

HiTek Power Ltd  
Hawthorn Road  
Littlehampton  
West Sussex  
BN17 7LT  
United Kingdom

Tel:+44 (0) 1903 712400  
Fax: +44 (0) 1903 712500  
[hvsales@aei.com](mailto:hvsales@aei.com)  
[www.advanced–energy.com](http://www.advanced–energy.com)/High\_Voltage

Company Reg. No. 908344

This document shall not be copied, reproduced or made available in any form or for any purpose, other than for which it is supplied, without the prior written consent of HiTek Power Ltd.

Standard Protocol for HV Power supplies

Protocol Revision ‘2’

INTENTIONALLY BLANK

Contents

[1 Introduction 5](#_Toc482782317)

[2 Terminology 6](#_Toc482782318)

[3 Message Format 7](#_Toc482782319)

[4 Message Groups 8](#_Toc482782320)

[5 PSU Messages 9](#_Toc482782321)

[6 MODULE Messages 11](#_Toc482782322)

[7 OUTPUT Messages 12](#_Toc482782323)

[8 Examples 14](#_Toc482782324)

[9 Output state, faults, masks and trips 15](#_Toc482782325)

[10 Output calibration 17](#_Toc482782326)

[11 Not defined in this base protocol 19](#_Toc482782327)

[12 Protocol Usage Guidelines 19](#_Toc482782328)

[13 Protocol implementation guide 22](#_Toc482782329)

[14 APPENDIX 29](#_Toc482782330)

[Document History 31](#_Toc482782331)

INTENTIONALLY BLANK

# Introduction

## General

This document describes the AE communication protocol for HV Power Supplies.

For latest GUI software downloads and protocol Software Development Kit (SDK) visit  
<http://hitekpowersoftware.com/>

Messages in this document are supported by all PSUs. Other messages are included in the respective PSU specification.

The protocol operates over a stream of data bytes such as RS232 or TCP/IP.

## Disclaimer

Important usage instructions are provided here and must be understood before a quote is accepted.

“You must get feedback from the person or company designing your control software”

A change after the quote has been accepted can incur additional cost and delay.

# Terminology

## PSU

The PSU is the ‘HV Power supply unit’.  
It is called ‘*system*’ in the case of ‘*system* status’ or ‘*system* type’ to avoid confusion with ‘output status’.  


## Module

A functional block within the PSU that has a microcontroller, for example ‘grounded’, ‘gun’, ‘wafer’ or ‘floating’.

## Output

An independently controllable output, for example ‘beam’, ‘extractor’, ‘suppressor’ or filament’.

## Controller

A separate workstation or PLC running a software application. It is responsible for monitoring and controlling the PSU and other subsystems  


## Fault

A fault is active when an operating parameter moves outside a specified limit.

## Trip

A trip is the automatic and immediate shutdown of one or more outputs in response to a fault. The output may be described as ‘tripped’

## Mask

A mask may be used to prevent a trip. A mask will not prevent a fault.

# Message Format

## Request

A request is a single line of ASCII text that takes one of the three forms:

* A request to change the value of a named parameter.  
  This consists of a parameter name, an ‘=’ (equals) character, and a value.
* A request to retrieve the value of a named parameter.  
  This consists of a parameter name, and a ‘?’ (Question mark) character.
* A request to perform some named operation.  
  This consists of an operation name, and a ‘!’ (Exclamation mark) character.

The values supplied when requesting a change to a parameter are real-world values expressed in SI units. For example:

|  |  |
| --- | --- |
| ***Request*** | ***Notes*** |
| VDEM=1000 | Requests the voltage demand to be set to 1000V |
| VDEM? | Requests the voltage demand value |
| CLEAR! | A request to clear all faults |
|  |  |

## Response

A response is a single line of ASCII text that takes one of the three forms:

* A response containing the value of a parameter.  
  This will contain the name of the parameter, a ‘:’ (colon) character, and the value of the parameter.
* A response indicating that an operation completed correctly.  
  This contains the name of the parameter or operation, and a ‘$’ (dollar) character.
* A response indicating that there was an error performing the operation.  
  This contains the name of the parameter or operation, a ‘\*’ (star, or asterisk) character, and an error value.

