Programming Guide for CNC Control Cards

Programming Guide for CNC Control Cards

This manual produced on 24/04/2014.

Warranty

This software and/or hardware and accompanying documentation are provided 'as-is' and are not warranted to be fit for any specific purpose or usage.

The use of this software and/or hardware is undertaken at your own risk and Conqueror Design and Engineering Limited will not be responsible for any loss of data, time or income resulting from the use of this software and/or hardware.

Warranty

Updates are available from the website.
We recommend getting the latest manual updates when working with the product.

Table of Contents

1 Direct control via the RS232

It is possible to control the Conqueror Design and Engineering CNC control cards directly from the serial port without using the EaziCNC or EaziCNCLite programmes.

The CNC cards accept a subset of the full set of G & M codes (for instance in most cases no circular interpolation, no splines and no canned cycles) that EaziCNC supports, coordinates need to be in millimetres or steps (depending on the card) and only absolute positioning is supported. Also you need to deal with the handshaking from the machine to make sure that commands do not get overwritten before they can be executed.

To test this from, for instance, HyperTerminal...

1. Open a connection to the card. For an MPC4, MPC5, M101 or M641 card the parameters will be 115200 baud, 8 data bits, no parity and 1 stop bit with Xon/Xoff handshaking. For an M100 card the parameters will be 38400 baud, 8 data bits, no parity and 1 stop bit with Xon/Xoff handshaking.
2. Type <CTRL+E>... this will turn on the echo so that you can see the keys typed and the responses from the card. You should also turn on the option to add line-feeds to carriage returns (*N.B. <CTRL+N> turns the echo off... the controller will not send anything back via the RS232 that is not requested... this keeps the interface 'clean' for very rapid communication from a programme like EaziCNC*).
3. If you hit <Enter> you should see the '>' prompt.
4. You can now enter commands such as 'ST' to show the status, 'M3' to turn on the spindle and 'G0 X10' to move (before you do any moves you may need to send an 'EC' to clear the error-state... by default the controller will power-up with error #3 - power-on).
5. If you want to see a rolling display of the coordinate changes as a movement command is executed then enter 'D1' (and 'D0' to turn the feature off).
6. The controller will send status information back to a control programme if certain characters are sent... for instance sending a '@' will return X, Y, Z and E positions, the key panel code and a status byte as a packed hexadecimal string. There are other codes which will return information from the box either as text, packed hexadecimal or binary-blobs.

In the raw 'terminal' mode any axis commands (such as 'X10' or 'Y1.26') will be interpreted as millimetres on an MPC4, MPC5, M101 or M641 card using the parameters to convert them into an accurate number of steps... the coordinates shown by the status commands are also shown in millimetres with 2 decimal places (0.01) but no decimal points are shown. On an M100 card the coordinates are interpreted and shown as numbers of steps. On all cards any of the commands that return data in a packed format return the raw step values.

There are a couple of ways to proceed depending on how sophisticated your control of the machine needs to be... you can either write your own interface to the controller which sends commands out of the serial port direct *or* there is a DLL available which contains a 'virtual'-machine that supports all of the commands that EaziCNC does and that can be easily integrated into any programme. The DLL takes care of all the serial communication to the machine and exposes a command function and a query function to the calling programme. The DLL is an

extra cost option.

2 Communication

The CNC control cards are passive. With the exception of a prompt character '>' sent when they are powered on they will not initiate any communication with the controlling computer.

There are two types of communication with the control card... sending commands and sending queries.

Sending commands...

To send a command a text string is simply sent to the serial port. The string consists of text tokens each of which is one or two letters which may or may not be followed by a number. Each text token must be separated from the next by a <Space> (ASCII 32) character and the whole command is completed with a carriage-return <CR> (ASCII 13). There is no limit to the length of a text line sent but any repeated tokens will simply overwrite the previous values and each token can be no more than 16 characters long.

Example commands...

G0<SPACE>X10<CR>
G1<SPACE>Y10.1<CR>
G1<SPACE>Z4<CR>
X0<CR>

When a command has been sent to the controller then the controller may be busy for some time processing the command. During this time another command can be sent to the controller and will be stored in the internal buffer. The controller can be repeatedly queried to check when it is again free or more commands can be sent and the handshaking can be used to control the flow of commands. If you are using the buffer then it is *extremely* important to respect the Xon/ Xoff handshaking or you will overwrite commands in the buffer before they can be processed.

A number of commands will cause the control to return a text value (such as 'ES' which will return the error code status)... this is not the same as a query command. One of the main differences is that such commands are queued so they will not return any value until the previous commands have been executed.

**Warning* If a command sent to the control causes an error... such as hitting a limit switch... or an error condition occurs for some other reason then commands will be ignored until the error condition is cleared.*

