Matrix+ Command Reference (R1)

Command mnemonics are listed alphabetically. The 24 Matrix+ new commands are highlighted:

AnimSchAdd AnXXXX $
Adds an animation file AnXXXX to the animation schedulers list. Files will be loaded in the order they are listed, when the animation engine encounters a AnimSchESC command in the animation file it is running. The $ text reference option can be edited to describe the animation file, making the list easier to understand

AnimSchClr
Clear the animation schedulers list, and reset the scheduler. In effect turning off this function, or preparing it for a new list of animation files to be added using AnimSchAdd. As a schedule list is stored in an animation file, one list can call up another, which can clear the current list and add new schedule. Making the process unlimited.

AnimSchESC
Animation exit point, used only if the scheduler is running, having been pre-loaded with animation file references. This command is placed at a point in your animation script where an exit to another animation is permitted; normally after a number of counted cycles, or prior to a cycle repeating itself. On encountering this command, the animation engine will check to see if a schedule has been loaded. If so, it will load the designated animation file, otherwise it will ignore this command and continue with the current animation indefinitely.

AnimSchRun N
Invoke the animation scheduler, if one or more file references have been loaded using the AnimSchAdd command. This command is normally placed at the end of a list of scheduled files. The number N is used to reference the first animation file to run; which does not have to be the first one in the list. Note that setting N=0 refers to the first file reference, and N=1 the second, and so on.

BckgndDim N
Dim the whole background frame by an amount N, towards a pixel RGB level of zero. Depending on the value of the pixel, this command may need to be called several times to get to a value of zero. The same value of N is subtracted from each RGB component. Hence the contribution of each colour and the combined colour effect, will change, as the values get smaller.

BckgndDimE2
Dim all background frame buffer pixels by a half of their current value. If called repeatedly, this will dim the background buffer pixels to RGB 0,0,0 in 9 cycles (ie. 255, 127, 63, 31, 15, 7, 3, 1, 0) from max to off.

BckgndDimTo [RGB] D
Dim background frame towards a target RGB value, by an incrementing value of D. Depending on the initial colour of the background, and to what degree it differs from the target RGB colour, this command may need to be used several times in order for the target RGB colour to be reached.

BckgndLoad FrXXXX
Load image file FrXXXX, from the microSD card, into background frame buffer. The time needed for this operation to complete is proportional to the number of pixels set in the source image file FrXXXX, and could have an adverse effect on achieving high frame rate settings.

BckgndNext FrXXXX FrYYYY
Load next background image file in a range, starting with FrXXXX and ending with FrYYYY. Once the last image file has been loaded, the next time this command is encountered the animation engine will cycle back to the first file, and repeat the whole cycle continuously.

BckgndRot Up/Right/Down/Left
Rotate the whole of the background image, in the buffer in one of 4 directions: Up/Right/Down/Left. This effect is like having the image wrapped round a barrel, which is rotating one pixel at each command.

BckgndWinFlip X0,Y0 X1,Y1 Left-Right/Up-Down
Flip part of the image in the background buffer, in a region defined by X0Y0,X1Y1 in one of 2 directions; left-Right about the Y-axis, or Up-Down about the X-axis.

BckgndWinRot X0,Y0 X1,Y1 Up/Right/Down/Left
Rotate part of the image in the background buffer, in a region defined by X0Y0,X1Y1, in one of 4 directions; Up, Right, Down, Left. This effect, is like having that part of the image wrapped round a barrel, which is rotating one pixel at each command. This is useful for scrolling objects or text in a scene.

Bright B
Set the overall brightness value 0 – 255 for the whole panel. The FastLED library used in this project has a function which effectively controls the brightness of each and every pixel value as they are sent to the LED strip display.

This command is normally placed at the beginning of your script, but can be used at any time to adjust the display. Note that the Brightness slider within the MATRIX app, works in the same way as this command. Bright adjustment does not apply to the Matrix simulation engine, only to the LED display.

Bright=Var **'A' - 'Z'**

Set the overall brightness value, between 0 – 255 for the whole panel, using the value of a variable 'A' - 'Z'. This allows you to use variables as counters within the script to effect brightness changes dynamically. The FastLED library used in this project has a function which effectively controls the brightness of each and every pixel value as they are sent to the LED strip display. Note that the Brightness slider within the MATRIX app, works in the same way, using the FastLED command. Bright adjustment does not apply to the Matrix simulation engine, only to the LED display.

ClrBckgnd

Clear the background frame buffer, effectively setting the value of each LED to RGB 0,0,0. This is often used to remove a previous image, before loading another, as image files are effectively transparent and only contain data on pixel colours, not for those that are off.

ClrBckgndXY **X0,Y0** **X1,Y1**

Clear the rectangular region X0,Y0 to X1,Y1 within the background frame buffer, effectively setting the value of each LED to RGB 0,0,0. This is used to remove parts of an image, and effectively makes that part of the background image transparent when it is loaded into the Frame buffer.

