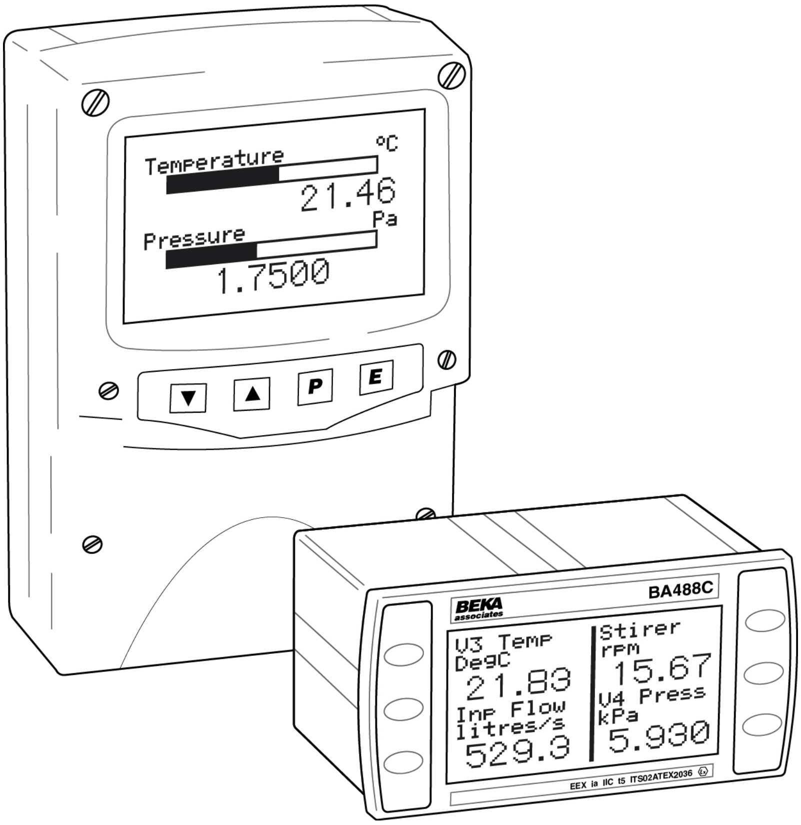
Serial text [Data] display

Programming Guide

[Version 3.4 Firmware]



Issue 7
22 July 2014

This guide applies to the following models:

BA488C - *Panel mounted, Intrinsically Safe*
BA484D - *Field mounted, Intrinsically Safe*
BA688C - *Panel mounted, Safe Area*
BA684D - *Field mounted, Safe Area*

Contents

Introduction

This guide describes the BEKA Mode Protocol for the BA488C, BA484D, BA688C and BA684D Serial Text Displays. This information is only required when programming a host to communicate with these displays; it is not required by the end user. The target audience for this guide are software programmers with some experience in communicating with ASCII devices. As the Modbus protocol is now natively supported (RTU slave only), connection to standard PLCs and industrial computers is greatly simplified.

- For hardware installation information, please refer to the separate instruction manuals available for each model.

- For an overview of how to use these displays on a Modbus system, please refer to the "Serial Text Display - Modbus Interface Guide".

The BEKA protocol is very straightforward, being loosely based on the principals of HTML. Simple text messages can be displayed by using only a handful of commands. However, with a bit more perseverance, some quite advanced displays can be created.

What's in this Programming Guide

- A description of the instrument display
- An overview of the protocol
- Specific information on more advanced features
- A command summary, where the commands are grouped together by function and presented in a series of tables
- A command reference, where each command is listed in alphabetical order and covered in detail. The information is presented in a consistent layout and examples given to demonstrate the use of the command in context.

What's in the Instruction Manuals

- An overview of the instrument
- Intrinsic Safety Certification information
- System Design and Installation
- Configuration
- Programming Overview
- Maintenance

What's in the Modbus Interface Guide

- An overview of the instrument
- A description of the memory map for the Modbus protocol
- A description of the various data types that are used
- Instructions on how to use the instrument in its standard non-programmed modes

Other sources of information

Our website at ***www.beka.co.uk*** has several files available to download:

- All of the examples in this guide
- Demonstration programs showing the capabilities of the display
- A 'Virtual Instrument' – a PC based simulator that behaves exactly like the real thing. This can be used during program development or to demonstrate the application to end users

Enhanced Features

This product is an enhanced version of our original Serial Text display (BA488C / BA484D) . It is compatible with all existing applications as none of the existing commands have been altered.

The following features have been added as standard in the Version 3 firmware, released in July 2005:

Modbus Protocol (RTU Slave Only)	This allows a display to easily be attached to most process control systems and PLCs as a Modbus Slave. Registers are provided to control the display without using the BEKA protocol commands and show the process variables on a series of standard screens. However, more complex applications can be implemented by using BEKA commands over the Modbus protocol. Please see the "Serial Text Display – Modbus Interface Guide" for details.
Cyclic Data & Mapped Variables	For ASCII serial devices without Modbus, a set of commands has been added to allow the standard screens to be easily used. The addition of Mapped Variables within the product makes it much easier to update displayed values automatically, without the need for screen redraws.
Standard Screens	A set of standard screens is included to display up to 8 parameters - no programming is necessary.
Scripting	The display can run through a set of commands independently of the host it is attached to. This allows an application to be loaded into the display only once, leaving the host free to perform other tasks.
Pattern Matching	A fixed format data string from a dumb instrument can be decoded by the display and the results shown on screen. This feature allows a local display to be incorporated anywhere on the installation without using host resources.
Speed Increase	A command has been added (SA) to increase the internal speed of the Intrinsically Safe versions of the display at the expense of a reduction in backlight intensity. Screen redraws are much faster, and more complex pattern matching can be achieved.
Soft fonts storage	There is now the ability to save Font 5 soft fonts.

The following features have been added as standard in the Version 3.4 firmware, released in September 2009:

Integer Variables	Modbus variables can now be input as 16 or 32 bit integers in addition to floating point format. This makes the display easier to interface to simpler devices.
Script String Variables	String variables have been added to make it easier to display operator information.
Standard Screens	Two more standard screens have been added to display 8 parameters - no programming is necessary. The following existing commands are now permitted in standard screen mode: <CE>, <CP>, <OEn>, <ODn>, <SBn>.
Hide Unused Variables	In applications where fewer than 8 variables are required, it is now possible to hide unused values making it more intuitive for the operator.
Status visibility	The ability to show or hide the data status has been added. This is useful when the context of that information has the potential to confuse the operator.
Scripting	Several more scripting commands have been added.

The safe area versions of the product are not limited by Intrinsically Safe power limitations, and will execute commands approximately four times faster. They are also able to communicate with the host at speeds up to 115KBaud via RS485, or up to 19.2KBaud via RS232

After reading through this guide, if you still have a problem getting the results you need then email us at ***support@beka.co.uk*** and we will do our best to help you.

Instrument Features

A detailed overview of the instrument is given in the instruction manual for each product. This should be read before implementing any system using this instrument. However it is useful to summarise the main features of the display before attempting to design any controlling software application.

Display

The instrument display is organised as 120 pixels horizontally by 64 pixels vertically. Each pixel is approximately 0.7mm square which makes it ideal for displaying text and simple graphics. The size of the pixels improves the contrast and hence the readability at greater distances.

The display is also backlit by an ultra-efficient green LED module which enables the screen to be viewed in all conditions, from bright sunlight to total darkness.

Switch Inputs

There are six switches on the front of the panel mounted instrument, and four on the field mounted instrument. Both models have the option of overriding these with up to six external switches which can be sized and labelled to suit the application.

Switch Outputs

There are two switch outputs available, which are under total control of the host. These are totally isolated and can be energised or de-energised independently of each other.

A few words about Modes…

It is worth reviewing the different modes that are referred to in this manual - these can become confusing if taken into the wrong context!

Operational Modes	Refers to the communication protocol between the host and the instrument These can range from Mode 0 (the simplest) to Mode 4 (the most complex). This mode essentially determines how much error checking is applied to the data during transmission. See the Protocol section (Page 9) for a detailed explanation
Row and Pixel Modes	Refers to the way text and graphics are positioned on the screen The simplest and quickest mode is Row Mode – think of it as being able to position objects on a page with ruled lines. In this mode the screen is split up into eight horizontal rows each eight pixels high. Text is then aligned with these rows Pixel Mode allows objects to be placed anywhere – but the drawback is that it takes a bit longer for the display to be updated See the <RM> Row Mode and <PM> Pixel Mode commands for further information
Write Modes	Refers to the way text and graphics are written on the screen Mode 0 is normal : objects appear as a black image on a clear background Mode 3 is inverse : objects appear as a clear image on a black background Modes 1 and 2 are more complex and are used for special effects See the Write Mode section and also the <WM> command
Background Modes	Refers to the image that appears when the screen is flashed A text or graphic object can be flashed against a clear background, a black background or an inverse of that image See the <BM> Background Mode, <FL> Flashing and <EF> Enable Flashing commands
Key Modes	Refers to the format of the key-press data that is returned to the host. Mode 0 is the simplest, where data is returned as a single byte describing the last key pressed. Mode 1 also returns a single byte, but this time individual key status is returned as the six least significant bits of this byte. Mode 2 returns 6 individual bytes showing the status of each key as an ASCII 0 or 1 See the Response format section (Page 10)

The “Command Reference” section (Page 24) shows which modes are applicable to each command.

Display Features

Some powerful features are built into the display that allow relatively complex visual effects to be generated with only a few simple commands. The command reference section of this programming guide has many examples of what can be achieved with a little creativity and lateral thought.

One of the most important concepts to understand is the mechanism of writing to the display.

The display has a foreground and a background. Objects are written to the foreground by sending commands to the instrument. The background is updated automatically, although commands are available to control what is actually written there. These choices are described as the "Background Modes"

When an object needs to be written to the foreground there are a number of choices available that affect the appearance of that object. These choices will also effect what is written to the background, so these choices are described as the "Write Modes"

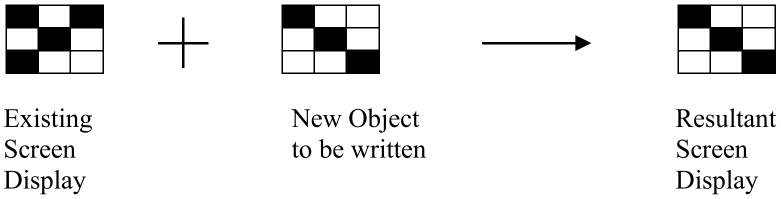
Write Modes

A new object can be added to the screen in four ways, each being associated with a particular Write Mode. However, the write mode is ignored in two cases where it is not considered appropriate, namely restored frames and bargraphs. In these cases, changing the appearance of such items may render them meaningless.

The four modes are:

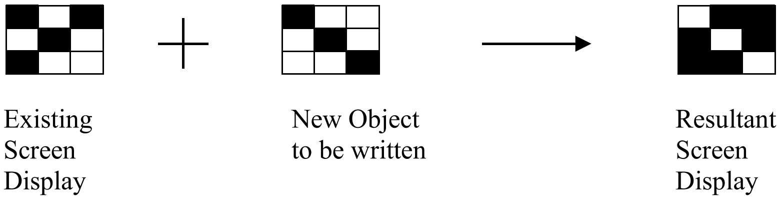
Write Mode 0 is the 'normal' method of updating the screen. The object is written to the screen where it over-writes the current screen contents i.e. if a pixel is set on the new object being written, then the corresponding pixel is set on the screen. If a pixel is not set on the new object, it is cleared on the screen

For example:



Write Mode 3 is almost the same as Mode 0, except that the resulting image is the inverse of the new object. The object is written to the screen where it over-writes the current screen contents i.e. if a pixel is set on the new object being written, then the corresponding pixel is cleared on the screen. If a pixel is not set on the new object, it is set on the screen

For example:

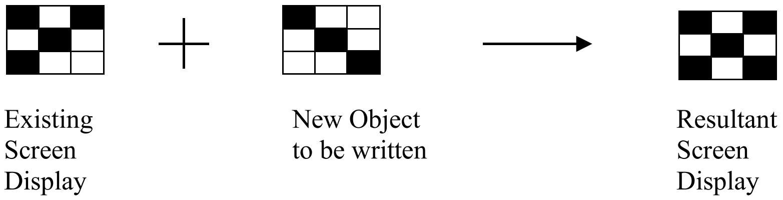


Write Mode 1 is slightly more complex in that the new object is 'ORed' with the existing screen contents i.e. if a pixel is set on the new object being written OR the corresponding pixel is set on the existing screen, then the pixel is set on the screen. The pixel is only ever cleared if both the new object and existing screen are clear.

This can be summarised in a table as follows:

Existing screen display	New Object to be written	Resultant screen display
not set	not set	not set
not set	set	set
set	not set	set
set	set	set

For example:

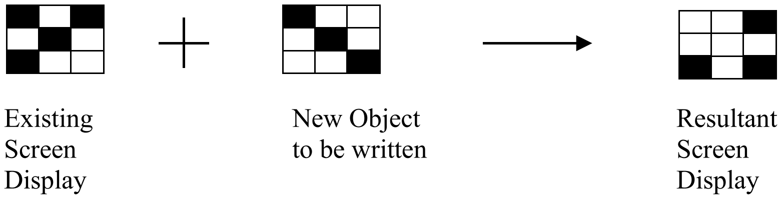


Write Mode 2 is the most complex in that the new object is 'XORed' with the existing screen contents i.e. if a pixel is set on the new object being written OR the corresponding pixel is set on the existing screen, then the pixel is set on the screen BUT if *both* are set then the pixel is cleared. The pixel is also cleared if both the new object and existing screen are clear.

This can be summarised in a table as follows:

Existing screen display	New Object to be written	Resultant screen display
not set	not set	not set
not set	set	set
set	not set	set
set	set	not set

For example:



Background Modes

The background is only ever visible when the screen is set to flash; the foreground image alternates with the background image every second i.e. If the background is clear, then some text on the foreground will disappear and re-appear every second. Alternatively, the background can be made all black. This gives a totally different visual effect which can be more noticeable. However by modifying the background so that it is the inverse of the foreground will make a very eye-catching effect.

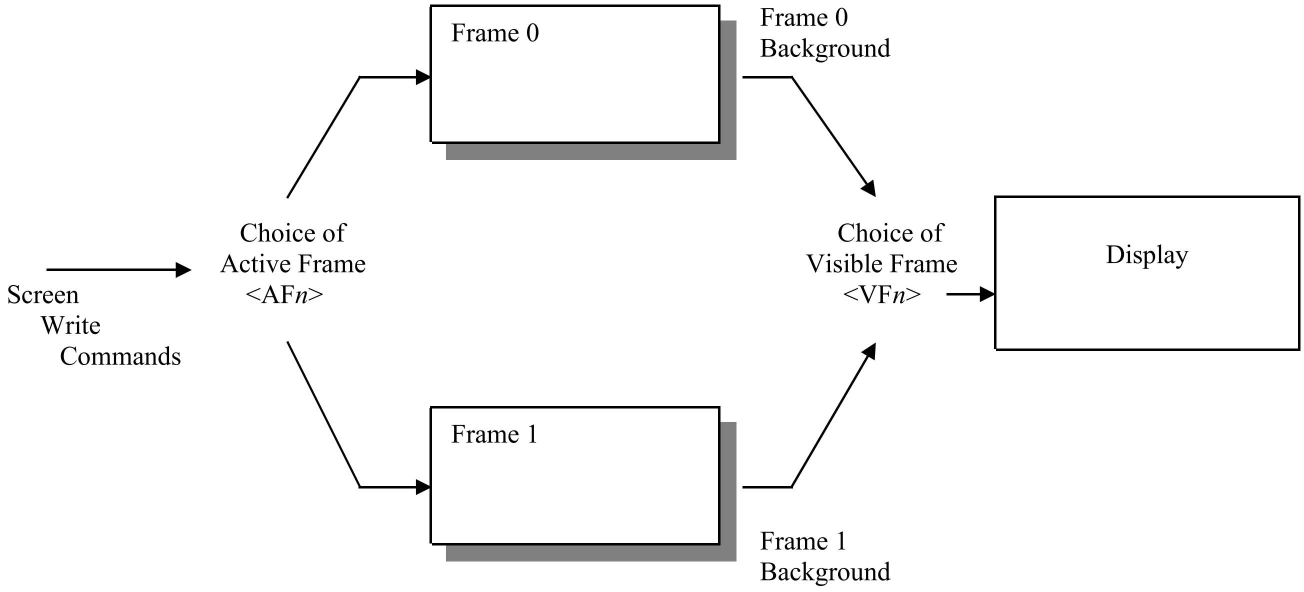
Rather than force the host to do all this work, the background is updated by the instrument automatically. The <BM> Background Mode command is used to control whether the background is clear, black or the inverse of what's written. Once the <BM> command is issued, the background is updated automatically by each new screen object. Therefore it is possible to have all three flashing effects on the screen at once, simply by changing the Background Mode during the construction of the screen.

Frames

Active and Visible Frames

Another concept to grasp is the commands never actually write directly to the screen. Instead there are two “display buffers” that we refer to as ‘Frame 0’ and ‘Frame 1’. Only one of these frames is visible at any time, which is selected by the <VF*n*> Visible Frame command.

Similarly, only one of the frames is “Active” – that is, becomes the destination for all screen write commands. The destination is selected by the <AF*n*> Active Frame command.



Whilst this may seem complex at first, it is actually a very powerful method of displaying one message while building up another screen of data hidden from view. This hidden screen can then be made visible by issuing a single command. This is especially useful where the host cannot sustain a high data rate, or where very complex screens are being generated. As far as the operator is concerned the display updates almost immediately, even though it may have taken several seconds to construct.

For simple applications, frames can be disregarded. The unit powers up with both the Active Frame and Visible Frame set to 0. If the <AF*n*> and <VF*n*> commands are not issued, then the instrument behaves as if screen writes act directly on the display.

Important Note

For brevity this manual simply refers to commands writing to the screen. This has been done to keep the description of each command as simple as possible, so as to convey the main principals of that command. In reality, all commands write to the active frame, with one notable exception, <RL> Restore Logo.

Saved Frame Locations

It is possible to store the screen contents for later use by saving the Visible Frame to memory via the <SF*nm*> Save Frame command. (This command can actually save either frame, but for simplicity disregard this for now)

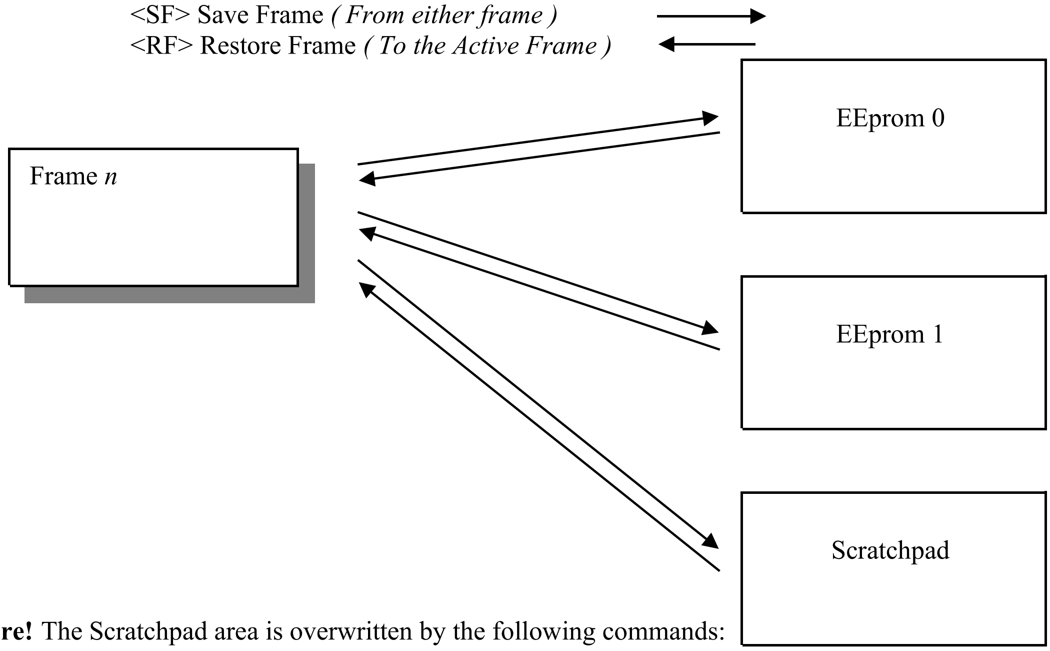
There are two types of memory available for saves:

- Non-volatile EEprom that is retained on power fail and
- A Scratchpad area in RAM that is lost when power is removed from the unit.

It is important to be aware that saves to EEprom take about 3 seconds, whilst saves to the scratchpad are immediate.

There are two independent EEprom locations that may be used as required.

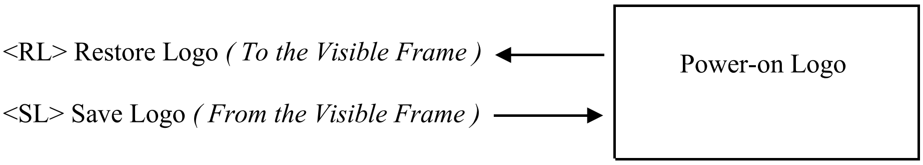
In addition to the two EEprom locations there is a totally separate location, also in EEprom, that is used to store a power-on logo. This is a full screen graphic that appears when the unit is first turned on, or after the unit is re-booted. It is possible to use this area as another storage location, but whatever is in that location will also be used as the logo.


<SF> Save Frame *(From either frame)*
<RF> Restore Frame *(To the Active Frame)*
Frame *n*
EEprom 0
EEprom 1
Scratchpad

Beware! The Scratchpad area is overwritten by the following commands:

<BD>, <DF>, <DG>, <DL>, <LH>, <LV>, <RB>, <RL> & <SL>

Restoring a frame after one of these commands will give unpredictable results.


<RL> Restore Logo *(To the Visible Frame)*
<SL> Save Logo *(From the Visible Frame)*
Power-on Logo

More information can be found in the Command Reference (Page 24) section.

The BEKA Protocol

The BEKA protocol is very loosely based on the principals behind HTML. Fundamentally, the intention is to make the scripts that generate a screen display "human readable", in the same way that the source for a web page may be read.

The main features that achieve this are:

- It is a pure ASCII protocol except for graphics downloads, checksums and CRCs
- Commands are always two characters, case insensitive, enclosed in angled brackets
- All commands are active until overridden by another command
- Some commands require parameters
    - Parameters follow the command directly
    - Multiple parameters are separated by commas
    - Any detected parameter error causes the command to be ignored, and an error is returned
    - A command and its parameters are enclosed within a single set of angled brackets
- No spaces are allowed in commands or parameter strings (except for written text strings)
- Any characters not enclosed in angled brackets are written directly to the screen at the current cursor position, unless error checking is active

Features have been added to maintain the data integrity between host and display. These allow the host to be confident that the display is actually showing valid data that has not become corrupted during transmission. The level of checking is adjustable, depending on the application.

- The unit's response to received messages is programmable. Modes are:
    - No response
    - Response to every correctly formatted command
    - Response to a combination of correctly formatted commands
- Where a response is returned, a user must wait for the response before sending further commands
- Message error checking is programmable. Modes are:
    - No error checking
    - Simple checksum
    - 16bit Cyclic Redundancy Check
- Switch status is encoded into the returned message, or can be explicitly requested

Key presses are latched and are sent back to the host with each message. Once sent, the latches are cleared automatically.

Three Key Modes are available according to the application's requirements:

> Mode 0 is the default, and shows the last key that was pressed.
>
> Mode 1 can be used in applications that need to determine if more than one key has been pressed. It can also be used if the external keys are push-to-break (normally closed).
>
> The major disadvantage is that the returned byte is not human readable using a standard terminal.
>
> Mode 2 is provided to overcome this limitation by transmitting six ASCII bytes, each one representing a key, instead of one binary byte. This is useful for debugging, but the relatively high transmission overhead makes it undesirable in general use.

The Key Mode is configured using the display keypad and in-built menus. Refer to the instruction manual for more information.

In all cases it should be remembered that the key status shows the keys that have been pressed since the last response. To determine whether a key is being pressed at any given time, the application should check the second of two consecutive responses.

Command format

The command format is: <AB[param1],[param2]…,[paramN]>

where:

AB is the command.
[ ] indicates optional parameters separated by comas

example:

<CM4,90>	Cursor Move to Row 4 Column 90
<CS>	Clear Screen

Response format

Key Mode 0 (Default)

The response format is of the form: Ka *or* Ea *or* ?a *or* Xa *or* Sa *or* Ba *or* P0

where:

K indicates that the previous command/command set has been accepted.
E indicates a parameter or communications error has been detected in the previous command string.
? indicates that the command is unrecognised.
X indicates that a communications error has been detected by the hardware e.g. parity or framing.
S indicates that the display is running a script.
B indicates that the display is in a "Busy" state within a script.
"a" returns the key status i.e. the number of the key that was last pressed (1=Key 1, 2=Key 2, ….. 6=Key 6)
P indicates that a message has been received but NOT actioned, as the unit is being configured by a local user.

example:

K0	Command accepted, no keys pressed
E4	Command error detected, key 4 pressed
?6	Command unrecognised, key 6 pressed
P0	Command rejected, unit being configured (Any key presses are discarded)
S3	Command rejected, unit running a script, key 3 pressed
X0	Communication error detected
B0	Script engine in a 'Busy' state

Note: The unit is configured by a local user accessing the configuration menu directly on the instrument front panel. Access to this menu can be denied by issuing the <CP> Configuration Prohibit command.

Key Mode 1

The response format is the same as Mode 0 but the meaning of "a" is modified. In this mode "a" is not a printable ASCII character. Individual key status is returned as the six least significant bits of this byte. The most significant bit of the byte is always set so that the returned data can be distinguished from the Mode 0 data.

In binary notation the format of this returned byte is as follows:

msb							lsb
b7	b6	b5	b4	b3	b2	b1	b0
1	0	0	0	0	0	0	0

b0 represents the status of Key 1 (0 =key open, 1=key closed)
b1 represents the status of Key 2
..
..
b5 represents the status of Key 6
b6 is always cleared (0)
b7 is always set (1)

Key Mode 2

The response format is similar to Mode 1 but the "a" is replaced with six consecutive ASCII characters. It is intended mainly for debugging purposes and hosts with limited processing capability.

In this mode the key press information is returned as 6 individual bytes. If a key has been pressed then the ASCII character "1" (hex 0x31, decimal 49) is returned else ASCII character "0" (hex 0x30, decimal 48) is returned. As the returned data is in ASCII notation a dumb terminal can be used to view the data stream.

Key1 is transmitted first.

example:

K000000	Command accepted, no keys pressed
K100010	Command accepted, Keys 1 and 5 have been pressed
E000100	Command error detected, key 4 pressed
?000001	Command unrecognised, key 6 pressed
P000000	Command rejected, unit being configured (Any key presses are discarded)

Operational Modes

The unit can be configured to expect data in a certain format. These formats are termed "Operational Modes" and range from a simple VDU like mode (0) to a fully error checked mode (4).

The operational mode is configured using the display keypad and in-built menus. Refer to the instruction manual for more information.

The modes are:

Mode 0: Commands are executed immediately, no reply message except when specifically requested. Plain text is written directly to the screen, no reply message.

Mode 1: Commands are executed immediately, a response is returned to each command. Plain text is written directly to the screen, no reply message.

Mode 2: Multiple commands can be sent, but these are not executed until a "Command Implement" <CI> command is sent. One reply is returned for each set of commands. An error in any of the commands will result in a Command Error response. Plain text is ignored.

Mode 3: As Mode 2 but the <CI> command is replaced by a <CCn> command where n is a single byte simple checksum of all characters sent (including spaces) up to, but not including the <CCn> command. The returned command has a similar single byte checksum appended to the end of the response. The command string is not actioned if the checksum of the data received does not match the parameter of the <CCn> command. Plain text is ignored

Mode 4: As Mode 3 but the <CCn> is replaced by <CRnn> where nn is a 16-bit CRC code.

CRC Generation

The 16-bit CRC used in the protocol is the same as used for the well known Modbus Protocol. Details are as follows:

The CRC is started by first preloading a 16-bit register to all 1's. Then a process begins of applying successive eight-bit bytes of the message to the current contents of the register. Only the eight bits of data in each character are used for generating the CRC. Start and stop bits, and the parity bit, do not apply to the CRC.

During generation of the CRC, each eight-bit character is exclusive ORed with the register contents. The result is shifted in the direction of the least significant bit (LSB), with a zero filled into the most significant bit (MSB) position. The LSB is extracted and examined. If the LSB was a 1, the register is then exclusive ORed with a preset, fixed value (A001 hex). If the LSB was a 0, no exclusive OR takes place.

This process is repeated until eight shifts have been performed. After the last (eighth) shift, the next eight-bit character is exclusive ORed with the register's current value, and the process repeats for eight more shifts as described above. The final contents of the register, after all the characters of the message have been applied, is the CRC value.

Generating a CRC

Step 1 Load a 16-bit register with FFFF hex (all 1's). Call this the CRC register.

Step 2 Exclusive OR the first eight-bit byte of the message with the low order byte of the 16-bit CRC register, putting the result in the CRC register.