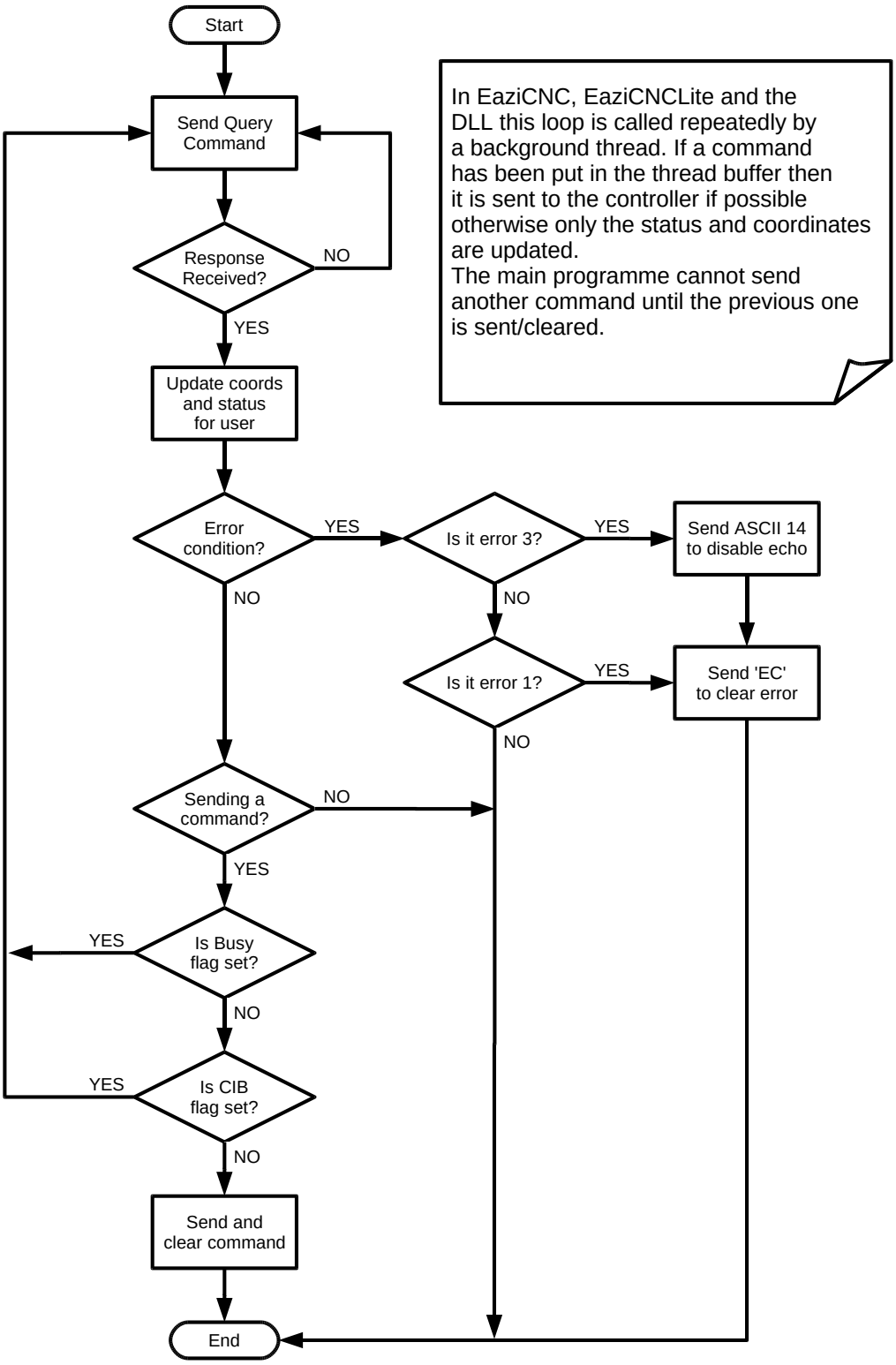
Sending queries...

Queries are sent using a number of trigger characters (such as '@'). Queries can be sent at any time... even when commands are queued in the buffer... and the response will be immediate.

For instance if the '@' character is sent to the control then it will respond with...

#00000000000000000000000FF03<CR>

2.1 Sample Communication Loop



3 Flags and values returned by the control

Status Byte

The **status byte** returned by many of the query commands consists of...

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CIB	MAN	CQB	BUSY	STOP	EB3	EB2	EB1

The **CIB - Command In Buffer** flag indicates that there is a command in the internal buffer. If the '**BUSY**' flag is not set then the command has not been completed with a carriage return, <CR> ASCII 13, yet. If the '**BUSY**' flag is set then the controller is probably waiting for the previous operation to complete before accepting the command from the buffer.

The **MAN - Manual Mode** flag indicates that the controller is in manual mode and the axes are being controlled using the jog-buttons or electronic hand-wheel (if supported)... it is not possible to send movement commands to the controller while it is in manual mode. Manual mode can be exited either using the buttons or by sending an <Esc> - Escape command.

The **CQB - Command Queued** flag indicates that a vector queued command is waiting to be executed after the current command completes. The flag indicates that at least one queued command is waiting... there may be more than one depending on the controller model.

The **BUSY** flag indicates that the control is busy executing a command. If the **CIB - Command in Buffer** flag is not set too then the next command can be queued for execution when the current command completes. Depending on the controller it may be possible to queue several commands.

The **STOP - Stop Activated** flag indicates that the emergency stop circuit on the card is activated... many of the commands will not operate if the emergency stop circuit is activated.

The low 3 bits, **EB3**, **EB2** and **EB1** are the current error code. The error codes are...

0 No error
1 Stopped
2 Limit Error
3 Power On
4 Command Error
5 Feedback Error

Key-code Byte

The key-code byte returned by the query commands consists of...

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
FB2	FB1	BN2	BN1	KC4	KC3	KC2	KC1

FB2 and **FB1** are the optical encoders used for thread synchronisation (if fitted).

BN1 and **BN2** (if fitted) are two front panel push-buttons.

The lower 4-bits, **KC4**, **KC3**, **KC2** and **KC1** are the key-code of the axis jog-button pressed.

The buttons are active low so if nothing is pressed the value will be 0xFF.

Limit Switch Status

The limit switch status byte returned by the query commands consists of...

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	1	Z- Limit	Z+ Limit	Y- Limit	Y+ Limit	X- Limit	Z+ Limit

Other limit switches are returned as key-codes in the key-code byte.

Relay Status

The relay status byte returned by the query commands consists of...

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Coolant On	Relay 3 On	Relay 2 On	Relay 1 On	Spindle On	Relay 4 On	Motors On	Direction

4 Useful Tips

If you are writing a programme to communicate with the control directly then here are a number of tips...

1. It is probably advisable to turn off the echoing of commands and characters from the control. Do this by sending a <CTRL+N> character (ASCII 14) to the control. If you do not do this then you might find spurious characters in the middle of any response packet when you query the control.
2. Do not query the control too frequently... you must let a suitable amount of time elapse between queries even if you think the query was lost. If you do not then you run the risk of stalling the control while multiple queries are answered. Typically you should query an MPC4, MPC5, M101 or M641 no more than 300 times a second and an M100 control no more than 100 times a second.
3. If you want to display coordinates on the screen or monitor the status of the controller then it is not advisable to use the Xon/Xoff handshaking to control the flow of commands. Instead poll the control using the query commands to find out when the next command can be sent.
4. EaziCNC and EaziCNCLite provide a continuous display of the CNC card state on the controlling computer and depending on the application you are using the control card for you will probably need to implement some form of coordinate display in your control programme.
5. EaziCNC and EaziCNCLite monitor the card continuously and deal with simple error codes (like power on) automatically. They also display other errors so that the user can deal with them and knows what state the machine is in.
6. If using the query commands it is best to disable the Xon/Xoff handshaking in the control computer otherwise it is possible that the flow control may stop the answer to a query command being received.
7. Be careful not to send stray coordinate commands (X, Y, Z, U, V and W tokens)... the G-code movement commands are modal and a change in coordinates is interpreted as a new movement command.
8. With the M100 controller use the G1 command for all positioning and specify a high feed-rate for rapid moves.
9. Do not send undefined tokens to the control... just because they are not listed here does not mean that they do not have a function on the control card.
10. If you need to use tool offsets then you should keep track of them within your programme and add or subtract the relevant values from the coordinates you send to the control card. Most of the control cards have support for tool offsets but explaining how to use them is beyond the scope of this document. If you can't work how to do it for yourself then use the DLL!
11. When the control is powered on it is in an error state (error code #3)... you will need to clear the error code (using EC - Error Clear) before you can execute any movement commands on the controller. If you are turning off the echoing of characters then you need to reset this after a power-on.
12. EaziCNC, EaziCNCLite and the DLL use a separate processor thread for the communication loop to the control card and the status display functions so that they can still operate when the main programme loop is blocked waiting to transfer commands to the controller.