ClrForgnd

Clear the foreground frame buffer, effectively setting the value of each LED to RGB 0,0,0. This is often used to remove a previous image, before loading another, as image files are effectively transparent and only contain data on pixel colours, not for those that are off.

ClrForgndXY **X0,Y0** **X1,Y1**

Clear the rectangular region X0,Y0 to X1,Y1 within the foreground frame buffer, effectively setting the value of each LED to RGB 0,0,0. This is used to remove parts of an image, and effectively makes that part of the foreground image transparent when it is loaded into the Frame buffer.

ClrFrame

Clear the output Frame buffer, effectively setting the value of each LED to RGB 0,0,0. This is often used to remove a previous image, before loading another, as image files are effectively transparent and only contain data on pixel colours, not for those that are off.

ClrFrameXY **X0,Y0** **X1,Y1**

Clear the rectangular region X0,Y0 to X1,Y1 within the output frame buffer, effectively setting the value of each LED to RGB 0,0,0. This is used to remove parts of a previously loaded image, creating a window into which text or sprites can be draw.

ClrSprite{M} **{00000000}**

Clear Sprites {M} image store. An 8-bit mask is used with many sprite commands, to reduce the number of commands needed. Each bit in the mask represents one of the 8 sprites. Sprite 0 is controlled by the rightmost least significant bit, and Sprite 7 is controlled by the leftmost, most significant bit. For example, to clear Sprite 1, which is the second sprite, we would set the mask to {00000010}. One, more, or all bits can be set in this way, to clear up to all 8 sprites with one command. Ie. {11111111}

DecVar **'A'-'Z'**

Decrement the assigned integer variable 'A' - 'Z' by 1. Which for example is equivalent to A = A - 1

Delay **D**

Set a 'Show' delay, in speed period units. Whilst the 'Speed' value sets the overall rate at which 'Show' commands are executed, there are time when we may want to temporarily slow down the animation process. The delay value **D** sets the number of times the 'Show' event is skipped, before being enacted upon. For example, a Delay of 1 effectively halves the speed of that part of the animation, by skipping a 'Show' event on each cycle.

Delay=[V] **'A'-'Z'**

Set a 'Show' delay, based on the value of the assigned variable, in speed period units. Whilst the 'Speed' value sets the over rate at which 'Show' commands are executed, there are time when we may want to temporarily slow down the animation process. The delay value D sets the number of times the 'Show' event is skipped, before being enacted upon. For example, a Delay of 1 effectively halves the speed of that part of the animation, by skipping a 'Show' event on each cycle.

DelayRnd **D0** **D1**

Set a 'Show' delay random range. A random Delay value, ranging from D0 to D1 inclusive, will be generated. This is used to set a 'Show' delay, based on the value of the assigned variable, in speed period units. Whilst the 'Speed' value sets the over rate at which 'Show' commands are executed, there are time when we may want to temporarily slow down the animation process. The delay value D sets the number of times the 'Show' event is skipped, before being enacted upon. For example, a Delay of 1 effectively halves the speed of that part of the animation, by skipping a 'Show' event on each cycle.

DrwBarRndXY X0,Y0 X1,Y1 W,H

Draw a filled box, using the current Pen colour, at a random location bound by the values of X0Y0 to X1Y1, with a width W and height H. The value of X will vary between X0 and X1 inclusive, and the range of Y will vary between Y0 and Y1 inclusive.

DrawBar[V,V] 'X','Y' 'W','H'

Draw a filled box using values obtained from the four variables, using the Pen colour. As variables, acting as counters, can be changed dynamically as the script is run, this for example enables you to move a bar across the display, and vary its size as it proceeds.

DrawBarXYWH X0,Y0 W,H

Draw a filled box using the four values, and the Pen colour. This enables you to draw a bar of a given size and colour at a specific location on the display. The X0,Y0 co-ordinates are the top left corner pixel of the bar.

DrwBoxRndXY X0,Y0 X1,Y1 W,H

Draw a box, using the current Pen colour, at a random location bound by the values of X0Y0 to X1Y1, with a width W and height H. The value of X will vary between X0 and X1 inclusive, and the range of Y will vary between Y0 and Y1 inclusive.

DrawBox[V,V] 'X','Y' 'W','H'

Draw a box using values obtained from the four variables, using the Pen colour. As variables, acting as counters, can be changed dynamically as the script is run, this for example enables you to move a box across the display, and vary its size as it proceeds.

DrawBoxXYWH X0,Y0 W,H

Draw a box using the four values, and the Pen colour. This enables you to draw a box of a given size and colour at a specific location on the display. The X0,Y0 co-ordinates are the top left corner pixel of the box.

DrawLineA-B X0,Y0 X1,Y1

Draw a line, using the current Pen colour, from X0,Y0 to X1,Y1. The command accepts +ve and -ve numbers for X and Y values, and does not require X1 to be greater than X0 for example. Also co-ordinates can be outside of the display area; in which case those part of the line will clearly not be draw.

