

SPEC – Power and Performance

Design Document

SPEC PTDaemon

Standard Performance Evaluation Corporation

Table of Contents

1	Overview	3
2	Code Structure	3
3	API / network protocol	3
3.1	Standard Commands	4
3.2	Optional Commands	6
3.3	Command Syntax for Multichannel Analyzers	6
4	Behavior of the Immediate Commands	7
5	Special Return Values	7
6	Command Line Options	8
7	Usage Examples.....	8
7.1	Common Initialization Steps	8
7.2	Fixed-time benchmarks.....	8
7.3	Fixed-work benchmarks (such as CPU2000)	8
7.4	Sampling Intervals.....	9
7.5	Timing Precision.....	9
8	Adding Measurement Devices.....	10
8.1	Implementing Accuracy Calculation and Range Checking.....	10

1 Overview

SPEC's power/temperature daemon (also known as PTDaemon, PTD or ptd) is used by benchmarks to offload the work of controlling a power analyzer or temperature sensor during measurement intervals to a system other than the SUT. It hides the details of different power analyzer interface protocols and behaviors from the benchmark software, presenting a common TCP-IP-based interface that can be readily integrated into different benchmark harnesses.

The benchmark harness connects to PTDaemon by opening a TCP port and using the simple commands detailed in the API section of this document. For larger configurations, multiple IP/port combinations can be used to control multiple devices.

PTDaemon can connect to multiple analyzer and sensor types, via protocols and interfaces specific to each device type. The device type is specified by a parameter passed locally on the command line on initial invocation of the daemon.

The communication protocol between the SUT and PTDaemon does not change regardless of device type. This allows the benchmark to be developed independently of the device types to be supported. It is both an API and a network protocol.

2 Code Structure

PTDaemon is implemented in C using a main process that controls initialization and the network command interface, with a separate thread that manages the power analyzer or temperature sensor. Some analyzers that do not operate with a standard command/response structure also require an additional thread to receive asynchronous data from the device.

Upon startup the main process performs the following steps:

- Parses command-line arguments and checks for validity
- Initializes the analyzer interface
- Connects to the analyzer and checks for valid responses
- Initializes the network connection and opens a socket in listen mode
- Optionally sets ampere and voltage ranges

At this point, the main process goes into a command-handling loop, where it receives commands written to the API specifications in the next section, parses them, performs any necessary actions, and sends a response back across the network.

The device thread is started any time a measurement interval is begun using the "Go" or "Timed" commands. It consists of a timed loop that calls the analyzer read function, logs the values, then sleeps any remaining time before the next scheduled sample. The device thread ends when either

- a "Timed" command has completed
- a "Stop" command is issued
- or, a network error occurs

The device thread will not disconnect when the network connection is closed cleanly by the remote end. This allows measurements to continue without an active network connection, allowing the possibility of measuring power during arbitrary periods including reboots or low power modes without the necessity of any third-party software control.

3 API / network protocol

- All commands and responses are in ASCII for ease of parsing and debugging.
- The command syntax is: [command name], [parameter1], ..., [parameter n]
- Commands are case sensitive.

- Lists of results are comma-separated and/or semicolon-separated.

3.1 Standard Commands

Hello : check for existence of PTDaemon

Syntax : "Hello\r\n"

Normal Response: "Hello, ptdaemon here!\r\n"

Error Response:

Help : list all available PTDaemon commands

Syntax : "Help\r\n"

Normal Response: "Accuracy Amps amps Go Hello Help Humidity humidity Identify Mark PF pf RE RF RR RT RW R* SR Stop Temperature temperature Timed Volts volts Watts watts X\r\n"

Error Response:

Identify : get name and capabilities of connected device, host and software

Syntax : "Identify\r\n"

Normal Response: "{device name},{averaging interval (ms)},{watts?},{volts?},{amps?},{pf?},{energy?},{frequency?},{valid for submissions?}, version={ptd_version_number-checksum},OS={host OS},mode={power | temperature}\r\n"

Error Response:

Note1: To allow for future PTDaemon additions, it is recommended that software parsing the return value allow for more values in the comma-separated list than currently listed.