5 Commands interpreted by the control

The following is a list of the commands supported by the control cards... do not confuse this with a list of the commands supported by EaziCNC/EaziCNCLite or the DLL... this is a list of JUST the commands that are supported directly by the control cards.

The valid text tokens in a command are...

A	X spline point 1
B	Y spline point 1
C	Z spline point 1
D	Display command or U spline point 1
EC	Error Clear
ES	Error Status
F	Feed Command
G	G-commands G0, G1, G5, etc.
I	Information Command or X centre of circle
J	Y centre of circle or Y spline point 2
K	Z centre of circle or Z spline point 2
L	U spline point 2
M	M-commands M3, M4, M5, etc.
MA	Manual Mode
P	Parameter Command
R	parameter value
ST	Status Command
SU	Set U
SV	Set V
SW	Set W
SX	Set X
SY	Set Y
SZ	Set Z
T	Tool Offset
U	coordinate (4th Axis)
V	coordinate (5th Axis)
W	coordinate (6th Axis)
X	coordinate (1st Axis)
Y	coordinate (2nd Axis)
Z	coordinate (3rd Axis)

....many tokens are only used if they given in a command that requires them. Any extra tokens included with a command that does not require them are simply ignored.

5.1 <CTRL+B> - 34 Char Hex Query

Sending a <CTRL+B> (ASCII 2) character to the control card will return a 34 character response string...

#00000000000000000000000000FF00<CR>

The format of the response string is...

#XXXXXXYYYYYYZZZZZZUUUUUUSSRRKKFF<CR>

....where

XXXXXX Is 6 hex digits representing a 24-bit number which is the X position in steps

YYYYYY Is 6 hex digits representing a 24-bit number which is the Y position in steps

ZZZZZZ Is 6 hex digits representing a 24-bit number which is the Z position in steps

UUUUUU Is 6 hex digits representing a 24-bit number which is the U position in steps

SS Is 2 hex digits representing an 8-bit number which is the spindle speed programmed (as a value from 0 to 255)

RR is 2 hex digits giving the relay status

KK Is 2 hex digits giving the current key code

FF Is 2 hex digits giving the current flags

<CR> Is a carriage return (ASCII 13)

Supported by all control cards.

5.2 <CTRL+C> - Flags and Buffer Count

Sending a <CTRL+C> (ASCII 3) character to the control card will return a 6 character response string...

&0000<CR>

The format of the response string is...

#FFBB<CR>

....where

FF	Is 2 hex digits giving the current flags
BB	Is 2 hex digits giving the number of commands queued in the command buffer
<CR>	Is a carriage return (ASCII 13)

Supported by all control cards..

5.3 <CTRL+D> - 19-Byte Binary Query

Sending a <CTRL+D> (ASCII 4) character to the control card will return a 19 byte packet response in binary.

The format of the response packet is...

<b0>	Start byte (always 15)
<b1><b2><b3>	24-bit signed number which is the X position in steps
<b4><b5><b6>	24-bit signed number which is the Y position in steps
<b7><b8><b9>	24-bit signed number which is the Z position in steps
<b10><b11><b12>	24-bit signed number which is the U position in steps
<b13>	8-bit number which is the spindle speed programmed (as a value from 0 to 255)
<b14>	relay status
<b15>	current key code
<b16>	current flags
<b17>	limit switch status
<b18>	8-bit checksum

The checksum byte is the 2s complement of the sum of the other bytes.

Supported by all control cards.

5.4 <CTRL+E> - Echo

Enables the echoing of all characters sent to the control. This is useful if the control card is to be controlled using a terminal programme.

This command also enables the output of 'human-friendly' characters on the responses sent by the control card.

Supported by all control cards.

5.5 <CTRL+N> - Echo Off

Disables the echoing of all characters sent to the control.

This command also disables the output of 'human-friendly' characters on the responses sent by the control card.

Supported by all control cards.

5.6 <CTRL+P> - 26-Byte or 59-Byte Binary Query

MPCx, M640, M641 with V1 firmware

Sending a <CTRL+P> (ASCII 16) character to the control card will return a 26 byte packet response in binary.

The format of the response packet is...

<b0>	Start byte (always 15)
<b1><b2><b3>	24-bit signed number which is the X position in steps
<b4><b5><b6>	24-bit signed number which is the Y position in steps
<b7><b8><b9>	24-bit signed number which is the Z position in steps
<b10><b11><b12>	24-bit signed number which is the U position in steps
<b13><b14><b15>	24-bit signed number which is the V position in steps
<b16><b17><b18>	24-bit signed number which is the W position in steps
<b19>	8-bit number which is the spindle speed programmed (as a value from 0 to 255)
<b20>	relay status
<b21>	current key code
<b22>	current flags
<b23>	active axis for hand-wheel control
<b24>	limit switch status
<b25>	8-bit checksum

The checksum byte is the 2s complement of the sum of the other bytes.

Only supported by control cards which support 6 axes.

M641 with V2 firmware, M401 and X-series (if the card operates in 6-axis mode with EaziCNC 2 then it uses this format)

Sending a <CTRL+P> (ASCII 16) character to the control card will return a 59 byte packet response in binary.

The format of the response packet is...

<b0>	Start byte (always 15)
<b1><b2><b3>	24-bit signed number which is the X position in steps
<b4><b5><b6>	24-bit signed number which is the Y position in steps
<b7><b8><b9>	24-bit signed number which is the Z position in steps
<b10><b11><b12>	24-bit signed number which is the U position in steps
<b13><b14><b15>	24-bit signed number which is the V position in steps
<b16><b17><b18>	24-bit signed number which is the W position in steps
<b19><b20><b21>	24-bit signed number which is the X encoder value
<b22><b23><b24>	24-bit signed number which is the Y encoder value
<b25><b26><b27>	24-bit signed number which is the Z encoder value
<b28><b29><b30>	24-bit signed number which is the U encoder value
<b31><b32><b33>	24-bit signed number which is the V encoder value
<b34><b35><b36>	24-bit signed number which is the W encoder value

<b37><b38><b39><b40> 32-bit encoder #6 value (usually the hand-wheel count)
<b41><b42><b43><b44> 32-bit encoder #7 value (varies from card to card)
<b45><b46><b47><b48> 32-bit 2nd processor tick-count
<b49><b50> 16-bit 2nd processor communications failure count
<b51> feed-rate override setting (is in 5% increments so 20 is 100%/ default)
<b52> 8-bit number which is the spindle speed programmed (as a value from 0 to 255)
<b53> relay status
<b54> current key code
<b55> current flags
<b56> active axis for hand-wheel control
<b57> limit switch status
<b58> 8-bit checksum

5.7 <Esc> - Escape Command

Sending an <Esc> character (ASCII 27) to the control will cancel any outstanding commands, halt any movement and also set the error condition to #1 - Stopped.

