Digital Controller of Double Power Supply Units

Firmware Version 1-00

User Manual

Document version 2, created on Oct-15-2021

Contents

Figure List

Table List

Software Utilities

The software utilities can be found in the directory "Program" of the enclosed software package. Before using them with a device with a USB interface, the virtual USB port driver must be installed (see section "Driver Installation"). The utilities do not require any additional installation, you only need to copy them to a suitable directory on your computer.

Before starting the utilities, you need to obtain the number of the port to which the device is connected. Devices with an RS-232 interface have to be connected to a physical serial port (for instance COM1). In Windows™ systems, you can find out its number using the device manager. Devices with a USB interface use a virtual serial port for communication. The details are described in the section "Driver Installation".

Utility COM-HVPSU2D-Control

`COM-HVPSU2D-Control` is a Windows™ program that runs in text mode. It enables you to control and monitor the pulse controller, manage its configurations, and backup and restore its data. Executing the utility `COM-HVPSU2D-Control.exe` in a Windows™ command shell[†] without any additional parameters displays a help text with the list of all available commands:

```
COM-HVPSU2D-Control
```

To start the program without any error messages, at least the number of the COM port must be given as a parameter:

```
COM-HVPSU2D-Control 6
```

This command starts the utility `COM-HVPSU2D-Control` and assumes that the device is connected to the (virtual) port COM6. When successful, the utility displays the following message:

[†] Press the Windows key + R to open the "Run" dialog box. Type "cmd" and press Enter, this opens a window with a command prompt. Then change the directory to the one containing the program files using the command "cd". Finally, execute the given command by copying & pasting and pressing "Enter". A better and more comfortable alternative to the Windows™ command shell are utilities such as "Total Commander", "File Commander/W", or "File and archive manager (FAR)". Please use an internet search engine to find out how to obtain these applications.

```
COM6 opened for the communication port 0
Press '?' for help
```

The utility enters the interactive mode and waits for command input.

In case of any problems, check whether the port number matches the system settings and whether the connected device is powered on and working properly. If an error occurs, please consult the section "Error Codes". Tables 10 and 11 explain the possible error messages; they should help you to locate the reason for the software failure.

To check the communication, press the 'p' key to obtain the product identification text. The device should respond as follows:

```
Product identification: HV-PSU-CTRL-2D, Rev.1-00
```

If the device responds properly, you can enter other program commands. Press '?' to obtain the help listing of all available commands[‡].

In practice, you may prefer to use the command line mode instead of the interactive mode. For instance, the former allows you to save complete commands in batch files for repeated usage.

Tables 2-9 summarize all allowable command line parameters of the utility `COM-HVPSU2D-Control`. Besides the parameters, the tables also list the functions of the software interface (see section "Software Interface") that are called by the utility when the respective parameter is executed. Refer to the descriptions of the functions for more details. Note that several command line parameters, e.g. most of the commands for managing the configurations, do not have any equivalent functions. They either call several functions or they use functions not intended to be called by the user directly.

The parameters are processed from left to right. When encountering an error in the command line, the program stops with an error text indicating the command line parameter in which the error occurred and displaying the help text with the list of all available commands.

In several cases, the command line parameters can be specified as either a lowercase or an uppercase character. Use capitals, i.e. uppercase characters, if a continuous operation should be initiated. In

[‡] Note that keyboard layouts different to the US one may cause issues when evaluating several characters. We recommend to switch to the US keyboard layout when using the utility COM-HVPSU2D-Control in the interactive mode.

Tab. 2. Command line parameters of the program `COM-HVPSU2D-Control` - General control.

Parameter	Explanation
`-b, -B`	get the device buffer status (see function **`COM_HVPSU2D_GetBufferState`**)
`-z, -Z`	purge the communication (see function **`COM_HVPSU2D_DevicePurge`**)
`-t, -T`	terminate the program
`-q, -Q`	quiet mode
`-g`	debug mode
`-G`	debug mode with output into `Debug.txt`
`-$`*`speed`*	set the communication speed to the value *`speed`* (see function **`COM_HVPSU2D_SetBaudRate`**)
`-!`	restart the device (see function **`COM_HVPSU2D_Restart`**)
`-?`	show the help text with the list of all available commands

the continuous mode, an operation is repeated until the user interrupts it by pressing a certain key - usually Esc.

Numerical integer values can be specified either in decimal code or in hexadecimal or binary ones when characters `h` or `b` are appended. This implies that, for example, `16`, `10h` or `10000b` all specify the same value, namely sixteen.

If you wish to specify a name parameter containing spaces or special characters, such as a pathname of a file containing spaces, use the conventions valid for your operating system. In Windows™ systems, for instance, enclose the names in quotation marks.

General Control

The command line parameters for the general control are listed in Tab. 2. They are intended to control the function of the utility.

The communication speed defaults to 9600 baud (see section "Software Interface"). This is usually sufficient for any short data transfer but may be too slow when the device data are queried periodically or when large data amounts should be transferred. In such cases, the communication speed can be set to a higher value using the parameter `-$`. To set the communication speed to the maximum supported value of 230.4 kbaud, enter the following command:

```
COM-HVPSU2D-Control 6 -$230400
```

Note that the parameter `-$` is processed the same as any other parameter, thus the changed communication speed is only valid for all subsequent commands, i.e. for all parameters following the parameter `-$`. The communication speed can be changed several times within the command line. This makes it possible to increase the communication speed for data-intensive transmissions and later reset it to the default value for other commands. Also note that the communication speed is automatically reset to the default value of 9600 baud when the utility stops. For more details, see section "Software Interface" and function **`COM_HVPSU2D_SetBaudRate`**.

If the command line includes the parameter `-t`, the program terminates without processing any following parameters. If you do not specify the parameter `-t` at all, the program does not stop and enters the interactive mode after having processed the complete command line.

The quiet mode activated by the parameter `-q` reduces the text output of the program. In contrast to that, the debug mode provides a detailed output for error analysis. It can be activated by the parameter `-g`. The parameter `-G` activates the debug mode and additionally creates a protocol file `Debug.txt` in which it saves the time of occurrence and the descriptions of all communication issues. In case of any problems that you cannot resolve yourself, describe the issues in detail and send this to the manufacturer together with the debug protocol `Debug.txt`.

To restart the device, the parameter -! has to be specified. The utility waits until the device completes its restart and responds again. During the restart, the complete configuration is stored in the non-volatile memory (NVM). Thus, a restart can be used to store data and prevent an accidental data loss.

Tab. 3. Command line parameters of the program `COM-HVPSU2D-Control` - System monitoring.

Parameter	Explanation
`-v`	get the device firmware version (see function **`COM_HVPSU2D_GetFWVersion`**)
`-V`	get the device hardware version (see functions **`COM_HVPSU2D_GetHWType`** and **`COM_HVPSU2D_GetHWVersion`**)
`-d`	get the device firmware date (see function **`COM_HVPSU2D_GetFWDate`**)
`-p`	get the product identification text (see function **`COM_HVPSU2D_GetProductID`**)
`-n, -N`	get the product number (see function **`COM_HVPSU2D_GetProductNo`**)
`-u`	get the device uptime (see functions **`COM_HVPSU2D_GetUptime`** and **`COM_HVPSU2D_GetTotalTime`**)
`-U`	get the device uptime periodically
`-c`	get the CPU data (see function **`COM_HVPSU2D_GetCPUData`**)
`-C`	get the CPU data periodically
`-h`	get the device housekeeping data (see function **`COM_HVPSU2D_GetHousekeeping`**)
`-H`	get the device housekeeping data periodically

System Monitoring

The command line parameters for system monitoring are listed in Tab. 3. They are useful in case of a malfunction to collect the data necessary for describing the issues.

Using the parameter `-u` or `-U`, you can monitor the uptime of the device:

```
COM-HVPSU2D-Control 6 -u -t
```

Tab. 9. Command line parameters of the program `COM-HVPSU2D-Control` - Backup and restore.

Parameter	Explanation
`-y` *`FileName`*	backup the NVM data into a text file with the name *`FileName`*
`-Y` *`FileName`*	restore the NVM data from a text file with the name *`FileName`*

Backing Up and Restoring the Data

All data stored in the device's NVM, i.e. the device settings including the current configuration as well as the user configurations can be backed up or restored (see Tab. 9). A restore procedure rolls back the device exactly to the state it was in at the time of the backup. Thus, if any adjusting of the settings is planned, it is recommended to first create a backup with which the original state can be restored.

Since the back up and restore procedures transfer a large amount of data, it is recommended to increase the communication speed by specifying the parameter `-$` (see section "General Control"). To back up the system memory into a data file `Memory.txt` at a communication speed of 230.4 kbaud, execute the following command:

```
COM-HVPSU2D-Control 6 -$230400 -y MemoryData.txt
   -t
```

This command downloads the memory data from the device into the file `Memory.txt`.

To restore the data at the same communication speed, execute the following command:

```
COM-HVPSU2D-Control 6 -$230400 -Y MemoryData.txt
   -t
```

This command uploads the memory data from the file `Memory.txt` to the device. Since all data in the NVM will be overwritten, the utility asks for a confirmation twice.

Utility FlashLoader

FlashLoader is a simple Windows™ program running in text mode. It enables you to upgrade the firmware of the pulse controller. You should perform an upgrade when you have received or downloaded a new firmware file from the device manufacturer. Launching the utility `FlashLoader.exe` without any parameters displays a simple help text with the expected syntax of the command line.

