

# Lambda GENESYS Power Supply

*IRM software support*

Fri, Jun 1, 2007

Lambda makes a GENESYS series of power supplies that can be accessed via an RS-232 serial port connection. Previous work that was done to support their zero-UP series was adapted to support this series. The user is Seva (Vsevolod) Kamerdzhev for the TEL1 work. This note describes this adaptation from the local application called ZUPS to the new GENS that supports the GENESYS series. (See the previous note, Lambda Power Supply about ZUPS.)

The protocol used for the GENESYS series is similar, but by no means identical, to that used for the ZUP series. The command format uses delimiters of a CR (0x0D), whereas the ZUP series used delimiters of ":" and ";". The format of a reply to the status query (STT?) for GENS includes parentheses and commas, whereas that for ZUP lacked such punctuation. The status bytes are now returned in hexadecimal, rather than in binary as in the ZUPS case. In GENS, the normal reply to non-query commands is OK.

In GENS, the reply to the STT? command looks like this:

```
MV(000.20),PV(000.00),MC(0.0000),PC(0.2169),SR(84),FR(00)
```

When sending a value in a setting command, the value must be separated from the command by a space for GENS, whereas in ZUPS, there was no space. A reply to a query for a single numeric value includes only the numeric value, with no indicating identifier.

The means of querying the supply for the peak voltage and peak current specs is different. In ZUPS, we used a REV? query of the software version, which also included the model number. But in GENS, we use the IDN? command, which returns a reply such as follows:

```
LAMBDA,GEN300-2.5
```

In GENS, there is an operational status byte and a fault register. The fault register works with the fault enable register, which GENS does not support, making the fault register non-useful. The third status byte is used to hold the "down" status that indicates whether the serial port communication seems to be working ok. It also holds error codes that may occur in reply to setting commands.

In GENS, support is included for setting the power supply into remote or local status.

As with ZUPS, GENS can support up to 31 power supplies, numbered from 1. In practice, there is only one power supply that will be used for TEL1. Connection is via RS-232 at 9600 Baud, 8 bits, no parity, one stop.

## **Commands used in GENS for GENESYS**

Each command is terminated with a carriage return (0x0D).

ADR nn	Set target ps address for subsequent communications
IDN?	Get ps model id, as in "LAMBDA,GEN300-2.5"
RMT 0	Transition to local mode
RMT 1	Transition to remote mode.

PV n	Set output voltage, where allowed range depends upon ps model.
PC n	Set output current, where allowed range depends upon ps model.
OUT 1	Set output on
OUT 0	Set output off
FLD 1	Arm foldback protection
FLD 0	Release foldback protection
OVP n	Set over voltage protection level in volts, depending on ps model.
OVP?	Returns present over voltage protection value.
UVL n	Sets under voltage protection limit in volts, depending on ps model.
UVL?	Returns under voltage protection limit value.
AST 1	Set auto-restart on
AST 0	Set auto-restart off

Just as in ZUPS, eight analog channels are mapped by GENS:

- \*Measured voltage
- \*Measured current
- \*Over-voltage level
- \*Under-voltage limit
- Set voltage
- Set current
- Peak output voltage
- Peak output current

The asterisk (\*) indicates that the actual voltage and actual current channels are controllable, along with the over-voltage and under-voltage values. The set voltage and set current are readings of what the power supply thinks are the current programmed setting values. They are certainly affected by changes made to the actual voltage and current channels, but they are not the basis for control.

Just the same as in ZUPS, the new GENS uses a data file to obtain the numeric formats used by the particular power supply model, identified by its peak voltage and peak current. The new data file name is DATAGEND. Correspondingly, there is a diagnostic data stream GENSLOG.

Entries in the DATAGEND data file are:

<i>Field</i>	<i>Size</i>	<i>Meaning</i>
pV	2	peak voltage rating, in volts
pA	2	peak current rating, in amps
fSV	1	format for setting voltage, as 0xwd, where w=width, d=decimals
fSC	1	format for setting current
fOP	1	format for setting over-voltage protection level
fUP	1	format for setting under-voltage protection limit

**LA Parameters**

The following list of parameters are used with the GENS local application:

<i>Prompt</i>		<i>Meaning</i>
ENABLE	B	Enable Bit#
SUPPLIES		Number of power supplies to scan
BASE	C	Base Channel# for analog data, 8 channels per supply
BASE	B	Base Bit# for digital data, 4 bytes per supply