In order to be sure that the control is ready for new commands the <Escape> command can be sent to clear any existing activity.

Supported by all control cards.

5.8 @ - 30 Char Hex Query

Sending an '@' (ASCII 64) character to the control card will return a 30 character response string...

#000000000000000000000000FF00<CR>

The format of the response string is...

#XXXXXXYYYYYYZZZZZZUUUUUUKKFF<CR>

....where

XXXXXX	Is 6 hex digits representing a 24-bit number which is the X position in steps
YYYYYY	Is 6 hex digits representing a 24-bit number which is the Y position in steps
ZZZZZZ	Is 6 hex digits representing a 24-bit number which is the Z position in steps
UUUUUU	Is 6 hex digits representing a 24-bit number which is the U position in steps
KK	Is 2 hex digits giving the current key code
FF	Is 2 hex digits giving the current flags
<CR>	Is a carriage return (ASCII 13)

Supported by all control cards.

5.9 D - Display Command

The D - display command selects whether or not coordinates will be continuously output when the control card is operating.

'D0<CR>' will turn output off and 'D1<CR>' will turn output on.

Supported by all control cards.

5.10 EC - Error Clear

The **EC - Error Clear** command clears any software controllable error... it will not clear hardware errors such as an emergency stop activation or limit switch error.

Supported by all control cards.

5.11 ES - Error Status

The **ES - Error Status** command returns the current error code.

The error codes are...

0	No error
1	Stopped
2	Limit Error
3	Power On
4	Command Error
5	Feedback Error

Supported by all control cards.

5.12 F - Feed Command

The **F - Feed Command** sets the feed rate for movement commands. It also sets the current feed for manual movement using the jog-buttons.

For the M100 controller the feed is set in steps-per-second but for all others controllers it is set in millimetres-per-minute.

Supported by all control cards.

5.13 FF - Fast Feed

The **FF - Fast Feed** command sets the rapid feed rate in steps/sec for 'G0' movement commands on an M100 controller.

The M100 controller has no non-volatile memory so no parameters are stored between power-ups. The default step rate for rapid moves is 2750 steps/sec which is the maximum rate... we recommend not using the G0 command on the M100 but instead using a G1 command with a high feed rate.

Only supported by the M100 control card.

5.14 G0 - Rapid Move

The **G0 - Rapid Move** command will move the axes to the new programmed position at the rapid feed rate.

Examples...

G0 X10 Y0.5 Z-2<CR>
G0 X0<CR>
G0 U-0.01<CR>

Movements occurring in several axis at the same time are linear interpolated so that the moves all start and complete at the same time... the speed of the move is controlled by how long it takes to make the largest axis movement at the rapid feed rate.

For the M100 controller the coordinates need to be given in whole numbers of steps. For all other controllers the coordinates are given in millimetres and are converted according to the settings of the control card parameters.

Supported by all control cards.

5.15 G1 - Feed Move

The **G1 - Feed Move** command will move the axes to the new programmed position at the programmed feed rate. The feed rate can be set before the command or a new feed rate can be set as part of the command. Once set the feed rate remains set until changed.

Examples...

G1 X10 Y0.5 Z-2 F100<CR>
G1 X0<CR>
G1 U-0.01 F10<CR>

Movements occurring in several axis at the same time are linear interpolated so that the moves all start and complete at the same time... the speed of the move is controlled by how long it takes to make the largest axis movement at the selected feed rate.

For the M100 controller the coordinates need to be given in whole numbers of steps. For all other controllers the coordinates are given in millimetres and are converted according to the settings of the control card parameters.

Supported by all control cards.

5.16 G5 - Queued Mode

The **G5 - Queued Mode** will turn on or off vectored command queuing.

Vectored command queuing allows an extra level of command queuing so that, for instance, circular or spline moves that are being approximated by a series of short linear interpolation moves will be smoother on the controller. It also turns off the acceleration and deceleration between consecutive moves when there is a movement command waiting in the queue and because of this it is only suitable for sequences of moves that do not include a direct reversal of direction on an axis.

To turn on queued mode 'G5 P1' is used and to turn it off 'G5 P0'.

N.B. If you need circular interpolation or splines then we highly recommend using one of our control cards (M641A) which supports that directly or using EaziCNC/ EaziCNCLite or the DLL.

Supported by all control cards.

5.17 G29 - Home Axes

The **G29 - Home Axes** command will drive each axis for which a coordinate is given to the limit switch and set the datum to the programmed home position.

Example...

G29 X0 Y0 Z0<CR>

Only supported by control cards which have homing support.

5.18 G33 - Sychronised Move

The **G33 - Synchronised Move** command will move the axes to the new programmed position in-sync with the spindle. If the spindle speed varies during the move then so will the speed of the movement of the axes. This is intended primarily for thread cutting on lathes but can also be used with milling machines for tapping.

Examples...

G1 Z10 P1.25<CR>
G1 X1 Z10 P1<CR>

Only supported by control cards which have spindle synchronisation support.

5.19 G54 - Set Home Position

The **G54 - Set Home Position** command will set the home position for use with the G29 - Home Axes command.

Example...

G54 X105 Y150 Z200<CR>

Only supported by control cards which have homing support.

5.20 G92 - Set Datum

The **G92 - Set Datum** command sets the current axis coordinates to the coordinates given.

Example...

G92 X0 Y0 Z0<CR>

For the M100 controller the coordinates need to be given in whole numbers of steps. For all other controllers the coordinates are given in millimetres and are converted according to the settings of the control card parameters.

The same result can also be achieved using the SX - Set X, SY - Set Y, SZ - Set Z, SU - Set U , SV - Set V and SW - Set W commands. One advantage of G92 is that multiple axes datum can be set at the same time.

Supported by all control cards.

5.21 I - Information Command

The **I - Information Command** returns information about the control card.

I0 returns the card name and revision code, I1 returns the firmware version number, I2 returns the firmware date and I3 returns the processor ID.

Supported by all control cards.

5.22 M3 - Spindle On CW

The **M3 - Spindle On CW** turns the spindle on clockwise.

