Please Note:

There is an updated version of this document available as part of the pys3g library:

https://github.com/makerbot/s3g/blob/master/doc/s3g_protocol.markdown

This document is considered to be deprecated and will be removed in the future.

Cheers,
Matt Mets (matt.mets@makerbot.com)

Overview

This document describes the protocol by which 3rd and 4th generation RepRap electronics communicate with their host machine, as well as the protocol by which the RepRap host communicates with its subsystems. The same simple packet protocol is used for both purposes.

Where to find implementations of this protocol

Firmware

For Gen3 and Gen4 electronics (Cupcake, Thing-O-Matic, etc): http://github.com/makerbot/G3Firmware.

For MightyBoard (Replicator): http://github.com/makerbot/MightyBoardFirmware

Host software

Currently the only software suite implementing this protocol is ReplicatorG. It can be found on GitHub at http://github.com/makerbot/ReplicatorG.

Network Overview

The RepRap Generation 3 electronics set consists of several hardware components:

1. A single **Master Controller** which controls the 3-axis steppers, communicates with a host PC, supports a storage card and controls a set of toolheads.
2. A set of **Stepper Drivers** which drive the steppers based on signals from the master controller. The communications between the drivers and the master controller is outside of the scope of this document.

3. A number of **Tool Controllers** which control extruders, cutters, frostruders, etc.

The two communications channels covered by this document are:

1. The **Host Network**, between a host computer and the master controller
2. The **Slave Network**, between the master controller and one or more tool controllers.

The host network is generally implemented over a standard TTL-level RS232 serial ion. The slave network is implemented as an RS485 serial bus driven by SN75176A transceivers.

Packet Protocol

Protocol Overview

Each network has a single master: in the case of the host network, this is the host computer, and in the case of the slave network, this is the master controller. All network communications are initiated by the network master; a slave node can never initiate a data transfer.

Data is sent over the network as a series of simple packets. Packets are variable-length, with a maximum payload size of 32 bytes.

Each network transaction consists of at least two packets: a master packet, followed by a response packet. Every packet from a master must be responded to.

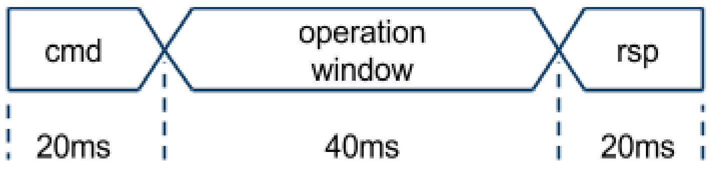
Commands will be sent in packets. All commands are query/response. The master in each pair will always initiate communications, never the slave. All packets are synchronous; they will wait for a response from the client before sending the next packet. The firmware will continue rendering buffered commands while receiving new commands and replying to them.

Timeouts

Packets must be responded to promptly. No command should ever block. If a query would require more than the timeout period to respond to, it must be recast as a poll-driven operation.

~~All communications, both host-mb and mb-toolboard, are at 38400bps~~. It should take approximately 1/3rd ms. to transmit one byte at those speeds. The maximum packet size of 32+3 bytes should take no more than 12ms to transmit. We establish a 20ms. window from the reception of a start byte until packet completion. If a packet is not completed within this window, it is considered to have timed out.

It is expected that there will be a lag between the completion of a command packet and the beginning of a response packet. This may include a round-trip request to a toolhead, for example. This window is expected to be 36ms. at the most. Again, if the first byte of the response packet is not received by the time 36ms. has passed, the packet is presumed to have timed out.



Maximum packet transaction times

Handling Packet Failures

If a packet has timed out, the host or board should treat the entire packet transaction as void. It should:

- Return its packet reception state machine to a ready state.
- Presume that no action has been taken on the transaction
- Attempt to resend the packet, if it was a host packet.

Command Buffering

To ensure smooth motion, as well as to support print queueing, we'll want certain commands to be queued in a buffer. This means we won't get immediate feedback from any queued command. To this end we will break commands down into two categories: action commands that are put in the command buffer, and query commands that require an immediate response. In order to make it simple to differentiate the commands on the firmware side, we will break them up into two sets: commands numbered 0-127 will be query commands, and commands numbered 128-255 will be action commands to be put into the buffer. The firmware can then simply look at the highest bit to determine which type of packet it is.

Command Types

Range	Type	Description
0-127	Query Commands	Commands that can be executed immediately, and thus do not need to

		be queued. Generally return pre-calculated information and should be fast.
128-255	Action Commands	Commands that must be executed one at a time, and in a particular order. These commands should be buffered for smooth operation. Many of there commands may take up a large amount of time to complete.

Packet structure

Command packets are sent from the Master to the Slave. This can either be from the Host CPU to the Master uC or from the Master uC to the Slave uC. It consists of a two-byte header, a variable-length payload, and a 1 byte CRC footer.

The packet consists of:

Index	Name	Details
0	Start byte	This byte always has the value 0xD5 and is used to ensure synchronization.
1	Length	This byte indicates the length of the payload (excluding the header and CRC) in bytes.
2...	Payload	The packet payload. This is appended to the buffer and either immediately executed and removed (if a query type) or buffered for execution (if an action type).
Length+2	CRC	This is the 8-bit iButton/

		Maxim CRC of the payload.

