RQ Technology

Electronic Solutions, Inc. motor controls utilizing RQ Technology represent a new methodology for control communications. Unlike the one-way communication of the earlier RP Technology, RQ (Remote Query) is a bi-directional protocol that allows the motor control, or other RQ device, to answer back. Each RQ device retains all of its original RP Technology and functionality.

RQ devices are connected together to form an RQ Bus. A unit called an RQ Bridge is also connected to the RQ Bus to interface with an outside Controller/PC utilizing an RS-232 or RS-422 serial communication link. Since RQ allows unsolicited messages, half-duplex RS-485 will not work.

The purpose of the RQ Bus is to provide Home Automation developers with a platform to build broad networks of RQ devices. The RQ Bus is actually a network mapping of interconnected devices, supervised by a Controller/PC. The PC is able to perform query/response communication through an RQ Bridge to direct the network operations and individual node parameters. Higher level functions such as scene management and the network view of node position is supported by the Controller/PC. Lower level functions such as setting individual RQ device Load Thresholds, Deadbeat Counter setup, and Demo Mode selection continue to be supported through legacy button press commands on the RP Bus for both RP and RQ devices.

All RQ devices have an address, and commands are addressed directly to a node or, in some cases, globally to the entire Bus. In many cases, an RQ device responds to a command by placing a response RQ message onto the Bus that is picked up by the RQ Bridge and transmitted over the Serial Communications uplink to the Controller/PC. For some commands, an RQ device will not respond directly to the received command, but will perform the requested operation and place one or more relevant unsolicited messages onto the RQ Bus. There are also cases where unsolicited messages are initiated by a RQ device independent of any downlink commands from the Controller/PC.

RQ Bus Architecture and Implementation

The bi-directional RQ Bus is implemented using 6-wire phone cable with modular RJ-14 jacks/plugs. Nodes on the Bus are daisy-chained to each other and to an RQ Bridge device. Physical position on the Bus is irrelevant and total physical Bus wiring length can be up to 1000 feet, with up to 100 RQ devices on a Bus. The RQ Bridge receives its power from the RQ Bus and must be within 100 feet of an RQ device that is powering the Bus.

The RQ Bridge communicates with an outside Controller/PC over an RS-232 serial connection utilizing a black 6p4c mod jack (an ESI adapter provides conversion from DB-9 Serial COM port connector to 4-wire phone cable). An RS-422 connection is provided with 5 spring-loaded terminals. Straps are provided for selection of RS-232 or RS-422 and RS-422 termination resistors may also be selected with straps.

The Bus cabling supports both the RQ Bus as well as the legacy RP Bus. Within the 6-wire RQ Bus cabling, 3 wires are actually used for the RQ Bus and the remaining 3 wires are used for the legacy RP Bus. By physically separating RQ from RP within the same cable wiring, legacy RP devices are able to communicate with RQ and RP devices over the RP Bus. Although legacy RP devices do not respond to RQ signals, they do continue responding to the original RP button press sequences. Indeed, RQ devices respond also to the RP Bus button press sequences for control and programming purposes.

Caution: RP devices cannot be arbitrarily mixed within the cabling of RQ devices. Since RP devices do not propagate the RQ Bus, it is important to first wire the RQ devices to create an unbroken RQ Bus, and then connect an RP Bus of RP devices to the end of the RQ Bus. Alternatively, an RQ Splitter (e.g., SP2E2Q) may be used to "break" into an RQ Bus to attach an RP Bus without disrupting the integrity of the RQ Bus.
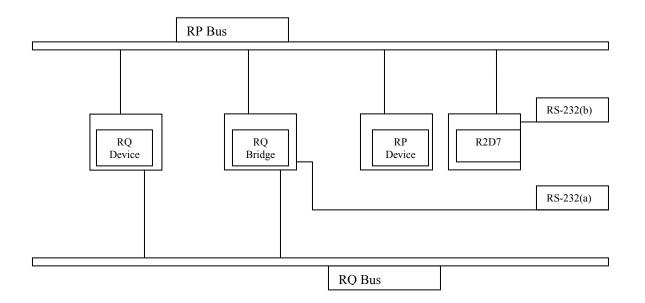
The diagram below depicts the overall communications connections of an RQ environment. The RQ Bus and the RP Bus are shown as physically separated communications paths, which they actually are. However, in reality, both the RQ Bus and the RP Bus are each 3 wires of the 6-wire phone cabling that interconnects all RQ and RP devices. Also shown in the diagram is the RQ Bridge, which provides a Serial Communications link (a) for a Controller/PC to interface with the RQ Bus.