FlashLoader is a universal utility for many different devices. The pulse controller uses a variable data rate for communication, thus a utility with the revision number 1-20 or later must be used and it must be launched with the command line parameter `-$`.

Before upgrading the firmware, you should first test the device and the communication by verifying the current firmware version. To do so, execute the following command:

```
FlashLoader 6 Firmware.txt -$ -v
```

where `Firmware.txt` is the file containing the current firmware and the number 6 indicates the port COM6 to which the device is connected. The program should produce the following output:

```
Code file Firmware.txt from 06/18/2021, 12:00:00
Flash Loader 2.00
Verifying code file Firmware.txt
Verification completed on Mon, 06/21/2021, 12:34:56
23293 (5AFDh) bytes processed, 24064 (5E00h) bytes
verified
Resetting the target
Program completed successfully
```

For the verifying procedure, a flash-loader utility on the device is activated. When the verification is completed without any errors, the device is restarted.

! **Attention:** To ensure that the device cannot activate the attached switches or other peripherals and produce erratic signals while FlashLoader is active, disconnect the power cables of the switches or turn off the power supply units providing the supply voltages for the switches.

If any error occurs, do not proceed with the firmware upgrade. If you cannot resolve the issues, contact the manufacturer. Note that even if the verification fails and the flash loader on the device remains active, it is safe to power the device off to restart it. However, a safer and more comfortable alternative is to execute the following command:

```
FlashLoader 6 -$ -i -f
```

This prevents the utility on the host computer from initializing the flash loader utility on the microcontroller and sends the reset command to the device.

! **Attention:** If the flash loader on the device is still active and you do not specify the command line parameter `-i`, the programming utility sends data to the device at a wrong data rate during initialization. This data stream cannot be received properly, the device ends up in an undefined state and does not respond anymore. It must be powered off and on to restart the controller. Since there is no other way to stop the flash loader on the device and resume normal operation, be sure to exactly follow the instructions in this section.

If the verification has succeeded, you may start the firmware upgrade by entering the command:

```
FlashLoader 6 Firmware.txt -$
```

The parameter `Firmware.txt` is the file with the new firmware. The program should produce the following output:

```
Code file Firmware.txt from 06/18/2021, 12:00:00
Flash Loader 2.00
Programming code file Firmware.txt
Programming completed on Mon, 06/21/2021, 12:34:56
23293 (5AFDh) bytes processed, 24064 (5E00h) bytes
programmed
Resetting the target
Program completed successfully
```

For the programming procedure, a flash loader utility on the device is activated as well. When the programming is completed, the device is restarted with the new firmware. You can recognize this by the startup sequence of the LED on the front panel of the device.

If an error occurs, the flash loader utility on the microcontroller may remain active. This is the case if the device did not restart. In this case, you should reattempt the upgrade with the command line parameter `-i`:

```
FlashLoader 6 Firmware.txt -$ -i
```

This will prevent the utility on the host computer from initializing the flash loader utility on the microcontroller and it will only try to reprogram the file `Firmware.txt`. If the error persists, contact the manufacturer.

! **Attention:** You must not power down the device if the firmware upgrade has not succeeded. Otherwise, the device will not operate properly or it might not even restart at all. If this were to happen, it would be necessary to reprogram the device in the factory.

Driver Installation

Installation of the Virtual Port for the USB Interface

The virtual port driver is required for the operation of the device with a USB interface. If your operating system is Windows™, please note the following:

- Please use the update function of the operating system at the host computer or download the most recent driver from the homepage of the manufacturer of the USB adapter. The drivers are located at the following address: http://www.ftdichip.com/Drivers/VCP.htm. Please choose the correct driver version according to your operating system.
- To install the driver, administrative rights are required.
- The installation is described in detail in the "Installation Guides" available at the abovementioned address. Please read this description carefully before starting the installation.
- After the installation, the number of the virtual port can be set. You can change the settings in the device manager by opening the settings of the device *USB Serial Port (COMx)*. To modify the settings, administrative rights are required. The settings are applied immediately, you do not need to reboot the PC to activate them.

The software can also be used on computers running Linux. You can run it using the Windows™ emulator Wine (see http://www.winehq.org/).

Starting with Linux Kernel 3.0.0-19, all FTDI devices are already supported without the need to compile any additional kernel modules. For more details, consult the homepage of the manufacturer of the USB adapter: http://www.ftdichip.com/Drivers/VCP.htm.

The system has to be configured in the following way:

- Use a program such as 'dmesg' to find out which USB port the device is connected to: Look for a line similar to "FTDI USB Serial Device converter now attached to ttyUSB0"
- Link the Linux device to the virtual COM port of wine:
  ```
  ln -s /dev/ttyUSB0 ~ /.wine/dosdevices/com6
  ```
  This assumes that the device is attached to ttyUSB0 and will be linked with COM6.

Software Interface

The software interface for the device consists of a 32- or 64-bit dynamic link library `COM-HVPSU2D.dll`. Both versions are located in the directory "Program" of the enclosed software package. The software interface is a stand-alone software package, it does not require any additional library or driver, except for the virtual port driver (see section "Driver Installation").

The user functions in the dynamic link library `COM-HVPSU2D.dll` can be called from any conventional programming language. For the details, please consult the user manual of your compiler. The definition of the interface functions is located in the declaration file `COM-HVPSU2D.h` written in C/C++. If your compiler cannot create an import library from the dynamic link library `COM-HVPSU2D.dll`, please link the library `COM-HVPSU2D.lib` instead of the dynamic link library to your project.

Functionality of the Software Interface

The software interface typically controls up to 16 communication channels for the data transfer to and from the device (see the constant `COM_HVPSU2D_MAX_PORT` in the declaration file `COM-HVPSU2D.h`). This means it can control up to 16 devices at a time.

Before utilizing any function from the dynamic link library `COM-HVPSU2D.dll`, the software should check the version of the library by calling the function **`COM_HVPSU2D_GetSWVersion`**. Note that a library with a different version number can contain different implementations of the functions and calling them may lead to unpredictable results; in most cases, the software will crash.

Each communication channel must be opened before starting the communication. The opening procedure (function **`COM_HVPSU2D_Open`**) configures the used physical (RS-232) or virtual (USB) serial port and clears the port buffers.

The communication channel should be closed at the end of the program. If this does not happen, the software interface automatically does it for you when the dynamic link library `COM-HVPSU2D.dll` is unloaded from the system memory.

The communication speed of the serial port defaults to 9600 baud but can be changed to any value up to 230.4 kbaud (see function **`COM_HVPSU2D_SetBaudRate`**). Please note that the highest speed can be used with any USB connection.

When using the RS-232 interface, check the host hardware for the maximum available communication speed. Most simple serial ports in PCs only support communication speeds up to 115.2 or 128 kbaud. They may also lose data characters when receiving the data from the device. If you encounter such issues, replace the serial-port hardware by a high-speed adapter card with a dedicated system driver. USB adapters providing one or more serial ports usually do not cause any issues either. However, due to the latency of the USB protocol, they are significantly slower than serial ports placed directly on a PC main-board or than serial-port cards.

If a real-time control of the hardware with a temporal resolution of 1 ms or better should be implemented, a high-speed adapter card with a serial port (RS-232) is the recommended solution. When run at the communication speed of 230.4 kbaud, a typical command consisting of 2-3 ASCII characters can be sent within only 0.1-0.2 ms. In contrast to that, a typical USB transmission may take up to 50 ms. This means that, usually, only less than 10 bidirectional transfers per second can be achieved when USB is used.

The communication uses the handshake lines of the serial port. Thus, when using the RS-232 interface, be sure that you use a connection cable that connects all pins in the DE-09 (D-Sub-9) connector. The device emulates a null-modem, i.e. the line DTR is replicated by the hardware as DSR and DCD and the line RTS is replicated by the firmware as the signal CTS. If time-consuming tasks like bulk operations with the NVM are performed, the device deasserts the signal CTS and indicates that it is not ready to respond. If the host deasserts the lines DTR and/or RTS, the input communication buffer of the device is cleared. This can be used to repair the communication if the response becomes erratic (see function **`COM_HVPSU2D_Purge`**). Moreover, if the lines DTR and/or RTS are deasserted for more than 100 ms, the communication interface is reset and the communication speed is set to the default value of 9600 baud. This happens automatically when the software stops using the serial port, thus the communication with the device always starts at the default speed of 9600 baud.

As a first parameter, most of the interface functions require the variable `PortNumber`, which is the number of the communication channel that should be used for the operation in question. This number is an unsigned integer that must be lower than `COM_HVPSU2D_MAX_PORT`, i.e. it must be in the range from 0 to 15.

The return value of most functions contains a number indicating the success of the operation. The return value is a signed 32-bit number (`int`). The last return value can be reloaded by the function **`COM_HVPSU2D_GetInterfaceState`**. Table 10 summarizes the possible return values together with the error messages, which can also be obtained by the function **`COM_HVPSU2D_GetErrorMessage`**. If a data transfer failure has occurred, you can find the reason by calling the functions **`COM_HVPSU2D_GetIOState`** and **`COM_HVPSU2D_GetIOErrorMessage`**. The former returns the last I/O error, the latter the corresponding error message (see Tab. 11).

If you encounter any problems with the error messages or with establishing the communication, please contact the manufacturer of the device.