Command length is implicit in the command structure; no explicit separator is needed.

Command structure

Host Commands

The payload of a packet consists of one command. Each command contains a header, as follows:

Index	Name	Details
0	Command	The command code to send to the device.
1-N	Arguments	Arguments to this command (details below as 'command payload').

Slave Commands

The payload of a packet consists of one command. Each command contains a header, as follows:

Index	Name	Details
0	Slave ID	The ID of the slave device being addressed. The value 127 represents any available device; see below. New as of firmware version 2.92. Previously, 255.
1	Command	The command code to send to the device.
2-N	Arguments	Arguments to this

		command (details below as 'command payload').

Slave IDs

The slave ID is the ID number of a toolhead. A toolhead may only respond to commands that are directed at its ID. If the packet is corrupt, the slave should *not* respond with an error message to avoid collisions.

The exception to this is the slave ID 127. This represents any listening device. The address 127 should only be used when setting the ID of a slave.

New as of firmware version 2.92. Previously, the broadcast address was 255.

Response Packets

Response packets look just like command packets. The only difference is the payload is guaranteed to contain a response code as the first byte, as described below. The only exception is certain debugging packets, which will specifically indicate such in their description.

Response Packet Payload Structure

Index	**Name**	**Details**
0	Response Code	This is a standard response code that is included with all packets and is described below.
1-N	Response Data	This contains response-specific data. Length is implicit based on the command being responded-to.

If the command was a non-request command, the response data is always empty.

Response Code Values

Note: These are changed starting in version 2.92. Previously, they were 0x0 - 0x7.

Response Code	**Interpretation**
0x80	Generic error, packet discarded.
0x81	Success.
0x82	Action buffer overflow, entire packet discarded.
0x83	CRC mismatch, packet discarded.
0x84	Query packet too big, packet discarded.
0x85	Command not supported/recognized.
0x86	Success; expect more packets. Used when a single reponse packet cannot contain the entire message to be retrieved.
0x87	Downstream timeout (for example, a toolhead timed out).

Data Formats

The protocol is a byte-oriented protocol. Payloads may contain various information in a variety of formats, but will generally be limited to the formats described below. Multi-byte datatypes will always be transmitted in Little-endian mode. Remember, this means that the least significant byte ("littlest") byte is sent first! For a more in-depth discussionof little-endian storage, see: http://en.wikipedia.org/wiki/Endianness#Little-endian

Various Data Types

Type	**Range**
uint8	0 to 255
uint16	0 to 65,535
int16	-32767 to 32,767
uint32	0 to 4,294,967,295
int32	-2,147,483,647 to 2,147,483,647

Test Commands

The command codes of the form 0xFX and 0x7X are reserved for diagnostic test packets. The firmware is not guaranteed to implement any of these operations.

0x70 - Echo test

This command indicates that the client should resend the original packet data, *minus* the echo test byte. *The client will not send a response code--the remaining payload will be copied verbatim.* This will persuade the client to send any valid packet the host wants.

Payload: variable.
Response: resend of payload.

0x71 - Generate bad response packet

This command will request that the client respond with a test packet that exhibits the given error.

1	No response
2	Skip start byte
3	Bad packet length (specified length too long)
4	Bad/incorrect CRC
5	Start byte, then timeout

Payload: 1 byte.
    uint8: Error type to simulate.

0x72 - Simulate bad packet reception

This command will essentially instruct the recipient to behave as if the received packet had experienced the specified error. The error codes are:

1	Received noise
2	Bad packet length (specified length too long)
3	Bad/incorrect CRC
4	Timeout

Payload: 1 byte.
  uint8: Error type to simulate.

0x73 - Passthru to slave

This command will pass the enclosed packet on to its slave bus, and return a copy of the returned packet if the packet receipt is successful. If the packet receipt is unsuccessful (for example, if the packet times out) it should return a generic error. *The client will not send a response code if successful-- instead it will just mirror as much of the packet as possible.*

Payload: N bytes.
  The *payload* of the packet to send to the slave bus.

Response: N bytes.
  Complete *payload* of response packet, truncated to fit if necessary.

0x74 - Clear command queue

This command will clear the command queue completely and immediately.

Payload: 0 bytes.

0x75 - "NO SUCH COMMAND" command

This command code has no definition, and is reserved an an unimplemented command for test purposes.

0x76 - Set Debug Code

This command sets the blink code for the target board. Setting the code to '0' will turn off the debug code.

Payload: 2 bytes.
  uint8: the debug register to set. Should always be '0'.
  uint8: the value to set the debug register to. For register '0', the number of blinks to display.
Response: 0 bytes.

0x77 - Get Debug Code

Retrieve the value of a debug register. '0' is the debug register containing the blink code. Other registers are undefined.

Payload: 1 byte.

uint8: the debug register to read. Should always be '0'.
Response: 1 byte.
uint8: the value of the debug register.

0x78 - Get Debug Buffer

(only on the firmware_debug branch in github/makerbot:G3Firmware)
Retrieve the contents of the debug buffer and clears it.

