(int& aBitRate, int& aSampleRate, int& aChannels)
{	aSampleRate = SendMessage(iWindowHandle, WM_WA_IPC, 0, IPC_GETINFO);
	aBitRate = SendMessage(iWindowHandle, WM_WA_IPC, 1, IPC_GETINFO);
	aChannels = SendMessage(iWindowHandle, WM_WA_IPC, 2, IPC_GETINFO);
}

void CWinampHandler::FullScreen()
{}

void CWinampHandler::SetSeek(int aSeek)
{	SendMessage(iWindowHandle, WM_WA_IPC, aSeek * 1000, IPC_JUMPTOTIME);
}

void CWinampHandler::GetPlaylistLength(int& aPlaylistLength)
{	aPlaylistLength = SendMessage(iWindowHandle, WM_WA_IPC, 0, IPC_GETLISTLENGTH);
}

int CWinampHandler::GetVolume()
{	return SendMessage(iWindowHandle, WM_WA_IPC, -666, IPC_SETVOLUME);
}
```

BlueTServer.h

```
#if
!defined(AFX_BLUETSERVER_H__655F5449_5DBC_4F5F_B6EF_E32F2C5CE207__INCLUDED_)
#define AFX_BLUETSERVER_H__655F5449_5DBC_4F5F_B6EF_E32F2C5CE207__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
#ifndef __AFXWIN_H__
#endif
#include "resource.h"

class CBlueTServerApp : public CWinApp
{
public:
        CBlueTServerApp();

public:
        virtual BOOL InitInstance();
        DECLARE_MESSAGE_MAP()
};
#endif
```

BlueTServerDlg.h

```
#if
!defined(AFX_BLUETSERVERDLG_H__2FF1B449_EC5A_4856_9F7D_CE65FBFED6B8__INCLUDED_)
#define AFX_BLUETSERVERDLG_H__2FF1B449_EC5A_4856_9F7D_CE65FBFED6B8__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif
#define MAJOR_VERSION       1
#define MINOR_VERSION       00
#define PATCH_VERSION       ""
#define NUM_HANDLERS        3
#define TIMER_ID            1
#include "BlueTServer.h"

enum TStopMode
        {
        EImmediate,
        EFadeOut,
        EAtEndOfSong
        };

class CBlueTServerDlgAutoProxy;
class CDirTreeNode
{
public:
        CDirTreeNode();
        ~CDirTreeNode();
        unsigned char iNodeType;
        unsigned char iFileType;
        char* iFileName;

        BOOL iWrittenToBuffer;
```

```
	CDirTreeNode* iParent;
	CDirTreeNode* iFirstChild;
	CDirTreeNode* iNextSibling;
};

class CAppHandler
{
public:
	virtual BOOL CheckRunning(const char* aPath, BOOL aStartAutomatically) = 0;
	virtual int GetVersion() = 0;
	virtual void GetInfo(unsigned char& aPlayStatus, int& aTotalTime, int& aCurrentTime,
		BOOL& aShuffle, BOOL& aRepeat, char* aSongNameGuess, int
SongNameGuessSize,
		BOOL aWaitForPlaying = TRUE) = 0;
	virtual void PlaySong(const char* aFileName) = 0;
	virtual void AddSongToPlaylist(const char* aFileName) = 0;
	virtual void SetVolume(int aVolume) = 0;
	virtual int GetVolume() = 0;
	virtual void Play() = 0;
	virtual void Pause() = 0;
	virtual void Stop(TStopMode aStopMode) = 0;
	virtual void FastForward() = 0;
	virtual void Rewind() = 0;
	virtual void NextTrack() = 0;
	virtual void PreviousTrack() = 0;
	virtual void SetShuffle(BOOL aShuffleOn) = 0;
	virtual void SetRepeat(BOOL aRepeatOn) = 0;
	virtual void GetPlaylist(char*& aPlaylist, int& aPlaylistLength, int& aPlaylistPos) = 0;
	virtual void SelectInPlaylist(int aIndex) = 0;
	virtual void RemoveAllFromPlaylist() = 0;
	virtual void OneSecondTick() = 0;
	virtual void GetDetailedInfo(int& aBitRate, int& aSampleRate, int& aChannels) = 0;
	virtual void FullScreen() = 0;
	virtual void SetSeek(int aSeek) = 0;
	virtual void GetPlaylistLength(int& aPlaylistLength) = 0;
};

class CBlueTServerDlg : public CDialog
{
	DECLARE_DYNAMIC(CBlueTServerDlg);
	friend class CBlueTServerDlgAutoProxy;

public:
	CBlueTServerDlg(CWnd* pParent = NULL);
	~CBlueTServerDlg();
	virtual LRESULT WindowProc(UINT message, WPARAM wParam, LPARAM lParam);

	enum { IDD = IDD_BLUETSERVER_DIALOG };
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);
protected:
	CBlueTServerDlgAutoProxy* m_pAutoProxy;
	HICON m_hIcon;

	BOOL CanExit();

	virtual BOOL OnInitDialog();
	afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
```

```
	afx_msg void OnClose();
	afx_msg void OnConnect();
	afx_msg void OnExit();
	afx_msg void OnSize(UINT nType, int cx, int cy);
	afx_msg void OnAddpath();
	afx_msg void OnBrowsepath();
	afx_msg void OnClearpath();
	afx_msg void OnSelchangeAppcombo();
	afx_msg void OnStopwhenswitching();
	afx_msg void OnListenonsocket();
	DECLARE_MESSAGE_MAP()

public:
	int FindCommPortInRegistry();
	void InitialiseSettingsFromWizard(int aDefaultApplication, const char* aAppPath,
		const char* aMusicPath, const char* aCommPortName);

private:
	static UINT BluetoothThreadFunction(LPVOID pParam);
	static UINT SocketThreadFunction(LPVOID pParam);
	void BluetoothConnect(char* aCommPort);
	void ReadCommand(HANDLE aHandle, SOCKET aSocket = 0);
	void HandleBluetooth();
	void HandleSocket();
	BOOL CheckAppRunning();
	BOOL SetPrivilege(HANDLE hToken, LPCTSTR lpszPrivilege, BOOL bEnablePrivilege);
	BOOL ReadFromPhone(SOCKET aSocket, unsigned char* aBuffer, int aLength);
	void WriteListToPhone(SOCKET aSocket);
	void WriteDirectoryListToPhone(SOCKET aSocket);
	void WriteInfoToPhone(SOCKET aSocket);
	void WriteInf2ToPhone(SOCKET aSocket);
	void Play();
	void PlayFile(BOOL aAddToPlaylist, SOCKET aSocket);
	void GetFileInfo(SOCKET aSocket);
	void DownloadFile(SOCKET aSocket);
	void SetVolume(SOCKET aSocket);
	void GetVolume(SOCKET aSocket);
	void Stop(TStopMode aStopMode);
	void Pause();
	void NextTrack();
	void PreviousTrack();
	void FastForward();
	void Rewind();
	void Shuffle(SOCKET aSocket);
	void Repeat(SOCKET aSocket);
	void ShutDown();
	void SendAck(SOCKET aSocket);
	void WritePlaylist(SOCKET aSocket);
	void SelectInPlaylist(SOCKET aSocket);
	void RemoveAllFromPlaylist();
	void GetDetailedInfo(SOCKET aSocket);
	void FullScreen();
	void WriteVersion(SOCKET aSocket);
	void SetSeek(SOCKET aSocket);
	void WritePlaylistLength(SOCKET aSocket);
	void WriteIniFile();
	void ReadIniFile();
	void GenerateFileList(const char* aRoot, BOOL aRecursive);
	void ReadDir(char* aDir, CDirTreeNode* aNode, BOOL aRecursive);
```

```
        BOOL WriteNodeToBuffer(CDirTreeNode* aNode, unsigned char*& aBufferPtr, int&
                aSizeRemaining);
        void ResetWrittenFlag(CDirTreeNode* aNode);
        CAppHandler* FindHandlerForFile(const char* aFileName);
        void StopListeningOnSocket();
        void ListenOnSocket();
        void CopyIntIntoBuffer(int aValue, char* aBuffer);
        void Log(char* aFormatString, ...);

private:
        CDirTreeNode* iDirTreeRoot;
        CDirTreeNode* iDirTreeCurrentNode;

        unsigned char iCurrentVolume;
        HANDLE iBluetoothHandle;
        SOCKET iSocketHandle;
        BOOL iExitNow;
        int iErrorCount;
        CWinThread* iBluetoothThread;
        CWinThread* iSocketThread;

        BOOL iListenOnSocket;
        BOOL iClientClosed;
        SOCKADDR_IN iClientAddr;
        int iClientAddrLength;

        NOTIFYICONDATA iNotifyIconData;
        char iCommPortName[32];
        char iRootPath[MAX_PATH];
        char iStartDirectory[MAX_PATH];
        BOOL iHidden;
        CAppHandler* iAppHandler;
        CAppHandler* iAppHandlers[NUM_HANDLERS];
        char iAppPaths[NUM_HANDLERS][MAX_PATH];
        char iExtensionsHandled[NUM_HANDLERS][MAX_PATH];
        int iDefaultAppIndex;
        int iCurrentAppIndex;
};
#endif
```

Dlgproxy.h

```
#if
!defined(AFX_DLGPROXY_H__B719A39A_0FFE_485F_8C8E_C8D450A853A1__INCLUDED_)
#define AFX_DLGPROXY_H__B719A39A_0FFE_485F_8C8E_C8D450A853A1__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
class CBlueTServerDlg;
class CBlueTServerDlgAutoProxy : public CCmdTarget
{
        DECLARE_DYNCREATE(CBlueTServerDlgAutoProxy)
        CBlueTServerDlgAutoProxy

public:
        CBlueTServerDlg* m_pDialog;
public:
        virtual void OnFinalRelease();
```

```
protected:
	virtual ~CBlueTServerDlgAutoProxy();

	DECLARE_MESSAGE_MAP()
	DECLARE_OLECREATE(CBlueTServerDlgAutoProxy)
	DECLARE_DISPATCH_MAP()
	DECLARE_INTERFACE_MAP()
};
#endif
```

MediaPlayer.h

```
#ifndef MEDIAPLAYER_H__
#define MEDIAPLAYER_H__
#include "BlueTServerDlg.h"

class CMediaPlayerHandler : public CAppHandler
	{
public:
	virtual BOOL CheckRunning(const char* aPath, BOOL aStartAutomatically);
	BOOL DoCheckRunning(const char* aWindowName, const char* aPath, BOOL
aStartAutomatically);
	virtual int GetVersion();
	virtual void GetInfo(unsigned char& aPlayStatus, int& aTotalTime, int& aCurrentTime,
		BOOL& aShuffle, BOOL& aRepeat, char* aSongNameGuess, int
aSongNameGuessSize);
	virtual void PlaySong(const char* aFileName);
	virtual void AddSongToPlaylist(const char* aFileName);
	virtual void SetVolume(int aVolume);
	virtual void Play();
	virtual void Pause();
	virtual void Stop(BOOL aFadeOut);
	virtual void FastForward();
	virtual void Rewind();
	virtual void NextTrack();
	virtual void PreviousTrack();
	virtual void SetShuffle(BOOL aShuffleOn);
	virtual void SetRepeat(BOOL aRepeatOn);

private:
	HWND iWindowHandle;
};
#endif
```

MediaPlayerHandler.h

```
#ifndef MEDIAPLAYERHANDLER_H__
#define MEDIAPLAYERHANDLER_H__
#include "BlueTServerDlg.h"

class CMediaPlayerHandler : public CAppHandler
{
public:
	CMediaPlayerHandler();
	virtual BOOL CheckRunning(const char* aPath, BOOL aStartAutomatically);
	virtual int GetVersion();
```

```
	virtual void GetInfo(unsigned char& aPlayStatus, int& aTotalTime, int& aCurrentTime,
		BOOL& aShuffle, BOOL& aRepeat, char* aSongNameGuess, int
aSongNameGuessSize,
		BOOL aWaitForPlaying);
	virtual void PlaySong(const char* aFileName);
	virtual void AddSongToPlaylist(const char* aFileName);
	virtual void SetVolume(int aVolume);
	virtual int GetVolume();
	virtual void Play();
	virtual void Pause();
	virtual void Stop(TStopMode aStopMode);
	virtual void FastForward();
	virtual void Rewind();
	virtual void NextTrack();
	virtual void PreviousTrack();
	virtual void SetShuffle(BOOL aShuffleOn);
	virtual void SetRepeat(BOOL aRepeatOn);
	virtual void GetPlaylist(char*& aPlaylist, int& aPlaylistLength, int& aPlaylistPos);
	virtual void SelectInPlaylist(int aIndex);
	virtual void RemoveAllFromPlaylist();
	virtual void OneSecondTick();
	virtual void GetDetailedInfo(int& aBitRate, int& aSampleRate, int& aChannels);
	virtual void FullScreen();
	virtual void SetSeek(int aSeek);
	virtual void GetPlaylistLength(int& aPlaylistLength);

private:
	HWND iWindowHandle;
	unsigned char iPlayStatus;
	int iVolume;
	char iMediaPlayerCommandLine[256];
};
#endif
```

Msppt8vr.h

```
class _Application : public COleDispatchDriver
{
public:
	_Application() {}
	_Application(LPDISPATCH pDispatch) : COleDispatchDriver(pDispatch) {}
	_Application(const _Application& dispatchSrc) : COleDispatchDriver(dispatchSrc) {}

public:
	CString GetName();
	CString GetVersion();
	LPDISPATCH NewShow(LPCTSTR FileName, long AdvanceMode, long Kiosk);
	void Quit();
};

class SlideShowView : public COleDispatchDriver
{
public:
	SlideShowView() {}
	SlideShowView(LPDISPATCH pDispatch) : COleDispatchDriver(pDispatch) {}
	SlideShowView(const SlideShowView& dispatchSrc) : COleDispatchDriver(dispatchSrc) {}

public:
```

```
        LPDISPATCH GetParent();
        void Next();
        void Previous();
        void GotoSlide(long Index, long ResetSlide);
        long GetSlidesCount();
        long GetSlideNumber();
        void Exit();
};
```

PowerPointHandler.h

```
#ifndef POWERPOINTHANDLER_H__
#define POWERPOINTHANDLER_H__
#include "BlueTServerDlg.h"
#import "MSPPT8VR.OLB"
using namespace PowerPointViewer;

class CPowerPointHandler : public CAppHandler
        {
public:
        CPowerPointHandler();
        virtual ~CPowerPointHandler();
        virtual BOOL CheckRunning(const char* aPath, BOOL aStartAutomatically);
        virtual int GetVersion();
        virtual void GetInfo(unsigned char& aPlayStatus, int& aTotalTime, int& aCurrentTime,
                BOOL& aShuffle, BOOL& aRepeat, char* aSongNameGuess, int
                        aSongNameGuessSize, BOOL aWaitForPlaying);
        virtual void PlaySong(const char* aFileName);
        virtual void AddSongToPlaylist(const char* aFileName);
        virtual void SetVolume(int aVolume);
        virtual int GetVolume();
        virtual void Play();
        virtual void Pause();
        virtual void Stop(TStopMode aStopMode);
        virtual void FastForward();
        virtual void Rewind();
        virtual void NextTrack();
        virtual void PreviousTrack();
        virtual void SetShuffle(BOOL aShuffleOn);
        virtual void SetRepeat(BOOL aRepeatOn);
        virtual void GetPlaylist(char*& aPlaylist, int& aPlaylistLength, int& aPlaylistPos);
        virtual void SelectInPlaylist(int aIndex);
        virtual void RemoveAllFromPlaylist();
        virtual void OneSecondTick();
        virtual void GetDetailedInfo(int& aBitRate, int& aSampleRate, int& aChannels);
        virtual void FullScreen();
        virtual void SetSeek(int aSeek);
        virtual void GetPlaylistLength(int& aPlaylistLength);

private:
        _ApplicationPtr iPowerPoint;
        SlideShowViewPtr iSlideShowView;
        int iNumSlides;
        char iSlideShowName[_MAX_PATH];
        BOOL iScreenSaverEnabled[3];
        };
#endif
```

Resource.h

```
#define IDM_ABOUTBOX                    0x0010
#define IDD_ABOUTBOX                    100
#define IDP_OLE_INIT_FAILED             100
#define IDS_ABOUTBOX                    101
#define IDD_BLUETSERVER_DIALOG          102
#define IDP_SOCKETS_INIT_FAILED         103
#define IDR_MAINFRAME                   128
#define IDI_BLUETICON                   129
#define IDI_TRAYICON                    130
#define IDC_EDITPATH                    1000
#define IDC_ADDPATH                     1001
#define IDC_CLEARPATH                   1002
#define IDC_COMPORT                     1003
#define IDC_CONNECT                     1004
#define IDC_APPCOMBO                    1005
#define IDC_FILETYPE                    1006
#define IDC_APPPATH                     1009
#define IDC_BROWSEPATH                  1010
#define IDC_LOGBOX                      1011
#define IDC_EXIT                        1012
#define IDC_LISTENONSOCKET              1013
#define IDC_STARTAPP                    1014
#define IDC_STOPWHENSWITCHING           1015
#define IDC_STARTMINIMISED              1016
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        132
#define _APS_NEXT_COMMAND_VALUE         32771
#define _APS_NEXT_CONTROL_VALUE         1017
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

Stdafx.h

```
#if !defined(AFX_STDAFX_H__069C5267_E0CB_422E_96FF_6736794FF7AE__INCLUDED_)
#define AFX_STDAFX_H__069C5267_E0CB_422E_96FF_6736794FF7AE__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif #define VC_EXTRALEAN
#include <afxwin.h>
#include <afxdisp.h>
#include <afxdtctl.h>
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>
#endif
#include <afxsock.h>
#ifndef IMPLEMENT_OLECREATE2
#define IMPLEMENT_OLECREATE2(class_name, external_name, l, w1, w2, b1, b2, b3, b4, b5, b6, b7,
b8)\ AFX_DATADEF COleObjectFactory class_name::factory(class_name::guid, \
RUNTIME_CLASS(class_name), TRUE, _T(external_name)); \
const AFX_DATADEF GUID class_name::guid = \
{ l, w1, w2, { b1, b2, b3, b4, b5, b6, b7, b8 } };
#endif
#endif
```

Winamp.h

```
#ifndef _WAFE_H_
#define _WAFE_H_
#define WM_WA_IPC WM_USER
#define IPC_GETVERSION 0
#define IPC_DELETE 101
#define IPC_STARTPLAY 102
#define IPC_ISPLAYING 104
#define IPC_GETOUTPUTTIME 105
#define IPC_JUMPTOTIME 106
#define IPC_WRITEPLAYLIST 120
#define IPC_SETPLAYLISTPOS 121
#define IPC_SETVOLUME 122
#define IPC_SETPANNING 123
#define IPC_GETLISTLENGTH 124
#define IPC_SETSKIN 200
#define IPC_GETSKIN 201
#define IPC_EXECPLUG 202
#define IPC_GETPLAYLISTFILE 211
#define IPC_GETPLAYLISTTITLE 212
#define IPC_GETLISTPOS 125
#define IPC_GETINFO 126
#define IPC_GETEQDATA 127
#define IPC_SETEQDATA 128
#define IPC_ADDBOOKMARK 129
#define IPC_RESTARTWINAMP 135
#define IPC_MBOPEN 241
#define IPC_INETAVAILABLE 242
#define IPC_UPDTITLE 243
#define IPC_CHANGECURRENTFILE 245
#define IPC_GETMBURL 246
#define IPC_REFRESHPLCACHE 247
#define IPC_MBBLOCK 248
#define IPC_MBOPENREAL 249
#define IPC_GET_SHUFFLE 250
#define IPC_GET_REPEAT 251
#define IPC_SET_SHUFFLE 252
#define IPC_SET_REPEAT 253
#define IPC_PLAYFILE 100
#define IPC_CHDIR 103
#define WINAMP_OPTIONS_EQ            40036 // toggles the EQ window
#define WINAMP_OPTIONS_PLEDIT        40040 // toggles the playlist window
#define WINAMP_VOLUMEUP              40058 // turns the volume up a little
#define WINAMP_VOLUMEDOWN            40059 // turns the volume down a little
#define WINAMP_FFWD5S                40060 // fast forwards 5 seconds
#define WINAMP_REW5S                 40061 // rewinds 5 seconds
#define WINAMP_BUTTON1               40044
#define WINAMP_BUTTON2               40045
#define WINAMP_BUTTON3               40046
#define WINAMP_BUTTON4               40047
#define WINAMP_BUTTON5               40048
#define WINAMP_BUTTON1_SHIFT         40144
#define WINAMP_BUTTON2_SHIFT         40145
#define WINAMP_BUTTON3_SHIFT         40146
#define WINAMP_BUTTON4_SHIFT         40147
#define WINAMP_BUTTON5_SHIFT         40148
#define WINAMP_BUTTON1_CTRL          40154
```

```
#define WINAMP_BUTTON2_CTRL          40155
#define WINAMP_BUTTON3_CTRL          40156
#define WINAMP_BUTTON4_CTRL          40157
#define WINAMP_BUTTON5_CTRL          40158
#define WINAMP_FILE_PLAY            40029 // pops up the load file(s) box
#define WINAMP_OPTIONS_PREFS         40012 // pops up the preferences
#define WINAMP_OPTIONS_AOT           40019 // toggles always on top
#define WINAMP_HELP_ABOUT            40041 // pops up the about box :)
#endif
```

WinampHandler.h

```
#ifndef WINAMP_HANDLER_H__
#define WINAMP_HANDLER_H__
#include "BlueTServerDlg.h"

class CWinampHandler : public CAppHandler
{
public:
        virtual BOOL CheckRunning(const char* aPath, BOOL aStartAutomatically);
        BOOL DoCheckRunning(const char* aWindowName, const char* aPath, BOOL
aStartAutomatically);
        virtual int GetVersion();
        virtual void GetInfo(unsigned char& aPlayStatus, int& aTotalTime, int& aCurrentTime,
                BOOL& aShuffle, BOOL& aRepeat, char* aSongNameGuess, int aSongNameGuessSize,
                BOOL aWaitForPlaying);
        virtual void PlaySong(const char* aFileName);
        virtual void AddSongToPlaylist(const char* aFileName);
        virtual void SetVolume(int aVolume);
        virtual int GetVolume();
        virtual void Play();
        virtual void Pause();
        virtual void Stop(TStopMode aStopMode);
        virtual void FastForward();
        virtual void Rewind();
        virtual void NextTrack();
        virtual void PreviousTrack();
        virtual void SetShuffle(BOOL aShuffleOn);
        virtual void SetRepeat(BOOL aRepeatOn);
        virtual void GetPlaylist(char*& aPlaylist, int& aPlaylistLength, int& aPlaylistPos);
        virtual void SelectInPlaylist(int aIndex);
        virtual void RemoveAllFromPlaylist();
        virtual void OneSecondTick();
        virtual void GetDetailedInfo(int& aBitRate, int& aSampleRate, int& aChannels);
        virtual void FullScreen();
        virtual void SetSeek(int aSeek);
        virtual void GetPlaylistLength(int& aPlaylistLength);
private:
        HWND iWindowHandle;
        int iWinampVersion;
        char iWinampPath[MAX_PATH];
};
#endif
```

BlueTApp.cpp

```
#include "BlueTApp.h"
#include "BlueTDocument.h"
#include <eikenv.h>

TUid CBlueTApp::AppDllUid() const
   {
            return KUidBlueT;
   }

CApaDocument* CBlueTApp::CreateDocumentL()
   {
             return CBlueTDocument::NewL(*this);
   }

EXPORT_C CApaApplication* NewApplication()
   {
            return new CBlueTApp;
    }

GLDEF_C TInt E32Dll(TDllReason)
   {
            return KErrNone;
   }
```