Direct Command Control

As an alternative, direct control by a terminal program or another software is possible. When using self-written software instead of the supplied dynamic link library `COM-HVPSU2D.dll`, the programmer must make sure to collect the data response after issuing a command and to handle the possible communication errors.

The hardware communication protocol uses 8 data bits, 2 stop bits, and even parity. The commands use ASCII characters only. Each data transfer consists of a command character, optional control data, and a termination character.

The termination character is the Carriage-Return code (CR, 0D hexadecimal; in the following text, it is referred to as the symbol ⏎). The termination character initiates the command execution in the device, which then responses with ASCII characters that are similar to the command.

Most parameters are transferred as a sequence of hexadecimal digits in uppercase; the most significant digit is transferred first. In several commands, Boolean parameters are used. They use the uppercase characters 'Y' for true and 'N' for false. An ASCII character string is transferred as is but without the null-termination. Instead, the termination character of the command indicates the end of the string.

If a transmitted command is misspelled or unknown or if it contains invalid parameters, the device will not process it. The command and its data are cleared from the input buffer and the device does not provide any response. This implies that the communication control should specify a certain timeout and if no response is received within this timeout, the command has to be considered erroneous. The timeout depends on the interface latency; with USB, the recommended value is 100 ms. With the RS-232 interface, significantly shorter timeouts can be used. However, the proper timeout value may also be influenced by the operating system of the host and the serial-port hardware. Thus, it should be determined experimentally. If the error detection is not time-critical in the application in question, the value recommended for the USB interface (i.e. 100 ms) can also be used for the RS-232 interface.

If the device does not respond or if the response is invalid, use the handshake lines of the serial port to clear the input device buffer (see section "Functionality of the Software Interface" and function **`COM_HVPSU2D_Purge`**) before reattempting the operation. The clearing procedure should start with deasserting the lines DTR and/or RTS and waiting for an inactive signal CTS which indicates that the device is not ready and that it has recognized the clear command. Then, the abovementioned handshake lines should be reasserted and the device should respond with activating the signal CTS to indicate that it is ready to receive further commands. Please ensure that the software controlling the communication does not start to send any data to the device while the signal CTS is still inactive, i.e. when the device is not yet ready. The device would either not receive the data at all or the first characters of the sent sequence may be lost.

If you use a communication speed different from the default value of 9600 baud, be sure to deassert the lines DTR and/or RTS for a short time only. It is recommended to assert the handshake lines immediately after the device has responded with an inactive signal CTS. If the handshake lines were deasserted for a longer time period (see section "Functionality of the Software Interface"), the communication speed would be set to the default value and it would need to be adjusted again by the function **`COM_HVPSU2D_SetBaudRate`**.

The direct control is summarized in the following sections by the items "Command" and "Response" below the declaration of a particular function. Note that several functions like **`COM_HVPSU2D_Open`** or **`COM_HVPSU2D_Close`** do not send any data to the device and thus do not have any equivalent direct control commands.

Error Codes

Tab. 10. Return values of the interface functions

Return value	Error message	Description
0	No error	The data transfer was completed successfully.
-1	PortNumber out of range	The parameter PortNumber specified when calling the function is out of range.
-2	Error opening the port	The port could not be opened. For possible reasons, see Tab. 11.
-3	Error closing the port	The port could not be closed. For possible reasons, see Tab. 11.
-4	Error purging the port	The port buffers could not be cleared.
-5	Error setting the port control lines	The port control lines could not be set.
-6	Error reading the port status lines	The port status lines could not be read.
-7	Error sending command	The data transfer to the device failed. For possible reasons, see Tab. 11.
-8	Error sending data	
-9	Error sending termination character	
-10	Error receiving command	The data transfer from the device failed. For possible reasons, see Tab. 11.
-11	Error receiving data	
-12	Error receiving termination character	
-13	Wrong command received	The device sent an unexpected response.
-14	Wrong argument received	
-15	Wrong argument passed to the function	One of the arguments passed to the function was out of the allowable range.

Return value	Error message	Description
-16	Error setting the baud rate	The data rate could not be set. Check if the host is able to communicate at the desired speed.
-100	Device not connected	The port status lines indicate that the device is not connected.
-101	Device not ready	The port status lines indicate that the device is not ready. Communication with the device is only possible if it is not executing any process.
-102	Device state could not be set to not ready	The device did not react properly. Try to reset the communication or restart the device by powering it off and on.
-400	Error opening the file for debugging output	The file for the debugging output cannot be opened for writing. Check if you have permission to perform this action or if the file already exists and is in use by another application.
-401	Error closing the file for debugging output	The file for the debugging output cannot be closed. Check if access to the file is still possible.

Tab. 11. I/O errors

Return value	Error message	Description
0	No error	The data transfer was completed successfully.
1	Port has not been opened yet	You attempted to use the communication channel before having opened it.
2	Cannot open the port	The specified port could not be opened. Either the port does not exist or it is currently being used by another program.
3	Cannot obtain communication error	The system could not get the last communication error of the port.
4	Cannot get the state of the port	The system could not get the state of the port.
5	Cannot set the state of the port	The system could not set the state of the port.
6	Break timing error	The system could not issue a break with the proper timing.
7	Cannot place the transmission line in a break state	The system could not issue a break. Check the data sheet of the used communication port.
8	Cannot place the transmission line in a nonbreak state	The system could not clear the break state. Check the data sheet of the used communication port.
9	Cannot set the timeouts for the port	The system could not set the timeouts for the port.
10	Cannot clear the port	The system could not clear the port buffers.
11	Error reading data from the port	The system could not read data from the port. Most probably, no data is available because the device is either disconnected or does not respond.
12	Error writing data to the port	The system could not write data to the port.

Return value	Error message	Description
13	Wrong data amount written to the port	The system could not write the proper amount of data to the port.
14	Error setting the control lines of the port	The system could not set the state of the port control lines.
15	Error reading the status lines of the port	The system could not get the state of the port status lines.
16	Device is busy	The system could not access the device since a background process is active. Wait until it finishes.

Communication Control

Function COM_HVPSU2D_Open

```
int COM_HVPSU2D_Open (WORD PortNumber,
  WORD COMNumber);
```

Opens the communication channel and returns an error code according to Tab. 10.

The function opens the channel with the number `PortNumber` and attaches it to the serial port with the number `COMNumber`. The variable `PortNumber` must be lower than the number of implemented channels `COM_HVPSU2D_MAX_PORT` (see the declaration file `COM-HVPSU2D.h`).

The serial port number `COMNumber` must point to a valid serial port to which the controller is attached. Note that this number is the number of the COM port, i.e. `COMNumber = 1` points to the port COM1. The function accepts all numbers that are supported by the operating system, i.e. any port between COM1 and COM255 can be used for the communication.

You must call the function **`COM_HVPSU2D_Open`** prior to any other communication function. If the function returns an error, the communication channel remains closed and no data communication is possible.

Function COM_HVPSU2D_Close

```
int COM_HVPSU2D_Close (WORD PortNumber);
```

Closes the communication channel and returns an error code according to Tab. 10.

You can use this function to free the used port for another application.

If an application that has exclusively used the software interface `COM-HVPSU2D.dll` finishes, the opened communication channel closes automatically. Thus, the programmer does not need to call the function **`COM_HVPSU2D_Close`** explicitly.

Function COM_HVPSU2D_SetBaudRate

```
int COM_HVPSU2D_SetBaudRate (WORD PortNumber,
  unsigned & BaudRate);
```

Command: *$BBBBB*⏎

Response: *$BBBBB*⏎

Sets the communication speed to the baud rate given by the variable `BaudRate`, modifies it to the set value (see below) and returns an error code according to Tab. 10.

The device is optimized for communication speed of 9600 baud and its multiples, i.e., for instance, 115.2 or 230.4 kbaud (see function **COM_HVPSU2D_GetCPUData**). If the communication speed is set to another value such as 128 kbaud, the device will use a slightly different speed. This value is returned in the variable `BaudRate`. Note that if the difference between the communication speed set in the host and in the device is less than about 5%, the communication can still be established without any errors.

If communicating with the device is no longer possible, call the function **COM_HVPSU2D_Purge** in order to reset the communication speed to the default value of 9600 baud. You can also interrupt the communication, this deactivates the handshake lines and resets communication speed as well.

The direct command (*$BBBBB*⏎) contains 5 hexadecimal digits (*BBBBB*) for the variable `BaudRate`. The response has the same format; if the command is executed successfully, it returns the value of the baud rate set by the device. The response is still transmitted at the old data rate. After it has been received, the data rate must be set to the new value in the host UART to enable communication with the device.

Function COM_HVPSU2D_Purge

```
int COM_HVPSU2D_Purge (WORD PortNumber);
```

Clears the port data buffers and returns an error code according to Tab. 10.

This function can be used to repair a disturbed communication. In case of a user program crash, this function should be called to erase data incorrectly received from the device. It also deactivates the handshake lines to clear the communication buffer of the device and to reset the communication speed to the default value of 9600 baud.

Note that the function **`COM_HVPSU2D_Purge`** is automatically called by the function **`COM_HVPSU2D_Open`** to establish a clean communication start independent of the previous data transfers.

Function COM_HVPSU2D_GetBufferState

```
int COM_HVPSU2D_GetBufferState
  (WORD PortNumber, BOOL & Empty);
```

Command: `Z`⏎

Response: `Z`*E*⏎

Saves the state of the device's input data buffer in the variable `Empty` and returns an error code according to Tab. 10.