Step 3 Shift the CRC register one bit to the right (toward the LSB), zero filling the MSB. Extract and examine the LSB.

Step 4 If the LSB is 0, repeat Step 3 (another shift). If the LSB is 1, Exclusive OR the CRC register with the polynomial value A001 hex (1010 0000 0000 0001).

Step 5 Repeat Steps 3 and 4 until eight shifts have been performed. When this is done, a complete eight-bit byte will have been processed.

Step 6 Repeat Steps 2 ... 5 for the next eight-bit byte of the message. Continue doing this until all bytes have been processed.

Result The final contents of the CRC register is the CRC value.

This CRC value is then appended to the message. The LSB of the CRC is sent first followed by the MSB.

Multidrop Operation:

Multidrop operation is possible. A unique unit address between 1 and 247 has to be given to each instrument by using the display keypad and in-built menus. Refer to the instruction manual for more information.

Command <MC*n*> Make Connection is used to define the instrument address (*n*) to which subsequent commands are directed. This “virtual connection” remains in force until it is explicitly released by issuing the <RC> release Connection command. These two commands are necessary in order to confirm that all instruments receive and react to the commands successfully. Once connected, the units respond in exactly the same way as a single units would, with error responses being issued if a problem occurs.

If a unit has a non-zero address, then on power-up, a <MCn> command must be sent prior to any other command, even if it is the only unit on the line.

Graphics Transfers

File Format

In all cases the file format used is a two colour (black and white) bitmap in standard Windows™ / OS2 format. These commonly have a .BMP extension on most PC applications.

Downloads

The protocol is extended as follows to cover the simple graphics download commands <DS> and <DG> and <DF*n*>

To avoid confusion, a download is defined as being from the host to the display

All download functions operate using the same basic sequence:

1) Host sends download command (DS,DG or DF) followed by a <CI>, <CCn>, or <CRnn> command as per the current operational mode
2) Display acknowledges the command with a K0 response
3) Host starts to transmit graphic data within the 2 second timeout period
4) Host transmits a <CI>, <CCn>, or <CRnn> command as per the current operational mode, any check byte(s) being calculated from all of the bytes in the .BMP file
5) Display acknowledges the command with a two byte response

There is a 2 second timeout for the download operation, during which time if no bytes are received the download is aborted and an error response is returned. Of course, the actual total download time depends on the speed of the serial link.

<DS> Download Screen command

The command <DS> is issued with any additional bytes (checksum, CRC etc) as required by the current operational mode. The command is acknowledged if correctly received.

A binary download (from the host to the display) of the graphic file is then expected within the timeout period of 2 seconds. The image must be exactly 120x64 pixels and if not an error response is returned.

After the file has been downloaded the <CI>, <CCn>, or <CRnn> command must be sent as per the current operational mode, the check byte(s) being calculated from all of the bytes in the .BMP file

The download is acknowledged if it was correctly received (including checksum or CRC checks) and the image is displayed. The downloaded image disregards the Write Mode setting and is displayed normally i.e. as though it was preceded by a <WM0> command. In addition, it only adopts the display attributes concerned with Flashing. All other attributes are ignored.

<DG> Download Graphic command

Command <DG> follows exactly the same mechanism as the <DS> command above, but any size of image can be sent up to 120x64. Files in excess of this size will cause an error response.

The display must be in Pixel Mode <PM> and the downloaded image is displayed at the current cursor position.

The image dimensions are computed from the bitmap file that is sent; no parameters are necessary.

The image is drawn upwards and to the right of the current cursor position. If any part of the image exceeds the display bounds the image is NOT displayed and an error response is returned.

The downloaded image adopts the display attributes currently in force (Normal, OR, XOR, Inverse, Flashing, Steady), and the Write Mode setting is taken into account.

Please note:

The <DS> command is just a special case of the <DG> command but because of its fixed size is executed much more quickly.

Graphics can be uploaded to a hidden frame using the <AF> command to select the destination, and the <VF> command to make it visible when complete.

<DF*n*> Download Font command

The display has the capacity of storing four user defined characters for each font size. These "Soft Characters" can then be written onto the screen by using the <WS*n*> command. They may also be underlined and flashed using attributes, as any other character

After the <DFn> command is issued, the display expects a binary download of the soft character. The required image size depends on the currently active font

Font:	F1	Image Size (v x h):	8 x 6 pixels
	F2		16 x 10 pixels
	F3		24 x 15 pixels
	F4		32 x 19 pixels
	F5		48 x 29 pixels.

The image must be exactly as defined above otherwise an error response is returned.

Nothing is drawn to the screen during this command

Uploads

The protocol is also extended to give the facility of obtaining a screen dump from the display. The main use for this is in the preparation of instruction manuals, but it could also be used in a debugging role.

<UE> Upload Enable command

Because a graphic upload generates a significant amount of data, there must be safeguards in place to ensure that the data is really required. The <US> Upload Screen command will therefore not respond unless it is immediately preceded with the <UE> Upload Enable command.

<US> Upload Screen command

The command <US> is issued with any additional bytes (checksum, CRC etc) as required by the current operational mode. The command is acknowledged if correctly received.

After a short delay of 500ms, a 1086 byte block of data is sent to the host.

(This delay is introduced to allow the host to set itself up for data reception).

A command acknowledge then follows with the check bytes as per the current operational mode. The check bytes include the data block bytes and the acknowledge, but not the check bytes themselves.

The 1086 byte data block, once saved to file, is a graphics image of the screen in 2-colour Windows .BMP format

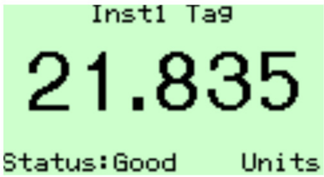
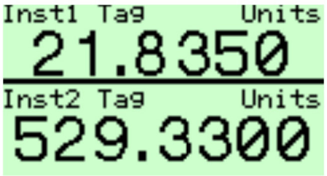
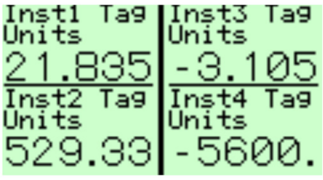
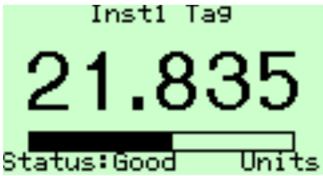
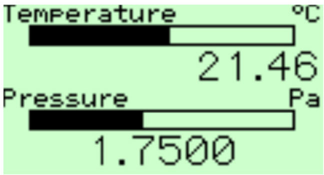
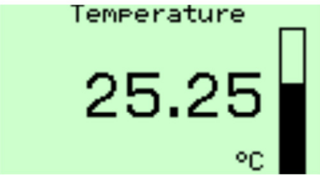
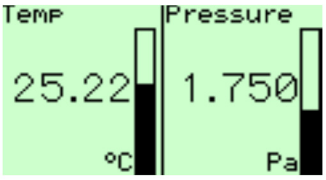
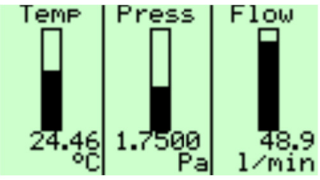
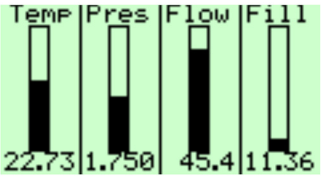
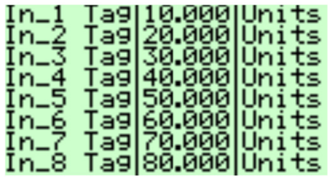
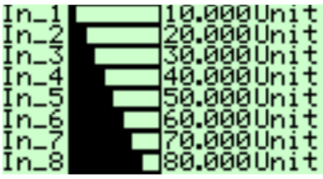
Standard Screens

There are eleven standard screens available which require very little application programming. They are capable of displaying a selection of up to eight process variables, together with their units of measure and tag description. Once a screen format has been chosen, each input variable can be brought into view by pressing the up and down arrow keys.

These standard screens are ideal for many simple applications and can be implemented very quickly. However, where a unique display format is required these can be built up using the commands that can be found later in this Programming Guide.

> *For hosts that support the Modbus protocol, registers can be written to directly, without using any BEKA protocol commands. Please see the "Serial Text Display – Modbus Interface Guide" for further information.*

The screen format is selected by either using the local menu (as described in the Instruction Manual) or by using the <SO*n*> Screen Option BEKA protocol command. One of eleven standard display formats can be selected as shown in the following table:

Screen Option 1	 Inst1 Tag 21.835 Status:Good Units	Screen Option 2	 Inst1 Tag Units 21.8350 Inst2 Tag Units 529.3300
Screen Option 3	 Inst1 Tag Units 21.835 | Inst3 Tag Units -3.105 Inst2 Tag Units 529.33 | Inst4 Tag Units -5600.	Screen Option 4	 Inst1 Tag 21.835 Status:Good Units
Screen Option 5	 Temperature °C 21.46 Pressure Pa 1.7500	Screen Option 6	 Temperature 25.25 °C
Screen Option 7	 Temp 25.22 °C | Pressure 1.750 Pa	Screen Option 8	 Temp 24.46 °C | Press 1.7500 Pa | Flow 48.9 l/min
Screen Option 9	 Temp 22.73 | Pres 1.750 | Flow 45.4 | Fill 11.36	Screen Option 10	 In_1 Tag|10.000|Units In_2 Tag|20.000|Units In_3 Tag|30.000|Units In_4 Tag|40.000|Units In_5 Tag|50.000|Units In_6 Tag|60.000|Units In_7 Tag|70.000|Units In_8 Tag|80.000|Units
Screen Option 11	 In_1 10.000Unit In_2 20.000Unit In_3 30.000Unit In_4 40.000Unit In_5 50.000Unit In_6 60.000Unit In_7 70.000Unit In_8 80.000Unit	Screen Option 0	Custom screens

(Setting Screen Option to Zero <SO0> will allow custom screens to be displayed by using BEKA protocol commands.)

Up to eight process variables can be displayed. Commands are provided to "Map" a numeric ASCII string to one of eight Input Variables IN_1 to IN_8. Each variable can also have an associated "Tag description" and "Units" displayed alongside by using the appropriate commands. The front panel buttons are used to scroll the display between the input variables. Unused variables can be hidden by using the <VL*n*> Variable Last command or the menu option.

The "Status" of the data is used to mark the data **GOOD** or **BAD**. When marked **BAD** the appearance of the value will be in inverse video i.e. clear pixels on a dark background. The host can therefore signal to the operator that a displayed value may be suspect. Screens 1 and 4 also display the text "Status:Good" or "Status:Bad", which may be inappropriate for some applications. It is now possible to hide this extra text by using the <SH*n*> Status Hide command.

The number of decimal places that are displayed is selectable between a value of 0 and 5. A value of 0 will display no decimal places, 1 will display one decimal place and so on. However, a value of 5 will select an automatic mode whereby the maximum number of decimal places is shown, dependant on the available space.

The following list of commands are the only ones necessary to use the standard screens:

Function	Command	Meaning
Send 40 bytes of binary data to update all the input variables with one command. A faster alternative then using the <CV> command multiple times.	<CD>	Cyclic Data
Data can be automatically marked as **BAD** if cyclic data is not received within a set period.	<CTn>	Cyclic-data Timeout
Update a single variable using a numeric ASCII string	<CVn,string>	Cyclic Variable
The number of decimal places to display.	<DDn,m>	Define Decimal
Define upper and lower limits for the bargraphs.	<DLn,m,p>	Define Limits
Assign "Tag" information for each input variable.	<DTn,string>	Define Tag
Assign "Units" information for each input variable.	<DUn,string>	Define Units
Show or hide the data status.	<SHn>	Status Hide
Select the standard screen format to display.	<SOn>	Screen Option
Select which value to display using the current standard screen – equivalent to pressing the buttons	<SVn>	Show Variable
Select the last variable to show when scrolling through the list.	<VLn>	Variable Last

Full details of each command are listed in the "Command Reference" section (Page 24).

Cyclic Data

Basic Concepts

The concept of "Cyclic Data" comes from the Foundation Fieldbus and Profibus protocols. We have developed a range of indicators that use these protocols, and have taken some of their features and brought them to the simpler Serial Text Display product range.

In more complex protocols, an instrument either supplies or takes in data at regular intervals. Depending on the configuration, a group of instruments can make up a small subsystem without very much intervention from a host controller. The Serial Text Display has always been designed for use in a simple Host-Client arrangement, meaning that there has always been a significant amount of work to do by the host. By taking some of the concepts from these more advanced protocols, we have reduced the burden on the hosts to a much simpler level.

Instead of the host having to update all the features on the screen, we have made it possible to use the standard screens by sending data in a fixed format from the host. the programmer has the option of updating each variable one at a time with the <CV> Cyclic Variable command, or all eight at once by using the <CD> Cyclic Data command.

Updating using ASCII values

The <CV> Cyclic Variable command is used to update each variable individually by sending the numeric value as a simple ASCII string.

The syntax is <CV*n*,*string*> where n represents the input variable to be updated (= 1 to 8) and string is the numeric value to be displayed. For example, the command <CV1,123.456> will set the input variable 1 to a value of "123.456". The string is converted to a numeric value and appears on the standard screens, drawing a bargraph if appropriate. The value can be updated when necessary by simply sending the same command with the new value.

Note that there is no mechanism to set the data **GOOD** or **BAD** – all data is assumed to be **GOOD**.

Updating using Binary Data

The <CD> Cyclic Data command is used to update all eight variables at the same time. Instead of an ASCII string, the values are sent as IEEE floating point numbers. In addition, the status of each value can be set **GOOD** or **BAD**. The data format is a single status byte followed by 4-byte float for each variable. The status byte for each value should be set to zero to mark that value as **BAD**, or any value between 1 and 255 to mark it as **GOOD**.

The command <CD> is issued with any additional bytes (checksum, CRC etc) as required by the current operational mode. The command is acknowledged if correctly received.

A binary download (from the host to the display) of the input values is then expected. The length must be (1+4)x8 = 40 bytes and if not an error response is returned.

After the data has been downloaded the <CI>, <CCn>, or <CRnn> command must be sent as per the current operational mode, the check byte(s) being calculated from all 40 bytes of the input data.

The download is acknowledged if it was correctly received (including checksum or CRC checks) and the values are displayed.

There is a 2 second timeout for the download operation, during which time if no bytes are received the download is aborted and an error response is returned. Of course, the actual total download time depends on the speed of the serial link.

Data Timeout

If valid data is not regularly sent to each input variable then there is a risk that an operator might take some inappropriate action in response to stale data. A mechanism has been built into the display that indicates when the data is older than a predetermined timeout value. The data is displayed as **BAD** if the timeout value has been exceeded, but the "Status" remains unchanged. Sending valid data once again to the input variable will display the value normally, marked **GOOD** or **BAD** according to the "Status".

As the acceptable timeout value is totally dependant on the application and the host capabilities, it is adjustable from 10 Seconds to 40 Minutes. For simple applications the timeout value can be set to 0 which disables the facility.

The <CT*n*> Cyclic-data Timeout command is used to set the acceptable timeout value for all eight variables. The duration is set by the value "*n*" which is multiplied by 10 Seconds to give the actual time period.

Mapped Variables

If the standard screens are not suitable for a particular application, the functionality of mapped variables can be used to construct a unique screen which can be automatically updated as new cyclic data is received.

Up to eight process variables can be mapped and displayed on the screen at any one time. The position and appearance of each variable can be independently assigned, and horizontal and vertical bargraphs may be shown alongside numeric values if desired.

The following list of commands are those that are used to place automatically updating variables on user-defined custom screens:

Function	Command	Meaning
Place a horizontal or vertical bargraph on the screen and assign it to one of the eight input variables.	<DBn,m,p,q,r>	Define Bargraph
Place a numeric variable on the screen at the current cursor position in the current font size.	<DVn,m,p,q>	Define Variable
Remove one or all previously defined bargraphs.	<EBn>	Erase Bargraph
Remove one or all previously defined variabless.	<EVn>	Erase Variable
Clear the screen and remove all mapped variables and bargraphs.	<NS>	New Screen

Full details of each command are listed in the “Command Reference” section (Page 24).

Scripting & Pattern Matching

As it is not necessary to use these functions in many conventional applications, they are documented in their own section towards the back of this manual starting on page 115.

Command Summary

There are 94 commands that can be arranged into 7 functional groups:

Screen Handling & Text	- used to control the screen in text mode
Attributes	- affect the appearance of text and graphics
Pixel Graphics	- draw graphical objects on the screen
Line Graphics	- draw lines and boxes on the screen
System	- affect the operation of the text display
Mapped Variables	- link text and graphics to cyclic data values
Scripting	- allows control by a user defined sequence of commands

Screen Handling & Text

Command	Meaning
<CLn>	Clear Line
<CMy,x>	Cursor Move
<CS>	Clear Screen
<CW>	Clear Window
<EL>	Erase Line
<FS>	Fill Screen
<FW>	Fill Window
<HC>	Home Cursor
<LN>	Line New
<RS>	Request Status
<SD>	Screen Defaults
<WSn>	Write Soft character
<WTstring>	Write Text

Attributes

Command	Meaning
<BMn>	Background Mode
<CA>	Centre Align
<DWyt,yb,xl,xr>	Define Window
<EF>	Enable Flashing
<F1>	Font 1
<F2>	Font 2
<F3>	Font 3
<F4>	Font 4
<F5>	Font 5
<FL>	Flashing
<IF>	Inhibit Flashing
<LA>	Left Align
<LF>	Line Feed
<NA>	No Align
<NL>	No Linefeed
<NU>	No Underline
<RA>	Right Align
<ST>	Steady
<SW>	Smart Wrap
<TW>	Text Wrap
<UL>	UnderLine
<WMn>	Write Mode

Pixel Graphics

Command	Meaning
<DG>	Download Graphic
<DS>	Download Screen
<HRn,m,p>	Horizontal Rotate
<HSn,m,r,s,t,u,v>	Horizontal Scroll
<UE>	Upload Enable
<US>	Upload Screen

Line Graphics

Command	Meaning
<BDylength,xlength,lwidth>	Box Draw
<HBn,m>	Horizontal Bargraph
<LHxlength, lwidth>	Line Horizontal
<LVylength, lwidth>	Line Vertical
<VBn,m>	Vertical Bargraph

Mapped Variables

Command	Meaning
<CD>	Cyclic Data
<CTn>	Cyclic-data Timeout
<CVn,m>	Cyclic Variable
<DBn,m,p,q,r>	Define Bargraph
<DDn,m>	Define Decimal
<DLn,m,p>	Define Limits
<DTn,string>	Define Tag
<DUn,string>	Define Units
<DVn,m,p,q>	Define Variable
<EBn>	Erase Bargraph
<EVn>	Erase Variable
<NS>	New Screen
<SVn>	Show Variable

System

Command	Meaning
<AFn>	Active Frame
<CCn>	Check Code
<CE>	Configuration Enable
<CI>	Command Implement
<CP>	Configuration Prohibit
<CRn,m>	Cyclic Redundancy check
<DFn>	Download Font
<FR>	Font Restore
<GBn>	Graphic Block
<KF>	Keep Fonts
<MCn>	Make Connection
<ODn>	Output De-energised
<OEn>	Output Energised
<PM>	Pixel Mode
<RB>	ReBoot
<RC>	Release Connection
<RFn>	Restore Frame
<RLx>	Restore Logo
<RM>	Row Mode
<SAn>	Speed Adjust
<SBn>	Set Backlight
<SFn,m>	Save Frame
<SHn>	Status Hide
<SL>	Save Logo
<SOn>	Screen Option
<SSn>	Screens to Scroll
<TOn>	Time Out
<VFn>	Visible Frame
<VLn>	Variable Last

Scripting

Command	Meaning
<CU>	Cyclic Update
<CX,n,string>	Cyclic Text
<DPn>	Download Program
<ES>	Execute Script
<GE>	Get Error
<GL>	Get Line
<JRn>	Jump Register
<KS>	Kill Script
<SEn>	Script Event
<TS>	Terminate Script

Note that these scripting commands are not documented in the command reference section, but can be found in their own section towards the back of this manual starting on page 115.

Command Reference

The following section lists each command in alphabetical order. Each page is formatted in the same way so that commands can be compared and reviewed easily.

The following page explains the format of each page:

<..> Command

Group

Description This is a brief description of the command

Parameters The allowable range of values

Initial Value The value at initialisation, if applicable

Modes Some commands are only available in certain modes

Notes Detailed comments

Uses Describes where the command may be used

Example A simple example showing how to use the command

Be aware that most examples assume a <SD> command has been issued to clear the screen first

This section may be omitted for complex commands; additional information can be found in other parts of this guide.

Gotchas! Common pitfalls to be aware of

See Also Other related commands

<AF*n*> Active Frame

System Command

Description Specify that all writes are directed to Frame *n*

Parameters *n* = 0 or 1 - frame number

Initial Value Frame 0 is the default at power up, or after a <SD> Screen Defaults command

Modes All Modes

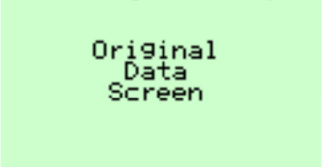
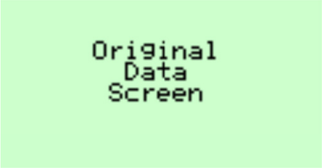
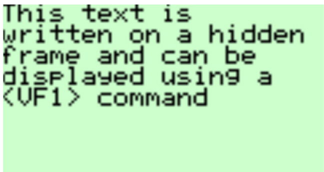
Notes All commands (with the notable exception of <RL*n*>) write to the Active Frame, not the Visible Frame. This gives the flexibility to make complex data screens appear relatively quickly, to provide an intuitive user interface

Detailed information about the use of frames can be found in the Frames Section (Page 7).

Uses The <AF> command allows:

- Complex screens to be drawn and then displayed when they are complete
- Rapid switching between two different information screens

Example

 Original Data Screen	Assume the display is showing some data, and the active frame and visible frame are both set to 0
`<AF1>`	Set active frame to 1; LCD display screen unaltered
`<CS>`	Clear the hidden frame; LCD display screen unaltered
`<SW>`	Set smart wrap formatting on to cope with a long line of text
`<WTThis text is written on a hidden frame and can displayed using a <VF1>> command>`	Write out the text to the hidden frame; LCD display screen unaltered
 Original Data Screen	LCD display screen at this point
`<VF1>`	Make frame 1 visible
 This text is written on a hidden frame and can be displayed using a <VF1> command	

Gotchas! Make sure that the section on Frames (Page 7) is read and understood

Frame *n* may or may not currently be visible. Use the <VF> command to achieve the desired result

See Also **VF** Visible Frame

<BDy,x,l>   Box Draw

Description Draws a box *y* pixels high, *x* pixels wide with a line thickness of *l*

Parameters

y = 2 to 64	- height
x = 2 to 120	- width
l = 1 to 32	- line thickness

Modes Pixel mode only

Notes The box is drawn from the current cursor position upwards and to the right.

The cursor position is unchanged after the command

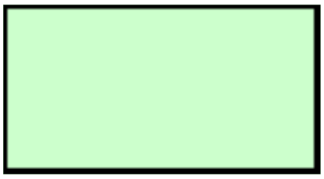
The parameters may be any value that will keep the box being drawn on-screen. If any part of the defined box is off-screen, then the box is not drawn and an error response it returned to the host.

Uses The <BD> command allows:

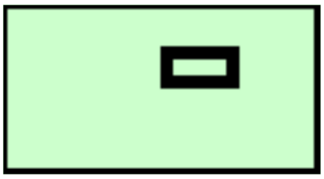
- information to be segmented
- borders to be drawn
- line images to be constructed

Example

`<CS>`	Clear Screen
`<PM>`	Set Pixel Mode
`<CM63,0>`	Move the cursor to the bottom left had corner of the display LCD
`<BD64,120,1>`	A box ,a single pixel thick, is drawn round the edge of the display LCD



`<CM31,60>`	Move the cursor to the centre of the LCD display
`<BD16,30,5>`	A box 16 by 30 pixels, 5 pixels thick, is drawn with its bottom left hand corner in the centre of the LCD display



Gotchas! The entire box must fit on the screen, otherwise nothing will be drawn and an error response generated

See Also

LH	Line Horizontal
LV	Line Vertical

<BM*n*> Background Mode

Description Defines the appearance of the ‘flashing’ attribute

Parameters *n* = 0 to 2 - flashing style

Initial Value 0

Modes All Modes

Notes The flash background is defined by the value *n*
n = 0 sets all pixels off
n = 1 sets all pixels on
n = 2 sets the pixels to the inverse of the character or graphic being written

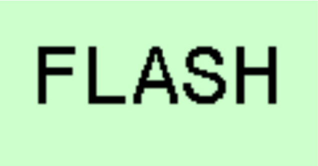
To use this command the flashing attribute <FL> must be set for each object, and then the enable flash <EF> command sent

Uses The <BM> command allows:

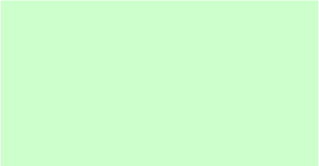
- attention grabbing messages
- special effects

Example

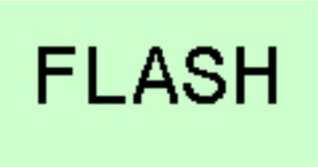
`<BM0>`	Background to flashing characters is all pixels off
`<WTFLASH>`	Write the text FLASH to the screen



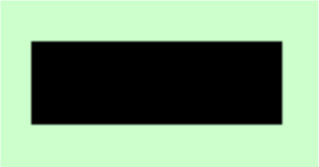
Alternates each second with



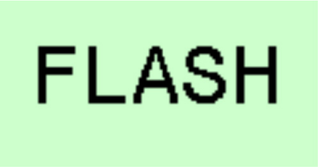
`<BM1>`	Background to flashing characters is all pixels on
`<WTFLASH>`	Write the text FLASH to the screen



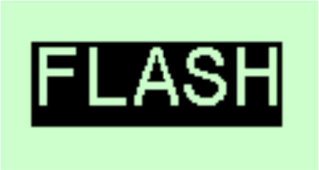
Alternates each second with



`<BM2>`	Background to flashing characters is all pixels are the inverse of the image being flashed
`<WTFLASH>`	Write the text FLASH to the screen



Alternates each second with



Gotchas! To use these effects successfully, the background mode must be set **before** the screen is written to.

See Also
EF Enable Flash
FL Flashing

<CA> Centre Align

Description Set the attribute so that written text is aligned horizontally within the screen or defined window

Parameters None

Initial Value Not aligned; Text appears at the current cursor position

Modes All Modes

Notes This command only affects text written after the attribute has been set.

In Pixel Mode <PM> the centring is always based on the full screen. In row mode the text is centred in the currently defined window, which by default is the full screen.

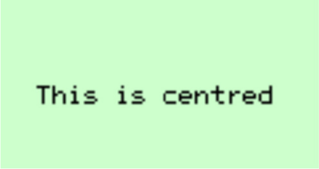
The attribute is cancelled by the <NA> command or any of the other text alignment commands <LA>, <RA>, <SW> & <TW>

Uses The <CA> command allows:

- Text to be automatically aligned without the need for cursor move commands
- Tidy screen presentation

Example

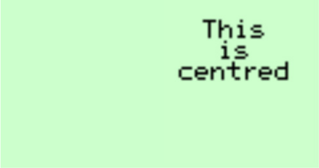
`<PM>`	Set Pixel Mode
`<CM40,0>`	Set up the vertical position: move the cursor to 40 pixels down from the top of the screen, on the left hand side of the screen
`<CA>`	Align all following text centrally
`<WTThis is centred>`	Write the message



This is centred

The horizontal position is calculated from the length and size of the text

`<RM>`	Set Row Mode
`<DW0,7,60,119>`	Define a window as the right hand half of the screen
`<CM1,0>`	Move the cursor to row 1 (one row from the top)
`<CA>`	Align all following text centrally
`<WTThis>`	Write the text
`<LN>`	Move the cursor down one line
`<WTis>`	Write the text
`<LN>`	Move the cursor down one line
`<WTcentred>`	Write the text



This
is
centred

See Also

LA	Left Align
NA	No Align
RA	Right Align
SW	Smart Wrap
TW	Text Wrap

<CC*n*> Check Code

System

Description Command terminator with 8-bit checksum

Parameters The parameter is an 8 bit checksum of all the characters in the preceding command string.
To calculate the value of the parameter, sum the ASCII values of all the characters in the command string up to but not including the <CC*n*> command. Divide this sum by 256 decimal (0x100 hex). The checksum is the remainder after the division and is sent as a single byte.

Modes Operational Mode 3 only

Notes This command is the signal to verify the checksum of the preceding command string and, if correct, action the commands.

If the checksum is not correct, no commands are actioned and an error response is returned.

The <CC*n*> command terminator is used in the most basic of the error checked modes, Operational Mode 3. If higher data security is required consider using Operational Mode 4 which has two parameters in the <CR*nn*> command terminator, representing a 16 bit CRC of the preceding command string.