DrawLineRnd X0,Y0 X1,Y1

Draw a random line, using the current Pen colour, within a region defined by X0,Y0 to X1,Y1. The start and end points will be constrained to be within the region specified, but are not likely to be at their limits. This naturally generates lines of varying length, and varying orientation.

DrawLine[V,V] 'A','B' 'X','Y'

Draw a line, using the current Pen colour, with end points set by the values of variables from A,B to X,Y where variable 'A' represents co-ordinate X0, 'B' represents co-ordinate Y0, and so on. This enables you to use variables as counters within your script, to move a line across the display area for example.

Else ------

Must be used in conjunction with an If… command, and placed before the associated EndIf command. When the condition tested in the If command fails, then the animation engine will run the code immediately following the Else command, as the alternative path. An EndIf command should then be placed to terminate that block of code.

EndIf -----

This command terminates an 'If' command code block. This command must be used to close off a conditional 'If' command, so that the animation engine knows where to proceed from, should the 'If' condition not be met. The MATRIX app runs checks in the background on your script as it evolves, to ensure that 'EndIf' commands are being used correctly. In the left-hand select column a '/' character will appear next to each 'If' command, followed by a vertical line or + character until the appropriate number of 'EndIf' commands have been encountered.

EndLoop

Branch to the 'Loop' + 1 line. Normally your script will contain a main, so that the commands run continuously, rather than once. This statement acts as the end point for that loop, directing the animation engine to go back to the line immediately following the 'Loop' command.

FillBckgndRGB [RGB]
Fill background frame buffer with a specified colour in RGB format. The whole frame is filled with the specified colour and any existing graphics will be lost.

FillForgndRGB [RGB]
Fill foreground frame buffer with a specified colour in RGB format. The whole frame is filled with the specified colour and any existing graphics will be lost.

FillFrameRGB [RGB]
Fill output Frame buffer with a specified colour in RGB format. The whole frame is filled with the specified colour and any existing graphics will be lost.

Font[W] Normal/Narrow
Define the font style to be used when drawing text in an animation. The narrow option works by stripping out columns of pixels in the original character; there by making it narrower. Note that this option does not affect the font used in text scrolling, it will be normal font by default.

For... N
Start a For... until Next loop, N times loop, to run a section of code repeatedly for a fixed number of cycles. These loops can be nested to 10 levels maximum.

ForgndDim N
Dim the whole foreground frame by an amount N, towards a pixel RGB level of zero. Depending on the value of the pixel, this command many need to be called several times to get to a value of zero. The same value of N is subtracted from each RGB component. Hence the contribution of each colour and the combined colour effect, will change, as the values get smaller.

ForgndDimE2
Dim all foreground frame buffer pixels by a half of their current value. If called repeatedly, this will dim the foreground buffer pixels to RGB 0,0,0 in 9 cycles (ie. 255, 127, 63, 31, 15, 7, 3, 1, 0) from max to off.

ForgndDimTo [RGB] D
Dim foreground frame towards a target RGB value, by an incrementing value of D. Depending on the initial colour of the background, and to what degree it differs from the target RGB colour, this command may need to be used several times in order for the target RGB colour to be reached.

Frontload FrXXXX
Load image file FrXXXX, from the microSD card, into foreground frame buffer. The time needed for this operation to complete is proportional to the number of pixels set in the source image file FRXXXX, and could have an adverse effect on achieving high frame rate settings.

ForgndNext FrXXXX FrYYYY
Load next foreground image file in a range, starting with FrXXXX and ending with FrYYYY. Once the last image file has been loaded, the next time this command is encountered the animation engine will cycle back to the first file, and repeat the whole cycle continuously.

ForgndRot Up/Right/Down/Left
Rotate the whole of the foreground image, in the buffer in one of 4 directions: Up/Right/Down/Left. This effect is like having the image wrapped round a barrel, which is rotating one pixel at each command.

ForgndWinFlip X0,Y0 X1,Y1 Left-Right/Up-Down
Flip part of the image in the foreground buffer, in a region defined by X0Y0,X1Y1 in one of 2 directions; left-Right about the Y-axis, or Up-Down about the X-axis.

ForgndWinRot X0,Y0 X1,Y1 Left-Right/Up-Down
Rotate part of the image in the foreground buffer, in a region defined by X0Y0,X1Y1, in one of 4 directions; Up, Right, Down, Left. This effect, is like having that part of the image wrapped round a barrel, which is rotating one pixel at each command. This is useful for scrolling objects or text in a scene.

ForRnd N0 N1
Start a For... until Next loop, for a random number of N0 to N1 times loop, to run a section of code repeatedly for a random number of cycles. The minimum number of cycles will be N0 times and the maximum N1 time. These loops can be nested to 10 levels maximum.

FrameDim N
Dim the whole output Frame by an amount N, towards a pixel RGB level of zero. Depending on the value of the pixel, this command may need to be called several times to get to a value of zero. The same value of N is subtracted from each RGB component. Hence the contribution of each colour and the combined colour effect, will change, as the values get smaller.