Response: N bytes.
The current contents of the debug buffer, the data in the buffer is whatever the firmware puts in there.

0xF0 - Command queue filler

This is a nop command that will remain on the command queue until cleared. It may specify padding bytes to help fill space on the buffer.

Payload: 1 + N byte.
uint8: the number of padding bytes following this byte. The total space on the command buffer will be 2+N.
uint8 * N: padding bytes. The actual values are ignored.

Master Microcontroller Commands (3-axis controller)

Query Commands

0 - Get Version - Query firmware for version information

This command allows the host and firmware to exchange version numbers. It also allows for simple automated discovery of the firmware. Version numbers will always be stored as a single number, Arduino / Processing style. Thus, the versions will be 0 to 65535.

Payload (2 bytes) - host version
uint16: Host Version.

Response (2 bytes) - firmware version
uint16: Firmware Version

1 - Init - Initialize firmware to boot state

Initialization consists of:

- resetting current position to 0,0,0
- clearing command buffer
- setting range to eeprom value (if it exists) otherwise its 0.

Payload (0 bytes)
Response (0 bytes)

2 - Get Available Buffer Size - Determine how much free memory we have for buffering.

This command will let us know how much buffer space we have available for action commands. It can be used to determine if and when the buffer is available for writing. If we are writing to the SD card, it will generally always report the maximum number of bytes available.

Payload (0 bytes)

Response (4 bytes)
uint32: number of bytes available in the command buffer.

3 - Clear Buffer - Empty the command buffer

This command will empty our buffer, and reset all pointers, etc to the beginning of the buffer. If writing to an SD card, it will reset the file pointer back to the beginning of the currently open file. Obviously, it should halt all execution of action commands as well.

Payload (0 bytes)
Response (0 bytes)

4 - Get Position - Get the current position of the tool

Useful for determining current position of the toolhead. It is up to the host software to add or subtract the toolhead offset to determine the actual position of the toolhead. It will also return the status of the various endstops for diagnostic information.

Payload (0 bytes)

Response (13 bytes)
int32: current x position, in steps
int32: current y position, in steps
int32: current z position, in steps
uint8: bits marked for endstop status: (7-0) : | N/A | N/A | z max | z min | y max | y min | x max | x min |

~~**5 - Get Range - Get the maximum range of travel on all axes. - OBSOLETE**~~

~~When find axes maximums is called, it will record the maximum as the range. internally, it will keep track of its range and this function will~~

~~simply report that internal value. We cannot assume that there are endstop switches for all axes at both min and max positions. The firmware will know this, so the "Find Axes min/max" commands may be partially no-ops in this case. The firmware should always respect this range and never go beyond it.~~

~~**Payload (0 bytes)**~~
~~**Response (12 bytes)**~~
~~**uint32**: x range, units in steps~~
~~**uint32**: y range, units in steps~~
~~**uint32**: z range, units in steps~~

~~**6 - Set Range - Set the maximum range of travel on all axes. - OBSOLETE**~~

~~This command tells the firmware what its maximum range of travel is. The firmware should always respect this. This should be recorded into the eeprom for future usage.~~

~~**Payload (12 bytes)**~~
~~**uint32**: x range, units in steps~~
~~**uint32**: y range, units in steps~~
~~**uint32**: z range, units in steps~~

~~**Response (0 bytes)**~~

7 - Abort Immediately - Stop Machine, Shut Down Job Permanently

This function is intended to be used to terminate a print during printing. Extruder and accessories off, steppers disabled, everything shuts down.

Payload (0 bytes)
Response (0 bytes)

8 - Pause / Unpause - Halt Execution Temporarily

This function is intended to be called infrequently by the end-user in order to make build-time adjustments immediately and easily. No buffers or other run-time variables are changed. The pause command should also prompt the firmware to send a pause command to the toolhead.

On pause, it stops all movement, stops extrusion, moves up a 'safe z' amount, and waits for new commands.
On unpause, it lowers to original Z, optionally restarts extrusion, and resumes movement.
agm: Currently we do not do the "safe Z" backoff. I'd like to introduce that as a parameter instead.

Payload (0 bytes)

Response (0 bytes)

~~**9 - Probe - Move in Z axis (negative) until probe hits something**~~

~~This command will wait until the buffer is clear to avoid conflicts. It will also prevent any buffered commands from executing for the duration of the probe.~~

~~**Payload (6 bytes)**~~

~~**uint32**: feedrate (in microseconds between steps) (max 71.58 minutes)~~

~~**uint16**: timeout (in seconds before abort) (max = 18 hours) (default = 1 minute)~~

~~**Response (4 bytes)**~~

~~**uint32**: Z position when probe has been triggered.~~

10 - Tool Query - Query a tool for information

This command is for sending a query to the tool. The master firmware will then pass the query along to the appropriate tool, as well as passing the response back as well. This allows the tool specific commands to be developed independently between.

Payload (2 + N bytes)

uint8: the index of the tool to query
uint8: the query command for the tool.
N bytes: command specific payload, variable size (could be 0)

Response (0-N bytes)

0-N bytes: the command specific response, if any.

