API user guide
for CPS 3600 family

Introduction

This document describes the application programming interface (API) for the CPS 3600 family of power supplies. The API is compatible with all products in the 3600 family. It allows a single computer to monitor and control up to 16 systems of identical or mixed product types.

Distribution

The API consists of the following files, which are included in the Windows software development kit (SDK) for the CPS 3600 family, available at cpshv.com:

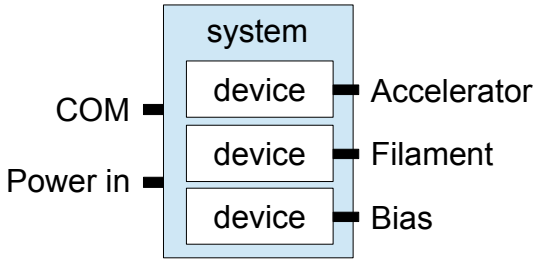
- `c3600.dll` – API executable. This must reside in the same directory as the application program (or in the DLL search path). Note that this is a 32-bit library, which means it can be used on both 32- and 64-bit machines, but only with 32-bit applications.
- `c3600.lib` – Library file for the API. This is used when building an application, to link the application to the DLL.
- `c3600.h` – C/C++ header. Include this in C/C++ source files that call API functions.
- `c3600.vb` – VB.NET module. Include this in VB.NET projects that call API functions.

Terminology

The API views each product in the 3600 family as a system of devices in which:

- A *system* is a collection of one or more devices in a common enclosure, with system-level connectors for input power and communication.
- A *device* is an embedded single-output power supply with associated setpoint and meters.

For example, the system shown to the right has three devices: a high-voltage accelerator supply, a filament current source, and a bias voltage generator.



Hardware addresses

Every device in a system, and the system itself, is an addressable resource. API functions use these arguments to address a particular resource:

- `sysid` – system ID in range 0 to 15.
- `devtype` – enumerated device type (e.g., `DEVTYPE_ACCELERATOR`).
- `devindex` – device index.

A system is addressed by its `sysid` alone, whereas a device is addressed by its `sysid`, `devtype`, and `devindex`. These arguments are explained in detail below.

System ID (sysid)

The API allows a computer to communicate with up to 16 systems. To facilitate this, a unique system ID in the range 0 to 15 is assigned to each system. The application software assigns an ID to a system by calling `C3600_OpenSystem()`.

Device type (devtype)

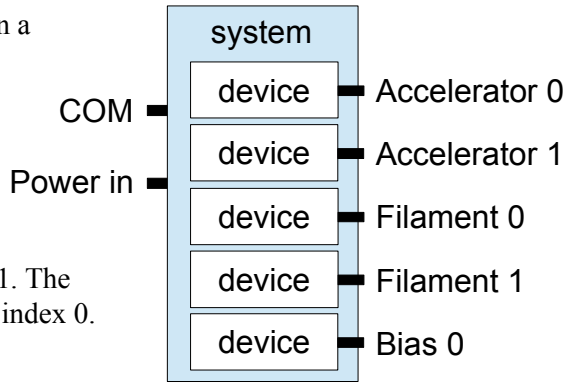
Every device has a factory-assigned, enumerated type which is based on its output characteristics and intended purpose:

```
typedef enum DEVTYPE {           // DEVICE TYPES ---
    DEVTYPE_UNIMPLEMENTED = 0,   //  No device available
    DEVTYPE_UNKNOWN       = 1,   //  Invalid device type
    DEVTYPE_ACCELERATOR   = 2,   //  Accelerator
    DEVTYPE_BIAS          = 3,   //  Bias
    DEVTYPE_EXTRACTOR     = 4,   //  Extractor
    DEVTYPE_FILAMENT      = 5,   //  Filament
    DEVTYPE_LENS          = 6,   //  Lens
    DEVTYPE_SUPPRESSOR    = 7    //  Suppressor
} DEVTYPE;
```

Device index (devindex)

The API automatically assigns an index to each device based on its device type. Application software uses the device index to address a particular device.

In general, for any given device type T, the first type T device in a system is assigned index 0. If a system has multiple type T devices, those devices are assigned sequential index numbers starting at 0.



For example, the system shown to the right has two devices of type `DEVTYPE_ACCELERATOR`, which are assigned index 0 and 1. Similarly, the two filament supplies (type `DEVTYPE_FILAMENT`) are assigned index 0 and 1. The system has a single `DEVTYPE_BIAS` device, which is assigned index 0.