FrameDimE2

Dim all output Frame buffer pixels by a half of their current value. If called repeatedly, in association with 'Show' commands, this will dim the display in 9 cycles (ie. 255, 127, 63, 31, 15, 7, 3, 1, 0) from max to off.

FrameDimTo [RGB] D

Dim output Frame pixels towards a target RGB value, by an incrementing value of D. Depending on the initial colour of the frame, and to what degree it differs from the target RGB colour, this command may need to be used several times in order for the target RGB colour to be reached.

FrameLoad FrXXXX

Load image file FrXXXX, from the microSD card, into the output Frame buffer. The time needed for this operation to complete is proportional to the number of pixels set in the source image file FRXXXX, and could have an adverse effect on achieving high frame rate settings.

FrameNext FrXXXX FrYYYY

Load next output Frame image file in a range, starting with FrXXXX and ending with FrYYYY. Once the last image file has been loaded, the next time this command is encountered the animation engine will cycle back to the first file, and repeat the whole cycle continuously.

FrameRot Up/Right/Down/Left

Rotate the whole of the output Frame image, in the buffer in one of 4 directions: Up/Right/Down/Left. This effect is like having the image wrapped round a barrel, which is rotating one pixel at each command.

FrameWinFlip X0,Y0 X1,Y1 Left-Right/Up-Down

Flip part of the image in the output Frame buffer, in a region defined by X0Y0,X1Y1 in one of 2 directions; left-Right about the Y-axis, or Up-Down about the X-axis.

FrameWinRot X0,Y0 X1,Y1 Up/Right/Down/Left

Rotate part of the image in the output Frame buffer, in a region defined by X0Y0,X1Y1, in one of 4 directions; Up, Right, Down, Left. This effect, is like having that part of the image wrapped round a barrel, which is rotating one pixel at each command. This is useful for scrolling objects or text in a scene.

FrameWxH W=ww H=hhh

This tells the animation engine the dimensions of your display in pixels (LEDs), enabling it to correctly calculate pixel co-ordinates in X and Y. The command is on the 2nd line of your script. The values of W and H can be modified by clicking on them, but the line can't be removed or replaced. Default values are W=48 and H=16. They are held within the range W= 16 to 48, and H= 16 to 128. If the frame width, set in the MATRIX+ app, does not match that of the script, when the animation is run, this will result in an error and the animation wont run.

Gosub $

Branch to a labelled $ line, whilst storing the return line reference for when a Return command is encountered. Whilst line labels are unique, a returning function can return to different parts of the script that have Gosub calls.

GosubIfSwitch $ SwN State

Branch to a labelled $ line, if switch SwN has been pressed, whilst storing the return line reference for when a Return command is encountered. Whilst line labels must be unique, a returning function can return to different parts of the script that have Gosub calls. Values of SwN range from 0 to 15, representing the 16 switches in a 4x4 switch matrix. States and transition are defined as:

Off - switch is open
Off/On - switch has been pressed since it was last checked
On - switch is closed
On/Off - switch has been released since it was last checked

GosubIfSwMat $

Branch to a labelled $ line, if any switch has been pressed, whilst storing the return line reference for when a Return command is encountered. Whilst line labels must be unique, the returning function can return to different parts of the script that have Gosub calls.

GoTo $

Branch to a labelled $ line, and continue programme execution from there. This is one way of creating a simple loop, or could be part of an If… code block, changing the overall function of the script depending on variables.

GoToIfScroll $

Branch to a labelled $ line, if scrolling text has not completed the overall scrolling process. This is a way of creating a loop, which waits for the text scrolling to complete before getting on with other parts of the animation.

GoToIfSprit{M} $ {00000000}

Branch to labelled $ line if the specified sprites are still moving, and not at their target co-ordinates. This is a way of creating a loop, which waits for one or more sprites to reach their target destinations. Each bit in the 8-bit mask represents one of the 8 sprites. Sprite 0 is controlled by the rightmost least significant bit, and Sprite 7 is

controlled by the leftmost, most significant bit. For example, to test the movement of Sprite 1, which is the second sprite, we would set the mask to {00000010}. One, more, or all bits can be set in this way, to test up to all 8 sprites with one command. Ie. {11111111}

GoToIfSwitch $ SwN State

Branch to a labelled $ line, if switch SwN has been pressed, and continue program execution from there. This is one way of creating a simple loop, or could be part of an If… code block, changing the overall function of the script depending on variables. Values of SwN range from 0 to 15, representing the 16 switches in a 4x4 switch matrix. States and transition are defined as:

Off - switch is open
Off/On - switch has been pressed since it was last checked
On - switch is closed
On/Off - switch has been released since it was last checked

GotoIfSwMat[] $

Branch to a labelled $ line, if any switch has been pressed, and continue program execution from there. This is one way of creating a simple loop, or could be part of an If… code block, changing the overall function of the script depending on variables.