Uses The <CC*n*> command allows:

- Commands to be queued but not actioned until required
- Basic message error checking

Example Assume the text display is in operational mode 3.

In order to clear the screen, a <CS> command must be sent followed by the <CC*n*> command where n represents the 8-bit sum of the characters "<", "C", "S", ">"

ASCII values of the example command are:

"<"	=	60 decimal	(3C hex)
"C"	=	67 decimal	(43 hex)
"S"	=	83 decimal	(53 hex)
">"	=	62 decimal	(3E hex)

In decimal notation:

The sum is 60+67+83+62 = 272 and the checksum is the remainder after division by 256.
Hence the checksum is remainder of the division 272/256 = 16

In hexadecimal notation:

The sum is 3C+43+53+3E =110 hex and the checksum is the Least Significant Byte (LSB) of this sum.
Hence the checksum is 10 hex.

The checksum must be sent as a single byte 16 decimal (10 hex).

Hence the full command string is: **`<CS><CC[16]>`**

Note! The square brackets are not sent, they are just there to emphasise that a single byte 16 is sent. The checksum, as in this case, may not be a printable ASCII character.

Gotchas! The checksum is always a single byte, and may be an unprintable character.

See Also

CI Command Implement
CR Cyclic Redundancy Check

<CD> Cyclic Data

Description Update all eight input variables using floating point numbers

Parameters None

Modes All Modes

Notes The concepts and usage of Standard Screens, Cyclic Data and Mapped Variables are discussed in their own section starting on Page 15.

The <CD> Cyclic Data command is used to update all eight input variables at the same time. Instead of an ASCII string, the values are sent as IEEE floating point numbers. In addition, the status of each value can be set **GOOD** or **BAD**. The data format is a single status byte followed by 4-byte float for each variable. The status byte for each value should be set to zero to mark that value as **BAD**, or any value between 1 and 255 to mark it as **GOOD**.

The command <CD> is issued with any additional bytes (checksum, CRC etc) as required by the current operational mode. The command is acknowledged if correctly received.

A binary download (from the host to the display) of the input values is then expected. The length must be (1+4)x8 = 40 bytes and if not an error response is returned.

After the data has been downloaded the <CI>, <CCn>, or <CRnn> command must be sent as per the current operational mode, the check byte(s) being calculated from all 40 bytes of the input data.

The download is acknowledged if it was correctly received (including checksum or CRC checks) and the values are displayed.

There is a 2 second timeout for the download operation, during which time if no bytes are received the download is aborted and an error response is returned. Of course, the actual total download time depends on the speed of the serial link.

Uses The <CD> command allows:

- Displayed variables may be updated and the screen redrawn with minimal host overhead
- Standard screens to be used for the convenient display of up to eight process variables with little application programming
- More complex screens can be built up using other “Mapped Variable commands”

Example See the Standard Screens section (Page 15)

See Also

CT	Cyclic-data Timeout
CV	Cyclic Variable

<CE>Configuration Enable

Description Control access to the configuration menus

Parameters None

Initial Value This is the default

Modes All Modes

Notes The <CE> and <CP> commands control access to the main and quick access menus used for unit configuration.

The <CP> command will prevent user access to the configuration menus via the dual P-E and P-UpArrow key presses.

The <CE> command will re-enable user access to the menus.

The Quick Access menu can also be disabled within the ‘Display’ section of the main menu. When it is disabled in this way the <CE> command has no effect on the access to this menu.

The commands have no effect on the LCD display screen.

The instrument sends a different response to commands from the host when it is in the programming menus.

Uses The <CE> command allows:

- changes to instrument configuration to be made after a <CP> Configuration Prohibit command

Example

<CP>	Lock out the menus
Any	A set of commands or operator instructions that should not be interrupted by adjustments to the display configuration be made
<CE>	Re-enable access to the menus for maintenance

There is no effect on the display LCD screen
when these commands are used

See Also **CP** Configuration Prohibit

<CI> Command Implement

Description Command terminator without any checksum

Parameters None

Modes Operational Mode 2 only

Notes The <CI> terminator is the signal to action the preceding command string.

It is used only in Operational Mode 2 where a string of commands can be "queued" and then actioned at the same time. This is essentially the same as the more complex Operational Modes 3 or 4, but without any error checking

Uses The <CI> command allows:

- Commands to be queued but not actioned until required
- Commands to be as actioned as quickly as possible as there is no error checking or responses to the host until all the commands have been actioned.
- Complex screens to be designed and tested just using a terminal emulator

Example Assume the display is in Operational Mode 2.

The command string:

<CS><FS><CS><FS><CI>

will clear the display LCD, then turn all the pixels on , clear the display again, and turn all the pixels on again. The command string <CS><FS><CS><FS> is only actioned when the <CI> command is received.

See Also

CC	Check Code
CR	Cyclic Redundancy Check

<CL*n*> Clear Line

Description Clears a complete line on the screen

Parameters *n* = 0 to 7 - line number

Modes Row Mode Only

Notes There are 8 lines on the screen numbered 0 to 7, 0 being the top line.

The command clears a number of lines upwards from the stated line, depending on the current font:
For font 1 <F1> only one line is cleared
For font 2 <F2> two lines are cleared, and so on.

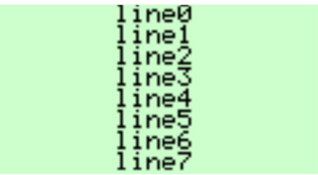
This command is window aware. If a window is in use, the line numbers are relative to the window. That is, line 0 is the top line of the window.

Cursor position is unchanged by this command

Uses The <CLn> command allows:

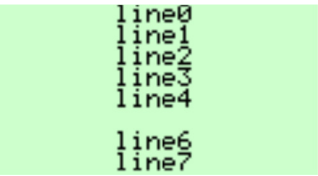
- Message/status information to be cleared
- Ensures new messages can be written without leaving part of an old message in place
- Clearing of lines in a window

Example Assume the initial display is:



```
line0
line1
line2
line3
line4
line5
line6
line7
```

`<F1>`	Make sure we know the font in use
`<CL5>`	Clear line 5



```
line0
line1
line2
line3
line4

line6
line7
```

Gotchas! The current font size must be taken into account before issuing this command.
In the example above, if Font 2 were in use then line 5 and line 4 would be blanked.

See Also

DW	Define Window
EL	Erase Line
F*n*	Font *n*

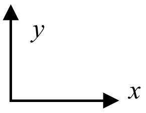
<CM*y*,*x*> Cursor Move

Description

Moves the cursor on the screen

Parameters

Row Mode:	Pixel Mode:
y = 0 to 7	*y* = 0 to 63
x = 0 to 119	*x* = 0 to 119



In both modes, co-ordinate 0,0 is at the top left of the screen

Modes

All Modes

Notes

This command moves the cursor to the position defined by the parameters *y* and *x*. The cursor is never visible; it can be considered an insertion point on the screen for text and graphics.

Text and graphics are always drawn upwards and to the right of the current cursor position.

When text is written, the cursor is placed at the end of the inserted text. When graphics images are written to the screen, the cursor position is unaltered.

In Row mode this command is window aware. If a window is in use the parameters *y* and *x* are relative to the current window.

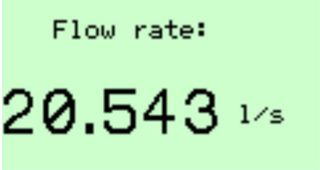
Uses

The <CM> command allows:

- Positioning of text and graphics objects

Example

`<CS>`	Clear Screen
`<RM>`	Set Row Mode
`<F1>`	Small 8x6 pixel font
`<CM1,20>`	Move the cursor to the second row, 20 pixels from the left edge of screen
`<WTFlow Rate:>`	Write a heading
`<F3>`	
`<CM6,0>`	Move the cursor to the next to bottom row, on left edge of screen
`<WT20.543>`	Write in process value
`<F1>`	Small font again
`<CM5,90>`	Move cursor to row 5, 30 pixels from right had side of screen
`<WTl/s>`	Write in units



```
   Flow rate:

20.543 l/s
```

Gotchas!

The horizontal resolution is always 1 pixel.

See Also

DW Define Window

<CP>Configuration Prohibit

Description Control access to the configuration menus

Parameters None

Initial Value <CE> Configuration Enable is active on power up

Modes All Modes

Notes The <CE> and <CP> commands control access to the main and quick access menus used for unit configuration.

The <CP> command will prevent user access to the configuration menus via the dual P-E and P-UpArrow key presses.

The <CE> command will re-enable user access to the menus.

The Quick Access menu can also be disabled within the 'Display' section of the main menu. When it is disabled in this way the <CE> command has no effect on the access to this menu.

The commands have no effect on the display LCD screen.

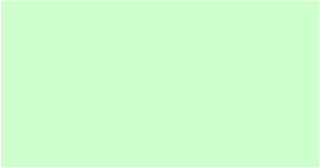
The instrument sends a different response to commands from the host when it is in the programming menus.

Uses The <CP> command allows:

- The prevention of unauthorised changes to instrument configuration
- The prevention of operators missing messages from the host, due to the instrument being in programming mode

Example

`<CP>`	Lock out the menus
`< Anything >`	A set of commands or operator instructions that should not be interrupted by adjustments to the display configuration be made
`<CE>`	Re-enable access to the menus for maintenance



There is no effect on the display LCD screen
when these commands are used

See Also **CE** Configuration Enable

<CR*n,m*> Cyclic Redundancy Check

Description Command terminator with 16-bit checksum

Parameters The parameters *n* and *m* are two 8 bit bytes of the 16 bit checksum of all the characters in the preceding command string, *n* being the LSB, and *m* the MSB.

To calculate the value of the parameter, use the ASCII values of all the characters in the command string up to but not including the <CR*nm*> command. See the section on CRC Generation (page 11) .

Modes Operational Mode 4 only

Notes This command is the signal to verify the checksum of the preceding command string and, if correct, action the commands.

If the checksum is not correct, no commands are actioned and an error response is returned.

Uses The <CR*n,m*> command allows:

- Commands to be queued but not actioned until required
- Rigorous message error checking

Example Assume the text display is in operational mode 4.

To clear the screen the following command needs to be sent:

<CS><CRnm>

The parameters n and m are calculated by running the ASCII value of the characters “<”, “C”, “S”, “>” through the CRC algorithm. When this is done the CRC code generated is 0x8040

Hence n = 40 hex (64 decimal), m = 80 hex (128 decimal).

The full command is then:

`<CS><CR[64][128]>`

Note! The square brackets are not sent, they are just there to emphasise that n and m are single 8-bit bytes. The check bytes may not be printable ASCII characters.

The CRC code for the message <WTHello World> is 0x721B

Hence n = 1B hex (27 decimal), m = 72 hex (114 decimal).

The full command is then:

`<WTHello World> <CR[27][114]>`

Gotchas! Make sure that the section on CRC Generation (page 11) is read and understood!

See Also

CC	Check Code
CI	Command Implement

<CS>Clear Screen

Description Turn all pixels off, creating a blank screen

Parameters None

Initial Value All pixels are off

Modes All Modes

Notes This command also:

- Removes any windows that may be defined
  (equivalent to issuing a <DW0,7,0,119> command)
- Homes the cursor
  (equivalent to issuing a <HC> command)

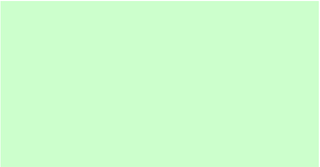
Uses The <CS> command provides:

- A known starting point before drawing a new screen

Example

```
<CS>
```

Clear Screen



Gotchas! Defined variables and bargraphs are NOT cleared by this command – or rather they are, but are updated again automatically. Use the EV and EB commands to remove the mappings, or use the NS command to clear the screen and all the mappings.

If windows are being used, they must be defined after this command

See Also

CW	Clear Window
DW	Define Window
EB	Erase Bargraph
EV	Erase Variable
FS	Fill Screen
HC	Home Cursor
NS	New Screen

<CT*n*> Cyclic Data Timeout

Description A mechanism to warn that an input variable has not been updated for (*n* x 10) Seconds

Parameters *n* = 0 to 255 - Multiples of 10 Seconds

Initial Value 0, no timeout active

Modes All Modes

Notes If valid data is not regularly sent to each input value then there is a risk that an operator might take some inappropriate action in response to stale data. A mechanism has been built into the display that indicates when the data is older than a predetermined timeout value. The data is displayed as **BAD** if the timeout value has been exceeded, but the "Status" remains unchanged. Sending valid data once again to the input variable will display the value normally, marked **GOOD** or **BAD** according to the "Status".

As the acceptable timeout value is totally dependant on the application and the host capabilities, it is adjustable from 10 Seconds to 40 Minutes. For simple applications the timeout value can be set to 0 which disables the facility.

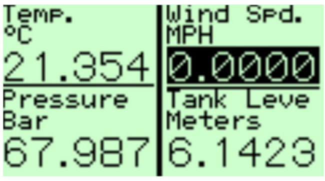
The parameter *n* sets a timeout period of *n* x 10 seconds.
n = 0 deactivates the timeout function.

Uses The <CT> command allows:

- Users to be warned that displayed variable may be out of date

Example

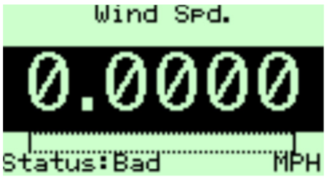
<SO3> Select a four variable screen



The "Wind Speed" reading is shown here as being older than the timeout value

<SO4> Change to a single variable plus bargraph

<SV3> Show the third input (In our case, Wind Speed)



The bargraph, is drawn as a dotted outline showing that it is invalid

Gotchas! Despite the similarity in the name, the <TO> Timeout command is fundamentally different and are not usually used in conjunction with each other.

See Also

CD	Cyclic Data
CV	Cyclic Variable
TO	Timeout

<CV*n,string*> Cyclic Variable

Description Update a single input variables using ASCII formatted text

Parameters

n	= 1 to 8 - Input Variable number
string	= any 7-bit numeric ASCII string

Modes All Modes

Notes The concepts and usage of Standard Screens, Cyclic Data and Mapped Variables are discussed in their own section starting on Page 15.

The <CV> Cyclic Variable command is used to update each variable individually by sending the numeric value as a simple ASCII string.

The syntax is <CV*n,string*> where n represents the input variable to be updated (= 1 to 8) and string is the numeric value to be displayed. For example, the command <CV1,123.456> will set the input variable 1 to a value of "123.456". The string is converted to a numeric value and appears on the standard screens, drawing a bargraph if appropriate. The value can be updated when necessary by simply sending the same command with the new value.

Note that there is no mechanism to set the data **GOOD** or **BAD** – all data is assumed to be **GOOD**.

Uses The <CV> command allows:

- Displayed variables may be updated and the screen redrawn with minimal host overhead
- Standard screens to be used for the convenient display of up to eight process variables with little application programming
- More complex screens can be built up using other "Mapped Variable commands"

Example See the Standard Screens section (Page 15)

See Also

CD	Cyclic Data
CT	Cyclic-data Timeout

<CW> Clear Window

Description Turn all pixels off within a defined window

Parameters None

Initial Value All pixels are off

Modes Row Mode only

Notes This command also homes the cursor in the defined window area, equivalent to issuing a <HC> command.

Apart from its main use in just clearing the contents of a window, it can also be used to create frames to contain text and graphics. Using this technique is much faster than using a <BD> Box Draw command in Pixel Mode.

Uses The <CW> command allows:

- A known starting point before updating a window
- Creation of a simple border

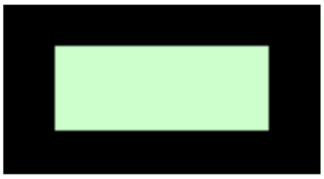
Example

`<RM>` Set Row Mode

`<FS>` Turn all screen pixels on

`<DW2,5,20,100>` Define a window two rows from top and bottom, 20 pixels in from both sides

`<CW>` Set all the pixels in the window area to off



See Also

BD	Box Draw
CS	Clear Screen
FW	Fill Window
HC	Home Cursor

<DB*n,m,p,q,r*> Define Bargraph

Description This command specifies how an input variable is written as a bargraph to the screen

Parameters

n = 1 to 8	- the input variable number ***IN_1*** to ***IN_8***
m = 5 to 120	- the total length of the bargraph in pixels
p = 0 to 8	- Lower limit of bargraph 0 = use static limit defined by the <DL> command 1 - 8 = Use the process variable connected to the specified input ***IN_1*** to ***IN_8***
q = 0 to 8	- Upper limit of the bargraph 0 = use static limit defined by the <DL> command 1 - 8 = Use the process variable connected to the specified input ***IN_1*** to ***IN_8***
r = 0 or 1	- Alignment: 0 = Horizontal bar, 1 = Vertical bar.

Modes Row Mode only

Notes The bargraph variable defined by this command is drawn at the current cursor position. The variable is automatically updated as new cyclic data is received.

An empty bargraph with a dotted outline indicates that data with "bad" status is being displayed, or that the input value is outside the defined limits.

Only one instance of a particular input can be on the displayed at a time. If the command is issued when there is already a bargraph of the specified input on-screen, the variable is moved from the old to the new position.

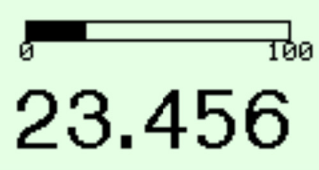
The command will fail and a parameter error result is produced if any part of the defined bargraph is off-screen

Uses The <DB> command allows:

- A process variable to be shown on screen as a continuously updated bargraph
- Up to 8 bargraphs to be displayed on a single screen
- Bargraph definitions to be saved together with on-screen text and graphics using the <SF> command

Example

`<SD><CM1,10>`	Start from a known screen and move to Row 1
`<DL1,0.0,100.0>`	Define the bargraph limits for ***IN_1*** as 0.0 and 100.0
`<DB1,100,0,0,0>`	Define a horizontal bargraph 100 pixels long using ***IN_1***, and the limits specified by the DL command
`<CM2,8><WT0>`	Move to row 2 and write in the scale "0"
`<CM2,101><WT100>`	Move cursor to the right and write "100"
`<F4>`	Specify a large font (32x19 pixels)
`<CM7,5>`	Move to Row 7
`<DV1,6,3,0>`	Define a variable using ***IN_1***, 6 character field, 3 chars after the decimal point, and written left aligned



The process variable linked to ***IN_1*** in this example has a "good" status and a value of "23.456"

Gotchas! Only one instance of a particular input can be on the displayed at a time, although a bargraph can be displayed along with its numeric value.
Bargraphs cannot be removed with the <CS> or <SD> commands.

See Also
DV Define Variable
EB Erase Bargraph

<DD*n,m*> Define Decimal

Description Set the number of decimal places displayed on standard screens

Parameters
n = 1 to 8 - Input variable
m = 0 to 5 - Number of decimal places
0 = Show no decimal places
1 - 4 = Show 1 – 4 decimal places
5 = Automatically format to show the maximum number of decimal places

Initial Value All input variables are set to Auto format
Once changed, these settings are retained in non-volatile memory

Modes All, when showing Standard Screens

Notes The number of decimal places that are displayed is selectable between a value of 0 and 5.

Each input variable can be configured to have a different number of decimal places, according to the application requirements.

A value of *m* = 0 will display no decimal places, *m* = 1 will display one decimal place and so on. However, a value of *m* = 5 will select an automatic mode whereby the maximum number of decimal places is shown, dependant on the available space. Variables are centred when automatically formatted, and right-justified when manually formatted.

Information written in this way is saved to non-volatile memory and is retained if power to the display is lost.

Uses The <DD> command allows:

- The standard screens to show the operator the required precision
- Automatic formatting of the numeric data without additional programming

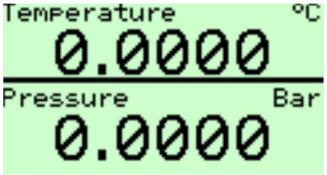
Example

```
<SO2>
<SV1>
```

Set Screen Option to 2

```
<CV1,0>
<CV2,0>
```

Send data to variables 1 and 2



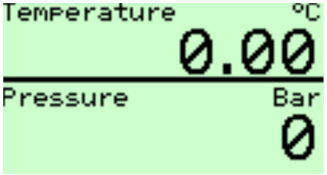
Temperature °C
0.0000
Pressure Bar
0.0000

The default appearance is to display as many decimals as possible on a given screen

Note that the variables are centred on the display

```
<DD1,2>
<DD2,0>
```

Set the desired number of decimals



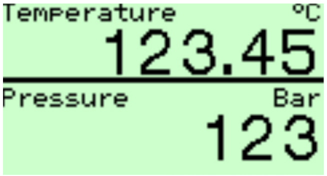
Temperature °C
0.00
Pressure Bar
0

The same data is presented as desired

This time the variables are right-justified

```
<CV1,123.457>
<CV2,123.456>
```

Send more data to variables 1 and 2



Temperature °C
123.45
Pressure Bar
123

The number of decimals is preserved as the data is updated

See Also **DV** Define Variable

<DF*n*> Download Font

Description Download soft fonts to the display

Parameters *n* = 0 to 3 - soft font character

Modes All Modes

Notes A soft font is any user defined image that is the same size as the current font.
The display will store 4 soft fonts (*n* = 0 to 3) for each font F1 to F5.

Soft characters are written to the screen by using the <WS*n*> command and may be used in both Row and Pixel Modes. They may also be underlined and flashed using attributes, as any other character.

After the <DF*n*> command is issued, the display expects a download of a Windows 2-colour BMP file of the soft character. The required image size depends on the currently active font

Font:		Image Size (v x h):
	F1	8 x 6 pixels
	F2	16 x 10 pixels
	F3	24 x 15 pixels
	F4	32 x 19 pixels
	F5	48 x 29 pixels.

The download mechanism is identical to the <DG> Download Graphic and <DS> Download Screen commands. Detailed information is in the Graphics Transfer Section (Page 13).

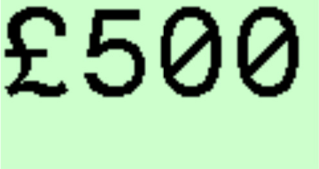
Soft fonts are lost when power is removed from the display. Fonts can be saved / restored as a block using the <KF> Keep Fonts and <FR> Font Restore commands

Uses The <DF> command allows:

- Any special character to be stored in the display so that it can be written to the screen just like any other character

Example

`<CS>`	Clear Screen
`<F5>`	Set largest font size
`<DF0>`	Tell the display that a soft character number 0 (for Font 5) is going to be downloaded
`Binary download of graphics file`	Send a .BMP file of the required soft character to the display. In this case a 48 x 29 pixel image of a GBP symbol (£)
`<WS0>`	Write the soft character to the screen
`<WT500>`	Write normal text



£500

Gotchas! Make sure that the section on Graphics Transfer (Page 13) is read and understood!

Soft characters can be underlined with the <UL> attribute – care should be taken when designing fonts if this attribute is to be used.

See Also

DG	Download Graphic
KF	Keep Font
FR	Font Restore
WS	Write Soft Character

<DG> Download Graphic

Description Download a graphics image to the screen and display it at the current cursor position

Parameters None

Modes Pixel Mode Only

Notes The size of the image is computed from the data sent. If any part of the image would be off-screen when drawn then nothing is drawn on the screen and an error response returned to the host.

The download mechanism is identical to the <DF*n*> Download Font and <DS> Download Screen commands. Detailed information is in the Graphics Transfer Section (Page 13).

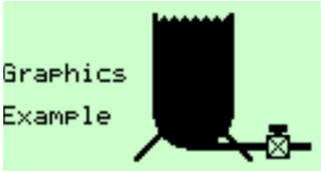
The cursor position is unchanged by this command.

Uses The <DG> command allows:

- Complex images to be generated on a PC and then downloaded to the display.
- Images can form a backdrop onto which standard text or data is then added.
- Pictures can sometimes convey simple messages more easily than text.

Example

`<CS>`	Clear Screen
`<RM>`	Set Row Mode
`<F1>`	Small 8x6 pixel font
`<CM3,0>`	Move the cursor to the start of the 4th row.
`<WTGraphics>`	Write the word “Graphics”
`<CM5,0>`	Move the cursor to the start of the 6th row.
`<WTExample>`	Write the word “Example”
`<PM>`	Change to Pixel Mode
`<CM60,50>`	Move the cursor to three pixels from the bottom of the screen, 50 pixels from the left hand side
`<DG>`	Tell the display to expect a graphics image download
`Binary download of .BMP file`	Download a 56 x 67 pixel image of a tank to the display



Gotchas! Make sure that the section on Graphics Transfer (Page 13) is read and understood

If any part of the graphic would be off-screen, then nothing is drawn and an error response is returned to the host

See Also

DS	Download Screen
US	Upload Screen

<DL*n,m,p*> Define Limit

Description This command defines static minimum and maximum limits for bargraph variables so that optimum scaling can be chosen

Parameters

n = 1 to 8	- Input variable number ***IN_1*** to ***IN_8***
m = any 7-bit numeric ASCII string	- Lower limit for the bargraph variable
p = any 7-bit numeric ASCII string	- Upper limit for the bargraph variable

Initial Value *m* = 0 and *p* = 100

Modes All Modes

Notes Valid values for *m* and *p* are 10 characters maximum, including the minus sign and decimal point

i.e. Between the range –999,999,999 to 9,999,999,999 ***(NB** Commas only shown for clarity)*

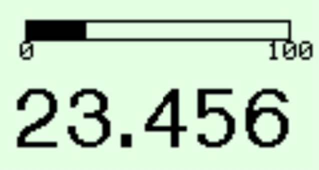
Variables which have a value outside the lower or upper limits are shown by empty dotted outline bargraphs

Uses The <DL> command allows:

- Optimum scaling of bargraphs
- Displaying of process variables in the form of bargraphs when used in conjunction with the <DB> command

Example

`<SD>`	Start from a known screen
`<CM1,10>`	Move to Row 1
`<DL1,0.0,100.0>`	Define the bargraph limits for ***IN_1*** as 0.0 and 100.0
`<DB1,100,0,0,0>`	Define a horizontal bargraph 100 pixels long using ***IN_1***, and the limits specified by the DL command
`<CM2,8>`	Move to row 2 and write in the scale
`<WT0>`	Write "0
`<CM2,101>`	Move cursor to the right
`<WT100>`	Write "100"
`<F4>`	Specify a large font (32x19 pixels)
`<CM7,5>`	Move to Row 7
`<DV1,6,3,0>`	Define a variable using ***IN_1***, 6 character field, 3 chars after the decimal point, and written left aligned



The process variable linked to ***IN_1*** in this example has a "good" status and a value of "23.456"

Gotchas! This command also affects all bargraphs that are mapped to that process variable on hidden and saved frames.

See Also **DB** Define Bargraph

<DS>Download Screen

Description Download a full-screen 64 x 120 pixel graphic image to the screen.

Parameters None

Modes All Modes

The <WM*n*> Write Mode has no effect on this command

Notes This command is really just a special case of the <DG> command, but because of the fixed size is executed much faster.

This command ignores the current Write Mode setting, and draws the downloaded image to the screen normally.

All other attributes are ignored, except those concerned with the ability to Flash the image.

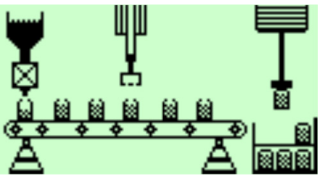
The download mechanism is identical to the <DF*n*> Download Font and <DG> Download Graphic commands. Detailed information is in the Graphics Transfer Section (Page 13).

The cursor position is unchanged by the command.

Uses The <DS> command allows:

- A full screen image to form a backdrop onto which standard text or data is then added
- A customised logo to appear at power on when used with the <SL> Save Logo command

Example

<CS>	Clear Screen
<DS>	Tell the display to expect a 64 x 120 pixel graphics image that it should display full screen.
Binary download of .BMP file is now sent	
	Image is displayed when received

Gotchas! The logo is not stored in EEprom unless the <SL> Save Logo command is issued.

See Also

RL	Restore Logo
SL	Save Logo

<DT*n,string*> Define Tag

Description Set Tag information to be displayed next to the input variable on standard screens

Parameters

n = 1 to 8	- the input variable number ***IN_1*** to ***IN_8***
string	- any 7-bit ASCII string up to 16 characters long

Initial Value "Inst1_Tag", "Inst2_Tag" … "Inst8_Tag"
Once changed, these settings are retained in non-volatile memory

Modes All, when showing Standard Screens

Notes Each of the eight process variables may be displayed with an identification tag that can have up to sixteen alphanumeric characters. Numbers, punctuation plus upper & lower case letters are available.

Information written in this way is saved to non-volatile memory and is retained if power to the display is lost.

The <DT> command is used to send an ASCII string with a maximum length of 16 characters.