The spindle speed can be given as the 'S' parameter. For an M100 card this is a value from 0-255 and from other cards it is a spindle speed in RPM which is translated according to the parameter settings.

Even on cards with no spindle relay or no spindle speed support this still updates all of the flags on the card.

Examples...

M3<CR>
M3 S200<CR>
M3 S20000<CR>

Supported by all control cards.

5.23 M4 - Spindle On CCW

The **M4 - Spindle On CCW** turns the spindle on counter-clockwise.

The spindle speed can be given as the 'S' parameter. For an M100 card this is a value from 0-255 and from other cards it is a spindle speed in RPM which is translated according to the parameter settings.

Even on cards with no spindle relay or no spindle speed support this still updates all of the flags on the card.

Examples...

M4<CR>
M4 S200<CR>
M4 S20000<CR>

Supported by all control cards.

5.24 M5 - Spindle Off

The **M5 - Spindle Off** turns the spindle off.

Even on cards with no spindle relay or no spindle speed support this still updates all of the flags on the card.

Example...

M5<CR>

Supported by all control cards.

5.25 M6 - Change Tool

The **M6 - Change Tool** command changes the tool offset that is being used to modify the tool position.

Some control cards have a tool offset table in which offsets in X, Y and Z for tools can be stored.

The tool offset being used can be changed using this command.

Example...

M6 T2<CR>

Only supported by control cards which have a tool offset table in non-volatile RAM.

5.26 M8 - Coolant On

The **M8 - Coolant On** turns the coolant relay on.

Even on cards with no coolant relay this still updates the flag on the card.

Example...

M8<CR>

Supported by all control cards.

5.27 M9 - Coolant Off

The **M8 - Coolant Off** turns the coolant relay off.

Even on cards with no coolant relay this still updates the flag on the card.

Example...

M9<CR>

Supported by all control cards.

5.28 M2/M30 - Programme End

The **M2 - Programme Stop** or **M30 - Programme End** command will not stop execution of any following commands but they DO shut down the spindle, the coolant and any relays that have been turned on with the M90 command.

Examples...

M2<CR>
M30<CR>

Supported by all control cards.

5.29 M90 - Relay On

The **M90 - Relay On** turns an auxiliary relay on.

The relay number is given as parameter 'P'.

Even on cards with no relays this still updates the flag on the card.

Examples...

M90 P5<CR>
M90 P6<CR>

Supported by all control cards.

5.30 M91 - Relay Off

The **M91 - Relay Off** turns an auxiliary relay off.

The relay number is given as parameter 'P'.

Even on cards with no relays this still updates the flag on the card.

Examples...

M91 P7<CR>
M91 P8<CR>

Supported by all control cards.

5.31 M98 - Motors On

The **M98 - Motors On** turns the stepper motor enable signal on.

Example...

M98<CR>

Supported by all control cards.

5.32 M99 - Motors Off

The **M99 - Motors Off** turns the stepper motor enable signal off.

Example...

M99<CR>

Supported by all control cards.

5.33 MA - Manual Mode

The MA - Manual Mode command will place the controller into manual mode.

In manual mode the axes are controlled by the jog-buttons and the electronics hand-wheel (if fitted).

Example...
MA<CR>

Supported by all control cards.

5.34 P - Parameter Command

The **P - Parameter Command** sets or returns the value of the control card parameters.

If a parameter value is given as 'R' then the parameter is set to that value otherwise the existing value is returned.

If 'P99' is sent then parameters 0-19 are returned, if 'P199' is sent then parameters 20-39 are returned and if 'P299' is sent then parameters 40-59 are returned.

Examples...

P5 R600<CR>
P4<CR>

Supported by all control cards.

5.35 ST - Status Command

The **ST - Status Command** returns a text string with the current state of the control card.

The main purpose of this command is to allow the control card to be used and setup with a simple terminal application... it is unlikely to be of any use when writing your own control programme because the format of the output varies from control card to control card.

Example...

ST<CR>

Supported by all control cards.

5.36 SU - Set U

The **SU - Set U** command sets the current axis coordinates to the coordinates given.

Example...

SU14<CR>

For the M100 controller the coordinates need to be given in whole numbers of steps. For all other controllers the coordinates are given in millimetres and are converted according to the settings of the control card parameters.

The same result can also be achieved using the G92 - Set Datum command. One advantage of G92 is that multiple axes datum can be set at the same time.

Supported by all control cards.

5.37 SV - Set V

The **SV - Set V** command sets the current axis coordinates to the coordinates given.

Example...

SV-10<CR>

For the M100 controller the coordinates need to be given in whole numbers of steps. For all other controllers the coordinates are given in millimetres and are converted according to the settings of the control card parameters.

The same result can also be achieved using the G92 - Set Datum command. One advantage of G92 is that multiple axes datum can be set at the same time.

Supported by all control cards.

5.38 SW - Set W

The **SW - Set W** command sets the current axis coordinates to the coordinates given.

Example...

SW0.25<CR>

For the M100 controller the coordinates need to be given in whole numbers of steps. For all other controllers the coordinates are given in millimetres and are converted according to the settings of the control card parameters.

The same result can also be achieved using the G92 - Set Datum command. One advantage of G92 is that multiple axes datum can be set at the same time.

Supported by all control cards.

5.39 SX - Set X

The **SX - Set X** command sets the current axis coordinates to the coordinates given.

Example...

SX10<CR>

For the M100 controller the coordinates need to be given in whole numbers of steps. For all other controllers the coordinates are given in millimetres and are converted according to the settings of the control card parameters.

The same result can also be achieved using the G92 - Set Datum command. One advantage of G92 is that multiple axes datum can be set at the same time.

Supported by all control cards.

5.40 SY - Set Y

The **SY - Set Y** command sets the current axis coordinates to the coordinates given.

Example...

SY4.5<CR>

For the M100 controller the coordinates need to be given in whole numbers of steps. For all other controllers the coordinates are given in millimetres and are converted according to the settings of the control card parameters.

The same result can also be achieved using the G92 - Set Datum command. One advantage of G92 is that multiple axes datum can be set at the same time.

Supported by all control cards.

5.41 SZ - Set Z

The **SZ - Set Z** command sets the current axis coordinates to the coordinates given.

Example...

SZ3<CR>

For the M100 controller the coordinates need to be given in whole numbers of steps. For all other controllers the coordinates are given in millimetres and are converted according to the settings of the control card parameters.

The same result can also be achieved using the G92 - Set Datum command. One advantage of G92 is that multiple axes datum can be set at the same time.

Supported by all control cards.

5.42 T - Tool Offset

The **T - Tool Offset** command allows a tool offset to be set or viewed.