When a large amount of data should be transferred to the device, this function can be used to ensure that the input data buffer contains enough free space. If the return value of the variable `Empty` is false, the input buffer is not empty and there is no guarantee that the device will be able to receive the data. This situation can occur if the device has just received a large amount of data and has not yet finished processing it. In such a case, the call to the function **`COM_HVPSU2D_GetBufferState`** should be repeated after several milliseconds until the return value becomes `true`.

The response to the direct command (`Z`⏎) contains one Boolean character (*E*) for the variable `Empty`.

Function COM_HVPSU2D_DevicePurge

```
int COM_HVPSU2D_DevicePurge (WORD PortNumber,
  BOOL & Empty);
```

Command: `z`⏎

Response: `z`*E*⏎

Clears the device's output data buffer and saves the state of the device's input data buffer in the variable `Empty` like the function

COM_HVPSU2D_GetBufferState. The return value is an error code according to Tab. 10.

This function can be used to repair a disturbed communication. If the device does not respond properly, the function **COM_HVPSU2D_DevicePurge** should be called repeatedly until it returns the value `true` in the variable `Empty`.

The response to the direct command (`z`⏎) contains one Boolean character (*E*) for the variable `Empty`.

Device Control

Function COM_HVPSU2D_GetInterlockEnable

```
int COM_HVPSU2D_GetInterlockEnable
  (WORD PortNumber, BOOL & ConOut,
  BOOL & ConBNC);
```

Command: `l`⏎

Response: `l`*OB*⏎

Saves the interlock-enable flags in the variables `ConOut` and `ConBNC` and returns an error code according to Tab. 10.

If the variable `ConOut` is true, the interlock loop in the output connector is enabled and thus, if the loop is open, it disables the function of the power supply units. The variable `ConBNC` serves the same purpose for the interlock at the BNC connector. If both interlock loops are enabled, they both must be closed to enable the function of the power supply units.

The response to the direct command (`l`⏎) contains two Boolean characters (*OB*) for the variables `ConOut` and `ConBNC`, respectively.

Function COM_HVPSU2D_SetInterlockEnable

```
int COM_HVPSU2D_SetInterlockEnable
  (WORD PortNumber, BOOL ConOut,
  BOOL ConBNC);
```

Command: `l`*OB*⏎

Response: `l`*OB*⏎

Sets the interlock-enable flags to the values given by the variables `ConOut` and `ConBNC` and returns an error code according to Tab. 10.

For more details, see function **`COM_HVPSU2D_GetInterlockEnable`**.

The direct command (`l`*OB*⏎) contains two Boolean characters (*OB*) for the variables `ConOut` and `ConBNC`, respectively. If the command is executed successfully, the device responds by repeating the command characters.

Function COM_HVPSU2D_GetMainState

```
int COM_HVPSU2D_GetMainState
  (WORD PortNumber, WORD & State);
```

Command: M↵

Response: M*SSSS*↵

Saves the main device status in the variable `State` and returns an error code according to Tab. 10.

The possible values of the variable `State` are given by the constants `COM_HVPSU2D_XXX` (see the declaration file `COM-HVPSU2D.h`). If the device is working properly, the variable `State` returns the value `COM_HVPSU2D_STATE_ON`. Values higher or equal to `COM_HVPSU2D_STATE_ERROR` indicate an error. Note that the detected errors are also indicated by the LED on the front panel.

The response to the direct command (`M↵`) contains 4 hexadecimal digits (*SSSS*) for the variable `State`.

Function COM_HVPSU2D_GetDeviceState

```
int COM_HVPSU2D_GetDeviceState
  (WORD PortNumber, DWORD & DeviceState);
```

Command: S↵

Response: S*DDDDDDDD*↵

Saves the detailed state of the device in the variable `DeviceState` and returns an error code according to Tab. 10.

The variable `DeviceState` is a bit combination of the constants `COM_HVPSU2D_DEVST_XXX` (see the declaration file `COM-HVPSU2D.h`). If the device is working properly, the variable `DeviceState` returns the value `COM_HVPSU2D_DEVST_OK` (i.e. zero), nonzero values indicate an error. The errors detected by the firmware set the state of the device (see function **`COM_HVPSU2D_GetMainState`**).

The response to the direct command (`S↵`) contains 8 hexadecimal digits (*DDDDDDDD*) for the variable `DeviceState`.

Function COM_HVPSU2D_GetHousekeeping

```
int COM_HVPSU2D_GetHousekeeping
  (WORD PortNumber, double & VoltRect,
  double & Volt5V0, double & Volt3V3,
  double & TempCPU);
```

Command: `H↵`

Response: `H`*FFFFDDDDCCCCTTTT*`↵`

Saves the housekeeping data in the variables `VoltRect`, `Volt5V0`, `Volt3V3`, and `TempCPU`, and returns an error code according to Tab. 10.

The return values in the variables `VoltRect`, `Volt5V0`, and `Volt3V3` are supply voltages in V, their nominal values are 10 V, 5.0 V, and 3.3 V. The return value in the variable `TempCPU` is the temperature of the CPU in ºC.

Note that the return value in the variable `VoltRect` is the rectified voltage supplied by the mains transformer. Its value changes with changing mains voltage but the controller works properly as long as it is larger than about 6 V.

The response to the direct command (`H↵`) contains 3 x 4 hexadecimal digits for the variables `VoltRect` (*FFFF*), `Volt5V0` (*DDDD*), and `Volt3V3` (*CCCC*), followed by 4 hexadecimal digits (*TTTT*) for the variable `TempCPU`. The voltage values are in mV, i.e. in order to convert them into values in V, they have to be divided by 10^3. The temperature variable `TempCPU` is received in units of 10 mK, i.e. it has to be converted by subtracting the offset of 27315 and dividing by 100 to get the value in ºC.

Function COM_HVPSU2D_GetSensorData

```
int COM_HVPSU2D_GetSensorData
  (WORD PortNumber, double
  Temperature [COM_HVPSU2D_SEN_COUNT]);
```

Command: T↵

Response: T*AAAA..ZZZZ*↵

Saves the temperature-sensor data in the array `Temperature` and returns an error code according to Tab. 10.

The return values in the array `Temperature` are temperatures in ºC measured by the temperature sensors.

The size of the array `Temperature` must be sufficient for storing data from all sensors, i.e. it must be equal to the constant `COM_HVPSU2D_SEN_COUNT` (see the declaration file `COM-HVPSU2D.h`).

The response to the direct command (`T↵`) contains 4 hexadecimal digits for each sensor temperature (*AAAA..ZZZZ*). For the temperature units, see the function **`COM_HVPSU2D_GetHousekeeping`**.

Function COM_HVPSU2D_GetLEDData

```
int COM_HVPSU2D_GetLEDData (WORD PortNumber,
  BOOL & Red, BOOL & Green, BOOL & Blue);
```

Command: L↵

Response: L*RGB*↵

Saves the colors of the LED on the front panel in the variables `Red`, `Green`, and `Blue`, and returns an error code according to Tab. 10.

The colors in the variables `Red`, `Green`, and `Blue` are Boolean values, i.e. they indicate if each color is turned on or off.

The response to the direct command (`L↵`) contains three Boolean characters for each color value in the variables `Red` (*R*), `Green` (*G*), and `Blue` (*B*).

Management of the Power Supply Units

Function COM_HVPSU2D_GetPSUState

```
int COM_HVPSU2D_GetPSUState (WORD PortNumber,
  DWORD & Status);
```

Command: ***sn*↵**

Response: ***snSSSSSS*↵**

Saves the status bits of the PSU controller in the variable `Status` and returns an error code according to Tab. 10.

The return value in the variable `Status` is the 24-bit status word of the PSU controller. Its bits are shown in Tab. 12. Note that you cannot modify the bit values directly, the function **`COM_HVPSU2D_GetPSUState`** can only be read out. Some of the bits can be changed indirectly by calling the functions **`COM_HVPSU2D_SetPSUEnable`** (bits `ST_PSU0_ENB_CTRL` and `ST_PSU1_ENB_CTRL`), **`COM_HVPSU2D_SetPSUFullRange`** (bits `ST_PSU0_FULL_CTRL` and `ST_PSU1_FULL_CTRL`), and **`COM_HVPSU2D_SetInterlockEnable`** (bits `ST_ILOCK_OUT_DIS` and `ST_ILOCK_BNC_DIS`). The bit `ST_PSU_ENB_CTRL` is controlled by the device controller and is set if no error conditions exist and the device is enabled (see function **`COM_HVPSU2D_SetDeviceEnable`**).

The direct command (s*n*↵) contains one decimal character (*n*) for the variable `PSU`. The response (s*nSSSSSS*↵) contains the decimal character (*n*) for the variable `PSU` followed by 6 hexadecimal digits for the variable `Status` (*SSSSSS*).

Tab. 12. Status bits of the PSU modules.