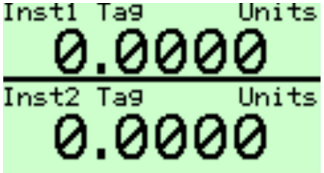
Uses The <DT> command allows:

- Displayed variables to have an associated description on screen
- Standard screens to be used for the convenient display of up to eight process variables with little application programming

Example

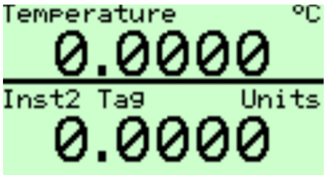
`<SO2>`	Select Screen format 2
`<SV1>`	Show the first variable

`<CV1,0>`
`<CV2,0>`
	Send data to the input variables



```
Inst1 Tag        Units
  0.0000
Inst2 Tag        Units
  0.0000
```

`<DT1,Temperature>`	Change the Tag to read "Temperature"
`<DU1,`C>`	Change the Units to read " ºC " Note the use of the special character ` which is converted to the degree symbol (See the **<DU>** command)



```
Temperature         °C
  0.0000
Inst2 Tag        Units
  0.0000
```

Gotchas! The string cannot include '>' as this is the command terminator
Some Standard Screens are unable to show the full length of the string due to lack of space

See Also **DU** Define Units

<DU*n,string*> Define Units

Description Set “Units of Measure” information to be displayed next to the input variable on standard screens

Parameters

n = 1 to 8	- the input variable number ***IN_1*** to ***IN_8***
string	- any 7-bit ASCII string up to 8 characters long

Initial Value “Units”
Once changed, these settings are retained in non-volatile memory

Modes All, when showing Standard Screens

Notes Each of the eight process variables may be displayed with units of measure that can have up to eight alphanumeric characters. Numbers, punctuation plus upper and lower case letters are available.

Information written in this way is saved to non-volatile memory and is retained if power to the display is lost.

To simplify temperature display, the ` character (alt+096) is mapped to the degrees symbol.

For example, the string **`Temp `C`** is displayed as **`Temp °C`**

The <DU> command is used to send an ASCII string with a maximum length of 8 characters. The command needs to be repeated for each input variable, in order to set the appropriate units of measure.

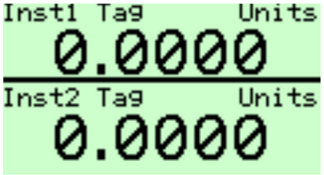
Uses The <DU> command allows:

- Displayed variables to have an associated unit of measure on screen
- Standard screens to be used for the convenient display of up to eight process variables with little application programming

Example

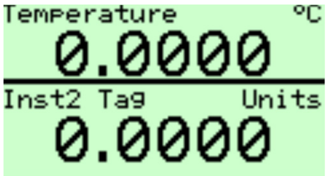
`<SO2>`	Select Screen format 2
`<SV1>`	Show the first variable

`<CV1,0>`
`<CV2,0>`
	Send data to the input variables



```
Inst1 Tag     Units
   0.0000
Inst2 Tag     Units
   0.0000
```

`<DT1,Temperature>`	Change the Tag to read “Temperature”
`<DU1,`C>`	Change the Units to read “ ºC ” Note the use of the special character ` which is converted to the degree symbol



```
Temperature      °C
   0.0000
Inst2 Tag     Units
   0.0000
```

Gotchas! The string cannot include '>' as this is the command terminator
Some Standard Screens are unable to show the full length of the string due to lack of space

See Also **DT** Define Tag

<DV*n,m,p,q,r*> Define Variable

Description This command specifies how a process variable is written to the screen

Parameters

n = 1 to 8	- the input number ***IN_1*** to ***IN_8***
m = 1 to 10	- the total variable string length
p = 0 to 4	- the maximum number of digits after the decimal point (Over-ridden by *m*)
q = 0 or 1	- Alignment. 0 = left align, 1 = right align in the defined field.
r = 0 or 1	- Strict Format 0 = Precision reduces to fit field width 1 = Display over-range if specified format not available for variable

Initial Value None

Modes Row Mode only

Notes The variable defined by this command is written at the current cursor position and in the current font. The variable is automatically updated as new cyclic data is received.

Data with "bad" status is written in "inverse" i.e. White numbers on a Black background.

Only one instance of a particular input can be displayed at a time. If the command is issued when there is already a variable from the specified input on-screen, the variable is moved from the old to the new position.

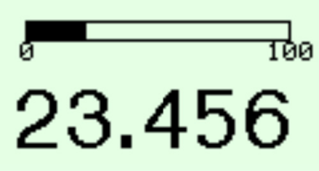
The command will fail and a parameter error result is produced if any part of the defined variable is off-screen.

Uses The <DV> command allows:

- Up to 8 process variables to be shown and continuously updated on screen
- The status condition of the variable to be indicated
- Variable definitions to be saved with text and graphics using the <SF> command

Example

`<SD>`	Start from a known screen
`<CM1,10>`	Move to Row 1
`<DL1,0.0,100.0>`	Define the bargraph limits for ***IN_1*** as 0.0 and 100.0
`<DB1,100,0,0,0>`	Define a horizontal bargraph 100 pixels long using ***IN_1***, and the limits specified by the DL command
`<CM2,8><WT0>`	Move to row 2 and write in the scale "0"
`<CM2,101>`	Move cursor to the right
`<WT100>`	Write "100"
`<F4>`	Specify a large font (32x19 pixels)
`<CM7,5>`	Move to Row 7
`<DV1,6,3,0>`	Define a variable using ***IN_1***, 6 character field, 3 chars after the decimal point, and written left aligned



The process variable linked to ***IN_1*** in this example has a "good" status and a value of "23.456"

Gotchas! Parameter "*m*" overrides parameter "*p*"
A Defined Variable cannot be removed by a Clear Screen or Screen Defaults command – Use the <EV> Erase Variable command instead

See Also

EV	Erase Variable
DB	Define Bargraph

<DW*yt*,*yb*,*xl*,*xr*> Define Window

Description

Defines an area of the screen into which certain screen write commands are constrained

Parameters

yt = 0 to 7 - top row of the window area
yb = 0 to 7 - bottom row of the window area
xl = 0 to 119 - pixel column of the left hand side of the window
xr = 0 to 119 - pixel column of the right hand side of the window

Initial Value

The initial window size is the full screen <DW0,7,0,119>

Modes

Row Mode Only

Notes

When a window is in use, all cursor related commands are relative to the window area. For example:

<HC> will home the cursor in the window area
<CM0,0> will move the cursor to the top row, left hand side of the window area.

The window may be redefined at any time without affecting the screen contents. In this way a window can be removed by defining the whole screen as a new window i.e.<DW0,7,0,119>

The <CS> Clear Screen and <PM> Pixel Mode commands also remove a window definition.

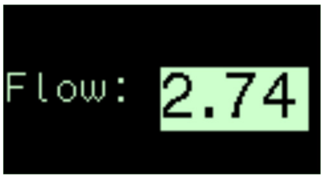
Uses

The <DW> command allows:

- Text to be scrolled in a window
- Trend graphs to be drawn by combining the use of the Horizontal Scroll <HS> command
- Static text and graphics to be protected; headings, footers or titles can be left in place while different messages are displayed and cleared within a window

Example

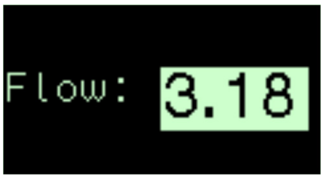
`<FS>`	Fill Screen
`<RM>`	Set Row Mode
`<F2>`	16x10 pixel font
`<CM4,0>`	Move the cursor to fifth row from the top, at the left of the screen
`<WM3>`	Write characters in Inverse mode (clear character on black background)
`<WTFlow:>`	Write the text "Flow:"
`<WM0>`	Back to normal write mode (black character on white background)
`<DW3,5,60,115>`	Define window for a value to be written
`<F3>`	Larger font
`<CW>`	Clear the window
`<WT2.74>`	Write out a value



```
Flow: 2.74
```

Subsequent values then only need:

`<HC>`	Home the cursor in the window area
`<WT3.18>`	Write out the new value



```
Flow: 3.18
```

Gotchas!

The <CS> Clear Screen and <PM> Pixel Mode commands remove any defined windows

See Also

CW Clear Window

<EB*n*> Erase Bargraph

Description This command erases bargraphs created with the <DB> command from the screen.

Parameters

n = 0	- Erases all defined bargraphs from the screen
n = 1 to 8	- Erases only the specified bargraph from the screen

Initial Value None

Modes Pixel and Row Modes only

Notes This command acts in two different ways depending on the value of "*n*"

There is no need to erase a bargraph if you just want to change its position. Just set up the new attributes and issue another <DB> command

To erase bargraphs from stored frames, restore the frame, erase the bargraphs and re-save the frame.

Uses The <EB> command allows:

- Bargraphs to be permanently erased from the display. The <CS> Clear Screen command does not work on defined variables (not for long anyway!)

Example Self explanatory

Gotchas!

See Also

DB	Define Bargraph
EV	Erase Variable
NS	New Screen

<EF> Enable Flashing

Description Flash text and graphics written with the Flashing <FL> attribute set

Parameters None

Initial Value Inhibited (No Flashing)

Modes All Modes

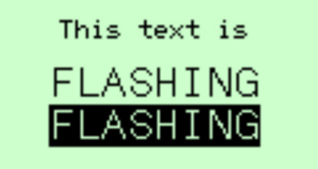
Notes The <EF> is a global command that affects the whole screen
The opposite of this command is the <IF> Inhibit Flash command

Uses The <EF> command allows:

- Flashing text generally attracts attention.
- Sending the <EF> command after the text is written ensures that all the text (written with the flashing attribute set) starts flashing at the same time.

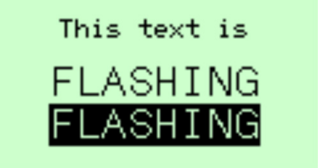
Example

<SD>	Set a known state for the display
<CM1,0>	Move the cursor to the first line down from the top of the screen
<CA>	All text is aligned centrally
<WTThis text is>	Write out the text
<BM2>	Set the flash background to the inverse of the foreground image
<F2>	Font size 2, 16 x 10 pixels
<CM4,0>	Move the cursor to the fourth line down from the top of the screen
<FL>	Set the Flashing attribute, so any text written will flash if flashing is enabled
<WTFLASHING>	Write out the text
<CM6,0>	Move the cursor to the sixth line down from the top of the screen
<WM3>	Write the foreground text as inverse (white on black background)
<WTFLASHING>	Write out some text



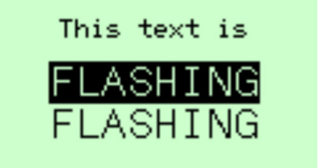
This text is
FLASHING
FLASHING

<EF>	Enable the flashing for the whole screen



This text is
FLASHING
FLASHING

Alternating each second with



This text is
FLASHING
FLASHING

Gotchas! Attributes are not saved and restored when screens are moved to and from memory with the <SFnm> and <RFm> commands

See Also

BM	Background Mode
FL	Flashing
IF	Inhibit Flashing
ST	STeady

<EL> Erase Line

Description Erase any text or graphics from the current cursor position to the end of the row

Parameters None

Modes Row Mode only

Notes The command erases a number of lines upwards from the current cursor position, depending on the current font:

For font 1 <F1> only one line is erased
For font 2 <F2> two lines are erased, and so on.

This command is window aware. If a window is in use the command erases only to the end of the window row.

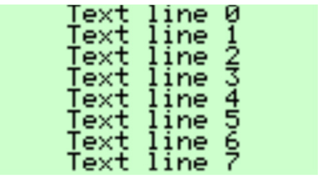
Cursor position is unchanged by this command

Uses The <CLn> command allows:

- Message/status information of variable length to be erased.
- Ensures new messages can be written without leaving part of an old message in place

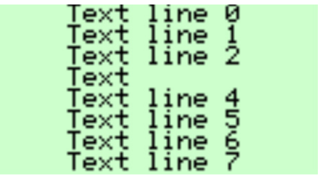
Example

<F1>	Sets the single row font, 8 x 6 pixels
<CA>	Turn on the centre align attribute
<WTText line 0>	Write out a line of text
<LN>	Down to the next line
….	Last two commands repeated up to…….
<WTText line 7>	



```
Text line 0
Text line 1
Text line 2
Text line 3
Text line 4
Text line 5
Text line 6
Text line 7
```

<CM3,50>	Move the cursor to row 3, 50 pixels in from the left of the screen
<EL>	Text erased from this cursor position to the end of the screen.



```
Text line 0
Text line 1
Text line 2
Text
Text line 4
Text line 5
Text line 6
Text line 7
```

Gotchas! The current font size must be taken into account before issuing this command.

See Also

CL	Clear Line
DW	Define Window
F*n*	Font *n*

<EV*n*> Erase Variable

Description This command erases variables created with the <DV> command from the screen.

Parameters

n = 0	- Erases all defined variables from the screen.
n = 1 to 8	- Erases only the specified variable from the screen

Initial Value None

Modes Pixel and Row Modes only

Notes This command acts in two different ways depending on the value of "*n*"

There is no need to erase a variable if you just want to change its position or font. Just set up the new attributes and issue another <DV> command.

To erase variables from stored frames, restore the frame, erase the variables and re-save the frame.

Uses The <EV> command allows:

- Mapped Variables to be permanently erased from the display. The <CS> Clear Screen command does not work on defined variables (not for long anyway!)

Example Self explanatory

Gotchas!

See Also

DV	Define Variable
EB	Erase Bargraph
NS	New Screen

<F1> Font 1

Description Define the text size written by the <WT> command as 8 x 6 pixels

Parameters None

Initial Value Font 1 is the default used on initialisation

Modes All Modes

Notes Font 1 is a single row font, each character being 8 pixels high by 6 pixels wide.
Font 1 does NOT have true decenders
Font 1 has a full 7-bit ASCII character set
This command also homes the cursor to the top left character position.

There are five separate commands that define the text size written by the <WT> command. They also affect free text written in Operational Modes 0 and 1.

The font sizes are as follows:

F1	Single row font	8 x 6 pixels
F2	Two row font	16 x 10 pixels
F3	Three row font	24 x 15 pixels
F4	Four row font	32 x 19 pixels
F5	Six row font	48 x 29 pixels

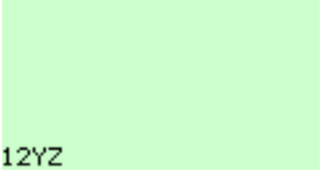
All fonts have a full 7-bit character set, except F5
All fonts have true decenders, except <F1>

Uses The <F1> command allows:

- The maximum number of characters on the screen
- Creation of long messages without resorting to multiple screens

Example

<CS>	Clear Screen
<F1>	Select font 1
<CM7,0>	Move cursor to the lower left of the display
<WT12YZ>	Write "12YZ"



12YZ

Gotchas! The cursor is homed by this command, and may need to be moved to the desired position before writing anything

See Also

DF*n*	Download Font
WS*n*	Write Soft Character
WT	Write Text

<F2> Font 2

Description Define the text size written by the <WT> command as 16 x 10 pixels

Parameters None

Initial Value Font 1 is the default used on initialisation

Modes All Modes

Notes Font 2 is a two-row font, each character being 16 pixels high by 10 pixels wide.
Font 2 has true decenders
Font 2 has a full 7-bit ASCII character set
This command also homes the cursor to the top left character position.

There are five separate commands that define the text size written by the <WT> command. They also affect free text written in Operational Modes 0 and 1.

The font sizes are as follows:

F1	Single row font	8 x 6 pixels
F2	Two row font	16 x 10 pixels
F3	Three row font	24 x 15 pixels
F4	Four row font	32 x 19 pixels
F5	Six row font	48 x 29 pixels

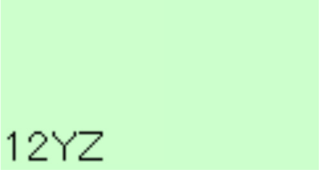
All fonts have a full 7-bit character set, except F5
All fonts have true decenders, except <F1>

Uses The <F2> command allows:

- Improved readability over font 1

Example

`<CS>`	Clear Screen
`<F2>`	Select font 2
`<CM7,0>`	Move cursor to the lower left of the display
`<WT12YZ>`	Write “12YZ”



12YZ

Gotchas! The cursor is homed by this command, and may need to be moved to the desired position before writing anything

See Also

DF*n*	Download Font
WS*n*	Write Soft Character
WT	Write Text

<F3> Font 3

Description Define the text size written by the <WT> command as 24 x 15 pixels

Parameters None

Initial Value Font 1 is the default used on initialisation

Modes All Modes

Notes Font 3 is a three-row font, each character being 24 pixels high by 15 pixels wide.
Font 3 has true decenders
Font 3 has a full 7-bit ASCII character set
This command also homes the cursor to the top left character position.

There are five separate commands that define the text size written by the <WT> command. They also affect free text written in Operational Modes 0 and 1.

The font sizes are as follows:

F1	Single row font	8 x 6 pixels
F2	Two row font	16 x 10 pixels
F3	Three row font	24 x 15 pixels
F4	Four row font	32 x 19 pixels
F5	Six row font	48 x 29 pixels

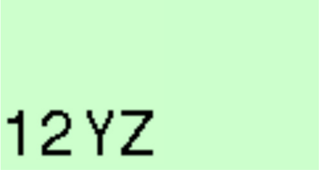
All fonts have a full 7-bit character set, except F5
All fonts have true decenders, except <F1>

Uses The <F3> command allows:

- Improved readability over font 2

Example

`<CS>`	Clear Screen
`<F3>`	Select font 3
`<CM7,0>`	Move cursor to the lower left of the display
`<WT12YZ>`	Write “12YZ”



```
12YZ
```

Gotchas! The cursor is homed by this command, and may need to be moved to the desired position before writing anything

See Also

DF*n*	Download Font
WS*n*	Write Soft Character
WT	Write Text

<F4> Font 4

Description Define the text size written by the <WT> command as 32 x 19 pixels

Parameters None

Initial Value Font 1 is the default used on initialisation

Modes All Modes

Notes Font 4 is a four-row font, each character being 32 pixels high by 19 pixels wide.
Font 4 has true decenders
Font 4 has a full 7-bit ASCII character set
This command also homes the cursor to the top left character position.

There are five separate commands that define the text size written by the <WT> command. They also affect free text written in Operational Modes 0 and 1.

The font sizes are as follows:

F1	Single row font	8 x 6 pixels
F2	Two row font	16 x 10 pixels
F3	Three row font	24 x 15 pixels
F4	Four row font	32 x 19 pixels
F5	Six row font	48 x 29 pixels

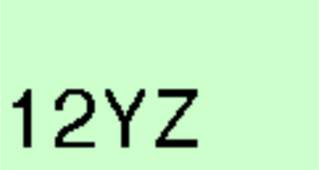
All fonts have a full 7-bit character set, except F5
All fonts have true decenders, except <F1>

Uses The <F4> command allows:

- An important parameter to dominate the screen layout

Example

`<CS>`	Clear Screen
`<F4>`	Select font 4
`<CM7,0>`	Move cursor to the lower left of the display
`<WT12YZ>`	Write “12YZ”



12YZ

Gotchas! The cursor is homed by this command, and may need to be moved to the desired position before writing anything

See Also

DF*n*	Download Font
WS*n*	Write Soft Character
WT	Write Text

<F5> Font 5

Description Define the text size written by the <WT> command as 48 x 29 pixels

Parameters None

Initial Value Font 1 is the default used on initialisation

Modes All Modes

Notes Font 5 is a six-row font, each character being 48 pixels high by 29 pixels wide.
Font 5 has true decenders
Font 5 has a limited 7-bit ASCII character set consisting of the following:

0 to 9, A to Z, space, comma, full-stop, plus, minus.

This command also homes the cursor to the top left character position.

There are five separate commands that define the text size written by the <WT> command. They also affect free text written in Operational Modes 0 and 1.

The font sizes are as follows:

F1	Single row font	8 x 6 pixels
F2	Two row font	16 x 10 pixels
F3	Three row font	24 x 15 pixels
F4	Four row font	32 x 19 pixels
F5	Six row font	48 x 29 pixels

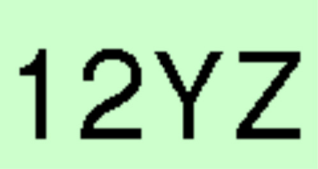
All fonts have a full 7-bit character set, except F5
All fonts have true decenders, except <F1>

Uses The <F5> command allows:

- Maximum visibility
- Eye-catching warnings when used with the <FL> flashing attribute
- Display of one critical process variable

Example

`<CS>`	Clear Screen
`<F5>`	Select font 5
`<CM7,0>`	Move cursor to the lower left of the display
`<WT12YZ>`	Write "12YZ"



```
12YZ
```

Gotchas! The cursor is homed by this command, and may need to be moved to the desired position before writing anything
F5 has a limited character set

See Also

DF*n*	Download Font
WS*n*	Write Soft Character
WT	Write Text

<FL> Flashing

Description Set the flashing attribute, so that any subsequently written text or graphic will flash when the global attribute <EF> is set.

Parameters None

Initial Value Steady (No Flashing)

Modes All Modes

Notes The <BMn> Background Mode attribute controls what background appears when the image flashes.

This attribute applies to all writes to the screen except bargraphs.

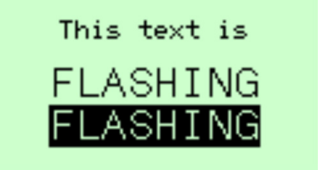
The Flashing attribute is cancelled by the <ST> STeady attribute.

Uses The <FL> command allows:

- Attention grabbing messages to be displayed
- Screens to be built with both flashing and non-flashing text and graphics
- Sending the <EF> command after the text is written ensures that all the text (written with the flashing attribute set) starts flashing at the same time.

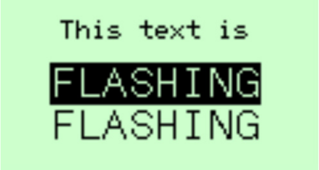
Example

`<SD>`	Set a known state for the display
`<CM1,0>`	Move the cursor to the first line down from the top of the screen
`<CA>`	All text is aligned centrally
`<WTThis text is>`	Write out the text
`<BM2>`	Set the flash background to the inverse of the foreground image
`<F2>`	Font size 2, 16 x 10 pixels
`<CM4,0>`	Move the cursor to the fourth line down from the top of the screen
`<FL>`	Set the Flashing attribute, so any text written will flash if flashing is enabled
`<WTFLASHING>`	Write out the text
`<CM6,0>`	Move the cursor to the sixth line down from the top of the screen
`<WM3>`	Write the foreground text as inverse (white on black background)
`<WTFLASHING>`	Write out some text
`<EF>`	Enable the flashing for the whole screen



This text is
FLASHING
FLASHING

Alternating each second with



This text is
FLASHING
FLASHING

Gotchas! Flashing messages with a blank background can cause the message to be missed on a glance at the display. If this could be a problem, use a flash background which is the inverse of the of the image <BM2> as in the example above.

Attributes are not saved and restored when screens are moved to and from memory with the <SFnm> and <RFm> commands

See Also

BM	Background Mode
EF	Enable Flashing
IF	Inhibit Flashing
ST	Steady

<FR>Font Restore

System

Description Recover previously stored soft fonts from EEprom

Parameters None

Modes All Modes

Notes This command recovers all the soft fonts in sizes F1 to F5 from EEprom, overwriting any that may have been downloaded, but not kept. This command is required as all currently defined soft fonts (except those stored in EEprom) are lost when power is removed from the instrument.

Fonts can only be recovered as an entire block. There is no provision for restoring a single soft font number, or a single font size.

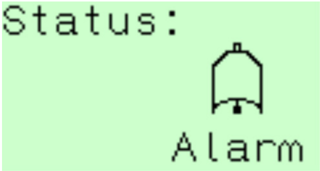
This command is used in conjunction with the Download Font <DF*n*> and Keep Fonts <KF> commands.

Uses The <FR> command allows:

- Time to be saved, rather than having to download soft font sets after a power down

Example

`<CS>`	Clear the screen
`<F2>`	Define the font size required
`<WTStatus:>`	Write out the text “Status:”
`<CM7,65>`	Bottom line of screen, 65 pixels from the left of the screen
`<WTAlarm>`	Write the text “Alarm”
`<FR>`	Recover soft fonts. **N.B.** The position of the command is unimportant. The command could have been issued at any time after power-up and before the Write Soft <WS> command.
`<F4>`	Choose the font size
`<CM5,80>`	Go to the position to write the character
`<WS3>`	Writes character number 3 (bell) in soft font size F4 to the screen



Status:

Alarm

Gotchas! This is an ‘all-or-nothing’ command – all fonts of all sizes are restored at once

Performing a <FR> Font Restore without first downloading and saving the desired characters will yield unpredictable results

See Also

DF	Download Font
KF	Keep Font

<FS> Fill Screen

Description Turn all pixels on, creating a black screen

Parameters None

Initial Value All pixels are off

Modes All Modes

Notes This command also:
Removes any windows that may be defined
(equivalent to issuing a <DW0,7,0,119> command)
Homes the cursor
(equivalent to issuing a <HC> command)

Uses The <FS> command provides:

- A known starting point before drawing a new screen

Example `<FS>` Fill Screen

Gotchas! If windows are being used, they must be defined after this command

See Also

CS	Clear Screen
CW	Clear Window
DW	Define Window
HC	Home Cursor

<FW> Fill Window

Description Turn all pixels on within a defined window

Parameters None

Initial Value All pixels are off

Modes Row Mode only

Notes This command also homes the cursor in the defined window area, equivalent to issuing a <HC> command.

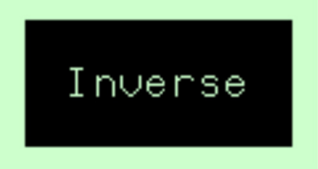
Apart from its main use in just filling the contents of a window, it can also be used to create inverse frames to contain text and graphics. Using this technique is much faster than using a Box Draw <BD> command in Pixel Mode.

Uses The <FW> command allows:

- A known starting point before updating a window
- Creation of a simple border

Example

`<RM>`	Set Row Mode
`<CW>`	Turn all screen pixels on
`<DW1,6,10,110>`	Define a window one row from top and bottom, 10 pixels in from both sides
`<FW>`	Set all the pixels in the window area to on
`<F2>`	Set required font size
`<CM3,0>`	Move the cursor to the third row down in the window
`<CA>`	Centre align the following text
`<WM3>`	Set inverse mode
`<WTInverse>`	Write out the text



Inverse

See Also

BD	Box Draw
CS	Clear Screen
DW	Define Window
HC	Home Cursor

<GB*n*> Graphic Block

System

Description This command specifies the graphics block that can be accessed through the GRAPHIC_DATA register when using the Modbus protocol

Parameters *n* = 0 to 16 - Block segment number

Initial Value *n* = 0

Modes All Screen Modes

Notes This command has no purpose if the Modbus protocol is not being used. A command error will be returned if an invalid attempt to use it is made.

Because of the register size, a maximum of 64 bytes of data may be sent over in a single transaction. Graphics data sent to and received by the display often need to be much larger than this, so the data has to be split into 64 byte segments.

The <GBn> parameter specifies which block of 64 bytes is being sent or read from the display

Graphics commands will ignore any data past the end of file in a block and data in any higher numbered blocks

Uses The <GB> command allows:

- The data length limitations in transferring graphics data to be overcome

Example

`<GB0>`	Set Graphics Block 0 Read/Write first 64 bytes of data to the display using the GRAPHIC_DATA parameter
`<GB1>`	Set Graphics Block 1 Read/Write the next 64 bytes of data to the display using the GRAPHIC_DATA parameter
`<GB2>`	Set Graphics Block 2 and so on until all the data has been transferred
`<DG>`	Now issue the command to process all previously transmitted blocks.

Gotchas! Can only be used in conjunction with the Modbus protocol
Make sure that the section on Graphics Transfer (Page 13) is read and understood

See Also

DF	Download Font
DG	Download Graphic
DS	Download Screen
US	Upload Screen

<HB*n*,*m*> Horizontal Bargraph

Description

Draw a horizontal bargraph *n* pixels long with *m* pixels filled

Parameters

n = 3 to 120	- Length of bargraph
m = 0 to *n*	- Number of filled pixels, starting from the left

Modes

Row Mode only

The <WM*n*> Write Mode has no effect on this command

Notes

The horizontal bargraph is drawn at the current cursor position.

The cursor is restored to its original position after the command.

The number of filled pixels has to be less than or equal to the overall length of the bargraph. Note that the first and last pixels are always filled in to form the frame, so <HB80,0> and <HB80,1> are visually identical, as are <HB80,79> and <HB80,80>

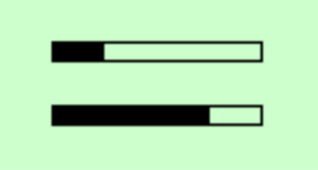
Uses

The <HB> command allows:

- Simple graphical representation of values or progress
- Bargraphs to be combined without restriction with other text and graphics

Example

`<SD>`	Return screen to known state
`<CM2,20>`	Move cursor to the second row down, 20 pixels from the left of the screen
`<HB80,20>`	Draw a horizontal bargraph 80 pixels long of which 20 pixels are filled. (25% fill)
`<CM5,20>`	Move the cursor to the fifth row down
`<HB80,60>`	Draw another horizontal bargraph 80 pixels long but this time with 60 pixels filled (75% fill)



Gotchas!

Bargraphs created by this command are static and are not automatically updated (See the DB command)

See Also

DB	Define Bargraph
EB	Erase Bargraph
VB	Vertical Bargraph

<HC> Home Cursor

Description Return the cursor to the top left of the screen

Parameters None

Modes All Modes

Notes This command is a special case of the <CM> Cursor Move command. The vertical position of the cursor is set such that the currently active font will display normally at the top left of the screen.

