SDK - HTDisplayDll.dll Manual
VC++6.0 IDE

Updated: 18-OCT-2016

Note:

HTHardDll.dll was compiled under VC++6.0.
WORD: unsigned short
BOOL: bool
ULONG: unsigned long

The following ifdef block is the standard way of creating macros which make exporting from a DLL simpler. All files within this DLL are compiled with the DLL_API symbol defined on the command line. this symbol should not be defined on any project that uses this DLL. This way any other project whose source files include this file see DLL_API functions as being imported from a DLL, wheras this DLL sees symbols defined with this macro as being exported.

```
#ifndef DLL_API
#define DLL_API extern "C" __declspec(dllimport)
#endif
```

Define __stdcall:
#defineWIN_API __stdcallHTDispalyDll SDK

1. DLL_API void WINAPI HTDrawGrid(

HDC **hDC**, //handle of drawing context
int **nLeft**, // the left of the drawing area
int **nTop**, // the top of the drawing area
int **nRight**, // the right of the drawing area
int **nBottom**, // the bottom of the drawing area
USHORT **nHoriGridNum**, // the number of horizontal grid spaces
USHORT **nVertGridNum**, // the number of vertical grid spaces
USHORT **nBright**, // brightness
USHORT **IsGrid** // whether to draw grid scale
);

Draws a white (or gray) grid on black background.

nLeft, **nTop**, **nRight**, and **nBottom** specify the size of the grid. These values can be obtained from GetClientRect() if the grid is to fill the entire window.
nHoriGridNum specifies the number of horizontal divisions in the display. For example, if nHoriGridNum=10, there will be 9 vertical lines drawn on the grid – 4 to the left of the bright center line, and 4 to the right, resulting in 10 divisions.
nVertGridNum is analogous to nHoriGridNum in the vertical direction.
nBright specifies the brightness of the grid. 0 is invisible, 255 is maximum. If a number greater than 255 is supplied, it is taken as 255.
IsGrid is really a boolean. If IsGrid=0, only the bright center lines are drawn in the horizontal and vertical direction, i.e., nHoriGridNum and nVertGridNum have no effect. If IsGrid=1, the subdivisions are drawn.

2. DLL_API void WINAPI HTDrawWaveInYT(

HDC **hDC**, // handle of the drawing context
RECT **Rect**, // the rect for drawing
COLORREF **clrRGB**, // the color of the line
USHORT **nDisType**, // display type : Line or Dot
short* **pSrcData**, // the source data for drawing
ULONG **nSrcDataLen**, //the source data length
ULONG **nDisDataLen**, // the display length data for drawing
ULONG **nCenterData**, // half of the source data
USHORT **nDisLeverPos**, //the display position (Zero Level)
double **dbHorizontal**, // the horizontal factor of zoom out/in
double **dbVertical**, // the vertical factor of zoom out/in
USHORT **nYTFormat**, //format: normal or scan
ULONG **nScanLen** // the scan data length, only invalidate in scan mode
);

Draw the waveform for one channel of data. This function is for Visual C++. For Visual Basic, use HTDrawWaveInYTVB() below.

Rect specifies the drawing area. For the whole window, get it from GetClientRect().
clrRGB is a Windows color. Use the RGB() macro in Visual C++ to generate it For example, yellow is RGB(255, 255, 0).
nDisType 0=lines; 1=dots. Each sample is displayed as a single dot. If nDistype is set to 0, these dots are connected with lines, usually resulting in a smoother appearance.
pSrcData is the address of an array containing the data to display. Usually, this will be the pCH1Data or pCH2Data read by dsoReadHardData(). It is an array of type short with data values ranging from 0 to 255.
nSrcDataLen is the length of valid data in the above array (not necessarily equal to the array size). Use the value of nReadLen that was specified with dsoReadHardData(). This is needed so that the display can be clipped if it goes outside the bounds of the data buffer.
nDisDataLen and **nCenterData** specify the portion of the data to display. nCenterData denotes the midpoint of the data sequence to be displayed. To display the entire buffer, set nDisDataLen equal to nSrcDataLen, and nCenterData to half of that value. To display the data from sample 150 to 200, set nDisDataLen to 50, and nCenterData to 175. nCenterData is an unsigned long, so cannot be a negative number, even if that would theoretically make some data displayable. Note that the Hantek function does not bounds-check the value of nCenterData, so the programmer *must* do it, first. Using a value of nCenterData that is out of the bounds of the data buffer possibly could cause a runtime error when the Hantek function tries to do an unreasonably large heap allocation. This is likely due to a negative number being passed as an allocation size, and taken as a huge positive number.
nDisLeverPos is a vertical offset for the display. Probably should have been named nDisLevelPos, but the typo persists. If nDisLeverPos is set to 255, then a data value of 0 will correspond to the bottom of the display, and a data value of 255 will correspond to the top of the display. Setting a lower value for nDisLeverPos moves the displayed trace up, and setting a higher value moves the trace down.
dbHorizontal is a horizontal zoom factor. dbHorizontal of 2.0 causes the display of twice as many samples (subject to valid data limits), while a factor of 0.5 displays half as many samples. Another way of thinking of this is that nDisDataLen is multiplied by dbHorizontal before the actual display is done. nCenterData is unaffected by the horizontal zoom operation, so a zoom in/out is horizontally about the center point at nCenterData.
dbVertical is a vertical zoom factor, but operates without taking into account the zero reference level. A way of thinking about this is that each data sample is divided by dbVertical before display. If unsigned data is to be displayed (such as from Normal or Single Trigger mode), to compesate for the shifting effect on the vertical display, nDisLeverPos should be set to the a value of n+128/dbVertical, where n is desired zero level on the display. If signed data is to be displayed (such as from Auto Trigger mode), then nDisLeverPos can simply be set to n.
nYTFormat I don't understand. 0=normal, 1=scan. Possibly, the scan mode is used to simulate an actual analog oscilloscope by rendering only part of the trace, so that the user can see the beam going across the screen at low frequencies?

nScanLen I also don't understand. Appears to be ignored unless nYTFormat is 1. Perhaps it is the index of the last valid sample in the input data?

3. DLL_API void WINAPI HTDrawWaveInYTVB(

```
HDC hDC,
int left,
int top,
int right,
int bottom,
USHORT R,
USHORT G,
USHORT B,
USHORT nDisType,
short* pSrcData,
ULONG nSrcDataLen,
ULONG nDisDataLen,
ULONG nCenterData,
USHORT nDisLeverPos,
double dbHorizontal,
double dbVertical,
USHORT nYTFormat,
ULONG nScanLen
);
```

This is supposed to be the same as HTDrawWaveInYT(), except with the color reference in Visual Basic style. I didn't actually use this function, so don't really know.