In all cases, the name of the parameter or operation will be the same as that given in the matching request. For example:

|  |  |  |
| --- | --- | --- |
| ***Request*** | ***Response*** | ***Notes*** |
| B.VDEM=1000 | VDEM$ | Requests the voltage demand to be set to 1000V, and a response indicating that the request was processed correctly. |
| B.VDEM? | VDEM:1000 | Requests the voltage demand value, and a response indicating a value of 1000V. |
| B.IMON? | IMON:0.001 | Requests the current monitor value, and a response indicating a value of 0.001A, or 1mA. |
| B.IMON=0 | IMON\*READONLY | A request to set the current monitor, and an error response indicating that the current monitor value is read-only. |
| RESET! | RESET$ | A request to reset all outputs to a known state. |
|  |  |  |

# Message Groups

Messages are grouped into 3 categories

* PSU
* Module (Module identifier prefix may be required)
* Output (Output identifier prefix may be required)



## Module Identifier

If the PSU has more than one module then prefix the message with a ‘module identifier’.  
For example 'Grounded' or 'Floating Deck'

|  |  |
| --- | --- |
| **GND.**SWVER? | **FD.**SWVER? |
| **GND.**TEMP? | **FD.**TEMP? |

## Output Identifier

If the PSU has more than one output then prefix the message with an ‘output identifier’.  
For example 'Beam' or ‘filament’

|  |  |
| --- | --- |
| **B.**STA? | **F.**STA? |
| **B.**EN=1 | **F.**EN=1 |

# PSU Messages

A collection of messages for the PSU as a whole, for example:  
 Serial number, system type and system status

For definition of ‘PSU’, see Section 2.1

## Control (Write Only)

|  |  |  |
| --- | --- | --- |
| ***Name*** | ***Parameter*** | ***Notes*** |
| RESET! | - | Reset all outputs to their default state (off & demands to zero) |
| CLEAR! | - | Clear all latched output fault flags |
| RESTART! | - | Restart / reboot all microcontrollers |

## Monitoring (Read Only)

|  |  |  |
| --- | --- | --- |
| ***Name*** | ***Parameter*** | ***Notes*** |
| STAT? | System Status [*flags*] | See ‘System Status’ in separate PSU specification. The response is different for each PSU model. Flags must include:   * Interlock * Enabled (One flag for each output) * HVON (On flag for each output) * Fault (Logical OR of all fault flags) * Output status (ONLY PSUs with one output)   Note: Multiple output PSUs have a separate output status message. Refer to the PSU specification. |
| PASSWORD? | Operating Mode *[string]* | Set the operating mode by password. Modes are:  *Normal* Standard user mode. Power on default.  *Engineering* Some trips can be masked. Serial numbers and calibration can be updated.  *Design* As Engineering mode. All trips including interlocks can be masked out |

## Information (Read Only)

|  |  |  |
| --- | --- | --- |
| ***Name*** | ***Parameter*** | ***Notes*** |
| SYSTYPE? | System Type *[string]* | PSU model. Revision may be added to the end with a ‘.REV’ separator  <*product & variant*>.REV<*revision*>  Example: EG353-02.REV1 XRG70-903-17.REV2 |
| PROTOCOL? | Version *[uint16]* | Version of the protocol implemented by the PSU. See document title |
| SERIAL? | Serial Number [*uint32*] | Unit serial number. Also printed on the outside of the PSU |

# MODULE Messages

A collection of messages for an individual module, for example:  
 Temperature, software version, input supply monitoring and fans.

A PSU has one or more modules.

For definition ‘module’, see Section 2.2

If a PSU has more than one module then prefix the message with a ‘module identifier’.

To get a list of ‘module identifiers’ use the ‘MODULES?’ command.

## Control Parameters (Write Only)

|  |  |  |
| --- | --- | --- |
| ***Name*** | ***Parameter*** | ***Notes*** |
| - | - | - |

## Monitoring Parameters (Read Only)

|  |  |  |
| --- | --- | --- |
| ***Name*** | ***Parameter*** | ***Notes*** |
| - | - | - |

## Information Parameters (Read Only)

|  |  |  |
| --- | --- | --- |
| ***Name*** | ***Parameter*** | ***Notes*** |
| SWVER? | Software Version *[uint16]* | Reference to the source code management system |

# OUTPUT Messages

A collection of messages for a specific output, for example:   
 Demand, monitor and output status

For definition of ‘output, see Section 2.3

If a PSU has more than one output then prefix the message with an ‘output identifier’

To get a list of ‘output identifiers’ use the ‘OUTPUTS?’ command.