For example, with <F1> active <HC> is equivalent to <CM0,0>. Similarly with <F5> active <HC> is equivalent to <CM4,0> (in Row Mode)

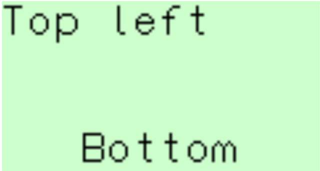
Home cursor is done automatically by commands such as <CS>, <FS>, <CW>, <FW> and setting any font size

Uses The <HC> command allows:

- Any subsequently written text to be easily positioned at the top of the display
- A starting point for constructing new screens

Example

`<SD>`	Put the display in a known state
`<F2>`	Set the font size
`<CM7,30>`	Cursor down to the bottom of the screen
`<WTBottom>`	Write out some text
`<HC>`	Home the cursor
`<WTTop left>`	Write out some text showing the effect on the cursor of the <HC> command



```
Top left


   Bottom
```

Gotchas! Make sure that the font size is selected before issuing the <HC> command

See Also

CS	Clear Screen
CW	Clear Window
F*n*	Font
FS	Fill Screen
FW	Fill Window

<HR*n,m,p*> Horizontal Rotate

Description Rotates the screen though the scratchpad RAM by one pixel column at a time.

Parameters

n = 0 or 1 - rotates the screen either Left (n = 0) or Right (n = 1)
m = 0 to 7 - The first row to scroll.
p = 0 to 7 - The last row to scroll

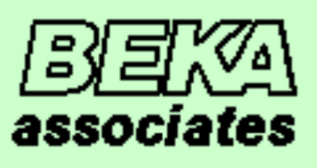
Modes Row Mode only

Notes This command rotates the screen one pixel column at a time through scratchpad ram and back again.

Uses The <HR> command allows:

- Visual effects suitable for demonstrations and exhibitions
- a moving screen to indicate that the host is functioning correctly even though no useful information is available to view

Example



Assume the default logo is displayed, and that we are writing to Frame 0

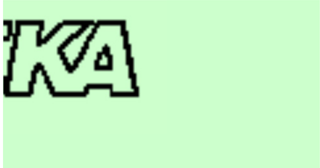
```
<SF1,2>
<CS>
```

Save this frame to scratchpad RAM
Now clear the current screen

Nothing is visible to the user, but the logo is still in scratchpad ram

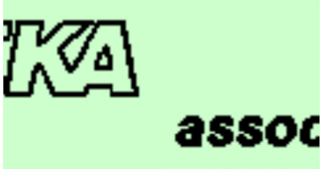
```
<HR1,0,4>
<HR1,0,4>
. . . .
<HR1,0,4>
```

Rotate the top 5 rows one pixel to the right
Repeat the command to rotate another pixel
... and repeat a total of 60 times



```
<HR0,5,7>
<HR0,5,7>
. . . .
<HR0,5,7>
```

Now rotate the lower 3 rows one pixel to the left
Repeat the command to rotate another pixel
... and repeat a total of 60 times



Gotchas! The command only rotates through one pixel. It must be repeated by the host as necessary.

See Also **HS** Horizontal Scroll

<HS*n,m,p,q,r,s,t*> Horizontal Scroll

Description Scrolls a defined area of the screen by one pixel

Parameters

n = 0 or 1	- scrolls the screen either Left (n = 0) or Right (n = 1)
m = 0 to 7	- The first row to scroll.
p = 0 to 7	- The last row to scroll
q = 0 to 64	- Starting position of line 1
r = 0 to 64	- Length of line 1
s = 0 to 64	- Starting position of line 2
t = 0 to 64	- Length of line 2

Modes Row Mode only

Notes This command scrolls a defined area of the screen, left or right by one pixel. In addition, two vertical lines of any length may be drawn in the 'new' pixel column.

The parameters *q* and *s* define the starting positions of these two new lines, in pixels above the bottom of Row *p.* The length of these lines are defined by the corresponding parameters *r* and *t*, again in pixels. These lines are drawn in the blank pixel column created by the left or right pixel block move. The lines may overlap if necessary.

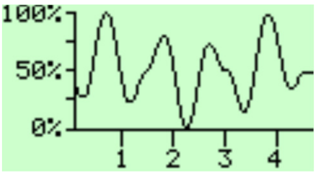
If no lines are required, set *q,r,s,t* to zero.

By default the command acts on the whole width of the screen, but as it is a window aware command, the effective width may be controlled by setting up a suitable window.

Uses The <HS> command allows:

- Line, bar, block charts that scroll with time
- Visual effects

Example To use this command effectively takes more commands than can easily be listed here. However consider the following screen which will illustrate the possibilities.



This illustrates the use of the command in displaying a trend graph. The 'y' axis is drawn with the <LH> and <LV> commands

Similarly, the initial 'x' axis is drawn in the same way, but the <HS> command can be used to 'move' the axis with the data if desired.

A window is set up just to the right of the vertical 'y' axis and the whole height of the screen.

The command <HS0,0,7,0,0,0,0> will scroll the graph area and the horizontal 'x' axis left by one pixel.

The line draw parameters are used to:

1. Draw in the next point on the graph
2. Draw in the 'x' axis and its marker lines

The scale values are created with normal cursor moves and write text commands

Gotchas! If the command is used in a window, the parameters are relative to that window.

See Also

DW	Define Window
HR	Horizontal Rotate
LH	Line Horizontal
LV	Line Vertical

<IF> Inhibit Flashing

Description Inhibit the automatic 1 second flash of any text or graphics drawn with the <FL> attribute

Parameters None

Initial Value Inhibited (No Flashing)

Modes All Modes

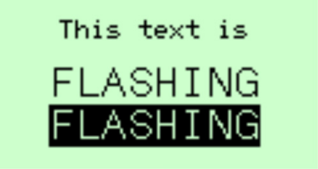
Notes This command acts on the whole screen.
Flashing can be re-enabled by using the <EF> command.

Uses The <IF> command allows:

- A menu structure to be built up of flashing and static screens
- A simple method of acknowledging operator input

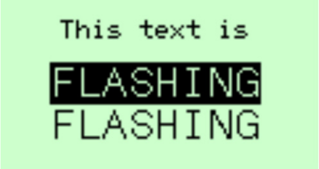
Example

`<SD>`	Set a known state for the display
`<EF>`	Enable the flashing for the whole screen
`<CM1,0>`	Move the cursor to the first line down from the top of the screen
`<CA>`	All text is aligned centrally
`<WTThis text is>`	Write out the text
`<BM2>`	Set the flash background to the inverse of the foreground image
`<F2>`	Font size 2, 16 x 10 pixels
`<CM4,0>`	Move the cursor to the fourth line down from the top of the screen
`<FL>`	Set the Flashing attribute, so any text written will flash if flashing is enabled
`<WTFLASHING>`	Write out the text
`<CM6,0>`	Move the cursor to the sixth line down from the top of the screen
`<WM3>`	Write the foreground text as inverse (white on black background)
`<WTFLASHING>`	Write out some text



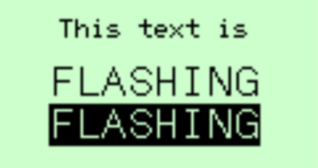
```
This text is
FLASHING
FLASHING
```

Alternating each second with



```
This text is
FLASHING
FLASHING
```

`<IF>` Now inhibit the flashing



```
This text is
FLASHING
FLASHING
```

(Steady image)

Gotchas! When this command is received, the foreground image will be immediately displayed, even if the background was actually on screen at that time

See Also

EF	Enable Flashing
FL	Flashing
IF	Inhibit Flashing
ST	Steady

<KF>Keep Fonts

System

Description Save previously download soft fonts to EEprom

Parameters None

Modes All Modes

Notes This command causes all the soft fonts in font sizes F1 to F5 to be saved to EEprom.

Downloaded soft fonts not stored in this way are lost when power is removed.

It is not possible to save just an individual soft font number or even all the soft fonts in a given size. Soft fonts are restored with the <FR> command. This is not done automatically on power-up.

The font data is written as a block and will overwrite any previously stored data.

To add a soft font definition to the current stored values they should be restored to the instrument memory first. The new font can then be downloaded and the entire new font set re-saved

Uses The <KF> command allows:

- A quicker method of providing soft fonts after power-on.

Example

`<FR>`	Get any existing font data
`<F4>`	Set required font size
`<DF3>`	Tell the display to expect a BMP file download of font data
`Binary download of 32 x 19 pixel image`	Send the file
`<KF>`	Save fonts

Gotchas! Fonts are not restored automatically on power-up

See Also

DF	Download Font
FR	Font Restore

<LA>Left Align

Description Set the attribute so that written text is aligned to the left of the display or defined window

Parameters None

Initial Value Not aligned; Text appears at the current cursor position

Modes All Modes

Notes This command sets the attribute that causes text written with the <WT> command to be aligned at the left hand side of the screen (or window, if defined).

It only affects text written after the attribute has been set.

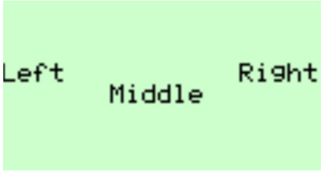
The attribute is cancelled by the <NA> command or any of the other text alignment commands <CA>, <RA>, <SW> & <TW>

Uses The <LA> command allows:

- Text to be automatically aligned without the need for cursor move commands
- Tidy screen presentation

Example

`<SD>`	Set the display to a known state
`<CM3,60>`	Move the cursor to the middle of row 3
`<LA>`	Set left align attribute
`<WTLeft>`	Left align the word 'Left' on the current row
`<RA>`	Set the right align attribute
`<WTRight>`	Right align the word 'Right' on the current row.
`<LN>`	Move cursor one row down
`<CA>`	Set centre align attribute
`<WTMiddle>`	The word 'Middle' is written centre aligned on the current row.



```
Left                Right
       Middle
```

See Also

CA	Centre Align
NA	No Align
RA	Right Align
SW	Smart Wrap
TW	Text Wrap

<LF> Line Feed

Description Add a line feed character after a carriage return character has been received

Parameters None

Initial Value This attribute is cleared; Line Feed and Carriage Return are independent actions

Modes Row Mode only

Notes This command causes the display to add a line feed character after a carriage return character has been received.

This has the effect of moving the cursor to the beginning of the next row down when a single 'carriage return' character (13 decimal, 0x0D in hex) is received.

If the cursor is already on the bottom line of the display or window, the current screen is scrolled up one line and the cursor positioned at the beginning of the bottom line.

The <NL> command cancels this attribute, making LF and CR independent actions. <NL> is the default condition.

Uses The <LF> command allows:

- The display to be used as a dumb terminal
- Hosts that only send CR instead of CR+LF to be accommodated

See Also **NL** No LineFeed

<LH*x*,*l*> Line Horizontal

Description Draw a horizontal line *x* pixels long with a line thickness of *l*

Parameters

x = 1 to 120	- length
l = 1 to 64	- line thickness

Modes Pixel mode only

Notes The line is drawn from the current cursor position upwards and to the right.

The cursor position is unchanged after the command

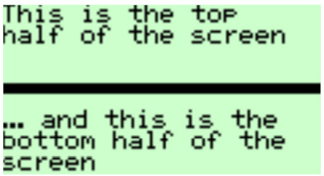
The parameters may be any value that will keep the line being drawn on-screen. If any part of the defined line is off-screen, then the line is not drawn and an error response it returned to the host.

Uses The <LH> command allows:

- information to be segmented
- borders to be drawn
- line images to be constructed

Example

`<SD>`	Set the display to a known state
`<PM>`	Set display to Pixel Mode
`<CM33,0>`	Move the cursor to pixel row 33, at the left of the screen
`<LH120,4>`	Draw a horizontal line 120 pixels long and 4 pixels wide
`<RM>`	Back to Row Mode
`<SW>`	Turn Smart Wrap attribute on. Text wraps without splitting words
`<HC>`	Home the cursor to top left of screen
`<WTThis is the top half of the screen>`	Write out some text
`<CM5,0>`	Cursor move to sixth row down
`<WT … and this is the bottom half of the screen>`	Write out some more text



```
This is the top
half of the screen

… and this is the
bottom half of the
screen
```

Gotchas! The entire line must fit on the screen, otherwise nothing will be drawn and an error response generated

See Also

BD	Box Draw
LV	Line Vertical

<LN>Line New

Description Send a 'CR + LF' to move the cursor down one line and to the left hand side of the screen or window

Parameters None

Modes Row Mode Only

Notes This command sends a 'Carriage Return' + 'Line Feed' to the display so that the cursor is moved down one line and to the left hand side of the screen or window.

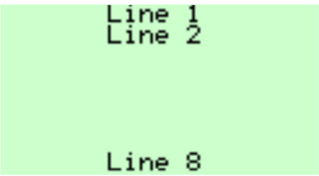
If the cursor is already on the bottom line the display will scroll up one line, leaving the cursor on the new bottom line.

Uses The <LN> command allows:

- A vertical scroll of text (and graphics) to occur if the cursor is already on the bottom line
- A quicker but more limited version of the Cursor Move command

Example

`<SD>`	Set the display to a known state
`<CA>`	Align all following text centrally
`<WTLine 1>`	Write some text
`<LN>`	Move the cursor down one line
`<WTLine 2>`	Write some more text
`<CM7,0>`	Move to the bottom line
`<WTLine 8>`	Write some more text



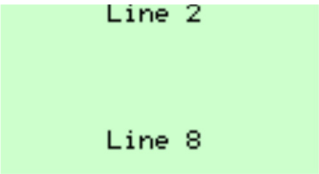
```
Line 1
Line 2





Line 8
```

`<LN>`	Move the cursor down one line



```
Line 2





Line 8
```

See Also **SW** Smart Wrap

<LV*y*,*l*> Line Vertical

Description Draw a vertical line *y* pixels high with a line thickness of *l*

Parameters

y = 1 to 64	- height
l = 1 to 120	- line thickness

Modes Pixel mode only

Notes The line is drawn from the current cursor position upwards and to the right.

The cursor position is unchanged after the command

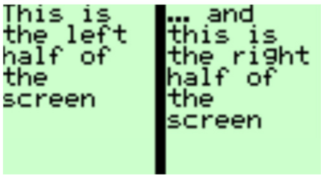
The parameters may be any value that will keep the line being drawn on-screen. If any part of the defined line is off-screen, then the line is not drawn and an error response it returned to the host.

Uses The <LV> command allows:

- information to be segmented
- borders to be drawn
- line images to be constructed

Example

`<SD>`	Set the display to a known state
`<PM>`	Set display to Pixel Mode
`<CM63,58>`	Move the cursor to pixel row 63, in the middle of the screen
`<LV64,4>`	Draw a vertical line 64 pixels long and 4 pixels wide
`<RM>`	Back to Row Mode
`<DW0,7,0,57>`	Define a window on the left half of the screen
`<SW>`	Turn Smart Wrap attribute on. Text wraps without splitting words
`<HC>`	Home the cursor to top left of window
`<WTThis is the left half of the screen>`	Write out some text
`<DW0,7,63,119>`	Define a window on the right half of the screen
`<HC>`	Home the cursor to the top left of the window
`<WT … and this is the right half of the screen>`	Write out some more text



```
This is      … and
the left     this is
half of      the right
the          half of
screen       the
             screen
```

Gotchas! The entire line must fit on the screen, otherwise nothing will be drawn and an error response generated

See Also

BD	Box Draw
LH	Line Horizontal

<MC*n*> Make Connection

Description The following commands are intended for the instrument with address ‘*n*’

Parameters *n* = 1 to 247 - address range

Modes This command is used in multidrop or multiple instrument configurations

Notes Only the instrument with address ‘*n*’ will acknowledge this command. Each instrument must have a unique address; commands cannot be ‘broadcast’ to several displays at once.

This command remains in force until cancelled by a <RC> Release Connection command. After an <RC> command has been confirmed by the currently active instrument, no instruments will respond to any commands until a further <MC*n*> command is sent to a valid instrument.

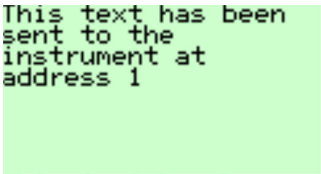
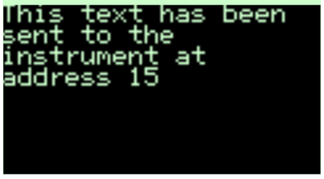
Multiple instrument configurations must send a valid <MC*n*> command when powered up as all instruments with a non-zero address will initially assume they are not ‘connected’.

Single instrument configurations with address 0 will return an error response to this command.

Uses The <MC> command allows:

- Multiple instruments to be connected to one host port

Example

`<MC1>`	Make connect to the instrument with address 1
`<CS>`	Clear the screen on instrument address 1
`<SW>`	Set the smart wrap attribute on instrument address 1
`<WTThis text has been sent to the instrument at address 1>`	Send this text to the instrument with address 1
`<RC>`	Release the ‘connection’ to instrument 1
`<MC15>`	Make a ‘connection’ to the instrument with address 15
`<WM3>`	Set inverse write mode on the instrument at address 15
`<FS>`	Fill the screen on instrument address 15
`<SW>`	Set the smart wrap attribute on instrument address 15
`<WTThis text has been sent to the instrument at address 15>`	Send this text to the instrument with address 15
`<RC>`	Release the ‘connection’ to instrument 15
 This text has been sent to the instrument at address 1	Instrument address 1:
 This text has been sent to the instrument at address 15	Instrument address 15:

Gotchas! There is no such thing as a ‘broadcast address’.

See Also **RC** Release Connection

<NA> No Align

Description Cancel all of the text alignment attributes <LA>, <RA>, <CA>, <SW> and <TW>

Parameters None

Initial Value This is the default

Modes All Modes

Notes This command clears all alignment attributes so that text written with the <WT> command appears at the current cursor position.

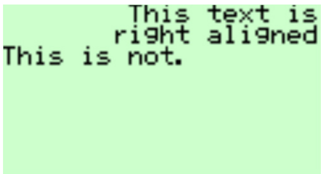
It only affects text written after the attribute has been set.

Uses The <NA> command allows:

- manual formatting after special alignment attributes have been used

Example

`<SD>`	Set screen to known state
`<RA>`	Set right alignment attribute on
`<WTThis text is>`	Write out some text
`<LN>`	Cursor to next line down, left of screen
`<WTright aligned>`	Write some more text
`<LN>`	Cursor to next line down, left of screen
`<NA>`	Cancel text alignment attribute
`<WTThis is not.>`	Write some text, this time it appears at the current cursor position.



```
        This text is
       right aligned
This is not.
```

Gotchas! <NA> also cancels <SW> Smart Wrap and <TW> Text Wrap

See Also

CA	Centre Align
LA	Left Align
RA	Right Align
SW	Smart Wrap
TW	Text Wrap

<NL>No Linefeed

Description Cancel the automatic execution of a ‘CR + LF’ when just a single ‘CR’ is received

Parameters None

Initial Value This is the default

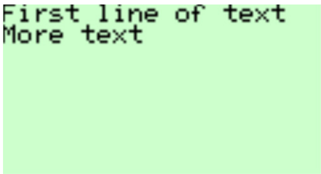
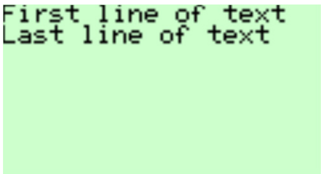
Modes All Modes

Notes This command reverses the action of the <LF> command by cancelling the automatic execution of a ‘carriage return’ + ‘linefeed’ when just a single ‘carriage return’ is received.

Uses The <NL> command allows:

- The display to be used as a dumb terminal
- Hosts that send CR and LF separately to be accommodated

Example

`<SD>`	Set screen to known state
`<LF>`	Set Linefeed attribute on
`<WTFirst line of text>`	Write a line of text
`<WT[CR]>`	Send a [CR] character, with the Line Feed attribute set, this would be interpreted as [CR]+[LF] ***Note! The square brackets are not sent, they are just there to show that a Carriage Return character (ASCII 13) is sent.***
`<WTMore text>`	This text written on the line below
 First line of text More text	
`<NL>`	Turn off line feed attribute
`<WT[CR]>`	Send another [CR] character
`<WTLast line of text>`	As the Line Feed attribute has been turned off, the display has only actioned the [CR] so this text overwrites the “More Text” string sent earlier.
 First line of text Last line of text	

See Also **LF** Line Feed

<NS>New Screen

Mapped Variables

Description Clears the screen and removes all mapped variables, bargraphs and cyclic text strings

Parameters None

Initial Value None

Modes All Modes

Notes The <NS> command is functionally equivalent to issuing the three commands <CS><EV0><EB0> in succession.

This command also:

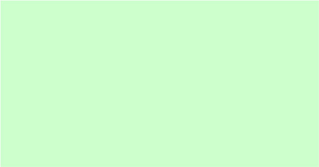
- Removes any windows that may be defined
  (equivalent to issuing a <DW0,7,0,119> command)
- Homes the cursor
  (equivalent to issuing a <HC> command)

Uses The <NS> command allows:

- An efficient method of clearing the screen after mapped variables have been used

Example

```
<NS>                    New Screen
```



Gotchas! If windows are being used, they must be defined after this command

See Also

CS	Clear Screen
EB	Erase Bargraph
EV	Erase Variable

<NU> No Underline

Description Cancel the <UL> Underline attribute

Parameters None

Initial Value This is the default

Modes All Modes

Notes This command cancels the ‘Underline’ attribute so that text written with the <WT> command appears without being underlined

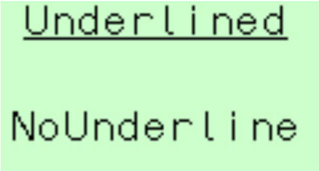
It only affects text written after the attribute has been set.

Uses The <NU> command allows:

- A combination of underlined and plain text to appear on the same screen

Example

<SD>	Set screen to known state
<CA>	Centre align the text
<UL>	Set Underline attribute on
<F2>	Choose a font size (not F1)
<WTUnderlined>	Write out some text that is underlined
<NU>	Cancel the underline attribute
<CM6,0>	Move the cursor down
<WTNoUnderline>	Write out some more text which is not underlined



Underlined

NoUnderline

See Also **UL** Underline

<OD*n*> Output De-energised

System

Description Control the state of the output contacts, making it de-energised

Parameters *n* = 1 or 2 - Output number

Initial Value De-energised (open circuit) on power up

Modes All Modes

Notes These commands allow the user to control the state of the output contacts.
There are two isolated solid state contacts per display, A1 – A2 and A3 – A4

The parameter *n* selects which output is being controlled:
n = 1 controls the output A1-A2
n = 2 controls the output A3-A4

The command <OD*n*> turns off (de-energises) output *n*
The command <OE*n*> turns on (energises) output *n*

Uses The <OD> command allows:

- The display to control alarms, annunciators, sounders etc. under program control

Example

`<OE1>`	Output A1 – A2 is energised (short circuit)
`<OE2>`	Output A3 – A4 is energised (short circuit)
`<OD1>`	Output A1 – A2 is de-energised (open circuit)
`<OD2>`	Output A3 – A4 is de-energised (open circuit)

There is no effect on the display LCD screen when these commands are used

See Also **OE** Output Energised

<OE*n*> Output Energised

Description Control the state of the output contacts, making it energised

Parameters *n* = 1 or 2 - Output number

Initial Value De-energised (open circuit) on power up

Modes All Modes

Notes These commands allow the user to control the state of the output contacts.
There are two isolated solid state contacts per display, A1 – A2 and A3 – A4

The parameter *n* selects which output is being controlled:
n = 1 controls the output A1-A2
n = 2 controls the output A3-A4

The command <OE*n*> turns on (energises) output *n*
The command <OD*n*> turns off (de-energises) output *n*

Uses The <OE> command allows:

- The display to control alarms, annunciators, sounders etc. under program control

Example

`<OE1>`	Output A1 – A2 is energised (short circuit)
`<OE2>`	Output A3 – A4 is energised (short circuit)
`<OD1>`	Output A1 – A2 is de-energised (open circuit)
`<OD2>`	Output A3 – A4 is de-energised (open circuit)

There is no effect on the display LCD screen
when these commands are used

See Also **OD** Output De-energised

<PM> Pixel Mode

System

Description Put the unit into Pixel Mode

Parameters None

Initial Value Row Mode

Modes All Operational Modes

Notes This command allows all text to have pixel positional resolution in both vertical and horizontal directions, rather than being constrained into rows as with Row Mode.

Most graphics commands require the display to be in Pixel Mode.

The vertical parameters for the cursor move command <CM> are 0 to 63 when in Pixel Mode.

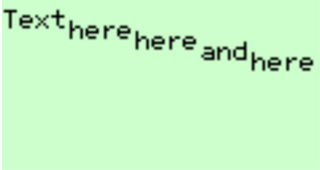
Pixel modes writes to the screen are always slower than the corresponding Row Mode write. It is recommended that Row Mode operations are used whenever possible to optimise the response time. Alternatively, complex screens can be written to the non-active frame and then made visible; This gives the appearance of a fast redraw after a short pause.

Uses The <PM> command allows:

- Flexibility of text and graphics positioning
- Tidy screen presentation

Example

`<PM>`	Set Pixel mode
`<CM11,1>`	Move the cursor to Line 11, Row 1
`<WTText>`	Write the word 'Text'
`<CM15,26>`	Move the cursor to Line 15, Row 26
`<WThere>`	Write the word 'here'
`<CM19,51>`	Move the cursor to Line 19, Row 51
`<WThere>`	Write the word 'here'
`<CM23,76>`	Move the cursor to Line 23, Row 76
`<WTand>`	Write the word 'and'
`<CM27,95>`	Move the cursor to Line 27, Row 95
`<WThere>`	Write the word 'here'



Text here here and here

Gotchas! Pixel mode is much slower than Row mode

The <PM> Pixel Mode command clears any currently defined window

See Also

AF	Active Frame
RM	Row Mode
VF	Visible Frame

<RA> Right Align

Description Set the attribute so that written text is aligned to the right of the display or defined window

Parameters None

Initial Value Not aligned; Text appears at the current cursor position

Modes All Modes

Notes This command sets the attribute that causes text written with the <WT> command to be aligned at the right hand side of the screen (or window, if defined). Effectively, the horizontal cursor position is ignored and the text is automatically positioned such that it ends on the right hand edge.

It only affects text written after the attribute has been set.

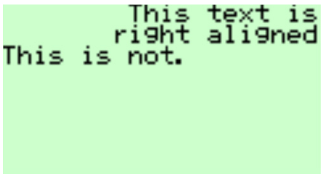
The command is cancelled by the <NA> command or any of the other text alignment commands <CA>, <LA>, <SW> & <TW>

Uses The <RA> command allows:

- Labelling the right hand ‘soft keys’
- Constraining text away from text or images on the left of the screen
- Text to be automatically aligned without the need for cursor move commands

Example

`<SD>`	Set screen to known state
`<RA>`	Set right alignment attribute on
`<WTThis text is>`	Write out some text
`<LN>`	Cursor to next line down, left of screen
`<WTright aligned>`	Write some more text
`<LN>`	Cursor to next line down, left of screen
`<NA>`	Cancel text alignment attribute
`<WTThis is not.>`	Write some text, this time it appears at the current cursor position.



```
       This text is
      right aligned
This is not.
```

See Also

CA	Centre Align
LA	Left Align
NA	No Align
SW	Smart Wrap
TW	Text Wrap

<RB> Reboot

System

Description Cause a complete restart of the instrument, just as if it had been powered up

Parameters None

Modes All Modes

Notes This command causes a complete restart of the instrument, just as if it had been powered up after being switched off.

The receipt of this command is acknowledged in the normal way and then the instrument is restarted by causing a deliberate watchdog timeout.

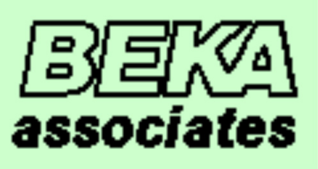
This can be used to force a complete restart of the instrument, which may be needed if the host and display are independently powered.

An alternative is to issue the <SD> Screen Defaults command, which simply initialises the display to a known state

Uses The <RB> command allows:

- The entire instrument to be put into a known state

Example

`<RB>`	Reboot the instrument A delay of up to 2 seconds may elapse before the watchdog timeout restarts the hardware
	The unit reads its configuration settings from EEprom and displays the power-on logo

Gotchas! Soft fonts must be restored with the <FR> Font Restore command after a reboot

See Also **SD** Screen Default

<RC> Release Connection

System

Description Disconnect the currently ‘connected’ instrument

Parameters None

Initial Value All instruments with non-zero addresses power up with no connection active

Modes This command is used in multidrop or multiple instrument configurations

Notes After an <RC> command has been confirmed by the currently active instrument, no instruments will respond to any commands until a further <MC*n*> command is sent to a valid instrument.

Multiple instrument configurations must send a valid <MC*n*> command when powered up as all instruments with a non-zero address will initially assume they are not ‘connected’.

Single instrument configurations with address 0 will return an error response to this command.