BlueTAppUi.cpp

```
#include "BlueTApp.h"
#include "BlueTAppUi.h"
#include "BlueTBrowseContainer.h"
#include "BlueTControlContainer.h"
#include "BlueT.hrh"
#include <avkon.hrh>
#include <aknpopup.h>
#include "BlueTBrowseView.h"
#include "BlueTControlView.h"
#include "BlueTPlaylistView.h"
#include "BlueTDisplaySettingsView.h"
#include "BlueTDisplaySettingsContainer.h"
#include "BlueTBehaviourSettingsView.h"
#include "BlueTBehaviourSettingsContainer.h"
#include <BlueT.rsg>

CBlueTAppUi::~CBlueTAppUi()
{           iPlaylist.ResetAndDestroy();
            iPlaylist.Close();
            delete iIncomingCallListener;
            delete iSongTimer;
            delete iBluetoothHandler;
            delete iDecoratedTabGroup;
}

TInt CBlueTAppUi::LoadSettings()
{           RFile file;
            TInt ret;
            TFileName iniFilename;
            BuildIniPath(iniFilename);
            ret = file.Open(iEikEnv->FsSession(), iniFilename, EFileRead | EFileShareAny);
            if (ret != KErrNone)
                        return ret;
            TBuf8<114> settingsBuf;
            ret = file.Read(settingsBuf);
            file.Close();
```

```
	TLex8 lex(settingsBuf);
	lex.Val(iSettings.iActionOnIncomingCall);
	lex.SkipSpace();
	lex.Val(iSettings.iDisplayType);
	lex.SkipSpace();
	lex.Val(iSettings.iTimeDisplay);
	lex.SkipSpace();
	lex.Val(iSettings.iSkinFileIndex);
	lex.SkipSpace();
	lex.Val(iSettings.iSongBrowserFont);
	lex.SkipSpace();

	TInt volume = KStartVolume;
	lex.Val(volume);
	lex.SkipSpace();
	iCurrentVolume = (TUint8) volume;
	for (TInt i = 0; i < 6; i++)
		{
		TInt tempByte;
		lex.Val(tempByte);
		iSettings.iAutoConnectAddress[i] = (TUint8) tempByte;
		lex.SkipSpace();
		}
	lex.Val(iSettings.iAutoConnectPort);
	lex.SkipSpace();
	lex.Val(iSettings.iUseId3Tags);
	lex.SkipSpace();
	lex.Val(iSettings.iDownloadDrive);
	lex.SkipSpace();
	return ret;
}

TInt CBlueTAppUi::SaveSettings()
{	RFile file;
	TInt ret = KErrNone;
	TFileName iniFilename;
	BuildIniPath(iniFilename);
	if (iDisplaySettingsView->SettingsContainer())
		TRAP(ret, iDisplaySettingsView->SettingsContainer()->iListBox->StoreSettingsL());
	if (iBehaviourSettingsView->SettingsContainer())
		TRAP(ret, iBehaviourSettingsView->SettingsContainer()->iListBox->StoreSettingsL());
	if (ret != KErrNone)
		return ret;
	ret = file.Replace(iEikEnv->FsSession(), iniFilename, EFileShareAny);
	if (ret != KErrNone)
		return ret;

	TBuf8<256> settingsBuf;
	settingsBuf.Format(_L8("%1d %1d %1d %02d %1d %03d"),
		iSettings.iActionOnIncomingCall,
		iSettings.iDisplayType,
		iSettings.iTimeDisplay, iSettings.iSkinFileIndex,
		iSettings.iSongBrowserFont, iCurrentVolume);
	ret = file.Write(settingsBuf);
	for (TInt i = 0; i < 6; i++)
		{
		settingsBuf.Format(_L8(" %03d"), iSettings.iAutoConnectAddress[i]);
		file.Write(settingsBuf);
		}
	settingsBuf.Format(_L8(" %02d %1d %02d "), iSettings.iAutoConnectPort,
		iSettings.iUseId3Tags, iSettings.iDownloadDrive);
	file.Write(settingsBuf);
	file.Close();
	return ret;
}
```

```
void CBlueTAppUi::HandleCommandL(TInt aCommand)
{
CBlueTBrowseContainer* browseContainer = BrowseView()->BrowseContainer();
switch (aCommand)
	{	case EEikCmdExit:
			SaveSettings();
			Exit();
			break;
		case EBlueTCmdRefreshList:
			iBluetoothHandler->RequestList(ETrue);
			break;
		case EBlueTCmdPlay:
			if (browseContainer)
			browseContainer->PlaySelectedFileL(EFalse);
			break;
		case EBlueTCmdAddToPlaylist:
			if (browseContainer)
			browseContainer->PlaySelectedFileL(ETrue);
			break;
		case EBlueTCmdDownload:
			if (browseContainer)
			browseContainer->DownloadSelectedFileL();
			break;
		case EBlueTCmdShuffle:
			iBluetoothHandler->RequestShuffle(!iShuffle);
			break;
		case EBlueTCmdRepeat:
			iBluetoothHandler->RequestRepeat(!iRepeat);
			break;
		case EBlueTCmdFullScreen:
			iBluetoothHandler->RequestFullScreen();
			break;
		case EBlueTCmdShutDown:
			{
			TBuf<256> resourceBuf;
			iEikEnv->ReadResource(resourceBuf,
			R_SHUTDOWN_QUERY);
			if (iEikEnv->QueryWinL(resourceBuf, KNullDesC))
			iBluetoothHandler->RequestShutDown();
			}
			break;
		case EBlueTCmdFind:
			if (browseContainer)
			browseContainer->FindSongL(EFalse);
			break;
		case EBlueTCmdFindServer:
			iBluetoothHandler->RequestFindServer();
			break;
		case EBlueTCmdExitDisplaySettings:
			ReloadSkin();
		case EBlueTCmdExitBehaviourSettings:
			iTabGroup->SetActiveTabByIndex(1);
			ActivateLocalViewL(KControlViewId);
			break;
		case EBlueTCmdFindInPlaylist:
			iPlaylistView->FindSongL(EFalse);
			break;
		case EBlueTCmdTrackInfo:
			iBluetoothHandler->RequestDetailedTrackInfo();
			break;
		case EBlueTCmdPlayFromPlaylist:
			if (iPlaylistView->IsFocused())
			{		TInt selection = iPlaylistView->GetSelectedFile();
				if (selection != KErrNotFound)
				iBluetoothHandler->RequestSelectInPlaylist(selection);
			}
			break;
```

```
		case EBlueTCmdRefreshPlaylist:
			GetPlaylist();
			break;
		case EBlueTCmdRemoveAllFromPlaylist:
			iBluetoothHandler->RequestRemoveAllFromPlaylist();
			break;
		case EBlueTCmdShowDisplaySettings:
			ActivateLocalViewL(KDisplaySettingsViewId);
			break;
		case EBlueTCmdShowBehaviourSettings:
			ActivateLocalViewL(KBehaviourSettingsViewId);
			break;
		default:
		break;
	}
}

void CBlueTAppUi::ConnectComplete(TInt aError, const TBTDevAddr& aAddress, TInt aPort)
{	if ((aError != KErrNone) && (aError != KErrCancel))
			{
			TBuf<256> formatBuf;
			TBuf<256> resourceBuf;
			iEikEnv->ReadResource(resourceBuf, R_CONNECT_FAILURE);
			formatBuf.Format(resourceBuf, aError);
			iEikEnv->InfoWinL(formatBuf, KNullDesC);
			}
		if (aError == KErrNone)
			{
			iSettings.iAutoConnectAddress = aAddress;
			iSettings.iAutoConnectPort = aPort;
			}
		DrawControlView(KBlueTDrawEverything);
}

void CBlueTAppUi::CommandComplete(TBlueTCommand aCommand, TInt aError)
{	TInt err;
		if (aError != KErrNone)
		{
		TBuf<256> formatBuf;
		switch (aError)
		{	case KErrCancel:
			case KErrTooBig:
				return;
			case KErrDisconnected:
			case KErrNotReady:
				iEikEnv->ReadResource(formatBuf, R_DISCONNECTED_ERROR);
				break;
			case KErrCouldNotConnect:
			case KErrHCIConnectFailed:
				iEikEnv->ReadResource(formatBuf, R_COULDNOTCONNECT_ERROR);
				break;
			case KErrCorrupt:
				iEikEnv->ReadResource(formatBuf, R_CORRUPT_ERROR);
				break;
			case KErrRfcommBadAddress:
			case -6004:
				iEikEnv->ReadResource(formatBuf, R_BLUETOOTH_ERROR);
				break;
			case KErrTimedOut:
				iEikEnv->ReadResource(formatBuf, R_TIMEOUT_ERROR);
				break;
			case KErrPathNotFound:
				iEikEnv->ReadResource(formatBuf, R_PATHNOTFOUND_ERROR);
				break;
			default:
			{	TBuf<256> resourceBuf;
```

```
                        iEikEnv->ReadResource(resourceBuf, R_UNKNOWN_ERROR);
                        formatBuf.Format(resourceBuf, aError);
                }
                        break;
        }
        TRAP(err, iEikEnv->InfoWinL(KNullDesC, formatBuf));
        return;
        }
        if (aCommand == EBlueTCurrentTrackInfo)
        {DrawControlView(KBlueTDrawEverything & ~KBlueTDrawBackground);}
        else if (aCommand == EBlueTRemoveAllFromPlaylist)
                {
                iPlaylist.ResetAndDestroy();
                iPlaylistIndex = 0;
                }
        else if (aCommand == EBlueTGetPlaylist)
                {
                if (iPlaylistView->IsFocused())
                        TRAP(err, iPlaylistView->RefreshPlaylistL(iPlaylist,
                iPlaylistIndex));
                }
}

void CBlueTAppUi::RefreshFileListL(CDirTreeNode* aDirectory)
{       iCurrentDirectory = aDirectory;
        iBrowseView->SetListLoaded(ETrue);
        if (BrowseView()->BrowseContainer())
                {
                BrowseView()->BrowseContainer()->RefreshFileListL(aDirectory);
                BrowseView()->BrowseContainer()->DrawNow();
                }
        if (!iSongTimer->IsActive())
                iSongTimer->Start(0, 1000000, TCallBack(SongTimerCallback, this));
}

void CBlueTAppUi::GetDirectoryList(CDirTreeNode* aDirectory)
{
        iBluetoothHandler->RequestDirectoryList(aDirectory);
}

void CBlueTAppUi::SetInfo(TBool aPlaying, TBool aPaused, TInt aSongLength, TInt aCurrentTime, TBool
                        aShuffle, TBool aRepeat, const TDesC8& aSongNameGuess)
{       TSongState newSongState = ESongStopped;
        if (aPaused)
                newSongState = ESongPaused;
        else if (aPlaying)
                newSongState = ESongPlaying;
        if (newSongState != iSongState)
        {
                iSongState = newSongState;
                DrawControlView(KBlueTDrawTrackName | KBlueTDrawTrackTime);
        }
        iCurrentTime = aCurrentTime;
        iSongLength = aSongLength;
        if (iShuffle != aShuffle)
                DrawControlView(KBlueTDrawShuffle);
                iShuffle = aShuffle;
        if (iRepeat != aRepeat)
                DrawControlView(KBlueTDrawRepeat);
                iRepeat = aRepeat;
        if (aSongNameGuess.Length() > iSongNameGuess.MaxLength())
                iSongNameGuess.Copy(aSongNameGuess.Left(iSongNameGuess.MaxLength()));
        else
                iSongNameGuess.Copy(aSongNameGuess);
        if (iBehaviourSettingsView->SettingsContainer())
                iBehaviourSettingsView->SettingsContainer()->DrawDeferred();
        if (iDisplaySettingsView->SettingsContainer())
```

```
                iDisplaySettingsView->SettingsContainer()->DrawDeferred();
}

void CBlueTAppUi::SetDetailedInfo(TInt aSampleRate, TInt aBitRate, TInt aChannels)
{       iSampleRate = aSampleRate;
        iBitRate = aBitRate;
        iChannels = aChannels;
        TRAPD(err, DisplayDetailedInfoDialogL(aSampleRate, aBitRate, aChannels));
}

void CBlueTAppUi::PlayFile(CDirTreeNode* aFileNode, const TDesC8& aFileName, TBool aAddToPlaylist)
{       CDirTreeNode* node = aFileNode->iFirstChild;
        HBufC8* fileName;
        if (node)
        {
        if (!aAddToPlaylist)
                {
                iPlaylist.ResetAndDestroy();
                iPlaylistIndex = 0;
                }
        AddFilesToPlaylist(node);
        }
        else
        {       fileName = aFileNode->iFileName->Alloc();
                if (fileName)
                {       if (aAddToPlaylist)
                        {
                                iPlaylist.Append(fileName);
                        }
                        else
                        {       iPlaylist.ResetAndDestroy();
                                iPlaylist.Append(fileName);
                                iPlaylistIndex = 0;
                        }
                }
                if (aAddToPlaylist)
                        iBluetoothHandler->RequestAddFileToPlaylist(aFileName);
                else
                        iBluetoothHandler->RequestPlayFile(aFileName);
                if (!aAddToPlaylist)
                {
                iTabGroup->SetActiveTabByIndex(1);
                TRAPD(ignore, ActivateLocalViewL(TUid::Uid(iTabGroup->TabIdFromIndex(1))));
        }
        iCurrentFile = aFileNode;
}

void CBlueTAppUi::AddFilesToPlaylist(CDirTreeNode* aNode)
{       if (aNode)
        {
        if (aNode->iFirstChild)
        {       AddFilesToPlaylist(aNode->iFirstChild);
                if (aNode->iFirstChild)
                {
                        if (aNode->iNextSibling)
                        AddFilesToPlaylist(aNode->iNextSibling);
                }
        }
        else
        {       if (!aNode->iFirstChild)
                {
                        HBufC8* fileName;
                        fileName = aNode->iFileName->Alloc();
                        if ((fileName)&&(aNode->iFileType!=1))
                        {
                        iPlaylist.Append(fileName);
                        }
```

```
                                aNode = aNode->iNextSibling;
                                while (aNode)
                                {
                                fileName = aNode->iFileName->Alloc();
                                if (fileName)
                                iPlaylist.Append(fileName);
                                aNode = aNode->iNextSibling;
                                }
                        }
                }
}

void CBlueTAppUi::DownloadFile(const TDesC8& aFileName)
{
iBluetoothHandler->RequestFileInfo(aFileName);
}

TInt CBlueTAppUi::SongTimerCallback(TAny* aPtr)
{
        CBlueTAppUi* appUi = (CBlueTAppUi*) aPtr;
        appUi->HandleSongTimer();
        return 0;
}

void CBlueTAppUi::HandleSongTimer()
{
if (iSongState == ESongPlaying)
{       iCurrentTime++;
        if ((iCurrentTime >= iSongLength) && (iSongLength > 0))
        {       if (iPlaylist.Count() > 0)
                {       if (iPlaylistIndex < (iPlaylist.Count() - 1))
                                iPlaylistIndex++;
                        else if (iRepeat)
                                iPlaylistIndex = 0;
                }
                iBluetoothHandler->RequestInfo();
                iSongLength = -1;
        }
if (!ControlView()->ControlContainer() ||!ControlView()->ControlContainer()->CheckFastForwardAndRewind())
        {       DrawControlView(KBlueTDrawTrackName | KBlueTDrawTrackTime);
        }
}
if (BrowseView()->BrowseContainer())
                BrowseView()->BrowseContainer()->ScrollSelectedSong();
else if (PlaylistView()->IsFocused())
                PlaylistView()->ScrollSelectedSong();
}

void CBlueTAppUi::IncomingCall(const TDesC& /*aName*/)
{       switch (iSettings.iActionOnIncomingCall)
        {
        case TBlueTSettings::EPause:
                if (iSongState == ESongPlaying)
                iBluetoothHandler->RequestPause();
                break;
        case TBlueTSettings::EStop:
                iBluetoothHandler->RequestStop();
                break;
        case TBlueTSettings::EMute:
                iCurrentVolume = 0;
                iBluetoothHandler->RequestSetVolume(iCurrentVolume);
                break;
        default:
                break;
        }
        iIncomingCallListener->StartListening();
}
```

```
TInt CBlueTAppUi::ReloadSkin()
{
	TInt err = KErrNone;
	TRAP(err, iControlView->SkinHandler()->LoadSkinL());
	return err;
}

TFileName& CBlueTAppUi::SkinPathL()
{	if (iSkinPath.Length() == 0)
	{	_LIT(KSkins, "skins");
		TFileName app = iEikEnv->EikAppUi()->Application()->AppFullName();
		TParsePtr ptr(app);
		iSkinPath.Copy(ptr.DriveAndPath());
		iSkinPath.Append(KSkins);
	}
	return iSkinPath;
}

void CBlueTAppUi::BuildIniPath(TFileName & aIniFileName)
{	TFileName app = iEikEnv->EikAppUi()->Application()->AppFullName();
	TParsePtr parse(app);
	aIniFileName.Copy(parse.DriveAndPath());
	aIniFileName.Append(parse.Name());
	_LIT(KIni, ".ini");
	aIniFileName.Append(KIni);
}

void CBlueTAppUi::SelectInPlaylist(TInt aIndex)
{
	iPlaylistIndex = aIndex;
	iBluetoothHandler->RequestSelectInPlaylist(aIndex);
}

CDirTreeNode* CBlueTAppUi::CurrentDirectory()
{
	return iCurrentDirectory;
}

void CBlueTAppUi::SetCurrentDirectory(CDirTreeNode* aDirectory)
{
	iCurrentDirectory = aDirectory;
}

TBlueTSettings& CBlueTAppUi::Settings()
{
	return iSettings;
}

TUint CBlueTAppUi::DrawFlags()
{
	return iDrawFlags;
}

void CBlueTAppUi::SetDrawFlags(TUint aFlags)
{
	iDrawFlags = aFlags;
}

CBluetoothHandler* CBlueTAppUi::BluetoothHandler()
{
	return iBluetoothHandler;
}

TSongState CBlueTAppUi::SongState()
{
	return iSongState;
}
```

```
RPointerArray<HBufC8>& CBlueTAppUi::Playlist()
{
	return iPlaylist;
}

TInt& CBlueTAppUi::PlaylistIndex()
{
	return iPlaylistIndex;
}

void CBlueTAppUi::SetPlaylistIndex(TInt aIndex)
{
	iPlaylistIndex = aIndex;
}

TInt CBlueTAppUi::CurrentTime()
{
	return iCurrentTime;
}

TInt CBlueTAppUi::SongLength()
{
	return iSongLength;
}

TBool CBlueTAppUi::Shuffle()
{
	return iShuffle;
}

TBool CBlueTAppUi::Repeat()
{
	return iRepeat;
}

TInt CBlueTAppUi::CurrentVolume()
{
	return iCurrentVolume;
}

const TDesC& CBlueTAppUi::SongNameGuess()
{
	if (iSongNameGuess.Length() > 0)
		return iSongNameGuess;
	else
		return KBlueTName;
}

void CBlueTAppUi::SetSkinAuthor(const TDesC8& aAuthor)
{
	iSkinAuthor = aAuthor;
}

void CBlueTAppUi::SetSkinName(const TDesC& aName)
{
	iSkinName = aName;
}

void CBlueTAppUi::SetDrawPending(TBool aPending)
{
	iDrawPending = aPending;
}

void CBlueTAppUi::DrawControlView(TUint aDrawFlags)
{	if (!iDrawPending)
	{	iDrawFlags = aDrawFlags;
```

```
                iDrawPending = ETrue;
                if (ControlView()->ControlContainer())
                        ControlView()->ControlContainer()->AppInitiatedDraw();
        }
        else
                {
                iDrawFlags |= aDrawFlags;
                }
}

void CBlueTAppUi::SetVolume(TInt aVolume)
{
        iCurrentVolume = (TUint8) aVolume;
        iBluetoothHandler->RequestSetVolume(iCurrentVolume);
}

void CBlueTAppUi::ReceivedVolume(TInt aVolume)
{
        iCurrentVolume = (TUint8) aVolume;
        DrawControlView(KBlueTDrawVolume);
}

CCoeEnv* CBlueTAppUi::CoeEnv()
{
        return iEikEnv;
}

void CBlueTAppUi::GetPlaylist()
{
        iPlaylist.ResetAndDestroy();
        iPlaylistIndex = 0;
        iBluetoothHandler->RequestGetPlaylist();
}

void CBlueTAppUi::ConstructL()
{       BaseConstructL();
        CEikStatusPane* sp = StatusPane();
        iNaviPane = (CAknNavigationControlContainer*)sp->ControlL(TUid::Uid(EEikStatusPaneUidNavi));
        iDecoratedTabGroup = iNaviPane->ResourceDecorator();
        if (iDecoratedTabGroup)
                iTabGroup = (CAknTabGroup*) iDecoratedTabGroup->DecoratedControl();
                iEikEnv = CEikonEnv::Static();
                iSettings.iSongBrowserFont = TBlueTSettings::EMedium;
                iCurrentVolume = KStartVolume;
                LoadSettings();
                iDrawFlags = KBlueTDrawEverything;
                iBrowseView = new (ELeave) CBlueTBrowseView;
                CleanupStack::PushL(iBrowseView);
                iBrowseView->ConstructL();
                AddViewL(iBrowseView);
                CleanupStack::Pop();
                iPlaylistView = new (ELeave) CBlueTPlaylistView;
                CleanupStack::PushL(iPlaylistView);
                iPlaylistView->ConstructL();
                AddViewL(iPlaylistView);
                CleanupStack::Pop();
                iControlView = new (ELeave) CBlueTControlView;
                CleanupStack::PushL(iControlView);
                iControlView->ConstructL();
                AddViewL(iControlView);
                CleanupStack::Pop();
                iDisplaySettingsView = new (ELeave) CBlueTDisplaySettingsView;
                CleanupStack::PushL(iDisplaySettingsView);
                iDisplaySettingsView->ConstructL();
                AddViewL(iDisplaySettingsView);
                CleanupStack::Pop();
                iBehaviourSettingsView = new (ELeave) CBlueTBehaviourSettingsView;
```

```
                    CleanupStack::PushL(iBehaviourSettingsView);
                    iBehaviourSettingsView->ConstructL();
                    AddViewL(iBehaviourSettingsView);
                    CleanupStack::Pop();
                    SetDefaultViewL(*iBrowseView);
                    iBluetoothHandler = CBluetoothHandler::NewL(this,
                    iSettings.iAutoConnectAddress, iSettings.iAutoConnectPort);
                    iSongTimer = CPeriodic::NewL(EPriorityNormal);
                    iIncomingCallListener = CIncomingCallListener::NewL(*this);
                    iIncomingCallListener->StartListening();
}

TKeyResponse CBlueTAppUi::HandleKeyEventL(const TKeyEvent& aKeyEvent, TEventCode aType)
{
if ((iTabGroup == NULL) || (aType != EEventKey))
        return EKeyWasNotConsumed;
        TInt active = iTabGroup->ActiveTabIndex();
        TInt count = iTabGroup->TabCount();
        TKeyResponse response = EKeyWasNotConsumed;
if (iPlaylistView->IsFocused())
{       response = iPlaylistView->OfferKeyEventL(aKeyEvent, aType);
        if (response == EKeyWasConsumed)
                return response;
}
if ((iDisplaySettingsView->SettingsContainer())&&(iDisplaySettingsView->SettingsContainer()->IsFocused()))
{       response = iDisplaySettingsView->SettingsContainer()->OfferKeyEventL(aKeyEvent, aType);
        if (response == EKeyWasConsumed)
                return response;
}
if ((iBehaviourSettingsView->SettingsContainer())&&(iBehaviourSettingsView->SettingsContainer()->IsFocused()))
{       response = iBehaviourSettingsView->SettingsContainer()->OfferKeyEventL(aKeyEvent, aType);
        if (response == EKeyWasConsumed)
                return response;
}
if ((iControlView->ControlContainer()) && (iControlView->ControlContainer()->IsFocused()))
{       response = iControlView->ControlContainer()->OfferKeyEventL(aKeyEvent, aType);
        if (response == EKeyWasConsumed)
                return response;
}
switch (aKeyEvent.iCode)
    {   case EKeyDevice3:
        {       if (iBrowseView->BrowseContainer())
                iBrowseView->BrowseContainer()->FileSelectedL();
        }
                break;
        case EKeyLeftArrow:
                if (iBluetoothHandler->IsActive())
                        break;
                if ((active > 0))
                {       if (iControlView->Skin().iValid)
                                active--;
                        else
                                active = 0;
                        iTabGroup->SetActiveTabByIndex(active);
                        ActivateLocalViewL(TUid::Uid(iTabGroup->TabIdFromIndex(active)));
                }
                else
                {       if (iBrowseView->BrowseContainer())
                        iBrowseView->BrowseContainer()->UpOneLevelL();
                }
                break;
        case EKeyRightArrow:
                if (iBluetoothHandler->IsActive())
                        break;
                if ((active + 1) < count)
                {       if (iControlView->Skin().iValid)
                                active++;
```

```
                        else
                                active = (count - 1);
                        iTabGroup->SetActiveTabByIndex(active);
                        if (active == 2)
                        GetPlaylist();
                        ActivateLocalViewL(TUid::Uid(iTabGroup->TabIdFromIndex(active)));
                }
                else
                {       if (iBrowseView->BrowseContainer())
                        iBrowseView->BrowseContainer()->FileSelectedL();
                }
                break;
        case EKeyUpArrow:
                if (active == 0)
                        return EKeyWasNotConsumed;
                if (iCurrentVolume < (256 - KVolumeStep))
                        iCurrentVolume += KVolumeStep;
                else
                        iCurrentVolume = 255;
                        iBluetoothHandler->RequestSetVolume(iCurrentVolume);
                        DrawControlView(KBlueTDrawVolume);
                        break;
        case EKeyDownArrow:
                if (active == 0)
                        return EKeyWasNotConsumed;
                if (iCurrentVolume >= KVolumeStep)
                        iCurrentVolume -= KVolumeStep;
                else
                        iCurrentVolume = 0;
                        iBluetoothHandler->RequestSetVolume(iCurrentVolume);
                        DrawControlView(KBlueTDrawVolume);
                        break;
        case '7':
                if (iBrowseView->BrowseContainer())
                        iBrowseView->BrowseContainer()->PlaySelectedFileL(ETrue);
                        break;
        case '3':
                if (iBrowseView->BrowseContainer())
                        iBrowseView->BrowseContainer()->FindSongL(EFalse);
                        break;
        case '6':
                if (iBrowseView->BrowseContainer())
                        iBrowseView->BrowseContainer()->FindSongL(ETrue);
                        break;
        case 63499:
                if (iBluetoothHandler->IsActive())
                        break;
                if ((active + 1) < count)
                {       if (iControlView->Skin().iValid)
                                active++;
                        else
                                active = (count - 1);
                                if (active == 2)
                                GetPlaylist();
                }
                else
                {       active = 0;
                }
                iTabGroup->SetActiveTabByIndex(active);
                ActivateLocalViewL(TUid::Uid(iTabGroup->TabIdFromIndex(active)));
                break;
        default:
                return EKeyWasNotConsumed;
        break;
    }
    return EKeyWasConsumed;
}
```

```
void CBlueTAppUi::DisplayDetailedInfoDialogL(TInt aSampleRate, TInt aBitRate, TInt aNumChannels)
{	CEikTextListBox* list = new(ELeave) CAknSinglePopupMenuStyleListBox;
	CleanupStack::PushL(list);
	CAknPopupList* popupList = CAknPopupList::NewL(list,R_AVKON_SOFTKEYS_BACK,
                                                       AknPopupLayouts::EMenuWindow);
	CleanupStack::PushL(popupList);
	list->ConstructL(popupList, CEikListBox::ELeftDownInViewRect);
	list->CreateScrollBarFrameL(ETrue);
	list->ScrollBarFrame()->SetScrollBarVisibilityL( CEikScrollBarFrame::EOff,CEikScrollBarFrame::EAuto);
	CDesCArrayFlat* items = new (ELeave) CDesCArrayFlat(5);
	CleanupStack::PushL(items);
	TBuf8<256> formatBuf8;
	TBuf<256> formatBuf;
	TBuf<256> resourceBuf;
	if ((aSampleRate > 0) && (aBitRate > 0) && (aNumChannels > 0))
	{	iEikEnv->ReadResource(resourceBuf, R_BITRATE_DISPLAY);
		formatBuf.Format(resourceBuf, aBitRate);
		items->AppendL(formatBuf);
		_LIT(KStereo, "stereo");
		_LIT(KMono, "mono");
		iEikEnv->ReadResource(resourceBuf,R_SAMPLERATE_DISPLAY);
		formatBuf.Format(resourceBuf, aSampleRate, (aNumChannels == 1) ?
		&KMono : &KStereo);
		items->AppendL(formatBuf);
	}
	iEikEnv->ReadResource(resourceBuf, R_SKIN_DISPLAY);
	formatBuf.Format(resourceBuf, &iSkinName);
	items->AppendL(formatBuf);
	iEikEnv->ReadResource(formatBuf, R_UNKNOWN_AUTHOR);
	if (iSkinAuthor != KNullDesC8)
	{	formatBuf8.Format(_L8(" (%S)"), &iSkinAuthor);
		formatBuf.Copy(formatBuf8);
	}
	items->AppendL(formatBuf);
	CTextListBoxModel* model = list->Model();
	model->SetItemTextArray(items);
	model->SetOwnershipType(ELbmOwnsItemArray);
	CleanupStack::Pop();
	list->ClearSelection();
	iEikEnv->ReadResource(resourceBuf, R_TRACK_INFO);
	popupList->SetTitleL(resourceBuf);
	popupList->ExecuteLD();
	CleanupStack::Pop();
	CleanupStack::PopAndDestroy();
}

CBlueTControlView* CBlueTAppUi::ControlView()
{
	return iControlView;
}

CBlueTBrowseView* CBlueTAppUi::BrowseView()
{
	return iBrowseView;
}

CBlueTPlaylistView* CBlueTAppUi::PlaylistView()
{
	return iPlaylistView;
}
```

BlueTBehaviourSettingsContainer.cpp

```
#include "BlueTBehaviourSettingsContainer.h"
#include "BlueTBehaviourSettingsView.h"
#include "BlueTApp.h"
#include "BlueTAppUi.h"
#include "BlueT.hrh"
#include <BlueT.rsg>
#include <eikfutil.h>
#include <gdi.h>
#include <barsread.h>
#include <aknsettingitemlist.h>

CBlueTBehaviourSettingItemList::CBlueTBehaviourSettingItemList(TBlueTSettings& aData) : iData(aData)
{
}

CAknSettingItem* CBlueTBehaviourSettingItemList::CreateSettingItemL(TInt aSettingId)
{	switch (aSettingId)
	{
	case EBlueTActionOnIncomingCallId:
		return new (ELeave) CAknEnumeratedTextPopupSettingItem (aSettingId, iData.iActionOnIncomingCall);
	case EBlueTDownloadDriveId:
		return new (ELeave) CAknEnumeratedTextPopupSettingItem (aSettingId, iData.iDownloadDrive);
	default:
	break;
	}
	return NULL;
}

void CBlueTBehaviourSettingItemList::PopulateDriveLettersL (CAknEnumeratedTextPopupSettingItem* aSettingItem)
{	TDriveList driveList;
	User::LeaveIfError(CEikonEnv::Static()->FsSession().DriveList(driveList));
	CArrayPtrFlat<CAknEnumeratedText>* enumTextArray =
		new (ELeave) CArrayPtrFlat<CAknEnumeratedText>(4);
	CleanupStack::PushL(enumTextArray);
	CArrayPtrFlat<HBufC>* poppedUpTextArray = new (ELeave) CArrayPtrFlat<HBufC>(4);
	CleanupStack::PushL(poppedUpTextArray);
	TBuf<1> driveLetter;
	driveLetter.SetMax();
	TInt count = 0;
	for (TInt i = 0; i < 25; i++)
	{	if (driveList[i])
		{	driveLetter[0] = 'A' + i;
			poppedUpTextArray->AppendL(driveLetter.AllocL());
			enumTextArray->AppendL(new (ELeave)
			CAknEnumeratedText(count, driveLetter.AllocL()));
			count++;
		}
	}
	aSettingItem->SetEnumeratedTextArrays(enumTextArray,
	poppedUpTextArray);
	CleanupStack::Pop(2);
	aSettingItem->HandleTextArrayUpdateL();
}

void CBlueTBehaviourSettingsContainer::ConstructL(CBlueTAppUi* aAppUi,CBlueTBehaviourSettingsView* aView,
                                                  const TRect& aRect)
{	iAppUi = aAppUi;
	iView = aView;
	CreateWindowL();
	SetRect(aRect);
	ActivateL();
	iListBox = new (ELeave) CBlueTBehaviourSettingItemList(iAppUi->Settings());
	iListBox->SetMopParent(this);
	iListBox->ConstructFromResourceL (R_BLUET_BEHAVIOUR_SETTING_ITEM_LIST);
	iListBox->PopulateDriveLettersL((CAknEnumeratedTextPopupSettingItem*)
		(*iListBox->SettingItemArray())[KDownloadDriveSettingIndex]);
```

```
	iListBox->LoadSettingsL();
	iListBox->SetRect(aRect);
	iListBox->ActivateL();
}

CBlueTBehaviourSettingsContainer::~CBlueTBehaviourSettingsContainer()
{	if (iListBox)
	{	TRAPD(ignore, iListBox->StoreSettingsL());
	}
	delete iListBox;
}

void CBlueTBehaviourSettingsContainer::SizeChanged()
{
	if (iListBox)
		iListBox->SetRect(Rect());
}

TInt CBlueTBehaviourSettingsContainer::CountComponentControls() const
{	if (iListBox)
		return 1;
	else
		return 0;
}

CCoeControl* CBlueTBehaviourSettingsContainer::ComponentControl(TInt aIndex) const
{	switch (aIndex)
	{
	case 0:
		return iListBox;
	default:
		return NULL;
	}
}

void CBlueTBehaviourSettingsContainer::Draw(const TRect& aRect) const
{	CWindowGc& gc = SystemGc();
	gc.SetPenStyle(CGraphicsContext::ENullPen);
	gc.SetBrushColor(KRgbGray);
	gc.SetBrushStyle(CGraphicsContext::ESolidBrush);
	gc.DrawRect(aRect);
}

void CBlueTBehaviourSettingsContainer::HandleControlEventL(CCoeControl* /*aControl*/, TCoeEvent /*aEventType*/)
{
}

TKeyResponse CBlueTBehaviourSettingsContainer::OfferKeyEventL(const TKeyEvent& aKeyEvent,
TEventCode aType)
{	return iListBox->OfferKeyEventL(aKeyEvent, aType);
}
```

BlueTBehaviourSettingsView.cpp

```
#include <aknviewappui.h>
#include <avkon.hrh>
#include <BlueT.rsg>
#include "BlueT.hrh"
#include "BlueTBehaviourSettingsView.h"
#include "BlueTBehaviourSettingsContainer.h"

void CBlueTBehaviourSettingsView::ConstructL()
{
	BaseConstructL(R_BLUET_BEHAVIOUR_SETTINGS_VIEW);
}
```

```
CBlueTBehaviourSettingsView::~CBlueTBehaviourSettingsView()
{	if (iContainer)
		AppUi()->RemoveFromViewStack(*this, iContainer);
		delete iContainer;
}

TUid CBlueTBehaviourSettingsView::Id() const
{
	return KBehaviourSettingsViewId;
}

void CBlueTBehaviourSettingsView::HandleCommandL(TInt aCommand)
{
	switch (aCommand)
	{
	case EAknSoftkeyOk:
		break;
	case EAknSoftkeyBack:
		AppUi()->HandleCommandL(EBlueTCmdExitBehaviourSettings);
		break;
	default:
		AppUi()->HandleCommandL(aCommand);
	break;
	}
}

void CBlueTBehaviourSettingsView::HandleClientRectChange()
{	if (iContainer)
		iContainer->SetRect(ClientRect());
}

void CBlueTBehaviourSettingsView::DoActivateL(const TVwsViewId& /*aPrevViewId*/, TUid
                                              /*aCustomMessageId*/,const TDesC8& /*aCustomMessage*/)
{	Cba()->SetCommandSetL(R_AVKON_SOFTKEYS_OPTIONS_BACK);
	if (!iContainer)
	{	CBlueTAppUi* appUi = (CBlueTAppUi*) AppUi();
		iContainer = new (ELeave) CBlueTBehaviourSettingsContainer;
		iContainer->SetMopParent(this);
		iContainer->ConstructL(appUi, this, ClientRect());
		AppUi()->AddToStackL(*this, iContainer);
	}
}

void CBlueTBehaviourSettingsView::DoDeactivate()
{	Cba()->SetCommandSetL(R_AVKON_SOFTKEYS_OPTIONS_EXIT);
	Cba()->DrawDeferred();
	if (iContainer)
		AppUi()->RemoveFromViewStack(*this, iContainer);
		delete iContainer;
		iContainer = NULL;
}

CBlueTBehaviourSettingsContainer* CBlueTBehaviourSettingsView::SettingsContainer()
{
	return iContainer;
}
```