Some control cards have a tool offset table in which offsets in X, Y and Z for tools can be stored. If this command is given with an X, Y or Z coordinate then the tool offset in the table will be updated. If the command is given with no coordinates then the current offset values from the table are returned. If the command is given as 'T99' then the entire tool offset table is returned (this can be used to determine how many tool offsets are supported in the table).

Tool 0 will always have 0 offsets and is the base tool from which all offsets are taken.

Examples...

T1 X-1 Y0 Z0<CR>
T1<CR>
T99<CR>

Only supported by control cards which have a tool offset table in non-volatile RAM.

6 DLL programming

The DLL ('EaziDLL.dll' allows the more advanced circular interpolation and spline functions to be used with older controls.

The DLL also deals with the background communications with the control so it may be easier to implement than programming directly for the RS232 port.

The DLL functions (as of DLL version 2.01) are..

Delphi

```
function EZVersion:PAnsiChar; stdcall; external 'EaziDLL.dll'
procedure EZInit; stdcall; external 'EaziDLL.dll'
function EZStart(com:Integer; bb:Integer; vw:Boolean; raw:Boolean):Integer; stdcall; external
'EaziDLL.dll'
procedure EZStop; stdcall; external 'EaziDLL.dll'
function EZStatus:Integer; stdcall; external 'EaziDLL.dll'
function EZStatusEx:Integer; stdcall; external 'EaziDLL.dll'
function EZGetVal(c:AnsiChar):Double; stdcall; external 'EaziDLL.dll'
function EZGetParam(p:Integer):Integer; stdcall; external 'EaziDLL.dll'
procedure EZSetParam(p:Integer; v:Integer); stdcall; external 'EaziDLL.dll'
function EZGetReply:PAnsiChar; stdcall; external 'EaziDLL.dll'
function EZGetMachineValue(s:PAnsiChar):PAnsiChar; stdcall; external 'EaziDLL.dll'
function EZBusy:Boolean; stdcall; external 'EaziDLL.dll'
function EZQueued:Boolean; stdcall; external 'EaziDLL.dll'
function EZError:Boolean; stdcall; external 'EaziDLL.dll'
function EZLastError:Integer; stdcall; external 'EaziDLL.dll'
function EZLastErrorMessage:PAnsiChar; stdcall; external 'EaziDLL.dll'
procedure EZClearError; stdcall; external 'EaziDLL.dll'
procedure EZSendCommand(c:PAnsiChar); stdcall; external 'EaziDLL.dll'
procedure EZWaitCommand(c:PAnsiChar); stdcall; external 'EaziDLL.dll'
function EZGetMacID:PAnsiChar; stdcall; external 'EaziDLL.dll'
procedure EZInterpret(c:PAnsiChar); stdcall; external 'EaziDLL.dll'
function EZWaitForMachine(flow:Boolean):Integer; stdcall; external 'EaziDLL.dll'
```

MS Visual Studio 2013 / dotNET

```
[DllImport("EaziDLL.dll")] static extern IntPtr EZVersion();
[DllImport("EaziDLL.dll")] static extern void EZInit();
[DllImport("EaziDLL.dll")] static extern int EZStart(Int32 com, int bb, int vw,
int raw);
[DllImport("EaziDLL.dll")] static extern void EZStop();
[DllImport("EaziDLL.dll")] static extern int EZStatus();
[DllImport("EaziDLL.dll")] static extern int EZStatusEx();
[DllImport("EaziDLL.dll")] static extern Double EZGetVal(char c);
[DllImport("EaziDLL.dll")] static extern int EZGetParam(int p);
[DllImport("EaziDLL.dll")] static extern void EZSetParam(int p, int v);
[DllImport("EaziDLL.dll")] static extern IntPtr EZGetReply();
[DllImport("EaziDLL.dll")] static extern IntPtr EZGetMachineValue([MarshalAs(
UnmanagedType.LPStr)] string c);
[DllImport("EaziDLL.dll")] static extern Boolean EZBusy();
[DllImport("EaziDLL.dll")] static extern Boolean EZQueued();
[DllImport("EaziDLL.dll")] static extern Boolean EZError();
[DllImport("EaziDLL.dll")] static extern int EZLastError();
[DllImport("EaziDLL.dll")] static extern IntPtr EZLastErrorMessage();
[DllImport("EaziDLL.dll")] static extern void EZClearError();
[DllImport("EaziDLL.dll")] static extern void EZSendCommand([MarshalAs(
UnmanagedType.LPStr)] string c);
[DllImport("EaziDLL.dll")] static extern void EZWaitCommand([MarshalAs(
UnmanagedType.LPStr)] string c);
[DllImport("EaziDLL.dll")] static extern IntPtr EZGetMacID();
[DllImport("EaziDLL.dll")] static extern void EZInterpret([MarshalAs(UnmanagedType
.LPStr)] string c);
[DllImport("EaziDLL.dll")] static extern int EZWaitForMachine(Boolean flow);
```

6.1 EZVersion

Delphi

function EZVersion:PAnsiChar; stdcall; external 'EaziDLL.dll'

MS Visual Studio 2013 / dotNET

```
[DllImport("EaziDLL.dll")] static extern IntPtr EZVersion();
```

The EZVersion function returns the version and compile time of the DLL.

The returned value is a null-terminated string of ANSI characters (single-byte per character).

If calling this routine from dotNET a generic pointer (IntPtr) is returned and the **Marshal.PtrToStringAnsi()** call should be used to convert this to a useable string.

6.2 EZInit

Delphi

procedure EZInit; stdcall; external 'EaziDLL.dll'

MS Visual Studio 2013 / dotNET

```
[DllImport("EaziDLL.dll")] static extern void EZInit();
```

The EZInit procedure initializes the background thread that is used for communication with the machine through the serial port.

The EZInit procedure should be called before any other routines.

6.3 EZStart

Delphi

function EZStart(com:Integer; bb:Integer; vw:Boolean; raw:Boolean):Integer; stdcall; external 'EaziDLL.dll'

MS Visual Studio 2013 / dotNET

```
[DllImport("EaziDLL.dll")] static extern int EZStart(Int32 com, int bb, int vw,
int raw);
```

The EZStart function initiates communication with the CNC control card.

com is the serial port to be used - for instance to use COM1 com is set to 1. 0 is not a valid port number.

bb is the baud-rate to be used for communication. Earlier cards (such as the M401) use 38400 and later cards use 115200.

vw selects whether or not V&W coordinates are supported in the communications. This also determines whether the feed-back/encoder values are read as it selects between the 26 and 59 byte binary packet formats.

raw selects whether the control card needs coordinate date to be sent in steps or as metric coordinates.