## Control Parameters (Read / Write)

|  |  |  |
| --- | --- | --- |
| ***Name*** | ***Parameter*** | ***Notes*** |
| EN | Enable *[bool]* | 0=Output Disable, 1=Output Enable |
| VD | Voltage Demand *[float] in Volts* | Controller set point |
| VS | Voltage Slew *[float] Volts/Second* | Controller set point |
| ID | Current Demand *[float] Volts/Second* | Desired set point |
| IS | Current Slew *[float] Volts/Second* | Ramp rate up and down. Desired set point |
| WD | Wobbler Depth  *[float] Factor* | Sinusoidal component added to output (set point) 0=off, 0.5=50% of Demand, 1.0=100% of Demand |
| WF | Wobbler Frequency  *[float] Hz* | Frequency of sinusoidal component |
| MASK  (or TRIP in the case of EG353) | Trip Mask *[flags]* | Faults can be ‘Masked off’ to prevent the output from entering the ‘tripped’ state. For definition of ‘fault’ and ‘trip’, see Section 2.5 & 2.6  If a fault bit is set and the corresponding MASK bit is CLEAR, the output will NOT trip If a fault bit is set and the corresponding MASK bit is SET, the output will trip For a list if ‘Mask flags’, see Section 6.3.2 |

## Control Parameters (Write Only)

|  |  |  |
| --- | --- | --- |
| ***Name*** | ***Parameter*** | ***Notes*** |
| CLEAR! | Clear Faults | Clear latched fault flags (FLT). Will not clear active faults. |

## Monitoring Parameters (Read Only)

|  |  |  |
| --- | --- | --- |
| ***Name*** | ***Parameter*** | ***Notes*** |
| ST? | Output Status *[flags]* | See ‘Output Status’ in separate PSU specification. The response is different for each PSU model.  The following flags must be implemented: See Section 6.3.1 ‘Output Status Flags’ |
| FLT? | Fault *[flags]* | See section 6.3.2 ‘Output Fault & Mask Flags’ |
| VA? | Voltage Actual *[float] Volts* | Actual demand with slew and wobble |
| VM? | Voltage Monitor *[float] Volts* | Measured on the output |
| IA? | Current Actual *[float] Amps* | Actual demand with slew |
| IM? | Current Monitor *[float] Amps* | Measured on the output |

### Output Status Flags

|  |  |  |
| --- | --- | --- |
| *Bit* | *Purpose* | *Description* |
| 0 | Enabled | Corresponds to the EN command. There is a copy of this flag in the ‘system status’ message STA? Cleared on power-up or RESET! |
| 1 | Powered | Output is generating voltage (Active). There is a copy of this flag in the ‘system status’ message STA? |
| 4 | Ramp | Ramping in progress |
| 5 | Wobble | Wobble active |
| 12 | Reserved | Reserved |
| 13 | Fault | Fault is active on this output |

### Output Fault & Mask Flags

|  |  |  |  |
| --- | --- | --- | --- |
| *Bit* | *Purpose* | | *Description* |
| 0 | Interlock | Set if the interlock is opened | |
| 4 | Input supply fault | Set if the input supply is outside 10% of the nominal 24V. | |
| 5 | Internal | Software error / system communications error | |
| 8 | Temperature | Set if the internal ambient exceeds specification | |
| 12 | Over Current | Over current condition | |
| 13 | Over Voltage | Over voltage condition | |

## Information Parameters (Read Only)

|  |  |  |
| --- | --- | --- |
| ***Name*** | ***Parameter*** | ***Notes*** |
| VMAX? | Max Voltage *[float] Volts* | Demand for voltage control output Can be negative, often the greatest magnitude |
| VMIN? | Min Voltage *[float] Volts* | Demand for voltage control output Often 0V |
| IMAX? | Max Current *[float] Amps* | Demand for current control output Can be negative, often the greatest magnitude |
| IMIN? | Min Current *[float] Amps* | Demand for current control output Often 0A |

# Examples

## Single output PSU

This example has one module and one output.

SYSTYPE?

STATUS?

SWVER?

## Multi output PSU

This example has two modules (grounded and floating deck) and 4 outputs (Beam, suppressor, extractor and filament).

SYSTYPE?

STATUS?

GND.SWVER?

FD.SWVER?

# Output state, faults, masks and trips

At any given time, each PSU output is in one of three states: ‘Off’, ‘On’ or ‘Tripped’. Transitions between states occur under the conditions shown below.



## Turning the output on

Set the demand  
VD=-1000

Enable the output  
EN=1

Notes

* The controller can only enable the output if all unmasked fault flags are clear
* The PSU will set HVON flag in the status message when the output voltage exceeds 50V
* If the demand is not set by the controller then the last set point will be used.

## Output Trip

A trip is defined as the automatic and immediate shutdown of one or more outputs in response to a fault condition

When the output is in the ‘OFF state’ the over current, over voltage and arcs fault flags will be disabled. If they are required, enable the output EN=1 with the demand set to zero.