11 - Is Finished - See if the machine is currently doing anything

This command lets us know if the machine is finished executing its current command queue.

Payload (0 bytes)

Response (1 byte)

uint8: 0 if still in progress; 1 if finished.

12 - Read from EEPROM

Read the specified number of bytes from the given offset in the EEPROM and return it in the response packet. The maximum read size is 32 bytes.

Payload (3 bytes)

uint16: the offset of the read

uint8: the number of bytes to return, N. N <= 16.

Response (N bytes)

N bytes: the data read from the EEPROM.

13 - Write to EEPROM

Write the given bytes to the EEPROM starting at the specified location. The maximum payload size is 16 bytes.

Payload (3+N bytes)

uint16: the offset of the write
uint8: the length of the data
N bytes: the data to write

Response (1 byte)

uint8: the number of bytes successfully written.

14 - Capture to file

Capture all subsequent commands up to the next "end capture" command to a file with the given name on the SD card. The file will be stored in the root of the fat16 filesystem on the SD card. The maximum file name length permitted is 12 characters, including the '.' and file name extension. Welcome to the brave new world of MS-DOS 2.0!

Payload (N bytes)

N bytes: the name of the file in ascii, terminated with a null

Response (1 byte)

uint8: response code

- 0 if operation was successful
- 1 if no SD card was present
- 2 if SD card init failed
- 3 if partition table could not be read
- 4 if filesystem could not be opened
- 5 if root directory could not be opened
- 6 if SD card is locked

15 - End capture

Complete an ongoing file capture.

Payload (0 bytes)

Response (4 bytes)

uint32: the total size of the capture in bytes

16 - Playback capture

Play back a file containing captured data to the makerbot. The maximum file name length permitted is 12 characters, including the '.' and file name extension. While the makerbot is in playback mode, it will only respond to pause, unpause, and stop commands.

Payload (N bytes)

N bytes: the name of the file in ascii, terminated with a null

Response (1 byte)

uint8: response code

- 0 if playback successfully initiated
- 1 if no SD card was present
- 2 if SD card init failed
- 3 if partition table could not be read
- 4 if filesystem could not be opened
- 5 if root directory could not be opened
- 7 if file was not found

17 - Reset

Reset the board remotely. Useful for reprogramming the board without having to fiddle with switches. The board will reset immediately after sending the response.

Payload (0 bytes)

Response (0 bytes)

18 - Get next filename

Retrieve the volume name of the SD card or the next valid filename from the SD card. If a non-zero value is passed to the "restart" parameter, the file list will begin again from the start of the directory. The file list state will be reset if any other SD operations are performed.

If all the filenames have been retrieved already, an empty string is returned.

Payload (1 byte)

uint8: restart (0 to continue reading, 1 to begin anew)

Response (N+1 bytes)

uint8: an SD response code, one of the following:

- 0 if operation was successful
- 1 if no SD card was present
- 2 if SD card init failed
- 3 if partition table could not be read
- 4 if filesystem could not be opened
- 5 if root directory could not be opened

N bytes: the name of the file in ascii, terminated with a null. If the operation was unsuccessful, this will be the empty string. Even if there was an SD card error, the null string should be returned.

~~19 - Read debug registers~~

~~Retrieve the value in the specified debugging register. These are essentially "scratch" registers for development. All registers are 8-bit.~~
~~0: number of CRC errors in extruder responses~~
~~1: number of malformed packet responses from extruder~~
~~2: number of timeouts on packet responses from extruder~~
~~Payload (1 byte)~~
~~uint8: index of debug register~~
~~Response (1 byte)~~
~~uint8: value of register~~

20 - Get build name

Note: This possibly had a different implementation in the past. The following description is of how it is working in the Gen3 and MightyBoard codebases as of 4-17-2012

Retrieve the name of the file currently being built, if any.

Payload length limitations restrict this field to 31 characters.

Payload (0 bytes)

Response (N+1)
char[N+1]: a null-terminated string representing the build name

21 - Get Extended Position - Get the current position on 5 axes

New in version 2.4
Retrieve the current position of all axes that this machine supports.

Payload (0 bytes)

Response (22 bytes)
int32[n]: position of axis n, in steps.
int16: bits marked for endstop status: (A-0) : | b max | b min | a max | a min | z max | z min | y max | y min | x max | x min |

22 - Extended Stop - Stop subset of systems

New in version 2.7

Stop the stepper motor motion or other subsystems.

Payload (1 byte)
uint8: bitfield indicating which subsystems to shut down
bit 0: halt all stepper motion immediately.
bit 1: clear command queue.

Response (1 byte)
int8: Result code.

- 0 if command terminated normally and subsystems were stopped.
- 1 for generic error (command failed).

23 - Get Motherboard Status

Proposed for version 2.9
Retrieve some information about the motherboard

Payload (0 bytes)
Response (1 byte)
uint8: As below:

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	POWER_ ERROR	N/A	WDRF	BORF	EXTRF	PORF	N/A	N/A

POWER_ERROR - An error was detected with the system power. For Gen4 electronics, this means ATX_5V is not present.