The return value is the status byte (also readable using EZStatus). Return values less than 0 or greater than 255 indicate problems with the serial port, the communication or the machine.

Error code	Error
-1 / xFFFFFFFF	Communication port OK but no control connected
256 / x100	The COM port is blocked or not available
258 / x102	The COM port could not be configured

6.4 EZStop

Delphi

```
procedure EZStop; stdcall; external 'EaziDLL.dll'
```

MS Visual Studio 2013 / dotNET

```
[DllImport("EaziDLL.dll")] static extern void EZStop();
```

EZStop halts the background process that communicates with the control card. EZStop is only usually called as part of closing the program but may be called to change baud-rates or the serial port.

6.5 EZStatus

Delphi

function EZStatus:Integer; stdcall; external 'EaziDLL.dll'

MS Visual Studio 2013 / dotNET

```
[DllImport("EaziDLL.dll")] static extern int EZStatus();
```

The return value is the status byte (also readable using EZStatus). Return values less than 0 or greater than 255 indicate problems with the serial port, the communication or the machine.

Error code	Error
-1 / xFFFFFFFF	Communication port OK but no control connected
256 / x100	The COM port is blocked or not available
258 / x102	The COM port could not be configured

For the other bits/data in the status byte see the section Flags and values returned by the control.

6.6 EZStatusEx

Delphi

function EZStatusEx:Integer; stdcall; external 'EaziDLL.dll'

MS Visual Studio 2013 / dotNET

```
[DllImport("EaziDLL.dll")] static extern int EZStatusEx();
```

The returned values is a 32-bit composite of the 8-bit keyboard, spindle, relay and limit bytes (in low to high order).

For a description of the bit values see the section Flags and values returned by the control.

Please note the exact keyboard mapping varies from control card to control card.

6.7 EZGetVal

Delphi

function EZGetVal(c:AnsiChar):Double; stdcall; external 'EaziDLL.dll'

MS Visual Studio 2013 / dotNET

```
[DllImport("EaziDLL.dll")] static extern Double EZGetVal(char c);
```

This command can be used to read values from the control card and from DLL interpreter. The c parameter is a character index for value - so for instance to read the current X coordinate an 'X' is sent.

The return value is a double (a 64-bit floating point representation). The axis scaling parameters need to be correctly configured for the return value to be correct.

6.8 EZGetParam

Delphi

```
function EZGetParam(p:Integer):Integer; stdcall; external 'EaziDLL.dll'
```

MS Visual Studio 2013 / dotNET

```
[DllImport("EaziDLL.dll")] static extern int EZGetParam(int p);
```

Reads one of the parameters [0..99] that is used to configure the control card and DLL.

N.B. These parameters are the copies in the DLL and **NOT** those in the control card.

EaziCNC reads the parameters during control card configuration and stores a local copy of them (in the '.ini' file). These are then reloaded whenever EaziCNC is loaded.

The DLL initializes with a generic set of parameters. If you do not set the more important parameters to match the values in the control card during startup then the returned coordinates and some of the behaviours of the card will not be correct.

Some control cards store the parameters and in this case the EZGetMachineValue function can be used to read a parameter and then that value set in the DLL. Other control cards, such as the M401, do not store parameters at all and require that the correct parameters are stored and provided to the DLL by the program calling the DLL.

6.9 EZSetParam

Delphi

procedure EZSetParam(p:Integer; v:Integer); stdcall; external 'EaziDLL.dll'

MS Visual Studio 2013 / dotNET

```
[DllImport("EaziDLL.dll")] static extern void EZSetParam(int p, int v);
```

Sets one of the parameters [0..99] that is used to configure the control card and DLL. The p value is the parameter number and v is the parameter value (0..65535).

N.B. These parameters are the copies in the DLL and **NOT** those in the control card.

EaziCNC reads the parameters during control card configuration and stores a local copy of them (in the '.ini' file). These are then reloaded whenever EaziCNC is loaded.

The DLL initializes with a generic set of parameters. If you do not set the more important parameters to match the values in the control card during startup then the returned coordinates and some of the behaviours of the card will not be correct.

Some control cards store the parameters and in this case the EZGetMachineValue function can be used to read a parameter and then that value set in the DLL. Other control cards, such as the M401, do not store parameters at all and require that the correct parameters are stored and provided to the DLL by the program calling the DLL.

6.10 EZGetReply

Delphi

function EZGetReply:PAnsiChar; stdcall; external 'EaziDLL.dll'

MS Visual Studio 2013 / dotNET

```
[DllImport("EaziDLL.dll")] static extern IntPtr EZGetReply();
```

This functions returns any reply that was received from the control card when a command was executed.

Commands prefixed by a '?' when sent to EZSendCommand or EZWaitCommand and commands prefixed by '?$?' when sent to EZInterpret will return a reply from the control.

The returned value is a null-terminated string of ANSI characters (single-byte per character).

If calling this routine from dotNET a generic pointer (IntPtr) is returned and the **Marshal.PtrToStringAnsi()** call should be used to convert this to a useable string.

6.11 EZGetMachineValue

Delphi

function EZGetMachineValue(s:PAnsiChar):PAnsiChar; stdcall; external 'EaziDLL.dll'

MS Visual Studio 2013 / dotNET

```
[DllImport("EaziDLL.dll")] static extern IntPtr EZGetMachineValue([MarshalAs(
UnmanagedType.LPStr)] string c);
```

This function is shorthand for the following...

```
EZWaitForMachine(False);
EZSendCommand(s);
EZWaitForMachine(False);
return EZGetReply();
```

This is most useful, for instance, if a parameter is to be read...

Delphi

```
result := EZGetMachineValue('?P8'#13);
```

dotNet

```
result := Marhall.PtrToStringAnsi.(EZGetMachineValue("?P8\r"));
```

The returned value is a null-terminated string of ANSI characters (single-byte per character).

If calling this routine from dotNET a generic pointer (IntPtr) is returned and the **Marshal.PtrToStringAnsi()** call should be used to convert this to a useable string.

6.12 EZBusy

Delphi

function EZBusy:Boolean; stdcall; external 'EaziDLL.dll'

MS Visual Studio 2013 / dotNET

```
[DllImport("EaziDLL.dll")] static extern Boolean EZBusy();
```

Returns the Busy status. The control/DLL is busy if any of the following is true...

- A command is being processed
- A command is in the queue
- A command is in the serial/command buffer

Monitoring or sampling the EZBusy state before sending a command will effectively wait for a command to complete before another is sent.

Monitoring the EZQueued state allows faster command throughput because the next command can be sent as soon as the control is ready for it.