Communication timeouts (maxwait)

Many API functions send commands to a system and expect to receive replies. If, due to communication error or equipment failure, an expected reply fails to arrive, the application program could hang. To prevent this, the API functions provides a `maxwait` argument.

`maxwait` specifies the maximum time the application is willing to wait for a system reply. If a reply is not received within `maxwait` milliseconds, the function will terminate and return `C3600_ERR_READ_TIMEOUT`.

The code examples in this document set `maxwait` to 1000 ms, which in most cases will be long enough to guarantee no timeouts during normal operation.

Error codes

Most of the API functions return an error code. Error code `C3600_ERR_OK` (zero) is returned if no errors are detected; all other error codes are negative values. See `c3600.h` or `c3600.vb` for a complete list of error codes.

Thread safety

Except where otherwise noted, all API functions are thread safe.

API functions

C3600_GetErrorString()

Prototype

```
const char * C3600_GetErrorString(int errcode);
```

Arguments

`errcode` – any API error code.

Return value

Pointer to a plaintext string that describes `errcode`.

Description

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

Example

```
int errcode = C3600_ApiClose();
if (errcode != C3600_ERR_OK)
  printf("C3600_ApiClose() error: %s\n", C3600_GetErrorString(errcode));
```

C3600_ApiOpen()

Prototype

```
int C3600_ApiOpen(int *version);
```

Arguments

`version` – buffer that will receive API version number.

Return value

API error code.

Description

This must be the first API function called by the application program. It allocates and initializes API resources and copies the API version number to `version`. This function is not thread safe.

Example

```
inv ver, errcode = C3600_ApiOpen(&version);
if (errcode != C3600_ERR_OK)
  printf("ERROR! Problem opening 3600 API, errcode = %d.\n", errcode);
else
  printf("API version = %d.%d.%d\n", ver >> 24, (ver >> 16) & 0xFF, ver & 0xFFFF);
```

C3600_ApiClose()

Prototype

```
int C3600_ApiClose(void);
```

Arguments

none

Return value

API error code.

Description

This must be the last API function called by the application program. It closes the API and frees all API

resources. This function is not thread safe.

C3600_OpenSystem()

Prototype

```
int C3600_OpenSystem(int sysid, int port, int maxwait);
```

Arguments

`sysid` – system ID in range 0 to 15.

`port` – serial port index.

`maxwait` – maximum milliseconds to wait for system to reply.

Return value

API error code.

Description

This function must be called once for each system. It gathers information about the system, enumerates devices, and enables subsequent communication with the system. This function is not thread safe.

`port` specifies the index of the serial port (on the host computer) used to communicate with the system. USB compatible systems use virtual serial ports, whereas other systems use RS-232 serial ports. In either case, the index of a serial port is the COM number minus one. For example, when using COM1, set `port` = 0.

Some USB systems require multiple virtual serial ports. In such cases, the computer must use a contiguous sequence of COM numbers for each system, with `port` indicating the lowest serial port index in the sequence. For example, in the case of a four-port system that uses COM20 to COM23, set `port` = 19 (port index of COM20).

C3600_CloseSystem()

Prototype

```
int C3600_CloseSystem(int sysid);
```

Arguments

`sysid` – system ID in range 0 to 15.

Return value

API error code.

Description

This function closes a system. After calling this, further communication with the system (and its devices) is prohibited. This function is not thread safe.

C3600_GetSysInfo()

Prototype

```
int C3600_GetSysInfo(int sysid, SYS_INFO *info, int maxwait);
```

Arguments

`sysid` – system ID in range 0 to 15.

`info` – buffer that will receive system info.

`maxwait` – maximum milliseconds to wait for system reply.

Return value

API error code.

Description

This function transfers system attributes to the `info` buffer.

Example

```
SYS_INFO info;
int errcode = C3600_GetSysInfo(0, &info, 1000);
if (errcode == C3600_ERR_OK)
  printf("CPS model number = %s\n", info.model);
```

C3600_GetDevInfo()

Prototype

```
int C3600_GetDevInfo(int sysid, DEVTYPE devtype, int devindex, DEV_INFO *info, int maxwait);
```

Arguments

`sysid` – system ID in range 0 to 15.

`devtype` – enumerated device type.