BlueTBrowseContainer.cpp

```
#include <eiktxlbm.h>
#include <eiklbv.h>
#include "BlueTBrowseContainer.h"
#include "BlueTApp.h"
#include "BlueTAppUi.h"
#include <AknQueryDialog.h>
#include <AknUtils.h>
#include "BlueTControlView.h"
#include <BlueT.rsg>
#include <gdi.h>

void CBlueTBrowseContainer::ConstructL(CBlueTAppUi* aAppUi, const TRect& aRect)
{	iAppUi = aAppUi;
	CreateWindowL();
	iListItems = new (ELeave) CDesC16ArrayFlat(4);
	const CFont* font;
	switch (iAppUi->Settings().iSongBrowserFont)
	{
	case TBlueTSettings::EHuge:
		font = CEikonEnv::Static()->TitleFont();
		break;
	case TBlueTSettings::ELarge:
		font = CEikonEnv::Static()->NormalFont();
		break;
	case TBlueTSettings::EMedium:
		font = CEikonEnv::Static()->LegendFont();
		break;
	case TBlueTSettings::ESmall:
	default:
		font = CEikonEnv::Static()->DenseFont();
		break;
	}
	iListBox = new (ELeave) CBlueTListBox(font);
	iListBox->ConstructL(this, EAknListBoxLoopScrolling | CEikListBox::EPaintedSelection);
	CBlueTListItemDrawer* drawer = (CBlueTListItemDrawer*) iListBox->View()->ItemDrawer();
	drawer->SetColours(iTextForegroundColour, iTextBackgroundColour, iHighlightForegroundColour,
			iHighlightBackgroundColour);
	iListBox->Model()->SetOwnershipType(ELbmDoesNotOwnItemArray);
	iListBox->Model()->SetItemTextArray(iListItems);
	iListBox->View()->SetBackColor(iTextBackgroundColour);
	iListBox->View()->ItemDrawer()->SetBackColor(iTextBackgroundColour);
	iListBox->SetItemHeightL(KItemHeight);
	iScrollBarFrame = iListBox->CreateScrollBarFrameL(ETrue);
	iListBox->ScrollBarFrame()->SetScrollBarVisibilityL(
	CEikScrollBarFrame::EAuto, CEikScrollBarFrame::EAuto);
	SetRect(aRect);
	RefreshFileListL(iAppUi->CurrentDirectory());
	iListBox->Model()->SetItemTextArray(iListItems);
	iListBox->HandleItemAdditionL();
	iListBox->SetCurrentItemIndex(0);
	ActivateL();
}

CBlueTBrowseContainer::~CBlueTBrowseContainer()
{
	delete iListBox;
	delete iListItems;
}

void CBlueTBrowseContainer::SizeChanged()
{
	iListBox->SetRect(Rect());
}
```

```
TInt CBlueTBrowseContainer::CountComponentControls() const
{
	if (iListBox)
		return 1;
	else
		return 0;
}

CCoeControl* CBlueTBrowseContainer::ComponentControl(TInt aIndex) const
{	switch (aIndex)
	{	case 0:
		   return iListBox;
		default:
		   return NULL;
	}
}

void CBlueTBrowseContainer::Draw(const TRect& /*aRect*/) const
{
}

void CBlueTBrowseContainer::HandleControlEventL(CCoeControl* /*aControl*/, TCoeEvent /*aEventType*/)
{
}

TKeyResponse CBlueTBrowseContainer::OfferKeyEventL(const TKeyEvent& aKeyEvent,TEventCode aType)
{	if ((aKeyEvent.iCode == EKeyUpArrow) || (aKeyEvent.iCode == EKeyDownArrow))
	{	((CBlueTListItemDrawer*) iListBox->View()->ItemDrawer())->ResetScrolling();
		return iListBox->OfferKeyEventL(aKeyEvent, aType);
	}
	else if ((aKeyEvent.iScanCode == '0') && (aType == EEventKey))
	{	((CBlueTListItemDrawer*) iListBox->View()->ItemDrawer())->ResetScrolling();
		TInt currentIndex = iListBox->CurrentItemIndex();
		TInt itemsOnScreen = 6;
		if (iAppUi->Settings().iDisplayType == TBlueTSettings::EFullScreen)
		itemsOnScreen = 8;
		if (iListBox->Model()->ItemTextArray()->MdcaCount()>(currentIndex + itemsOnScreen))
		{	iListBox->SetCurrentItemIndex(currentIndex +
			itemsOnScreen);
		}
		else
		{
		iListBox->SetCurrentItemIndex(iListBox->Model()->ItemTextArray()-> MdcaCount() - 1);
		}
		iListBox->DrawNow();
		return EKeyWasConsumed;
	}
	else if ((aKeyEvent.iScanCode == '2') && (aType == EEventKey))
	{	((CBlueTListItemDrawer*) iListBox->View()->ItemDrawer())->ResetScrolling();
		TInt currentIndex = iListBox->CurrentItemIndex();
		TInt itemsOnScreen = 6;
		if (iAppUi->Settings().iDisplayType == TBlueTSettings::EFullScreen)
			itemsOnScreen = 8;
		if (currentIndex > itemsOnScreen)
		{	iListBox->SetCurrentItemIndex(currentIndex - itemsOnScreen);
		}
		else
		{	iListBox->SetCurrentItemIndex(0);
		}
		iListBox->DrawNow();
		return EKeyWasConsumed;
	}
	return EKeyWasNotConsumed;
}

void CBlueTBrowseContainer::RefreshFileListL(CDirTreeNode* aDirectory)
{	((CBlueTListItemDrawer*) iListBox->View()->ItemDrawer())->ResetScrolling();
```

```
	iAppUi->SetCurrentDirectory(aDirectory);
	iListItems->Reset();
	TBuf<256> resourceBuf;
	iAppUi->CoeEnv()->ReadResource(resourceBuf, R_UP_ITEM);
	if (aDirectory)
	{	if (aDirectory->iParent)
		iListItems->AppendL(resourceBuf);
		CDirTreeNode* node = aDirectory->iFirstChild;
		TBuf<256> displayBuf;
		TMusicFileType fileType;
		while (node)
		{
		if (!FileShouldBeDisplayed(node))
		{	node = node->iNextSibling;
			continue;
		}
		displayBuf.Copy(*node->iFileName);
		fileType = GetFileType(displayBuf);
		if ((node->iFileType == KFileTypeDir) || (node->iFileType == KFileTypeUnexpandedDir))
		{	displayBuf.Append(_L("\\"));
		}
		else
		{	if (fileType == EMp3File)
			displayBuf.Delete(displayBuf.Length() - 4, 4);
		}
		iListItems->AppendL(displayBuf);
		node = node->iNextSibling;
		}
		iListBox->Model()->SetItemTextArray(iListItems);
		iListBox->HandleItemAdditionL();
		iListBox->SetCurrentItemIndex(0);
	}
	else
	{	iListBox->Model()->SetItemTextArray(iListItems);
		iListBox->View()->DrawEmptyList(iListBox->Rect());
		iListBox->DrawNow();
	}
}

void CBlueTBrowseContainer::FileSelectedL()
{	if ((iListBox->CurrentItemIndex() == 0) && (iAppUi->CurrentDirectory()->iParent))
	{	UpOneLevelL();
		return;
	}
	CDirTreeNode* node = FindSelectedItem();
	if (!node)
		return;
	if (node->iFileType == KFileTypeDir)
	{	RefreshFileListL(node);
		iListBox->ScrollToMakeItemVisible(0);
		DrawNow();
		return;
	}
	if (node->iFileType == KFileTypeUnexpandedDir)
	{	iAppUi->GetDirectoryList(node);
		return;
	}
	TBuf8<1024> fullName;
	GetFullNodeFileName(node, fullName);
	TBool canPlay = EFalse;
	if ((iAppUi->Playlist().Count() == 0) || ((iAppUi->PlaylistIndex() == (iAppUi->Playlist().Count()-1))
	   &&((iAppUi->SongState() == ENoSongLoaded) || (iAppUi->SongState() == ESongStopped))))
	{	canPlay = ETrue;
	}
	if (canPlay)
	{	iAppUi->PlayFile(node, fullName, ETrue);	}
}
```

```
void CBlueTBrowseContainer::PlaySelectedFileL(TBool aAddToPlaylist)
{	CDirTreeNode* node = FindSelectedItem();
	if (!node)
		return;
	TBuf8<1024> fullName;
	GetFullNodeFileName(node, fullName);
	iAppUi->PlayFile(node, fullName, aAddToPlaylist);
}

void CBlueTBrowseContainer::DownloadSelectedFileL()
{	CDirTreeNode* node = FindSelectedItem();
	if (!node)
		return;
	TBuf8<1024> fullName;
	GetFullNodeFileName(node, fullName);
	iAppUi->DownloadFile(fullName);
}

CDirTreeNode* CBlueTBrowseContainer::FindSelectedItem()
{	TInt index = iListBox->CurrentItemIndex();
	if ((index < 0) || (index >= iListItems->Count()))
		return NULL;
	if (iAppUi->CurrentDirectory()->iParent)
		index--;
	CDirTreeNode* node = iAppUi->CurrentDirectory()->iFirstChild;
	for (TInt count = 0; count < index; count++)
	{	while (!FileShouldBeDisplayed(node))
			node = node->iNextSibling;
		if (node && node->iNextSibling)
			node = node->iNextSibling;
		else
			return NULL;
	}
	while (!FileShouldBeDisplayed(node))
		node = node->iNextSibling;
	return node;
}

void CBlueTBrowseContainer::GetFullNodeFileName(CDirTreeNode* aNode, TDes8& aFileName)
{	aFileName.Zero();
	if (!aNode)
		return;
	aFileName.Copy(*aNode->iFileName);
	if (!aNode->iParent)
		return;
	do
	{	aNode = aNode->iParent;
		if (*aNode->iFileName != KNullDesC8)
		{	aFileName.Insert(0, _L8("\\"));
			aFileName.Insert(0, *aNode->iFileName);
		}
	}
	while (aNode->iParent);
}

void CBlueTBrowseContainer::UpOneLevelL()
{	if (iAppUi->CurrentDirectory() && iAppUi->CurrentDirectory()->iParent)
	{	CDirTreeNode* childDirectory = iAppUi->CurrentDirectory();
		RefreshFileListL(iAppUi->CurrentDirectory()->iParent);
		((CBlueTListItemDrawer*) iListBox->View()->ItemDrawer())->ResetScrolling();
		CDirTreeNode* node = iAppUi->CurrentDirectory()->iFirstChild;
		TInt count = 0;
		while (node)
		{	if (node == childDirectory)
			{
			if (iAppUi->CurrentDirectory()->iParent)
```

```
                                        iListBox->SetCurrentItemIndex(count + 1);
                                else
                                        iListBox->SetCurrentItemIndex(count);
                                        break;
                                }
                                node = node->iNextSibling;
                                count++;
                        }
                        iListBox->ScrollToMakeItemVisible(iListBox->CurrentItemIndex());
                        DrawNow();
                }
}

TMusicFileType CBlueTBrowseContainer::GetFileType(const TDesC& aFileName)
{       TMusicFileType fileType = EUnknownFile;
        if ((aFileName.Length() > 4) && (!aFileName.Right(4).CompareF(_L(".mp3"))))
                fileType = EMp3File;
        else if ((aFileName.Length() > 4) && (!aFileName.Right(4).CompareF(_L(".m3u"))))
                fileType = EMp3File;
        else if ((aFileName.Length() > 4) && (!aFileName.Right(4).CompareF(_L(".mp2"))))
                fileType = EMusicFile;
        else if ((aFileName.Length() > 4) && (!aFileName.Right(4).CompareF(_L(".wma"))))
                fileType = EMusicFile;
        else if ((aFileName.Length() > 4) && (!aFileName.Right(4).CompareF(_L(".wav"))))
                fileType = EMusicFile;
        else if ((aFileName.Length() > 4) && (!aFileName.Right(4).CompareF(_L(".mid"))))
                fileType = EMusicFile;
        else if ((aFileName.Length() > 4) && (!aFileName.Right(4).CompareF(_L(".mod"))))
                fileType = EMusicFile;
        else if ((aFileName.Length() > 4) && (!aFileName.Right(4).CompareF(_L(".cda"))))
                fileType = EMusicFile;
        else if ((aFileName.Length() > 4) && (!aFileName.Right(4).CompareF(_L(".ogg"))))
                fileType = EMusicFile;
        return fileType;
}

void CBlueTBrowseContainer::SetListEmptyStringL(const TDesC& aEmptyString)
{       if (iListBox)
        {       iListBox->View()->SetListEmptyTextL(aEmptyString);
                DrawNow();
        }
}

void CBlueTBrowseContainer::SetColours(const TRgb& aTextForeground, const TRgb& aTextBackground, const
                                        TRgb& aHighlightForeground, const TRgb& aHighlightBackground)
{       iTextForegroundColour = aTextForeground;
        iTextBackgroundColour = aTextBackground;
        iHighlightForegroundColour = aHighlightForeground;
        iHighlightBackgroundColour = aHighlightBackground;
        if (iListBox)
        {       CBlueTListItemDrawer* drawer =(CBlueTListItemDrawer*) iListBox->View()->ItemDrawer();
                drawer->SetColours(iTextForegroundColour, iTextBackgroundColour,
                        iHighlightForegroundColour, iHighlightBackgroundColour);
                iListBox->View()->SetBackColor(iTextBackgroundColour);
        }
}

void CBlueTBrowseContainer::ScrollSelectedSong()
{       TInt index = iListBox->CurrentItemIndex();
        if ((index < 0) || (index >= iListBox->Model()->ItemTextArray()->MdcaCount()))
                return;
        if (((CBlueTListItemDrawer*) iListBox->View()->ItemDrawer())->ScrollSong(index))
                DrawNow();
}

CDirTreeNode* CBlueTBrowseContainer::RecursiveFind(CDirTreeNode* aNode, TDes8& aName)
{       while (aNode)
```

```
	{	if (!FileShouldBeDisplayed(aNode))
		{	aNode = aNode->iNextSibling;
			continue;
		}
		if (aNode->iFileName->FindF(aName) != KErrNotFound)
		{	iResultsFound++;
			if (iResultsFound > iFindResultsToSkip)
				return aNode;
		}
		if (aNode->iFirstChild)
		{	CDirTreeNode* foundNode = RecursiveFind(aNode->iFirstChild, aName);
			if (foundNode)
			{	iResultsFound++;
				if (iResultsFound > iFindResultsToSkip)
				return foundNode;
			}
		}
		aNode = aNode->iNextSibling;
	}	return NULL;
}

void CBlueTBrowseContainer::FindSongL(TBool aFindNext)
{	TFileName lastSearchedSong = iLastSearchedSong;
	CAknTextQueryDialog* dialog = NULL;
	if (aFindNext)
	{	if (iLastSearchedSong == KNullDesC)
			return;
	}
	else
	{	dialog = CAknTextQueryDialog::NewL(iLastSearchedSong);
		dialog->SetPredictiveTextInputPermitted(ETrue);
		CleanupStack::PushL(dialog);
	}
	iResultsFound = 0;
	if (aFindNext || dialog->ExecuteLD(R_FIND_DIALOG))
	{	if (iLastSearchedSong != lastSearchedSong)
			iFindResultsToSkip = 0;
			TBuf8<256> songToFind;
			songToFind.Copy(iLastSearchedSong);
			CDirTreeNode* currentNode = iAppUi->CurrentDirectory();
		while (currentNode && currentNode->iParent)
			currentNode = currentNode->iParent;
		CDirTreeNode* foundNode = RecursiveFind(currentNode, songToFind);
		if (foundNode)
		{	RefreshFileListL(foundNode->iParent);
			if (iListBox->Model()->ItemTextArray()->MdcaCount() <= 1)
			{	iListBox->DrawNow();
				iFindResultsToSkip++;
				if (!aFindNext)
				CleanupStack::Pop();
				return;
			}
			TInt nodeIndex = 0;
			if (foundNode->iParent->iParent)
				nodeIndex++;
				currentNode = foundNode->iParent->iFirstChild;
				while (currentNode && (currentNode != foundNode))
				{	if (FileShouldBeDisplayed(currentNode))
						nodeIndex++;
					currentNode = currentNode->iNextSibling;
				}
			if (nodeIndex < iListBox->Model()->ItemTextArray()->MdcaCount())
			{	((CBlueTListItemDrawer*) iListBox->View()->ItemDrawer())->ResetScrolling();
				iListBox->SetCurrentItemIndex(nodeIndex);
				iListBox->ScrollToMakeItemVisible(nodeIndex);
				iListBox->DrawNow();
			}
```

```
                        else
                        {
                                iListBox->ScrollToMakeItemVisible(0);
                        }
                        iFindResultsToSkip++;
                }
                else
                {       TBuf<256> resourceBuf;
                        iAppUi->CoeEnv()->ReadResource(resourceBuf, R_FIND_FAILURE);
                        lastSearchedSong.Format(resourceBuf, &iLastSearchedSong);
                        CEikonEnv::Static()->InfoWinL(lastSearchedSong,
                        KNullDesC);
                        iFindResultsToSkip = 0;
                }
        }
        if (!aFindNext)
                CleanupStack::Pop();
}

TBool CBlueTBrowseContainer::FileShouldBeDisplayed(const CDirTreeNode* aNode)
{       if (!aNode)
                return EFalse;
        if (aNode->iFileType != KFileTypeFile)
                return ETrue;
        TBuf<256> fileName;
        fileName.Copy(*aNode->iFileName);
        TMusicFileType fileType = GetFileType(fileName);
        if (fileType != EUnknownFile)
                return ETrue;
        return EFalse;
}
```

BlueTBrowseView.cpp

```
#include <aknviewappui.h>
#include <avkon.hrh>
#include <BlueT.rsg>
#include "BlueTBrowseView.h"
#include "BlueTAppUi.h"
#include "BlueTBrowseContainer.h"

void CBlueTBrowseView::ConstructL()
{       BaseConstructL(R_BLUET_BROWSE_VIEW);
        iTextForegroundColour = KRgbBlack;
        iTextBackgroundColour = KRgbWhite;
        iHighlightForegroundColour = KRgbBlack;
        iHighlightBackgroundColour = TRgb(0x80, 0x80, 0xFF);
}

CBlueTBrowseView::~CBlueTBrowseView()
{       if (iContainer)
        AppUi()->RemoveFromViewStack(*this, iContainer);
        delete iContainer;
}

TUid CBlueTBrowseView::Id() const
{
   return KBrowseViewId;
}

void CBlueTBrowseView::HandleCommandL(TInt aCommand)
{       switch (aCommand)
        {
        case EAknSoftkeyOk:
                break;
```

```
		case EAknSoftkeyExit:
			AppUi()->HandleCommandL(EEikCmdExit);
			break;
		default:
			AppUi()->HandleCommandL(aCommand);
		break;
		}
}

void CBlueTBrowseView::HandleClientRectChange()
{		if (iContainer)
			iContainer->SetRect(ClientRect());
}

void CBlueTBrowseView::DoActivateL(const TVwsViewId& /*aPrevViewId*/, TUid /*aCustomMessageId*/,
                                   const TDesC8& /*aCustomMessage*/)
{		if (!iContainer)
		{	iContainer = new (ELeave) CBlueTBrowseContainer;
			iContainer->SetMopParent(this);
			CBlueTAppUi* appUi = (CBlueTAppUi*) AppUi();
			iContainer->SetColours(iTextForegroundColour, iTextBackgroundColour,
                                   iHighlightForegroundColour, iHighlightBackgroundColour);
			if (appUi->Settings().iDisplayType == TBlueTSettings::EFullScreen)
				iContainer->ConstructL(appUi, TRect(0, 0, KFullScreenWidth, KFullScreenHeight));
			else
				iContainer->ConstructL(appUi, ClientRect());
			TBuf<256> resourceBuf;
			if (iListLoaded)
				appUi->CoeEnv()->ReadResource(resourceBuf, R_EMPTY_SONG_LIST);
			else
				appUi->CoeEnv()->ReadResource(resourceBuf, R_LOADING_SONG_LIST);
			iContainer->SetListEmptyStringL(resourceBuf);
			AppUi()->AddToStackL(*this, iContainer);
			iContainer->DrawNow();
		}
}

void CBlueTBrowseView::DoDeactivate()
{		if (iContainer)
		AppUi()->RemoveFromViewStack(*this, iContainer);
		delete iContainer;
		iContainer = NULL;
}

void CBlueTBrowseView::ReadSkinDescriptionL(const TDesC& aFileName)
{		iTextForegroundColour = KRgbBlack;
		iTextBackgroundColour = KRgbWhite;
		iHighlightForegroundColour = KRgbBlack;
		iHighlightBackgroundColour = TRgb(0x80, 0x80, 0xFF);
		RFile file;
		User::LeaveIfError(file.Open(CEikonEnv::Static()->FsSession(),aFileName, EFileShareAny |
                                     EFileRead));
		CleanupClosePushL(file);
		TInt fileSize;
		User::LeaveIfError(file.Size(fileSize));
		HBufC8* fileBuf = HBufC8::NewLC(fileSize);
		TPtr8 fileBufPtr = fileBuf->Des();
		User::LeaveIfError(file.Read(fileBufPtr));
		TLex8 lex(*fileBuf);
		TPtrC8 ptr(lex.NextToken());
		while (ptr != KNullDesC8)
		{	lex.SkipSpace();
			if (!ptr.CompareF(_L8("textforegroundcolour:")))
				ReadSkinColour(lex, iTextForegroundColour);
			else if (!ptr.CompareF(_L8("textbackgroundcolour:")))
				ReadSkinColour(lex, iTextBackgroundColour);
			else if (!ptr.CompareF(_L8("highlightforegroundcolour:")))
```

```
                        ReadSkinColour(lex, iHighlightForegroundColour);
                else if (!ptr.CompareF(_L8("highlightbackgroundcolour:")))
                        ReadSkinColour(lex, iHighlightBackgroundColour);
                else
                        lex.SkipSpace();
                ptr.Set(lex.NextToken());
        }
        CleanupStack::PopAndDestroy(2);
}

void CBlueTBrowseView::ReadSkinColour(TLex8& aLex, TRgb& aColour)
{       TInt colour;
        aLex.Val(colour);
        aColour.SetRed(colour);
        aLex.SkipSpace();
        aLex.Val(colour);
        aColour.SetGreen(colour);
        aLex.SkipSpace();
        aLex.Val(colour);
        aColour.SetBlue(colour);
        aLex.SkipSpace();
}

CBlueTBrowseContainer* CBlueTBrowseView::BrowseContainer()
{
        return iContainer;
}

void CBlueTBrowseView::SetListLoaded(TBool aListLoaded)
{       if (iContainer && !iListLoaded && aListLoaded)
        {
        TBuf<256> resourceBuf;
        ((CBlueTAppUi*)AppUi())->CoeEnv()->ReadResource(resourceBuf, R_EMPTY_SONG_LIST);
        iContainer->SetListEmptyStringL(resourceBuf);
        }
        iListLoaded = aListLoaded;
}
```

BluetControlContainer.cpp

```
#include "BlueTControlContainer.h"
#include "BlueTControlView.h"
#include <eikenv.h>

void CBlueTControlContainer::ConstructL(CBlueTAppUi* aAppUi, CBlueTControlView* aView, const TRect& aRect)
{       iAppUi = aAppUi;
        iView = aView;
        iTrackNameBitmap = new (ELeave) CFbsBitmap;
        User::LeaveIfError(iTrackNameBitmap->Create(iView->Skin().iTrackNameRect.Size(), EColor16M));
        iTrackNameDevice = CFbsBitmapDevice::NewL(iTrackNameBitmap);
        User::LeaveIfError(iTrackNameDevice->CreateContext(iTrackNameContext));
        iTrackTimeBitmap = new (ELeave) CFbsBitmap;
        User::LeaveIfError(iTrackTimeBitmap->Create(iView->Skin().iTrackTimeRect.Size(), EColor16M));
        iTrackTimeDevice = CFbsBitmapDevice::NewL(iTrackTimeBitmap);
        User::LeaveIfError(iTrackTimeDevice->CreateContext(iTrackTimeContext));
        CreateWindowL();
        iFocusIndex = 1;
        iHeight = aRect.Height();
        iWidth = aRect.Width();
        SetRect(aRect);
        ActivateL();
}
```

```
CBlueTControlContainer::~CBlueTControlContainer()
{	delete iTrackNameContext;
	delete iTrackNameDevice;
	delete iTrackNameBitmap;
	delete iTrackTimeContext;
	delete iTrackTimeDevice;
	delete iTrackTimeBitmap;
}

void CBlueTControlContainer::SizeChanged()
{
}

TInt CBlueTControlContainer::CountComponentControls() const
{
	return 0;
}

CCoeControl* CBlueTControlContainer::ComponentControl(TInt /*aIndex*/) const
{	return NULL;
}

void CBlueTControlContainer::AppInitiatedDraw() const
{	Window().Invalidate();
	ActivateGc();
	Window().BeginRedraw();
	DrawRequiredControls(iAppUi->DrawFlags());
	Window().EndRedraw();
	DeactivateGc();
}

void CBlueTControlContainer::DrawRequiredControls(TUint aRequired) const
{	CWindowGc& gc = SystemGc();
	if (!iView->Skin().iValid)
	{	gc.SetPenStyle(CGraphicsContext::ENullPen);
		gc.SetBrushColor(KRgbGray);
		gc.SetBrushStyle(CGraphicsContext::ESolidBrush);
		gc.DrawRect(Rect());
		iAppUi->SetDrawPending(EFalse);
		iAppUi->SetDrawFlags(KBlueTDrawEverything);
		return;
	}
	if (aRequired & KBlueTDrawBackground)
		gc.BitBlt(Rect().iTl, iView->Skin().iBitmap, Rect());
	if (aRequired & KBlueTDrawVolume)
		DrawVolume(Rect());
	if (aRequired & KBlueTDrawTrackTime)
		DrawTrackTime(Rect());
	if (aRequired & KBlueTDrawShuffle)
		DrawShuffle(Rect());
	if (aRequired & KBlueTDrawRepeat)
		DrawRepeat(Rect());
	if (aRequired & KBlueTDrawTrackName)
		DrawTrackName(Rect());
	if (aRequired & KBlueTDrawButtons)
		DrawButtons(Rect());
	iAppUi->SetDrawPending(EFalse);
	iAppUi->SetDrawFlags(KBlueTDrawEverything);
}

void CBlueTControlContainer::Draw(const TRect& /*aRect*/) const
{
	DrawRequiredControls(KBlueTDrawEverything);
}
```

```
void CBlueTControlContainer::DrawTrackName(const TRect& aRect) const
{	CWindowGc& gc = SystemGc();
	const CFont* font;
	if (iView->Skin().iTrackNameRect.Size() != iTrackNameBitmap->SizeInPixels())
	{	iTrackNameDevice->Resize(iView->Skin().iTrackNameRect.Size());
		iTrackNameContext->Resized();
	}
	TRect trackNameRect(TPoint(0, 0), iView->Skin().iTrackNameRect.Size());
	iTrackNameContext->BitBlt(TPoint(0, 0), iView->Skin().iBitmap,iView->Skin().iTrackNameRect);
	font = iView->SkinHandler()->BlueTFontToCFont(iView->Skin().iTrackNameFont);
	if (font)
		iTrackNameContext->UseFont(font);
		iTrackNameContext->SetPenColor(iView->Skin().iTrackNameColour);
	TInt baseline = iView->Skin().iTrackNameRect.Height() / 2 + font->AscentInPixels() / 2;
	TBuf<256> formatBuf;
	if ((iAppUi->SongState() == ESongPlaying) || (iAppUi->SongState() == ESongPaused))
	{	if ((!iAppUi->Settings().iUseId3Tags) && (iAppUi->Playlist().Count() >
		  iAppUi->PlaylistIndex()) &&(!iAppUi->Shuffle()))
		{	formatBuf.Copy(*(iAppUi->Playlist()[iAppUi->PlaylistIndex()]));
		if ((formatBuf.Length() > 4) && (!formatBuf.Right(4).CompareF(_L(".mp3"))))
			formatBuf.Delete(formatBuf.Length() - 4, 4);
		}
		else
		{	formatBuf.Copy(iAppUi->SongNameGuess());
		}
		TInt trackNameChars = font->TextCount(formatBuf, iView->Skin().iTrackNameRect.Width());
		if (trackNameChars == formatBuf.Length())
		{	iTrackNameContext->DrawText(formatBuf, /*iView->Skin().iTrackNameRect*/
					trackNameRect, baseline, CGraphicsContext::ECenter, 0);
		}
		else
		{	TInt offset =(iAppUi->CurrentTime() % ((formatBuf.Length() –
								trackNameChars + 10) / 2)) * 2;
			offset -= 4;
			if (offset < 0)
				offset = 0;
			else if (offset > (formatBuf.Length() - trackNameChars))
				offset = formatBuf.Length() - trackNameChars;
			TPtrC drawPtr = formatBuf.Mid(offset, trackNameChars);
			iTrackNameContext->DrawText(drawPtr, /*iView->Skin().iTrackNameRect*/
					trackNameRect, baseline, CGraphicsContext::ECenter, 0);
		}
	}
	else
	{	iTrackNameContext->DrawText(_L("BlueT"), /*iView->Skin().iTrackNameRect*/
					trackNameRect, baseline, CGraphicsContext::ECenter, 0);
	}
	iTrackNameContext->DiscardFont();
	gc.BitBlt(aRect.iTl + iView->Skin().iTrackNameRect.iTl,iTrackNameBitmap);
}

void CBlueTControlContainer::DrawTrackTime(const TRect& aRect) const
{	CWindowGc& gc = SystemGc();
	const CFont* font;
	if (iView->Skin().iTrackTimeRect.Size() != iTrackTimeBitmap->SizeInPixels())
	{	iTrackTimeDevice->Resize(iView->Skin().iTrackTimeRect.Size());
		iTrackTimeContext->Resized();
	}
	TRect trackTimeRect(TPoint(0, 0), iView->Skin().iTrackTimeRect.Size());
	iTrackTimeContext->BitBlt(TPoint(0, 0),iView->Skin().iBitmap, iView->Skin().iTrackTimeRect);
	font = iView->SkinHandler()->BlueTFontToCFont(iView->Skin().iTrackTimeFont);
	if (font)
		iTrackTimeContext->UseFont(font);
		iTrackTimeContext->SetPenColor(iView->Skin().iTrackTimeColour);
		TInt baseline = iView->Skin().iTrackTimeRect.Height() / 2 + font->AscentInPixels() / 2;
	if (((iAppUi->SongState() == ESongPlaying) || (iAppUi->SongState() == ESongPaused)) &&
		(iAppUi->SongLength() > 0) && (iAppUi->CurrentTime() >= 0))
```

```
	{	TBuf<256> formatBuf;
		if (iAppUi->Settings().iTimeDisplay == TBlueTSettings::ECountsUp)
		{
		formatBuf.Format(_L("%02d:%02d"), iAppUi->CurrentTime() / 60, iAppUi->CurrentTime() % 60);
		}
		else
		{	TInt timeLeft = iAppUi->SongLength() - iAppUi->CurrentTime();
			if (timeLeft < 0)
				formatBuf.Copy(_L("--:--"));
			else
				formatBuf.Format(_L("-%02d:%02d"), timeLeft / 60, timeLeft % 60);
		}
		iTrackTimeContext->DrawText(formatBuf, trackTimeRect, baseline, CGraphicsContext::ECenter, 0);
		}
	else
		{
		iTrackTimeContext->DrawText(_L("--:--"), trackTimeRect, baseline, CGraphicsContext::ECenter, 0);
		}
	iTrackTimeContext->DiscardFont();
	gc.BitBlt(aRect.iTl + iView->Skin().iTrackTimeRect.iTl, iTrackTimeBitmap);
}

void CBlueTControlContainer::DrawButtons(const TRect& aRect) const
{	CWindowGc& gc = SystemGc();
	TInt buttonWidth = iView->Skin().iButtonsRect.Width() / KNumButtons;
	TInt buttonHeight = iView->Skin().iButtonsRect.Height();
	if (!(iAppUi->DrawFlags() & KBlueTDrawBackground))
	{
	gc.BitBlt(aRect.iTl + iView->Skin().iButtonsRect.iTl, iView->Skin().iBitmap, iView->Skin().iButtonsRect);
	}
	if (iFocusIndex > -1)
	{	if (iButtonPressed)
		{	gc.BitBlt(aRect.iTl + iView->Skin().iButtonsRect.iTl + TPoint(iFocusIndex *
			buttonWidth, 0),iView->Skin().iBitmap,TRect(iFocusIndex * buttonWidth, iHeight +
			buttonHeight, (iFocusIndex + 1) * buttonWidth, iHeight +   (buttonHeight * 2)));
		}
		else
		{	gc.BitBlt(aRect.iTl + iView->Skin().iButtonsRect.iTl + TPoint(iFocusIndex *
			buttonWidth, 0),iView->Skin().iBitmap,TRect(iFocusIndex * buttonWidth, iHeight,
			(iFocusIndex + 1) * buttonWidth, iHeight + buttonHeight));
		}
	}
}

void CBlueTControlContainer::DrawShuffle(const TRect& aRect) const
{	CWindowGc& gc = SystemGc();
	TInt buttonHeight = iView->Skin().iButtonsRect.Height();
	if (iAppUi->Shuffle())
	{	gc.BitBlt(aRect.iTl + iView->Skin().iShuffleRect.iTl, iView->Skin().iBitmap,
		TRect((iView->Skin().iVolumeSliderSize.iWidth * 2) + iView->Skin().iRepeatRect.Width(),
		iHeight + (buttonHeight * 2), (iView->Skin().iVolumeSliderSize.iWidth * 2) +
		iView->Skin().iRepeatRect.Width() + iView->Skin().iShuffleRect.Width(), iHeight +
		(buttonHeight * 2) + iView->Skin().iShuffleRect.Height()));
	}
	else if (!(iAppUi->DrawFlags() & KBlueTDrawBackground))
	{	gc.BitBlt(aRect.iTl + iView->Skin().iShuffleRect.iTl,iView->Skin().iBitmap,
			iView->Skin().iShuffleRect);
	}
}

void CBlueTControlContainer::DrawRepeat(const TRect& aRect) const
{	CWindowGc& gc = SystemGc();
	TInt buttonHeight = iView->Skin().iButtonsRect.Height();
	if (iAppUi->Repeat())
	{	gc.BitBlt(aRect.iTl + iView->Skin().iRepeatRect.iTl, iView->Skin().iBitmap,
		TRect(iView->Skin().iVolumeSliderSize.iWidth * 2, iHeight +(buttonHeight * 2),
		(iView->Skin().iVolumeSliderSize.iWidth * 2) + iView->Skin().iRepeatRect.Width(),
```

```
                iHeight + (buttonHeight * 2) + iView->Skin().iRepeatRect.Height()));
        }
        else if (!(iAppUi->DrawFlags() & KBlueTDrawBackground))
        {       gc.BitBlt(aRect.iTl + iView->Skin().iRepeatRect.iTl, iView->Skin().iBitmap,
                        iView->Skin().iRepeatRect);
        }
}

void CBlueTControlContainer::DrawVolume(const TRect& aRect) const
{       CWindowGc& gc = SystemGc();
        if (!(iAppUi->DrawFlags() & KBlueTDrawBackground))
        {       TRect maxVolumeRect;
                GetVolumeSliderArea(maxVolumeRect);
                gc.BitBlt(aRect.iTl + maxVolumeRect.iTl, iView->Skin().iBitmap, maxVolumeRect);
        }
        TInt buttonHeight = iView->Skin().iButtonsRect.Height();
        TPoint volumeDistance = iView->Skin().iVolumeRect.iBr - iView->Skin().iVolumeRect.iTl;
        TInt volumeX = iView->Skin().iVolumeRect.iBr.iX - (volumeDistance.iX * iAppUi->CurrentVolume()) / 255;
        TInt volumeY = iView->Skin().iVolumeRect.iBr.iY - (volumeDistance.iY * iAppUi->CurrentVolume()) / 255;
        for (TInt y = 0; y < iView->Skin().iVolumeSliderSize.iHeight; y++)
        {       for (TInt x = 0; x < iView->Skin().iVolumeSliderSize.iWidth; x++)
                {       TRgb colour;
                        iView->Skin().iBitmap->GetPixel(colour,TPoint(x + iView->Skin().iVolumeSliderSize.
                        iWidth, y + iHeight + (buttonHeight * 2)));
                        if (colour != KRgbBlack)
                        {       gc.BitBlt(aRect.iTl + TPoint(volumeX + x, volumeY + y),
                                iView->Skin().iBitmap,TRect(x, y + iHeight + (buttonHeight * 2),
                                x + 1, y + iHeight + 1 + (buttonHeight * 2)));
                        }
                }
        }
}

void CBlueTControlContainer::HandleControlEventL(CCoeControl* /*aControl*/, TCoeEvent aEventType)
{
}

TKeyResponse CBlueTControlContainer::OfferKeyEventL(const TKeyEvent& aKeyEvent, TEventCode aType)
{       if (!IsFocused())
                return EKeyWasNotConsumed;
        switch (aKeyEvent.iScanCode)
        {       case EStdKeyDevice3:
                        if (aType == EEventKeyDown)
                        {       iButtonPressed = ETrue;
                        }
                        else if (aType == EEventKeyUp)
                        {       if (iCheckedButtonPressedAlready)
                                {
                                iCheckedButtonPressedAlready = EFalse;
                                }
                                else if (iButtonPressed)
                                {
                                ExecuteCommand(iFocusIndex);
                                }
                                iButtonPressed = EFalse;
                        }
                        iAppUi->DrawControlView(KBlueTDrawButtons);
                        return EKeyWasConsumed;
                        break;
                default:
                break;
        }
        switch (aKeyEvent.iCode)
        {       case EKeyRightArrow:
                        if (iFocusIndex < (KNumButtons-1))
                        {       iFocusIndex++;
                                iAppUi->DrawControlView(KBlueTDrawButtons);
```

```
                                        return EKeyWasConsumed;
                                }
                                break;

                        case EKeyLeftArrow:
                                if (iFocusIndex > 0)
                                {
                                iFocusIndex--;
                                iAppUi->DrawControlView(KBlueTDrawButtons);
                                return EKeyWasConsumed;
                                }
                                break;
                        default:
                        break;
                }
                if ((aKeyEvent.iCode >= '1') && (aKeyEvent.iCode <= (KNumButtons + '0')))
                {       iFocusIndex = aKeyEvent.iCode - '1';
                        iAppUi->DrawControlView(KBlueTDrawButtons);
                        ExecuteCommand(iFocusIndex);
                        return EKeyWasConsumed;
                }
                return EKeyWasNotConsumed;
}

void CBlueTControlContainer::ExecuteCommand(TInt aIndex)
{       switch (aIndex)
        {       case 0:
                        if (iAppUi->PlaylistIndex() > 0)
                        iAppUi->PlaylistIndex()--;
                        iAppUi->BluetoothHandler()->RequestPreviousTrack();
                        break;
                case 1:
                        iAppUi->BluetoothHandler()->RequestPlay();
                        break;
                case 2:
                        iAppUi->BluetoothHandler()->RequestPause();
                        break;
                case 3:
                        iAppUi->BluetoothHandler()->RequestStop();
                        break;
                case 4:
                        if (iAppUi->PlaylistIndex() < (iAppUi->Playlist().Count()-1))
                        iAppUi->PlaylistIndex()++;
                        iAppUi->BluetoothHandler()->RequestNextTrack();
                        break;
                default:
                        break;
        }
}

TBool CBlueTControlContainer::CheckFastForwardAndRewind()
{       if (iButtonPressed)
        {       if (iCheckedButtonPressedAlready)
                {       if (iFocusIndex == 0)
                        {       iAppUi->BluetoothHandler()->RequestRewind();
                                return ETrue;
                        }
                        else if (iFocusIndex == 4)
                        {       iAppUi->BluetoothHandler()->RequestFastForward();
                                return ETrue;
                        }
                }
                else
                {       if ((iFocusIndex == 0) || (iFocusIndex == 4))
                        iCheckedButtonPressedAlready = ETrue;
                }
        }
```

```
	else
	{	iCheckedButtonPressedAlready = EFalse;
	}
	return EFalse;
}

void CBlueTControlContainer::GetVolumeSliderArea(TRect& aVolumeSliderArea) const
{	if (iView->Skin().iVolumeRect.iTl.iX < iView->Skin().iVolumeRect.iBr.iX)
	{	aVolumeSliderArea.iTl.iX = iView->Skin().iVolumeRect.iTl.iX;
		aVolumeSliderArea.iBr.iX = iView->Skin().iVolumeRect.iBr.iX;
	}
	else
	{	aVolumeSliderArea.iTl.iX = iView->Skin().iVolumeRect.iBr.iX;
		aVolumeSliderArea.iBr.iX = iView->Skin().iVolumeRect.iTl.iX;
	}
	if (iView->Skin().iVolumeRect.iTl.iY < iView->Skin().iVolumeRect.iBr.iY)
	{	aVolumeSliderArea.iTl.iY = iView->Skin().iVolumeRect.iTl.iY;
		aVolumeSliderArea.iBr.iY = iView->Skin().iVolumeRect.iBr.iY;
	}
	else
	{	aVolumeSliderArea.iTl.iY = iView->Skin().iVolumeRect.iBr.iY;
		aVolumeSliderArea.iBr.iY = iView->Skin().iVolumeRect.iTl.iY;
	}
	aVolumeSliderArea.iBr.iX += iView->Skin().iVolumeSliderSize.iWidth;
	aVolumeSliderArea.iBr.iY += iView->Skin().iVolumeSliderSize.iHeight;
}
```