Although the RQ Bridge does have an RP Bus connection, there is no ability for the Controller/PC (connected through

the RQ Bridge) to communicate with the RP Bus at this time. RP programming is typically done using an IR or RF handheld transmitter with the appropriate IR/RF receiver (not shown) plugged into an RP device (shown connected to the RP Bus in the diagram below). This methodology allows low level programming of the RP and RQ devices. In most cases, however, the RQ programming interface will suffice, since many common low level motor control programming commands (calibration, motor reverse, etc.) have been implemented with RQ commands (see *RQ6ØAUMHG Command Summary*).

Also shown in the diagram below is a legacy R2D7 device connected to the RP Bus, which allows a Serial Interface (b) where a Controller/PC can connect for RP Bus programming using ASCII strings instead of button presses. A single PC with multiple COM ports could be connected to the two Serial Interfaces or two separate PCs could be connected for programming and control. Most likely, a single PC may be connected at (b) for RP programming and then reconnected at (a) for RQ programming and ongoing network operations.

Large RQ Networks

Large networks of RQ devices require unique considerations. Electrically, the RQ Bus is designed to reliably support 1ØØ nodes. This is a theoretical and prudent limit based on component values. Component values vary within specified tolerances and node counts over 1ØØ on an RQ Bus may at first appear to work, but can encounter performance issues based on unique combinations of operations and message traffic.

Although the RQ Bus supports up to 1ØØ nodes, the legacy RP Bus does not. RP Bus maximum node count is a function of not only the node count itself, but also the types of nodes. While 1ØØ RQ nodes on a single RQ Bus can be connected reliably, the parallel legacy RP Bus in that same configuration may not operate. For complete information on working with the RP Bus within a large node count RQ Bus, see *RQ Technical Note 1*.

The address range limitation of an RQ Network is 46,655 nodes. Very large RQ Networks can be configured by utilizing multiple RQ Busses connected with various RQ support accessories. Particular attention must be paid to potential Network and Bus message traffic congestion when configuring large networks. Some RQ Message commands can generate large amounts of response message traffic.

RQ Protocol Summary
V1.∅

RQ Communication Messages

ASCII strings are used to form RQ messages. *Downlink* refers to messages from a Controller/PC to an RQ Bridge transmitted over the Serial Interface, while *Uplink* messages flow from an RQ Bridge to a Controller/PC. Improperly formatted messages or message content that is out of range will cause the message to be discarded by the RQ Bridge and an appropriate Uplink error message generated.

The RQ Bridge receives command messages from the outside controller/PC and, for the most part, passes these messages onto the RQ Bus. These messages may be directly addressed to an RQ device or they may be Global Messages directed at all RQ devices on an RQ Bus. Conversely, RQ devices push messages onto the RQ Bus that are directed to a particular RQ Bridge (there may be more than one Bridge on an RQ Bus) as a *solicited* response to a previously received message. In addition, RQ devices push messages onto the RQ Bus that are *unsolicited*. These unsolicited messages relate to motor position, direction, and other information. Unsolicited messages are picked up by all Bridges on an RQ Bus and uplinked over each Bridge's Serial Interface to their respective Controller/PC. Message format on the actual RQ Bus is similar, but not identical, to uplink/downlink messages.

There is no timeout of messages on the RQ Bus and any messages addressed to non-existent nodes will be lost. Although <x-on> and <x-off> are implemented for flow control between the controller and bridge, applications running on the Controller/PC must be aware of possible network buffer overflow and take care in sending downlink messages that generate a large volume of response messages.

At power up, RQ device nodes do not issue any RQ messages. However, RQ Bridges issue a Version uplink message and an <X-on> character when powered on. *Warning: Any downlink <X-off> without an <X-on> will cause a Bridge to stop communicating completely until the Bridge power is cycled off and back on*.

Verbose is a communications parameter of an RQ Bridge which is set to ON as a factory default. This allows full message communication by the Bridge as defined above. Verbose can be turned OFF, but is **not recommended**. With Verbose turned OFF, communications are severely limited and message throughput is very much dependent on message command content and addressing.