Note2: additional "extension" fields as of October 2008, which follow the mode string, are "<accuracy estimation enabled?>, <range setting enabled?>, <number of channels>"

Example Response: "ZES LMG95-ZES ZIMMER Electronic systems

GmbH;LMG95;11780806;3.131,1000,1,1,1,1,1,1,1,version=1.3.3d-de613e2f,OS=Windows

XP,mode=power,1,1\r\n"

The LMG95 is a 1-channel analyzer which supports PTDaemon accuracy checking and PTDaemon range setting.

Mark : mark different sections of the PTDaemon log to assist in identification of benchmark phases. The {logfile marker} will be logged to the PTDaemon output log along with the sample-by-sample power readings.

Syntax : "Mark,{ logfile marker }\r\n"

Normal Response: "Marking measurements with {logfile marker}\r\n"

Example: "Mark,first measurement\r\n"

Error Response:

Timed : start a new timed power measurement at a specified sampling rate, ignoring specified numbers of samples at rampup and rampdown time

Syntax : "Timed,{Samples},{Sample_ms},{Rampup_samples},{Rampdown_samples}\r\n"

Normal Response: "Timed measurement, %d Samples at %dms with %d rampup samples and %d rampdown samples\r\n"

Example: "Timed,240,1000,5000\r\n" means start a timed measurement of 240 samples at 1 second intervals after a 5 second rampup period

Error Response:

Go : start a new untimed power measurement

Syntax : "Go,{Sample_ms},{Rampup_samples}\r\n"

Optional format: "Go,{Sample_ms},{Rampup_samples},{logfile marker}\r\n"

Normal Response: "Starting untimed measurement, sampling at %dms with %d rampup samples\r\n"

Example: “Go,1000,5\r\n” means start an untimed measurement at 1 second intervals after 5 rampup samples

Special case: a Sample_ms value of 0 indicates that the analyzer should be sampled at its default averaging interval. Use of the default averaging interval provides the most accurate power measurements.

The “logfile marker” is an optional parameter (see also Mark command)

Error Response:

Stop : stop an untimed power measurement after the completion of the current sample

Syntax : “Stop\r\n”

Normal Response: “Stopping untimed measurement\r\n”

Error Response:

X : exit program

Watts, Volts, Amps, PF, Temperature, Humidity : read aggregated (avg, min, max) measured {wattage, voltage, amp, power factor, temperature, humidity} values, ignoring error readings and rampup/rampdown values

Syntax : “Watts\r\n”

Normal Response: “Watts,{avg_watts},{min_watts},{max_watts},{total sample count},{bad sample count},{valid sample count}\r\n”

Error Response: “Watts,-1.0,0,0,0,0,0” when no valid samples exist

- Sample count definitions
 - total sample count is incremented each time an attempt is made (or should have been made) to read the device, successful or not
 - valid samples is incremented each time the device is read successfully AND not rampup or rampdown sampling; cumulative readings are only added to in the valid sample case
 - bad samples is incremented each time a device read fails or is missed due to time
 - At the end of a measurement period, the cumulative values are divided by the valid sample count to get the averages
 - total sample count = Valid samples + bad samples + rampup/rampdown samples

watts, volts, amps, pf, temperature, humidity : read accumulated measured {wattage, voltage, amp, power factor, temperature, humidity} sample values from the last measurement interval

Syntax : “watts\r\n”

Normal Response: “watts,{totalsamples},{watts1},{watts2},{watts3},...\r\n”

Error Response:

“R” commands: instantaneous read commands

Note: use of the R commands for purposes other than instantaneous readings is **strongly discouraged**. The probability of missing or duplicate readings is greatly increased by the use of the R command set. For accurate characterization of power usage over time, the Timed and Go commands, when used properly, are the only commands designed to provide accurate results.