BluetControlView.cpp

```
#include <aknviewappui.h>
#include <avkon.hrh>
#include <BlueT.rsg>
#include "BlueTBrowseView.h"
#include "BlueTControlView.h"
#include "BlueTPlaylistView.h"
#include "BlueTControlContainer.h"
#include "BlueTAppUi.h"
#include "BlueTSkinHandler.h"

void CBlueTControlView::ConstructL()
{
	BaseConstructL(R_BLUET_CONTROL_VIEW);
	iSkinHandler = CBlueTSkinHandler::NewL((CBlueTAppUi*) AppUi(), iSkin);
}

CBlueTControlView::~CBlueTControlView()
{	if (iContainer)
		AppUi()->RemoveFromViewStack(*this, iContainer);
		delete iSkinHandler;
		delete iContainer;
}

TUid CBlueTControlView::Id() const
{
	return KControlViewId;
}

void CBlueTControlView::HandleCommandL(TInt aCommand)
{
	switch (aCommand)
	{	case EAknSoftkeyOk:
			break;
		case EAknSoftkeyExit:
			AppUi()->HandleCommandL(EEikCmdExit);
			break;
```

```
                default:
                        AppUi()->HandleCommandL(aCommand);
                 break;
        }
}

void CBlueTControlView::HandleClientRectChange()
{       if (iContainer)
        {   iContainer->SetRect(ClientRect());
        }
}

void CBlueTControlView::DoActivateL(const TVwsViewId& /*aPrevViewId*/,TUid /*aCustomMessageId*/,
                                    const TDesC8& /*aCustomMessage*/)
{       if (!iContainer)
        {       iContainer = new (ELeave) CBlueTControlContainer;
                iContainer->SetMopParent(this);
                CBlueTAppUi* appUi = (CBlueTAppUi*) AppUi();
                if (appUi->Settings().iDisplayType == TBlueTSettings::EFullScreen)
                   iContainer->ConstructL(appUi, this, TRect(0, 0, KFullScreenWidth, KFullScreenHeight));
                else
                   iContainer->ConstructL(appUi, this, ClientRect());
                 AppUi()->AddToStackL(*this, iContainer);
        }
}

void CBlueTControlView::DoDeactivate()
{       if (iContainer)
        AppUi()->RemoveFromViewStack(*this, iContainer);
        delete iContainer;
        iContainer = NULL;
}

CBlueTControlContainer* CBlueTControlView::ControlContainer()
{
        return iContainer;
}

CBlueTSkinHandler* CBlueTControlView::SkinHandler()
{
        return iSkinHandler;
}

TBlueTSkin& CBlueTControlView::Skin()
{
        return iSkin;
}
```

BlueTDisplaySettingsContainer.cpp

```
#include "BlueTDisplaySettingsContainer.h"
#include "BlueTDisplaySettingsView.h"
#include "BlueTApp.h"
#include "BlueTAppUi.h"
#include "BlueT.hrh"
#include <BlueT.rsg>
#include <eikfutil.h>
#include <gdi.h>
#include <barsread.h>
#include <aknsettingitemlist.h>
#include <msvapi.h>

CBlueTDisplaySettingItemList::CBlueTDisplaySettingItemList(TBlueTSettings&          aData): iData(aData)
{
}
```

```
CAknSettingItem* CBlueTDisplaySettingItemList::CreateSettingItemL(TInt aSettingId)
{	switch (aSettingId)
	{
	case EBlueTBrowserFontId:
		return new (ELeave) CAknEnumeratedTextPopupSettingItem(aSettingId, iData.iSongBrowserFont);
	case EBlueTDisplayTypeId:
		return new (ELeave) CAknEnumeratedTextPopupSettingItem(aSettingId, iData.iDisplayType);
	case EBlueTTimeDisplayId:
		return new (ELeave) CAknEnumeratedTextPopupSettingItem(aSettingId, iData.iTimeDisplay);
	case EBlueTSkinFileNameId:
		return new (ELeave) CAknEnumeratedTextPopupSettingItem(aSettingId, iData.iSkinFileIndex);
	case EBlueTInstallSkinId:
		return new (ELeave) CAknEnumeratedTextPopupSettingItem(aSettingId, iData.iInstallSkinIndex);
	case EBlueTUninstallSkinId:
		return new (ELeave) CAknEnumeratedTextPopupSettingItem(aSettingId, iData.iUninstallSkinIndex);
	case EBlueTUseId3TagsId:
		return new (ELeave) CAknEnumeratedTextPopupSettingItem(aSettingId, iData.iUseId3Tags);
	default:
		break;
	}
	return NULL;
}

void CBlueTDisplaySettingsContainer::GetSkinListL (CAknEnumeratedTextPopupSettingItem* aSettingItem,
                                                   TBool aAddSelectItem)
{	CDir* entryList;
	TFileName skinPath;
	TInt ret;
	if (iAppUi->Settings().iDisplayType == TBlueTSettings::EInWindow)
		skinPath.Format(_L("%S\\%S"), &iAppUi->SkinPathL(), &KMatchInWindowSkinBitmaps);
	else
		skinPath.Format(_L("%S\\%S"), &iAppUi->SkinPathL(), &KMatchFullScreenSkinBitmaps);
	ret = CEikonEnv::Static()->FsSession().GetDir(skinPath, KEntryAttNormal,ESortByName, entryList);
	if (ret != KErrNone)
	{	delete entryList;
		TBuf<256> formatBuf;
		formatBuf.Format(_L("Couldn't get list of skins: %d"), ret);
		CEikonEnv::Static()->InfoWinL(formatBuf, KNullDesC);
		return;
	}
	TBuf<256> formatBuf;
	TBuf<256> resourceBuf;
	CEikonEnv::Static()->ReadResource(resourceBuf, R_SELECT_SKIN_MSG);
	CleanupStack::PushL(entryList);
	CArrayPtrFlat<CAknEnumeratedText>* enumTextArray = new (ELeave)
	                                    CArrayPtrFlat<CAknEnumeratedText>(4);
	CleanupStack::PushL(enumTextArray);
	CArrayPtrFlat<HBufC>* poppedUpTextArray = new (ELeave)	CArrayPtrFlat<HBufC>(4);
	CleanupStack::PushL(poppedUpTextArray);
	if (aAddSelectItem)
	{	poppedUpTextArray->AppendL(resourceBuf.AllocL());
		enumTextArray->AppendL(new (ELeave) CAknEnumeratedText(0, resourceBuf.AllocL()));
	}
	TInt entryCount = entryList->Count();
	for (TInt i = 0; i < entryCount; i++)
	{	formatBuf.Copy((*entryList)[i].iName.Left((*entryList)[i].iName.Length() -7));
		poppedUpTextArray->AppendL(formatBuf.AllocL());
		CAknEnumeratedText* enumText = new (ELeave)
		CAknEnumeratedText(i, formatBuf.AllocL());
		CleanupStack::PushL(enumText);
		enumTextArray->AppendL(enumText);
	}
	aSettingItem->SetEnumeratedTextArrays(enumTextArray,
	poppedUpTextArray);
	CleanupStack::Pop(entryCount + 2);
	if (entryCount <= iAppUi->Settings().iSkinFileIndex)
		iAppUi->Settings().iSkinFileIndex = 0;
```

```
	if (entryCount <= iAppUi->Settings().iUninstallSkinIndex)
		iAppUi->Settings().iUninstallSkinIndex = 0;
	aSettingItem->HandleTextArrayUpdateL();
	CleanupStack::PopAndDestroy();
}

void CBlueTDisplaySettingsContainer::GetInboxSkinListL (CAknEnumeratedTextPopupSettingItem* aSettingItem)
{	TBuf<256> resourceBuf;
	CEikonEnv::Static()->ReadResource(resourceBuf, R_SELECT_SKIN_MSG);
	iInboxSkinBitmapFileNames->Reset();
	iInboxSkinDescriptionFileNames->Reset();
	CArrayPtrFlat<CAknEnumeratedText>* enumTextArray =
		new (ELeave) CArrayPtrFlat<CAknEnumeratedText>(4);
	CleanupStack::PushL(enumTextArray);
	CArrayPtrFlat<HBufC>* poppedUpTextArray = new (ELeave)
		CArrayPtrFlat<HBufC>(4);
	CleanupStack::PushL(poppedUpTextArray);
	poppedUpTextArray->AppendL(resourceBuf.AllocL());
	enumTextArray->AppendL(new (ELeave) CAknEnumeratedText(0,
	resourceBuf.AllocL()));
	TInt msgStoreDrv = MessageServer::CurrentDriveL(CEikonEnv::Static()->FsSession());
	TBuf<256> inboxPathAndDrive;
	inboxPathAndDrive.Format(_L("%C:%S"), 'A'+(msgStoreDrv-EDriveA), &KInboxPath);
	TBuf<256> totalPath;
	totalPath.Format(_L("%S\\%S"), &inboxPathAndDrive, &KMatchAll);
	TEntryArray* entryArray1 = new (ELeave) TEntryArray; CleanupStack::PushL(entryArray1);
	TEntryArray* entryArray2 = new (ELeave) TEntryArray; CleanupStack::PushL(entryArray2);
	TEntryArray* entryArray3 = new (ELeave) TEntryArray; CleanupStack::PushL(entryArray3);
	RDir dir;
	User::LeaveIfError(dir.Open(CEikonEnv::Static()->FsSession(), totalPath, KEntryAttDir));
	dir.Read(*entryArray1);
	dir.Close();
	TBuf<256> formatBuf;
	TInt totalFileCount = 0;
	for (TInt i = 0; i < entryArray1->Count(); i++)
	{	totalPath.Format(_L("%S\\%S\\%S"), &inboxPathAndDrive, &(*entryArray1)[i].iName, &KMatchAll);
		User::LeaveIfError(dir.Open(CEikonEnv::Static()->FsSession(), totalPath, KEntryAttDir));
		dir.Read(*entryArray2);
		dir.Close();
		for (TInt j = 0; j < entryArray2->Count(); j++)
		{	if (iAppUi->Settings().iDisplayType == TBlueTSettings::EInWindow)
			{	totalPath.Format(_L("%S\\%S\\%S\\%S"), &inboxPathAndDrive,
				&(*entryArray1)[i].iName, &(*entryArray2)[j].iName, &KMatchInWindowSkins);
			}
			else
			{	totalPath.Format(_L("%S\\%S\\%S\\%S"), &inboxPathAndDrive,
				&(*entryArray1)[i].iName, &(*entryArray2)[j].iName, &KMatchFullScreenSkins);
			}
			TInt rc = dir.Open(CEikonEnv::Static()->FsSession(), totalPath, KEntryAttDir);
			if (rc != KErrNone)
				continue;
			dir.Read(*entryArray3);
			dir.Close();
			for (TInt k = 0; k < entryArray3->Count(); k++)
			{	totalPath.Format(_L("%S\\%S\\%S\\%S"), &inboxPathAndDrive,
				&(*entryArray1)[i].iName,&(*entryArray2)[j].iName, (*entryArray3)[k].iName);
				if (!(*entryArray3)[k].iName.Right(3).CompareF(_L("png")))
				{	iInboxSkinBitmapFileNames->AppendL(totalPath);
					formatBuf.Copy((*entryArray3)[k].iName.Left((*entryArray3)
							[k].iName.Length() - 7));
					poppedUpTextArray->AppendL(formatBuf.AllocL());
					CAknEnumeratedText* enumText = new (ELeave)
					CAknEnumeratedText(totalFileCount + 1, formatBuf.AllocL());
					CleanupStack::PushL(enumText);
					enumTextArray->AppendL(enumText);
					totalFileCount++;
				}
```

```
                                        else if
                                        (!(*entryArray3)[k].iName.Right(3).CompareF(_L("txt")))
                                        {       iInboxSkinDescriptionFileNames->AppendL(totalPath);
                                        }
                                }
                        }
                }
                aSettingItem->SetEnumeratedTextArrays(enumTextArray, poppedUpTextArray);
                if (totalFileCount <= iAppUi->Settings().iInstallSkinIndex)
                        iAppUi->Settings().iInstallSkinIndex = 0;
                aSettingItem->HandleTextArrayUpdateL();
                CleanupStack::Pop(totalFileCount);
                CleanupStack::PopAndDestroy(3);
                CleanupStack::Pop(2);
}

void CBlueTDisplaySettingsContainer::ConstructL(CBlueTAppUi* aAppUi, CBlueTDisplaySettingsView* aView,
                                                const TRect& aRect)
{       iAppUi = aAppUi;
        iView = aView;
        iInboxSkinBitmapFileNames = new (ELeave) CDesC16ArrayFlat(2);
        iInboxSkinDescriptionFileNames = new (ELeave) CDesC16ArrayFlat(2);
        CreateWindowL();
        SetRect(aRect);
        ActivateL();
        iListBox = new (ELeave) CBlueTDisplaySettingItemList(iAppUi->Settings());
        iListBox->SetMopParent(this);
        iListBox->ConstructFromResourceL(R_BLUET_DISPLAY_SETTING_ ITEM_LIST);
        GetSkinListL((CAknEnumeratedTextPopupSettingItem*)
                (*iListBox->SettingItemArray())[KSkinSettingIndex], EFalse);
        GetSkinListL((CAknEnumeratedTextPopupSettingItem*)
                (*iListBox->SettingItemArray())[KUninstallSkinSettingIndex], ETrue);
        GetInboxSkinListL((CAknEnumeratedTextPopupSettingItem*)
                (*iListBox->SettingItemArray())[KInstallSkinSettingIndex]);
        iListBox->LoadSettingsL();
        iListBox->SetRect(aRect);
        iListBox->ActivateL();
}

CBlueTDisplaySettingsContainer::~CBlueTDisplaySettingsContainer()
{       if (iListBox)
        {       TRAPD(ignore, iListBox->StoreSettingsL());
        }
        delete iInboxSkinBitmapFileNames;
        delete iInboxSkinDescriptionFileNames;
        delete iListBox;
}

void CBlueTDisplaySettingsContainer::SizeChanged()
{       if (iListBox)
                iListBox->SetRect(Rect());    }

TInt CBlueTDisplaySettingsContainer::CountComponentControls() const
{       if (iListBox)
                return 1;
        else
                return 0;
}

CCoeControl* CBlueTDisplaySettingsContainer::ComponentControl(TInt aIndex) const
{       switch (aIndex)
        {       case 0:
                        return iListBox;
                default:
                        return NULL;
        }
}
```

```
void CBlueTDisplaySettingsContainer::Draw(const TRect& aRect) const
{	CWindowGc& gc = SystemGc();
	gc.SetPenStyle(CGraphicsContext::ENullPen);
	gc.SetBrushColor(KRgbGray);
	gc.SetBrushStyle(CGraphicsContext::ESolidBrush);
	gc.DrawRect(aRect);
}

void CBlueTDisplaySettingsContainer::HandleControlEventL(CCoeControl* /*aControl*/, TCoeEvent /*aEventType*/)
{
}

TKeyResponse CBlueTDisplaySettingsContainer::OfferKeyEventL(const TKeyEvent& aKeyEvent,TEventCode aType)
{	TBuf<256> resourceBuf;
	CEikonEnv::Static()->ReadResource(resourceBuf, R_SELECT_SKIN_MSG);
	if (iListBox)
	{	if ((aKeyEvent.iCode == EKeyUpArrow) || (aKeyEvent.iCode == EKeyDownArrow) ||
		    (aKeyEvent.iCode == EKeyDevice3))
		{	TKeyResponse response = iListBox->OfferKeyEventL(aKeyEvent, aType);
			if ((response == EKeyWasConsumed) && (aKeyEvent.iCode == EKeyDevice3))
			{
			if (iListBox->ListBox()->CurrentItemIndex() == 0)
			{	iListBox->StoreSettingsL();
				GetSkinListL((CAknEnumeratedTextPopupSettingItem*)
				(*iListBox->SettingItemArray())[KSkinSettingIndex], EFalse);
				GetSkinListL((CAknEnumeratedTextPopupSettingItem*)
				(*iListBox->SettingItemArray())[KUninstallSkinSettingIndex], ETrue);
				GetInboxSkinListL((CAknEnumeratedTextPopupSettingItem*)
				(*iListBox->SettingItemArray())[KInstallSkinSettingIndex]);
				iListBox->LoadSettingsL();
			}
			else if (iListBox->ListBox()->CurrentItemIndex() == KInstallSkinSettingIndex)
			{	iListBox->StoreSettingsL();
				CAknEnumeratedTextPopupSettingItem* settingItem =
					(CAknEnumeratedTextPopupSettingItem*)
					(*iListBox->SettingItemArray())[KInstallSkinSettingIndex];
				const TDesC& settingText = settingItem->SettingTextL();
				if (settingText != resourceBuf)
				{	TBuf<256> formatBuf;
					for (TInt i = 0; i < iInboxSkinBitmapFileNames->Count(); i++)
					{
					TInt lastBackslash = (*iInboxSkinBitmapFileNames)[i].LocateReverse('\\');
					formatBuf.Copy((*iInboxSkinBitmapFileNames)[i].Mid(lastBackslash + 1,
						(*iInboxSkinBitmapFileNames)[i]. Length() - lastBackslash - 8));
					if (!formatBuf.CompareF(settingText))
					{
					for (TInt j = 0; j < iInboxSkinDescriptionFileNames->Count(); j++)
					{
					lastBackslash =(*iInboxSkinDescriptionFileNames)[j].LocateReverse('\\');
					formatBuf.Copy((*iInboxSkinDescriptionFileNames)[j].Mid(lastBackslash
					+ 1,(*iInboxSkinDescriptionFileNames)[j].Length() - lastBackslash - 8));
					if (!formatBuf.CompareF(settingText))
					{
					InstallSkin((*iInboxSkinBitmapFileNames)[i],
						(*iInboxSkinDescriptionFileNames)[j]); iListBox->ListBox()->
						SetCurrentItemIndexAndDraw(1);
					GetSkinListL((CAknEnumeratedTextPopupSettingItem*)
					(*iListBox->SettingItemArray())[KSkinSettingIndex], EFalse);
					GetSkinListL((CAknEnumeratedTextPopupSettingItem*)
					(*iListBox->SettingItemArray())[KUninstallSkinSettingIndex], ETrue);
					iListBox->LoadSettingsL();
					return response;
					}	}	}	}
				}
			}
```

```
                                else if (iListBox->ListBox()->CurrentItemIndex()==KUninstallSkinSettingIndex)
                                {
                                iListBox->StoreSettingsL();
                                CAknEnumeratedTextPopupSettingItem* settingItem =
                                (CAknEnumeratedTextPopupSettingItem*) (*iListBox->SettingItemArray())
                                [KUninstallSkinSettingIndex];
                                const TDesC& settingText = settingItem->SettingTextL();
                                if (settingText != resourceBuf)
                                {       TBuf<256> resourceBuf;
                                        iAppUi->CoeEnv()->ReadResource(resourceBuf,R_UNINSTALL_SKIN_QUERY);
                                        TBuf<256> formatBuf;
                                        formatBuf.Format(resourceBuf, &settingText);
                                        if (CEikonEnv::Static()->QueryWinL(formatBuf, KNullDesC))
                                        {
                                        if (iAppUi->Settings().iDisplayType == TBlueTSettings::EFullScreen)
                                        {
                                        formatBuf.Format(_L("%S\\%S_fs.png"), &iAppUi->SkinPathL(), &settingText);
                                        CEikonEnv::Static()->FsSession().Delete(formatBuf);
                                        formatBuf.Format(_L("%S\\%S_fs.txt"), &iAppUi->SkinPathL(), &settingText);
                                        CEikonEnv::Static()->FsSession().Delete(formatBuf);
                                        }
                                        else
                                        {
                                        formatBuf.Format(_L("%S\\%S_iw.png"), &iAppUi->SkinPathL(), &settingText);
                                        CEikonEnv::Static()->FsSession().Delete(formatBuf);
                                        formatBuf.Format(_L("%S\\%S_iw.txt"), &iAppUi->SkinPathL(), &settingText);
                                        CEikonEnv::Static()->FsSession().Delete(formatBuf);
                                        }
                                        GetSkinListL((CAknEnumeratedTextPopupSettingItem*)
                                        (*iListBox->SettingItemArray())[KSkinSettingIndex], EFalse);
                                        GetSkinListL((CAknEnumeratedTextPopupSettingItem*)
                                        (*iListBox->SettingItemArray())[KUninstallSkinSettingIndex], ETrue);
                                        }
                                        else
                                        {
                                        iAppUi->Settings().iUninstallSkinIndex = 0;
                                        }
                                        iListBox->LoadSettingsL();
                                }       }
                        }
                return response;
                        }
                }
                return EKeyWasNotConsumed;
}

void CBlueTDisplaySettingsContainer::InstallSkin(const TDesC& aBitmapFileName,const TDesC& aDescriptionFileName)
{       TInt lastBackslash = aDescriptionFileName.LocateReverse('\\');
        TPtrC filePath = aDescriptionFileName.Left(lastBackslash);
        TPtrC nameWithoutExtension = aDescriptionFileName.Mid(lastBackslash + 1, aDescriptionFileName.Length() -
                                                                lastBackslash - 5);
        TBuf<256> sourceBuf;
        TBuf<256> destBuf;
        TBuf<256> resourceBuf;
        sourceBuf.Format(_L("%S\\%S.txt"), &filePath, &nameWithoutExtension);
        destBuf.Format(_L("%S\\%S.txt"), &iAppUi->SkinPathL(), &nameWithoutExtension);
        TInt ret = EikFileUtils::CopyFile(sourceBuf, destBuf);
        if (ret != KErrNone)
        {       iAppUi->CoeEnv()->ReadResource(resourceBuf,R_COPY_SKIN_DESCRIPTION_ERROR);
                CEikonEnv::Static()->InfoWinL(resourceBuf, KNullDesC);
                return;
        }
        lastBackslash = aBitmapFileName.LocateReverse('\\');
        filePath.Set(aBitmapFileName.Left(lastBackslash));
        nameWithoutExtension.Set(aBitmapFileName.Mid(lastBackslash + 1, aBitmapFileName.Length() -
                                lastBackslash - 5));
```

```
		sourceBuf.Format(_L("%S\\%S.png"), &filePath, &nameWithoutExtension);
		destBuf.Format(_L("%S\\%S.png"), &iAppUi->SkinPathL(), &nameWithoutExtension);
		ret = EikFileUtils::CopyFile(sourceBuf, destBuf);
		if (ret != KErrNone)
		{	iAppUi->CoeEnv()->ReadResource(resourceBuf, R_COPY_SKIN_BITMAP_ERROR);
			CEikonEnv::Static()->InfoWinL(resourceBuf, KNullDesC);
			destBuf.Format(_L("%S\\%S.txt"), &iAppUi->SkinPathL(), &nameWithoutExtension);
			EikFileUtils::DeleteFile(destBuf);
			return;
		}
		iAppUi->CoeEnv()->ReadResource(resourceBuf, R_SKIN_INSTALLED_MSG);
		CEikonEnv::Static()->InfoWinL(resourceBuf, KNullDesC);
}
```