The over-current and loop-error fault flags are disabled during ramp and wobble

## Clearing a trip

There are two ways to exit the trip state:

### Reset all outputs

Send a ‘RESET!’ command. This will set all read/writable parameters to power on default values.

### Clear fault and disable

Clear the latch fault flags  
CLEAR!

Disable the output  
EN=0

Notes

* The controller can only clear the fault flags if the fault is no longer active
* The controller can only disable the output if all unmasked fault flags are clear

## FAULT and MASK flags

The flags are grouped into registers called FAULT and MASK. Each fault condition that can occur has a corresponding bit defined in Section 6.3.2

Each bit in the FAULT register represents a latch that is set when the corresponding fault condition is present. Bits in the Fault register are cleared by the controlling system using a “CLEAR!”, “RESET!” or “RESTART!” command. Bits in the FAULT register cannot be cleared while the corresponding fault is still active

If any bit is set in the FAULT register, *and* the corresponding bit is set in the MASK register, *and* the output is on, *then* the output will switch to the tripped state. See the ‘MASK’ message in Section 6.1 for more details.  
This example shows two active faults. One is ‘masked off’ and the other will cause a trip.

|  |  |
| --- | --- |
| FAULT | 1100 |
| MASK | 0110 |
| ‘TRIPPED’ if not equal to zero | 0100 |

## Automatic actions

The PSU will only perform one automatic action, a trip. All other actions MUST be explicitly requested by the controller.

### Outputs operate independently

A condition on one output, fault or otherwise, will have no effect on other outputs. This behaviour is deemed specific to the application and is therefore the responsibility of the controller.

### Module Faults

A fault that affects a module is applied to all outputs managed by that module. For example, temperature and ‘input supply voltage out of range’.

### PSU Fault

A fault that affects the PSU is applied to all outputs, For example, interlock open.

**IMPORTANT**: Consider actions required to protect the load. Alternative shutdown sequences must be handled by the controller using this protocol. See Section 10.4 to verify the protocol meets real time requirements.

# Not defined in this base protocol

These messages are not included in the base protocol. If required they are included in the PSU specification.

* Control loop error voltage direct read (Useful for potted outputs)
* Large number of parameters required for digital control loop configuration / monitoring
* PWM sync control to manage transformer efficiency (if required) – Phase, Frequency, lock status, fixed or automatic
* Beam blank related functions
* Monitoring and control of aux voltage regulators
* Power (in WATTS) for demands / monitors
* Local or remote interface selection. Some products can be switched between analogue interface, digital interface and front panel
* Range selection
* Master / slave configuration related
* Warning flags
* User settable warning limits
* User settable fault limits
* Arc fault conditions – conditions under which an for arc
* Arc disabling
* Arc detect data capture – Fast (timestamps) and slow (ADC measurements)
* Logging related – What is logged and how to extract log data
* Fan control, monitoring and faults such as not connected / stall
* Communication settings – Baud rates / Network settings / MAC address / IP Address / DHCP settings / RS485 address settings
* Notification are not supported

# Protocol Usage Guidelines

These guidelines help the ‘systems integrator’ and ‘controller software architect’ understand how to communicate with the PSU. It is important to understand the boundary between functions carried out by the PSU and those functions that need to be implemented by the controller. For example, consider the following:

*This is a PSU function:*  
Automatically shut down associated outputs if the temperature exceeds specified range

*This is a ‘controller’ function:*  
Shutdown all outputs in a predefined order and at predefined ramp rate before the maximum temperature is exceeded

Note: In this example the temperature is used by the controller to take action *before* an automatic shutdown.

## Before starting

Use the provided ‘CmdTerm’ tool to perform common control and monitoring functions. This tool displays raw ASCII messages and converts them into human readable format.

You should emulate the messages generated by your controller software with the load connected. This will help you visualise and design a typical protocol exchange for your application. It will also help identify problems and risks early on.

## Operation sequencing

The protocol does not make any allowance for overlapping operations. This simplifies the protocol, and implies that all operations are performed in strict sequence.

In principle, all operations *are* performed strictly in the order that the requests are issued. However, the internal operation of the unit means that there may be some limited re-ordering.

* A series of operations on a single output will only be re-ordered in a benign fashion. For example, if the demand is changed and the output then switched on, these will never be re-ordered so that the output is switched on at the previous demand level.
* Operations on two different outputs *may* be re-ordered due to internal scheduling delays. If the ordering is important then poll the relevant status and wait for the first operation to complete.
* Read-only parameters are updated once the action has completed. For example, enabling an output will set the 'enabled' flag in the output status register once the output has turned on, not when the message is received.