Returning to digital control, 8 bits of write-only digital control occupy a single byte:

<i>Bit#</i>	<i>Meaning</i>	<i>Command</i>
7	Enable output	OUT 1
6	Disable output	OUT 0
5	Auto-restart on	AST 1
4	Auto-restart off	AST 0
3	Set remote mode	RMT 1
2	Set local mode	RMT 0
1	Foldback protection on	FLD 1
0	Foldback protection off	OUT 1

The format of the status registers in the first 3 of the 4 bytes allotted per supply are:

<i>Bit#</i>	<i>Meaning</i>
7	local/remote mode, 1=local, 0=remote
6	n.u.
5	fold back protection, 1=enabled
4	auto-restart mode, 1=enabled
3	fault status, 1=at least one enabled and occurred
2	no fault, 1=no faults enabled, or supply is normal
1	supply on and in constant current mode
0	supply on and in constant voltage mode

<i>Bit#</i>	<i>Meaning (when set)</i>
7	J1 enable terminal opened
6	front panel OUT button pressed to off
5	J1 shut-off occurred
4	OVP shut-off occurred
3	Foldback shutdown occurred
2	OTP shutdown occurred
1	AC fail has occurred
0	n.u.

<i>Bit#</i>	<i>Meaning</i>
7	down, 1=serial port eliciting no response
6	command err code (3 bits) 1,2,4,6,7
5	"
4	"
3	n.u.

2	program error code (3 bits) 1,2,3,4,5
1	"
0	"

There is no need for a special analog control type designed to support control of these power supplies. The controllable channels are dummy channels, where the analog control type field is simply `0x12000000`. The local application monitors changes that occur in the dummy setting values for these channels. When a change is noticed, it arranges to have the new setting promptly passed to the supply at the next convenient time. A subsequent update in the set voltage or set current channel does not affect the setting fields of the actual voltage and actual current channels. As a result, the actual voltage and actual current channel setting field values reflect what the user desires and what the local application should act upon, whereas the set voltage and set current channel readings reflect what the power supply thinks are its current programmed settings. A user is likely to be interested in the latter settings only when analyzing a problem.

The digital control options are handled as a set of dummy bits that cause the indicated effect to be delivered to the supply. This scheme is analogous to that used for HLRF systems. A dummy bit is set to "1" to ask that a digital action be taken, and the LA clears the bit when it has accomplished that action. Eight bits may be enough for all digital actions supported by the power supply.

All in all, one has a set of 8 analog channels and 4 digital bytes that make up the software interface to each power supply. This approach means that no system changes are required to implement this kind of power supply support; all the required logic is provided by the local application. In addition, configuring the front end to support a set of such power supplies is easy; merely install a single instance—only one serial port is available—of the local application, and also define the set of relevant channels and digital status bits and dummy control bits for each supply.

By use of the action control bit approach, none of the particulars of the GENESYS protocol for these control actions need be forced on the user. The local application merely delivers the appropriate commands as necessary.

### *Sequencing*

Suppose there is a fixed set of possible commands that can be generated for each supply, so that rather than a command queue, there is a list of all possible commands. The act of placing a command into a queue dedicated to a single power supply is then replaced with marking the command for that power supply to be done. When a new setting is seen, by monitoring the setting field of the channel associated with the actual current, for example, and that command is already marked but not yet sent, the program merely overwrites the setting value for that command. (If it is not marked, then set the value anyway and mark it.)

When the sequencing reaches a new power supply, it should check for possible marked commands in some priority order, yet to be determined. If any is found marked, it arranges to send out that command and clear the mark. After that is completed, it sends the usual `STP?` command out, and upon return of the reply data moves on to the next supply.

To back up, when starting to serve a given supply, it should first check whether that supply is "down," meaning that its attempt to communicate failed the last time, say, because no

reply data was returned following the last STT? command. If the supply is down, the first step should be to send the IDN? query to get the peak voltage and peak current specs for the supply. Only if that succeeds in prompting a reply does it then send out the usual STT? command. If there is no reply from the IDN? command, it just moves on to the next supply.

With this sequencing logic, there is only one possible command to be done before issuing the usual STT? command. This means that the time to sequence through all supplies, even in the case of high activity of setting requests, should be no more than twice as long as it is in the absence of any such setting requests.

In addition, as noted above, there should be logic that occasionally marks commands to be done less frequently, such as querying the over-voltage and under-voltage values. Throttling such slow requests can be done if no commands are marked, and a counter counts down for that supply. That would mean the specification of time would be in units of the time required for executing the entire sequence, passing through all power supplies. This time depends mostly upon the number of supplies and somewhat on the amount of setting activity.