RQ Message Format

An RQ message always begins with a "!" (a.k.a. Bang) and ends with an "end character". There will always be an Address (3 ASCII characters) and a Command (1 ASCII character) as shown in the table below. In some cases the Data field will contain a variable number of characters or even no Data. A question mark ("?") in the Data field signifies a request. For downlink messages, the "end character" can be either ";" or <CR> (both are treated the same). The uplink message "end character" is selectable as either ";" or <CR> by a Bridge parameter.

RQ Address is always three ASCII characters composed of only ∅-9 and A-Z. For the case of ∅∅∅ (global command), all nodes are addressed and for that reason, no node can have ∅∅∅ as its address. All RQ Bridges are factory addressed at BR1 and RQ devices are randomly addressed from the factory (range is "C∅∅" – "ZZZ"). Command and Data fields are ASCII characters, but not "!" and not <control> characters (∅-31). The following table breaks down the message format into its component fields.

Start Character	Address	Command	Data	End Character
!	K∅B	N	Bob's Bedroom	;
!	M1∅	r	?	;
!	BR1	v	?	;
!	BR1	v	B1∅	;

The first message format shown above is a command ("N") to "name" an RQ node (which has an address of "K∅B") as "Bob's Bedroom". This message will make its way from the Controller/PC (over the Serial Interface) to the RQ Bridge where the message is checked for validity. The RQ Bridge passes the message onto the RQ Bus, where an RQ device with address "K∅B" receives the message, stores the name field, and echoes the message back onto the RQ Bus. The

RQ Bridge sees the echoed message and sends it (over the uplink Serial Interface) to the Controller/PC. The original command message string "!KØBNBob's Bedroom;" ends up echoed as a response back to the Controller/PC (as noted in the <u>RQ6ØAUMHG Table of Commands</u> at the end of this document). Not all downlink messages generate a response.

The second message in the table above is directed to an RQ node and the command is requesting the current position information for that node. The downlink message is "!M1Ør?;" and the response will not be an echo back, but will take the form of " !M1Ør5Ø;", indicating that the position (a window shade for example) is 5Ø% open.

The third message above is directed to the Bridge. It is a Version request and does not go onto the RQ Bus, but generates an uplink response message (fourth in the table above) as "!BR1vB1Ø;", which tells the application program on the Controller/PC that the Bridge is version 1.Ø.

Example Messages

Downlink Message	Uplink Message	Comments
!M11v?;	!M11vA1Ø;	Request to <u>M11</u> for version, <u>A</u>C Motor Control response w/ <u>1Ø</u> (version 1.Ø)
!M34r?;	!M34r76;	Request M34 position, M34 responds 76% and not moving
	!M34<36;	M34 is at 36% and moving toward reference (alt. possible response)
	!M34Enc;	Position is unknown because unit is not calibrated (alt. possible response)
!M15m62;	!M15>18;	M15 move to 62%, responds at 18% moving away from reference
	!M15r62;	Later, Responds at 62% when finished moving

Global Command Messages

With an address of "ØØØ", a message becomes a Global Command. An application program may use this form of message to gather information that could be used to create a "map" of the RQ Network. The only Global Command that a Bridge will respond to is the Version request. A downlink message of "!ØØØv?;" will cause the Bridge to respond with an uplink message of "!BR1vB1Ø" and then pass the original downlink message onto the RQ Bus. Every powered-on RQ device on the RQ Bus will respond, creating a flurry of response messages onto the RQ Bus. The RQ Bus Protocol itself sorts out the attempts by all of the RQ devices to respond at once, but there is no safeguard for buffer overflow if a large number of messages are generated by a multitude of rapid Global Command Messages from a Controller/PC. In such a situation, an uplink error message such as "!BR1Eml;" will be generated. A detailed description of the RQ Bus Protocol is beyond the scope of this document and is not required for normal operation and programming of an RQ Network. Typical Global Command Messages are shown in the table below.

Downlink Message	Uplink Message	Comments
!ØØØv?;	!BR1vB1Ø; … !MØ1vA1Ø; …	The Bridge, then all devices respond with version
!ØØØr?;	… !MØ1r28; …	All devices show position (Bridge does not respond)
!ØØØgJ;	!MØ1>18;!MØ2>87;!MØ2r9 Ø; … !MØ1r9Ø; …	Bridge does not respond. Devices programmed for scene "J" (but not at scene "J") will respond with current location and direction, then send another message when they arrive at scene "J". Any devices already at scene "J" location, will not respond at all. In this case, scene "J" is 9Ø% and MØ1 responds at 18 moving toward 99. MØ2 responds at 87 moving toward 99 and almost immediately announces arrival at 9Ø. Later, MØ1 finally arrives at 9Ø and announces also.