Bit	Value	Description
0	ST_ILIM_CTRL	control of the inrush current limiter
1	ST_LED_CTRL_R	red LED color is on
2	ST_LED_CTRL_G	green LED color is on
3	ST_LED_CTRL_B	blue LED color is on
4	ST_PSU0_ENB_CTRL	control enable PSU #0
5	ST_PSU1_ENB_CTRL	control enable PSU #1
6	ST_PSU0_FULL_CTRL	control full range of PSU #0
7	ST_PSU1_FULL_CTRL	control full range of PSU #1
8	ST_ILOCK_OUT_DIS	disable interlock at output connector
9	ST_ILOCK_BNC_DIS	disable interlock at BNC connector
10	ST_PSU_ENB_CTRL	control enable PSUs
11	–	–
12	ST_ILIM_ACT	state of inrush current limiter
13	ST_PSU0_FULL_ACT	state of full range of PSU #0
14	ST_PSU1_FULL_ACT	state of full range of PSU #1
15	ST_RES_N	state of the reset signal RESn (if 0, device is reset)
16	ST_ILOCK_OUT_ACT	evaluated state of interlock at output connector
17	ST_ILOCK_BNC_ACT	evaluated state of interlock at BNC connector
18	ST_ILOCK_ACT	interlock state
19	ST_PSU_ENB_ACT	PSUs enabled
20	ST_PSU0_ENB_ACT	PSU #0 enabled
21	ST_PSU1_ENB_ACT	PSU #1 enabled
22	ST_ILOCK_OUT	state of interlock at output connector
23	ST_ILOCK_BNC	state of interlock at BNC connector

Function COM_HVPSU2D_GetPSUEnable

```
int COM_HVPSU2D_GetPSUEnable
   (WORD PortNumber, BOOL & PSU0,
  BOOL & PSU1);
```

Command: e⏎

Response: e*BB*⏎

Saves the state of the enable flags of the PSU modules in the variables `PSU0` and `PSU1` and returns an error code according to Tab. 10.

If the variable `PSU0` or `PSU1` is true the respective PSU module is enabled.

The response to the direct command (`e⏎`) contains two Boolean characters (*BB*) for the variables `PSU0` and `PSU1`, respectively.

Function COM_HVPSU2D_SetPSUEnable

```
int COM_HVPSU2D_SetPSUEnable
   (WORD PortNumber, BOOL PSU0, BOOL PSU1);
```

Command: e*BB*⏎

Response: e*BB*⏎

Sets the enable flags of the PSU modules to the values given by the variables `PSU0` and `PSU1` and returns an error code according to Tab. 10.

For more details, see function **`COM_HVPSU2D_GetPSUEnable`**.

The direct command (`e`*BB*⏎) contains two Boolean characters (*BB*) for the variables `PSU0` and `PSU1`, respectively. If the command is executed successfully, the device responds by repeating the command characters.

Function COM_HVPSU2D_HasPSUFullRange

```
int COM_HVPSU2D_HasPSUFullRange
  (WORD PortNumber, BOOL & PSU0,
  BOOL & PSU1);
```

Command: r⏎

Response: r*BB*⏎

Saves the range switching capability of the PSU modules in the variables `PSU0` and `PSU1` and returns an error code according to Tab. 10.

If the variable `PSU0` or `PSU1` is true, the respective PSU module is capable of switching the voltage range (see function **`COM_HVPSU2D_GetPSUFullRange`** for more details). If the variable is false, the respective PSU module cannot be switched. It is still possible to call the function **`COM_HVPSU2D_SetPSUFullRange`**, but it will not switch the voltage range.

The response to the direct command (`r⏎`) contains two Boolean characters (*`BB`*) for the variables `PSU0` and `PSU1`, respectively.

Function COM_HVPSU2D_GetPSUFullRange

```
int COM_HVPSU2D_GetPSUFullRange
  (WORD PortNumber, BOOL & PSU0,
  BOOL & PSU1);
```

Command: p⏎

Response: p*BB*⏎

Saves the full-range flags of the PSU modules in the variables `PSU0` and `PSU1` and returns an error code according to Tab. 10.

If the variable `PSU0` or `PSU1` is true, the respective PSU module is configured for the full voltage range. Thus, it can generate voltages up to the nominal output value. If the variable is false, the PSU module can generate approximately half of the nominal output voltage but double the output current.

The response to the direct command (`p⏎`) contains two Boolean characters (*`BB`*) for the variables `PSU0` and `PSU1`, respectively.

Function COM_HVPSU2D_SetPSUFullRange

```
int COM_HVPSU2D_SetPSUFullRange
  (WORD PortNumber, BOOL PSU0, BOOL PSU1);
```

Command: p*BB*⏎

Response: p*BB*⏎

Sets the full-range flags of the PSU modules to the values given by the variables `PSU0` and `PSU1` and returns an error code according to Tab. 10.

For more details, see function **`COM_HVPSU2D_GetPSUFullRange`**.

The direct command (`p`*BB*⏎) contains two Boolean characters (*BB*) for the variables `PSU0` and `PSU1`, respectively. If the command is executed successfully, the device responds by repeating the command characters.

Function COM_HVPSU2D_GetPSUOutputVoltage

```
int COM_HVPSU2D_GetPSUOutputVoltage
  (WORD PortNumber, unsigned PSU,
  double & Voltage);
```

Command: O*n*⏎

Response: O*nVVVVV*⏎

Saves the output voltage of the PSU module with the number `PSU` in the variable `Voltage` and returns an error code according to Tab. 10.

The variable `PSU` is the number of the PSU module. It can be 0 (constant `COM_HVPSU2D_PSU_POS` in the declaration file `COM-HVPSU2D.h`) or 1 (constant `COM_HVPSU2D_PSU_NEG`).

The return value in the variable `Voltage` is the value of the output voltage in V that was previously set by the function **`COM_HVPSU2D_SetPSUOutputVoltage`**.

The direct command (`O`*n*⏎) contains one decimal character (*n*) for the variable `PSU`. The response (`O`*nVVVVV*⏎) contains the decimal character (*n*) for the variable `PSU` followed by 5 hexadecimal digits for the variable `Voltage` (*VVVVV*). The voltage value is in mV, i.e. in order to convert it into a value in V, it has to be divided by 10^3.

Function COM_HVPSU2D_SetPSUOutputVoltage

```
int COM_HVPSU2D_SetPSUOutputVoltage
   (WORD PortNumber, unsigned PSU,
   double Voltage);
```

Command: O*nVVVVV*↵

Response: O*nVVVVV*↵

Sets the output voltage of the PSU module with the number `PSU` to the value in the variable `Voltage` and returns an error code according to Tab. 10.

For more details, see function **`COM_HVPSU2D_GetPSUOutputVoltage`**.

The direct command (`O`*nVVVVV*↵) contains one decimal character (*n*) for the variable `PSU` followed by 5 hexadecimal digits for the variable `Voltage` (*VVVVV*). The voltage value is in mV, for the parameter unit conversion, see function **`COM_HVPSU2D_GetPSUOutputVoltage`**. If the command is executed successfully, the device responds by repeating the command characters.

Function COM_HVPSU2D_GetPSUSetOutputVoltage

```
int COM_HVPSU2D_GetPSUSetOutputVoltage
   (WORD PortNumber, unsigned PSU,
   double & VoltageSet,
   double & VoltageLimit);
```

Command: o*n*↵

Response: o*nSSSSSLLLLL*↵

Saves the set output voltage and the voltage limit of the PSU module with the number `PSU` in the variables `VoltageSet` and `VoltageLimit`, and returns an error code according to Tab. 10.

The variable `PSU` is the number of the PSU module. It can be 0 (constant `COM_HVPSU2D_PSU_POS` in the declaration file `COM-HVPSU2D.h`) or 1 (constant `COM_HVPSU2D_PSU_NEG`).

The return values in the variables `VoltageSet` and `VoltageLimit` are voltages in V. The latter is the limit of the output voltage, the value `VoltageSet` is the voltage that was previously set by the function **`COM_HVPSU2D_SetPSUOutputVoltage`**. However, it is adjusted to be lower than or equal to the voltage limit `VoltageLimit`.

The direct command (`o`*`n`*`↵`) contains one decimal character (*`n`*) for the variable `PSU`. The response (`o`*`nSSSSSLLLLL`*`↵`) contains the decimal character (*`n`*) for the variable `PSU` followed by 2 x 5 hexadecimal digits for the variables `VoltageSet` (*`SSSSS`*) and `VoltageLimit` (*`LLLLL`*). The voltage values are in mV, for the parameter unit conversion, see function **`COM_HVPSU2D_GetPSUOutputVoltage`**.

Function COM_HVPSU2D_GetPSUOutputCurrent

```
int COM_HVPSU2D_GetPSUOutputCurrent
  (WORD PortNumber, unsigned PSU,
  double & Current);
```

Command: `I`*`n`*`↵`

Response: `I`*`nIIIIII`*`↵`

Saves the output current of the PSU module with the number `PSU` in the variable `Current` and returns an error code according to Tab. 10.

The variable `PSU` is the number of the PSU module, for more details, see function **`COM_HVPSU2D_GetPSUOutputVoltage`**.

The return value in the variable `Current` is the value of the output current in A that was previously set by the function **`COM_HVPSU2D_SetPSUOutputCurrent`**..

The direct command (`I`*`n`*`↵`) contains one decimal character (*`n`*) for the variable `PSU`. The response (`I`*`nIIIIII`*`↵`) contains the decimal character (*`n`*) for the variable `PSU` followed by 6 hexadecimal digits for the variable `Current` (*`IIIIII`*). The current value is in µA, i.e. in order to convert it into A, it has to be divided by 10^6.

Function COM_HVPSU2D_SetPSUOutputCurrent

```
int COM_HVPSU2D_SetPSUOutputCurrent
  (WORD PortNumber, unsigned PSU,
  double Current);
```

Command: I*nIIIIII*↵

Response: I*nIIIIII*↵

Sets the output current of the PSU module with the number `PSU` to the value in the variable `Current` and returns an error code according to Tab. 10.