In the case that the node is “up,” there should already be a peak voltage and current value available. But any time no such values exist, that again is a reason to send the IDN? command before doing anything else.

**Programming details**

The sequencing logic for driving the serial port can be used in both the 15 Hz cycle call of the LA as well as a serial input call, which occurs upon receiving a line of text in response to a query command. In the DoCycle routine, there might be the following:

If serial NOT busy, look for command waiting to be sent, send it, set serial busy.  
 Else if serial busy, and if time-out not exceeded, do nothing.  
 Else declare current PS down, AdvancePS.

In the DoSerial routine,  
 Process received serial input  
 Update data pool with latest values  
 If last comand was STT?, AdvancePS.

**Diagnostics**

It may be useful to include a diagnostic to record serial activities in a data stream. One might be called “GENSLOG “. Two kinds of records can be used: one is written whenever a setting command is sent; a second is written every time a reply is received. Each record is 16 bytes long, with the last 8 bytes used for the time-of-day, down to half ms within the current cycle, in keeping with the typical cases of such diagnostics. The first 8 bytes have two forms:

	<i>Field</i>	<i>Size</i>	<i>Meaning</i>
(1)	psNum	1	One-byte power supply address in range 1–31
	command	3	Three-character ascii command code
	setData	4	BCD data value for setting, using 0xA for decimal point
(2)	psNum	1	One-byte power supply address in range 1–31
	command	3	Three-character ascii command code
	nCRecv	1	#characters received in reply
	eTime	3	Elapsed time from command to reply received, in ms units

The difference between the two forms is that a command that sends a numeric value to a targeted power supply is of the first form. A command that sends no numeric value simply uses the 4-byte field to hold any single digit value sent as an integer (with no decimal point indicator). The second format is used whenever a reply is received. If we use the BCD format for numeric values, then the `nRecv` byte can be used to indicate which format is given, since it will be nonzero for any reply, whereas the first byte of the BCD form will be zero, since no numeric values use more than 6 digits. The ms elapsed time values are not expected to use more than one byte. If desired, we could use all BCD format for these four bytes, which would make it easier to read. That would make the `0xA` digit stand out a bit more, as it will be part of each numeric value.

To illustrate how the two formats might look in hexadecimal, here are examples:

- (1) 0143 5520 0005 A200 ps#1, PC , 5.200 amps
- (1) 014F 5554 0000 0001 ps#1, OUT, 1 (set output on)
- (2) 0153 5454 5300 0075 ps#1, STT, 57 chars received, 75 ms

The times for type 1 formats indicate when the command started being sent out the serial port. The times for type 2 formats indicate when the reply data has been received. The elapsed time recorded, then, reflects the baud-rate time to send out the command, the time to process the command by the power supply, and the time to receive the reply text. It is assumed that the local application is called soon after the arrival of the CR-LF characters that terminate the reply text.

### Data structures

The structure suitable for the communications management with each power supply:

```
typedef struct {
    byte psState;      /* power supply up/down state */
    byte psAdr;       /* power supply address */
    short marks;      /* flags for signaling settings */
    short peakVolt;   /* peak voltage value */
    short peakCur;   /* peak current value */

    byte waitPeak;    /* count-down for MDL/REV request */
    byte waitRemote; /* count-down for remote/local req */
    byte waitOver;    /* count-down for overVolt request */
    byte waitUnder;   /* count-down for underVolt request */
    short version;    /* firmware version# *100 */
    short timeoutCnt; /* #timeouts due to lack of response to query */

    byte4 status;     /* 3 status bytes, 1 control byte */
    long spareL;      /* -- */

    float actVolt;    /* actual values */
    float actCurr;

    float actSetVolt; /* set values */
    float actSetCurr;
}
```

```

float actSetOver;
float actSetUnder;

float setVoltMon; /* last recognized setting values */
float setCurrMon;

float setOverMon
float setUnderMon;
} PSContext;

```

There is an array of these 64-byte structures that is indexed by power supply numbers 1–31. The values for the `marks` field are as follows:

	<i>Bit#</i>	<i>Meaning</i>
(hi byte)	7	Output ON
	6	Output OFF
	5	Auto-restart ON
	4	Auto-restart OFF
	3	Set Remote
	2	Set Local
	1	Foldback protection ARM
	0	Foldback protection RELEASE
(lo byte)	7	Set voltage
	6	Set current
	5	Set over-voltage
	4	Set under-voltage
	3	Get over-voltage
	2	Get under-voltage
	1	Get remote status
	0	Get model

When readings need to be updated, following processing of the reply data, the following procedure is used:

```
PROCEDURE SetRealR(chan: Integer; n: Integer; VAR data: Real);
```

The arguments are the starting channel#, the number of consecutive channels targeted, and the array of floats that are to be converted into 16-bit raw values and deposited in the reading fields of successive `ADATA` table entries.