RQ Addressing

RQ devices are randomly addressed from the factory in the range of CØØ to ZZZ. ØØØ is reserved for Global Commands and ØØ1 to BFF is reserved for user address setup. Although Bridges are factory addressed as BR1, they may be readdressed.

If, by chance or by mistake, a Bridge and an RQ device end up with the same address, the Bridge will "mask" the device, in most cases. If a downlink message is valid for the Bridge, it will respond with an uplink message, if appropriate, but the message will not be put onto the RQ Bus. This will prevent the RQ device with the same address from seeing the message. If the downlink message that is addressed to both the Bridge and an RQ device (with the same address) is not considered valid by the Bridge (as a Bridge appropriate message), it will be discarded and an uplink error message will be sent. However, if a downlink Global Verify message is sent, the address in the first response (the Bridge) will match the address in one other response within the whole group of responses.

!ØØØv?;	!BR1vB1Ø; …	Normal Bridge Version response shows up first
	… !BR1vA1Ø; …	In the flurry of responses will be an RQ device response such as this

In a mapping exercise, an application program running on the Controller/PC using the Version Global Command would verify all responses for uniqueness, since two or more RQ devices could also have the same addresses. The process of readdressing the Bridge or nodes might require some thought. Two RQ devices with the same address (e.g., XYZ) can be readdressed with the following command:

!XYZ~;   !NXQA;!KT6A;   2 devices addressed XYZ have been randomly readdressed to NXQ and KT6

Note that a node addressed the same as the Bridge cannot be readdressed using the "~" command. In this case, there are at least two options, shown below.

1. Directly readdress the Bridge away from the RQ device of the same address (BR1).
        !BR1@ZØ1;   !ZØ1A;   Bridge acknowledges new address, RQ device doesn't see command

2. Temporarily readdress the Bridge, then either directly readdress the device or have it randomize itself, followed by readdressing the Bridge back to its original address.
        !BR1@TMP;   !TMPA;   Bridge is temporarily out of the way
        !BR1~;   !RNDA;   RQ device reports new randomized address (CØØ – ZZZ)
        !TMP@BR1;   !BR1A;   Bridge acknowledges readdress to original

It is important to note that one Bridge cannot readdress another. In fact, there is no communications possible between Bridges. However, multiple Bridges may be installed on the same RQ Bus and communicate with the same RQ devices on that Bus.

There is no simple way to readdress a collection of RQ devices on an RQ Bus. A custom PC application could be written to aid in such a process.

<u>Application Programming</u>

On the Controller/PC, a typical Home Automation application program will provide ongoing management of an RQ Network.   The application program may also offer some direct manual command control, allowing the end user to intervene in particular situations.

A basic level of control over the network can be obtained with the use of a terminal emulator such as *"Hyperterminal"* on the PC. Running Hyperterminal with the appropriate Serial COM port open, allows character strings to be entered and responses to be viewed (see appropriate Table of Commands below).

Programming languages vary, but a general notion of how a modern programming language might look when controlling RQ devices is shown in the **Representative Programming Example** below. "BUTTON_EVENT" is a function that will be called based on a particular button (1,2, or 3) pressed on an interface unit (dvTP). Since it is determined to be a "PUSH" of the button, the function "SEND_STRING" is called to send to dvShades a literal string such as …
        "!SH1o;"   to open the shade
        "!SH1c;"   to close the shade
        "!SH1s;"   to stop the shade immediately

Programming Home Automation applications for an RQ Network is beyond the scope of this document. Detailed information on interfacing with an RQ Network can be found in the API document *"RQ Application Programming Interface"*.