`devindex` – device index.

`info` – buffer that will receive device info.

`maxwait` – maximum milliseconds to wait for system reply.

Return value

API error code.

Description

This function copies a device's attributes to the `info` buffer. See the header file for a description of the `DEV_INFO` structure.

Example

```
// Show voltage and current ranges for an accelerator power supply
DEV_INFO info;
int devindex = 0;  // select first accelerator supply in system
int errcode = C3600_GetDevInfo(0, DEVTYPE_ACCELERATOR, devindex, &info, 1000);
if (errcode != C3600_ERR_OK)
  printf("C3600_GetDevInfo() error: %s\n", C3600_GetErrorString(errcode));
else if (!info.is_detected)
  printf("system doesn't have an accelerator supply\n");
else if (!info.is_valid)
  printf("warning: corrupt device attributes\n");
else {
  printf("normal operating ranges for accelerator power supply:\n");
  printf("voltage: %d to %d V\n", info.range[0].min, info.range[0].max);
  printf("current: %d to %d A\n", info.range[1].min, info.range[1].max);
}
```

C3600_GetFaultFlags()

Prototype

```
int C3600_GetFaultFlags(int sysid, DEVTYPE devtype, int devindex, int *flags, int maxwait);
```

Arguments

`sysid` – system ID in range 0 to 15.

`devtype` – enumerated device type.

`devindex` – device index.

`flags` – buffer that will receive fault flags.

`maxwait` – maximum milliseconds to wait for system reply.

Return value

API error code.

Description

This function reads fault information from a device. See the header file for a list of fault flags.

Example

```
// Check for faults on a filament supply
int flags;
int errcode = C3600_GetFaultFlags(0, DEVTYPE_FILAMENT, 0, &flags, 1000);
if (errcode == C3600_ERR_OK)
  if (flags != 0)
    printf("WARNING! Faults detected on filament supply.\n");
```

C3600_GetStatusFlags()

Prototype

```
int C3600_GetStatusFlags(int sysid, DEVTYPE devtype, int devindex, int *flags, int maxwait);
```

Arguments

`sysid` – system ID in range 0 to 15.

`devtype` – enumerated device type.

`devindex` – device index.

`flags` – buffer that will receive status flags.

`maxwait` – maximum milliseconds to wait for system reply.

Return value

API error code.

Description

This function copies device status information to `flags`. See the header file for a list of status flags.

Example

```
// Report status of accelerator output enable
int flags;
int errcode = C3600_GetStatusFlags(0, DEVTYPE_ACCELERATOR, 0, &flags, 1000);
if (errcode == C3600_ERR_OK)
  if (flags & STATUS_OUTPUT_ENABLED)
    printf("HV output is enabled.\n");
```

C3600_SetOutput()

Prototype

```
int C3600_SetOutput(int sysid, DEVTYPE devtype, int devindex, double data, int enab, int maxwait);
```

Arguments

`sysid` – system ID in range 0 to 15.

`devtype` – enumerated device type.

`devindex` – device index.

`data` – setpoint value.

`enab` – output enable: 1 = enable; 0 = disable.

`maxwait` – maximum milliseconds to wait for system reply.

Return value

API error code.

Description

This function programs a device's output level and enables or disables the output.

`data` sets the desired output value. The measurement units depend on `devtype`: Amps are used for `DEVTYPE_FILAMENT`; Volts are used for all other device types.

`enab` controls the power supply output enable. Set this to 1 to enable the output, or 0 to disable the output.

Example

```
// Set anode voltage to 47.5 kV
int errcode = C3600_SetOutput(0, DEVTYPE_ACCELERATOR, 0, 47500, 1, 1000);
if (errcode == C3600_ERR_OK)
  printf("HV supply was enabled and set to 47.5 kV\n");
```

C3600_GetMeter()

Prototype

```
int C3600_GetMeter(int sysid, DEVTYPE devtype, int devindex, int meterid, double *data, int *raw, int maxwait);
```

Arguments

`sysid` – system ID in range 0 to 15.

`devtype` – enumerated device type.

`devindex` – device index.

`meterid` – enumerated meter type. Set to `METER_V`, `METER_I` or `METER_T`.

`data` – buffer that will receive calibrated meter data.

`raw` – buffer that will receive uncorrected meter data. Set to `NULL` if not needed.

`maxwait` – maximum milliseconds to wait for system reply.