IcMax N

Set the maximum LED current limit to the value of **N**, between 0.1 – 50.0 amps. This feature is off by default, but can be turned on at any point in the animation by using this command and making IcMax > 0. IN the active state the animation engine will build the frame graphics as normal, including the placement of sprites, then prior to the 'Show' event it will scan the output array to determine which LEDs are on, and estimate their cumulative power consumption. From this it can determine whether the current brightness level will cause the panel to draw more current than specified in IcMax, and if it would, it will automatically reduce the brightness for the show event to ensure that the current IcMax is not exceeded. If you are using low power supplies of small battery packs, this feature allows you to drive any animation with confidence. If reporting is ON you will see figures for the cumulative LED count, the predicted current at max brightness, the current brightness and the panel current.

IcOff

This command turns of the LED current monitoring function IcMax. With that function disabled, there is a risk that your LED patterns could draw more current than your 5v power source is capable of providing.

IfNotSpirate{M} {00000000}

If the specified sprites are not moving, and are at their target co-ordinates, then perform the commands placed after this command and before the next EndIf command. This is a way of creating scripts which do specific things once one or more sprites have reached their target destinations, like changing their image. Each bit in the 8-bit mask represents one of the 8 sprites. Sprite 0 is controlled by the rightmost least significant bit, and Sprite 7 is controlled by the leftmost, most significant bit. For example, to test the movement of Sprite 1, which is the second sprite, we would set the mask to {00000010}. One, more, or all bits can be set in this way, to test up to all 8 sprites with one command. Ie. {11111111}

IfSprite{M} {00000000}

If the specified sprites are moving, and not at their target co-ordinates, then perform the commands placed after this command and before the next EndIf command. This is a way of creating scripts which do specific things whilst waiting for one or more sprites to reach their target destinations, like rotating them as they move. Each bit in the 8-bit mask represents one of the 8 sprites. Sprite 0 is controlled by the rightmost least significant bit, and Sprite 7 is controlled by the leftmost, most significant bit. For example, to test the movement of Sprite 1, which is the second sprite, we would set the mask to {00000010}. One, more, or all bits can be set in this way, to test up to all 8 sprites with one command. Ie. {11111111}

IfSw[?]AnimLd N {state} AnXXXX

The specified switch state or transition is tested, and if true, then load and run the specified animation file AnXXXX. Values of N range from 0 to 15, representing the 16 switches in the 4x4 switch matrix. States and transition are defined as:

Off - switch is open
Off/On - switch has been pressed since it was last checked
On - switch is closed
On/Off - switch has been released since it was last checked

Using this command will turn on the keypad matrix scanner if it was not already activated, with a KeypadON command, as default. If you want this to work with Touchpads, then you need to have a TouchpadON command in your script, before this command, and the range of value N is restricted to 0 to 8 for GPIO touchpad pins.

IfSwitch[?] N {state}

The specified switch state or transition is tested, and if true, then perform the commands placed after this commend, and before the next EndIf command. Values of N range from 0 to 15, representing the 16 switches in the 4x4 switch matrix. States and transition are defined as:

Off - switch is open
Off/On - switch has been pressed since it was last checked
On - switch is closed
On/Off - switch has been released since it was last checked

Using this command will turn on the keypad matrix scanner if it was not already activated with a KeypadON command.

IfSwMatrix[]

Test for any switch being pressed, and if true then perform the commands placed after this command, and before the next EndIf command. The provides an efficient method of looking for a switch operation, without the need to read all 16 looking for one. Using this command will turn on the keypad matrix scanner if it was not already activated with a KeypadON command.

IfVar > 'A' > N

If the specified variable ('A' - 'Z') is greater than (>) the value N assigned, do the commands that follow until an EndIf command is encountered. Otherwise jump to the line immediately after the EndIf command.

IfVar >= 'A' >= N

If the specified variable ('A' - 'Z') is greater or equal to (>=) the value N assigned, do the commands that follow until an EndIf command is encountered. Otherwise jump to the line immediately after the EndIf command.

IfVar == 'A' == N

If the specified variable ('A' - 'Z') is equal to (==) the value N assigned, do the commands that follow until an EndIf command is encountered. Otherwise jump to the line immediately after the EndIf command.

IfVar != 'A' != N

If the specified variable ('A' - 'Z') is not equal to (!=) the value N assigned, do the commands that follow until an EndIf command is encountered. Otherwise jump to the line immediately after the EndIf command.

IfVar <= 'A' <= N

If the specified variable ('A' - 'Z') is less than or equal to (<=) the value N assigned, do the commands that follow until an EndIf command is encountered. Otherwise jump to the line immediately after the EndIf command.

IfVar < 'A' < N

If the specified variable ('A' - 'Z') is less than (<=) the value N assigned, do the commands that follow until an EndIf command is encountered. Otherwise jump to the line immediately after the EndIf command.