For more details, see function **`COM_HVPSU2D_GetPSUOutputCurrent`**.

The direct command (`I`*`nIIIIII`*↵) contains one decimal character (*n*) for the variable `PSU` followed by 6 hexadecimal digits for the variable `Current` (*`IIIIII`*). The current value is in µA, for the parameter unit conversion, see function **`COM_HVPSU2D_GetPSUOutputCurrent`**. If the command is executed successfully, the device responds by repeating the command characters.

Function COM_HVPSU2D_GetPSUSetOutputCurrent

```
int COM_HVPSU2D_GetPSUSetOutputCurrent
  (WORD PortNumber, unsigned PSU,
  double & CurrentSet,
  double & CurrentLimit);
```

Command: i*n*↵

Response: i*nSSSSSLLLLL*↵

Saves the set output current and the current limit of the PSU module with the number `PSU` in the variables `CurrentSet` and `CurrentLimit`, and returns an error code according to Tab. 10.

The variable `PSU` is the number of the PSU module. It can be 0 (constant `COM_HVPSU2D_PSU_POS` in the declaration file `COM-HVPSU2D.h`) or 1 (constant `COM_HVPSU2D_PSU_NEG`).

The return values in the variables `CurrentSet` and `CurrentLimit` are currents in A. The latter is the limit of the output current, the value `CurrentSet` is the current that was previously set by the function **`COM_HVPSU2D_SetPSUOutputCurrent`**. However, it is adjusted to be lower than or equal to the current limit `CurrentLimit`.

The direct command (`i`*n*⮰) contains one decimal character (*n*) for the variable `PSU`. The response (`i`*nSSSSSSLLLLLL*⮰) contains the decimal character (*n*) for the variable `PSU` followed by 2 x 6 hexadecimal digits for the variables `CurrentSet` (*SSSSSS*) and `CurrentLimit` (*LLLLLL*). The current values are in µA, for the parameter unit conversion, see function **`COM_HVPSU2D_GetPSUOutputCurrent`**.

Function COM_HVPSU2D_GetPSUData

```
int COM_HVPSU2D_GetPSUData (WORD PortNumber,
  unsigned PSU, double & Voltage,
  double & Current, double & VoltDropout);
```

Command: `m`*n*⮰

Response: `m`*nVVVVVIIIIIIDDDDD*⮰

Saves the measured values of the PSU module with the number `PSU` in the variables `Voltage`, `Current`, and `VoltDropout`, and returns an error code according to Tab. 10.

The variable `PSU` is the number of the PSU module, for more details, see function **`COM_HVPSU2D_GetPSUOutputVoltage`**.

The return value in the variable `Voltage` is the measured value of the output voltage in V, the value in the variable `Current` is the measured value of the output current in A. The value in the variable `Volt-Dropout` is the measured value of the voltage dropout on the output voltage regulator, The voltage dropout and the output current determine the power dissipation of the PSU module. For the device to function properly, the voltage dropout should not be lower than 5-10 V.

The direct command (`m`*n*⮰) contains one decimal character (*n*) for the variable `PSU`. The response (`m`*nVVVVVIIIIIIDDDDD*⮰) contains the decimal character (*n*) for the variable `PSU` followed by 5 hexadecimal digits for the variable `Voltage` (*VVVVV*), 6 hexadecimal digits for the variable `Current` (*IIIIII*), and 5 hexadecimal digits for the variable `VoltDropout` (*DDDDD*). The voltage values are in mV, for the parameter unit conversion, see functions **COM_HVPSU2D_GetPSUOutputVoltage** and **COM_HVPSU2D_GetPSUOutputCurrent**.

Function COM_HVPSU2D_GetPSUHousekeeping

```
int COM_HVPSU2D_GetPSUHousekeeping
  (WORD PortNumber, unsigned PSU,
  double & Volt24Vp, double & Volt12Vp,
  double & Volt12Vn, double & VoltRef);
```

Command: k*n*⏎

Response: k*nAAAABBBBCCCCDDDD*⏎

Saves the measured housekeeping values of the PSU module with the number `PSU` in the variables `Volt24Vp`, `Volt12Vp`, `Volt12Vn`, and `VoltRef`, and returns an error code according to Tab. 10.

The variable `PSU` is the number of the PSU module, for more details, see function **COM_HVPSU2D_GetPSUOutputVoltage**.

The return values in the variables `Volt24Vp`, `Volt12Vp`, `Volt12Vn`, and `VoltRef` are voltages in V. The first three values are the supply voltages of +24 V, +12 V, and –12 V, the last value is the reference voltage of the digital-to-analog converters of 10 V. The nominal values are unsigned, i.e. 24 V, 12 V, 12 V, and 10 V.

The direct command (k*n*⏎) contains one decimal character (*n*) for the variable `PSU`. The response (k*nAAAABBBBCCCCDDDD*⏎) contains the decimal character (*n*) for the variable `PSU` followed by 4 x 4 hexadecimal digits for the variables `Volt24Vp` (*AAAA*), `Volt12Vp` (*BBBB*), `Volt12Vn` (*CCCC*), and `VoltRef` (*DDDDD*). The voltage values are in mV, for the parameter unit conversion, see function **COM_HVPSU2D_GetPSUOutputVoltage**.

Function COM_HVPSU2D_GetADCHousekeeping

```
int COM_HVPSU2D_GetADCHousekeeping
  (WORD PortNumber, unsigned PSU,
  double & VoltAVDD, double & VoltDVDD,
  double & VoltALDO, double & VoltDLDO ,
  double & VoltRef, double & TempADC);
```

Command: h*n*⏎

Response: h*nAAAABBBBCCCCDDDDEEEEFFFF*⏎

Saves the measured housekeeping values of the analog-to-digital converter (ADC) of the PSU module with the number `PSU` in the variables `VoltAVDD`, `VoltDVDD`, `VoltALDO`, `VoltDLDO`, `VoltRef`, and `TempADC`, and returns an error code according to Tab. 10.

The variable `PSU` is the number of the PSU module, for more details, see function **COM_HVPSU2D_GetPSUOutputVoltage**.

The return values in the variables `VoltAVDD`, `VoltDVDD`, `VoltALDO`, `VoltDLDO`, and `VoltRef` are voltages in V. The first four values are the supply voltages of the ADC, their nominal values are 3.5 V for the analog supply voltage `AVDD`, 3.3 V for the digital supply voltage `DVDD`, 1.8 V for the analog core supply voltage `ALDO`, and 1.8 V for the digital core supply voltage `DLDO`. The value in the variable `VoltRef` is the internal reference voltage of the ADC, its nominal value is 2.5 V. The return value in the variable `TempADC` is the ADC temperature in ºC.

The direct command (h*n*⏎) contains one decimal character (*n*) for the variable `PSU`. The response (h*nAAAABBBBCCCCDDDDEEEEFFFF*⏎) contains the decimal character (*n*) for the variable `PSU` followed by 6 x 4 hexadecimal digits for the variables `VoltAVDD` (*AAAA*), `VoltDVDD` (*BBBB*), `VoltALDO` (*CCCC*), and `VoltDLDO` (*DDDDD*) , and `VoltRef` (*EEEE*), and `TempADC` (*FFFF*). The voltage values are in mV, for the parameter unit conversion, see functions **COM_HVPSU2D_GetPSUOutputVoltage** and **COM_HVPSU2D_GetHousekeeping**. The reference voltage `VoltRef` is measured indirectly by obtaining the ADC temperature using an external (*FFFF*) and internal (*EEEE*) reference voltage. Thus, the value of the reference voltage has to be determined as a quotient of the values *FFFF* and *EEEE* multiplied by 2.5 V: V_{ref} = *FFFF* / *EEEE* · 2.5 V.

Device Configuration

Function COM_HVPSU2D_GetDeviceEnable

```
int COM_HVPSU2D_GetDeviceEnable
  (WORD PortNumber, BOOL & Enable);
```

Command: E⏎

Response: E*B*⏎

Saves the enable state of the device in the variable `Enable` and returns an error code according to Tab. 10.

The value in the variable `Enable` determines the behavior of the device. If it is true, i.e. it is nonzero and no error conditions exist, the PSU modules are enabled (see bit `ST_PSU_ENB_CTRL` in Tab. 12, i.e. the constant `COM_HVPSU2D_ST_PSU_ENB_CTRL` in the declaration file `COM-HVPSU2D.h`). If the variable `Enable` is false, the PSU modules are disabled.

Note that the value of the enable state is stored in the NVM and is restored during startup. This means that if the device is enabled, it will be enabled at power-on and will produce output voltages.

The response to the direct command (`E⏎`) contains one Boolean character (*B*) for the variable `Enable`.

Function COM_HVPSU2D_SetDeviceEnable

```
int COM_HVPSU2D_SetDeviceEnable
  (WORD PortNumber, BOOL Enable);
```

Command: E*B*⏎

Response: E*B*⏎

Sets the enable state of the device to the value given by the variable `Enable` and returns an error code according to Tab. 10.

For more details, see function **`COM_HVPSU2D_GetDeviceEnable`**.

The direct command (`E`*B*`⏎`) contains one Boolean character (*B*) for the variable `Enable`. If the command is executed successfully, the device responds by repeating the command characters.

Configuration Management

Function COM_HVPSU2D_ResetCurrentConfig

```
int COM_HVPSU2D_ResetCurrentConfig
  (WORD PortNumber);
```

Command: *↵

Response: *↵

Resets the current configuration and returns an error code according to Tab. 10.