Representative Programming Example

```
BUTTON_EVENT[dvTP,1]  // Living room Shade Open   // SH1 = Address of shade
{
      PUSH:
      {
            SEND_STRING dvShades,"!SH1o;"
      }
}
BUTTON_EVENT[dvTP,2]  // Living room Shade Close
{
      PUSH:
      {
            SEND_STRING dvShades,"!SH1c;"
      }
}
BUTTON_EVENT[dvTP,3]  // Living room Shade Stop
{
      PUSH:
      {
            SEND_STRING dvShades,"!SH1s;"
      }
}
```


RQ Bridge Table of Commands

Command Character and Description		Direction	# Characters and Description of Data	

p	Set parameter (lowercase "p")	To Bridge	1	One character of parameter
				"N" = Verbose OFF (no unsolicited message or power up msg)
				    Warning: Only turn Verbose OFF in special situations
				      *(see **RQ Communication Messages** above for details)*
				"V" = Verbose ON (unsolicited messages and pwr up version msg)
				"C" = Use <CR> as end message character (uplink only)**
				"S" = Use ";" as end message character (uplink only)
				[responds by echoing msg]

@	Re-address	To Bridge	3	characters (∅-9 or A-Z) [responds with Acknowledge address change message]
N	Assign a name	To Bridge	v[*]	1-16 characters (no "?" for first character) [responds by echoing message]
N	Request the name	To Bridge	1	question mark [responds with Report name message]
N	Report name	From Bridge	v[*]	1-16 characters
v	Request version	To Bridge	1	question mark [responds with Report version message]
v	Report version	From Bridge	3	"B" + 2 characters of version (1∅ = version 1.∅)
U	Undefined / bad message	From Bridge	∅	None
A	Acknowledge address change	From Bridge	∅	None
E	Error	From Bridge	2	Characters describing error "do" = downlink buffer overflow, at least 1 message lost "ml" = uplink message(s) lost due to buffer overflow

* v = variable length message

** <CR> = Carriage Return

RQ60AUMHG Table of Commands

RQ Protocol Summary
V1.Ø

[1] may be an unsolicited message

Command Character and Description		Direction	# Characters and Description of Data	
@	Re-address	to Motor	3	characters (Ø-9 or A-Z) [responds with new address + "A"]
~	Randomize your address	to Motor	Ø	None [responds with new address + "A"]
<	Moving towards ØØ	from motor[1]	2	ØØ-99 = current position
>	Moving towards 99	from motor[1]	2	ØØ-99 = current position
A	Acknowledge address change	from motor	Ø	none
c	Close	to Motor	Ø	None [no response if all the way closed, otherwise responds with current position and direction, unsolicited msg] [later, unsolicited msg with final position]
d	Define a scene	to Motor	v^5	Scene[4], ØØ-99 (%) Scene[4],"NS" to not act on this scene - (minus sign) to clear all scenes [responds by echoing msg]
d	Request scene setting	to Motor	2	Scene[4], question mark [responds with Report scene setting]
d	Report scene setting	from Motor	3	Scene[4], ØØ-99 (%) Scene[4], "NS" not in scene
E	Error	from motor	2	Characters describing error bz = busy nc = not calibrated pu = position unknown ml = message lost (uplink or downlink)
g	Execute scene	to Motor	1	Scene[4] [no response, motor movement will cause unsolicited msg]
i	Identify	To Motor	Ø	none, Flash the LED for 2Ø seconds [no response]
m	Move to position	to Motor	2	ØØ-99 (%) = destination position[3] [responds with current position and direction, unsolicited msg] [later, unsolicited msg with final position] [responds with E command for errors]
N	Assign a name	To Motor	v^5	1-16 characters (no "?" for first character) [responds by echoing msg]
N	Request the name	To Motor	1	question mark [responds with Report name msg]
N	Report name	from motor	v^4	1-16 characters
o	Open	to Motor	Ø	None [no response if all the way open, otherwise responds with current position and direction, unsolicited msg] [later, unsolicited msg with final position]
p	Request parameter (lowercase "p")	To Motor	2	1 character for parameter, then question mark[2] [responds with Report parameter msg]
p	Set parameter (lowercase "p")	To Motor	v^5	1 character for parameter, then appropriate data[2] [see Parameter Table for responses]
p	Report parameter (lowercase "p")	from motor	v^5	1 character for parameter, then appropriate data[2]
r	Request current position	to Motor	1	question mark [responds with Report current position msg] [responds with E command for errors]
r	Report current position	from Motor[1]	2	ØØ-99 (%) = current position[3]
s	Stop	to Motor	Ø	None [no response, motor movement will cause unsolicited msg]
U	Undefined / bad message	from motor	Ø	None
v	Request version	to Motor	1	question mark [responds with Report version msg]
v	Report version	from motor	3	"A" + 2 characters of version (1Ø = version 1.Ø)

[2] see parameter table for parameters and appropriate data below

[3] ∅∅ means at reference (default = open), 99 means at limit away from reference (closed)

[4] means scene number ∅-9, A-Z, a-z

[5] v means variable length message

Setting Parameters

A variety of special motor control Parameters may be set. It is important for the motor to be stopped. Setting Parameters while the motor is moving may cause the position to drift.