## Real-time messaging constraints

Consider the real-time constraints of PSU monitoring and control using this protocol. Check you are able to poll the PSU and take necessary action to meet hard deadlines required by your application.

* @115200 baud a single character will take 78µs to transmit
* Time between receiving the last byte of a command and transmitting the first byte of the response is less than 300µs

Additional timing related constraints cannot be added after a quote has been accepted.

## Polling cycle

The controller must be in constant communication with the PSU. Consider the typical polling cycle that reads the system status flags and output monitors. After this a further polling of output status and fault flags may be required. The must also be a way to set demands and monitor actual voltages.

Consider using a library that manages this polling cycle for you and presents the controller with an event driven interface. Use call-backs for notification of status changes, faults and arcs. Pass relevant information with the call-backs to the controller. Use shadow registers to provide the controller instant access to monitor readings and demands.

## Message flow control

### Single request / response (recommended)

After sending one request, the controller waits for a response before sending the next request.

### Queued requests

The maximum number of queued requests depends on the length of those requests. The minimum buffer size is 80 characters which includes any line terminators. The order and timing of the PSU response is not guaranteed.

## Read / Write parameter value

A write to a read/write parameter will be rejected with an error message if the value is invalid or out of range.

A read of a read/write parameter will return the last accepted value. The read-back value is unaffected by the internal state of the PSU.

For example:

* Demands (.VD or .ID) are set by the external controller. The read back value is never changed by internal events. To be explicit, an interlock or trip will **not** change the demand set point. To determine the present value use the read-only ‘actual voltage demand’ (.VA) or ‘actual current demand’ (.IA) messages.
* Enable control (.EN) is set by the external controller. The read back value is never changed by internal events. To be explicit, an interlock or trip will **not** change the Enable control. To determine the present state use the read-only ‘Enabled flag’ in the status message. (STA)

## Stored Values / Default values

Only serial number, calibration and communication settings are stored in the PSU after a reset or power cycle. All other parameters are initialised to fixed ‘power on default values’.

If a ‘power on default value’ is not defined in the PSU specification it will be fixed by HiTek Power. The control software must explicitly set a different value if required.

## Logging

This is for performance monitoring and fault finding only. The controller may collect, store and display this data. The controller must not use this data for PSU monitoring and control. Do not use as an alternative to polling status, fault flags or arc records at runtime.

## Arc detector

The PSU will implement arc detection to protect the PSU output only. The detect algorithm runs independently on each output and only affects the fault flag of that output.

See the PSU specification if specialist arc detection is required to protect the load. The PSU will not process data specific to the dynamics of the connected load. When provided raw information about arc events, voltages and current must be processed by the controller using the messages provided in the PSU specification. If this data is not collected in a timely manner it will be overwritten.

## Interlocks

Hardware interlocks are provided and must be used. Do not use a parallel interlock system to disconnect the input power supply.

## Filament standby current

The filament standby function must be used if provided. Do not provide the filament standby current from an external source.

## Extended operating ranges

The maximum and minimum limits for many operating parameters are found in the PSU specification. For example: temperature, demands, over current and over voltage. The limits are enforced by a protocol ‘out of range’ error or a fault.

The PSU can only operate outside these limits in engineering mode. The PSU specification must list the limits affected by engineering mode.

# Protocol implementation guide

This section is for design engineers wishing to implement the AE protocol in a PSU.

## Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are used in the same way as in Internet standards documents. The meanings are described in [RFC 2119 (http://www.ietf.org/rfc/rfc2119.txt)](http://www.ietf.org/rfc/rfc2119.txt).

## Design principles

The control protocol is intended to be used for two distinct purposes.

Most importantly, it will be used as the interface to a controlling computer system for the purpose of system integration. To this end, it must be possible to rapidly process large numbers of operations in a well-defined manner.

In addition, it is targeted at a range of system sizes. A powerful microcontroller cannot be assumed to be present.

* It should be easy and fast to parse and generate. Secondly, it can be used as a simple user interface. This allows technical personnel to manipulate the system.
* It should be intuitive to use.
* Correct operation of the protocol should not impose timing constraints that prevent data input from a keyboard.
* It should be possible to connect to the protocol using simple, widely available software tools

## Data link

This specification does not define the type of data link that can be used with this protocol. Instead, it defines a minimum set of characteristics that the link must have.

### Characteristics

The link must be able to transfer printable ASCII characters, with minimal corruption. In this context, a printable ASCII character is any with a hexadecimal value of 0A, 0D, or 20 to 7E.

The protocol includes a simple error detection mechanism, so occasional corruption of the data is acceptable. However, in normal operation, the link should not remove, insert or re-order any characters.