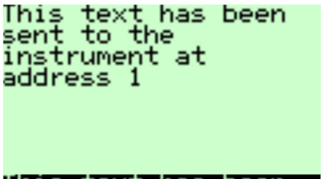
Uses The <RC> command allows:

- Multiple instruments to be connected to one host port

Example

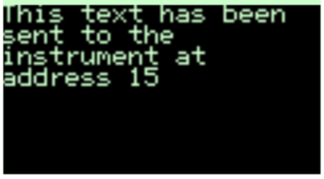
`<MC1>`	Make connect to the instrument with address 1
`<CS>`	Clear the screen on instrument address 1
`<SW>`	Set the smart wrap attribute on instrument address 1
`<WTThis text has been sent to the instrument at address 1>`	Send this text to the instrument with address 1
`<RC>`	Release the ‘connection’ to instrument 1
`<MC15>`	Make a ‘connection’ to the instrument with address 15
`<WM3>`	Set inverse write mode on the instrument at address 15
`<FS>`	Fill the screen on instrument address 15
`<SW>`	Set the smart wrap attribute on instrument address 15
`<WTThis text has been sent to the instrument at address 15>`	Send this text to the instrument with address 15
`<RC>`	Release the ‘connection’ to instrument 15

Instrument address 1:



```
This text has been
sent to the
instrument at
address 1
```

Instrument address 15:



```
This text has been
sent to the
instrument at
address 15
```

See Also **MC** Make Connection

<RF*n*> Restore Frame

Description Restore a previously saved frame to the currently active frame

Parameters *n* = 0 to 2 - Saved Frame memory location

Modes The <WM*n*> Write Mode has no effect on this command

Notes This command restores a frame image saved with the <SF> command to the currently active frame.

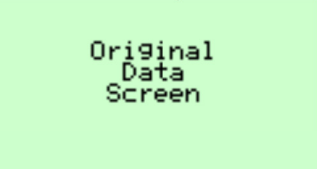
The parameter *m* specifies which memory location the stored frame is recovered from:
n = 0 specifies EEprom area 0
n = 1 specifies EEprom area 1
n = 2 specifies the scratchpad area in RAM

The scratchpad area is faster than the EEprom areas, but must be used with care as some commands will overwrite this location. See the <SF> Save Frame command for details

Uses The <RF> command allows:

- Images to be transferred from one frame to another
- Any data screen to be flashed using the sequence <SF0,2><FL><EF><BM2><RF2>

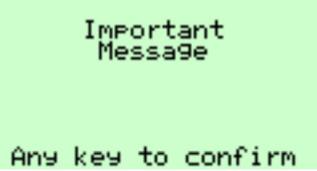
Example



Original
Data
Screen

Assume the display is showing some data, and our active frame is set to the same value as the visible frame

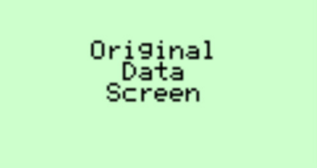
`<SF0,2>`	Save frame 0 to scratchpad RAM
`<CS>`	Clear screen for new message
`<LN>`	Move down a line
`<WTImportant>`	Write out a message….
`<LN>`	
`<WTMessage>`	
`<CM7,0>`	
`<WTAny key to confirm>`	



Important
Message

Any key to confirm

`<RS>`	Wait for an operator response. N.B. Loop sending this command until either a response or timeout
`<RF2>`	Restore the original screen from scratchpad



Original
Data
Screen

Gotchas! Attributes are not restored
The currently Active Frame may not be the currently Visible Frame

See Also

AF	Active Frame
SF	Save Frame
VF	Visible Frame

<RL*n*> Restore Logo

Description Restore a logo that has been saved with the <SL> command

Parameters *n* = 0 or 1 - Static or Scrolled

Initial Value There is a default “BEKA associates” logo built in that appears if no user logo is defined. (OEM versions of the product may have an alternative logo instead)

Modes All Modes

Notes This command is used to restore a logo that has been saved with the <SL> command. The parameter *n* specifies whether the logo scrolls, as on power up.
n = 0 specifies no scrolling
n = 1 specifies logo should scroll

Scrolling will start after 20 seconds and pause for 10 seconds between each screen scroll

If there is no saved logo, this command will restore the default BEKA logo.

The logo is always recovered to the current visible frame, overwriting the frame contents. Note this command is the only command that does not write to the current active frame.

Uses The <RL> command allows:

- A customised logo to appear if the system is not being used, or there have been no messages for a period of time
- A scrolling logo to reassure operators that the display is still functioning correctly, without any host programming

Example `<RL1>` Display the logo with scrolling enabled

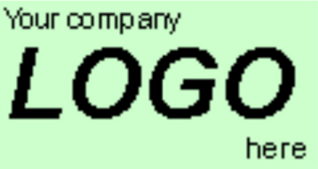


Image is displayed when received

Gotchas! This command is the only one that does not write to the currently active frame.

See Also

DS	Download Screen
RF	Restore Frame
SL	Save Logo

<RM> Row Mode

System

Description Put the unit into Row Mode

Parameters None

Initial Value Row Mode

Modes All Operational Modes

Notes This command enables Row Mode. In this mode the screen is split up into eight horizontal rows each eight pixels high. Text is then aligned with these rows

In this mode the vertical position in the Cursor Move command it limited to 0 to 7.

Windows are available in Row Mode to constrain and align text.

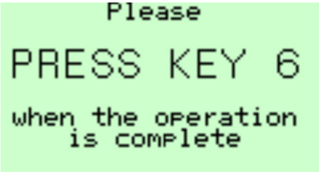
Writes to the display in Row Mode are always faster than Pixel Mode operation, and should be used wherever possible

Uses The <RM> command allows:

- Rapid display of text messages
- Simple text alignment

Example

`<CS>`	Clear screen for new message
`<RM>`	Set Row Mode
`<CA>`	Centre align the text
`<WTPlease>`	Write out message….
`<F2>`	Use a larger font size
`<CM3,0>`	Move the cursor to row 3
`<WTPRESS KEY 6>`	Write more text
`<F1>`	Back to the small font
`<CM5,0>`	Move the cursor to row 5
`<WTwhen the operation>`	Write out more text
`<LN>`	Next line down
`<WT is complete>`	Write out final line of text



```
         Please
PRESS KEY 6
when the operation
    is complete
```

See Also

PM	Pixel Mode
DW	Define Window

<RS>Request Status

Description Get key-press status information from the display

Parameters None

Initial Value None

Modes All Modes

Notes This command is used to get key-press status information from the display. It has no effect on the screen or any of the display settings.

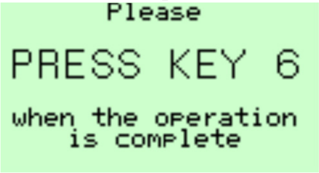
This command was included primarily to be able to read the keys in Operational Mode 0, where there is not normally a response to commands. However, it works in any mode and can be used in a loop waiting for an operator key-press.

Uses The <RS> command allows:

- Operator feedback
- The only method for checking the keys in Operational Mode 0

Example

`<CS>`	Clear screen for new message
`<RM>`	Set Row Mode
`<CA>`	Centre align the text
`<WTPlease>`	Write out message….
`<F2>`	Use a larger font size
`<CM3,0>`	Move the cursor to row 3
`<WTPRESS KEY 6>`	Write more text
`<F1>`	Back to the small font
`<CM5,0>`	Move the cursor to row 5
`<WTwhen the operation>`	Write out more text
`<LN>`	Next line down
`<WT is complete>`	Write out final line of text



Please

PRESS KEY 6

when the operation
is complete

`<RS>`	Wait for an operator response. N.B. Loop sending this command until either a response or timeout

See Also

CI	Command Implement
CC	Check Code
CR	Cyclic Redundancy check

<SA*n*> Speed Adjust

System

Description Adjusts the processor clock speed at the expense of the backlight intensity

Parameters *n* = 0 to 2 - 0 = default speed, 1 = 2 x default, 2 = 4 x default

Initial Value 0 = default speed (BA4xx Intrinsically Safe units only)
2 = 4 x default (BA6xx Safe Area units only)

Modes All modes

Notes The <SAn> command is only available for I.S. units connected as a single unit within a hazardous area. Safe area units (BA6xx) have no power limitations already operate at the fastest speed possible.

Increasing the processor clock speed increases the current taken by the circuitry, so the maximum backlight intensity is reduced to compensate.

The speed of internal processing and screen redrawing is increased, but all other timing parameters remain unchanged (e.g communication speed, flash rate)

In order to maintain code compatibility between different models, the <SA*n*> command does not return an error if the speed can not be adjusted.

Uses The <SA> command allows:

- Complex screen redraws to be completed in less time
- The speed can be increased and decreased at will, to suit the application

Example Self explanatory

See Also **RB** ReBoot
SB Set Backlight

<SB*n*> Set Backlight

System

Description Alter the intensity of the backlight

Parameters *n* = 0 to 40 - Backlight Intensity

Initial Value Dependant on setting made in configuration menus

Modes All Modes

Notes This command alters the intensity of the backlight depending on the parameter *n*:

n = 0 backlight off.
n = 40 backlight fully on.

The actual brightness of the backlight depends on the single/multiple unit configuration. See the instruction manual for further information

The new backlight intensity is not saved in EEprom. If permanent changes to the backlight intensity are required, use the configuration or quick access menus

The <RB> ReBoot command restores the backlight to the default value as part of the initialisation process

Uses The <SB> command allows:

- The backlight to be flashed to attract attention
- Panel illumination to be controlled by the host

Example

`<SB0>`	Turn the backlight off
`<SB40>`	Turn the backlight to full intensity

Gotchas! The current backlight intensity cannot be read back from the display, nor can the defaults be changed by the host

See Also

RB	ReBoot
SA	Speed Adjust

<SD>Screen Defaults

Description Cancels all attributes and returns the display to a known configuration

Parameters None

Initial Value This is the default at power up

Modes All Modes

Notes This command behaves as if the following commands were received by the display:

<AF0>	Active frame = 0
<VF0>	Visible frame = 0
<F1>	Small font 8 x 6 Pixels
<CS>	Clear Screen
<HC>	Cursor homed
<WM0>	Normal text
<RM>	Row Mode
<IF>	Inhibit Flashing
<ST>	Text Steady attribute
<NA>	No Text Alignment or Wrap
<BM0>	Background Mode = 0
<NU>	No Underline

As a consequence, the screen is cleared, window definitions are removed, display scrolling is turned off and key press data cleared

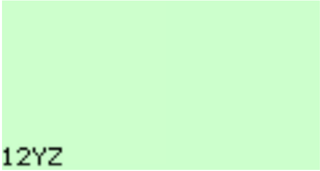
Uses The <SD> command allows:

- A known starting point for the creation of each screen

Example

`<SD>`	Set Screen defaults

`<CM7,0>`	Move cursor to the lower left of the display
`<WT12YZ>`	Write “12YZ”



```
12YZ
```

Gotchas! Use the <CS> Clear Screen for a less drastic initialisation

See Also

CS	Clear Screen
RB	ReBoot

<SF*n,m*> Save Frame

Description Save the specified frame *n* to memory location *m*

Parameters

n = 0 or 1	- frame number
m = 0 to 2	- memory location

Initial Value None

Modes All Modes

Notes The save frame command allows the specified frame *n* to be saved to memory location *m*.

m = 0 saves the frame m to EEprom area 0
m = 1 saves the frame m to EEprom area 1
m = 2 saves the frame m to scratchpad RAM

Saved frames are restored with the <RF*n*> command.

If more non-volatile frame storage is required, the <SL> Save Logo command can be used, but a frame saved using this command is automatically displayed on power-on.

The scratchpad RAM area is also used by the following commands:
<DL><DG><RB><SL><RL><LH><LV><BD><DF>
Use of any of these commands will corrupt a saved image in scratchpad ram.

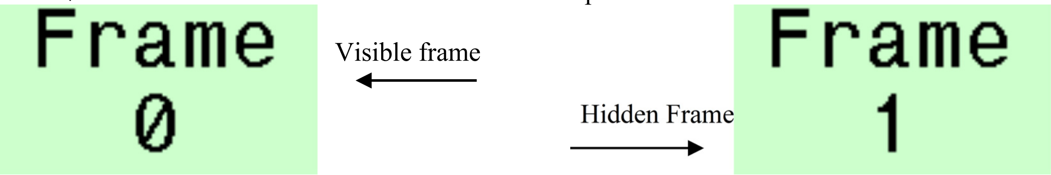
Detailed information about the use of frames can be found in the Frames Section (Page 7).

Uses The <SF> command allows:

- Complex screen backdrops to be saved, to which live data can then be added
- Temporary frame storage while another message is displayed
- Images to be moved between frames
- Normally static frames to flash, by saving them and then restoring them with the <FL> and <EF> attributes turned on. This is a simple way of indicating an alarm condition.

Example 1

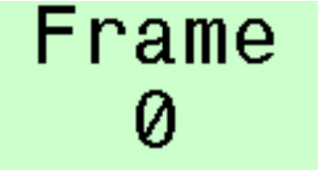
`<SD>`	Start with the active frame and visible frame set to 0
`<CS>`	Clear frame 0
`<F4>`	Set the required font
`<CA>`	Let the display centre the text automatically
`<WTFrame>`	Write out the word "Frame"
`<LN>`	Down a row
`<WT0>`	Write out the number "0"
`<AF1>`	Switch to the hidden frame
`<CS>`	Clear the hidden frame
`<WTFrame>`	Write out the word "Frame"
`<LN>`	Down a row
`<WT1>`	Write out the text to the hidden frame; LCD display screen unaltered
`<SF0,1>`	Save frame 0 to EEprom area 1
`<SF1,2>`	Save frame 1 to scratchpad RAM.



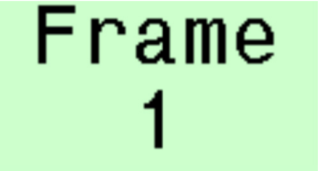
Example 2

`<SD>` Sets active and visible frames to 0 and clears the screen

`<RF1>` The text "Frame 0" is restored to the screen from EEprom


Frame
0

<RF2> The text “Frame 1” is restored to the screen from scratchpad RAM


Frame
1

Note: If this last sequence is repeated after the power has been removed and restored, then only the RF0 will restore the saved image correctly as the scratchpad ram contents will be undefined.

Gotchas!

Make sure that the section on Frames (Page 7) is read and understood

Be aware of the limitations regarding scratchpad RAM – unexpected results may easily occur

Frame *n* may or may not currently be visible. Use the <VF> command to achieve the desired result

See Also

RF	Restore Frame
VF	Visible Frame

<SH*n*> Status Hide

Description This command controls the visibility of the data status message on standard screens 1 and 4.

Parameters *n* = 0 - Status shown
n = 1 - Status hidden

Initial Value Set by the configuration menu

Modes All Screen Modes

Notes This command function is also found in the configuration menu

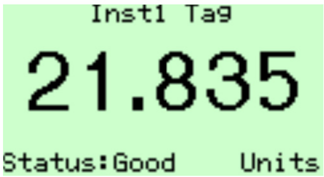
See the "Standard Screens" section on Page 15.for further details.

Uses The <SH> command allows:

- The displayed screen to be simplified where the extra text could be confusing

Example

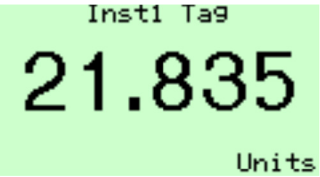
```
<SO1>
<SH0>
```



```
Inst1 Tag
21.835
Status:Good     Units
```

Standard screen, single variable displayed at a time
Status shown

```
<SH1>
```



```
Inst1 Tag
21.835
                Units
```

Standard screen, two variables displayed at a time
Status hidden

Gotchas! The local configuration menu can also be used to alter the visibility of the data status.
The value is still displayed in inverse video if the status is "Bad"

<SL> Save Logo

System

Description Save the currently visible frame contents as the power-on logo

Parameters None

Initial Value There is a default “BEKA associates” logo built in that appears if no user logo is defined. (OEM versions of the product may have an alternative logo instead)

Modes All Modes

Notes The screen may be drawn using the text and graphics commands or simply downloaded from the host as a .BMP file using <DS> or <DG>

A saved logo can be overwritten at any time by issuing another <SL> command.

If a user logo is no longer required, then clear the screen and execute the <SL> command. This will restore the default BEKA logo

Uses The <SL> command allows:

- A customised logo to appear at power on

Example

<CS> Clear Screen

<DS> Tell the display to expect a 64 x 120 pixel graphics image that it should display full screen.

Binary download of .BMP file is now sent

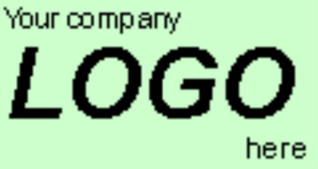


Image is displayed when received

<SL> Save Logo to EEprom

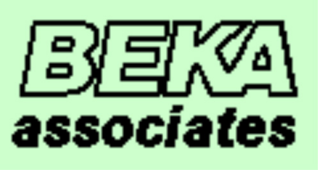
<RB> ReBoot the display

User logo is shown (and scrolled) on power up

<CS> Clear Screen

<SL> Save Logo to EEprom

<RB> ReBoot the display



Default BEKA logo is shown (and scrolled) on power up

See Also

DS Download Screen

DG Download Graphic

<SO*n*> Screen Option

Description This command allows the screen type to be changed remotely.

Parameters

n = 0	- Text Display Mode: programmable to generate custom screens
n = 1	- Standard screen; single variable displayed
n = 2	- Standard screen; two variables displayed
n = 3	- Standard screen; four variables displayed
n = 4	- Standard screen; single variable and a horizontal bargraph displayed
n = 5	- Standard screen; two variables and two horizontal bargraphs displayed
n = 6	- Standard screen; single variable and a vertical bargraph displayed
n = 7	- Standard screen; two variables and two vertical bargraphs displayed
n = 8	- Standard screen; three variables and three vertical bargraphs displayed
n = 9	- Standard screen; four variables and four vertical bargraphs displayed
n = 10	- Standard screen; eight variables displayed
n = 11	- Standard screen; eight variables and eight horizontal bargraphs displayed

Initial Value Set by the configuration menu

Modes All Screen Modes

Notes This command function is also found in the configuration menu

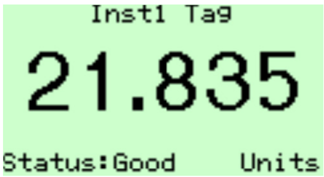
See the "Standard Screens" section on Page 15.for further details.

Uses The <SO> command allows:

- The displayed screen to be changed without having to enter the configuration menus

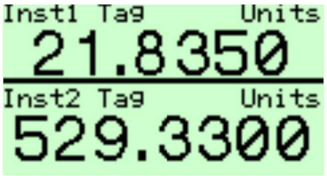
Example

<SO1>



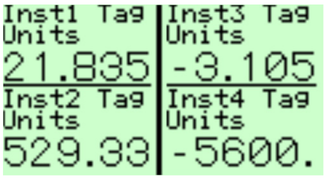
Standard screen, single variable displayed at a time

<SO2>



Standard screen, two variables displayed at a time

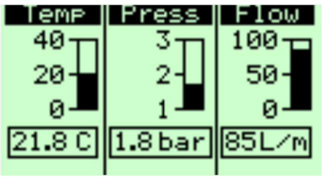
<SO3>



Standard screen, four variables displayed at a time

<SO0>

<SO0> on its own changes to the boot screen.
You have to program the display to your requirements.



Text Display Mode – An example of a custom screen
(See the appendix for the code used to generate this screen)

Gotchas! Changing Screen Option causes the unit to show the boot screen if changing to Text Display Mode

<SS*n*> Screens to Scroll

Description This command specifies how many of the saved frames can be scrolled through when the arrow keys on the display are pressed.

Parameters Values of n allowed and their meanings are:

n = 0	- No screens are available. Arrow keys have no effect
n = 1	- Only one screen accessible (Saved frame 0).
n = 2	- Two screens are available using the arrow keys. (Saved frames 0 and 1)
n = 3	- Three screens are available using the arrow keys. (Saved frames 0, 1 and 2)

Initial Value Default is <SS0>

Modes Pixel and Row Modes only

Notes Screens are restored to the current active frame. This frame is then made visible

Uses The <SO> command allows:

- A custom screen to be made visible by a single keypress
- Unused screens to be hidden

Example This command can be used to "hide" a user logo in the following way:

- Download the logo using the <GB> and <DS> commands
- Save the logo in EEprom area 2
- Set <SS2>
- Set <BS2>

The user logo will be displayed on power-up. The arrow keys restore the saved operational screens. The user logo cannot be accessed once an arrow key has been pressed. This only works in Text Display Mode with custom screens. There is no way to display a user logo in Screen Options 0 to 3

Gotchas! This only works in Text Display Mode with custom screens

See Also **BS** Boot Screen

<ST> Steady

Description Cancel the flashing attribute set with the <FL> command

Parameters None

Initial Value Steady (No Flashing)

Modes All Modes

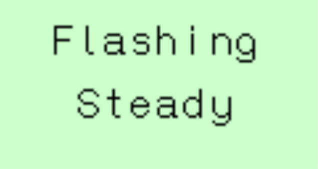
Notes

Uses The <ST> command allows:

- Screens to be built with both flashing and non-flashing text and graphics

Example

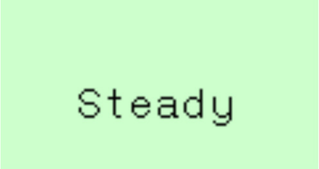
<SD>	Set the display in to a known state
<FL>	Set the flashing attribute
<EF>	Enable flashing
<F2>	Use font 2
<CA>	Align the text in the centre of the screen
<CM2,0>	Down to row 2
<WTFlashing>	Write the word "Flashing"
<ST>	Cancel the flashing attribute
<CM5,0>	Down to row 5
<WTSteady>	Write the word "Steady"



Flashing

Steady

Alternating each second with



Steady

See Also

BM	Background Mode
EF	Enable Flashing
FL	Flashing
IF	Inhibit Flashing

<SV*n*> Show Variable

Description Displays the selected input variable using the current Standard Screen

Parameters *n* = 1 to 8 - Input Variable number

Modes All, when showing Standard Screens

Notes The operator can select which input variable is displayed on the screen by pressing the up and down arrow keys. If necessary, the host can control which input variable is being shown by using this command.

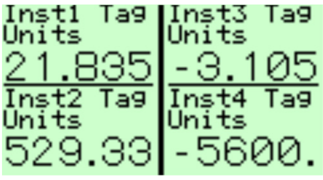
The <SV*n*> command forces the unit to display input variable *n* using the current screen format i.e. if 2 variables per screen are being shown, then issuing a <SV4> command will show input variables 3 and 4

Uses The <SV> command allows:

- An operator to be alerted to a particular value
- A host can display a sequence of input variables

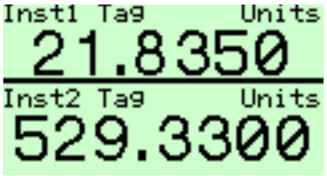
Example

`<SO2>` Start with a 4 variable display



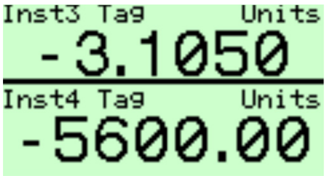
```
Inst1 Tag |Inst3 Tag
Units     |Units
21.835    |-3.105
Inst2 Tag |Inst4 Tag
Units     |Units
529.33    |-5600.
```

`<SO2>` Change to a 2 variable screen



```
Inst1 Tag      Units
  21.8350
Inst2 Tag      Units
 529.3300
```

`<SV3>` Now show input variable 3



```
Inst3 Tag      Units
 -3.1050
Inst4 Tag      Units
 -5600.00
```

The standard screen showing input variable 3 is displayed

See Also **SO** Screen Option

<SW> Smart Wrap

Description Force text that cannot fit on the current line, to be written on the next line without splitting words

Parameters None

Initial Value <NA> No Alignment

Modes Row Mode only

Notes With the <SW> attribute set, the <WT> command will automatically wrap long lines of text without splitting words. It means that the programmer does not have to worry about the formatting as long as the text all fits on the screen. The display will scroll in order to display all the text sent.

Smart Wrap is a text alignment attribute that cannot be used in conjunction with any other alignment command <CA>,<LA>, <RA> or <TW>. It is cancelled by the <NA> command.

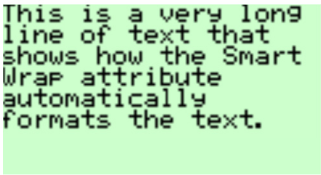
<SW> can be used with either the full screen, or within a window.

Uses The <SW> command allows:

- Simple formatting of text strings

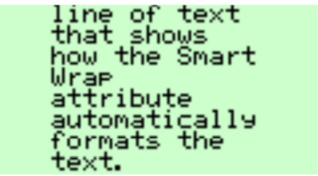
Example

`<SD>`	Set the display to a known state
`<SW>`	Set the Smart Wrap
`<WTThis is a very long line of text that shows how the Smart Wrap attribute automatically formats the text.>`	Write a lot of text. It all fits on screen



```
This is a very long
line of text that
shows how the Smart
Wrap attribute
automatically
formats the text.
```

`<CS>`	Clear the screen
`<DW0,7,20,100>`	Define a window
`<WTThis is a very long line of text that shows how the Smart Wrap attribute automatically formats the text.>`	Send the same line of text, but because of the narrowed window it does not all fit on screen. The display scrolls to accommodate all the text.



```
line of text
that shows
how the Smart
Wrap
attribute
automatically
formats the
text.
```

Note that the display has scrolled

See Also

CA	Centre Align
LA	Left Align
NA	No Align
RA	Right Align
TW	Text Wrap

<TO*n*> TimeOut

Description Activate a timer that warns if communications from the host ceases for (*n* x 10) Seconds

Parameters *n* = 0 to 255 - Multiples of 10 Seconds

Initial Value 0, no timeout active

Modes All Modes

Notes This command activates a timer that warns via a screen message that there has been no communication from the host for a defined period of time.

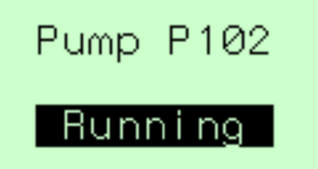
The parameter *n* sets a timeout period of *n* x 10 seconds.
n = 0 deactivates the timeout function.

In order to reset the timer, a valid command with a correct checksum (if used) must be received and acknowledged by the display. In a multidrop application, each individual display must be communicated to within their timeout period.

Uses The <TO> command allows:

- Users to be warned that the message displayed may be out of date

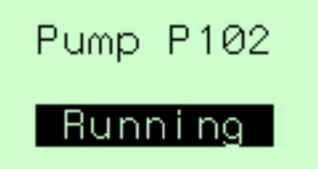
Example `<TO2>` Sets a timeout period of 2 x 10 = 20 seconds
Assume that the following screen was being displayed



Pump P102
Running

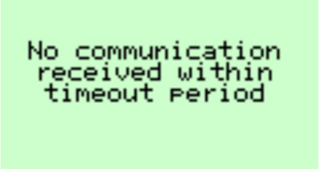
If no communication was received for more than 20 seconds the warning screen will alternate every second with the original screen.

When a communication is received, the warning message will not be displayed again until the timeout period has been exceeded once again.



Pump P102
Running

Alternating each second with



No communication
received within
timeout period

Gotchas! In normal operation, make sure that the host communicates at least once every timeout period. The <RS> Request Status command may be used for this purpose

See Also

RB ReBoot
RS Request Status

<TW> Text Wrap

Description Force text that cannot fit on the current line, to be written on the next line

Parameters None

Initial Value <NA> No Align

Modes Row Mode only

Notes This attributes forces any text that will not fit on the current line to be written on the following line. The operation is not intelligent in any way, the decision of whether to wrap to the next line is made on a character by character basis. This means words will usually flow across two lines.

Text written off the end of the bottom line will cause the screen to scroll.

The Text Wrap attribute may be used with the whole screen or constrained within a window.

It is cancelled by the <NA> No Align command.

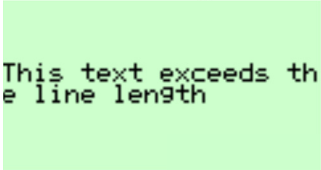
Text that exceeds the line length without either the <TW> or <SW> attributes set will not be written to the screen and an error is returned to the host.

Uses The <TW> command allows:

- Strings can be sent without worrying about their length
- Maximum visible message size, albeit with poor formatting

Example

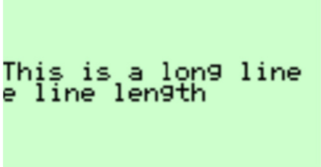
`<SD>`	Set the display to a known state
`<CM3,0>`	Cursor Move to line 3
`<TW>`	Set Text Wrap attribute
`<WTThis text exceeds the line length>`	Send a long line of text, which exceeds the screen width



```
This text exceeds th
e line length
```

Note that all the text is displayed without an error being returned to the host, but the word "the" is split on to two lines.