RW: Read Watts : read most recent wattage value; if a measurement interval is in progress, reads the last value from the measurement array; if no measurement in progress, reads the device directly

Normal Response: “Watts,{last watts value}\r\n”

RT: Read Temperature

RH: Read Humidity

R*: Read Watts,Volts,Amps,PF (or analogue Temperature,Humidity on temperature devices) : read most recent watts, volts, amps, pf values; if a measurement interval is in progress, reads the last value from the measurement array; if no measurement in progress, reads the device directly

Normal Response: "Watts,{last watts value},Volts,{last volts value},Amps,{last amps value},PF,{last PF value}\r\n"

RR: Read Ranges : read most recent settings for ampere and voltage ranges

Normal Response: "Ranges,{Amp Autorange},{Amp Range},{Volt Autorange},{Volt Range}\r\n"

Values: for autorange settings, -1 indicates "unknown", 0 = disabled, 1 = enabled

For range values, -1.0 indicates "unknown", >0 indicates actual value

Note: this command is implemented for all devices, but will simply return unknown for analyzers which do not provide range information or for which range-reading code has not been written

RE, RF: Read Energy, Read Frequency: these only operate outside of measurement intervals; if a measurement is in progress, "Meter busy" is returned

Normal Response: "WattHours,{last WH value}\r\n"

Normal Response: "Frequency,{frequency value}\r\n"

Error Response: "Meter busy\r\n"

Note: these commands are currently not implemented for most devices

3.2 Optional Commands

Uncertainty : read aggregated calculated uncertainty values for power

Syntax : "Uncertainty\r\n"

Normal Response: "Uncertainty,{average},{min},{max},{total sample count},{invalid sample count},{valid sample count},{unknown sample count}\r\n"

Error Response: "Uncertainty,-1.0,0,0,0,0,0" when no valid samples exist

- Sample count definitions
 - total sample count is incremented each time an attempt is made to calculate uncertainty, successful or not
 - valid samples are samples where uncertainty was calculated and was below the threshold (1%)
 - invalid samples are samples where uncertainty was calculated and found to be above the legal limit
 - unknown samples are samples where uncertainty could not be calculated. Possible reasons include unknown ranges, errors reading values from analyzer, and situations where code has not been implemented to calculate the uncertainty
 - total sample count = Valid samples + invalid samples + unknown samples

SR : set power analyzer ampere or voltage range

Syntax : "SR,{V | v | A | a},{Auto | value}\r\n"

Normal Response: "Range A changed\r\n" or "Range V changed\r\n"

Example: "SR,A,4.5\r\n" means set the power analyzer's ampere range to the lowest available ampere range of that power analyzer needed to measure 4.5A, e.g. 5A.

Note: this command cannot be used during a measurement interval

Error Response: "Meter busy"

3.3 Command Syntax for Multichannel Analyzers

Some of the multichannel analyzers supported by SPEC (see the Accepted Devices list) have the ability to set and read channels on an individual basis as well as in a summed mode. The standard command syntax is extended as follows for multichannel devices:

- The Identify command contains the number of channels in the last output parameter (as of October 2008).
- For the Set Range command (SR), the channel number is an optional comma-separated parameter at the end of the command string.
 - If the channel number is present, the command affects only the specified channel
 - If the channel number is absent, SR will set the range identically on all channels.
 - Examples:
 - “SR,A,a” : Ampere range of all channels is set to autorange
 - “SR,V,220,1” : Volt range of channel 1 is set to 200V
- For the commands Amps, Watts, Volts, PF, Accuracy, RE, RF, RW, R*, amps, watts, pf and volts the channel number is an optional comma-separated parameter at the end of the command string.
 - If the channel number is greater than 0, values will only be returned for the specified channel
 - If the channel number is 0, values will be returned for the summed channels followed by each individual channel, with each set separated by “;”
 - If the channel number is not specified, values will be returned for the summed channels only
 - Examples:
 - “Amps” : Ampere values are read from the sum channel
 - “Volts,1” : Voltage values are read from channel 1
 - “PF,0” : Power factors of the sum channel and all individual channels are read

4 Behavior of the Immediate Commands

As noted in the documentation above, the action taken for an immediate command differs depending on whether or not a measurement interval was in progress. Since serial ports and power analyzers are not multi-threaded devices, the power daemon must enforce restrictions on command usages to avoid any conflicts. Therefore, once a measurement interval is begun, the analyzer thread is considered to “own” the analyzer.

Thus, when a measurement interval is in progress, the value returned by an immediate command is the last valid value read by the device thread. When a measurement interval is not in progress, the code queries the device and returns the value from that query.