Return value

API error code.

Description

This function reads meter data from a device.

`meterid` specifies the type of meter to read. Set this to `METER_V`, `METER_I` or `METER_T` to read the measured output voltage, output current, or temperature.

When this function returns, the calibration-corrected meter data will be stored in `data`. Measurement units depend on `meterid`; the units for meter types `METER_V`, `METER_I` and `METER_T` are Volts, Amps and degrees C, respectively. Uncorrected meter data is returned in `raw`; set to `NULL` if this is not needed.

Example

```
// Measure anode voltage and beam current at output of accelerator power supply
double anodeV, beamI;
int devindex = 0;  // select first accelerator supply
int errcode = C3600_GetMeter(0, DEVTYPE_ACCELERATOR, devindex, METER_V, &anodeV, NULL, 1000);
if (errcode == C3600_ERR_OK) {
  errcode = C3600_GetMeter(0, DEVTYPE_ACCELERATOR, devindex, METER_I, &beamI, NULL, 1000);
  if (errcode == C3600_ERR_OK) {
    printf("anode voltage = %d V\n", (int)anodeV);
    printf("beam current = %d uA\n", (int)(beamI * 1000000));
  }
}
```

C3600_GetOutput()

Prototype

```
int C3600_GetOutput(int sysid, DEVTYPE devtype, int devindex, double *data, int *raw, int *enab, int maxwait);
```

Arguments

`sysid` – system ID in range 0 to 15.

`devtype` – enumerated device type.

`devindex` – device index.

`data` – buffer that will receive calibrated setpoint value. Set to `NULL` if not needed.

`raw` – buffer that will receive uncorrected setpoint value. Set to `NULL` if not needed.

`enab` – buffer that will receive output enable status: 1 = enabled; 0 = disabled.

`maxwait` – maximum milliseconds to wait for system reply.

Return value

API error code.

Description

This function reads the programmed setpoint and output enable from a device. When this function returns, the setpoint will be stored in `data` and the output enable will be stored in `enab`.

In normal operation, `data` will match the most recently programmed setpoint value. The measurement units depend on `devtype`: Amps are used for `DEVTYPE_FILAMENT`; Volts are used for all other device types. Note that the programmed setpoint is automatically zeroed when the device is reset; such resets may be expected (e.g., upon system boot or reset command) or unexpected (e.g., due to device watchdog timeout).

Similarly, in normal operation, `enab` will equal the most recently programmed value, or zero if the device has been reset since the output enable was last programmed.

The uncorrected setpoint is returned in `raw`; set to `NULL` if this is not needed.

Example

```
// Display programmed settings on a filament power supply
double data;
int enab;
int errcode = C3600_GetOutput(0, DEVTYPE_FILAMENT, 0, &data, NULL, &enab, 1000);
if (errcode == C3600_ERR_OK) {
  printf("programmed settings on filament supply: ");
  printf("setpoint = %d mA, ", (int)(data * 1000));
  printf("output %s\n", enab ? "enabled" : "disabled");
}
```

C3600_SysReset()

Prototype

```
int C3600_SysReset(int sysid, int hard, int maxwait);
```

Arguments

`sysid` – system ID in range 0 to 15.

`hard` – reset type: 1 = hard reset; 0 = soft reset.

`maxwait` – maximum milliseconds to wait for system reply.

Return value

API error code.

Description

This function invokes a system reset.

A system may become unresponsive due to arcing or other electrical disruptions, or as a result of application program errors. In such cases, this function can be used to restore normal operation.

If this function completes normally then all devices in the system were successfully reset (outputs disabled, setpoints zeroed) and normal operation can resume. If the function times out then it may be necessary to cycle system power to restore normal operation. For other types of errors, it is recommended to retry this function before resorting to power cycling.

`hard` specifies the method used to invoke the reset. It is recommended to set `hard` = 1.

In multi-threaded applications, the calling thread should notify other threads that the system has been reset.

Example

```
switch (C3600_SysReset(0, 1, 3000)) {  // allow 3 seconds – enough time for more complex systems
  case C3600_ERR_OK:
    printf("system reset successful\n");
    // todo: notify other threads that the system was reset
  case C2600_ERR_READ_TIMEOUT:
    printf("system reset failed; try power cycling the system\n");
  default:
    printf("system reset failed; try calling C3600_SysReset() again\n");
}
```