`<NA>`	Cancel Text Wrap
`<CM3,0>`	Move back to the same starting point
`<WTThis is a long line of text that wraps on to three rows>`	Send another long line of text, which exceeds the screen width



```
This is a long line
e line length
```

This time the first line is overwritten, but the second line is not because the text has not been wrapped.
Also, an error response is returned to the host to indicate that the write command failed

Gotchas! If text needs to wrap, but without splitting words, use the <SW> attribute instead.

See Also

NA	No Align
SW	Smart Wrap

<UE>Upload Enable

Description Enables the use of the Upload Screen <US> command

Parameters None

Initial Value Not enabled. <US> command will return an error unless preceded by <UE>

Modes All Modes

Notes This command enables the use of the Upload Screen <US> command, and must be sent immediately prior to that command.

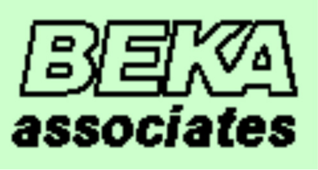
The Upload Screen <US> command is the only command that uploads data from the display, so this enable command is included to prevent accidental use of the <US> command which would disrupt normal communications for a few seconds.

Uses The <UE><US> commands allow:

- Screen contents to be uploaded to a host computer as a Windows format .BMP file. These screen captures can be included in operator user manuals and other documentation.

This combination of commands was used to generate the example screen-shots in this manual.

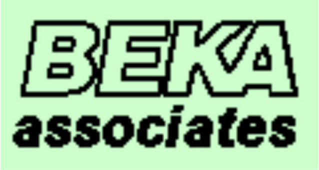
Example



Assume default logo is displayed

<UE>
<US>

Bitmap file of screen image is returned to host



Gotchas! In Operational Modes greater than 0, command responses and checksums will surround the data

See Also **US** Upload Screen

<UL>UnderLine

Description Set the Underline attribute, so that any subsequently written text is underlined.

Parameters None

Initial Value <NU> No Underline

Modes All Modes

Notes Once this attribute has been set, any text written in Fonts 2 to 5 are underlined in the decender area of the font. As Font 1 does not have decenders, this attribute is not recognised. If Font1 text really does need to be underlined, use a line draw command <LH> in pixel mode.

Characters defined in the soft fonts are also underlined using this command. This should be born in mind when defining the characters.

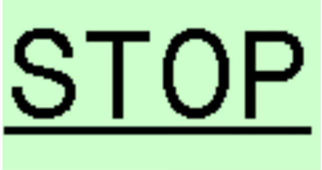
The Underline attribute is cancelled with the <NU> command.

Uses The <UL> command allow:

- Attention to be focussed onto certain text
- Screen presentation to be improved by the use of headings

Example

<SD>	Set the display in to a known state
<F5>	Maximum font size
<CM6,0>	Down to row 6
<CA>	Set centre align attribute
<UL>	Set underline attribute
<WTSTOP>	Write the message



STOP

Gotchas! Font 1 cannot be underlined using this method

See Also **US** Upload Screen

<US>Upload Screen

Description Upload the current screen contents to the host.

Parameters None

Initial Value Not enabled. <US> command will return an error unless preceded by <UE>

Modes All Modes

Notes Detailed information about the upload procedure is in the Graphics Transfer Section (Page 13).

The <US> command is acknowledged in the normal way. After a short gap (500ms), a 1086 byte block of data is sent to the host. A command acknowledge then follows with the check bytes as per the current operational mode. The check bytes include the data block bytes and the acknowledge, but not the check bytes themselves. The 1086 byte data block, saved to file is a graphics image of the screen in 2-colour Windows .BMP format

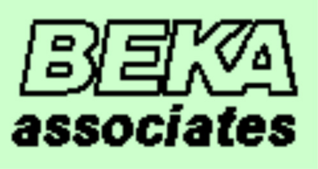
This command requires the Upload Enable <UE> command to be sent immediately prior to it.

Uses The <UE><US> commands allow:

- Screen contents to be uploaded to a host computer as a Windows format .BMP file. These screen captures can be included in operator user manuals and other documentation.

This combination of commands was used to generate the example screen-shots in this manual.

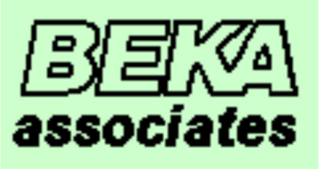
Example



Assume default logo is displayed

```
<UE>
<US>
```

Bitmap file of screen image is returned to host →



Gotchas! In Operational Modes greater than 0, command responses and checksums will surround the data

See Also **UE** Upload Enable

<VB*n*,*m*> Vertical Bargraph

Description Draw a vertical bargraph *n* pixels high with *m* pixels filled

Parameters
n = 3 to 64 - Height of bargraph
m = 0 to *n* - Number of filled pixels, starting from the bottom

Modes Row Mode only
The <WM*n*> Write Mode has no effect on this command

Notes The vertical bargraph is drawn at the current cursor position.

The cursor is restored to its original position after the command.

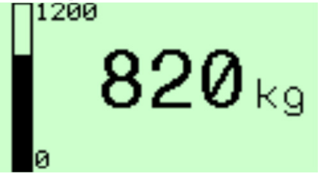
The number of filled pixels has to be less than or equal to the overall length of the bargraph.
Note that the first and last pixels are always filled in to form the frame, so <VB60,0> and <VB60,1> are visually identical, as are <VB60,59> and <VB60,60>

Uses The <VB> command allows:

- Simple graphical representation of values or progress
- Bargraphs to be combined without restriction with other text and graphics

Example

`<SD>`	Set the display in to a known state
`<CM7,5>`	Cursor down to the bottom row, five pixels in.
`<VB64,44>`	Draw a vertical bar 64 pixels long with 44 pixels filled
`<CM7,14>`	Cursor to bottom row 14 pixels in
`<WT0>`	Write a “0” as the lower scale value
`<CM0,14>`	Cursor to top row, 14 pixels in
`<WT1200>`	Write “1200” as the max scale value
`<F4>`	Large font
`<CM5,37>`	Cursor position for variable
`<WT820>`	Write out the value
`<F2>`	Smaller font
`<PM>`	Pixel mode so units label can be precisely positioned
`<CM42,97>`	Position of units label
`<WTkg>`	Write out the units



See Also **HB** Horizontal Bargraph

<VF*n*> Visible Frame

Description Page frame *n* is made visible

Parameters *n* = 0 or 1 - frame number

Initial Value 0

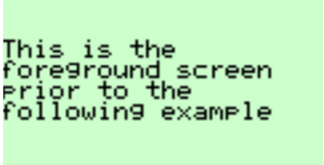
Modes All Modes

Notes The display comprises of two virtual screens, screen 0 and screen 1. Only one of these screens is visible at a time. The <VF*n*> command is issued to make the required screen visible. It is used in conjunction with the <AF*n*> Active Frame command.

Uses The <VF*n*> command allows

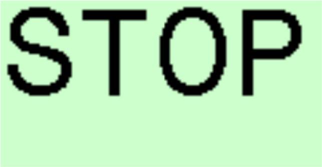
- complex screens to be drawn while hidden and then instantly displayed
- frequently used screens to be instantly restored
- a single command to alternate two images

Example



```
This is the
foreground screen
prior to the
following example
```

`<AF1>`	All writes to the display after this command are directed to screen 1, which is currently hidden
`<CS>`	Screen 1 is cleared, display still shows the initial message
`<F5>`	Large Font enabled, display still shows the initial message
`<WTSTOP>`	The word STOP is written on the hidden screen, display still shows the initial message
`<VF1>`	Screen 1 now made visible. The word STOP appears on the LCD screen



```
STOP
```

Gotchas! Cursor positions are not saved or restored with frames

This command only makes the selected frame visible; it does not change the frame that is written to. Make sure that the Active Frame <AFn> command is issued appropriately

See Also

AF	Active Frame
RF	Restore Frame

<VL*n*> Variable Last

Description This command controls the number of variables shown in standard screens.

Parameters *n* = 1 to 8 - Last variable to be shown

Initial Value 8

Modes All modes

Notes Only applicable when showing standard screens

Uses The <VL*n*> command allows:

- Unused variables to be hidden

Example When using standard screen 1, each press of the up arrow will show the next input value as follows:
If IN_1 is currently being displayed;
Up arrow will change the display to show IN_2
Down arrow will change the display to show IN_8.

If the <VL2> command is sent then the action will be modified as follows:
If IN_1 is currently being displayed;
Up arrow will change the display to show IN_2
Down arrow will change the display to show IN_2.

Gotchas! In multivariable screens, variable entries greater than *n* are just blank

See Also **SO** Screen Option

<WM*n*> Write Mode

Description Determine how text or graphics is drawn on the screen

Parameters *n* = 0 to 3 - mode number

Modes All Modes

Notes The write mode is defined by the value *n*

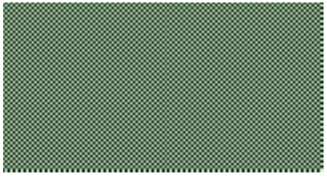
n = 0 data is written normally to the screen, over-writing the current screen contents
n = 1 data being written to the screen is 'ORed' with the current screen contents
n = 2 data being written to the screen is 'XORed' with the current screen contents
n = 3 the inverse of the data is written to the screen, over-writing the current screen contents

Detailed information is in the Display Features Section (Page 5)

Uses The <WM> command allows:

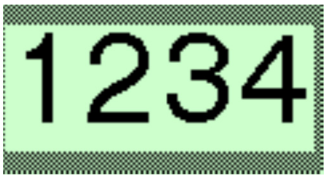
- Complete flexibility over the appearance of text and graphics
- Allows objects to be written that although they may overlap do not overwrite each other
- Inverse can be used to highlight
- XOR writes will undo what has been written

Example **Original screen**

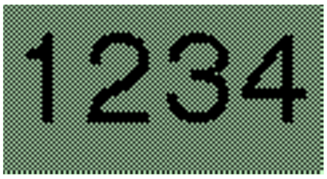


The following examples show the effect of writing the text '1234' in font 5 on a chequer-board background for the 4 write modes:

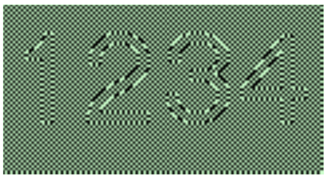
<WM0>



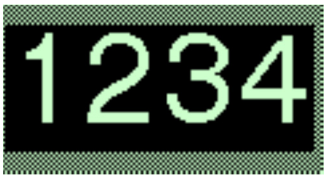
<WM1>



<WM2>



<WM3>



Gotchas! Write modes do not apply to Bargraphs or Restored Frames

See Also **BM** Background Mode

<WS*n*> Write Soft character

Description Write the soft character number *n* of the current font at the current cursor position

Parameters *n* = 0 to 3 - soft font character

Modes All Modes

Notes A soft font is any user defined image that is the same size as the current font.
The display can accommodate 4 soft fonts (*n* = 0 to 3) for each font F1 to F5.

The soft character written assume all the current attributes, just as any normal character.

Although normally used for text characters or symbols that not in the normal character set, the soft characters can be used to store and write any image of the correct size.

This command will assume that the soft font specified has already been downloaded or restored. No error is generated if a soft font does not exist, it just writes uninitialised data.

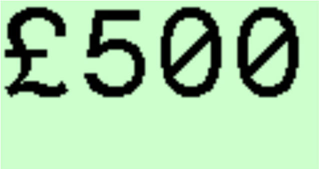
Soft fonts are lost when power is removed from the display. Most fonts can be saved / restored as a block using the <KF> Keep Fonts and <FR> Font Restore commands

Uses The <WS> command allows:

- Any special character to be written to the screen just like any other character

Example

`<CS>`	Clear Screen
`<F5>`	Set largest font size
`<DF0>`	Tell the display that a soft character number 0 (for Font 5) is going to be downloaded
`Binary download of graphics file`	Send a .BMP file of the required soft character to the display. In this case a 48 x 29 pixel image of a GBP symbol (£)
`<WS0>`	Write the soft character to the screen
`<WT500>`	Write normal text


£500

See Also

KF	Keep Font
FR	Font Restore

<WT*string*> Write Text

Description Write text to the display, using any set attributes

Parameters *string* = any 7-bit ASCII string

Initial Value None

Modes All Modes

Notes This command allows text to be written to the display and take advantage of all the attributes and formatting commands.

This command can be used in any mode, but it must be used in Operational Modes 2 to 4 in order to write text to the screen

Free text can be written to the screen in modes 0 and 1, but it is not confirmed and cannot be formatted

If the '>' character is required in a text string with the <WT> command the character should be included twice.

To simplify temperature display, the ` character (alt+096) is mapped to the degrees symbol.

For example, the string **`Temp `C`** is displayed as **`Temp °C`**

Font 1 can display an up-arrow ↑ by using ASCII character 130 (alt+0130), a down-arrow ↓ by using ASCII character 129 (alt+0129) and a block by using ASCII character 127 (alt+0127).

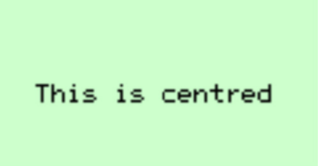
Text that exceeds the line length without either the <TW> or <SW> attributes set will not be written to the screen and an error is returned to the host.

Uses The <WT> command allows

- Text to be written !

Example

`<SD>`	Put the display into a known state
`<CM4,0>`	Cursor to row 4
`<CA>`	Align all following text centrally
`<WTThis is centred>`	Write the message



This is centred

Gotchas! Font 5 has a limited character set

See Also **WS** Write Soft Character

Advanced Commands

There are 8 commands in the standard BEKA protocol that are used to control the scripting functionality. There are a further 18 script commands that can be used within the scripts, and 14 Pattern Matching commands to handle fixed format data strings. These are summarised below and detailed information is given in the following chapters.

Scripting Control Commands

Command	Meaning
<CU>	Cyclic Text Update (Script String Variables)
<CX,n,string>	Cyclic Text
<DPn>	Download Program
<ES>	Execute Script
<JRn>	Jump Register
<KS>	Kill Script
<SEn>	Script Event
<TS>	Terminate Script

Script Commands

Command	Meaning
*BB	Become Busy
*BI	Become Idle
*BR	Become Ready
*DT	Delay Time
*DXn,m	Define Script String
*EDn,m	Event Disable
*EEn,m,lab	Enable Event
*GKn,lab	Go Key
*GRn,lab	Go Register
*GSlab	Go Subroutine
*GTlab	Go To
*LAlab	Label (6 characters max)
*LBlab	Label (6 characters max) and Become Busy
*MKn,m,p	Menu Key
*SR	Subroutine Return
*STn,lab	Script Timeout
*TEn,lab	Timer Event
*WKn,m	Wait Key

Pattern Matching Commands

Command	Meaning
*AVstring	Add to Variable
*IV	Initialise Variable
*MVn	Make Variable
*PCn,m	Pattern Control
*PDn,m,p	Pattern Definition
*PEn,label	Pattern Event
*PJn,m,p,q,label	Pattern Jump
*PKn	Pattern Kill
*PRstring	Pattern Response
*PSn,m	Pattern Size
*PTn,m,label	Pattern Timeout
*PVn,m,p	Pattern Variable
*PWn,m,p	Pattern Write
*VD	Variable Debug

Scripting

A table is presented at the end of this section (page 126) showing the syntax of each command. The following text should be read and understood before attempting to generate any scripts.

Overview

Scripts are a sequence of standard BEKA commands that are downloaded into the display. These commands are then automatically executed by the display without any host intervention. As the download program is stored in EEProm, it remains in the unit even when the power is turned off. There are a number of ways to start and stop scripts, depending on the protocol that is being used with the unit. If a script is present in the unit then it is automatically executed when the power is first applied.

To aid understanding, this section of the manual discusses each command in groups of like functionality rather than alphabetic order.

Downloading and controlling scripts

The previously described BEKA Protocol commands are used to construct the required program flow. To conserve memory space, each command is stripped of its usual < > delimiters and is instead terminated by a *nul* character (ASCII 0). The end of the script is indicated by a second *nul*.

The <DP*n*>Download Program command is used to transmit a script from the host to the display. The maximum size of a script can be 2048 bytes, but instead of one large download it is broken down into 32 64 byte blocks.

The transfer process MUST start with <DP0> and be done in ascending order. The value of n can be from 0 to 31, meaning that a 2048 byte script can be loaded. The end of the script is marked by a *nul* following the command terminator, i.e. two consecutive 0x00s.

e.g. CS*nul*WTHello world*nulnul* would (when downloaded and run) clear the screen and put "Hello world" on the top line of the display.

Some basic checks are carried out on the script as it is downloaded, so it is possible to get a parameter error returned. To help pinpoint the source of the error, some details will be displayed on the screen e.g. "Line too long". Each command delimited with a *nul* is treated as a line for error reporting

The following commands are used to control the running of the script:

Execute Script (<ES>) starts the script running from the first line.
Terminate Script (<TS>) stops the script operation but leaves the script present.
Kill Script (<KS>) removes the script from the unit.

Operation with BEKA protocol

Scripts are supported in when the display is in Text Display mode. It should be noted that the unit must be in Operation Mode 2 and above for scripts to operate (OP Mode 1 would simply display the contents of the script).
When a script is present in the unit a script control menu will appear in the configuration menus accessed by pressing the P+E buttons. This allows the script to be Started, Stopped and Erased by the user.

While the script is running it will reply S0 or B0 to all BEKA commands sent to it apart from a few special ones. These are Terminate Script <TS>, Cyclic Data <CD>, Jump Register <JRn>, Set Event <SEn>, Screen Option <SOn>.
(The B0 indicates that the Script is busy - see the <*BB>, <*BR> & <*LB> commands later.)

To aid program development, we offer a Programming Utility free of charge. The "download" option in this utility deals with the formatting automatically.

Operation with Modbus RTU Protocol

A separate guide is available which deals with the details of this protocol - the "*Serial Text Display – Modbus Interface Guide*". To gain a full understanding of the display, this document should be read in conjunction with this section. **Please note that the display operates as a slave device and cannot take the place of a master.**

A Script has to be downloaded via the Command String and Graphic Data register set using the technique defined in the *Modbus Interface Guide*. The `<ES>, <TS>, <KS>` commands can also be sent using the same register set, however it is normally much more convenient to use the Modbus registers provided for this purpose. For example, there is a Modbus coil that can be used to start and stop a script and a Modbus status register that shows if a script is running or not.

While the script is running it will set the "Result" Modbus input register to 16 or 32 for any BEKA commands sent apart from the few special ones listed above.

Script Flow

A script is a sequence of BEKA commands executed in order. The script engine starts at the top of the list and works its way to the bottom where it stops. This on its own is not very useful and some control over the flow of the script is needed. Also there needs to be a way for the host system and the user to influence the flow. So a special group of two letter commands has been added. In order to make the commands different from the usual BEKA commands they have been prefixed with a *. These commands are for use of the script engine and are meaningless to the normal BEKA command processor.

Script commands are processed in order one after the other. For example;

```
LN
WTHello
```

Will move down a line and write Hello on the screen. After writing the Hello, the script will stop.

```
LN
WTHello
ES
```

Will move down a line, write Hello and then restart at the top.

To develop more complex flows it is necessary to have a "GOTO" statement. It was decided that the old BASIC mechanism of "GOTO line number" was a bit limited so the concept of a label was added to the engine. The **`*GT`*`Label`*** command is therefore used. A label is any string of up to 6 characters but it is **NOT** case sensitive. A label is defined by the **`*LA`*`label`*** or **`*LB`*`label`*** commands. e.g.

```
NS
*LAloop
LN
WTHello
*GTloop
```

This has the same effect as the second example above, except that the screen is cleared the first time the script is run. The script engine allows the definition of 64 labels and because script space is limited it is often most efficient to use 1 or 2 character labels. In the above example the label "loop" appears twice and therefore occupies 8 bytes of the available 2048. By reducing the label to a single "L" 6 bytes can be saved allowing for two extra 2 letter commands to be added.

When processing long scripts it has been found that the ***BB** command was commonly used following a label command. To save script space the ***LB*label*** command has been introduced that performs the same function.

```
i.e.    *LAlabel          *LBlabel
        *BB         =
        (13 bytes)        (9 bytes)
```

In programming terms we often need to execute the same sequence of commands several times during the operation of a program so we have the Goto Subroutine **`*GS`*`label`*** and Subroutine Return **`*SR`**. The number of subroutines is limited by the number of labels but it should be noted that the script engine does not support nesting of subroutines. Such constructs should be avoided in simplistic systems. It should be noted that a subroutine has an overhead of 9 bytes before any commands are added (assuming a single character label) and so needs to be used carefully.

To demonstrate the use of a subroutine, our simple example could be extended to:

```
NS
*LAloop
*GSsub
*GTloop
*LAsub
LN
WTHello
*SR
```

The script first clears the screen and then runs round the loop but each time round the loop it heads off to the subroutine "sub" and executes the **`LN`** Line New and **`WT`** Write Text commands.

Key Handling

Most users will want to have some control over the data being presented to them and so we need to be able to go to a label based on a key press. This is achieved by using the **`*GK`*`n,label`*** Goto Key command, where ***n*** is the number of the key to be acted upon i.e **`*GK1,one`** would check to see if Key 1 is pressed and if it was it would go to the label "one". By using a value of zero for ***n*** then the engine will go to the label if **ANY** of the keys are pressed.

```
NS
WTPress Key 1 or 2
*LAloop
*GK1,one
*GK2,two
*GTloop
*LAtwo
CM7,0
WTScreen 2
EL
*GTloop
*LAone
CM7,0
WTScreen 1
EL
*GTloop
```

In this example the script clears the screen and writes up the message instructing the user to press key 1 or 2 it then drops into a loop which checks the keys. If key 1 is pressed the flow is altered to go to label one. This causes the script engine to write "Screen 1" on the bottom line of the screen and clears the rest of the line. It then reverts to the scanning of the keys. The same applies to key 2.

The display has a two local menus that are accessed by pressing a combination of keys. The main menu is used to configure the display, and is accessed by pressing the " **P** " and " **E** " keys simultaneously. The Quick Access menu is used to provide a local operator a few facilities that may be needed, without giving the ability to mis-configure the display. The Quick Access menu is reached by pressing the " **P** " and " ↑ " keys simultaneously. There is a slight possibility that these key combinations may interfere with a script that has been written, so the facility has been added to redefine the key that accesses the menus. The **`*MK`*`n,m,p`*** Menu Key command is used for this purpose, where ***n*** is the number of the key (1 – 6) that should be pressed to enter the menu, ***m*** is the type of menu to be redefined (0 = main menu and 1 = Quick Access menu), and ***p*** is set to 0 to allow the new key to be used in addition to the default key combination, or ***p*** is set to 1 to inhibit default key combination. The command may be issued twice to redefine both menu keys, and the default key combinations can be restored by setting ***n*** to a value of 0

As an example, the command **`*MK3,1,1`** will disable the " **P** " and " ↑ " key combination to the Quick Access menu, but will enter this menu whenever button 3 is pressed.

Jump Register

The text display has the ability to store screens in memory, so a mechanism was added for the host to easily recall a stored screen e.g. for use as an alarm message. In order to do this within a script a register has been introduced which can be set from 1 to 255 by the host system. The **<JR*n*>** Jump Register BEKA command is used to set the value of the register to n. The register will remain set to the value until either the host changes it or the script clears it by execution the `*GRn` Goto Register command where ***n*** matches the value of the register.

```
NS
WTPress Key 1 or 2
*LAloop
*GK1,one
*GK2,two
*GR100,error1
*GR200,error2
*GTloop
*LAtwo
CM7,0
WTScreen 2
EL
*GTloop
*LAone
CM7,0
WTScreen 1
EL
*GTloop
*LAerror1
CM7,0
WTHost Alarm 1
EL
*GTloop
*LAerror2
CM7,0
WTHost Alarm 2
EL
*Gtloop
```

Here the operation is the same as for the Goto Key example, but has been extended to allow the host to put up an alarm message. If the host issues a **<JR100>** command then the message “Host Alarm 1” will appear on the display and the Jump Register will be reset to 0. Similarly, if the host were to issue a **<JR200>** command then “Host Alarm 2 “ will appear on the display and the Jump Register will be reset to 0.

Note that setting the jump register to any other number will not have any effect and the register will remain set to that number until the host changes it.

Delays and Waits

In the real world humans need time to input data and make decisions on it, thus it is often necessary for the display device to wait for the operator to catch up. Delay Times are set with the `*DTn` command where ***n*** is the time to wait in 100ms intervals.

```
OE1
*DT5
OD1
```

Would turn on the output 1 of the text display for 500ms.

In the same way we might want to wait for an operator to press a key but get back to the main program flow if they were away from the process. So we have the `*WKn,t` Wait for Key command, where ***n*** is the key required (or 0 for any key) and ***t*** is the time that the system will wait before moving on in 100ms intervals.

```
*LAStart
NS
F1
WTPipe Flow
F3
CM3,0
DV1,5,0,1,1
F1
*LAloop
*GR100,tb
*GTloop
*LAtb
NS
F1
CA
WTTea Break
CM7,0
WTPress 1 To Cancel
NA
*WK1,100
*GTStart
```

In this example the text display will show the cyclic data value for variable 1 on the screen for the user to monitor the process. At tea break time the host sets the jump register to 100, the script engine sees this and changes the screen to display "Tea Break" and asks the user to press key 1 to cancel the message. If the operator is present and presses 1 then the process monitor screen is returned. However if there is no-one there to press the key, then script engine will restore the process view after 10 seconds.

Events

In the examples presented so far, everything has been sequential and things only happen when the script engine executes a line of code. The script engine can not act on a key press until the script gets to a point where it checks for any presses with a ***GK** type of command. While this works in many simple applications it is often not the preferred method of operation.

A mechanism to interrupt the program flow when a particular event has occurred has been incorporated. An Event is an external influence, which redirects the operation of the script engine immediately. There are two external influences that can trigger an Event; a key press and the host setting the Event Register.

The Event Register can support a maximum of 20 events. Like the Jump Register, the Event Register is set with the **<SE*n*>** Set Event command where ***n*** is in the range 1 to 20. Once the event has been seen by the script engine, the event register is cleared to 0. Events are enabled with the **`*EEn,m,label`** command, where ***n*** = 0 to respond to a key press event, and ***n*** = 1 to respond to the Event Register; ***m*** = the number of the key or register and ***label*** is where the script engine is to jump to.

Once the event has occurred and the script engine has finished executing the instructions ,it needs to do nothing until the next event occurs. This can simply be achieved by using a label and a goto, but this keeps the script engine busy and occupies valuable script memory. It is possible that the text display may be dealing with cyclic data while doing all of this which may affect the perceived responsiveness of the instrument. To lessen the load on the processor, there is an idle state for the script engine: ***BI** Become Idle tells the script engine to nothing except watch for an external event.

Script Event Update (Firmware 3.4 Onwards)

One problem with the <SE*n*> command is that the event is always executed when the command is sent. There are applications on some small PLC's where it would be helpful if the event was only executed when the value of *n* was updated to a new value.

This has been implemented by defining a new set of values for *n*. If *n* is in the range 21 to 40 then event (*n*-20) will be executed once and then not again until the event number is changed, regardless of the number of times the command is executed.

For example:

SE1	*Event 1 executed*
SE1	*Event 1 executed again*
SE1	*Event 1 executed again*
SE21	*Nothing happens as Event 1 has already been executed*
SE22	*Event 2 is executed*
SE22	*Nothing happens as Event 2 has already been executed*
SE22	*Nothing happens as Event 2 has already been executed*
SE21	*Event 1 is executed*

Finally, having created an event, it might have to be disabled it to prevent it happening too many times. In this case the `*EDn,m` Event Disable command can be used, where the parameters ***n*** and ***m*** correspond to the respective `*EEn,m,label` command. However, if ***m*** is set to zero then all events of type ***n*** are disabled.

```
*EE0,2,key2
*EE1,10,evnt10
*EE1,20,evnt20
SD
WTEvent Demo
*BI
*LAevnt20
CM7,0
WTEvent 20 Occurred
EL
*DT10
CL7
*BI
*LAEvnt10
CM7,0
WTEvent 10 Occurred
EL
*DT5
CL7
*BI
*LAkey2
CM7,0
WTKey 2 Pressed
EL
*DT5
CL7
*BI
```

This example clears the screen, puts up the title, defines the events and then sits in the idle mode waiting for an event to occur. When key 2 is pressed the script engine bursts into life and jumps to the label “key2”, where the “Key 2 Pressed” message is shown on the screen for 500ms. The screen is then cleared again before going back to the idle state. Similarly, the host issuing a **<SE10>** command will cause the message “Event 10 Occurred” to appear on the screen for 500ms. As an exercise it is left to the reader to work out what happens when the host issues a **<SE20>** command!

The use of Events allows the programmer to create different process views of the plant and display them depending on what the user needs to see at that time. It is also possible for the host to interrupt normal operation to show alarm messages etc. However, there is a potential problem while the script is defining the events; In the previous example, assume the script engine has executed the first ***EE** line and moved on to the next line. That first event is now live and will be executed if that event occurs. This means that there is a danger that the host or user could trigger an event before all the required events are enabled. To illustrate this further, assume the script engine has just executed line 2 of the above script when the user presses key 2. The message “Key 2 Pressed” appears on the screen and the script engine becomes idle. But now the event which should be triggered if the event register is set to 20 has not been enabled, so part of the program has not been initialised.