This function resets all configuration values of the PSU modules. It resets the values of the output voltages and currents (see functions **COM_HVPSU2D_GetPSUOutputVoltage** and **COM_HVPSU2D_GetPSUOutputCurrent**), enables the device (see function **COM_HVPSU2D_GetDeviceEnable**), enables the PSU modules (see function **COM_HVPSU2D_GetPSUEnable**), enables their full range (see function **COM_HVPSU2D_GetPSUFullRange**), and disables the interlock loops (see function **COM_HVPSU2D_GetInterlockEnable**).

It is recommended to call this function before setting up a new configuration.

If the direct command (`*↵`) is executed successfully, the device responds by repeating the command characters.

Function COM_HVPSU2D_SaveCurrentConfig

```
int COM_HVPSU2D_SaveCurrentConfig
  (WORD PortNumber, unsigned ConfigNumber);
```

Command: *JNN*↵

Response: *JNN*↵

Saves the current device configuration to the configuration in the NVM with the number given by the variable `ConfigNumber` and returns an error code according to Tab. 10.

The configuration with the number given by the variable `ConfigNumber` can be any integer between 0 and

COM_HVPSU2D_MAX_CONFIG-1 (see the declaration file COM-HVPSU2D.h).

Note that the call to the function **COM_HVPSU2D_SaveCurrentConfig** overwrites the previously saved configuration data without any warning.

Any configuration stored in the NVM can be restored, i.e. loaded as the current device configuration by the function **COM_HVPSU2D_LoadCurrentConfig**.

The direct command (J*NN*↵) contains two hexadecimal digits (*NN*) for the variable ConfigNumber. If the command is executed successfully, the device responds by repeating the command characters.

Function COM_HVPSU2D_LoadCurrentConfig

```
int COM_HVPSU2D_LoadCurrentConfig
  (WORD PortNumber, unsigned ConfigNumber);
```

Command: j*NN*↵

Response: j*NN*↵

Loads the current device configuration from the configuration in the NVM with the number given by the variable ConfigNumber and returns an error code according to Tab. 10.

There must be a saved configuration in the NVM from a prior call to the function **COM_HVPSU2D_SaveCurrentConfig**, otherwise the call to the function **COM_HVPSU2D_LoadCurrentConfig** fails.

For more details, see function **COM_HVPSU2D_SaveCurrentConfig**.

The direct command (j*NN*↵) contains two hexadecimal digits (*NN*) for the variable ConfigNumber. If the command is executed successfully, the device responds by repeating the command characters.

Function COM_HVPSU2D_GetConfigName

```
int COM_HVPSU2D_GetConfigName
  (WORD PortNumber, unsigned ConfigNumber,
  char Name [COM_HVPSU2D_CONFIG_NAME_SIZE]);
```

Command: *ANN*⏎

Response: *ANNS..S*⏎

Saves the name of the configuration in the NVM with the number given by the variable `ConfigNumber` in the variable `Name` and returns an error code according to Tab. 10.

For the configuration number in the variable `ConfigNumber`, see function **`COM_HVPSU2D_SaveCurrentConfig`**.

The variable `Name` passed to the function must be created before calling the function. Its size must match the value of the constant `COM_HVPSU2D_CONFIG_NAME_SIZE` (see the declaration file `COM-HVPSU2D.h`).

Note that the return value is a copy of the data from the NVM, thus there is no guarantee that the value is a null-terminated character string. To generate this string format, allocate the variable `Name` with one additional character and set the value of the last character of the array to 0 after calling the function **`COM_HVPSU2D_GetConfigName`**.

The call to the function **`COM_HVPSU2D_SaveCurrentConfig`** does not modify the configuration name, this is only possible with the function **`COM_HVPSU2D_SetConfigName`**.

The direct command (*A`NN`*⏎) contains two hexadecimal digits (*NN*) for the variable `ConfigNumber`. If the command is executed successfully, the device response (*ANNS..S*⏎) has the same beginning as the command and additionally returns `2*COM_HVPSU2D_CONFIG_NAME_SIZE` hexadecimal digits (*S..S*) for the variable `Name`. Note that in this way, the variable `Name` is transferred in a binary format and thus can be used to store any general character, not only ASCII ones.

Function COM_HVPSU2D_SetConfigName

```
int COM_HVPSU2D_SetConfigName
  (WORD PortNumber, unsigned ConfigNumber,
  const char
  Name [COM_HVPSU2D_CONFIG_NAME_SIZE]);
```

Command: ***ANNS..S*⏎**

Response: ***ANNS..S*⏎**

Sets the name of the configuration in the NVM with the number given by the variable `ConfigNumber` to the value given by the variable `Name` and returns an error code according to Tab. 10.

For more details, see function **`COM_HVPSU2D_GetConfigName`**.

The direct command (*ANNS..S*⏎) contains two hexadecimal digits (*NN*) for the variable `ConfigNumber` and `2*COM_HVPSU2D_CONFIG_NAME_SIZE` hexadecimal digits (*S..S*) for the variable `Name`. If the command is executed successfully, the device responds by repeating the command characters.

Function COM_HVPSU2D_GetConfigFlags

```
int COM_HVPSU2D_GetConfigFlags
  (WORD PortNumber, unsigned ConfigNumber,
  bool & Active, bool & Valid);
```

Command: ***XNN*⏎**

Response: ***XNNF*⏎**

Saves the flags of the configuration in the NVM with the number given by the variable `ConfigNumber` in the variables `Active` and `Valid`, and returns an error code according to Tab. 10.

For the configuration number in the variable `ConfigNumber`, see function **`COM_HVPSU2D_SaveCurrentConfig`**.

The variables `Active` and `Valid` define the properties of a particular configuration in the NVM. The value `Valid` defines whether the configuration is handled as a valid one. If the value is reset, the configuration is considered to be empty.

The value `Active` determines if the configuration is active (value set) or deleted (value reset). When this flag is flipped by calling the function **`COM_HVPSU2D_SetConfigFlags`**, the particular configuration can be deleted or undeleted.

Note that only a valid active configuration can be loaded successfully by the function **`COM_HVPSU2D_LoadCurrentConfig`**.

The direct command (`X`*NN*↵) contains two hexadecimal digits (*NN*) for the variable `ConfigNumber`. If the command is executed successfully, the device response (`X`*NNF*↵) has the same beginning as the command and additionally returns one hexadecimal digit (*F*) for the configuration flags. The configuration flags are a bit array, the least significant bit (bit 0) corresponds to the variable `Active` and the next bit (bit 1) to the variable `Valid`. This means that an empty configuration returns a flag value of 0, an active one 3, and a deleted one a value of 2.

Function COM_HVPSU2D_SetConfigFlags

```
int COM_HVPSU2D_SetConfigFlags
  (WORD PortNumber, unsigned ConfigNumber,
  bool Active, bool Valid);
```

Command: X*NNF*↵

Response: X*NNF*↵

Sets the flags of the configuration in the NVM with the number given by the variable `ConfigNumber` to the values given by the variables `Active` and `Valid`, and returns an error code according to Tab. 10.

For more details, see function **`COM_HVPSU2D_GetConfigFlags`**.

The direct command (`X`*NNF*↵) contains two hexadecimal digits (*NN*) for the variable `ConfigNumber` and one hexadecimal digit (*F*) for the configuration flags (for more details, see function **`COM_HVPSU2D_GetConfigFlags`**). If the command is executed successfully, the device responds by repeating the command characters.

Function COM_HVPSU2D_GetConfigList

```
int COM_HVPSU2D_GetConfigList
  (WORD PortNumber,
  bool Active [COM_HVPSU2D_MAX_CONFIG],
  bool Valid [COM_HVPSU2D_MAX_CONFIG]);
```

Command: W⏎

Response: W*F*..*F*⏎

Saves the flags of all configurations in the NVM in the variables `Active` and `Valid`, and returns an error code according to Tab. 10.

The variables `Active` and `Valid` passed to the function are Boolean arrays that must be created before calling the function. Their size must match the value of the constant `COM_HVPSU2D_CONFIG_NAME_SIZE` (see the declaration file `COM-HVPSU2D.h`).

For the detailed description of the values `Active` and `Valid`, see function **COM_HVPSU2D_GetConfigFlags**.

The function **COM_HVPSU2D_GetConfigList** can be used to obtain a list of valid configurations without the necessity of periodically calling the function **COM_HVPSU2D_GetConfigFlags**. This would take a long time, especially with devices using a USB interface with a long latency.

The response to the direct command (`W⏎`) contains `[(COM_HVPSU2D_MAX_CONFIG*2)/4]` hexadecimal digits (*F*..*F*), where the brackets `[]` denote the integer part of the enclosed argument. The received digits are coded as bit arrays. Analogously to the flag coding for the function **COM_HVPSU2D_GetConfigFlags**, the least significant bit (bit 0) of the first digit corresponds to the variable `Active` and the next bit (bit 1) to the variable `Valid` of the first configuration (i.e. `ConfigNumber = 0`). The following bits (bit 2 and bit 3) of the digit correspond to the variables `Active` and `Valid`, respectively, of the second configuration (i.e. `ConfigNumber = 1`). The next hexadecimal digit is composed in the same way and contains the data of the next configuration pair (i.e. `ConfigNumber = 2` and `ConfigNumber = 3`), etc.