### Other link types

The protocol could be easily adapted to operate on packet-based data links. In that case, the natural packet boundaries are used to delimit the ‘lines’ of data, instead of the line-end characters.

## Line structure

The controller and system exchange lines of characters. Each such line comprises a number of printable ASCII characters, terminated by either a carriage-return or line-feed character.

The printable ASCII characters have codes in the range 20 (hex) to 7E (hex). Carriage‑return and line‑feed, otherwise referred to as CR and LF, have the ASCII codes 0D (hex) and 0A (hex) respectively.

The protocol currently defines four types of line.

### Empty lines

Empty lines that are received by either the controller or system must be silently ignored.

*Some systems issue a CR-LF pair as a line end. Since an empty line must be ignored, it is acceptable for an implementation to treat any CR or LF as a line end. If a line has a CR-LF pair as a terminator, the initial CR will cause the line to be interpreted, and the following LF will be treated as the end of an empty line, which will be ignored.*

### Comment lines

Comment lines start with a ‘;’ (semi-colon), and may contain arbitrary textual information. Such lines must be silently ignored.

This feature is provided to allow diagnostic information to be mixed with request or response sequences, while still permitting normal system control.

### Request lines

Request lines start with a parameter name, and are followed by one of the characters ‘=’, ‘!’ or ‘?’. These are requests from the controller to the system to perform some operation.

### Response lines

Response lines start with a parameter name, and are followed by one of the characters ‘:’, ‘$’ or ‘\*’. These are responses from the system to the controller, and indicate the outcome of a previous request.

## Requests and responses

All control of the system is performed by accessing a set of named parameters, and by invoking a set of named operations. Parameter and operation names consist of a sequence of letters, digits, underscores and dots. The first character of a name *must* be a letter or underscore.

The accessible parameters and operations will depend on the type of system in use, although there is a (small) set of parameters that must be defined by *all* systems that implement this protocol.

The set of parameters that is implemented in any given product will be defined as part of the product specification.

### Setting a parameter

|  |  |  |
| --- | --- | --- |
| ***Request*** | *NAME*=*VALUE* | Requests that the system set the value of parameter ‘NAME’ to ‘VALUE’. The form that ‘VALUE’ takes will depend on the type of the parameter. |
| ***Acknowledgement*** | *NAME*$ | This indicates that the parameter was set correctly. |
| ***Error response*** | *NAME*\**REASON* | This indicates that the parameter was not set. The cause of the failure is indicated by ‘REASON’. |

### Getting a parameter value

|  |  |  |
| --- | --- | --- |
| ***Request*** | *NAME*? | Requests the value of parameter ‘NAME’. |
| ***Acknowledgement*** | *NAME*:*VALUE* | This indicates that the value of the parameter is currently ‘VALUE’. |
| ***Error response*** | *NAME*\**REASON* | This indicates that the parameter cannot be retrieved. The cause of the failure is indicated by ‘REASON’. |

### Invoking an operation

|  |  |  |
| --- | --- | --- |
| ***Request*** | *NAME*! | Requests that the system perform operation ‘NAME’. |
| ***Acknowledgement*** | *NAME*$ | This indicates that the operation completed correctly. |
| ***Error response*** | *NAME*\**REASON* | This indicates that the operation failed. The cause of the failure is indicated by ‘REASON’. |

## Naming conventions

Parameters and operations *should* be assigned mnemonic names. That is, names of the form ‘VDEM’, ‘IMON’, ‘CLEAR’ and so on are preferred to ‘P1’, ‘P2’, ‘CMD1’.

Although the distinction may be irrelevant to computer systems, the use of mnemonic names will make the protocol more accessible to operators typing at a terminal, programmers coding control programs, and anyone who needs to examine a communications log.

Parameter and operation names are not case sensitive. Implementations *must* accept parameter names in upper, lower, or mixed case. That is ‘VDEM’, ‘vdem’, ‘Vdem’ and ‘vDEm’ are all equivalent.

Although responses are required to carry the same parameter or operation name as the matching request, the response need not use the same case as the request. Thus, ‘Vdem=1000’ is a legitimate response to any of the requests ‘VDEM?’, ‘vdem?’ or ‘vDEm?’.

Names *should not* have a context-dependent meaning. Typically, this will mean:

* If a parameter can be assigned a value, then getting that parameter will retrieve the most recently assigned value. The retrieved value may be slightly different due to rounding, quantisation and so on.
* An assigned name will normally be either a parameter name *or* an operation name. Occasionally it may make sense for a parameter and a related command to have the same name. An example of this is the name ‘CLEAR’ in the XRG50, which can be a command or a parameter name.