To prevent this happening the ***BB** Become Busy and the ***BR** Become Ready commands have been added. When a ***BB** is executed the text display will return B0 to all commands except **<TS>** Terminate Script. This can be used to tell the host that the display is busy setting itself up. When the initialisation section of script is completed the ***BR** command will put the script engine back to normal operation.

```
*BB
*EE0,2,key2
*EE1,10,evnt10
*EE1,20,evnt20
*BR
SD
WTEvent Demo
*BI
*LAevnt20
cm7,0
WTEvent 20 Occurred
EL
*DT10
CL7
*BI
*LAEvnt10
cm7,0
WTEvent 10 Occurred
EL
*DT5
CL7
*BI
*LAkey2
cm7,0
WTKey 2 Pressed
EL
*DT5
CL7
*BI
```

This example is the same as the previous one but the set up of the three events is protected.

Timer Event (Firmware 3.4 Onwards)

A new command to create a timer event after a specified number of seconds has been added. ***TEn,*label*** generates an event after *n* seconds has elapsed. It is a one shot event that can be cancelled by specifying a zero delay to the same label i.e. ***TE0,*label***

This example script will switch output1 on and off at 5 second intervals

```
SD              Set the display in to a known state
*TE5,on         Enable a Timer Event to happen after 5 seconds
*BI             Become Idle
*LBon
OE1             Set output 1 on
*TE5,off        Enable a Timer Event to happen after 5 seconds
*BI             Become Idle
*LBoff
OD1             Set output 1 off
*TE5,on         Enable a Timer Event to happen after 5 seconds
*BI             Become Idle
```

Script Errors

Errors encountered when running a script are displayed on the screen indicating the line number that the error occurred. Any error also stops the script engine from executing any further commands. If necessary, the host can read the error message with the **<GE>** command and the error line with the **<GL>** command.

Script Timeout (Firmware 3.4 Onwards)

The **<TO*n*>** command can be used in standard applications to show that the host has stopped sending data to the text display. A new script command ***ST*n,label*** has been added to perform this function within a running script.

As an example, the following script displays the value of input IN_1 on the screen while the host is sending data. 5 Seconds after the host stops sending data, the message 'Comms Lost, Press E to Clear' will appear at the bottom of the screen.

Once the HOST has started sending data again, Pressing E will clear the message and re-enable the timer.

```
*BB                      Become Busy
*EE0,6,cmsg              Set a Keyboard event to go to 'cmsg' when key 6 is pressed
*ST5,tout                Set a Script Time out to go to 'tout' if no communications for 5
                         seconds or more
SD                       Set Display to a known format
NS                       Clear all Mapped Variables
CM0,1
WTScript Timeout Test    Display a Message
F2                       Set Font 2
CM3,30
DV1,5,2,1,1              Display Mapped Variable IN_1
*BI                      Become Idle

*LBtout                  Define Timeout Event
F2                       Font 2
CM6,0                    Move Cursor to line 6
CA                       Set Centre Align
WM3                      Set Inverse Text
WTComms Lost             Display Message
WM0                      Clear Inverse Attribute
F1                       Set Font 1
CM7,0                    Move to Line 7
WTPress E to Clear>      Display Message
*BI                      Become Idle
*LBcmsg                  Clear Event
CL5                      Clear Line 5
CL6                      Clear Line 6
CL7                      Clear Line 7
*ST5,tout                Set a Script Time out to go to 'to' if no communications for 5
                         seconds or more
*BI                      Become Idle
```

Script Strings (Firmware 3.4 Onwards)

The text display can store 16 text strings of up to 16 characters in length. These can be placed anywhere on the screen using the ***DX*n,l*** command. Where *n* is the index of the string to display and *l* is the length of the field 1 to 16.

If the contents of the stored string is greater than the length parameter, then the display will only show the characters of the string up to the length, i.e. the string will be truncated.

On power up the strings are all cleared to blank strings. They can populated with the **<CX*n*,*string*>** command where ***n*** is the index of the string to be saved and ***string*** is the character string to save. This command does not have any visual effect on the screen display, but when all the strings that require to be updated have been populated with their new values, the **<CU>** command can be sent. This will cause all the strings being displayed on the screen to be updated simultaneously.

Using this command simplifies such tasks as showing the current time on the screen, or for a simple script to display a number of variables, together with their tags and units. Once the script has been run the host can use the **<CV*n,string*>** and **<CX*n,string*>** followed by **<CU>** to display as many values as needed in a common format.

If should be noted that unlike Mapped Variables and Bars, the location of script strings are not stored in any saved frames. Thus the use of a save frame and then restore frame will only restore the image of the string displayed at the time that the save frame was executed.

In the following example the time, 2 variables with their units and tags are displayed on then screen when the script has been downloaded and run.

```
SD
NS
CM0,90
*DX1,5
CM2,57
DV1,5,1,1,1
CM4,57
DV2,5,1,1,1
CM2,6
*DX2,8
CM4,6
*DX3,8
CM2,91
*DX4,4
CM4,91
*DX5,4
*BI
```

The host can now send the following commands to display the data associated with Tag_0001 and Tag_0002:

```
<CX1,00:00>            The 00:00 will be replaced by the system time
<CX2,Tag_0001>
<CX3,Tag_0002>
<CX4,PSI>
<CX5,Bar>
<CU>
<CV1,100.0>            The 100.0 would be the actual value of Tag_0001
<CV2,150.0>            The 150.0 would be the actual value of Tag_0002
```

By changing a few host data parameters, the information can be easily changed:

```
<CX1,00:00>            The 00:00 will be replaced by the system time
<CX2,Tag_0025>
<CX3,Tag_0027>
<CX4,`C>
<CX5,DegF>
<CU>
<CV1,23.4>             The 23.4 would be the actual value of Tag_0025
<CV2,3.0>              The 3.0 would be the actual value of Tag_0027
```

Therefore one script can be used to display lots of different data in the same format under the control of the host.

Scripting Control Commands

Command	Meaning	Description
<CU>	Cyclic Text Update	Causes all script string variables defined by *DX to be updated
<CX*n*,*string*>	Cyclic Text	Populates one of the 16 cyclic text strings *n* is the index number in the range 1 to 16 The length of each string can be up to 16 characters.
<DP*n*>	Download Program	Downloads a script into the display *n* is a block number in the range 0 to 31 containing exactly 64 bytes of downloaded data. See the section on Downloading Scripts (page 117)
<ES>	Execute Script	Starts a downloaded script running
<GE>	Get Error	The <GE> command is confirmed and then the last script error is returned as an 18 byte zero padded ASCII string. This data is followed by the standard response sequence for the current operational mode
<GL>	Get Line	The <GL> command is confirmed and then the last script error line number is returned as an 18 byte zero padded ASCII string. This data is followed by the Standard response sequence for the current operational mode
<JR*n*>	Jump Register	Sets the Jump Register to *n* *n* is a number in the range 1 to 255
<KS>	Kill Script	Clears a script from memory
<SE*n*>	Script Event	Load the Script Event register with the value *n*. *n* is a number in the range 1 to 20
<TS>	Terminate Script	Stops a downloaded script running

Please note that the above commands are not detailed any further in the command reference section of this manual. Their usage is fully discussed in the preceding Scripting section starting on page 117.

Script Commands

Command	Meaning	Description
*BB	Become Busy	BEKA commands return a "B" instead of 'K' 'E' etc. This is used in conjunction with the *BR command to warn the host that a script is actioning a particular set of commands e.g. Setting up a series of events.
*BI	Become Idle	Script processing is paused, but event handling continues
*BR	Become Ready	Cancels the *BB command. Normal command responses are restored.
*DT*n*	Delay Time	Waits for (*n* x 100ms). *n* is a number in the range 1 to 255
*DX*n,m*	Define Script String	The text display can store 16 text strings of up to 8 characters in length. *n* = the index of the string to display where *n* is in the range 1 to 16. *m* = the length of the field where *m* is in the range 1 to 8.
*ED*n*,*m*	Event Disable	Disables the defined event *n* = 0 to define key press events where *m* is in the range 0 to 6 *n* = 1 to define register events where *m* is in the range 0 to 20 *Note:* if *m* = 0 then all events in that class are disabled
*EE*n*,*m*,*label*	Enable Event	Goes to the label *label* on the defined event *n* = 0 to define key press events where *m* is in the range 1 to 6 *n* = 1 to define register events where *m* is in the range 1 to 20
*GK*n*,*label*	Goto Key	Goes to the label *label* if the key *n* has been pressed, otherwise it continues on to the next command in sequence *n* is in the range 0 to 6, 0 being any key
*GR*n*,*label*	Goto Register	Goes to the label *label* when the register = *n* *n* is in the range 1 to 255 *n* can be set externally by BEKA or Modbus commands to control the script flow
*GS*label*	Goto Subroutine	Executes the subroutine at the specified label *label* • The subroutine must end with a *SR command • Note that nested subroutines not allowed
*GT*label*	Go To	Goes to the label *label*(6 chars max)
*LA*label*	Label	Defines the label *label*(6 chars max)
*LB*label*	Label	Defines the label *label*(6 chars max) and then Becomes Busy (A space saving way of issuing a *LA*label* followed by *BB)

*MK*n*,*m*,*p*	Menu Key	Allows a specified key *n* to enter the display menu system *n* is in the range 0 to 6, where 0 disables the command *m* = 0 for main menu, 1 for Quick Access menu *p* = 0 to allow the default key combination *p* = 1 to inhibit the default key combination
*SR	Subroutine Return	Returns control to the calling script
*ST*n,label*	Script Timeout	Generates an event when there has been no communications with the text display for *n* seconds. It is a one shot event. The event can cancelled with *ST0,*label*
*TE*n,label*	Timer Event	Generates an event after *n* seconds has elapsed. It is a one shot event. The event can cancelled with *TE0,*label*
*WK*n*,*t*	Wait Key	Waits for key *n* to be pressed or a time of (*m* x 100ms) *n* is in the range 0 to 6, 0 being any key *t* is in the range 1 to 255

Please note that the above commands are not detailed any further in the command reference section of this manual. Their usage is fully discussed in the preceding Scripting section starting on page 117.

Pattern Matching

A table is presented at the end of this section (page 136) showing the syntax of each command. The following text should be read and understood before attempting to use any pattern matching within scripts.

Overview

Data from small weighing systems, barcode readers and other simple instruments are often output as a printable ASCII string. Commonly it takes a very simple form of a value, followed by the units, then a carriage return/line feed sequence e.g. 12.56g[cr][lf]. This type of output is often designed to be fed straight to a printer or remote proprietary equipment. However, some systems make the data more secure by adding a known start byte and a terminator round the ASCII data e.g. [stx]12.34kg[etx].

As there is always a need for local indication of process variables, a sub-set of the scripting commands have been incorporated to allow a user to capture data of this nature and display the resultant value on the text display screen either directly or using mapped variables and bar graphs.

Pattern matching falls into two distinct parts. Firstly the data has to be captured, and secondly it has to be processed into an appropriate form. In order to explain both capture and processing we will make use of a simple data format from a voltmeter which is:-

[STX], control (1 byte), value(6 bytes including the decimal point), [ETX]

Control	0x00 = No Units
	0x01 = kilo Volts
	0x02 = Volts
	0x04 = millivolts

if the control byte has 0x80 added to it the number is negative

Thus	[STX][0x04]55.678[ETX] will represent 55.678mV
	[STX][0x01]55.678[ETX] will represent 55.678kV
	[STX][0x82]55.678[ETX] will represent -55.678V

Pattern Capture

In order to capture data it has to have a defined structure. The structure is often fixed by the manufacturer but can be configured in some cases. In our example above, the pattern is 9 bytes in length including the two fixed framing bytes, one at the start and one at the end. The 7 bytes in the middle can change depending on the range and value measured by the system.

The BEKA display can be set up to handle 4 separate patterns at the same time or to use one pattern to control and redefine other patterns. For the purpose of this document we will only work with one pattern.

Although the pattern definition commands may be used in any order an error will result if you try to enable a pattern before having set up the key matching elements.

The **`*PS`*`n`*`,`*`m`*** Pattern Size command defines the length of the data to be captured. . ***n*** is the pattern number which must be in the range of 1 to 4, and ***m*** is the length which must be in the range of 1 to 32. Thus in the case of the pattern above **`*PS1,9`** would define the size of pattern 1 to be 9 bytes.

Once the size of the pattern has been defined we need to tell the Script engine to jump to a decoding routine when it gets a match. This is done using Events which work in the same way as the keypress event described in the scripting section. The **`*PE`*`n,label`*** Pattern Event command is used for this purpose, which causes the script to execute commands starting at label *label*. In just two simple commands we have configured the display to capture 9 bytes of data and go and process it.

In order to start the whole capture process off, the system has to be enabled by using the **`*PC`*`n`*`,`*`m`*** Pattern Control command where ***n*** is the pattern number, ***m*** = 1 enables pattern matching and m=0 disables matching. Note that ***n*** may take the value of 0 which will control all 4 patterns simultaneously.

Switching the matching facility on will disable the current Modbus or BEKA communication protocol and hand serial data handling over to the pattern matching system. The script can be stopped by a menu selection or in some cases it can be useful to have a ‘Escape Key’ defined. Once the matching has been started, the script engine will interrupt whatever it is doing as soon as a match occurs, so the **`*BB`** Become Busy and **`*BR`** Become Ready can be used to prevent new matches from interfering with the flow of the script.

While the script is busy, any data sent from the host to the display will be ignored. However, once the display has been set ready again, it will then latch on to the next complete match.

So the following simple script shows how to capture 9 bytes of data and enable the matching.

```
*BB
*PS1,9
*PE1,gotit
*PC1,1
*BI
*LAgotit
*BB
data processing goes here which will be discussed later.
*BI
```

While this script will work, it will make a match out of any 9 bytes that arrive on the serial interface even if they are not part of the sequence sent from the host but due to noise on a long wire. In order to harden up the match we need to tell the pattern matching system that there are specific bytes to look for, as well as the specified number of bytes. This is done with the **`*PD`*n*`,`*m*`,`*p*** Pattern Define command. In this command ***n*** is again the pattern number (1 – 4), ***m*** informs the system which byte of the message to look at and ***p*** tells the system what that byte should be. So for our example **`*PD1,1,2`** will tell the system that the first byte of pattern 1 must have the value of 0x02 ([STX]) for a match to be made. **`*PD1,9,3`** would perform a similar action and ensure that the 9th byte is always 0x03 (ETX).

In the example we cannot specify any other bytes as they might change. So now our script looks like:-

```
*BB
*PS1,9
*PE1,gotit
*PD1,1,2
*PD1,9,3
*PC1,1
*BI
*LAgotit
*BB
data processing goes here which will be discussed later.
*BI
```

Once this is started off, the script engine will stop what it is doing when 9 bytes of data arrive with the first byte as a [STX] character and the last byte as a [ETX] character. Anything else will be ignored by the matching process. The 9 bytes of data is then held for the use of the data processing section until a new match is received.

Once defined, patterns are retained while a script is running. The moment a script is stopped the definition is lost and matching is switched off. This is to allow normal host communication to resume via BEKA protocol or Modbus. If a pattern needs to be removed during a running script, the **`*PK`*n*** Pattern Kill command can be used. This command switches off the matching for the pattern ***n***, removes all **`*PD`** definitions for that pattern and removes the Event. Once a pattern has been killed, it is necessary to redefine the structure from scratch as discussed above.

Pattern Processing

Now that the data has been captured it has to be processed ready for display. The purpose of the Serial Text Display is to supply a user with information in a manner which makes it easy for them to do their job. This discussion will concentrate on how to get the data from the matched string and display it in the simplest manner. More complex and visually pleasing displays can be created as the user wishes.

Processing the captured data falls in to three basic areas; Firstly there is the **direct acti**on of taking an ASCII byte from the matched string and writing it to the current cursor position. Secondly there is **indirect action**, where the bytes from the match string are fed through to the routine cyclic data section of the text display, allowing the use of bar graphs. Finally there is the **decision making** part of the process in which the data in one or more bytes directs the flow of the script.

Direct Action Commands

The direct action command is `*PWn,m,p` Pattern Write. It takes bytes from the matched data and writes them at the current cursor position on the display using any attributes that have been previously set. The parameter ***n*** defines the pattern number to use (1-4), ***m*** indicates the position of the first byte to be extracted and ***p*** tells the system how many characters to write. In our voltmeter example a `*PW1,3,6` would write the value transmitted from the host straight on the screen.

So we can now develop our simple script to this:

```
*BB
*PS1,9
*PE1,gotit
*PD1,1,2
*PD1,9,3
*PC1,1
*BI
*LAgotit
*BB
F2
CM2,0
*PW1,3,6
*BI
```

This script sets up the pattern to match the data as before. When a match arrives it will write the value received on to the screen of the display in font size 2. The cursor move **CM2,0** ensures that the latest value is written over the old value.

Indirect Action Commands

The indirect action commands are a little more complex. These commands work on a stored variable rather than writing directly to the screen. The concepts behind cyclic data and mapped variables should be understood before attempting to use this type of action. There are commands to add data to the store and then to convert the store into cyclic data. When a script is first executed the store is empty, and once it has been converted to cyclic data, it is cleared. However there may be times when the store needs to be cleared under script control. The Initialise Variable `*IV` command performs this function.

To add data to the store the `*PVn,m,p` Pattern Variable command is used in the same way as the `*PW` command, except that the data is added to the store rather than being written directly to the screen. The store is relatively small and can only handle 10 bytes but this is adequate for the purpose. The `*PV` command will only handle numerical data:
i.e. 0 to 9 and the characters “E + - .” Trying to use any other characters will cause an error. In the default state the error is hidden from the user as the `*PV` command will just ignore the whole sequence. Thus the on screen data will not be updated until the `*PV` command gets supplied with valid data again. During programme development the `*VD` Variable Debug command can be used to force the display to report an error to the user, stopping the script in the process. Once the `*VD` command has been issued, the debug mode is enabled until the script is stopped and restarted without the command.

The next stage is to convert the stored value into cyclic data so that the cyclic data mechanism of the display can update the mapped variables and bar graphs automatically. The `*MVn` Make Variable command is used to perform this task. The parameter ***n*** is the target variable and must be in the range of 1 to 8, corresponding to the cyclic data values IN_1 to

IN_8. It is important to note that ***MV** will only convert real numbers in a numeric format that it understands e.g. 1.234, 12E-2, -1000.7 etc. It cannot cope with non numbers and will just give an error. As with the ***PV** command above, the error causes the ***MV** command to ignore the data and not update the display. Once again the ***VD** command can be used to have the error reported for debug purposes.

In our previous example program, the value from the voltmeter was written directly to the screen and was not connected to cyclic data. The following example sets up a mapped variable and a bar graph and then puts the data received from the voltmeter into this mapped variable. The displayed value and bargraph are automatically updated each time new data is received, without having to redraw the screen in any way.

```
*BB
NS
F2
CM2,0
DV1,6,1,0,1
CM3,0
DB1,120,0,0,0
DL1,0,100
*PS1,9
*PE1,gotit
*PD1,1,2
*PD1,9,3
*PC1,1
*BI
*LAgotit
*BB
*PV1,3,6
*MV1
*BI
```

The **DV** command sets up the variable IN_1 on the screen with a field width of 6 and 1 decimal place and the **DB** command defines a bar below the variable (This is far from pretty but it demonstrates the basic steps!). Finally the **DL** command defines the upper and lower limits of the bargraph to be 0 and 100.

Some data streams include flags that set the position of the decimal point or indicate the sign of the measured value so it may be necessary to add ASCII characters to the store before it is processed with the ***MV** command. The ***AVstring** Add Variable command does exactly this. ***AV** combined with ***PV** allows the store to be set as needed. E.g.

```
*IV
*AV-
*PV1,3,6
*MV1
```

The above code initialises the store, loads a minus sign into it and then adds the value from the matched data before sending it to cyclic data IN_1. This it makes it appear as negative number. Note that any number of ***AV** commands can be used before or after each ***PV** command.

In order to decide if the value is positive or negative we need to be able to make a decision and influence the flow of the script. For our voltmeter example we need to determine if the MSB of the control byte set or not. If it is we need to add the minus sign before adding the variable with the ***PV** command.

Decision Making Commands

The ***PJ*n*,*m*,*p*,*q*,*label*** Pattern Jump command allows the user to do this decision making. As before ***n*** is the pattern number, ***m*** is the byte in the matched stream to be examined, ***p*** is a mask value, which is ANDed to the data before the test is made, allowing bits that are not important to be hidden during the test, and ***q*** is the value to be expected. If the conditions of the test are true, then script flow is redirected to the label. If the test fails, the script carries on from the next line.

This sounds complicated but an example will demonstrate how straightforward the process is. In our voltmeter example, the control byte consists of two parts. The lower three bits indicate the units and the MSB indicates the sign. If we wanted to just look at the sign bit, ignoring the 3 unit bytes, we would set the mask to 0x80 (or 128 in decimal).

Control byte value	0x01	0x04	0x82
Mask value	0x80	0x80	0x80
Result after ANDing	0x00	0x00	0x80
Expected Value	0x00	0x00	0x00
Result = Expected ?(i.e. Jump)	YES	YES	NO

To use this in our example code we would test the second byte:

```
*PJ,1,2,128,0,pos
*AV-
*LApos
*PV1,3,6
*MV1
```

which will add a minus sign when the top bit is set.

To make the life of the user easier, they also need to know what scale the displayed value is. The units are defined in the bottom three bits of the same control byte. Using just three jumps we can test for these and apply the units to the display. This time the mask value is changed to ignore the sign bit:

Control byte value	0x01	0x04	0x82
Mask value	0x07	0x07	0x07
Result after ANDing	0x01	0x04	0x02
Expected Value	0x01	0x01	0x01
Result = Expected ?(i.e. Jump)	YES	NO	NO

This converts into code as

```
F1
CM2,61
*PJ1,2,7,1,addkv
*PJ1,2,7,2,addv
*PJ1,2,7,4,addmv
*GTdone
*LAaddkv
WTkV
*GTdone
*LAaddv
WTV
*GTdone
*LAaddmv
WTmV
*LAdone
EL
```

The cursor is lined up, the decision process is made, the units written to the screen and then the rest of the line is cleared so that a change of unit from mV to V does not end up with a unit of VV.

Now we have a completed script that will display the received value on screen and as a bar graph between 1 and 100 units. The user is also shown the units. The full script is shown below.

```
*BB
SD
NS
F2
CM2,0
DV1,6,1,0,1
CM3,0
DB1,120,0,0,0
DL1,0,100
*PS1,9
*PE1,gotit
*PD1,1,2
```

continued -->

```
*PD1,9,3
*PC1,1
*BI
*LAgotit
*BB
*PJ1,2,128,0,pos
*AV-
*Lapos
*PV1,3,6
*MV1
F1
CM2,61
*PJ1,2,7,1,addkv
*PJ1,2,7,2,addv
*PJ1,2,7,4,addmv
*GTdone
*Laaddkv
WTkV
*Gtdone
*Laaddv
WTV
*Gtdone
*Laaddmv
WTmV
*LADone
EL
*BI
```

This script will work correctly as long as the host keeps sending data, but if it stops the value on the display will freeze at the last value received. In many applications this will not be acceptable as it could mislead an operator. To protect against this the **`*PT`*`n,t,label`*** Pattern Timeout command should be used. As before ***n*** is the pattern number, ***t*** is the timeout in seconds and ***label*** is the place to go when a timeout occurs.

Once the **`*PT`** command has been issued and the associated pattern enabled, the event will fire after the timeout period if no pattern is matched. Once the pattern is correctly matched, the timeout timer is reset and it starts to count down again. Therefore if the pattern is always sent repeatedly before the timeout time, then the timeout event will not trigger. However if the host does not send the next pattern in time, the timeout event will occur and the operator can be warned. Once the event has happened, it will not happen again until it another valid pattern has been received or the pattern is disabled and restarted with the **`*PC`** command.

```
*PT1,10,tout
}
} Process Code
}
}
*LAtout
*BB
*IV
*AV-999.9
*MV1
CM4,0
WTNo Communications
*BI
```

If this code is added to the existing example, the displayed value will to go to "–999.9" and the message "No Communications" will be added to the screen display, alerting the user to a problem. This will only happen when the host fails to send the pattern for a period of 10 seconds. A **CL4** placed in the first line of the **`*LAgotit`** section of the program will ensure the "No Communication" message is removed when a valid pattern is next received.

Our final script, with all the facilities added is shown below:

```
*BB
SD
NS
F2
CM2,0
DV1,6,1,0,1
CM3,0
DB1,120,0,0,0
DL1,0,100
*PS1,9
*PE1,gotit
*PD1,1,2
*PD1,9,3
*PT1,10,tout
*PC1,1
*BI
*LAgotit
*BB
CL4
*PJ1,2,128,0,pos
*AV-
*LApos
*PV1,3,6
*MV1
F1
CM2,61
*PJ1,2,7,1,addkv
*PJ1,2,7,2,addv
*PJ1,2,7,4,addmv
*GTdone
*LAaddkv
WTkV
*GTdone
*LAaddv
WTV
*GTdone
*LAaddmv
WTmV
*LAdone
EL
*BI
*LATout
*BB
*IV
*AV-999.9
*MV1
F1
CM4,0
WTNo Communications
*BI
```

Responding to the host

In some cases it is likely that the user will want to send the a response to the host computer to tell it to change range or to continue with the process. The `*PR`***`string`*** Pattern Response command allows the user to do this. The string is limited to 28 bytes by the maximum line length. Non printable characters can be sent by using a backslash "\" to tell the handler that the next two bytes are hex representation. E.g. \03 would send the 0x03 value or [ETX]. To send a backslash then use "\\". The system does not do anything else while sending the string.

e.g. **`*PR\020.02V\0D`** would send **`[STX]0.02V[CR]`**

Note that this command can only be used by enabling the Pattern Matching function, and **cannot be used in normal scripts** as these responses will corrupt the host communications.

Packet Matching Commands

Note that Patterns, Events and Jumps must be defined before issuing the *PR command to enable the pattern matching function.

Command	Meaning	Description
*AV*string*	Add to Variable	Add the specified *string* to the cyclic data store *string* is any ASCII string up to 10 bytes long
*IV	Initialise Variable	Initialise the cyclic data store to a null string
*MVn	Make Variable	Convert the cyclic data store to the cyclic data value IN_*n* *n* is in the range 1 to 8 (IN_1 to IN_8)
*PC*n*,*m*	Pattern Control	Enable (*m*=1) or disable (*m*=0) pattern number *n* *n* is in the range 0 to 4, 0 being all patterns *m* is either 1 or 0 (enable or disable) Patterns, events and jumps must be defined before this command A short pause in script operation occurs when the first pattern is enabled
*PD*n*,*m*,*p*	Pattern Definition	Define pattern number *n* to match when byte *m* matches *p* *n* is in the range 1 to 4 *m* is in the range 1 to 32 *p* is in the range 1 to 255 Note that unspecified bytes are assumed to match.
*PE*n*,*label*	Pattern Event	Jumps to Event label *label* when pattern number *n* has been matched *n* is in the range 1 to 4
*PJ*n*,*m*,*p*,*q*,*label*	Pattern Jump	Byte *m* of pattern number *n* is masked with value *p* and compared to the expected value *q*. If it matches the script jumps to label *label* *n* is in the range 1 to 4 *m* is in the range 1 to 32 *p* is in the range 1 to 255 *q* is in the range 1 to 255
*PK*n*	Pattern Kill	Deletes and disables pattern number *n* *n* is in the range 1 to 4
*PR*string*	Pattern Response	Sends the specified string to the host system *string* is any ASCII string up to 28 bytes long Non-printable characters are sent as hex values using the format: Backslash followed by two ascii-hex values e.g. \0D 0..9, A..F and a..f can be used If a backslash is required in a string, send two backslashes together i.e.\\
*PS*n*,*m*	Pattern Size	Specifies the size of pattern number *n* as *m* bytes *n* is in the range 1 to 4 *m* is in the range 1 to 32
*PT*n*,*m*,*label*	Pattern Timeout	Jumps to Event label *label* if a timeout of *m* seconds for pattern *n* is exceeded. *n* is in the range 1 to 4 *m* is in the range 0 to 255 (*m*=0 disables the event)
*PV*n*,*m*,*p*	Pattern Variable	Write *p* bytes from pattern number *n* starting from byte *m* to the cyclic data store *n* is in the range 1 to 4 *m* is in the range 1 to 32 *p* is in the range 1 to 10
*PW*n*,*m*,*p*	Pattern Write	Write *p* bytes from pattern number *n* starting at byte *m* to the screen *n* is in the range 1 to 4 *m* is in the range 1 to 32 *p* is in the range 1 to 32 Note that normal screen attributes are applied.
*VD	Variable Debug	Enables error reporting of bad characters and float conversion errors in the *PV and *MV commands. Disabled by default.

BEKA
associates

BEKA Associates
Old Charlton Road
Hitchin
Hertfordshire
SG5 2DA

Tel:	+44 (0)1462 438301
Fax:	+44 (0)1462 453971
Web:	**www.beka.co.uk**
Email:	**support@beka.co.uk**
or	**sales@beka.co.uk**