WDRF - Watchdog reset flag was set at restart

BORF - Brown out reset flag was set at restart

EXTRF - External reset flag was set at restart

PORF - Power-on reset flag was set at restart

24 - Build Start Notification

Proposed for version 2.9

Tells the motherboard that it is about to begin a build, and provides some information about the job for status reporting. The maximum file name length permitted is 12 characters, including the '.' and file name extension.

Payload (4+N bytes)

uint32: Set number of steps in the build

N bytes: Length of name string (bytes)

25 - Build End Notification

Proposed for version 2.9
Tells the motherboard that the current build has ended.

Payload (0 bytes)

26 - Get communication statistics

Proposed for version 2.9
Gather statistics about toolhead communication.

Payload (5 bytes)
uint32: Number of packets received from the USB interface
uint32: Number of packets sent over the RS485 interface
uint32: Number of packets sent over the RS485 interface that were not responded to
uint32: Number of packet retries attempted
uint32: Number of bytes received over the RS485 interface that were discarded as noise

Buffered Commands

~~128 - RESERVED (Obsolete command)~~

129 - Queue point

This queues an absolute point to move to. This may make it easier to implement the host software, but it is much larger than the incremental points which will reduce your available command buffer size.

Payload (16 bytes)
int32: x coordinate (units are steps)
int32: y coordinate (units are steps)
int32: z coordinate (units are steps)
uint32: dda speed (in microseconds between steps on the max delta)

130 - Set Position

This command defines the current position of the toolhead as the given position. Ordinarily this is used to set a "home" position.

Payload (12 bytes)
int32: x position, in steps
int32: y position, in steps
int32: z position, in steps

131 - Find Axes Minimums - Blocking seek to hardware min switches, w/ timeout

This function will find the hardware minimum. It will home all axes at the same time, so it is up to the host software to generate two separate homing commands if you want to home the Z axis separately.

Payload (7 bytes)
uint8: axes flags, where bits 4-0 represent which axes to home: | N/A | N/A | N/A | B | A | Z | Y | X |
uint32: feedrate (in microseconds between steps)
uint16: timeout (in seconds before abort) (max = 18 hours) (default = 5 minutes)

132 - Find Axes Maximums - Blocking seek to hardware max switches w/ timeout

This function will find the hardware maximum, or position overflow (2^32), whichever is first.
It should also record this to the eeprom for future use. This function fill move all axes simulataneously, so it is important to generate two separate commands if you want to seek the Z axis separately.

Payload (7 bytes)
uint8: axes flags, where bits 4-0 represent which axes to home: | N/A | N/A | N/A | B | A | Z | Y | X |
uint32: feedrate (in microseconds between steps) (max 71.58 minutes)
uint16: timeout (in seconds before abort) (max = 18 hours) (default = ??)

133 - Delay - Simply stop and wait.

Exactly what it says.

Payload (4 bytes)
uint32: delay period (in microseconds) (max 71.58 minutes)

134 - Change Tool - Switch to a new tool

This will trigger a tool change. If the firmware supports automated tool changing, it will initiate its tool-changing routine. If it doesn't, it simply updates its currently selected tool and moves on.

Payload (1 byte)

uint8: index/address of the desired tool to select

135 - Wait For Tool Ready - Wait until a tool is ready before proceeding

This command is used to tell the machine to wait (for example for the extruder to heat up, etc.) It will poll the device with the 'are you ready' command until it responds affirmative

Payload (5 bytes)

uint8: the index of the tool to wait for before printing.

uint16: the delay between query packets sent to the tool head in milliseconds (default: 100 milliseconds)

uint16: the timeout before continuing without tool ready in seconds (default: 1 minute)

136 - Tool Action Command - Send an action command to a tool for execution

This command is for sending an action command to the tool. The master firmware will then pass the query along to the appropriate tool, as well as passing the response back as well.

Payload (3 + N bytes)

uint8: the index of the tool to query

uint8: the command for the tool.

uint8: the length of the command payload (N)

N bytes: command specific payload, variable size.

137 - Enable/Disable Axes - Explicitly power or depower steppers

This command is used to explicitly power steppers on or off. Generally, it is used to shut down the steppers after a build to save power and avoid generating excessive heat.

Payload (1 byte)

uint8: bitfield

bit 7: 1 to enable, 0 to disable

bit 6,5: ignored

bit 4: Axis 4 (B axis) on

bit 3: Axis 3 (A axis) on

bit 2: Axis 2 (Z axis) on

bit 1: Axis 1 (Y axis) on

bit 0: Axis 0 (X axis) on

138 - User Block

This command will halt further processing of the command queue until the block is cleared by user intervention. The "Proceed" command will release the block. This is not equivalent to a pause-- if you want to perform any toolhead operations, like stopping extrusion, you need to do so in a seperate command prior.

Payload (2 bytes):
uint16: Block identifier.

139 - Queue extended point

This queues an absolute point to move to. This may make it easier to implement the host software, but it is much larger than the incremental points which will reduce your available command buffer size. This is the 5D equivalent of *129 - Queue Point*.