5 Special Return Values

The power daemon uses special values to indicate problems with analyzer readings.

A “-1.0” value returned by an immediate command indicates that the previous attempt to read that value from the analyzer failed due to communication or other device error.

A “-2.0” value indicates that the previous attempt to read the analyzer during a Go or Timed command was skipped because the reading attempt was started past its scheduled time.

These error values are not included in the average, minimum or maximum values calculated by PTDaemon. They will be counted in the “bad sample count” field of commands returning aggregated values.

6 Command Line Options

PTDaemon is called with the following parameters:

ptd [options] <device-type-#> <device-port>

<device-type-#>: With `-h` option a list of all available power and temperature devices is shown.
 <device-port>: The device-port number, e.g. COM1 for Windows or /dev/sttys0 for Linux.

The following options are available:

-p port	Network port, default 8888
-q	Quiet mode, minimal output to console
-v	Verbose mode, extra debug output
-l logfile	Log standard output to logfile
-d debugfile	Log debug output to file
-t	Temperature mode, default is power mode
-V <value>	Set analyzer voltage range to lowest needed for value or autorange if value = 'a\n'
-A <value>	Set analyzer ampere range to lowest needed for value or autorange if value = 'a\n'
-u	List unsupported devices that may work
-c <channel>	Select channel number for multichannel analyzers operating in single channel mode

Options only available on Windows:

-g	Use GPIB-USB interface (default is RS232)
----	---

7 Usage Examples

7.1 Common Initialization Steps

During benchmark initialization, open a connection to `device_ip:device_port` and use the *Identify* command to retrieve PTDaemon's configuration. Items to be checked include determining whether PTDaemon is in power or temperature mode, and checking whether the measurement device has been accepted by SPEC. Save the results of the Identify command for later use by the benchmark reporting code. Exit the benchmark (or disable power measurement) if invalid responses are found.

7.2 Fixed-time benchmarks

At the beginning of each measurement interval, use the {averaging interval} output of the Identify command to determine the device's averaging interval. Then divide the programmed measurement interval, rampup time and rampdown time by the device's averaging interval to determine the number of total, rampup and rampdown samples. For example, for a benchmark with a 240 second measurement interval, and a device with a 1000ms averaging rate, *"Timed,240,1000,5,5"*

This gives you a 5 second rampup and rampdown (power not measured) and 230 seconds of steady-state during the 240 second run.

After the end of each measurement interval, use the *Watts*, *Volts*, *Amps* and *PF* commands to retrieve, display and store the power measurements for that interval. If the entire set of individual measurements is desired, those may be retrieved with the *watts*, *volts*, *amps* and *pf* commands.

7.3 Fixed-work benchmarks (such as CPU2000)

At the beginning of each sub-benchmark, use the *"Go,0,0"* command to start data collection at the analyzer's default averaging interval. At completion of the sub-benchmark, when all threads have

completed, use the “*Stop*” command to stop data collection, then , use the *Watts*, *Volts*, *Amps* and *PF* commands to retrieve, display and store the power measurements for that sub-benchmark. If the entire set of individual measurements is desired, those may be retrieved with the *watts*, *volts*, *amps* and *pf* commands.

7.4 Sampling Intervals

The sampling interval used should **always** match the device’s averaging interval as returned by the Identify command. The averaging interval of a device is the period of time over which the high-speed samples taken by the device are averaged when returning a reading across its communication interface. If a different value is chosen, the data returned over a period of time either will not contain samples of the entire time period, or will contain overlapping samples.

7.5 Timing Precision

Due to the different characteristics of the many supported power analyzers, there can be some time variation between the time of a reading and the actual time range represented by that reading. Some analyzers output their readings continuously, independent of any start/stop requests from PTDaemon. Others run their measurement windows at fixed time intervals based on their own internal clock, while PTDaemon must request the measured data. Still others provide their readings on a sliding window basis, giving PTDaemon the measurement for a window that ends at the time PTDaemon makes the request. The speed of analyzer interfaces varies from 9600baud to 115200baud or faster. Also, some analyzers can provide all readings in response to a single command, while others need separate commands for each of the 4 readings. Thus, for an analyzer with an averaging window of 1 second, a reading from PTDaemon could represent a measurement ending anywhere from 0-2 seconds previously.