BlueTDisplaySettingsView.cpp

```
#include <aknviewappui.h>
#include <avkon.hrh>
#include <BlueT.rsg>
#include "BlueT.hrh"
#include "BlueTDisplaySettingsView.h"
#include "BlueTDisplaySettingsContainer.h"

void CBlueTDisplaySettingsView::ConstructL()
{
	BaseConstructL(R_BLUET_DISPLAY_SETTINGS_VIEW);
}

CBlueTDisplaySettingsView::~CBlueTDisplaySettingsView()
{	if (iContainer)
	AppUi()->RemoveFromViewStack(*this, iContainer);
	delete iContainer;
}

TUid CBlueTDisplaySettingsView::Id() const
{
	return KDisplaySettingsViewId;
}

void CBlueTDisplaySettingsView::HandleCommandL(TInt aCommand)
{	switch (aCommand)
	{	case EAknSoftkeyOk:
		break;
		case EAknSoftkeyBack:
		iContainer->iListBox->StoreSettingsL();
		AppUi()->HandleCommandL(EBlueTCmdExitDisplaySettings);
		break;
		default:
		AppUi()->HandleCommandL(aCommand);
		break;
	}
}

void CBlueTDisplaySettingsView::HandleClientRectChange()
{	if (iContainer)
		iContainer->SetRect(ClientRect());
}

void CBlueTDisplaySettingsView::DoActivateL(const TVwsViewId& /*aPrevViewId*/, TUid /*aCustomMessageId*/,
                                            const TDesC8& /*aCustomMessage*/)
{	Cba()->SetCommandSetL(R_AVKON_SOFTKEYS_OPTIONS_BACK);
	if (!iContainer)
	{	CBlueTAppUi* appUi = (CBlueTAppUi*) AppUi();
		iOrigDisplayType = (TBlueTSettings::TDisplayType) appUi->Settings().iDisplayType;
		iOrigSkinFileIndex = appUi->Settings().iSkinFileIndex;
```

```
                iContainer = new (ELeave) CBlueTDisplaySettingsContainer;
                iContainer->SetMopParent(this);
                iContainer->ConstructL(appUi, this, ClientRect());
                AppUi()->AddToStackL(*this, iContainer);
        }
}

void CBlueTDisplaySettingsView::DoDeactivate()
{       Cba()->SetCommandSetL(R_AVKON_SOFTKEYS_OPTIONS_EXIT);
        Cba()->DrawDeferred();
        if (iContainer)
        {       CBlueTAppUi* appUi = (CBlueTAppUi*) AppUi();
                if ((appUi->Settings().iDisplayType != iOrigDisplayType) ||
                        (appUi->Settings().iSkinFileIndex != iOrigSkinFileIndex))
                {       appUi->ReloadSkin();
                }
        AppUi()->RemoveFromViewStack( *this, iContainer );
        }
        delete iContainer;
        iContainer = NULL;
}

CBlueTDisplaySettingsContainer* CBlueTDisplaySettingsView::SettingsContainer()
{
        return iContainer;
}
```

BlueTListBox.cpp

```
#include <gdi.h>
#include <eiktxlbm.h>
#include <AknUtils.h>
#include <BlueT.rsg>
#include "BlueTListBox.h"
#include "BlueTAppUi.h"

CBlueTListBoxView::CBlueTListBoxView()
{
}

void CBlueTListBoxView::DrawEmptyList(const TRect &aClientRect) const
{
        AknDraw::DrawEmptyList(aClientRect, *iGc, iListEmptyText->Des());
}

void CBlueTListBoxView::Draw(const TRect* aClipRect) const
{       if (iModel->NumberOfItems() > 0)
                CListBoxView::Draw(aClipRect);
        else
                DrawEmptyList(iViewRect);
}

CBlueTListItemDrawer::CBlueTListItemDrawer(MTextListBoxModel* aTextListBoxModel, const CFont* aFont)
        : CTextListItemDrawer(aTextListBoxModel, aFont), iPersistentFont(aFont)
{
}

void CBlueTListItemDrawer::DrawActualItem(TInt aItemIndex, const TRect& aActualItemRect, TBool
        aItemIsCurrent, TBool /*aViewIsEmphasized*/, TBool, TBool /*aItemIsSelected*/) const
{       TRgb textColour;
        TRgb bgColour;
        if (aItemIsCurrent)
        {       textColour = iHighlightForegroundColour;
                bgColour = iHighlightBackgroundColour;
        }
```

```
		else
		{	textColour = iTextForegroundColour;
			bgColour = iTextBackgroundColour;
		}
		Gc()->UseFont(iPersistentFont);
		Gc()->SetBrushStyle(CGraphicsContext::ESolidBrush);
		Gc()->SetBrushColor(bgColour);
		Gc()->SetPenColor(bgColour);
		Gc()->SetPenColor(textColour);
		TBuf<256> displayBuf;
		if (aItemIsCurrent)
			displayBuf = iModel->ItemText(aItemIndex).Mid(iScrollOffset);
		else
			displayBuf = iModel->ItemText(aItemIndex);
			displayBuf.Insert(0, _L("  "));
			Gc()->DrawText(displayBuf, aActualItemRect,(aActualItemRect.Height() / 2) +
					(iFont->AscentInPixels() / 2));
			Gc()->SetPenColor(bgColour);
		}

void CBlueTListItemDrawer::SetColours(const TRgb& aTextForeground, const TRgb& aTextBackground, const
		TRgb& aHighlightForeground, const TRgb& aHighlightBackground)
{	iTextForegroundColour = aTextForeground;
		iTextBackgroundColour = aTextBackground;
		iHighlightForegroundColour = aHighlightForeground;
		iHighlightBackgroundColour = aHighlightBackground;
		iBackColor = iTextBackgroundColour;
}

TBool CBlueTListItemDrawer::ScrollSong(TInt aItemIndex)
{	TBuf<256> displayBuf;
		displayBuf = iModel->ItemText(aItemIndex);
		displayBuf.Insert(0, _L("  "));
		TInt trackNameChars = iPersistentFont->TextCount(displayBuf, KFullScreenWidth);
		if (trackNameChars < displayBuf.Length())
		{	iScrollOffset += 2;
			if (iScrollOffset > (displayBuf.Length() - trackNameChars + 1))
				iScrollOffset = 0;
			return ETrue;
		}
		return EFalse;
}

void CBlueTListItemDrawer::ResetScrolling()
{
		iScrollOffset = 0;
}

CBlueTListBox::CBlueTListBox(const CFont* aFont)
{
		iFont = aFont;
}

void CBlueTListBox::CreateItemDrawerL()
{
		iItemDrawer = new (ELeave) CBlueTListItemDrawer(Model(), iFont);
}

CListBoxView* CBlueTListBox::MakeViewClassInstanceL()
{
		return new (ELeave) CBlueTListBoxView();
}
```