## Parameter values

Parameter values may take a number of different forms, depending on the purpose of the parameter.

### Analogue values

Analogue values typically represent some real-world, physical, parameter. Most parameters will fall in to this category.

Such values are represented using a conventional decimal floating-point format, as used in (at least) the C, C++ and C# languages.

Parameters that represent physical quantities, especially those concerning electrical values *should be* expressed in SI units, unless there are compelling reasons otherwise.

There are a currently two exceptions for temperatures (Celsius) and fan speeds (RPM). These exceptions are permitted since they are in common use and are more readily understood.

|  |  |  |
| --- | --- | --- |
| ***Parameter type*** | ***Unit*** | ***Unit name*** |
| Time | s | Seconds |
| Voltage | V | Volts |
| Current | A | Amps |
| Frequency | Hz | Hertz |
| Temperature | C (not K) | Celsius |
| Rotational speed (for fans) | RPM | Revs-per-minute |

The decimal representation of these values comprises the following:

* An optional plus or minus sign
* A non-empty sequence of decimal digits, optionally containing a decimal point character.
* An optional exponent, which consists of an optional plus or minus and a non-empty sequence of decimal digits.

Implementations *may* accept additional formats. This may be convenient for users who are accessing the protocol by typing at terminal programs, and may simplify the implementation.

However such implementations should only output values that conform to this specification.

***Examples***

|  |  |
| --- | --- |
| VDEM=10000 | Set VDEM (which is perhaps a voltage demand) to 10000, or 10kV. |
| VDEM=10000.0 | As previous, with a decimal fraction. |
| VDEM=1e4 | As previous, in exponent form. |
| VDEM=+1.0e+4 | As previous, with signs. |

### Integer values

Integer values are commonly required where the value represents a count.

As with analogue values, the standard decimal representation should be used; an integer value consists of one or more decimal digits. Integer values may not carry a sign, decimal point or exponent.

Again, an implementation *may* accept additional formats, but should output values that conform strictly to this specification.

### Boolean values

Where a boolean value is required, it *may* be represented with the integer values 0 and 1, provided that the meaning of the two states is clear.

*That is, using 0/1 to represent off/on, false/true, disabled/enabled is acceptable. It would not be appropriate to represent an LED colour with 0/1 meaning red/green!*

### Bit masks and status registers

It can be useful to be able to represent a set of ‘bit’ flags in a single register. Power supplies may have an extensive set of ‘flag’ bits, and these are often presented in the form of a status register.

Such registers are represented as hexadecimal numbers.

The number of digits in a register value need not be related to the number of defined bits. Thus, if a register has only the least significant bit set, it may be represented as “1”, “01” or even “00000001”. All of these forms are acceptable, regardless of how many bits are defined for the register.

It is likely, but not required, that any given implementation will generate a fixed number of digits for any given parameter. For example, the XGR50’s ‘FAULT’ register has 10 defined bits, and is currently always represented as a 4-digit hexadecimal number. However, interfacing systems *must not* rely on this, since future firmware revisions may use a different number of digits, and may even vary the number of digits depending on the value being reported.

### Error values

Error responses – that is, those with a ‘\*’ (star) following the name – contain an error value.

An error value is an enumerated value, as described in the next section. The following values are defined.

|  |  |
| --- | --- |
| ***Value*** | ***Meaning*** |
| readonly | The accessed parameter cannot be modified. |
| writeonly | The accessed parameter can only be written. |
| range | The value supplied for a parameter is out of range. |
| type | The value supplied for a parameter has the wrong type. |
| unknown | The parameter or operation name is not recognised. |
| fail | The requested operation would normally be valid, but failed on this occasion. This will normally indicate some internal fault condition. |
| busy | The requested operation would normally be valid, but the system is not currently ready. This generally indicates that the operation can be retried after a simple delay. |

Implementations *may* define additional error values.

### Enumerated values

Enumerated values assign names to represent a selection from a set of possibilities.

*The only enumeration defined by the basic pro**tocol is the set of error codes.*

### Implementation notes

The specification is such that it is easy to implement in C or C++. Specifically, the format specifications for numbers are sub-sets of the formats supported by the standard C/C++ library.

* Floating point numbers (i.e. analogue parameters) may be generated using functions from the printf() family, by specifying the “%g”, “%e” or “%f” format controls.
* Integer values may be generated using functions from the printf() family, by specifying the “%u” format control.
* Register values may be generated using functions from the printf() family, by specifying the “%x” format control.
* Floating point numbers may be parsed using the strtof() or strtod() functions. If these functions are used, then some care may be required to ensure that the values ‘inf’ or ‘nan’ are handled in a sensible fashion.
* Integer values may be parsed using the strtoul() function. However, this function must not be allowed to auto-detect the base. The conversion base is passed as the third argument, and must be 10, and not zero. If strtoul() is allowed to determine the conversion base automatically, then “013” will be converted using base 8, giving a value of 11 (in decimal).
* Register values may also be parsed using the strtoul() function. In this case, the base must be specified as 16.