### **Digital status**

In order to deliver digital status to the data pool, it may be useful to deliver it to the dummy addresses found in the `BADDR` table. On the following cycle, then, the usual `RBINARY` Data Access Table entry (0x0405) will update the `BBYTE` table.

If the `BBYTE` table entries are targeted instead, the contents of the memory bytes pointed to by the dummy byte addresses will not reflect what is in the `BBYTE` table, since the `BBYTE` table is not updated every cycle for all power supply status bytes. The updating only takes place for a given power supply when new status data is received in response to the `STT?` command. In

between such updates, the normal updating of the `BYTE` table occurs using values that are held in the dummy bytes. This is why the contents of the dummy bytes need to be set. To be sure of the `BADDR` entry contents, require that it look like a non-volatile memory address.

### ***Restore***

If all of the analog control is handled by dummy channels, how will the settings be passed to the supplies during the Restore operation occurring following system reset? The Restore operation will only deliver the data to the dummy channels. To solve this problem, assume that the local application does all of the settings following initialization. It simply marks all of these setting activities to be performed, using the setting values it obtains from the dummy setting values. The assumption here is that these dummy values always represent the user intended values for the power supplies. If settable values are installed in a power supply locally, the computer will not know about them, although it will update the set value channels. It will not attempt to “discover” that the user has done something locally and update the dummy setting values to reflect the set values. The user is expected to perform settings remotely through the computer. Without doing this, the restore operation, which can result from resetting the front end or merely restarting the local application, cannot be expected to work correctly and deliver the proper values to the supplies.

### ***Post-implementation notes***

What is going on when nothing is happening? Each 15 Hz cycle, the current supply is queried by an `STT?` command. As soon as a reply is received, an `ADR` command is sent to select the next power supply in numeric sequence. The reply to the `STT?` command is expected to take more than one 15 Hz cycle to execute and return a reply at 9600 baud, so the result is that supplies are sequenced every 2 cycles. (Actually, it is almost possible to receive the reply within a single 15 Hz cycle; if that is so, we might sequence through the supplies at a 15 Hz rate. The unknown is the time that the supply requires between reception of the `STT?` command before it begins its 57-character reply message.)

When a setting is to be performed for the current supply, the appropriate setting message is sent, and on the following 15 Hz cycle, the `STT?` query is sent, and after receiving a reply, it moves to the next supply. The last command sent to a supply is the query command, and no more than one command is sent to that supply before the `STT?` command is sent. In the extreme case that settings are waiting to be sent to all supplies, the sequencing from supply to supply should take 3 cycles rather than 2.

### ***Installation steps***

The `GENS` local application supports up to 31 Lambda power supplies that use the `GENESYS` serial (RS232-based) protocol. The supplies are addressed beginning at 1. The number of supplies scanned is given by a parameter. Eight analog channels are allocated for each power supply in sequence, with the meanings of voltage, current, overvoltage protection, undervoltage limit, voltage setting, current setting, peak voltage, and peak current. Four binary bytes are allocated for each supply in sequence, including operational, fault, and program status bytes and one control byte. A data stream called “`GENSLOG`” is defined to hold diagnostic records of recent settings that have been made, plus a record of recent data that has been acquired, including the number of characters received and the time between sending the query command and receiving the reply. The serial port baud rate should be set to 9600 baud, the fastest rate supported by the `GENESYS` protocol. So, if one has to support 5 supplies, numbered from 1–5, there will be a sequence of 40 analog channels defined, plus 20 bytes (160 bits) defined. Of course, an entry for `GENS` must be placed in the

Local Applications table (LATBL) specifying the appropriate parameters, and the LOOPGENS program downloaded from node0508. The local application also needs access to a data file called "DATAGEND". The data file contains the specification of numeric formats assumed by the various power supply models in the GENESYS series, and it can support up to 24 models, each one defined by the two values of peak voltage and peak current.