BluetoothHandler.cpp

```
#include <eikenv.h>
#include <eikappui.h>
#include "BlueTApp.h"
#include "BluetoothHandler.h"
#include "BlueTAppUi.h"
#include <BlueT.rsg>
#include <in_sock.h>

_LIT(KSongList, "songlist.dat");

CBluetoothHandler::CBluetoothHandler(MBluetoothListener* aListener, const TBTDevAddr&
aAutoConnectAddress, TInt aAutoConnectPort): CActive(EPriorityStandard), iListener(aListener),
	iAutoConnectAddress(aAutoConnectAddress), iAutoConnectPort(aAutoConnectPort)
{
	CActiveScheduler::Add(this);
	iServiceClass = KSerialPortService;
}

CBluetoothHandler* CBluetoothHandler::NewL(MBluetoothListener* aListener,const TBTDevAddr&
	aAutoConnectAddress,TInt aAutoConnectPort)
{	CBluetoothHandler* self = new (ELeave) CBluetoothHandler(aListener,aAutoConnectAddress,
		aAutoConnectPort);
	CleanupStack::PushL(self);
	self->ConstructL();
	CleanupStack::Pop();
	return self;
}

void CBluetoothHandler::ConstructL()
{	iCommandTimer = CPeriodic::NewL(EPriorityStandard);
	User::LeaveIfError(iSocketServ.Connect());
	User::LeaveIfError(iResolver.Open(iSocketServ, KAfInet, KProtocolInetUdp));
	User::LeaveIfError(iNotifier.Connect());
	TFileName app = CEikonEnv::Static()->EikAppUi()->Application()->AppFullName();
	TParsePtr parse(app);
	TFileName songListFileName(parse.DriveAndPath());
	songListFileName.Append(KSongList);
	ReadSongListFile(songListFileName);
}

CBluetoothHandler::~CBluetoothHandler()
{	Cancel();
	CloseAllResources();
	delete iCommandTimer;
	delete iWaitDialog;
	delete iProgressDialog;
	delete iDirTreeRoot;
	delete iSdpSearchPattern;
	delete iSdpAgent;
	iResolver.Close();
	iSocketServ.Close();
	iNotifier.Close();
}

void CBluetoothHandler::CloseAllResources()
{	iState = EIdle;
	iConnected = EFalse;
	iCurrentCommand = EBlueTNone;
	iQueuedCommand = EBlueTNone;
	iWaitingForResponseData = EFalse;
	if (iCommandTimer)
		iCommandTimer->Cancel();
	if (iWaitDialog)
		iWaitDialog->ProcessFinishedL();
	if (iProgressDialog)
```

```
                iProgressDialog->ProcessFinishedL();
        iSongListFile.Close();
        iDownloadFile.Close();
        iSocket.Close();
}

void CBluetoothHandler::ReadSongListFile(const TDesC& aFileName)
{       TInt ret;
        ret = iSongListFile.Open(CEikonEnv::Static()->FsSession(), aFileName,EFileRead | EFileShareAny);
        if (ret != KErrNone)
        {       RequestList(ETrue);
                return;
        }
        delete iDirTreeRoot;
        iDirTreeRoot = NULL;
        iDirTreeCurrentNode = NULL;
        iTotalPacketBuf.Zero();
        iState = EReadingSongListFile;
        iSongListFile.Read(iPacketBuf, iStatus);
        SetActive();
}

void CBluetoothHandler::Connect()
{       iCurrentCommand = EBlueTConnect;
        iState = ESelectingServer;
        if (!iSocket.SubSessionHandle())
        {       TInt ret;
                ret = iSocket.Open(iSocketServ, KBTAddrFamily, KSockStream, KRFCOMM);
                if (ret != KErrNone)
                {       ConnectionFailed(ret);
                        return;
                }
        }
        TBTDevAddr zeroAddr;
        if ((iQueuedCommand == EBlueTFindServer) || (iAutoConnectAddress == zeroAddr))
        {       iNotifier.StartNotifierAndGetResponse(iStatus, KDeviceSelectionNotifierUid,
                        iDeviceSelectionPckg, iDeviceFoundPckg);
        }
        else
        {       iState = EConnecting;
                TBTSockAddr addr;
                addr.SetBTAddr(iAutoConnectAddress);
                addr.SetPort(iAutoConnectPort);
                iDeviceFoundPckg().SetDeviceAddress(iAutoConnectAddress);
                iRemotePort = iAutoConnectPort;
                iSocket.Connect(addr, iStatus);
        }
        SetActive();
}

void CBluetoothHandler::WriteCommand()
{       iWaitingForResponseData = EFalse;
        if (iConnected)
        {       if (iVersionState == ENotGotVersion)
                {
                iQueuedCommand = iCurrentCommand;
                iCurrentCommand = EBlueTGetVersion;
                iVersionState = EGettingVersion;
                iVersionQueuedPacketBuf.Copy(iPacketBuf);
                iPacketBuf.Copy(_L8("VERS"));
                if (iCommandTimer->IsActive())
                iCommandTimer->Cancel();
                iCommandTimer->Start(2 * 1000000, 1 * 1000000,TCallBack(CommandTimerCallBack, this));
                iState = EWriting;
                iSocket.Write(iPacketBuf, iStatus);
                SetActive();
                }
```

```
                    else
                    {
                    if (!iCommandTimer->IsActive() && (iCurrentCommand != EBlueTDownloadFile))
                    {       if ((iCurrentCommand != EBlueTList) &&(iCurrentCommand != EBlueTGetPlaylist))
                            {
                            iCommandTimer->Start(KNormalCommandTimeout, 1 * 1000000,
                                    TCallBack(CommandTimerCallBack, this));
                            }
                            else
                            {
                            iCommandTimer->Start(KLongCommandTimeout, 1 * 1000000,
                                    TCallBack(CommandTimerCallBack, this));
                            }
                    }
                    iState = EWriting;
                    iSocket.Write(iPacketBuf, iStatus);
                    SetActive();
                    }
            }
            else
            {       iQueuedCommand = iCurrentCommand;
                    Connect();
            }
}

void CBluetoothHandler::RequestList(TBool aListEntireTree)
{       if (iCurrentCommand != EBlueTNone)
                    return;
            if (iConnected)
            {       TRAPD(err, ShowWaitDialogL(R_REFRESHING_LIST_MSG));
            }
            delete iDirTreeRoot;
            iDirTreeRoot = NULL;
            iDirTreeCurrentNode = NULL;
            iDirTreeNodeToExpand = NULL;
            iCurrentCommand = EBlueTList;
            if (!aListEntireTree)
            {       iPacketBuf.Copy(_L8("DLST  "));
                    iPacketBuf[4] = 0;
                    iPacketBuf[5] = 0;
            }
            else
            {       iPacketBuf.Copy(_L8("LIST"));
            }
            WriteCommand();
}

void CBluetoothHandler::RequestGetPlaylist()
{       if (iCurrentCommand != EBlueTNone)
                    return;
            if (iConnected)
            {       TRAPD(err, ShowWaitDialogL(R_GETTING_PLAYLIST_MSG));
            }
            iCurrentCommand = EBlueTGetPlaylist;
            iPacketBuf.Copy(_L8("PLST"));
            WriteCommand();
}

void CBluetoothHandler::RequestDirectoryList(CDirTreeNode* aNode)
{       if (iCurrentCommand != EBlueTNone)
                    return;
            if (iConnected)
            {       TRAPD(err, ShowWaitDialogL(R_LISTING_FOLDER_MSG));
            }
            iDirTreeCurrentNode = iDirTreeNodeToExpand = aNode;
            iCurrentCommand = EBlueTList;
            TBuf8<256> fileName = *aNode->iFileName;
```

```
		Do
		{	aNode = aNode->iParent;
			if (aNode && (*aNode->iFileName != KNullDesC8))
			{	fileName.Insert(0, _L8("\\"));
				fileName.Insert(0, *aNode->iFileName);
			}
		}
		while (aNode && aNode->iParent);
		iPacketBuf.Format(_L8("DLST  %S"), &fileName);
		iPacketBuf[4] = (TUint8)((fileName.Length() >> 8) & 0xFF);
		iPacketBuf[5] = (TUint8)(fileName.Length() & 0xFF);
		WriteCommand();
}

void CBluetoothHandler::RequestInfo()
{	if (iCurrentCommand != EBlueTNone)
			return;
		iCurrentCommand = EBlueTCurrentTrackInfo;
		iPacketBuf.Copy(_L8("INFO"));
		WriteCommand();
}

void CBluetoothHandler::RequestPlayFile(const TDesC8& aFileName)
{	if (iCurrentCommand != EBlueTNone)
			return;
		iCurrentCommand = EBlueTPlayFile;
		iPacketBuf.Copy(_L8("PLAY"));
		iPacketBuf.Append(_L8("  "));
		iPacketBuf[4] = (TUint8)((aFileName.Length() >> 8) & 0xFF);
		iPacketBuf[5] = (TUint8)(aFileName.Length() & 0xFF);
		iPacketBuf.Append(aFileName);
		WriteCommand();
}

void CBluetoothHandler::RequestAddFileToPlaylist(const TDesC8& aFileName)
{	if (iCurrentCommand != EBlueTNone)
			return;
		iCurrentCommand = EBlueTAddFileToPlaylist;
		iPacketBuf.Copy(_L8("LADD"));
		iPacketBuf.Append(_L8("  "));
		iPacketBuf[4] = (TUint8) ((aFileName.Length() >> 8) & 0xFF);
		iPacketBuf[5] = (TUint8) (aFileName.Length() & 0xFF);
		iPacketBuf.Append(aFileName);
		WriteCommand();
}

void CBluetoothHandler::RequestFileInfo(const TDesC8& aFileName)
{	if (iCurrentCommand != EBlueTNone)
			return;
		iCurrentCommand = EBlueTFileInfo;
		iDownloadFileName.Copy(aFileName);
		iPacketBuf.Copy(_L8("FINF"));
		iPacketBuf.Append(_L8("  "));
		iPacketBuf[4] = (TUint8) ((aFileName.Length() >> 8) & 0xFF);
		iPacketBuf[5] = (TUint8) (aFileName.Length() & 0xFF);
		iPacketBuf.Append(aFileName);
		WriteCommand();
}

void CBluetoothHandler::RequestDownloadFile(const TDesC8& aFileName)
{	if (iCurrentCommand != EBlueTNone)
			return;
		iCurrentCommand = EBlueTDownloadFile;
		iDownloadFileName.Copy(aFileName);
		iPacketBuf.Copy(_L8("DOWN"));
		iPacketBuf.Append(_L8("  "));
```

```
		iPacketBuf[4] = (TUint8) ((aFileName.Length() >> 8) & 0xFF);
		iPacketBuf[5] = (TUint8) (aFileName.Length() & 0xFF);
		iPacketBuf.Append(aFileName);
		WriteCommand();
}

void CBluetoothHandler::RequestSetVolume(TUint8 aVolume)
{		if (iCurrentCommand != EBlueTNone)
			return;
		iCurrentCommand = EBlueTSetVolume;
		iPacketBuf.Copy(_L8("VOLM"));
		iPacketBuf.Append(_L8(" "));
		iPacketBuf[4] = aVolume;
		WriteCommand();
}

void CBluetoothHandler::RequestStop()
{		if (iCurrentCommand != EBlueTNone)
			return;
		iCurrentCommand = EBlueTStop;
		iPacketBuf.Copy(_L8("STOP"));
		WriteCommand();
}

void CBluetoothHandler::RequestPause()
{		if (iCurrentCommand != EBlueTNone)
			return;
		iCurrentCommand = EBlueTPause;
		iPacketBuf.Copy(_L8("PAUS"));
		WriteCommand();
}

void CBluetoothHandler::RequestPlay()
{		if (iCurrentCommand != EBlueTNone)
			return;
		iCurrentCommand = EBlueTPlay;
		iPacketBuf.Copy(_L8("STRT"));
		WriteCommand();
}

void CBluetoothHandler::RequestNextTrack()
{		if (iCurrentCommand != EBlueTNone)
			return;
		iCurrentCommand = EBlueTNextTrack;
		iPacketBuf.Copy(_L8("NEXT"));
		WriteCommand();
}

void CBluetoothHandler::RequestPreviousTrack()
{		if (iCurrentCommand != EBlueTNone)
			return;
		iCurrentCommand = EBlueTPreviousTrack;
		iPacketBuf.Copy(_L8("PREV"));
		WriteCommand();
}

void CBluetoothHandler::RequestFastForward()
{		if (iCurrentCommand != EBlueTNone)
			return;
		iCurrentCommand = EBlueTFastForward;
		iPacketBuf.Copy(_L8("FFWD"));
		WriteCommand();
}

void CBluetoothHandler::RequestRewind()
{		if (iCurrentCommand != EBlueTNone)
			return;
```

```
	iCurrentCommand = EBlueTRewind;
	iPacketBuf.Copy(_L8("RWND"));
	WriteCommand();
}

void CBluetoothHandler::RequestShuffle(TBool aShuffleOn)
{	if (iCurrentCommand != EBlueTNone)
		return;
	iCurrentCommand = EBlueTShuffle;
	iPacketBuf.Copy(_L8("SHFL "));
	iPacketBuf[4] = (TUint8) aShuffleOn;
	WriteCommand();
}

void CBluetoothHandler::RequestRepeat(TBool aRepeatOn)
{	if (iCurrentCommand != EBlueTNone)
		return;
	iCurrentCommand = EBlueTRepeat;
	iPacketBuf.Copy(_L8("REPT "));
	iPacketBuf[4] = (TUint8) aRepeatOn;
	WriteCommand();
}

void CBluetoothHandler::RequestShutDown()
{	if (iCurrentCommand != EBlueTNone)
		return;
	iCurrentCommand = EBlueTShutDown;
	iPacketBuf.Copy(_L8("SHUT"));
	WriteCommand();
}

void CBluetoothHandler::RequestFullScreen()
{	if (iCurrentCommand != EBlueTNone)
		return;
	iCurrentCommand = EBlueTFullScreen;
	iPacketBuf.Copy(_L8("FULL"));
	WriteCommand();
}

void CBluetoothHandler::RequestFindServer()
{	if (iCurrentCommand != EBlueTNone)
		return;
	if (iConnected)
		CloseAllResources();
	iCurrentCommand = EBlueTFindServer;
	iFindServerPort = 1;
	iPacketBuf.Copy(_L8("CHCK"));
	WriteCommand();
}

void CBluetoothHandler::RequestSelectInPlaylist(TInt aIndex)
{	if (iCurrentCommand != EBlueTNone)
		return;
	iCurrentCommand = EBlueTSelectInPlaylist;
	iPacketBuf.Copy(_L8("SLCT  "));
	iPacketBuf[4] = (TUint8) ((aIndex >> 8) & 0xFF);
	iPacketBuf[5] = (TUint8) (aIndex & 0xFF);
	WriteCommand();
}

void CBluetoothHandler::RequestRemoveAllFromPlaylist()
{	if (iCurrentCommand != EBlueTNone)
		return;
	iCurrentCommand = EBlueTRemoveAllFromPlaylist;
	iPacketBuf.Copy(_L8("RMAL"));
	WriteCommand();
}
```

```
void CBluetoothHandler::RequestDetailedTrackInfo()
{	if (iCurrentCommand != EBlueTNone)
		return;
	iCurrentCommand = EBlueTDetailedTrackInfo;
	iPacketBuf.Copy(_L8("DINF"));
	WriteCommand();
}

void CBluetoothHandler::RunL()
{	switch (iState)
	{
	case EReadingSongListFile:
		HandleReadFileCompleteL();
		break;
	case ESelectingServer:
		HandleSelectServerCompleteL();
		break;
	case EConnecting:
		HandleConnectionCompleteL();
		break;
	case EReading:
		HandleReadCompleteL();
		break;
	case EWriting:
		HandleWriteCompleteL();
		break;
	default:
	{	TBuf<256> formatBuf;
		formatBuf.Format(_L("Unexpected Bluetooth state %d"), iState);
		CEikonEnv::Static()->InfoWinL(KNullDesC, formatBuf);
	}
	break;
	}
}

void CBluetoothHandler::HandleReadFileCompleteL()
{	TInt ret = KErrNone;
	if ((iStatus.Int() >= KErrNone) && (iPacketBuf.Length() > 0))
	{	iTotalPacketBuf.Append(iPacketBuf);
		ret = ParseListPacketL(iTotalPacketBuf);
		if (ret >= KErrNone)
		{	iSongListFile.Read(iPacketBuf, iStatus);
			SetActive();
			return;
		}
	}
	if ((iStatus.Int() < KErrNone) || (ret < KErrNone))
	{	TBuf<256> resourceBuf;
		iListener->CoeEnv()->ReadResource(resourceBuf, R_READ_SONG_LIST_ERROR);
		TBuf<256> formatBuf;
		formatBuf.Format(resourceBuf, ret);
		CEikonEnv::Static()->InfoWinL(formatBuf, KNullDesC);
	}
	iState = EIdle;
	iSongListFile.Close();
	SortDirTree(iDirTreeRoot);
	iListener->RefreshFileListL(iDirTreeRoot);
}

void CBluetoothHandler::HandleSelectServerCompleteL()
{	if (iStatus.Int() == KErrNone)
	{	iNameEntry().iAddr.SetPort(7654);
		iState = EConnecting;
		iSocket.Connect(iNameEntry().iAddr, iStatus);
		SetActive();
	}
```

```
	else
	{	ConnectionFailed(iStatus.Int());
	}
	return;
	if ((iStatus.Int() != KErrNone) || !iDeviceFoundPckg().IsValidDeviceName()))
	{	ConnectionFailed(iStatus.Int());
		return;
	}
	if (iQueuedCommand == EBlueTFindServer)
	{	TInt err = KErrNone;
		if (!iProgressDialog)
		{	TRAP(err, iProgressDialog = new (ELeave) CAknProgressDialog
				(REINTERPRET_CAST(CEikDialog**, &iProgressDialog), ETrue));
		}
	if (err == KErrNone)
	{	iProgressDialog->PrepareLC(R_PROGRESS_NOTE);
		iProgressInfo = iProgressDialog->GetProgressInfoL();
		iProgressDialog->SetCallback(this);
		TBuf<256> resourceBuf;
		iListener->CoeEnv()->ReadResource(resourceBuf, R_FINDING_BLUET_SERVER);
		iProgressDialog->SetTextL(resourceBuf);
		iProgressDialog->SetTone(CAknNoteDialog::EConfirmationTone);
		iProgressDialog->RunLD();
		iProgressInfo->SetFinalValue(29);
		TBTSockAddr addr;addr.SetBTAddr(iDeviceFoundPckg().BDAddr());
		addr.SetPort(iFindServerPort);
		iState = EConnecting;
		iSocket.Connect(addr, iStatus);
		SetActive();
	}
	return;
	}
	if (!iWaitDialog)
	{	iWaitDialog = new (ELeave) CAknWaitDialog(REINTERPRET_CAST
			(CEikDialog**,&iWaitDialog));
		iWaitDialog->SetTone(CAknNoteDialog::EConfirmationTone);
		iWaitDialog->ExecuteLD(R_WAIT_NOTE);
	}
	if (iSdpAgent)
	{	delete iSdpAgent;
		iSdpAgent = NULL;
	}
	iSdpAgent = CSdpAgent::NewL(*this, iDeviceFoundPckg().BDAddr());
	if (iSdpSearchPattern)
	{	delete iSdpSearchPattern;
		iSdpSearchPattern = NULL;
	}
	iSdpCurrentRecordCount = 0;
	iSdpTotalRecordCount = 0;
	iSdpSearchPattern = CSdpSearchPattern::NewL();
	iSdpSearchPattern->AddL(iServiceClass);
	iSdpAgent->SetRecordFilterL(*iSdpSearchPattern);
	iSdpAgent->NextRecordRequestL();
	iState = EInquiringService;
}

void CBluetoothHandler::HandleConnectionCompleteL()
{	iCurrentCommand = EBlueTNone;
	if (iStatus.Int() != KErrNone)
	{	if (iQueuedCommand == EBlueTFindServer)
		{	FindServerNextChannel();
		}
		else
		{	ConnectionFailed(iStatus.Int());
		}
		return;
	}
```

```
		iConnected = ETrue;
		iListener->ConnectComplete(KErrNone, iDeviceFoundPckg().BDAddr(), iRemotePort);
		if (iQueuedCommand == EBlueTList)
		{	ShowWaitDialogL(R_REFRESHING_LIST_MSG);
			iCurrentCommand = iQueuedCommand;
			iQueuedCommand = EBlueTNone;
			WriteCommand();
		}
		else if (iQueuedCommand == EBlueTDownloadFile)
		{	ShowWaitDialogL(R_DOWNLOADING_FILE_MSG);
			iCurrentCommand = iQueuedCommand;
			iQueuedCommand = EBlueTNone;
			WriteCommand();
		}
		else if (iQueuedCommand == EBlueTGetPlaylist)
		{	ShowWaitDialogL(R_GETTING_PLAYLIST_MSG);
			iCurrentCommand = iQueuedCommand;
			iQueuedCommand = EBlueTNone;
			WriteCommand();
		}
		else if (iQueuedCommand == EBlueTFindServer)
		{	iCommandTimer->Start(5 * 1000000, 1 * 1000000, TCallBack(CommandTimerCallBack, this));
			iCurrentCommand = EBlueTFindServer;
			iQueuedCommand = EBlueTNone;
			iPacketBuf.Copy(_L8("CHCK"));
			WriteCommand();
		}
		else if (iQueuedCommand != EBlueTNone)
		{	if (iWaitDialog)
				iWaitDialog->ProcessFinishedL();
			iCurrentCommand = iQueuedCommand;
			iQueuedCommand = EBlueTNone;
			WriteCommand();
		}
		else
		{	if (iWaitDialog)
				iWaitDialog->ProcessFinishedL();
		}
}

void CBluetoothHandler::HandleReadCompleteL()
{	if (iStatus.Int() == KErrNone)
		{	ParseCommandResponse();
		}
		else
		{	if (iCurrentCommand == EBlueTFindServer)
			{	FindServerNextChannel();
				return;
			}
			iListener->CommandComplete(iCurrentCommand, iStatus.Int());
			CloseAllResources();
		}
}

void CBluetoothHandler::HandleWriteCompleteL()
{	if ((iStatus.Int() == KErrNone) &&	CommandExpectsResponse(iCurrentCommand))
		{	iState = EReading;
			iSocket.RecvOneOrMore(iPacketBuf, 0, iStatus, iXfrLength);
			SetActive();
		}
		else
		{	iCommandTimer->Cancel();
			if (iCurrentCommand == EBlueTFindServer)
			{	FindServerNextChannel();
				return;
			}
			iListener->CommandComplete(iCurrentCommand, iStatus.Int());
```

```
                        if (iStatus.Int() == KErrNone)
                        {       if (iCurrentCommand == EBlueTRemoveAllFromPlaylist)
                                {       iCurrentCommand = EBlueTNone;
                                        RequestGetPlaylist();
                                }
                                else if (iCurrentCommand != EBlueTSetVolume)
                                {       iCurrentCommand = EBlueTNone;
                                        RequestInfo();
                                }
                                else
                                {       iCurrentCommand = EBlueTNone;
                                }
                        }
                        else
                        {       CloseAllResources();
                        }
                }
}

void CBluetoothHandler::ParseCommandResponse()
{       TInt err;
        switch (iCurrentCommand)
        {
        case EBlueTList:
                ParseListResponse();
                break;
        case EBlueTCurrentTrackInfo:
                ParseCurrentTrackInfoResponse();
                break;
        case EBlueTFileInfo:
                ParseFileInfoResponse();
                break;
        case EBlueTGetVersion:
                ParseGetVersionResponse();
                break;
        case EBlueTGetVolume:
                ParseGetVolumeResponse();
                break;
        case EBlueTDownloadFile:
                ParseDownloadFileResponse();
                break;
        case EBlueTFindServer:
                ParseFindServerResponse();
                break;
        case EBlueTGetPlaylist:
                TRAP(err, ParseGetPlaylistResponseL());
                break;
        case EBlueTDetailedTrackInfo:
                ParseDetailedTrackInfoResponse();
                break;
        default:
                CEikonEnv::Static()->InfoWinL(_L("Unexpected packet"), KNullDesC);
        break;
        }
}

void CBluetoothHandler::ParseFindServerResponse()
{       iCommandTimer->Cancel();
        if (iPacketBuf.Find(_L8("Y")) > KErrNotFound)
        {       TRAPD(err, iProgressDialog->ProcessFinishedL());
                iProgressDialog->SetCallback(NULL);
                delete iProgressDialog;
                iProgressDialog = NULL;
                TBuf<256> resourceBuf;
                iListener->CoeEnv()->ReadResource(resourceBuf, R_FOUND_BLUET_SERVER);
                TBuf<256> formatBuf;
```

```
                formatBuf.Format(resourceBuf, iFindServerPort);
                CEikonEnv::Static()->InfoWinL(formatBuf, KNullDesC);
                iCommandTimer->Cancel();
                iListener->CommandComplete(iCurrentCommand, iStatus.Int());
                iCurrentCommand = EBlueTNone;
        }
        else
        {       FindServerNextChannel();
        }
}

void CBluetoothHandler::ParseListResponse()
{       if (!iWaitingForResponseData)
        {       TInt listAckIndex = iPacketBuf.Find(_L8("LISTACK"));
                if (listAckIndex > KErrNotFound)
                {
                iPacketBuf.Delete(0, listAckIndex + 7);
                iTotalPacketBuf.Zero();
                iWaitingForResponseData = ETrue;
                if (!iDirTreeNodeToExpand)
                {
                TFileName app = CEikonEnv::Static()->EikAppUi()->Application()->AppFullName();
                TParsePtr parse(app);
                TFileName songListFileName(parse.DriveAndPath());
                songListFileName.Append(KSongList);
                iSongListFile.Replace(CEikonEnv::Static()->FsSession(), songListFileName,EFileShareAny);
                }       }
                else
                {       iPacketBuf.Zero();
                        iSocket.RecvOneOrMore(iPacketBuf, 0, iStatus, iXfrLength);
                        SetActive();
                }
        }

        if (iWaitingForResponseData)
        {       TInt ret;
                if (!iDirTreeNodeToExpand)
                {       ret = iSongListFile.Write(iPacketBuf);
                        if (ret != KErrNone)
                        {
                        CEikonEnv::Static()->InfoWinL(_L("Write to file failed!"), KNullDesC);
                        iListener->CommandComplete(iCurrentCommand, ret);
                        CloseAllResources();
                        return;
                        }
                }
                iTotalPacketBuf.Append(iPacketBuf);
                ret = ParseListPacketL(iTotalPacketBuf);
                if (ret > 0)
                {       iPacketBuf.Zero();
                        iSocket.RecvOneOrMore(iPacketBuf, 0, iStatus, iXfrLength);
                        SetActive();
                }
                else
                {       iSongListFile.Close();
                        SortDirTree(iDirTreeRoot);
                        if (iWaitDialog)
                                iWaitDialog->ProcessFinishedL();
                                iCommandTimer->Cancel();
                                iListener->CommandComplete(iCurrentCommand, ret);
                                iCurrentCommand = EBlueTNone;
                        if (iDirTreeNodeToExpand)
                        {       iDirTreeNodeToExpand->iFileType = KFileTypeDir;
                                iListener->RefreshFileListL(iDirTreeNodeToExpand);
                        }
                        else
                        {       iListener->RefreshFileListL(iDirTreeRoot);      }}}}
```

```
void CBluetoothHandler::ParseCurrentTrackInfoResponse()
{	if (!iWaitingForResponseData)
	{	TInt infoAckIndex = iPacketBuf.Find(_L8("INFOACK"));
		if (infoAckIndex > KErrNotFound)
		{	iPacketBuf.Delete(0, infoAckIndex + 7);
			iTotalPacketBuf.Zero();
			iWaitingForResponseData = ETrue;
		}
		else
		{	iPacketBuf.Zero();
			iSocket.RecvOneOrMore(iPacketBuf, 0, iStatus, iXfrLength);
			SetActive();
		}
	}
	if (iWaitingForResponseData)
	{	iTotalPacketBuf.Append(iPacketBuf);
		if (iTotalPacketBuf.Length() < 11)
		{	iPacketBuf.Zero();
			iSocket.RecvOneOrMore(iPacketBuf, 0, iStatus, iXfrLength);
			SetActive();
		}
		else
		{	TBool playing = EFalse;
			TBool paused = EFalse;
			TInt playingByte = iTotalPacketBuf[0];
			if (playingByte & 0x1)
				playing = ETrue;
			if (playingByte & 0x2)
				paused = ETrue;
			TInt length = (iTotalPacketBuf[1] << 24) + (iTotalPacketBuf[2] << 16) +
				(iTotalPacketBuf[3] << 8) + iTotalPacketBuf[4];
			TInt currentTime = (iTotalPacketBuf[5] << 24) + (iTotalPacketBuf[6] << 16) +
				(iTotalPacketBuf[7] << 8) + iTotalPacketBuf[8];
			TBool shuffle = (TBool) iTotalPacketBuf[9];
			TBool repeat = (TBool) iTotalPacketBuf[10];
			TPtrC8 guessPtr;
			if (iTotalPacketBuf.Length() > 11)
			guessPtr.Set(&iTotalPacketBuf[11], iTotalPacketBuf.Length() - 11);
			else
				guessPtr.Set(KNullDesC8);
			iListener->SetInfo(playing, paused, length, currentTime, shuffle, repeat, guessPtr);
			iListener->CommandComplete(iCurrentCommand, KErrNone);
			iCommandTimer->Cancel();
			iCurrentCommand = EBlueTNone;
		}
	}
}

void CBluetoothHandler::ParseFileInfoResponse()
{	TBuf<256> formatBuf;
	TBuf8<256> fileNameBuf8;
	TBuf<256> resourceBuf;
	TInt finfAckIndex = iPacketBuf.Find(_L8("FINFACK"));
	if (finfAckIndex > KErrNotFound)
	{	iCommandTimer->Cancel();
		iPacketBuf.Delete(0, finfAckIndex + 7);
		iDownloadFileLength = (iPacketBuf[0] << 24) + (iPacketBuf[1] << 16) +(iPacketBuf[2] << 8)
			+ iPacketBuf[3];
		iPacketBuf.Delete(0, 4);
		TVolumeInfo volumeInfo;
		CEikonEnv::Static()->FsSession().Volume(volumeInfo,iListener->Settings().iDownloadDrive
			+ EDriveC);
		if (volumeInfo.iFree < (iDownloadFileLength + (128 * 1024)))
		{	iListener->CoeEnv()->ReadResource(resourceBuf, R_NOT_ENOUGH_SPACE);
			formatBuf.Format(resourceBuf, (iDownloadFileLength / 1024) + 128);
			CEikonEnv::Static()->InfoWinL(KNullDesC, formatBuf);
			iListener->CommandComplete(iCurrentCommand, KErrTooBig);
```

```
                        iCurrentCommand = EBlueTNone;
                        return;
                }
        formatBuf.Copy(KDownloadedFilesPath);
                formatBuf[0] = 'C' + iListener->Settings().iDownloadDrive;
                TInt backslashPos = iDownloadFileName.LocateReverse('\\');
        if (backslashPos > KErrNotFound)
        {       formatBuf.Append(_L("\\"));
                formatBuf.Append(iDownloadFileName.Mid(backslashPos + 1));
        }
        else
        {       formatBuf.Append(_L("\\BlueT.wav"));
        }
        TInt err = iDownloadFile.Replace(CEikonEnv::Static()->FsSession(),formatBuf, EFileShareAny);
        if (err != KErrNone)
        {       iListener->CommandComplete(iCurrentCommand, err);
                CloseAllResources();
                return;
        }
        iListener->CoeEnv()->ReadResource(resourceBuf, R_START_DOWNLOAD_MSG);
        formatBuf.Format(resourceBuf, &iDownloadFileName, iDownloadFileLength);
        CEikonEnv::Static()->InfoWinL(KNullDesC, formatBuf);
        if (!iProgressDialog)
        {       TRAP(err, iProgressDialog = new (ELeave) CAknProgressDialog(REINTERPRET_CAST
                        (CEikDialog**, &iProgressDialog), ETrue));
        }
        if (err == KErrNone)
        {       iProgressDialog->PrepareLC(R_PROGRESS_NOTE);
                iProgressInfo = iProgressDialog->GetProgressInfoL();
                iProgressDialog->SetCallback(this);
                iProgressDialog->RunLD();
                iProgressInfo->SetFinalValue(iDownloadFileLength);
        }
        iCurrentCommand = EBlueTNone;
        fileNameBuf8.Copy(iDownloadFileName);
        RequestDownloadFile(fileNameBuf8);
        }
        else
        {       iPacketBuf.Zero();
                iSocket.RecvOneOrMore(iPacketBuf, 0, iStatus, iXfrLength);
                SetActive();
        }
}

void CBluetoothHandler::ParseDownloadFileResponse()
{       if (!iWaitingForResponseData)
        {       TInt downAckIndex = iPacketBuf.Find(_L8("DOWNACK"));
                if (downAckIndex > KErrNotFound)
                {       iPacketBuf.Delete(0, downAckIndex + 7);
                        iDownloadFileReceived = 0;
                        iWaitingForResponseData = ETrue;
                }
                else
                {       iPacketBuf.Zero();
                        iSocket.RecvOneOrMore(iPacketBuf, 0, iStatus, iXfrLength);
                        SetActive();
                }
        }
        if (iWaitingForResponseData)
        {       TInt ret;
                ret = iDownloadFile.Write(iPacketBuf);
                if (ret != KErrNone)
                {       iProgressDialog->ProcessFinishedL();
                        iListener->CommandComplete(iCurrentCommand, ret);
                        CloseAllResources();
                        return;
                }
```

```
                    iProgressInfo->IncrementAndDraw(iPacketBuf.Length());
                    iDownloadFileReceived += iPacketBuf.Length();
                    if (iDownloadFileReceived >= iDownloadFileLength)
                    {       iProgressDialog->ProcessFinishedL();
                            iProgressDialog->SetCallback(NULL);
                            delete iProgressDialog;
                            iProgressDialog = NULL;
                            CEikonEnv::Static()->InfoWinL(KNullDesC, _L("Download complete"));
                            iListener->CommandComplete(iCurrentCommand, KErrNone);
                            iDownloadFile.Close();
                            iCurrentCommand = EBlueTNone;
                    }
                    else
                    {       iPacketBuf.Zero();
                            iSocket.RecvOneOrMore(iPacketBuf, 0, iStatus, iXfrLength);
                            SetActive();
                    }
            }
}

void CBluetoothHandler::ParseGetPlaylistResponseL()
{       if (!iWaitingForResponseData)
        {       iPlaylistPos = -2;
                TInt playlistAckIndex = iPacketBuf.Find(_L8("PLSTACK"));
                if (playlistAckIndex > KErrNotFound)
                {       iPacketBuf.Delete(0, playlistAckIndex + 7);
                        iTotalPacketBuf.Zero();
                        iWaitingForResponseData = ETrue;
                }
                else
                {       iPacketBuf.Zero();
                        iSocket.RecvOneOrMore(iPacketBuf, 0, iStatus, iXfrLength);
                        SetActive();
                }
        }
        if (iWaitingForResponseData)
        {       iTotalPacketBuf.Append(iPacketBuf);
                TInt ret = ParsePlaylistPacketL(iTotalPacketBuf);
                if (ret > 0)
                {       iPacketBuf.Zero();
                        iSocket.RecvOneOrMore(iPacketBuf, 0, iStatus, iXfrLength);
                        SetActive();
                }
                else
                {       if (iWaitDialog)
                        iWaitDialog->ProcessFinishedL();
                        iCommandTimer->Cancel();
                        iListener->CommandComplete(iCurrentCommand, ret);
                        iCurrentCommand = EBlueTNone;
                        RequestInfo();
                }
        }
}

void CBluetoothHandler::ParseDetailedTrackInfoResponse()
{       if (!iWaitingForResponseData)
        {       TInt infoAckIndex = iPacketBuf.Find(_L8("DINFACK"));
                if (infoAckIndex > KErrNotFound)
                {       iPacketBuf.Delete(0, infoAckIndex + 7);
                        iTotalPacketBuf.Zero();
                        iWaitingForResponseData = ETrue;
                }
                else
                {       iPacketBuf.Zero();
                        iSocket.RecvOneOrMore(iPacketBuf, 0, iStatus, iXfrLength);
                        SetActive();
        }       }
```

```
	if (iWaitingForResponseData)
	{	iTotalPacketBuf.Append(iPacketBuf);
		if (iTotalPacketBuf.Length() < 12)
		{	iPacketBuf.Zero();
			iSocket.RecvOneOrMore(iPacketBuf, 0, iStatus, iXfrLength);
			SetActive();
		}
		else
		{	TInt bitRate;
			TInt sampleRate;
			TInt channels;
			bitRate = (iTotalPacketBuf[0] << 24) + (iTotalPacketBuf[1] << 16) +
				(iTotalPacketBuf[2] << 8) + iTotalPacketBuf[3];
			sampleRate = (iTotalPacketBuf[4] << 24) + (iTotalPacketBuf[5] << 16) +
				(iTotalPacketBuf[6] << 8) + iTotalPacketBuf[7];
			channels = (iTotalPacketBuf[8] << 24) + (iTotalPacketBuf[9] << 16) +
				(iTotalPacketBuf[10] << 8) + iTotalPacketBuf[11];
			iCommandTimer->Cancel();
			iListener->SetDetailedInfo(sampleRate, bitRate, channels);
			iListener->CommandComplete(iCurrentCommand, KErrNone);
			iCurrentCommand = EBlueTNone;
		}
	}
}

void CBluetoothHandler::ParseGetVersionResponse()
{	if (!iWaitingForResponseData)
	{	TInt versAckIndex = iPacketBuf.Find(_L8("VERSACK"));
		if (versAckIndex > KErrNotFound)
		{	iPacketBuf.Delete(0, versAckIndex + 7);
			iTotalPacketBuf.Zero();
			iWaitingForResponseData = ETrue;
		}
		else
		{	iPacketBuf.Zero();
			iSocket.RecvOneOrMore(iPacketBuf, 0, iStatus, iXfrLength);
			SetActive();
		}
	}
	if (iWaitingForResponseData)
	{	iTotalPacketBuf.Append(iPacketBuf);
		if (iTotalPacketBuf.Length() < 2)
		{	iPacketBuf.Zero();
			iSocket.RecvOneOrMore(iPacketBuf, 0, iStatus, iXfrLength);
			SetActive();
		}
		else
		{	iCommandTimer->Cancel();
			iServerMajorVersion = iTotalPacketBuf[0];
			iServerMinorVersion = iTotalPacketBuf[1];
			if ((iServerMajorVersion >= 1) && (iServerMinorVersion >= 00))
			{
			iCurrentCommand = EBlueTGetVolume;
			iVersionState = EGettingVolume;
			iPacketBuf.Copy(_L8("GVOL"));
			WriteCommand();
			}
			else
			{
			iVersionState = EGotVersion;
			iCurrentCommand = iQueuedCommand;
			iPacketBuf.Copy(iVersionQueuedPacketBuf);
			WriteCommand();
			}
		}
	}
}
```

```
void CBluetoothHandler::ParseGetVolumeResponse()
{	if (!iWaitingForResponseData)
	{	TInt gvolAckIndex = iPacketBuf.Find(_L8("GVOLACK"));
		if (gvolAckIndex > KErrNotFound)
		{	iPacketBuf.Delete(0, gvolAckIndex + 7);
			iTotalPacketBuf.Zero();
			iWaitingForResponseData = ETrue;
		}
		else
		{	gvolAckIndex = iPacketBuf.Find(_L8("GVOLNAK"));
			if (gvolAckIndex > KErrNotFound)
			{	iCommandTimer->Cancel();
				iVersionState = EGotVolume;
				iCurrentCommand = iQueuedCommand;
				iPacketBuf.Copy(iVersionQueuedPacketBuf);
				WriteCommand();
				return;
			}
			else
			{	iPacketBuf.Zero();
				iSocket.RecvOneOrMore(iPacketBuf, 0, iStatus, XfrLength);
				SetActive();
			}
		}
	}
	if (iWaitingForResponseData)
	{	iTotalPacketBuf.Append(iPacketBuf);
		if (iTotalPacketBuf.Length() < 1)
		{	iPacketBuf.Zero();
			iSocket.RecvOneOrMore(iPacketBuf, 0, iStatus, iXfrLength);
			SetActive();
		}
		else
		{	iCommandTimer->Cancel();
			iListener->ReceivedVolume(iTotalPacketBuf[0]);
			iVersionState = EGotVolume;
			iCurrentCommand = iQueuedCommand;
			iPacketBuf.Copy(iVersionQueuedPacketBuf);
			WriteCommand();
		}
	}
}

TInt CBluetoothHandler::ParseListPacketL(TDes8& aPacket)
{	TChar c;
	TUint8 nodeType = 0x0;
	TUint8 fileType;
	TInt packetPos = 0;
	TInt packetLength = aPacket.Length();
	TPtrC8 namePtr;
	CDirTreeNode* previousNode;
	CDirTreeNode* siblingPtr;
	while (packetPos < packetLength)
	{	c = aPacket[packetPos];
		nodeType = (TUint8) ((c >> 4) & 0xF);
		fileType = (TUint8) (c & 0xF);
		namePtr.Set(ReadListFileName(aPacket, packetPos));
		if (nodeType == KNodeTypeLastNode)
			return 0;
		if ((namePtr.Length() == 0) && (nodeType != KNodeTypeRoot))
		{	aPacket.Delete(0, packetPos);
			return 1;
		}
		previousNode = iDirTreeCurrentNode;
		switch (nodeType)
		{
```

```
case KNodeTypeRoot:
        if (!iDirTreeRoot)
        {
        iDirTreeRoot = CDirTreeNode::NewL(nodeType, fileType, namePtr);
        iDirTreeCurrentNode = iDirTreeRoot;
        iDirTreeCurrentNode->iFirstChild = NULL;
        }
        else
        {}
        break;
case KNodeTypeChild:
        if (previousNode)
        {
        iDirTreeCurrentNode = CDirTreeNode::NewL(nodeType, fileType, namePtr);
        iDirTreeCurrentNode->iParent = previousNode;
        iDirTreeCurrentNode->iFirstChild = NULL;
        if (previousNode->iFirstChild)
        {
        siblingPtr = previousNode->iFirstChild;
        while (siblingPtr->iNextSibling)
        siblingPtr = siblingPtr->iNextSibling;
        siblingPtr->iNextSibling = iDirTreeCurrentNode;
        }
        else
        {
        previousNode->iFirstChild = iDirTreeCurrentNode;
        }       }
        else
        {
        return KErrCorrupt;
        }
        break;
case KNodeTypeOnlyChild:
        if (previousNode)
        {
        iDirTreeCurrentNode = CDirTreeNode::NewL(nodeType, fileType, namePtr);
        iDirTreeCurrentNode->iParent = previousNode;
        previousNode->iFirstChild = iDirTreeCurrentNode;
        }
        else
        {
        return KErrCorrupt;
        }
        break;
case KNodeTypeSibling:
        while (previousNode &&((previousNode->iNodeType == KNodeTypeOnlyChild) ||
                (previousNode->iNodeType == KNodeTypeLastSibling)))
        {
        previousNode = previousNode->iParent;
        }
        if (previousNode)
        {
        iDirTreeCurrentNode = CDirTreeNode::NewL(nodeType, fileType, namePtr);
        iDirTreeCurrentNode->iParent = previousNode->iParent;
        previousNode->iNextSibling = iDirTreeCurrentNode;
        }
        else
        {       return KErrCorrupt;
        }
        break;
case KNodeTypeLastSibling:
        while (previousNode && ((previousNode->iNodeType ==  KNodeTypeOnlyChild) ||
                (previousNode->iNodeType == KNodeTypeLastSibling)))
        {
        previousNode = previousNode->iParent;
        }
```

```
                                if (previousNode)
                                {
                                iDirTreeCurrentNode = CDirTreeNode::NewL(nodeType, fileType, namePtr);
                                iDirTreeCurrentNode->iParent = previousNode->iParent;
                                previousNode->iNextSibling = iDirTreeCurrentNode;
                                }
                                else
                                {
                                return KErrCorrupt;
                                }
                                break;
                        default:
                                return KErrCorrupt;
                                break;
                        }
                }
                aPacket.Zero();
                return 1;
}

TInt CBluetoothHandler::ParsePlaylistPacketL(TDes8& aPacket)
{       if (iPlaylistPos < -1)
        {       if (aPacket.Length() >= 2)
                {       iPlaylistPos = (aPacket[0] << 8) + aPacket[1];
                        iListener->SetPlaylistIndex(iPlaylistPos);
                        aPacket.Delete(0, 2);
                }
                else
                {       return 1;
                }
        }
        TInt newlinePos;
        while ((newlinePos = aPacket.Locate('\n')) >= 0)
        {       if (newlinePos == 0)
                {       aPacket.Delete(0, newlinePos + 1);
                        continue;
                }
                TPtrC8 linePtr = aPacket.Left(newlinePos);
                if (linePtr[newlinePos-1] == '\r')
                        linePtr.Set(aPacket.Left(newlinePos-1));
                if (linePtr[0] != '#')
                {       if (linePtr.Locate('\\') != KErrNotFound)
                                linePtr.Set(linePtr.Mid(linePtr.LocateReverse('\\') + 1));
                        if (linePtr.FindF(_L8(".mp3")) != KErrNotFound)
                                linePtr.Set(linePtr.Left(linePtr.FindF(_L8(".mp3"))));
                        HBufC8* lineBuf = linePtr.AllocL();
                        iListener->Playlist().Append(lineBuf);
                }
                aPacket.Delete(0, newlinePos + 1);
        }
        if ((iPlaylistPos >= -1) && (aPacket.Locate('\0') != KErrNotFound))
        {       aPacket.Zero();
                return 0;
        }
        return 1;
}

TPtrC8 CBluetoothHandler::ReadListFileName(const TDesC8& aPacket, TInt& aStartPos)
{       TInt startPos = aStartPos + 1;
        TInt endPos = aStartPos + 1;
        TInt packetLength = aPacket.Length();
        while ((endPos < packetLength) && (aPacket[endPos] != '\0'))
                endPos++;
        if (endPos < packetLength)
        {       aStartPos = endPos + 1;
                return aPacket.Mid(startPos, endPos - startPos);
        }
```

```
	return KNullDesC8();
}

void CBluetoothHandler::SortDirTree(CDirTreeNode* aDirectory)
{	if (!aDirectory || !aDirectory->iFirstChild)
	return;
	RPointerArray<CDirTreeNode> sortArray;
	TLinearOrder<CDirTreeNode> order(CompareNodes);
	CDirTreeNode* searchNode = aDirectory->iFirstChild;
	while (searchNode)
	{	sortArray.Append(searchNode);
		searchNode = searchNode->iNextSibling;
	}
	sortArray.Sort(order);
	aDirectory->iFirstChild = sortArray[0];
	aDirectory->iFirstChild->iNodeType = KNodeTypeChild;
	searchNode = aDirectory->iFirstChild;
	TInt count = sortArray.Count();
	for (TInt i = 1; i < count; i++)
	{	searchNode->iNextSibling = sortArray[i];
		searchNode->iNextSibling->iNodeType = KNodeTypeSibling;
		searchNode = searchNode->iNextSibling;
	}
	searchNode->iNodeType = KNodeTypeLastSibling;
	searchNode->iNextSibling = NULL;
	sortArray.Reset();
	searchNode = aDirectory->iFirstChild;
	while (searchNode)
	{	SortDirTree(searchNode);
		searchNode = searchNode->iNextSibling;
	}
}

TInt CBluetoothHandler::CompareNodes(const CDirTreeNode& aNode1, const CDirTreeNode& aNode2)
{	if (((aNode1.iFileType == KFileTypeDir) || (aNode1.iFileType == FileTypeUnexpandedDir)) &&
		(aNode2.iFileType == KFileTypeFile))
	{	return -1;
	}
	if (((aNode2.iFileType == KFileTypeDir) || (aNode2.iFileType == KFileTypeUnexpandedDir)) &&
		(aNode1.iFileType == KFileTypeFile))
	{	return 1;
	}
	return aNode1.iFileName->CompareF(*aNode2.iFileName);
}

void CBluetoothHandler::DoCancel()
{	switch (iState)
	{
	case ESelectingServer:
		iNotifier.CancelNotifier(KDeviceSelectionNotifierUid);
		break;
	case EInquiringService:
		break;
	case EConnecting:
		iSocket.CancelConnect();
		break;
	case EReading:
		iSocket.CancelRead();
		break;
	case EWriting:
		iSocket.CancelWrite();
		break;
	default:
		break;
	}
}
```

```
TInt CBluetoothHandler::RunError(TInt aError)
{	TBuf<256> formatBuf;
	if (aError != KErrNone)
	{	formatBuf.Format(_L("Unexpected error %d in state %d"), aError, iState);
		CEikonEnv::Static()->InfoWinL(formatBuf, KNullDesC);
	}
	return KErrNone;
}

TBool CBluetoothHandler::CommandExpectsResponse(TBlueTCommand aCommand)
{	if ((aCommand == EBlueTList) || (aCommand == EBlueTCurrentTrackInfo) || (aCommand ==
	EBlueTFileInfo) || (aCommand == (EBlueTDownloadFile) || (aCommand == EBlueTFindServer) ||
	(aCommand == EBlueTGetPlaylist) || (aCommand == EBlueTDetailedTrackInfo) || (aCommand ==
	EBlueTGetVersion) || (aCommand == EBlueTGetVolume))
	{	return ETrue;
	}
	else
	{	return EFalse;
	}
}

void CBluetoothHandler::AttributeRequestComplete(TSdpServRecordHandle /*aHandle*/, TInt aError)
{	if (aError != KErrNone)
	{	ConnectionFailed(aError);
		return;
	}
	iSdpCurrentRecordCount++;
	if (iSdpCurrentRecordCount >= iSdpTotalRecordCount)
	{	TBTSockAddr addr;
		addr.SetBTAddr(iDeviceFoundPckg().BDAddr());
		addr.SetPort(iRemotePort);
		iState = EConnecting;
		iSocket.Connect(addr, iStatus);
		SetActive();
	}
	else
	{
		iSdpAgent->NextRecordRequestL();
	}
}

void CBluetoothHandler::AttributeRequestResult(TSdpServRecordHandle /*aHandle*/, TSdpAttributeID aAttrID,
	CSdpAttrValue* aAttrValue)
{	(void) aAttrID;
	(void) aAttrValue;
	if (aAttrValue->Type() == ETypeDES)
	{	CSdpAttrValueDES* attrValueDES = (CSdpAttrValueDES*) aAttrValue;
		TRAPD(err, attrValueDES->AcceptVisitorL(*this));
	}
	delete aAttrValue;
}

void CBluetoothHandler::NextRecordRequestComplete(TInt aError, TSdpServRecordHandle aHandle,TInt
	aTotalRecordsCount)
{	if (aError != KErrNone)
	{	ConnectionFailed(aError);
	}
	else
	{	iSdpTotalRecordCount = aTotalRecordsCount;
		iSdpAgent->AttributeRequestL(aHandle, KSdpAttrIdProtocolDescriptorList);
	}
}

void CBluetoothHandler::VisitAttributeValueL(CSdpAttrValue &aValue, TSdpElementType aType)
{	(void) aType;
	if (aValue.Type() == ETypeUUID)
	{}
```

```
		else if (aValue.Type() == ETypeUint)
		{	iRemotePort = aValue.Uint();
		}
}

void CBluetoothHandler::StartListL(CSdpAttrValueList& /*aList*/)
{}

void CBluetoothHandler::EndListL()
{}

void CBluetoothHandler::ConnectionFailed(TInt aError)
{	if (iQueuedCommand != EBlueTNone)
	{	iListener->CommandComplete(iQueuedCommand, aError);
		iQueuedCommand = EBlueTNone;
	}
	else
	{	iListener->ConnectComplete(aError, iDeviceFoundPckg().BDAddr(), iRemotePort);
	}
	CloseAllResources();
}

void CBluetoothHandler::DialogDismissedL(TInt /*aButtonId*/)
{	iListener->CommandComplete(iCurrentCommand, KErrCancel);
	Cancel();
	iState = EIdle;
	iConnected = EFalse;
	iCurrentCommand = EBlueTNone;
	iQueuedCommand = EBlueTNone;
	iCommandTimer->Cancel();
	iSongListFile.Close();
	iDownloadFile.Close();
	iSocket.Close();
	iProgressDialog = NULL;
}

TInt CBluetoothHandler::CommandTimerCallBack(TAny* aBluetoothHandler)
{	((CBluetoothHandler*)aBluetoothHandler)->CommandTimedOut();
	return 0;
}

void CBluetoothHandler::CommandTimedOut()
{	Cancel();
	if (iCurrentCommand == EBlueTFindServer)
	{	FindServerNextChannel();
	}
	else if (iCurrentCommand == EBlueTGetVersion)
	{	iVersionState = EGotVersion;
		iCurrentCommand = iQueuedCommand;
		iPacketBuf.Copy(iVersionQueuedPacketBuf);
		WriteCommand();
	}
	else
	{	iCommandTimer->Cancel();
		iListener->CommandComplete(iCurrentCommand, KErrTimedOut);
		CloseAllResources();
	}
}

void CBluetoothHandler::FindServerNextChannel()
{	iCommandTimer->Cancel();
	iSocket.Close();
	iSocket.Open(iSocketServ, KBTAddrFamily, KSockStream, KRFCOMM);
	iFindServerPort++;
	if (iFindServerPort > 30)
	{	CloseAllResources();
		TBuf<256> resourceBuf;
```

```
                iListener->CoeEnv()->ReadResource(resourceBuf,
                R_FIND_BLUET_SERVER_ERROR);
                CEikonEnv::Static()->InfoWinL(resourceBuf, KNullDesC);
                iListener->CommandComplete(EBlueTFindServer, KErrNone);
                return;
        }
        if (iProgressInfo)
                iProgressInfo->IncrementAndDraw(1);
        TBTSockAddr addr;
        addr.SetBTAddr(iDeviceFoundPckg().BDAddr());
        addr.SetPort(iFindServerPort);
        iState = EConnecting;
        iQueuedCommand = EBlueTFindServer;
        iSocket.Connect(addr, iStatus);
        SetActive();
}

void CBluetoothHandler::ShowWaitDialogL(TInt aResourceId)
{       TBuf<256> resourceBuf;
        iListener->CoeEnv()->ReadResource(resourceBuf, aResourceId);
        if (!iWaitDialog)
        {       iWaitDialog = new (ELeave) CAknWaitDialog(REINTERPRET_CAST
                        (CEikDialog**,&iWaitDialog));
                iWaitDialog->SetTone(CAknNoteDialog::EConfirmationTone);
                iWaitDialog->SetTextL(resourceBuf);
                iWaitDialog->ExecuteLD(R_WAIT_NOTE);
        }
        else
        {       iWaitDialog->SetTextL(resourceBuf);
        }
}
```