Payload (24 bytes)
int32: x coordinate (units are steps)
int32: y coordinate (units are steps)
int32: z coordinate (units are steps)
int32: a coordinate (units are steps)
int32: b coordinate (units are steps)
uint32: dda speed (in microseconds between steps on the max delta)

140 - Set Extended Position

This command defines the current position of the toolhead as the given position. Ordinarily this is used to set a "home" position. This is the 5D equivalent of *130 - Set Position.*

Payload (20 bytes)
int32: x position, in steps
int32: y position, in steps
int32: z position, in steps
int32: a position, in steps
int32: b position, in steps

141 - Wait For Platform Ready - Wait until the build platform is ready before proceeding

New in version 2.4

This command is used to tell the machine to wait until the build platform is ready (target temperature has been reached). It will poll the device with the 'are you ready' command until it responds affirmative.

Payload (5 bytes)
uint8: the index of the tool to which the build platform is attached
uint16: the delay between query packets sent to the tool head in milliseconds (default: 100 milliseconds)
uint16: the timeout before continuing without tool ready in seconds (default: 1 minute)

142 - Queue extended point, new style

New in version 2.7
This queues a point to move to. It differs from old-style point queues (see command 139 et. al.) in that it no longer uses the DDA abstraction and instead specifies the total move time in microseconds. Additionally, each axis can be specified as relative or absolute. If the "relative" bit is set on an axis, then the motion is considered to be relative; otherwise, it is absolute.

Payload (25 bytes)
int32: x coordinate (units are steps)
int32: y coordinate (units are steps)
int32: z coordinate (units are steps)
int32: a coordinate (units are steps)
int32: b coordinate (units are steps)
uint32: duration of movement in microseconds
uint8: relative or absolute motion
bit 7,6,5: ignored
bit 4: Axis 4 (B axis) relative
bit 3: Axis 3 (A axis) relative
bit 2: Axis 2 (Z axis) relative
bit 1: Axis 1 (Y axis) relative
bit 0: Axis 0 (X axis) relative

143 - Store Home Positions

New in version 2.8
Record the current axes positions to EEPROM.
Payload (1 bytes)
uint8: As below:

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	N/A	N/A	N/A	B	A	Z	Y	X

144 - Recall Home Positions

New in version 2.8

Recall the axes positions stored in EEPROM.

Payload (1 bytes)

uint8: As below:

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	N/A	N/A	N/A	B	A	Z	Y	X

148 – Wait for Button

New in version 3.2; Mighty Board only

Wait for the user to press a button, or time out.

Payload (4 bytes)

uint8: Bit field of buttons to wait for. For 4.1 firmware use 0xFF.

uint16: Time in seconds before pause times out. A value of 0 indicates that the pause should never time out.

uint8: Options bitfield:

bit 0: abort on timeout - reset machine if timed out

bit 1: clear message on button push

bits 2-7: reserved

149 – Display Message to LCD command

New in version 3.2; Mighty Board only

This command will display a message to the LCD for a specified number of seconds, then revert to previous GUI if there is a running display.

Payload (N bytes)

uint8: Options bitmap:

bit 0: continuation bit. Messages with this bit set will be appended to the currently displayed message.

bits 1-7: reserved.

uint8: Cursor vertical position : range 0-3

uint8: Cursor horizontal position : range 0-19

uint8: Timeout in seconds. If 0, the message will not time out and will be displayed until superceded by another message or cleared by a button push.

uint8 * N (max 26): null-terminated list of characters to print. Text will auto-wrap at end of line. \n is recognized as new line start. \r is ignored. Note: words do not wrap automatically. You will need to insert newlines manually.

Slave Microcontroller Commands (Extruder)

Query Commands

0 - Get Version

This command allows the master and slave to exchange version numbers. It also allows for simple automated discovery of the firmware. Version numbers will always be stored as a single number, Arduino / Processing style. Thus, the versions will be 0 to 65535.

Payload (2 bytes) - host version

uint16: Host Version

Response (2 bytes) - firmware version

uint16: Firmware Version

2 - Get Toolhead Temperature

This returns the last recorded temperature. It's important for speed purposes that it does not actually trigger a temperature reading, but rather returns the last reading. The slave firmware should be constantly monitoring its temperature and keeping track of the latest readings.

Payload (0 bytes)

Response (2 bytes)

int16: Current Temperature in Celsius

17 - Get Motor 1 Speed (RPM)

Payload (0 bytes)

Response

uint32: speed in microseconds between complete rotations.

18 - Get Motor 2 Speed (RPM)

Payload (0 bytes)

Response

uint32: speed in microseconds between complete rotations.

19 - Get Motor 1 Speed (PWM)

Payload (0 bytes)

Response (1 byte)

uint8: current PWM value for the motor.

20 - Get Motor 2 Speed (PWM)

Payload (0 bytes)

Response (1 byte)

uint8: current PWM value for the motor.

25 - Read from EEPROM

Read the specified number of bytes from the given offset in the EEPROM and return it in the response packet. The maximum read size is 32 bytes.

Payload (3 bytes)

uint16: the offset of the read
uint8: the number of bytes to return, N. N <= 16.

Response (N bytes)

N bytes: the data read from the EEPROM.