IfVar > Var 'A' > 'B'

If the specified variable ('A' - 'Z') is greater than (>) the assigned variable 'B', do the commands that follow until an EndIf command is encountered. Otherwise jump to the line immediately after the EndIf command.

IfVar >= Var 'A' >= 'B'

If the specified variable ('A' - 'Z') is greater or equal to (>=) the assigned variable 'B', do the commands that follow until an EndIf command is encountered. Otherwise jump to the line immediately after the EndIf command.

IfVar == Var 'A' == 'B'

If the specified variable ('A' - 'Z') is equal to (==) the assigned variable 'B', do the commands that follow until an EndIf command is encountered. Otherwise jump to the line immediately after the EndIf command.

IfVar != Var 'A' != 'B'

If the specified variable ('A' - 'Z') is not equal to (!=) the assigned variable 'B', do the commands that follow until an EndIf command is encountered. Otherwise jump to the line immediately after the EndIf command.

IfVar <= Var 'A' <= 'B'

If the specified variable ('A' - 'Z') is less than or equal to (<=) the assigned variable 'B', do the commands that follow until an EndIf command is encountered. Otherwise jump to the line immediately after the EndIf command.

IfVar < Var 'A' < 'B'

If the specified variable ('A' - 'Z') is less than (<=) the assigned variable 'B', do the commands that follow until an EndIf command is encountered. Otherwise jump to the line immediately after the EndIf command.

IncVar 'A'

Increment the integer variable 'A' - 'Z' by +1. Which for example is equivalent to A = A + 1

IncVarToVar 'A' To 'B'

Increment the integer variable 'A' - 'Z' by +/-1 as needed in order to move the value of 'A' towards that of 'B'. Which for example is equivalent to: if (A < B) {A = A + 1;} else if (A > B) {A = A – 1;}

KeypadON **true/false**
Makes the animation engines simulator keypad visible/invisible. Also clears the keypad state variables in animation engines, and activates the keypad matrix scanner software. If the Touchpad was ON, then it will be disabled, as you can only have one or the other.

Label: **$**
Assigns a text word **$** or phrase as a labelled line reference. Labels are normally used in association with branch instructions, like Gosub and GoTo. Note that the labels, and references to them, must be an exact match, and case sensitive. You can use a combination of numbers 1234, as well as cased sensitive text, like: 4Me2U and you can include spaces between words. All labels must be unique. The actual line reference number used by the animation engine is calculated whenever the animation file is being saved to disc, or sent to the micro over the USB link to run dynamically.

LoadAnim **AnXXXX** **$**
Immediately loads animation file AnXXXX, if it exists, and runs it. The current animation will stop, the panel will be erased, and all variables will be reset to their default values. The $ text reference option can be edited to describe the animation file you are loading, making it easier to recognise when viewed later.

LoadBckgnd
Copies background frame buffer into the output Frame buffer. Only the pixels that have RGB values are actually copied, so an image can be perceived as being transparent in parts.

LoadBckWin[V] **'A','B'** **'C','D'** **'X','Y'**
Copies a region within the background frame buffer **A**,**B** to **C**,**D** into the output Frame buffer at a specified location **X**,**Y** using variables. Any of the 'A' – 'Z' variables can be used in the command. Only the pixels that have RGB values set in the background are actually copied, so an image can be perceived as being transparent in parts.

LoadBckWinXY **X0,Y0** **X1,Y1** **X2,Y2**
Copies a region within the background frame buffer **X0**,**Y0** to **X1**,**Y1** into the output Frame buffer at a specified location **X2**,**Y2** using specific values. Values are set in the range -127 to +128. Only the pixels that have RGB values set in the background are actually copied, so an image can be perceived as being transparent in parts.

LoadForgnd
Copies foreground frame buffer into the output Frame buffer. Only the pixels that have RGB values are actually copied, so an image can be perceived as being transparent in parts.

LoadForWin[V] **'A','B'** **'C','D'** **'X','Y'**
Copies a region within the foreground frame buffer **A**,**B** to **C**,**D** into the output Frame buffer at a specified location **X**,**Y** using variables. Any of the 'A' – 'Z' variables can be used in the command. Only the pixels that have RGB values set in the foreground are actually copied, so an image can be perceived as being transparent in parts.

LoadForWinXY **X0,Y0** **X1,Y1** **X2,Y2**
Copies a region within the foreground frame buffer **X0**,**Y0** to **X1**,**Y1** into the output Frame buffer at a specified location **X2**,**Y2** using specific values. Values are set in the range -127 to +128. Only the pixels that have RGB values set in the foreground are actually copied, so an image can be perceived as being transparent in parts.

Loop
Start of a Loop... End Loop code block. When the animation engine encounters this command it saved the line reference for future use with an EndLoop command. There is normally only one loop in a script, but it is permissible to have more, provided they are standalone and not nested, one inside the other.

Next
Last line of a For...Next loop, in which the animation engine will decrement the For counter, and if it is not zero it will branch to the line immediately following the For command. Otherwise, if zero, execution of the script will continue from after this Next command.