## Check values

Since RS232 links (at least) are ‘unreliable’ and may introduce errors, the protocol allows for the addition of a check value to any request or response.

The check value is an 8-bit number, computed from the contents of the request or response, as described in section 12.2. The value is formatted as two hexadecimal digits, and then a ‘#’ (hash) and the formatted value are appended to the request or response.

Suppose that the controller wishes to send the request ‘VDEM=1000’ (as in the examples above). Assume also that the check value is computed to be 208 (decimal). The two digit check value is D0, which is the hexadecimal representation of 208, so the resulting request is ‘VDEM=1000#D0’.

The following rules apply to the inclusion of check values:

* All implementations *must* accept check values.
* An implementation *may* choose not to verify check values.
* If the check value of a request is verified, and found to be incorrect, the request *must* be ignored, except for the production of diagnostic information. Such a request *must not* generate a response.
* If the check value of a response is verified, and found to be incorrect, the response *must* be ignored, except for the production of diagnostic information.
* If a request is received that carries a check value, the corresponding response *must* also carry a check value.
* Implementations *may* make check values mandatory. In that case, any request or response that does not carry a check value should be treated as if the check value were present but incorrect. It is expected that this feature will be controlled by a configuration option.

## Error handling

Lines that do not form a valid request or response should normally be ignored. In some cases, it may be appropriate to generate diagnostic information.

The following *must* be **silently** ignored:

* Empty lines.
* Comment lines; i.e. those that start with a ‘;’ (semi-colon)

The following *must* be ignored, but may generate some diagnostic information:

* Any line that is not empty, not a comment, and not a valid request or response.
* Requests received from a PSU system
* Responses received from a controller
* Requests or responses with an incorrect check value (see section 11.8)
* Responses where the parameter or operation name does not match the request.

# APPENDIX

## Formal syntax

The following section gives a syntax specification in an approximate BNF form

### Controller (request)

<ctrl\_line> ::= <request>

| <comment>

| <empty>

<request> ::= NAME ‘=’ VALUE [CHECK]

| NAME ‘?’ [CHECK]

| NAME ‘!’ [CHECK]

### PSU (response)

<sys\_line> ::= <response>

| <comment>

| <empty>

<response> ::= NAME ‘:’ VALUE [CHECK]

| NAME ‘$’ [CHECK]

| NAME ‘\*’ <error\_code> [CHECK]

<error\_code> ::= NAME

### General

<comment> ::= ‘;’ (any characters except CR or LF)

<empty> ::= (nothing)

VALUE ::= [SIGN] DIGITS

| [SIGN] DIGITS [‘.’ DIGITS] [EXPONENT]

| [SIGN] ‘.’ DIGITS [EXPONENT]

| NAME

| HEXVAL

EXPONENT ::= EXP [SIGN] DIGITS

NAME ::= NAME\_FIRST [NAME\_MORE]

EXP ::= (‘E’ or ‘e’)

SIGN ::= (‘+’ or ‘-’)

DIGITS ::= (one or more from 0..9)

HEXVAL ::= (one or more from 0..9, a..f, A..F)

NAME\_FIRST ::= (one of a..z, A..Z or ‘\_’)

NAME\_MORE ::= (one or more from 0..9, a..z, A..Z, ‘\_’ or ‘.’)

CHECK ::= HASH HEX\_DIGIT HEX\_DIGIT

HASH ::= ‘#’

HEX\_DIGIT ::= (one of 0..9, A..F, a..f)

## Checksum value calculation

The check value is calculated using the 8-bit CCITT-8 polynomial, (). The algorithm shifts left (i.e. processes the most significant bit first).

The following C code illustrates how to calculate the check value.

uint8\_t crc8 ( uint8\_t const \*buffer, unsigned int count )

{

uint16\_t crc = 0;

unsigned int bit;

while ( count > 0 )

{

crc ^= \*buffer;

for ( bit=0; bit<8; bit++ )

{

crc <<= 1;

if ( crc & 0x100 )

{

crc ^= 0x07;

}

}

}

return crc;

}

# Document History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Issue | Date | Release no. | Design | Sales |
|  |  |  |  |  |
|  |  | | | |
|  |  |  |  |  |
|  |  | | | |