HH Snap-In API

Designed by
TIMEWAVE TECHNOLOGY INC.
1025 Selby Ave, Suite 101, St Paul, MN 55104-6533
Phone: (651) 489-5080 ● Email: sales@timewave.com

February, 2006

Description

The HH Snap-In API is a standardized interface to provide the easy adding of soundcard communications. It provides a standardized method to create a Windows Win32 DLL that may be used dynamically with other Snap-In modules.

Provided is sample code for a module called HH_DUMB. This uses an external COM library to create a dumb terminal interface. It is not a true soundcard mode driver in that it does not use the actual audio data. For an example that demonstrates an actual soundcard mode – see HH_MT63.

Files

There are several standard files that are defined. This is a summary of what you will find in each of the files that make up your HH_DUMB DLL.

HH_DUMB.vcproj	This is the main project file for VC++ projects. It contains information about the version of Visual C++ that generated the file, and information about the platforms, configurations, and project features selected with the Application Wizard.
HH_DUMB.cpp	This is the main DLL source file that contains the definition of DllMain() and the exported Snap-In functions.
HH_DUMB.rc	This is a listing of all of the Microsoft Windows resources that the program uses. It includes the icons, bitmaps, and cursors that are stored in the RES subdirectory. This file can be directly edited in Microsoft Visual C++.
res\HH_DUMB.rc2	This file contains resources that are not edited by Microsoft Visual C++. You should place all resources not editable by the resource editor in this file.
HH_DUMB.def	This file contains information about the DLL that must be provided to run with Microsoft Windows. It defines parameters such as the name and description of the DLL. It also exports functions from the DLL.

StdAfx.h, StdAfx.cpp	These files are used to build a precompiled header (PCH) file named HH_DUMB.pch and a precompiled types file named StdAfx.obj.
Resource.h	This is the standard header file, which defines new resource IDs. Microsoft Visual C++ reads and updates this file.

Functions

There are a standard set of functions to be defined. These also need to be exported in the DEF file. They are prototyped in the header <HHSnapIn.h>.

```
// DLL entry point
extern "C" int APIENTRY
DllMain(HINSTANCE hInstance, DWORD dwReason, LPVOID lpReserved)
```

This is the DLL entry point. It always needs to be defined. If you are creating a new project, create an MFC Extension DLL.

In the DLL_PROCESS_ATTACH, do any construction code that needs to be done.
In the DLL_PROCESS_DETACH, do any destructors/cleanup that needs to be done.

```
// Get the setup information for the Snap-In.  This is used to generate
the list of Snap-Ins
extern "C"
BOOL WINAPI GetHHSetup(SNAPINSETUP *pSetup)
```

This is the discovery callback for the DLL. In this the specifics for the DLL need to be filled in. The predefined header SNAPINSETUP is passed.

```
typedef struct _tagSnapinSetup
{
      int          m_cbSize;
      WCHAR        m_wszName[256];
      HHMODE       m_dwMode;
      BOOL         m_bSelfDriven;
      BOOL         m_bDisableXMIT;
      BOOL         m_bDisableRECV;
} SNAPINSETUP;
```

m_cbSize	sizeof(SNAPINSETUP)
m_wszName[256]	The name of the DLL. This is always in Unicode.

m_dwMode	This is the DLL mode. This can be one of the preset values or a user defined value. These values are defined as: HH_PSK, HH_MT63, HH_MFSK, HH_THROB, HH_CW, HH_RTTY, HH_ASCII, HH_FEC_AMTOR, HH_AMTOR, HH_FEC_PACTOR, HH_PACTOR, HH_FEC_GTOR, HH_GTOR, HH_PACTOR2, HH_PACTOR3, HH_CLOVER, HH_NAVREX, HH_SSTV, HH_DUMB, HH_AFSK.
m_bSelfDriven	When set to TRUE, the soundcard data will not be used.
m_bDisableXMIT	When set to TRUE, the Encode functions will not be called.
m_bDisableRECV	When set to TRUE, the Decode functions will not be used.

```
// This is called when the DLL is started.
extern "C"
BOOL WINAPI InitHHMode(double fFrequency)
```

Called when the DLL is activated. The soundcard frequency is passed. 8000.0 Hz is always passed to this function.

```
// Return the set frequency
extern "C"
double WINAPI GetHHInternalFrequency(void)
```

Return the frequency passed in InitHHMode().

```
// Encode the data
extern "C"
int WINAPI EncodeHHDataA(short *fData, int cbLen, LPSTR lpStr, int
cbStr)
```

This is the actual encoding function for transmit. Two buffers are passed to the function, one for the audio data (unless m_bSelfDriven is set), and one with the text to encode.

```
// Decode the data
extern "C"
int WINAPI DecodeHHDataA(short *fData, int cbLen, LPSTR lpStr, int
cbStr)
```

This is the actual decoding function for receive. Two buffers are passed to the function, one with the audio data (unless m_bSelfDriven is set), and one for the decoded text.

```
// Generate the Twiddle/Diddle
extern "C"
int WINAPI GetHHTwiddle(short *psData, int cbLen)
```

When idle and m_bSelfDriven is not set, this should return the diddle.

```
// Pop up a setup dialog.
extern "C"
UINT WINAPI InvokeHHSetup(HWND hWndParent)
```

This should create a pop-up dialog and prompt for setting changes. If this is an MFC Extension DLL, this may be cast:

```
afxCurrentResourceHandle = HH_DUMBDLL.hResource;
afxCurrentInstanceHandle = HH_DUMBDLL.hModule;

CWnd *pWndParent = CWnd::FromHandlePermanent(hWndParent));
```

```
// Create a toolbar
extern "C"
HWND WINAPI GetHHToolbar(HWND hWndParent, DWORD dwTBStyle, UINT wID)
```

This should return a HWND from a CreateToolbar(). This will be used by the calling application to display runtime toolbars.

```
// Commands from the returned toolbar
extern "C"
BOOL WINAPI DoHHCommand(UINT command)
```

This will invoke a command. Return TRUE if the command is invoked, FALSE if the command is not found.

```
// Implements a user function
extern "C"
UINT WINAPI InvokeHHFuncx(HWND hWndParent, WPARAM wParam, LPARAM lParam)
```

This will invoke a function. *x* is a number.

```
// Load data
extern "C"
BOOL WINAPI LoadHHUserData(LPVOID lpData, UINT cbData)
// Save data
extern "C"
BOOL WINAPI SaveHHUserData(LPVOID lpData, UINT cbData)
```

These are for saving and loading data on a per snap-in basis.