Pause **S**
Sets a delay timer in 1/10ths second, where **S** is in the range from 0.1 – 60.0 seconds. When it encounters this command ESP32 the animation engine literally stops dead in its tracks, and waits for the time to pass.

PenRGB **[RGB]**
Assigns RGB colour values to the Pen ink, which is used in drawing functions like DrawBoxXYWH, etc.

PenRGB=[R,G,B] **'R'** **'G'** **'B'**
Assigns RGB colour values to the Pen ink to that of three variables, which is then used in drawing functions like DrawBoxXYWH. Note that the default 'R' 'G' 'B' variables can be any of the 'A' – 'Z' variables available.

PenRndRGB
Assigns random RGB colour values to the Pen ink based on an internal palette of 12 different colours. This provides a distinct, albeit limited in range, set of colours.

Report: text
Sends the specified text to the Matrix+ app console window. Useful in debugging your animation scripts, but not part of the LED animation. Use sparingly, as you can easily send a lot of text data over the serial link.

ReportON True/False
When the micros animation engine is running it can report information over the USB serial link to the MATRIX app, which in turn presents it in a black console window. This command allows that process to be turned either ON == True, or OFF == False. When the animation is running however, if OFF it can be turned back on again by clicking on the verbose button.

Return
Last line of a script function, that is expected to have been called from a Gosub... command. The first line of the function would be a reference Label. Coded functions of this type are normally placed towards the end of the script, but they can be called from any part of the script using the Gosub command.

SetVar= 'A' N
Set the value of an assigned variable 'A' - 'Z' to a value of N.

SetVar=Rnd 'A' = Rnd LL,UL
Set the value of an assigned variable 'A' - 'Z' to a random value, in the range of LL to UL inclusive.

Show
Sends contents of the Frame buffer data to the LED matrix over the serial wire.

Speed S
Assigns a new value to the frame speed, specified as S in frames per second (fps). Note that the frame speed can not be set below 1 fps, and that the upper limit may be reduced by the amount of script actions specified before a 'Show' command. The overall size of the LED matrix has an inverse effect on frame rate too, given the fixed data rate of the serial LED data, of 800 kb/s.

Speed=Var 'A'
Assigns a new value to the frame speed, contained in a variable 'A', in frames per second (fps). Note that the frame speed cannot be set below 1 fps, and that the upper limit may be reduced by the amount of script actions specified before a 'Show' command. The overall size of the LED matrix has an inverse effect on frame rate too, given the fixed data rate of the serial LED data, of 800 kb/s.

SpriteAtXY S X Y
Place sprite number S at location X,Y. Where sprite S is between 0 – 7, and the 0,0 co-ordinate reference is top left of the sprite 16x16 matrix. For sprites to appear in the output Frame you must also use the SpriteOn or SpriteOn{M} commands.

SpriteAtXY=[V] S 'X' 'Y'
Place sprite number S at location X,Y specified by variable values. Where sprite S is between 0 – 7, and the 0,0 co-ordinate reference is top left of the sprite 16x16 matrix. This allows for the position of sprites to be moved dynamically by variable based counters. For sprites to appear in the output Frame you must also use the SpriteOn or SpriteOn{M} commands.

SpriteAtRndXY S XLL,XUL YLL,YUL
Place sprite number S at location X,Y specified by two ranges of random values. Where sprite S is between 0 – 7, and the 0,0 co-ordinate reference is top left of the sprite 16x16 matrix. This allows for the position of sprites to be moved randomly within a range for X,Y. For sprites to appear in the output Frame you must also use the SpriteOn or SpriteOn{M} commands.

SpriteFlipH{M} {00000000} XLL,XUL YLL,YUL
Flip one or more sprite regions horizontally, for sprites specified by the mask {M} and the pixel regions on each sprite are bound by the same XLL,XUL and YLL,YUL values.

SpriteFlipV{M} {00000000} XLL,XUL YLL,YUL
Flip one or more sprite regions vertically, for sprites specified by the mask {M} and the pixel regions on each sprite are bound by the same XLL,XUL and YLL,YUL values.

SpriteLoad S SpXXXX
Load a 16x16 pixel image file into sprite S from file SpXXXX. Note that image files often contain a lot of transparency (pixels off), so the ClrSprite{M} is often used before this command, unless you deliberately want to combine the new image with the existing one.

SpriteOn S True/False
Turn sprite S visibility ON or OFF, defined by True and False. The animation engine looks at the ON/OFF status of each sprite just before a 'Show' event, and draws those that are set as ON onto the output Frame buffer.

SpriteOn{M} {00000000}
Turn sprite visibility ON or OFF, defined by the mask {00000000}, where a '1' means ON and a '0' means OFF. Sprite number 0 is the least significant bit of the mask, and sprite number 7 is controlled by the most significant bit of the mask{M}.The animation engine looks at the ON/OFF status of each sprite just before a 'Show' event, and draws those that are set as ON onto the output Frame buffer.