For that reason, there are several recommendations regarding PTDaemon usage:

- Power measurements should be started and stopped after a system has reached steady state. This can be accomplished either by delaying the Go or Timed command until steady state conditions are reached, or by using the rampup features of those commands to avoid measuring power during benchmark rampup.
- Longer measurements are recommended for better accuracy. Use of PTDaemon readings for non-steady-state measurements less than 10 seconds are likely to have a wide variability; 100 seconds or more would be preferable.

8 Adding Measurement Devices

Each measurement device needs its own module, along with modifications to meters.cpp and meters.h and the usage() output in ptd.cpp. For devices that give responses only to requests on their communications port, wt210.cpp is a good prototype. Other devices may output continuously after a simple command; WattsUpPro-usb.cpp is a simple prototype for one such type of analyzer; zes.cpp is a considerably more sophisticated example that creates an independent thread to capture all output from the device.

The three components of a device module are the Test(), Read() and Close() sections. The Test() routine opens the device connection and transfers the commands needed to initialize and prepare the device, and does enough response checking to ensure that the device is actually connected. The Read() routine takes a single sample and stores the data for further processing. The Close() routine does any cleanup commands needed to the device and closes the serial port.

A fourth component of the device module that is recommended for power analyzers starting with SPECpower_ssj2008 version 1.10 is the SetRange() section. This code adds the ability for the controller system to set ampere and voltage ranges at the beginning of a test or during a test to allow for greater measurement accuracy under different load conditions.

Logging function calls are used to maintain the device reading data during a sampling interval.

```
extern char* meterArg; // argument string for meter setup

bool TestMeterX();
// open connection to meter, do any initialization necessary
// use meterArg for COM1/COM2, ttyS0, etc.
// verify that meter gives correct responses
// return true if correct, false if error

bool ReadMeterX();
// send command(s) to meter to get Watts, Volts, Amps, PF
// receive response(s)
// log responses with appropriate Log* routines
// return false on error

// example reading watts from Extech
        if ((sign == 0) || (errchk == 14))
        {
                LogWatts(-1.0,0);
                fprintf(stdout, "**** Error in watts check bytes
*****\n");
        }
        else {
                LogWatts(watts,1);
        }
        fprintf(stdout, "watts = %f, ", watts);

bool CloseMeterX();
// do any meter commands necessary to stop
// close connection to meter
```

8.1 Implementing Accuracy Calculation and Range Checking

Starting with SPECpower_ssj2008 version 1.2, accuracy calculations and range checking are required for accepted power measurement devices. Calculating the accuracy of each individual power measurement, and ensuring that readings are within manufacturers' supported ranges, help to provide more accurate power readings.

There are many possible scenarios to consider. The following sections list common scenarios and the appropriate actions to be taken in those cases.

Range Checking:

- some analyzers provide validity indicators along with every reading; those are easy to mark when invalid (e.g. LogWatts(value,0))
- other analyzers return unique readings (very large values, for example) when out of range; those should be marked invalid with a LogWatts(-1.0,0) value
- still others may return a reading which is outside the specified range (e.g. specs are only valid from 2% to 130% of range); those should return the value along with an invalid mark (LogWatts(value,0))

Wattage Accuracy Calculations:

- for any wattage reading where any component (watts, v, a, pf) has been marked invalid for any reason (including out-of-range, or an error in taking the reading), accuracy should be marked unknown (LogAccuracy(-1.0,0)) unless the analyzer provides a validity indicator for the wattage value
- for a situation outside "normal" conditions for which code has not been written (e.g. power at a frequency of 40Hz, when the code is written for 50-60Hz accuracy calculations), accuracy should be marked as unknown (LogAccuracy(-1.0,0))
- for an analyzer whose ranges cannot be read dynamically, accuracy should be marked unknown unless the range can be set by the software during initialization
- for an analyzer whose range can be read dynamically, but which returns only an autoranging indication when autoranging is enabled, accuracy should be calculated based on the highest ranges provided by the analyzer