Miscellaneous Functions

Function COM_HVPSU2D_GetSWVersion

`WORD COM_HVPSU2D_GetSWVersion();`

Returns the version of the software interface (the dynamic link library `COM-HVPSU2D.dll`).

This function should be used to check whether a software interface with the correct version is being used. The function should be called prior to any other function of the software interface. It does not have any influence on the communication and can be called at any time.

The return value is an unsigned 16-bit integer (WORD). The higher byte contains the main version number, the lower byte the subversion, i.e. the version order within the main version. If the version numbers of a library `COM-HVPSU2D.dll` are different, check the change list provided by the manufacturer. There is no guarantee that a library `COM-HVPSU2D.dll` with different version numbers can be used without any changes. In most cases, the user software has to be recompiled or modified to match the new definition of the software interface.

Function COM_HVPSU2D_GetHWType

`int COM_HVPSU2D_GetHWType (WORD PortNumber, DWORD & HWType);`

`Command: t↵`

`Response: t`*`HHHHHH`*`↵`

Saves the device's hardware version in the variable `HWType` and returns an error code according to Tab. 10.

The return value in the variable `HWType` can be used to identify the hardware. i.e. to ensure that the connected device is the desired type. Please contact the manufacturer for further details.

The response to the direct command (`t↵`) contains 6 hexadecimal digits (*HHHHHH*) for the variable `HWType`.

Function COM_HVPSU2D_GetHWVersion

```
int COM_HVPSU2D_GetHWVersion
  (WORD PortNumber, WORD & Version);
```

Command: `v⏎`

Response: `v`*VVVV*`⏎`

Saves the device's hardware version in the variable `Version` and returns an error code according to Tab. 10.

The return value in the variable `Version` should be used to check whether the hardware is an appropriate version.

The return value is similar to the return value of the function **`COM_HVPSU2D_GetSWVersion`**. It is an unsigned 16-bit integer (WORD) containing the main version and the subversion numbers. Check the change list provided by the manufacturer to learn whether the software interface is compatible with the hardware.

The response to the direct command (`v⏎`) contains 4 hexadecimal digits (*VVVV*) for the variable `Version`.

Function COM_HVPSU2D_GetFWVersion

```
int COM_HVPSU2D_GetFWVersion
  (WORD PortNumber, WORD & Version);
```

Command: `V⏎`

Response: `V`*vvvv*`⏎`

Saves the device's firmware version in the variable `Version` and returns an error code according to Tab. 10.

The return value in the variable `Version` should be used to check whether the firmware is an appropriate version.

The return value is similar to the return value of the function **`COM_HVPSU2D_GetSWVersion`**. It is an unsigned 16-bit integer (WORD) containing the main version and the subversion numbers. Check the change list provided by the manufacturer to learn whether the software interface is compatible with the firmware.

The response to the direct command (`V↵`) contains 4 hexadecimal digits (*vvvv*) for the variable `Version`.

Function COM_HVPSU2D_GetFWDate

```
int COM_HVPSU2D_GetFWDate (WORD PortNumber,
  char * DateString);
```

Command: D↵

Response: D*dd..d*↵

Saves the device's firmware date in the variable `DateString` and returns an error code according to Tab. 10.

The return value in the variable `DateString` is a null-terminated character string with the firmware compilation date. The buffer passed to the function must be created before the function call, it must be at least 16 bytes long.

The response to the direct command (`D↵`) usually contains 11 characters (`dd..d`) for the variable `DateString`.

Function COM_HVPSU2D_GetProductNo

```
int COM_HVPSU2D_GetProductNo
  (WORD PortNumber, DWORD & Number);
```

Command: N↵

Response: N*nnnnnnnn*↵

Saves the device's product number in the variable `Number` and returns an error code according to Tab. 10.

The response to the direct command (`N↵`) contains 8 hexadecimal digits (*nnnnnnnn*) for the variable `Number`.

Function COM_HVPSU2D_GetProductID

```
int COM_HVPSU2D_GetProductID
  (WORD PortNumber, char * Identification);
```

Command: P↵

Response: P*ii..i*↵

Saves the device's product identification in the variable `Identification` and returns an error code according to Tab. 10.

The return value in the variable `Identification` is a null-terminated character string with the product identification. The buffer passed to the function must be created before the function call; it should be 60 characters long.

The response to the direct command (`P↵`) contains a variable number of characters (*ii..i*) for the variable `Identification`. Note that the termination character of the string is not transmitted.

Function COM_HVPSU2D_GetUptime

```
int COM_HVPSU2D_GetUptime (WORD PortNumber,
  DWORD & Seconds, WORD & Milliseconds,
  DWORD & Optime);
```

Command: U↵

Response: U*SSSSSSSSMMOOOOOOOO*↵

Saves the device uptime in the variables `Seconds` and `Milliseconds`, and the operating time in the variable `Optime`. The function return value is an error code according to Tab. 10.

The device uptime is the time elapsed from the last (re)start of the device, the operating time is the portion of the uptime in which the device was activated. The function **COM_HVPSU2D_GetUptime** can be used, for instance, to see whether the device has restarted unexpectedly or to check how long it has been operating.

The response to the direct command (`U↵`) contains 8 hexadecimal digits (*SSSSSSSS*) for the variable `Seconds`, 2 hexadecimal digits (*MM*) for the variable `Milliseconds`, and 8 hexadecimal digits (*OOOOOOOO*) for the variable `Optime`. To convert the value of the variable `Milliseconds` into the abovementioned value, i.e. ms, it has to be multiplied by 3.90625, i.e. divided by 256=100h and multiplied by 1000.

Function COM_HVPSU2D_GetTotalTime

```
int COM_HVPSU2D_GetTotalTime
  (WORD PortNumber, DWORD & Uptime,
  DWORD & Optime);
```

Command: `u↵`

Response: `u`*UUUUUUUUOOOOOOOO*`↵`

Saves the total uptime of the device in the variable `Uptime` and the total operating time in the variable `Optime`. The function return value is an error code according to Tab. 10.

The total uptime and the total operating time are the sum of all uptimes and operating times, respectively (see function **`COM_HVPSU2D_GetUptime`**), since the device has been manufactured.

The response to the direct command (`u↵`) contains 8 hexadecimal digits (*UUUUUUUU*) for the variable `Uptime` and 8 hexadecimal digits (*OOOOOOOO*) for the variable `Optime`.

Function COM_HVPSU2D_GetCPUData

```
int COM_HVPSU2D_GetCPUData (WORD PortNumber,
  double & Load, double & Frequency);
```

Command: `C↵`

Response: `C`*LLLFFFF*`↵`

Saves the load of the device's CPU in the variable `Load` and its operating frequency in the variable `Frequency`. The function return value is an error code according to Tab. 10.

The CPU load is a value between 0 (= 0%) and 1 (= 100%). Under normal conditions, the CPU load should not exceed 10%. Large data transfers or controlling many fans may increase the load to higher values.

The CPU operating frequency is a value stabilized by a frequency locking loop to about 15.7 MHz. The frequency is also used as a time base for the communication interface. The theoretical frequency value equals 15.6672 MHz, which is a multiple of the maximum communication speed of 230.4 kbaud (see the function **COM_HVPSU2D_SetBaudRate**).

The response to the direct command (C↵) contains 3 hexadecimal digits (*LLL*) for the variable Load and 4 hexadecimal digits (*FFFF*) for the variable Frequency. To convert the values into the abovementioned values, the variable Load has to be divided by 1000 and the variable Frequency multiplied by 1024.

Function COM_HVPSU2D_Restart

int COM_HVPSU2D_Restart (WORD PortNumber);

Command: #↵

Response: #↵

Restarts the device and returns an error code according to Tab. 10.

The function issues a reboot of the device's CPU and waits until the device responds again after the restart. This may take several seconds.

Note that after the restart, the communication speed is restored to the default value of 9600 baud. If a higher communication speed should be used, the function **COM_HVPSU2D_SetBaudRate** must be called again.

The response to the direct command (#↵) occurs at the currently selected data rate. After the response has been received, the data rate must be restored in the host UART to the default value of 9600 baud to be able to communicate with the device.

Error Handling

Function COM_HVPSU2D_GetInterfaceState

```
int COM_HVPSU2D_GetInterfaceState
  (WORD PortNumber);
```

Returns the state of the software interface according to Tab. 10.

This function can be used to obtain the last return value of an interface function. It does not have any influence on the communication and can be called at any time.

Function COM_HVPSU2D_GetErrorMessage

```
const char * COM_HVPSU2D_GetErrorMessage
  (WORD PortNumber);
```

Returns the error message corresponding to the state of the software interface (see function **COM_HVPSU2D_GetInterfaceState**). The return value is a pointer to a null-terminated character string according to Tab. 10.

This function does not have any influence on the communication and can be called at any time.

Function COM_HVPSU2D_GetIOState

```
int COM_HVPSU2D_GetIOState (WORD PortNumber);
```

Returns the interface state of the serial port according to Tab. 11.

This function can be used to obtain the result of the last I/O operation at the port. It does not have any influence on the communication and can be called at any time.

Function COM_HVPSU2D_GetIOErrorMessage

```
const char * COM_HVPSU2D_GetIOErrorMessage
  (WORD PortNumber);
```

Returns the error message corresponding to the interface state of the serial port (see function **COM_HVPSU2D_GetIOState**). The return value is a pointer to a null-terminated character string according to Tab. 11.

This function does not have any influence on the communication and can be called at any time.