BlueTPlaylistView.cpp

```
#include <aknviewappui.h>
#include <avkon.hrh>
#include <BlueT.rsg>
#include "BlueT.hrh"
#include "AknQueryDialog.h"
#include "BlueTPlaylistView.h"
#include "BlueTAppUi.h"

void CBlueTPlaylistView::ConstructL()
{       BaseConstructL(R_BLUET_PLAYLIST_VIEW);
        iTextForegroundColour = KRgbBlack;
        iTextBackgroundColour = KRgbWhite;
        iHighlightForegroundColour = KRgbBlack;
        iHighlightBackgroundColour = TRgb(0x80, 0x80, 0xFF);
}

CBlueTPlaylistView::~CBlueTPlaylistView()
{       if (iListBox)
                AppUi()->RemoveFromViewStack(*this, iListBox);
                delete iListBox;
                delete iListItems;
}

TUid CBlueTPlaylistView::Id() const
{
        return KPlaylistViewId;
}

void CBlueTPlaylistView::HandleCommandL(TInt aCommand)
{       switch (aCommand)
        {
```

```
		case EAknSoftkeyOk:
			break;
		case EAknSoftkeyExit:
			AppUi()->HandleCommandL(EEikCmdExit);
			break;
		default:
			AppUi()->HandleCommandL(aCommand);
			break;
		}
}

void CBlueTPlaylistView::HandleClientRectChange()
{	if (iListBox)
		iListBox->SetRect(ClientRect());
}

void CBlueTPlaylistView::DoActivateL(const TVwsViewId& /*aPrevViewId*/, TUid /*aCustomMessageId*/,
		const TDesC8& /*aCustomMessage*/)
{	if (!iListBox)
	{	CBlueTAppUi* appUi = (CBlueTAppUi*) AppUi();
		const CFont* font;
		switch (appUi->Settings().iSongBrowserFont)
		{
		case TBlueTSettings::EHuge:
			font = CEikonEnv::Static()->TitleFont();
			break;
		case TBlueTSettings::ELarge:
			font = CEikonEnv::Static()->NormalFont();
			break;
		case TBlueTSettings::EMedium:
			font = CEikonEnv::Static()->LegendFont();
			break;
		case TBlueTSettings::ESmall:
		default:
			font = CEikonEnv::Static()->DenseFont();
			break;
		}
		iListItems = new (ELeave) CDesC16ArrayFlat(4);
		iListBox = new (ELeave) CBlueTListBox(font);
		iListBox->ConstructL(NULL, EAknListBoxLoopScrolling | CEikListBox::EPaintedSelection);
		iListBox->Model()->SetOwnershipType(ELbmDoesNotOwnItemArray);
		iListBox->Model()->SetItemTextArray(iListItems);
		TBuf<256> resourceBuf;
		((CBlueTAppUi*)AppUi())->CoeEnv()->ReadResource(resourceBuf, R_EMPTY_PLAYLIST);
		iListBox->View()->SetListEmptyTextL(resourceBuf);
		iListBox->View()->SetBackColor(iTextBackgroundColour);
		iListBox->SetItemHeightL(KItemHeight);
		iScrollBarFrame = iListBox->CreateScrollBarFrameL(ETrue);
		iListBox->ScrollBarFrame()->SetScrollBarVisibilityL(CEikScrollBarFrame::EOff,
			CEikScrollBarFrame::EAuto);
		if (appUi->Settings().iDisplayType == TBlueTSettings::EFullScreen)
			iListBox->SetRect(TRect(0, 0, KFullScreenWidth, KFullScreenHeight));
		else
			iListBox->SetRect(ClientRect());
			iListBox->ActivateL();
	}
	CBlueTListItemDrawer* drawer = (CBlueTListItemDrawer*) iListBox->View()->ItemDrawer();
	drawer->SetColours(iTextForegroundColour, iTextBackgroundColour, iHighlightForegroundColour,
		iHighlightBackgroundColour);
	iListBox->DrawNow();
	iFocused = ETrue;
}

void CBlueTPlaylistView::DoDeactivate()
{	if (iListBox)
		AppUi()->RemoveFromViewStack(*this, iListBox);
		delete iListBox;
```

```
                iListBox = NULL;
                iFocused = EFalse;
}

TInt CBlueTPlaylistView::GetSelectedFile()
{       TInt index = iListBox->CurrentItemIndex();
        if ((index < 0) || (index >= iListBox->Model()->ItemTextArray()->MdcaCount()))
        return KErrNotFound;
        return index;
}

void CBlueTPlaylistView::ScrollSelectedSong()
{       TInt index = iListBox->CurrentItemIndex();
        if ((index < 0) || (index >= iListBox->Model()->ItemTextArray()->MdcaCount()))
        return;
        if (((CBlueTListItemDrawer*) iListBox->View()->ItemDrawer())->ScrollSong(index))
        iListBox->DrawNow();
}

void CBlueTPlaylistView::ReadSkinDescriptionL(const TDesC& aFileName)
{       iTextForegroundColour = KRgbBlack;
        iTextBackgroundColour = KRgbWhite;
        iHighlightForegroundColour = KRgbBlack;
        iHighlightBackgroundColour = TRgb(0x80, 0x80, 0xFF);
        RFile file;
        User::LeaveIfError(file.Open(CEikonEnv::Static()->FsSession(),aFileName, EFileShareAny | EFileRead));
        CleanupClosePushL(file);
        TInt fileSize;
        User::LeaveIfError(file.Size(fileSize));
        HBufC8* fileBuf = HBufC8::NewLC(fileSize);
        TPtr8 fileBufPtr = fileBuf->Des();
        User::LeaveIfError(file.Read(fileBufPtr));
        TLex8 lex(*fileBuf);
        TPtrC8 ptr(lex.NextToken());
        while (ptr != KNullDesC8)
        {       lex.SkipSpace();
                if (!ptr.CompareF(_L8("textforegroundcolour:")))
                        ReadSkinColour(lex, iTextForegroundColour);
                else if (!ptr.CompareF(_L8("textbackgroundcolour:")))
                        ReadSkinColour(lex, iTextBackgroundColour);
                else if (!ptr.CompareF(_L8("highlightforegroundcolour:")))
                        ReadSkinColour(lex, iHighlightForegroundColour);
                else if (!ptr.CompareF(_L8("highlightbackgroundcolour:")))
                        ReadSkinColour(lex, iHighlightBackgroundColour);
                else
                        lex.SkipSpace();
                ptr.Set(lex.NextToken());
        }
        CleanupStack::PopAndDestroy(2);
}

void CBlueTPlaylistView::ReadSkinColour(TLex8& aLex, TRgb& aColour)
{       TInt colour;
        aLex.Val(colour);
        aColour.SetRed(colour);
        aLex.SkipSpace();
        aLex.Val(colour);
        aColour.SetGreen(colour);
        aLex.SkipSpace();
        aLex.Val(colour);
        aColour.SetBlue(colour);
        aLex.SkipSpace();
}

TBool CBlueTPlaylistView::IsFocused()
{       return iFocused;
}
```

```
void CBlueTPlaylistView::RefreshPlaylistL(const RPointerArray<HBufC8>&  aPlaylist, TInt aIndex)
{	((CBlueTListItemDrawer*) iListBox->View()->ItemDrawer())->ResetScrolling();
	iListItems->Reset();
	for (TInt i = 0; i < aPlaylist.Count(); i++)
	{	HBufC* fileBuf = HBufC::NewLC(aPlaylist[i]->Length());
		fileBuf->Des().Copy(*aPlaylist[i]);
		iListItems->AppendL(*fileBuf);
		CleanupStack::PopAndDestroy();
	}
	iListBox->Model()->SetItemTextArray(iListItems);
	iListBox->HandleItemAdditionL();
	if ((aIndex >= 0) && (aIndex <= aPlaylist.Count()))
		iListBox->SetCurrentItemIndex(aIndex);
	iListBox->DrawNow();
}

TKeyResponse CBlueTPlaylistView::OfferKeyEventL(const TKeyEvent&	aKeyEvent, TEventCode aType)
{	CBlueTAppUi* appUi = ((CBlueTAppUi*) AppUi());
	switch (aKeyEvent.iCode)
	{
	case EKeyDevice3:
	{	TInt selection = GetSelectedFile();
		if (selection != KErrNotFound)
			appUi->SelectInPlaylist(selection);
	}
		break;
	case EKeyUpArrow:
	case EKeyDownArrow:
		((CBlueTListItemDrawer*) iListBox->View()->ItemDrawer())->ResetScrolling();
		return iListBox->OfferKeyEventL(aKeyEvent, aType);
		break;
	case '0':
	{	((CBlueTListItemDrawer*) iListBox->View()->ItemDrawer())->ResetScrolling();
		TInt currentIndex = iListBox->CurrentItemIndex();
		TInt itemsOnScreen = 6;
		if (appUi->Settings().iDisplayType == TBlueTSettings::EFullScreen)
				itemsOnScreen = 8;
		if (iListBox->Model()->ItemTextArray()->MdcaCount() > (currentIndex + itemsOnScreen))
		{	iListBox->SetCurrentItemIndex(currentIndex + itemsOnScreen);
		}
		else
		{	iListBox->SetCurrentItemIndex(iListBox->Model()->ItemTextArray()->MdcaCount() - 1);
		}
		iListBox->DrawNow();
	}
		break;
	case '2':
	{	((CBlueTListItemDrawer*) iListBox->View()->ItemDrawer())->ResetScrolling();
		TInt currentIndex = iListBox->CurrentItemIndex();
		TInt itemsOnScreen = 6;
		if (appUi->Settings().iDisplayType == TBlueTSettings::EFullScreen)
				itemsOnScreen = 8;
		if (currentIndex > itemsOnScreen)
		{	iListBox->SetCurrentItemIndex(currentIndex - itemsOnScreen);
		}
		else
		{	iListBox->SetCurrentItemIndex(0);
		}
		iListBox->DrawNow();
	}
		break;
	case '3':
		FindSongL(EFalse);
		break;
	case '6':
		FindSongL(ETrue);
		break;
```

```
		default:
			return EKeyWasNotConsumed;
			break;
		}
		return EKeyWasConsumed;
}

void CBlueTPlaylistView::FindSongL(TBool aFindNext)
{	TFileName lastSearchedSong = iLastSearchedSong;
		CAknTextQueryDialog* dialog = NULL;
		if (aFindNext)
		{	if (iLastSearchedSong == KNullDesC)
			return;
		}
		else
		{	dialog = CAknTextQueryDialog::NewL(iLastSearchedSong);
			CleanupStack::PushL(dialog);
			dialog->SetPredictiveTextInputPermitted(ETrue);
		}
		iResultsFound = 0;
		if (aFindNext || dialog->ExecuteLD(R_FIND_DIALOG))
		{	if (iLastSearchedSong != lastSearchedSong)
			iFindResultsToSkip = 0;
			TInt count = iListBox->Model()->ItemTextArray()->MdcaCount();
			TInt resultsFound = 0;
			for (TInt i = 0; i < count; i++)
			{	if ((*iListItems)[i].FindF(iLastSearchedSong) != KErrNotFound)
				{
				resultsFound++;
				if (resultsFound > iFindResultsToSkip)
				{
				((CBlueTListItemDrawer*) iListBox->View()->ItemDrawer())->ResetScrolling();
				iListBox->SetCurrentItemIndex(i);
				iListBox->DrawNow();
				iFindResultsToSkip++;
				if (!aFindNext)
				CleanupStack::Pop();
				return;
				}		}
			}
			lastSearchedSong.Format(_L("Couldn't find \"%S\""), &iLastSearchedSong);
			CEikonEnv::Static()->InfoWinL(lastSearchedSong, KNullDesC);
			iFindResultsToSkip = 0;
		}
		if (!aFindNext)
			CleanupStack::Pop();
}
```

BlueTSkinHandler.cpp

```
#include <eikenv.h>
#include <BlueT.rsg>
#include "BlueTSkinHandler.h"
#include "BlueTAppUi.h"
#include "BlueTControlView.h"
#include "BlueTBrowseView.h"
#include "BlueTPlaylistView.h"
#include "BlueTControlContainer.h"

CBlueTSkinHandler::CBlueTSkinHandler(CBlueTAppUi* aAppUi, TBlueTSkin& aSkin): iAppUi(aAppUi), iSkin(aSkin)
{}

CBlueTSkinHandler* CBlueTSkinHandler::NewL(CBlueTAppUi* aAppUi, TBlueTSkin& aSkin)
{	CBlueTSkinHandler* self = new (ELeave) CBlueTSkinHandler(aAppUi, aSkin);
		CleanupStack::PushL(self);
```

```
	self->ConstructL();
	CleanupStack::Pop();
	return self;
}

void CBlueTSkinHandler::ConstructL()
{	iSkin.iBitmap = new (ELeave) CFbsBitmap;
	User::LeaveIfError(iSkin.iBitmap->Create(TSize(KMaxSkinBitmapWidth, KMaxSkinBitmapHeight),
		EColor16M));
	iSkinLoader = CMdaImageFileToBitmapUtility::NewL(*this);
	TRAPD(err, LoadSkinL());
	if (err != KErrNone)
	{	TBuf<256> resourceBuf;
		iAppUi->CoeEnv()->ReadResource(resourceBuf, R_LOAD_SKIN_ERROR);
		CEikonEnv::Static()->InfoWinL(resourceBuf, KNullDesC);
	}
}

CBlueTSkinHandler::~CBlueTSkinHandler()
{	iSkinLoader->Close();
	delete iSkinLoader;
	delete iSkinScaler;
	delete iSkin.iBitmap;
}

void CBlueTSkinHandler::MiuoConvertComplete(TInt aError)
{	if (!iScaling)
	{	iSkinLoader->Close();
		if (aError != KErrNone)
		{	TBuf<256> resourceBuf;
			iAppUi->CoeEnv()->ReadResource(resourceBuf, R_LOAD_SKIN_ERROR);
			TRAPD(err, CEikonEnv::Static()->InfoWinL(KNullDesC, resourceBuf));
		}
		else
		{	iSkin.iValid = ETrue;
		}
		if (iAppUi->ControlView()->ControlContainer())
			iAppUi->ControlView()->ControlContainer()->DrawNow();
	}
	else
	{	if (aError != KErrNone)
		{	TBuf<256> resourceBuf;
			iAppUi->CoeEnv()->ReadResource(resourceBuf, R_LOAD_SKIN_ERROR);
			TRAPD(err, CEikonEnv::Static()->InfoWinL(KNullDesC, resourceBuf));
		}
		else
		{	iSkin.iValid = ETrue;
		}
		iAppUi->DrawControlView(KBlueTDrawEverything);
		iScaling = EFalse;
	}
}

void CBlueTSkinHandler::MiuoCreateComplete(TInt /*aError*/)
{}

void CBlueTSkinHandler::MiuoOpenComplete(TInt aError)
{	if (aError == KErrNone)
	{	TRAP(aError, iSkinLoader->ConvertL(*iSkin.iBitmap));
		if (aError != KErrNone)
		{	TBuf<256> resourceBuf;
			iAppUi->CoeEnv()->ReadResource(resourceBuf,
			R_LOAD_SKIN_ERROR);
			TRAP(aError, CEikonEnv::Static()->InfoWinL(resourceBuf, KNullDesC));
		}
	}
}
```

```
void CBlueTSkinHandler::LoadSkinL()
{	iSkin.iValid = EFalse;
	CDir* entryList;
	TBuf<256> formatBuf;
	TFileName skinPath;
	TInt ret;
	TBlueTSettings::TDisplayType displayType = BlueTSettings::TDisplayType)
		iAppUi->Settings().iDisplayType;
	TInt skinIndex = iAppUi->Settings().iSkinFileIndex;
	if (displayType == TBlueTSettings::EInWindow)
		skinPath.Format(_L("%S\\*_iw.png"), &iAppUi->SkinPathL());
	else
		skinPath.Format(_L("%S\\*_fs.png"), &iAppUi->SkinPathL());
	ret = CEikonEnv::Static()->FsSession().GetDir(skinPath, KEntryAttNormal,ESortByName, entryList);
	if (ret != KErrNone)
	{	delete entryList;
		TBuf<256> resourceBuf;
		iAppUi->CoeEnv()->ReadResource(resourceBuf, R_SKIN_LIST_ERROR);
		formatBuf.Format(resourceBuf, ret);
		CEikonEnv::Static()->InfoWinL(formatBuf, KNullDesC);
		return;
	}
	CleanupStack::PushL(entryList);
	if (entryList->Count() <= skinIndex)
	{	iAppUi->CoeEnv()->ReadResource(formatBuf, R_LOAD_SKIN_ERROR);
		CEikonEnv::Static()->InfoWinL(formatBuf, KNullDesC);
	}
	else
	{	const TDesC& entryName = (*entryList)[skinIndex].iName;
		TPtrC entryWithoutExt = entryName.Left(entryName.Length() - 4);
		iAppUi- SetSkinName(entryWithoutExt.Left (entryWithoutExt.Length() - 3));
		formatBuf.Format(_L("%S\\%S"), &iAppUi->SkinPathL(), &entryName);
		iSkinLoader->OpenL(formatBuf);
		formatBuf.Format(_L("%S\\%S.txt"), &iAppUi->SkinPathL(), &entryWithoutExt);
		ReadSkinDescriptionL(formatBuf);
		iAppUi->BrowseView()->ReadSkinDescriptionL(formatBuf);
		iAppUi->PlaylistView()->ReadSkinDescriptionL(formatBuf);
	}
	CleanupStack::PopAndDestroy();
}

void CBlueTSkinHandler::ReadSkinDescriptionL(const TDesC& aFileName)
{	iSkin.iButtonsRect.SetRect(0, 105, 175, 140);
	iSkin.iTrackNameRect.SetRect(10, 10, 165, 36);
	iSkin.iTrackTimeRect.SetRect(10, 40, 165, 76);
	iSkin.iShuffleRect.SetRect(10, 80, 20, 90);
	iSkin.iRepeatRect.SetRect(10, 100, 20, 110);
	iSkin.iVolumeRect.SetRect(170, 80, 175, 120);
	iSkin.iTrackNameColour = KRgbBlack;
	iSkin.iTrackTimeColour = KRgbBlack;
	iSkin.iTrackNameFont = EBlueTTitleFont;
	iSkin.iTrackTimeFont = EBlueTTitleFont;
	iAppUi->SetSkinAuthor(KNullDesC8);
	RFile file;
	User::LeaveIfError(file.Open(CEikonEnv::Static()->FsSession(),aFileName, EFileShareAny | EFileRead));
	CleanupClosePushL(file);
	TInt fileSize;
	User::LeaveIfError(file.Size(fileSize));
	HBufC8* fileBuf = HBufC8::NewLC(fileSize);
	TPtr8 fileBufPtr = fileBuf->Des();
	User::LeaveIfError(file.Read(fileBufPtr));
	TLex8 lex(*fileBuf);
	TPtrC8 ptr(lex.NextToken());
	while (ptr != KNullDesC8)
	{	lex.SkipSpace();
		if (!ptr.CompareF(_L8("buttons:")))
		{	ReadSkinRect(lex, iSkin.iButtonsRect);	}
```

```
                else if (!ptr.CompareF(_L8("trackname:")))
                {       ReadSkinRect(lex, iSkin.iTrackNameRect);
                }
                else if (!ptr.CompareF(_L8("tracktime:")))
                {       ReadSkinRect(lex, iSkin.iTrackTimeRect);
                }
                else if (!ptr.CompareF(_L8("shuffle:")))
                {       ReadSkinRect(lex, iSkin.iShuffleRect);
                }
                else if (!ptr.CompareF(_L8("repeat:")))
                {       ReadSkinRect(lex, iSkin.iRepeatRect);
                }
                else if (!ptr.CompareF(_L8("volume:")))
                {       ReadSkinRect(lex, iSkin.iVolumeRect);
                        lex.Val(iSkin.iVolumeSliderSize.iWidth);
                        lex.SkipSpace();
                        lex.Val(iSkin.iVolumeSliderSize.iHeight);
                        lex.SkipSpace();
                }
                else if (!ptr.CompareF(_L8("tracknamecolour:")))
                {       ReadSkinColour(lex, iSkin.iTrackNameColour);
                }
                else if (!ptr.CompareF(_L8("tracktimecolour:")))
                {       ReadSkinColour(lex, iSkin.iTrackTimeColour);
                }
                else if (!ptr.CompareF(_L8("tracknamefont:")))
                {       TInt font;
                        lex.Val(font);
                        iSkin.iTrackNameFont = (TBlueTFont) font;
                        lex.SkipSpace();
                }
                else if (!ptr.CompareF(_L8("tracktimefont:")))
                {       TInt font;
                        lex.Val(font);
                        iSkin.iTrackTimeFont = (TBlueTFont) font;
                        lex.SkipSpace();
                }
                else if (!ptr.CompareF(_L8("author:")))
                {       TChar c;
                        TBuf8<128> authorBuf;
                        c = lex.Get();
                        while ((c != '\r') && (c != '\n') && (c != 0))
                        {       authorBuf.Append(c);
                                c = lex.Get();
                        }
                        iAppUi->SetSkinAuthor(authorBuf);
                        lex.SkipSpace();
                }
                else
                {       lex.SkipSpace();
                }
                ptr.Set(lex.NextToken());
        }
        CleanupStack::PopAndDestroy(2);
}

void CBlueTSkinHandler::ReadSkinColour(TLex8& aLex, TRgb& aColour)
{       TInt colour;
        aLex.Val(colour);
        aColour.SetRed(colour);
        aLex.SkipSpace();
        aLex.Val(colour);
        aColour.SetGreen(colour);
        aLex.SkipSpace();
        aLex.Val(colour);
        aColour.SetBlue(colour);
        aLex.SkipSpace();               }
```

```
void CBlueTSkinHandler::ReadSkinRect(TLex8& aLex, TRect& aRect)
{	aLex.Val(aRect.iTl.iX);
	aLex.SkipSpace();
	aLex.Val(aRect.iTl.iY);
	aLex.SkipSpace();
	aLex.Val(aRect.iBr.iX);
	aLex.SkipSpace();
	aLex.Val(aRect.iBr.iY);
	aLex.SkipSpace();
}

const CFont* CBlueTSkinHandler::BlueTFontToCFont(TBlueTFont aBlueTFont)
{	CEikonEnv* eikonEnv = CEikonEnv::Static();
	switch (aBlueTFont)
	{
	case EBlueTTitleFont:
		return eikonEnv->TitleFont();
	case EBlueTDenseFont:
		return eikonEnv->DenseFont();
	case EBlueTLegendFont:
		return eikonEnv->LegendFont();
	case EBlueTAnnotationFont:
		return eikonEnv->AnnotationFont();
	case EBlueTSymbolFont:
		return eikonEnv->SymbolFont();
	case EBlueTNormalFont:
		return eikonEnv->NormalFont();
	default:
		break;
	}
	return NULL;
}
```

DirTreeNode.cpp

```
#include "DirTreeNode.h"

CDirTreeNode::CDirTreeNode(TUint8 aNodeType, TUint8 aFileType) : iNodeType(aNodeType),
	iFileType(aFileType)
{}

CDirTreeNode* CDirTreeNode::NewL(TUint8 aNodeType, TUint8 aFileType, const TDesC8& aFileName)
{	CDirTreeNode* self = new (ELeave) CDirTreeNode(aNodeType, aFileType);
	CleanupStack::PushL(self);
	self->ConstructL(aFileName);
	CleanupStack::Pop();
	return self;
}

void CDirTreeNode::ConstructL(const TDesC8& aFileName)
{	iFileName = aFileName.AllocL();
}

CDirTreeNode::~CDirTreeNode()
{	delete iFirstChild;
	delete iNextSibling;
	delete iFileName;
}
```

IncomingCallListener.cpp

```
#include <eikenv.h>
#include "IncomingCallListener.h"

CIncomingCallListener::CIncomingCallListener(MIncomingCallListener& aListener) :
	CActive(EPriorityStandard), iListener(aListener)
{
	CActiveScheduler::Add(this);
}

CIncomingCallListener* CIncomingCallListener::NewL(MIncomingCallListener& aListener)
{
	CIncomingCallListener* self = new (ELeave)
	CIncomingCallListener(aListener);
	CleanupStack::PushL(self);
	self->ConstructL();
	CleanupStack::Pop();
	return self;
}

void CIncomingCallListener::ConstructL()
{
	User::LeaveIfError(iTelServer.Connect());
	RTelServer::TPhoneInfo phoneInfo;
	User::LeaveIfError(iTelServer.GetPhoneInfo(0, phoneInfo));
	User::LeaveIfError(iPhone.Open(iTelServer, phoneInfo.iName));
	RPhone::TLineInfo lineInfo;
	User::LeaveIfError(iPhone.GetLineInfo(0, lineInfo));
	User::LeaveIfError(iLine.Open(iPhone, lineInfo.iName));
}

CIncomingCallListener::~CIncomingCallListener()
{
	Cancel();
	iLine.Close();
	iPhone.Close();
	iTelServer.Close();
}

void CIncomingCallListener::StartListening()
{
	iLine.NotifyIncomingCall(iStatus, iCallName);
	SetActive();
}

void CIncomingCallListener::RunL()
{
	User::LeaveIfError(iStatus.Int());
	iListener.IncomingCall(iCallName);
}

TInt CIncomingCallListener::RunError(TInt /*aError*/)
{
	return KErrNone;
}

void CIncomingCallListener::DoCancel()
{
	iLine.NotifyIncomingCallCancel();
}
```

BlueTApp.h

```
#ifndef BLUETAPP_H
#define BLUETAPP_H
#include <aknapp.h>

const TUid KUidBlueT = { 0x0B48A2AB };
class CBlueTApp : public CAknApplication
{
private:
   CApaDocument* CreateDocumentL();
   TUid AppDllUid() const;
};
#endif
```

BlueTAppUi.h

```
#ifndef BLUETAPPUI_H__
#define BLUETAPPUI_H__
#include <eikapp.h>
#include <eikdoc.h>
#include <e32std.h>
#include <coeccntx.h>
#include <aknviewappui.h>
#include <akntabgrp.h>
#include <aknnavide.h>
#include "BluetoothHandler.h"
#include "IncomingCallListener.h"

class CBlueTContainer;
class CBlueTAppUi;
class CBlueTBrowseView;
class CBlueTControlView;
class CBlueTDisplaySettingsView;
class CBlueTBehaviourSettingsView;
class CBlueTPlaylistView;

const TInt KFullScreenWidth = 176;
const TInt KFullScreenHeight = 188;
const TUint8 KVolumeStep = 20;
const TUint8 KStartVolume = 130;
const TUint KBlueTDrawBackground = 0x1;
const TUint KBlueTDrawTrackName = 0x2;
const TUint KBlueTDrawTrackTime = 0x4;
const TUint KBlueTDrawButtons = 0x8;
const TUint KBlueTDrawShuffle = 0x10;
const TUint KBlueTDrawRepeat = 0x20;
const TUint KBlueTDrawVolume = 0x40;
const TUint KBlueTDrawEverything = 0xFFFF;

_LIT(KBlueTName, "BlueT");

enum TSongState
{
        ENoSongLoaded = 0,
        ESongPlaying,
        ESongPaused,
        ESongStopped
};

class TBlueTSettings
{
public:
        enum TActionOnIncomingCall
        {
```

```
            EDoNothing,
            EPause,
            EStop,
            EMute
            };
            TInt iActionOnIncomingCall;

            enum TDisplayType
            {
            EInWindow,
            EFullScreen
            };
            TInt iDisplayType;

            enum TTimeDisplay
            {
            ECountsUp,
            ECountsDown
            };
            TInt iTimeDisplay;

            enum TSongBrowserFont
            {
            EHuge,
            ELarge,
            EMedium,
            ESmall
            };
            TInt iSongBrowserFont;

            TInt iSkinFileIndex;
            TInt iInstallSkinIndex;
            TInt iUninstallSkinIndex;

            enum TUseId3Tags
            {
            ENo,
            EYes
            };
            TInt iUseId3Tags;

            TInt iDownloadDrive;
            TBTDevAddr iAutoConnectAddress;
            TInt iAutoConnectPort;
};

class CBlueTAppUi : public CAknViewAppUi, public MBluetoothListener, public MIncomingCallListener
{
public:
   void ConstructL();
   ~CBlueTAppUi();

public:
            virtual void RefreshFileListL(CDirTreeNode* aDirectory);
            virtual void ConnectComplete(TInt aError, const TBTDevAddr& aAddress, TInt
                                    aPort);
            virtual void CommandComplete(TBlueTCommand aCommand, TInt aError);
            virtual void SetInfo(TBool aPlaying, TBool aPaused, TInt aSongLength, TInt aCurrentTime, TBool
aShuffle,               TBool aRepeat, const TDesC8& aSongNameGuess);
            virtual void SetDetailedInfo(TInt aSampleRate, TInt aBitRate, TInt aChannels);
            virtual TBlueTSettings& Settings();
            virtual void SetPlaylistIndex(TInt aIndex);
            virtual void ReceivedVolume(TInt aVolume);
            virtual RPointerArray<HBufC8>& Playlist();
            virtual CCoeEnv* CoeEnv();

            void PlayFile(CDirTreeNode* aFileNode, const TDesC8& aFileName, TBool aAddToPlaylist);
```

```
        void DownloadFile(const TDesC8& aFileName);
        void SelectInPlaylist(TInt aIndex);
        void GetDirectoryList(CDirTreeNode* aDirectory);
        TInt LoadSettings();
        TInt SaveSettings();
        TInt ReloadSkin();
        void DrawControlView(TUint aDrawFlags);
        void SetSkinAuthor(const TDesC8& aAuthor);
        void SetSkinName(const TDesC& aFileName);

        CDirTreeNode* CurrentDirectory();
        void SetCurrentDirectory(CDirTreeNode* aDirectory);
        TUint DrawFlags();
        void SetDrawFlags(TUint aFlags);
        void SetDrawPending(TBool aPending);
        CBluetoothHandler* BluetoothHandler();

        void HandleCommandL(TInt aCommand);
        CBlueTControlView* ControlView();
        CBlueTBrowseView* BrowseView();
        CBlueTPlaylistView* PlaylistView();
        TSongState SongState();
        TInt& PlaylistIndex();
        TInt CurrentTime();
        TInt SongLength();
        TBool Shuffle();
        TBool Repeat();
        TInt CurrentVolume();
        void SetVolume(TInt aVolume);
        const TDesC& SongNameGuess();

        TFileName& SkinPathL();
        void GetPlaylist();

public:
        void IncomingCall(const TDesC& aName);

private:
    virtual TKeyResponse HandleKeyEventL(const TKeyEvent& aKeyEvent,TEventCode aType);

        static TInt SongTimerCallback(TAny* aPtr);
        void HandleSongTimer();
        void BuildIniPath(TFileName& aIniFileName);
        void AddFilesToPlaylist(CDirTreeNode* aNode);
        void DisplayDetailedInfoDialogL(TInt aSampleRate, TInt aBitRate, TInt aNumChannels);

private:
    CAknNavigationControlContainer* iNaviPane;
    CAknTabGroup* iTabGroup;
    CAknNavigationDecorator* iDecoratedTabGroup;
        CEikonEnv* iEikEnv;
        CPeriodic* iSongTimer;
        CIncomingCallListener* iIncomingCallListener;
        CBlueTBrowseView* iBrowseView;
        CBlueTControlView* iControlView;
        CBlueTDisplaySettingsView* iDisplaySettingsView;
        CBlueTBehaviourSettingsView* iBehaviourSettingsView;
        CBlueTPlaylistView* iPlaylistView;
        CBluetoothHandler* iBluetoothHandler;
        CDirTreeNode* iCurrentDirectory;
        TUint8 iCurrentVolume;
        CDirTreeNode* iCurrentFile;
        TInt iSongLength;
        TInt iCurrentTime;
        TSongState iSongState;
        TBool iShuffle;
        TBool iRepeat;
```

```
        TInt iSampleRate;
        TInt iBitRate;
        TInt iChannels;
        TBlueTSettings iSettings;
        TUint iDrawFlags;
        TBool iDrawPending;
        RPointerArray<HBufC8> iPlaylist;
        TInt iPlaylistIndex;
        TFileName iSkinPath;
        TFileName iSongNameGuess;
        TBTDevAddr iLastDeviceAddress;
        TInt iLastPort;
        TBuf8<128> iSkinAuthor;
        TFileName iSkinName;
};
#endif
```