6.13 EZQueued

Delphi

```
function EZQueued:Boolean; stdcall; external 'EaziDLL.dll'
```

MS Visual Studio 2013 / dotNET

```
[DllImport("EaziDLL.dll")] static extern Boolean EZQueued();
```

Returns the Queued status. The control/DLL is queued if either of the following is true...

- A command is in the queue
- A command is in the serial/command buffer

Monitoring or sampling the EZBusy state before sending a command will effectively wait for a command to complete before another is sent.

Monitoring the EZQueued state allows faster command throughput because the next command can be sent as soon as the control is ready for it.

6.14 EZError

Delphi

function EZError:Boolean; stdcall; external 'EaziDLL.dll'

MS Visual Studio 2013 / dotNET

```
[DllImport("EaziDLL.dll")] static extern Boolean EZError();
```

Returns the Error state of the control/DLL.

If an error occurs during command processing this flag is set and remains set until EZClearError is called.

Some commands cannot be executed when the error flag is set.

6.15 EZLastError

Delphi

function EZLastError:Integer; stdcall; external 'EaziDLL.dll'

MS Visual Studio 2013 / dotNET

```
[DllImport("EaziDLL.dll")] static extern int EZLastError();
```

Return the last error code.

The error code can be retrieved after EZClearError is called so the EZError function needs to be called to check if there is an active error condition.

The error codes are...

0	No error
1	Stopped
2	Limit Error
3	Power On
4	Command Error
5	Feedback Error

6.16 EZLastErrorMessage

Delphi

function EZLastErrorMessage:PAnsiChar; stdcall; external 'EaziDLL.dll'

MS Visual Studio 2013 / dotNET

```
[DllImport("EaziDLL.dll")] static extern IntPtr EZLastErrorMessage();
```

Returns the error message for the last error.

The returned value is a null-terminated string of ANSI characters (single-byte per character).

If calling this routine from dotNET a generic pointer (IntPtr) is returned and the **Marshal.PtrToStringAnsi()** call should be used to convert this to a useable string.

6.17 EZClearError

Delphi

procedure EZClearError; stdcall; external 'EaziDLL.dll'

MS Visual Studio 2013 / dotNET

```
[DllImport("EaziDLL.dll")] static extern void EZClearError();
```

Clears any error state in the control/DLL.

Some error will persist even if cleared - such as a limit error - because the machine needs to be moved off of the limit.

6.18 EZSendCommand

Delphi

procedure EZSendCommand(c:PAnsiChar); stdcall; external 'EaziDLL.dll'

MS Visual Studio 2013 / dotNET

```
[DllImport("EaziDLL.dll")] static extern void EZSendCommand([MarshalAs(
UnmanagedType.LPStr)] string c);
```

EZSendCommand sends commands directly to the control bypassing the G-code interpreter in the DLL.

Commands need to be valid G-code strings with spaces separating the tokens and be terminated with a carriage return (ASCII 13) or the Escape code (ASCII 27).

The Escape code (ASCII 27) is a useful command to send as it will interrupt any command in the interpreter or on the control card and will also resynchronize the control and the interpreter (it is possible for the control and the DLL to get out of synchronization if commands are sent through both EZSendCommand and EZInterpret as the interpreter will be unaware of any changes to the control that did not pass through EZInterpret).

If sending an Escape code it needs to be sent as a null-terminated string and not a single character.

6.19 EZWaitCommand

Delphi

procedure EZWaitCommand(c:PAnsiChar); stdcall; external 'EaziDLL.dll'

MS Visual Studio 2013 / dotNET

```
[DllImport("EaziDLL.dll")] static extern void EZWaitCommand([MarshalAs(
UnmanagedType.LPStr)] string c);
```

EZWaitCommand sends commands directly to the control bypassing the G-code interpreter in the DLL.

Commands need to be valid G-code strings with spaces separating the tokens and be terminated with a carriage return (ASCII 13) or the Escape code (ASCII 27).

The Escape code (ASCII 27) is a useful command to send as it will interrupt any command in the interpreter or on the control card and will also resynchronize the control and the interpreter (it is possible for the control and the DLL to get out of synchronization if commands are sent through both EZSendCommand and EZInterpret as the interpreter will be unaware of any changes to the control that did not pass through EZInterpret).

If sending an Escape code it needs to be sent as a null-terminated string and not a single character.

EZWaitCommand waits for the control/DLL to finish processing the current command so if sending an Escape code it is better to use EZSendCommand.

6.20 EZGetMacID

Delphi

function EZGetMacID:PAnsiChar; stdcall; external 'EaziDLL.dll'

MS Visual Studio 2013 / dotNET

```
[DllImport("EaziDLL.dll")] static extern IntPtr EZGetMacID();
```

Returns the ID string for the the control card connected.

The returned value is a null-terminated string of ANSI characters (single-byte per character).

If calling this routine from dotNET a generic pointer (IntPtr) is returned and the **Marshal.PtrToStringAnsi()** call should be used to convert this to a useable string.

6.21 EZInterpret

Delphi

procedure EZInterpret(c:PAnsiChar); stdcall; external 'EaziDLL.dll'

MS Visual Studio 2013 / dotNET

```
[DllImport("EaziDLL.dll")] static extern void EZInterpret([MarshalAs(UnmanagedType
.LPStr)] string c);
```

EZInterpret processes commands through the G-code interpreter in the DLL. The G-code interpreter can process the arc and spline codes and checks the code before sending it to the control card.

As one of the checks the interpreter will not allows potentially damaging codes such as those that affect parameters and settings so these codes need to be prefixed by '?$?' if they need to be passed through.

Before calling complex codes it is best to call EZClearError as codes such as arcs and splines will not work if the Error flag is set.

6.22 EZWaitForMachine

Delphi

function EZWaitForMachine(flow:Boolean):Integer; stdcall; external 'EaziDLL.dll'

MS Visual Studio 2013 / dotNET

```
[DllImport("EaziDLL.dll")] static extern int EZWaitForMachine(Boolean flow);
```

Waits for the control/DLL to finish processing a command.

The flow parameter determines whether the routine waits for the EZBusy (False) or EZQueued (True) to be clear.

The returned parameter is the number of milliseconds that the routine waited.

7 Version History

24/04/2014 5th release
DLL updated to be dotNET compatible.
DLL documentation added to the programming manual.

02/06/2008 4th release
Information added for the X-series cards, M641/642 with V2 firmware and the M401 card.

06/11/2005 3rd release
Spelling corrections and a slight modification to the sample control loop flowchart.

26/10/2005 2nd release
FF - Fast Feed command added (applies to M100 only)

24/10/2005 Initial release
For M100, M441U, M640, M641, MPC4 and MPC5

Index