Application Programming Interface (API) for CPS USB-Compatible High-Voltage Probes

Introduction

CPS provides a free software development kit (SDK) for CPS USB-compatible high voltage probes. The SDK includes an API that allows application software to communicate with probes. The API is compatible with all CPS probe models and supports up to sixteen probes connected to a single computer. This document describes API functions and operation, and is intended for software developers who create probe application programs.

Different versions of the SDK are available for Linux and Windows. Although the files may differ, API functions and behavior are identical in both SDK versions.

Windows SDK

The Windows SDK includes the following files:

- `HVProbe.dll` – Windows API for all USB compatible CPS probes. This must reside in the same directory as your application program (or in its DLL search path). Please note that this is a 32-bit library, which means it can be used on both 32- and 64-bit machines, but only with 32-bit applications.
- `HVProbe.lib` – Library file for the API. This is used when building your application, to link the application to the DLL.
- `HVProbe.h` – C/C++ header. This must be included in all C/C++ source files that call API functions.
- `HVProbe.vb` – VB.NET module. This must be included in all VB.NET projects that call API functions.
- Source code, project files, and pre-built executables of simple C and VB.NET applications that show how to use the API.

Linux SDK

The Linux SDK is contained in `sdk_hvprobe.tar.bz2`. To extract the SDK files, open a terminal window and execute this command:

```
tar tjf sdk_hvprobe.tar.bz2
```

Refer to the extracted `README` file for further information.

Error Codes

All API functions return an error code. `HVP_ERR_OK` (zero) is returned if no errors are detected; all other error codes are negative values. See `HVProbe.h` or `HVProbe.vb` for a complete list of error codes.

Probe Identification

Most of the API functions require a ProbeID argument. This is a value between 0 and 15 that designates a particular probe. Not all values are legal; the maximum legal value is the number of detected probes minus one. For example, if two probes are detected, ProbeID may be either 0 or 1. If only one probe is detected, its ProbeID is 0.

An LED is located near the probe's USB connector, which flashes red when the computer is not communicating with the probe. This LED will momentarily turn blue when the probe communicates over USB. When multiple probes are connected to a computer, this feature can be used to determine the ProbeID associated with each probe.

Functions

HVP_OpenSystem

```
int HVP_OpenSystem(void);
```

This must be the first API function called by the application program. It enumerates (or reenumerates) the probes and enables access to all probes. An error code (negative value) is returned if an error was encountered, otherwise the function returns the number of detected probes (0 to 16).

Example

```
int nprobes = HVP_OpenSystem();
if (nprobes < 0)
  printf("ERROR! Problem opening probe API, error code = %d.\n", nprobes);
else
  printf("Detected %d probe(s).\n", nprobes);
```

HVP_CloseSystem

```
int HVP_CloseSystem(void);
```

This must be the last API function called by the application program. It closes all open probes and frees all API resources. The return value is meaningless and should be ignored.

HVP_OpenProbe

```
int HVP_OpenProbe(int ProbeID);
```

This function must be called once for each probe; it opens a probe to allow subsequent communication with it. No communications are possible with a probe until it has been opened by this function.

HVP_CloseProbe

```
int HVP_CloseProbe(int ProbeID);
```

This function closes a probe. After calling this, further communication with the probe is prohibited unless the probe is first reopened by calling HVP_OpenProbe() again. The application may call HVP_CloseSystem() to close all open probes; in this case it is not necessary to call HVP_CloseProbe() for each open probe.

HVP_GetAttributes

```
int HVP_GetAttributes(int ProbeID, PROBE_INFO *info);
```

This function reads the attributes of an open probe. The application must allocate a PROBE_INFO structure to receive the attributes.

Example

```
PROBE_INFO info;
int err = HVP_GetAttributes(ProbeID, &info);
if (err == HVP_ERR_OK) {
  printf("Model %d\n", info.model);
  printf("Serial number %d\n", info.serialnum);
  printf("Firmware %s\n", info.firmware_info);
}
```

HVP_GetStatus

```
int HVP_GetStatus(int ProbeID, int *status);
```

This function reads status information from an open probe. The application must allocate an int (status) to receive the information, which is a collection of flag bits. See the header file for a list of flags.

Example

```
int status;
int err = HVP_GetStatus(0, &status);
if (err == HVP_ERR_OK)
  if (status & HVP_STATUS_ALARM)
    printf("WARNING! Probe is near HV and ground lead is not connected.\n");
```

HVP_GetMeters

```
int HVP_GetMeters(int ProbeID, double buf[NUM_METERS]);
```

This function reads all meter data from an open probe. The application must allocate an array (buf) to receive the data. The array index of each meter is specified in METER_ID (in header file).

Example

```
double buf[NUM_METERS];
int err = HVP_GetMeters(0, buf);
if (err == HVP_ERR_OK) {
  printf("DC meter    = %f VDC\n",       buf[METER_DC] );
  printf("AC meter    = %f VAC RMS\n",   buf[METER_AC] );
  printf("DC min peak = %f VDC\n",       buf[METER_MIN]);
  printf("DC max peak = %f VDC\n",       buf[METER_MAX]);
  printf("Probe temp  = %f degrees C\n", buf[METER_T]  );
}
```

HVP_ErrorString

```
char *HVP_ErrorString(int errcode);
```

This function maps an API error code to an associated descriptive string.

Example

```
double buf[NUM_METERS];
int err = HVP_GetMeters(0, buf);
if (err != HVP_ERR_OK)
  printf("HVP_GetMeters() error: %s\n", HVP_ErrorString(err));
```

HVP_ReadApiVersion

```
int HVP_ReadApiVersion(void);
```

This function returns the API version number.

Example

```
int ver = HVP_ReadApiVersion();
printf("API version: %d.%d.%d\n", ver >> 24, (ver >> 16) & 0xFF, ver & 0xFFFF);
```