26 - Write to EEPROM

Write the given bytes to the EEPROM starting at the specified location. The maximum payload size is 16 bytes. ***WARNING:*** this operation can be potentially slow (about 3.3ms * N) and will block all other operations. It's not recommended to write to the EEPROM while a build is in progress.

Payload (3+N bytes)

uint16: the offset of the write
uint8: the length of the data
N bytes: the data to write

Response (1 byte)

uint8: the number of bytes successfully written.

30 - Get Build Platform Temperature

This returns the last recorded temperature of the build platform. It's does not actually trigger a temperature reading, but rather returns the last reading. The slave firmware should be regularly monitoring its temperature and keeping track of the latest readings.

Payload (0 bytes)

Response (2 bytes)

int16: Current build platform temperature, in Celsius.

32 - Get Toolhead Target Temperature

This command retrieves the last temperature set for the toolhead.

Payload (0 bytes)

Response (2 bytes)

int16: The last set extruder temperature, in Celsius.

33 - Get Build Platform Target Temperature

This command retrieves the last temperature set for the build platform.

Payload (0 bytes)

Response (2 bytes)

int16: The last set build platform temperature, in Celsius.

34 - Get firmware build name

Retrieve the firmware build name, which is a human-readable string describing the build. Names should be chosen with an eye towards disambiguating builds targeted for different tools.

Payload length limitations restrict this field to 31 characters.

Payload (0 bytes)

Response (N+1)
char[N+1]: a null-terminated string representing the build name

36 - Get Tool Status

New in version 2.7
Retrieve some information about the current tool

Payload (0 bytes)
Response (1 byte)
uint8: As below:

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	EXTRUDER_ERROR	PLATFORM_ERROR	WDRF	BORF	EXTRF	PORF	N/A	EXTRUDER_READY

EXTRUDER_ERROR - HIGH means an error was detected with the extruder heater (if the tool supports one), LOW means it is fine. The extruder heater will fail if an error is detected with the sensor (thermocouple) or if the temperature reading appears to be unreasonable.

PLATFORM_ERROR - HIGH means an error was detected with the platform heater (if the tool supports one), LOW means it is fine. The platform heater will fail if an error is detected with the sensor (thermocouple) or if the temperature reading appears to be unreasonable.

WDRF - Watchdog reset flag was set at restart

BORF - Brown out reset flag was set at restart

EXTRF - External reset flag was set at restart

PORF - Power-on reset flag was set at restart

EXTRUDER_READY - HIGH means extruder has reached target temperature, LOW means it has not.

37 - Get PID state

New in version 2.7

Retrieve the state variables of the PID controller. This is intended for tuning the PID constants.

Payload (0 bytes)

Response (12 bytes)

int16: Extruder heater error term

int16: Extruder heater delta term

int16: Extruder heater last output

int16: Platform heater error term

int16: Platform heater delta term

int16: Platform heater last output

Action Commands

3 - Set Toolhead Target Temperature

This sets the desired temperature for the heating element. The slave firmware will then attempt to maintain this temperature as closely as possible.

Payload (2 bytes)
int16: Desired Target Temperature in Celsius

4 - Set Motor 1 Speed (PWM)

This sets the motor speed as a PWM value. Motor 1 generally corresponds to the extruder motor (M101-M103, M108 GCodes). It should not actually enable the motor until the motor enable command is given.

Payload (1 byte)
uint8: Desired PWM speed

5 - Set Motor 2 Speed (PWM)

This sets the motor speed as a PWM value. Motor 2 generally corresponds to the spindle motor (??? GCodes). It should not actually enable the motor until the motor enable command is given.

Payload (1 byte)
uint8: Desired PWM speed

6 - Set Motor 1 Speed (RPM)

This sets the motor speed as an RPM value. Motor 1 generally corresponds to the spindle motor (M101-M103, M108 GCodes). It should not actually enable the motor until the motor enable command is given. For this command to actually work, a quadrature encoder, or tachometer must actually be attached to the motor.

Payload (4 bytes)

uint32: speed in microseconds between complete rotations.

7 - Set Motor 2 Speed (RPM)

This sets the motor speed as an RPM value. Motor 1 generally corresponds to the spindle motor (??? GCodes). It should not actually enable the motor until the motor enable command is given. For this command to actually work, a quadrature encoder, or tachometer must actually be attached to the motor.

Payload (4 bytes)

uint32: speed in microseconds between complete rotations.

8 - Set Motor 1 Direction

This sets the desired rotation direction of the motor. It does not turn the motor on or off.

Payload (1 byte)

uint8: 0 = reverse / counter clockwise, 1 = forward / clockwise

9 - Set Motor 2 Direction

This sets the desired rotation direction of the motor. It does not turn the motor on or off.

Payload (1 byte)

uint8: 0 = reverse / counter clockwise, 1 = forward / clockwise

10 - Toggle Motor 1 (on/off)

This turns the motor on or off. It is possible to also set the direction of the motor as well. This makes the command fairly efficient, since you can turn the motor on while setting the direction as well. When the motor is enabled, it will move at the previously specified speed (either RPM or PWM).