The Parameter itself is one character followed by data, if appropriate. Each parameter defines the data that follows, if any. For some parameters, the data is in the hexadecimal format (∅-9, and A-F) that represents four binary "bits" that are each actually Options. The appropriate hex character is built from adding up the Options, which are described as having values such as +1, +2, +4, and +8. The binary value doubles by moving left one bit, starting at the right.

All bits on (i.e., 1111) is decimal value 15 = hex F which is 1+2+4+8. All bits off (∅∅∅∅) is a decimal value of ∅ and also a hex value of ∅ and means that none of the Options are set. If the value of one desired Option is +1 and the value of another is +8, then the two are added for a decimal value of 9 and a hex value of 9 (= binary1∅∅1), which will set just the two Options selected.

As another example, If the value of one desired Option is +2 and the value of another is +8, then the two are added for a decimal value of 1∅ and a hex value of A (= binary1∅1∅), which will set just the two Options selected. All Options for a particular parameter must be considered at one time, since the hex character will turn on/off all four Options without regard for their previous values.

RQ6∅AUMHG Table of Parameters

	Parameter and Description	Direction	# of Data Characters and Description	
R	Resets an RQ device	To motor	1	"D" = Default. Resets all RQ and RP programming to factory default except RP main channel and RQ address and name [no response msg, device blinks red LED once]
T	Request Motor Travel Time	To Motor	1	"?" [responds with Report Motor Travel Time]
T	Report Motor Travel Time	from Motor	3	Travel time in seconds If travel time is not known, an "nc" error message is reported
T	Calibrate the motor travel time	To Motor	1	"C" = Calibrate. Calibration starts immediately. Direction and reference must be set BEFORE Calibration [no response]
M	Set Motor Options	To Motor	2	First Hex character options: +1 = not implemented, set it to ∅ +2 = not implemented, set it to ∅ +4 = Fast IR Release Time (similar to C6 O7 button presses, but won't set Momentary Action) +8 = Stop On Transmitter Button Release (C6 O8) Second Hex Character options: +1 = Momentary Motor Action (C6 O1)

				+2 = Reverse Motor Direction (C6 O2)
				+4 = Do NOT Act on *ALL* Buttons from Transmitter (C6 O3)
				+8 = not implemented, set it to 0
				(not implemented values will not affect anything)
				[responds by echoing msg]
M	Request Motor Options	To Motor	1	"?" [responds with Report Motor Options]

M	Report Motor Options	from Motor	2	First Hex character options:
				  +1 = not implemented, can be 0 or 1
				  +2 = not implemented, can be 0 or 1
				  +4 = Fast IR Release Time (equivalent to C6 O7 button presses)
				  +8 = Stop On Transmitter Button Release (C6 O8)
				Second Hex Character options:
				  +1 = Momentary Motor Action (C6 O1)
				  +2 = Reverse Motor Direction (C6 O2)
				  +4 = Do NOT Act on *ALL* Buttons from Transmitter (C6 O3)
				  +8 = not implemented, can be 0 or 1

P	Set Intermediate Position Options
	(uppercase "P")
		To Motor	2	First Hex character options:
				  No options implemented, this character MUST be 0
				Second Hex character options:
				  +1 = not implemented, should be 0
				  +2 = not implemented, should be 0
				  +4 = Set Reference as close limit instead of open limit
				  +8 = Set Highest Accuracy
				[responds by echoing msg]

P	Request Intermediate Position Options (uppercase "P")	To motor	1	"?" [responds with Report Intermediate Position Parameters]

P	Report Intermediate Position Options
	(uppercase "P")
		from Motor	2	First Hex character options:
				  No options implemented, this character will be 0
				Second Hex character options:
				  +1 = Motor is Calibrated
				  +2 = not implemented, may be 0 or 1
				  +4 = Reference is close limit instead of open limit
				  +8 = Highest Accuracy Set