BlueTBehaviourSettingsContainer.h

```
#ifndef BLUET_BEHAVIOUR_SETTINGS_CONTAINER_H__
#define BLUET_BEHAVIOUR_SETTINGS_CONTAINER_H__
#include <coecntrl.h>
#include <eikenv.h>
#include <aknlists.h>
#include <aknsettingitemlist.h>
#include "BlueTAppUi.h"

const TInt KDownloadDriveSettingIndex = 2;
class CBlueTBehaviourSettingsView;
class CBlueTBehaviourSettingItemList : public CAknSettingItemList
{
public:
        CBlueTBehaviourSettingItemList(TBlueTSettings& aData);
        void PopulateDriveLettersL(CAknEnumeratedTextPopupSettingItem* aSettingItem);

protected:
        virtual CAknSettingItem* CreateSettingItemL(TInt aSettingId);

private:
        TBlueTSettings& iData;
};

class CBlueTBehaviourSettingsContainer : public CCoeControl, MCoeControlObserver
{
public:
   void ConstructL(CBlueTAppUi* aAppUi, CBlueTBehaviourSettingsView* aView, const TRect& aRect);
   ~CBlueTBehaviourSettingsContainer();

public:
        TKeyResponse OfferKeyEventL(const TKeyEvent& aKeyEvent, TEventCode  aType);
        CBlueTBehaviourSettingItemList* iListBox;

private:
   void SizeChanged();
   TInt CountComponentControls() const;
   CCoeControl* ComponentControl(TInt aIndex) const;
   void Draw(const TRect& aRect) const;
   void HandleControlEventL(CCoeControl* aControl,TCoeEvent aEventType);

private:
        CEikScrollBarFrame* iScrollBarFrame;
        CBlueTAppUi* iAppUi;
        CBlueTBehaviourSettingsView* iView;
};
#endif
```

BlueTBehaviourSettingsView.h

```
#ifndef BLUET_BEHAVIOUR_SETTINGS_VIEW_H__
#define BLUET_BEHAVIOUR_SETTINGS_VIEW_H__

#include <aknview.h>
#include "BlueTAppUi.h"

const TUid KBehaviourSettingsViewId = {5};
class CBlueTBehaviourSettingsContainer;
class CBlueTBehaviourSettingsView : public CAknView
{
public:
   void ConstructL();
   ~CBlueTBehaviourSettingsView();

public:
   TUid Id() const;
   void HandleCommandL(TInt aCommand);
   void HandleClientRectChange();

public:
            CBlueTBehaviourSettingsContainer* SettingsContainer();

private:
   void DoActivateL(const TVwsViewId& aPrevViewId,TUid aCustomMessageId,
      const TDesC8& aCustomMessage);
   void DoDeactivate();

private:
            CBlueTBehaviourSettingsContainer* iContainer;
};
#endif
```

BlueTBrowseContainer.h

```
#ifndef BLUETBROWSECONTAINER_H__
#define BLUETBROWSECONTAINER_H__

#include <coecntrl.h>
#include <eikenv.h>
#include "BlueTListBox.h"

class CBlueTAppUi;
class CDirTreeNode;

enum TMusicFileType
            {
            EUnknownFile,
            EMusicFile,
            EMp3File
            };

class CBlueTBrowseContainer : public CCoeControl, public MCoeControlObserver
{
public:
   void ConstructL(CBlueTAppUi* aAppUi, const TRect& aRect);
   ~CBlueTBrowseContainer();
```

```
public:
        void FileSelectedL();
        void PlaySelectedFileL(TBool aAddToPlaylist);
        void DownloadSelectedFileL();
        void RefreshFileListL(CDirTreeNode* aDirectory);
        void UpOneLevelL();
        void GetFullNodeFileName(CDirTreeNode* aNode, TDes8& aFileName);
        void SetListEmptyStringL(const TDesC& aEmptyString);
        void ScrollSelectedSong();
        void FindSongL(TBool aFindNext);

        CDirTreeNode* RecursiveFind(CDirTreeNode* aNode, TDes8& aName);
        void SetColours(const TRgb& aTextForeground, const TRgb& aTextBackground, const TRgb&
                        aHighlightForeground, const TRgb& aHighlightBackground);
        TKeyResponse OfferKeyEventL(const TKeyEvent& aKeyEvent, TEventCode  aType);

private:
   void SizeChanged();
   TInt CountComponentControls() const;
   CCoeControl* ComponentControl(TInt aIndex) const;
        void Draw(const TRect& aRect) const;
        void HandleControlEventL(CCoeControl* aControl, TCoeEvent aEventType);

private:
        CDirTreeNode* FindSelectedItem();
        TMusicFileType GetFileType(const TDesC& aFileName);
        TBool FileShouldBeDisplayed(const CDirTreeNode* aNode);

private:
        CBlueTListBox* iListBox;
        CEikScrollBarFrame* iScrollBarFrame;
        CDesC16ArrayFlat* iListItems;
        CBlueTAppUi* iAppUi;
        CListBoxData* iListBoxData;
        CFont* iFont;


        TRgb iTextForegroundColour;
        TRgb iTextBackgroundColour;
        TRgb iHighlightForegroundColour;
        TRgb iHighlightBackgroundColour;
        TInt iSongOffsetIndex;
        TFileName iLastSearchedSong;
        TInt iFindResultsToSkip;
        TInt iResultsFound;
};
#endif
```

BlueTBrowseView.h

```
#ifndef BLUETBROWSEVIEW_H__
#define BLUETBROWSEVIEW_H__
#include <aknview.h>

const TUid KBrowseViewId = {1};

class CBlueTBrowseContainer;
class CBlueTBrowseView : public CAknView
{
public:
   void ConstructL();
   ~CBlueTBrowseView();

public:
   TUid Id() const;
```

```
   void HandleCommandL(TInt aCommand);
   void HandleClientRectChange();

public:
        void ReadSkinDescriptionL(const TDesC& aFileName);
        CBlueTBrowseContainer* BrowseContainer();
        void SetListLoaded(TBool aListLoaded);

private:
   void DoActivateL(const TVwsViewId& aPrevViewId,TUid aCustomMessageId,const TDesC8&
aCustomMessage);
   void DoDeactivate();

private:
        void ReadSkinColour(TLex8& aLex, TRgb& aColour);

private:
   CBlueTBrowseContainer* iContainer;
        TRgb iTextForegroundColour;
        TRgb iTextBackgroundColour;
        TRgb iHighlightForegroundColour;
        TRgb iHighlightBackgroundColour;
        TBool iListLoaded;
};
#endif
```

BluetControlContainer.h

```
#ifndef BLUETCONTROLCONTAINER_H__
#define BLUETCONTROLCONTAINER_H__
#include <coecntrl.h>
#include <eikcmbut.h>
#include "BlueTAppUi.h"

const TInt KNumButtons = 5;

class CBlueTControlView;
class CBlueTControlContainer : public CCoeControl, MCoeControlObserver
{
public:
        void ConstructL(CBlueTAppUi* aAppUi, CBlueTControlView* aView, const TRect& aRect);
        ~CBlueTControlContainer();

public:
        virtual TKeyResponse OfferKeyEventL(const TKeyEvent& aKeyEvent,TEventCode aType);

public:
        void DrawTrackName(const TRect& aRect) const;
        void DrawTrackTime(const TRect& aRect) const;
        void DrawButtons(const TRect& aRect) const;
        void DrawShuffle(const TRect& aRect) const;
        void DrawRepeat(const TRect& aRect) const;
        void DrawVolume(const TRect& aRect) const;
        void AppInitiatedDraw() const;
        void DrawRequiredControls(TUint aRequired) const;
        TBool CheckFastForwardAndRewind();

private:
        void SizeChanged();
        TInt CountComponentControls() const;
```

```
        CCoeControl* ComponentControl(TInt aIndex) const;
        void Draw(const TRect& aRect) const;
        void HandleControlEventL(CCoeControl* aControl,TCoeEvent aEventType);

private:
        void ExecuteCommand(TInt aIndex);
        void GetVolumeSliderArea(TRect& aVolumeSliderArea) const;

private:
        CBlueTControlView* iView;

private:
        TInt iFocusIndex;
        TBool iButtonPressed;
        TBool iCheckedButtonPressedAlready;
        CBlueTAppUi* iAppUi;
        CFbsBitmap* iTrackNameBitmap;
        CFbsBitGc* iTrackNameContext;
        CFbsBitmapDevice* iTrackNameDevice;
        CFbsBitmap* iTrackTimeBitmap;
        CFbsBitGc* iTrackTimeContext;
        CFbsBitmapDevice* iTrackTimeDevice;
        TInt iWidth;
        TInt iHeight;
};
#endif
```

BluetControlView.h

```
#ifndef BLUETCONTROLVIEW_H__
#define BLUETCONTROLVIEW_H__
#include <aknview.h>
#include <MdaImageConverter.h>
#include "BlueTSkinHandler.h"

const TUid KControlViewId = {2};

class CBlueTControlContainer;
class CBlueTControlView : public CAknView
{
public:
   void ConstructL();
   ~CBlueTControlView();

public:
   TUid Id() const;
   void HandleCommandL(TInt aCommand);
   void HandleClientRectChange();

public:
        CBlueTControlContainer* ControlContainer();
        CBlueTSkinHandler* SkinHandler();
        TBlueTSkin& Skin();

private:
   void DoActivateL(const TVwsViewId& aPrevViewId,TUid aCustomMessageId,const TDesC8&
aCustomMessage);
   void DoDeactivate();

private:
        CBlueTSkinHandler* iSkinHandler;

   CBlueTControlContainer* iContainer;
        TBlueTSkin iSkin;
};
#endif
```

BluetDisplaySettingsContainer.h

```
#ifndef BLUET_DISPLAY_SETTINGS_CONTAINER_H__
#define BLUET_DISPLAY_SETTINGS_CONTAINER_H__
#include <coecntrl.h>
#include <eikenv.h>
#include <aknlists.h>
#include <aknsettingitemlist.h>
#include "BlueTAppUi.h"

class CBlueTDisplaySettingsView;

_LIT(KInboxPath, "\\system\\mail\\00001001_S");
_LIT(KMatchAll, "*");
_LIT(KMatchFiles, "*_F");
_LIT(KMatchInWindowSkinBitmaps, "*_iw.png");
_LIT(KMatchFullScreenSkinBitmaps, "*_fs.png");
_LIT(KMatchInWindowSkins, "*_iw.*");
_LIT(KMatchFullScreenSkins, "*_fs.*");


const TInt KSkinSettingIndex = 1;
const TInt KInstallSkinSettingIndex = 5;
const TInt KUninstallSkinSettingIndex = 6;

class CBlueTDisplaySettingItemList : public CAknSettingItemList
{
public:
        CBlueTDisplaySettingItemList(TBlueTSettings& aData);

protected:
        virtual CAknSettingItem* CreateSettingItemL(TInt aSettingId);

private:
        TBlueTSettings& iData;
};

class CBlueTDisplaySettingsContainer : public CCoeControl, MCoeControlObserver
{
public:
   void ConstructL(CBlueTAppUi* aAppUi, CBlueTDisplaySettingsView* aView, const TRect& aRect);
   ~CBlueTDisplaySettingsContainer();

public:
        TKeyResponse OfferKeyEventL(const TKeyEvent& aKeyEvent, TEventCode aType);
        CBlueTDisplaySettingItemList* iListBox;

private:
   void SizeChanged();
   TInt CountComponentControls() const;
   CCoeControl* ComponentControl(TInt aIndex) const;
   void Draw(const TRect& aRect) const;
   void HandleControlEventL(CCoeControl* aControl,TCoeEvent aEventType);

private:
        void GetSkinListL(CAknEnumeratedTextPopupSettingItem* aSettingItem, TBool aAddSelectItem);
        void GetInboxSkinListL(CAknEnumeratedTextPopupSettingItem* aSettingItem);
        void InstallSkin(const TDesC& aBitmapFileName, const TDesC& aDescriptionFileName);

private:
        CDesC16ArrayFlat* iInboxSkinBitmapFileNames;
        CDesC16ArrayFlat* iInboxSkinDescriptionFileNames;
        CEikScrollBarFrame* iScrollBarFrame;
        CBlueTAppUi* iAppUi;
```

```
        CBlueTDisplaySettingsView* iView;
};

#endif
```

BluetDisplaySettingsView.h

```
#ifndef BLUET_DISPLAY_SETTINGS_VIEW_H__
#define BLUET_DISPLAY_SETTINGS_VIEW_H__
#include <aknview.h>
#include "BlueTAppUi.h"

const TUid KDisplaySettingsViewId = {4};

class CBlueTDisplaySettingsContainer;
class CBlueTDisplaySettingsView : public CAknView
{
public:
   void ConstructL();
   ~CBlueTDisplaySettingsView();

public:
   TUid Id() const;
   void HandleCommandL(TInt aCommand);
   void HandleClientRectChange();

public:
        CBlueTDisplaySettingsContainer* SettingsContainer();

private:
   void DoActivateL(const TVwsViewId& aPrevViewId,TUid aCustomMessageId,const TDesC8&
aCustomMessage);
   void DoDeactivate();

private:
        TInt iOrigSkinFileIndex;
        TBlueTSettings::TDisplayType iOrigDisplayType;
        CBlueTDisplaySettingsContainer* iContainer;
};
#endif
```

BlueTDocument.h

```
#ifndef BLUETDOCUMENT_H
#define BLUETDOCUMENT_H
#include <akndoc.h>

class CEikAppUi;

class CBlueTDocument : public CAknDocument
{
public:
   static CBlueTDocument* NewL(CEikApplication& aApp);
   virtual ~CBlueTDocument();

private:
   CBlueTDocument(CEikApplication& aApp);
   void ConstructL();

private:
   CEikAppUi* CreateAppUiL();
};
#endif
```

BlueTListBox.h

```
#ifndef BLUETLISTBOX_H__
#define BLUETLISTBOX_H__

#include <coecntrl.h>
#include <eikenv.h>
#include <eiktxlbx.h>
#include <eiklbi.h>

class CBlueTAppUi;
class CDirTreeNode;

const TInt KItemHeight = 20;

class CBlueTListBoxView : public CListBoxView
{
public:
	CBlueTListBoxView();
	virtual void DrawEmptyList(const TRect &aClientRect) const;
	virtual void Draw(const TRect* aClipRect) const;
};

class CBlueTListItemDrawer : public CTextListItemDrawer
{
public:
	CBlueTListItemDrawer(MTextListBoxModel* aTextListBoxModel, const CFont* aFont);
	virtual void DrawActualItem(TInt aItemIndex, const TRect& aActualItemRect, TBool aItemIsCurrent,
		TBool aViewIsEmphasized, TBool, TBool aItemIsSelected) const;
	void SetColours(const TRgb& aTextForeground, const TRgb& aTextBackground, const TRgb&
			aHighlightForeground, const TRgb& aHighlightBackground);
	TBool ScrollSong(TInt aItemIndex);
	void ResetScrolling();

private:
	TRgb iTextForegroundColour;
	TRgb iTextBackgroundColour;
	TRgb iHighlightForegroundColour;
	TRgb iHighlightBackgroundColour;
	const CFont* iPersistentFont;
	TInt iScrollOffset;
};

class CBlueTListBox : public CEikTextListBox
{
public:
	CBlueTListBox(const CFont* aFont);

protected:
	virtual void CreateItemDrawerL();
	virtual CListBoxView* MakeViewClassInstanceL();

private:
	const CFont* iFont;
};
#endif
```

BluetoothHandler.h

```
#ifndef BLUETOOTHHANDLER_H__
#define BLUETOOTHHANDLER_H__
#include <es_sock.h>
```

```
#include <bt_sock.h>
#include <btextnotifiers.h>
#include <btmanclient.h>
#include <btsdp.h>
#include <eikprogi.h>
#include <AknWaitDialog.h>
#include <AknProgressDialog.h>
#include "DirTreeNode.h"

_LIT(KDownloadedFilesPath, "c:\\nokia\\sounds\\digital");

const TInt KBlueTPort = 11;
const TInt KSerialPortService = 0x1101;
const TInt KNormalCommandTimeout = 20 * 1000000;
const TInt KLongCommandTimeout = 300 * 1000000;

enum TBlueTCommand
	{
	EBlueTNone = 0,
	EBlueTConnect,
	EBlueTList,
	EBlueTCurrentTrackInfo,
	EBlueTPlayFile,
	EBlueTAddFileToPlaylist,
	EBlueTFileInfo,
	EBlueTDownloadFile,
	EBlueTSetVolume,
	EBlueTPlay,
	EBlueTStop,
	EBlueTPause,
	EBlueTFastForward,
	EBlueTRewind,
	EBlueTNextTrack,
	EBlueTPreviousTrack,
	EBlueTShuffle,
	EBlueTRepeat,
	EBlueTShutDown,
	EBlueTFindServer,
	EBlueTGetPlaylist,
	EBlueTSelectInPlaylist,
	EBlueTRemoveFromPlaylist,
	EBlueTRemoveAllFromPlaylist,
	EBlueTDetailedTrackInfo,
	EBlueTFullScreen,
	EBlueTGetVersion,
	EBlueTGetVolume
	};

class TBlueTSettings;

class MBluetoothListener
{
public:
	virtual void ConnectComplete(TInt aError, const TBTDevAddr& aAddress, Tint aPort) = 0;
	virtual void CommandComplete(TBlueTCommand aCommand, TInt aError) = 0;
	virtual void RefreshFileListL(CDirTreeNode* aDirectory) = 0;
	virtual void SetInfo(TBool aPlaying, TBool aPaused, TInt aSongLength, TInt aCurrentTime,
		TBool aShuffle, TBool aRepeat, const TDesC8& aSongNameGuess) = 0;
	virtual void SetDetailedInfo(TInt aSampleRate, TInt aBitRate, TInt aChannels) = 0;
	virtual void ReceivedVolume(TInt aVolume) = 0;
	virtual TBlueTSettings& Settings() = 0;
	virtual void SetPlaylistIndex(TInt aIndex) = 0;
	virtual RPointerArray<HBufC8>& Playlist() = 0;
	virtual CCoeEnv* CoeEnv() = 0;
};
```

```
class CBluetoothHandler : public CActive, public MSdpAgentNotifier, public   MSdpAttributeValueVisitor,
public                  MProgressDialogCallback
{
public:
        enum TBluetoothState
        {
                EIdle,
                EReadingSongListFile,
                ESelectingServer,
                EInquiringService,
                EConnecting,
                EReading,
                EWriting
        };

        static CBluetoothHandler* NewL(MBluetoothListener* aListener, const TBTDevAddr&
                        aAutoConnectAddress, TInt aAutoConnectPort);
        ~CBluetoothHandler();

        void ReadSongListFile(const TDesC& aFileName);
        void Connect();

public:
        virtual void AttributeRequestComplete(TSdpServRecordHandle aHandle, Tint aError);
        virtual void AttributeRequestResult(TSdpServRecordHandle aHandle, TSdpAttributeID aAttrID,
                CSdpAttrValue* aAttrValue);
        virtual void NextRecordRequestComplete(TInt aError, TSdpServRecordHandle aHandle,
                TInt aTotalRecordsCount);

public:
   virtual void VisitAttributeValueL(CSdpAttrValue &aValue, TSdpElementType aType);
   virtual void StartListL(CSdpAttrValueList &aList);
   virtual void EndListL();

public:
   virtual void DialogDismissedL(TInt aButtonId);

public:
        void SortDirTree(CDirTreeNode* aDirectory);
        void RequestList(TBool aGetEntireTree);
        void RequestDirectoryList(CDirTreeNode* aNode);
        void RequestInfo();
        void RequestPlayFile(const TDesC8& aFileName);
        void RequestAddFileToPlaylist(const TDesC8& aFileName);
        void RequestFileInfo(const TDesC8& aFileName);
        void RequestDownloadFile(const TDesC8& aFileName);
        void RequestSetVolume(TUint8 aVolume);
        void RequestPlay();
        void RequestStop();
        void RequestPause();
        void RequestNextTrack();
        void RequestPreviousTrack();
        void RequestFastForward();
        void RequestRewind();
        void RequestShuffle(TBool aShuffleOn);
        void RequestRepeat(TBool aRepeatOn);
        void RequestShutDown();
        void RequestFindServer();
        void RequestGetPlaylist();
        void RequestSelectInPlaylist(TInt aIndex);
        void RequestRemoveAllFromPlaylist();
        void RequestDetailedTrackInfo();
        void RequestFullScreen();

        TPtrC8 ReadListFileName(const TDesC8& aPacket, TInt& aStartPos);
        static TInt CompareNodes(const CDirTreeNode& aNode1, const
        CDirTreeNode& aNode2);
```

```
private:
        void RunL();
        void DoCancel();
        TInt RunError(TInt aError);

private:
        CBluetoothHandler(MBluetoothListener* aListener, const TBTDevAddr& aAutoConnectAddress,
                TInt aAutoConnectPort);
        void ConstructL();
        void CloseAllResources();
        void ParseCommandResponse();
        TBool CommandExpectsResponse(TBlueTCommand aCommand);
        void WriteCommand();
        void HandleReadFileCompleteL();
        void HandleSelectServerCompleteL();
        void HandleConnectionCompleteL();
        void HandleReadCompleteL();
        void HandleWriteCompleteL();
        void ParseListResponse();
        void ParseCurrentTrackInfoResponse();
        void ParseFileInfoResponse();
        void ParseDownloadFileResponse();
        void ParseFindServerResponse();
        void ParseGetPlaylistResponseL();
        void ParseDetailedTrackInfoResponse();
        void ParseGetVersionResponse();
        void ParseGetVolumeResponse();

        TInt ParseListPacketL(TDes8& aPacket);
        TInt ParsePlaylistPacketL(TDes8& aPacket);
        void ConnectionFailed(TInt aError);
        static TInt CommandTimerCallBack(TAny* aBluetoothHandler);
        void CommandTimedOut();
        void FindServerNextChannel();
        void ShowWaitDialogL(TInt aResourceId);

private:
        RSocketServ iSocketServ;
        RSocket iSocket;
        TBluetoothState iState;
        RNotifier iNotifier;

        TBTDeviceResponseParamsPckg iDeviceSelectionPckg;
        TBTDeviceResponseParamsPckg iDeviceFoundPckg;
        TSockXfrLength iXfrLength;
        TBuf8<1024> iPacketBuf;
        TBuf8<2048> iTotalPacketBuf;

        CSdpAgent* iSdpAgent;
        CSdpSearchPattern* iSdpSearchPattern;
        TInt iSdpCurrentRecordCount;
        TInt iSdpTotalRecordCount;

        CAknWaitDialog* iWaitDialog;
        CAknProgressDialog* iProgressDialog;

        CEikProgressInfo* iProgressInfo;

        TUUID iServiceClass;
        TUint iRemotePort;

        TBool iConnected;

        enum TVersionState
                {
                ENotGotVersion,
```

```
                EGettingVersion,
                EGotVersion,
                EGettingVolume,
                EGotVolume
                };

        TVersionState iVersionState;
        TInt iServerMajorVersion;
        TInt iServerMinorVersion;
        TBuf8<1024> iVersionQueuedPacketBuf;

        TBlueTCommand iCurrentCommand;
        TBlueTCommand iQueuedCommand;
        TBool iWaitingForResponseData;

        RFile iSongListFile;
        RFile iDownloadFile;
        TFileName iDownloadFileName;
        TInt iDownloadFileLength;
        TInt iDownloadFileReceived;

        MBluetoothListener* iListener;

        CDirTreeNode* iDirTreeRoot;
        CDirTreeNode* iDirTreeCurrentNode;
        CDirTreeNode* iDirTreeNodeToExpand;

        TBTDevAddr iAutoConnectAddress;
        TInt iAutoConnectPort;
        TInt iFindServerPort;
        CPeriodic* iCommandTimer;
        TInt iPlaylistPos;
        TNameEntry iNameEntry;
        RHostResolver iResolver;
};
#endif
```

BlueTPlaylistView.h

```
#ifndef BLUETPLAYLISTVIEW_H__
#define BLUETPLAYLISTVIEW_H__
#include <aknview.h>
#include "BlueTListBox.h"

const TUid KPlaylistViewId = {3};

class CBlueTPlaylistView : public CAknView
{
public:
   void ConstructL();
   ~CBlueTPlaylistView();

public:
   TUid Id() const;
   void HandleCommandL(TInt aCommand);
   void HandleClientRectChange();

public:
        void ReadSkinDescriptionL(const TDesC& aFileName);
        TInt GetSelectedFile();
        TBool IsFocused();
        void RefreshPlaylistL(const RPointerArray<HBufC8>& aPlaylist, TInt aIndex);
        void ScrollSelectedSong();
        TKeyResponse OfferKeyEventL(const TKeyEvent& aKeyEvent, TEventCode aType);
        void FindSongL(TBool aFindNext);
```

```
private:
   void DoActivateL(const TVwsViewId& aPrevViewId,TUid aCustomMessageId,const TDesC8&
aCustomMessage);
   void DoDeactivate();

private:
	void ReadSkinColour(TLex8& aLex, TRgb& aColour);

private:
	CBlueTListBox* iListBox;
	CEikScrollBarFrame* iScrollBarFrame;
	CDesC16ArrayFlat* iListItems;

	TRgb iTextForegroundColour;
	TRgb iTextBackgroundColour;
	TRgb iHighlightForegroundColour;
	TRgb iHighlightBackgroundColour;

	TBool iFocused;

	TFileName iLastSearchedSong;
	TInt iResultsFound;
	TInt iFindResultsToSkip;
};
#endif
```

BlueTSkinHandler.h

```
#ifndef BLUETSKINHANDLER_H__
#define BLUETSKINHANDLER_H__
#include <MdaImageConverter.h>

const TInt KMaxSkinBitmapWidth = 176;
const TInt KMaxSkinBitmapHeight = 300;

enum TBlueTFont
	{
	EBlueTNoFont = 0,
	EBlueTTitleFont = 1,
	EBlueTDenseFont = 2,
	EBlueTLegendFont = 3,
	EBlueTAnnotationFont = 4,
	EBlueTSymbolFont = 5,
	EBlueTNormalFont = 6
	};

class TBlueTSkin
{
public:
	TRect iButtonsRect;
	TRect iTrackTimeRect;
	TRect iTrackNameRect;
	TRect iVolumeRect;
	TSize iVolumeSliderSize;
	TRect iShuffleRect;
	TRect iRepeatRect;
	TRgb iTrackNameColour;
	TRgb iTrackTimeColour;
	TBlueTFont iTrackNameFont;
	TBlueTFont iTrackTimeFont;
	CFbsBitmap* iBitmap;
	TBool iValid;
};
```

```
class CBlueTAppUi;

class CBlueTSkinHandler : public CBase, public MMdaImageUtilObserver
{
public:
	CBlueTSkinHandler(CBlueTAppUi* aAppUi, TBlueTSkin& aSkin);
	static CBlueTSkinHandler* NewL(CBlueTAppUi* aAppUi, TBlueTSkin& aSkin);
	~CBlueTSkinHandler();

public:
	virtual void MiuoConvertComplete(TInt aError);
	virtual void MiuoCreateComplete(TInt aError);
	virtual void MiuoOpenComplete(TInt aError);

public:
	void LoadSkinL();
	static const CFont* BlueTFontToCFont(TBlueTFont aBlueTFont);

protected:
	void ConstructL();

private:
	void ReadSkinDescriptionL(const TDesC& aFileName);
	void ReadSkinColour(TLex8& aLex, TRgb& aColour);
	void ReadSkinRect(TLex8& aLex, TRect& aRect);

private:
	CMdaImageFileToBitmapUtility* iSkinLoader;
	CMdaBitmapScaler* iSkinScaler;
	CBlueTAppUi* iAppUi;
	TBlueTSkin& iSkin;
	TBool iScaling;
};
#endif
```

DirTreeNode.h

```
#ifndef DIRTREENODE_H__
#define DIRTREENODE_H__
#include <e32base.h>

const TUint8 KNodeTypeRoot = 0x0;
const TUint8 KNodeTypeChild = 0x1;
const TUint8 KNodeTypeOnlyChild = 0x2;
const TUint8 KNodeTypeSibling = 0x3;
const TUint8 KNodeTypeLastSibling = 0x4;
const TUint8 KNodeTypeLastNode = 0xF;
const TUint8 KFileTypeDrive = 0x0;
const TUint8 KFileTypeDir = 0x1;
const TUint8 KFileTypeFile = 0x2;
const TUint8 KFileTypeUnexpandedDir = 0x3;

class CDirTreeNode : public CBase
{
public:
	static CDirTreeNode* NewL(TUint8 aNodeType, TUint8 aFileType, const TDesC8& aFileName);
	~CDirTreeNode();

	TUint8 iNodeType;
	TUint8 iFileType;
	HBufC8* iFileName;

	CDirTreeNode* iParent;
	CDirTreeNode* iFirstChild;
	CDirTreeNode* iNextSibling;
```

```
private:
        CDirTreeNode(TUint8 aNodeType, TUint8 aFileType);
        void ConstructL(const TDesC8& aFileName);
};
#endif
```

IncomingCallListener.h

```
#ifndef INCOMING_CALL_LISTENER_H__
#define INCOMING_CALL_LISTENER_H__
#include <e32base.h>
#include <etel.h>

class MIncomingCallListener
{
public:
        virtual void IncomingCall(const TDesC& aName) = 0;
};

class CIncomingCallListener : public CActive
{
public:
        static CIncomingCallListener* NewL(MIncomingCallListener& aListener);
        ~CIncomingCallListener();
        void StartListening();
        void RunL();
        TInt RunError(TInt aError);
        void DoCancel();

private:
        CIncomingCallListener(MIncomingCallListener& aListener);
        void ConstructL();
        RTelServer iTelServer;
        RPhone iPhone;
        RLine iLine;
        TName iCallName;
        MIncomingCallListener& iListener;
};
#endif
```