Payload (1 byte)

uint8: The bottom two bits of this command determine direction and enable status:

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	N/A	N/A	N/A	N/A	N/A	N/A	DIR	ENABLE

ENABLE - HIGH means motor is enabled, LOW means motor is disabled.

DIR - HIGH means motor rotates Clockwise, LOW means motor rotates Counter Clockwise

11 - Toggle Motor 2 (on/off)

This turns the motor on or off. It is possible to also set the direction of the motor as well. This makes the command fairly efficient, since you can turn the motor on while setting the direction as well. When the motor is enabled, it will move at the previously specified speed (either RPM or PWM).

Payload (1 byte)
uint8: The bottom two bits of this command determine direction and enable status:

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	N/A	N/A	N/A	N/A	N/A	N/A	DIR	ENABLE

ENABLE - HIGH means motor is enabled, LOW means motor is disabled.

DIR - HIGH means motor rotates Clockwise, LOW means motor rotates Counter Clockwise

12 - Toggle Fan (on/off)

This turns the fan, or the output connected to the fan output on or off.

Payload (1 byte)
uint8: The bottom bit of this command determine enable status:

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	N/A	N/A	N/A	N/A	N/A	N/A	N/A	ENABLE

ENABLE - HIGH means fan is enabled, LOW means fan is disabled.

13 - Toggle Valve (on/off)

This opens the valve, or the output connected to the valve output on or off.

Payload (1 byte)
uint8: The bottom bit of this command determine enable status:

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Name	N/A	N/A	N/A	N/A	N/A	N/A	N/A	ENABLE

ENABLE - HIGH means valve is enabled (open), LOW means valve is disabled (closed).

14 - Set Servo 1 Position

This is basically just a wrapper to the Arduino hardware Servo library on pins 9 & 10. It supports full servo angles, and can also do continuous rotation: **http://arduino.cc/en/Reference/Servo**

Payload (1 byte)
uint8: desired angle (from 0 to 180)

15 - Set Servo 2 Position

This is basically just a wrapper to the Arduino hardware Servo library on pins 9 & 10. It supports full servo angles, and can also do continuous rotation: **http://arduino.cc/en/Reference/Servo**

31 - Set Build Platform Target Temperature

This sets the desired temperature for the build platform element. The slave firmware will then attempt to maintain this temperature as closely as possible.

Payload (2 bytes)
int16: Desired build platform temperature, in Celsius.

38 - Set Motor 1 Speed (DDA)

Proposed for 2.8 (RG)
This sets the stepper motor speed as a DDA value, or microseconds between steps. It should not actually enable the motor until the motor enable command is given. For future implementation of 5D (vs 4D) two DDA codes are sent - the DDA to start with and the DDA to end with. The third uint32 is the number of steps to take. The direction to go is set by code **8 - Set Motor 1 Direction**.

Payload (12 bytes)

uint32: speed in microseconds between steps (start of movement).

uint32: speed in microseconds between steps (end of movement).
uint32: total steps to take.

39 - Set Motor 2 Speed (DDA)

Proposed for 2.8 (RG)

This sets the stepper motor speed as a DDA value, or microseconds between steps. It should not actually enable the motor until the motor enable command is given. For future implementation of 5D (vs 4D) two DDA codes are sent - the DDA to start with and the DDA to end with. The third uint32 is the number of steps to take. The direction to go is set by code **9 - Set Motor 2 Direction**.

Payload (12 bytes)
uint32: speed in microseconds between steps (start of movement).
uint32: speed in microseconds between steps (end of movement).
uint32: total steps to take.

Unclassified

1 - Init - Initialize firmware to boot state

Initialization consists of:

- setting desired temperature to zero
- turning off all devices (fan, coolant, etc.)
- detaching all servo devices
- setting desired RPM / PWM to zero

Payload (0 bytes)
Response (0 bytes)

16 - Filament Status

Payload (0 bytes)

Response (1 byte)

uint8: status: 0 - empty, 255 = full, number between 0-255 represents percentage of filament remaining.

21 - Select Tool

Tell a tool that it has been selected as the default tool. The tool may or may not choose to do anything with that information.

Payload (0 bytes)
Response (0 bytes)

22 - Is Tool Ready?

Asks the tool if it is ready to be used. Examples of ready states might be: spindle up to speed, temperature at target, etc.

Payload (0 bytes)

Response (1 byte)

uint8: 0 for not ready, 1 for ready.

23 - Pause/Unpause - Halt Execution Temporarily

Behaves in a similar fashion to the pause/unpause functionality of the motherboard.

Payload (0 bytes)
Response (0 bytes)

24 - Abort - Terminate all operations and reset

Behaves in a similar fashion to the abort functionality of the motherboard.

Payload (0 bytes)
Response (0 bytes)

35 - Is Build Platform Ready?

New in version 2.6

Asks the heated build platform if it is ready to be used, typically waits for temperature to be reached.

Payload (0 bytes)

Response (1 byte)

uint8: 0 for not ready, 1 for ready.

40 - Light indicator LED

Proposed for 2.9

This command turns on an indicator light (for gen 4, the motor direction LED). This command is intended to serve as visual feedback to an operator that the electronics are communicating properly. Note that it should not be used during regular operation, because it interferes with h-bridge operation.