SpriteRndXY{M} {00000000} XLL,XUL YLL,YUL
Set sprites, controlled by the mask {0000000} to random X,Y target locations. A different random set of target values, in the ranges XLL – XUL and YLL – YUL, are determined for each sprite using this command. The sprites are moved one pixel step towards their respective targets. Subsequent movements are achieved using SpriteTo{M}.

SpriteRot{M} {00000000} Up/Right/Down/Left
Rotate sprites defined by the mask {0000000} in one of 4 directions, Up/Right/Down/Left. They each move by one pixel, in the specified direction of rotation, each time this command is encountered.

SpriteTo{M} {00000000}
Move sprites defined by the mask {0000000} towards their respective targets, by one pixel in the longest direction of travel. Depending on the distance needed to travel to reach the target co-ordinates, this command will need to be run several times. The subsequent movement within the LED array is a staircasing effect.

SpriteToXY S X Y
Move sprites S towards its target location, by one pixel in the longest direction of travel. Depending on the distance needed to travel to the target co-ordinates, this command will need to be run several times. The subsequent movement within the LED array is a staircasing effect.

SpriteToXY=[V] S 'X' 'Y'
Moves sprite S immediately to co-ordinates X,Y specified by variables 'X' and 'Y'. Note that these are default variables and any of the 'A' – 'Z' range can be used for either X,Y values. The 0,0 datum co-ordinate of a sprite is the top left-hand corner of the 16x16 matrix.

Text $
Specify a new Text string $, to be used in text drawing commands. This new text value completely replaces any previously specified value.

Text+ $
Appends more text $ to the current Text string to extend its value. This is useful for functions like text scrolling, where a long string of text may be needed. Several of these commands can be used to make very long strings.

TextAtXY X Y
Draw the previously specified Text string at X, Y on the output Frame buffer. The 0,0 datum reference for characters is the top left-hand corner. The internal font uses variable width characters and this is accounted for by the animation engine when drawing text strings.

TextAtXY=[V] 'X' 'Y '
Draw the previously specified Text string at a location specified by two variables 'X', 'Y ' on the output Frame buffer. The 0,0 datum reference for characters is the top left-hand corner. The internal font uses variable width characters and this is accounted for by the animation engine when drawing text strings.

TextCtrXY X Y
Draw the previously specified Text string centred at X, position Y on the output Frame buffer. The 0,0 datum reference for characters is the top left-hand corner. The internal font uses variable width characters and this is accounted for by the animation engine when drawing text strings.

TextCtrXY=[V] 'X' 'Y '
Draw the previously specified Text string at a location specified by two variables 'X', 'Y ' centred at the X, positioned on the Y, on the output Frame buffer. The 0,0 datum reference for characters is the top left-hand corner. The internal font uses variable width characters and this is accounted for by the animation engine when drawing text strings.

Text+Num=[V] 'T'
Adds a text string, representing the number contained in variable 'A', to the existing text. For example, if Text was preloaded with "Value=" and the variable 'T' equalled 15, then Text would become "Value=15".

TextNum=[V] 'T'

Sets the Text variable equal to a string representing the number contained in variable 'A'. The existing value of Text is replaced by this command. For example, if Text was preloaded with "Anything" and the variable 'T' equalled 21, then Text would become "21".

TextRGB [RGB]
Assigns an RGB colour to be used when drawing subsequent Text commands.

TextRGB=[V] 'R' 'G' 'B'
Assigns an RGB colour, defined by three variables, to be used when drawing subsequent Text commands. The default variables 'R' 'G' 'B' can be any of the available 'A' – 'Z' variable references.

TextRhtXY X Y
Draw the previously specified Text string right-justified from X, position Y on the output Frame buffer. The 0,0 datum reference for characters is the top left-hand corner. The internal font uses variable width characters and this is accounted for by the animation engine when drawing text strings.

TextRhtXY=[V] 'X' 'Y '
Draw the previously specified Text string at a location specified by two variables 'X', 'Y ', right-justified from the X, positioned on the Y, on the output Frame buffer. The 0,0 datum reference for characters is the top left-hand corner. The internal font uses variable width characters and this is accounted for by the animation engine when drawing text strings.

TextRndRGB
Assigns random RGB colour values to subsequent 'Text' and 'Word' commands, based on an internal palette of 12 different colours. This provides a distinct, albeit limited in range, set of colours.

TextRndXY XLL,XUL YLL,YUL
Draw previously defined Text at random X,Y start values, in the ranges XLL,XUL and YLL,YUL. Whilst these ranges can fall outside of the LED matrix area, they clearly won't be visible there.

TextScroll
If used repeated ahead of 'Show' commands the animation engine will scroll the text, previously loaded with the Text command, from right to left in the Frame using the previously assigned colour. The scroll region is the full width of the LED display. If scrolling has begun, you can test to see if it is still in progress using the GoToIfScroll command, and develop a local loop around this process.

TextScrollY N
Assigns the value N to the scrolling text Y value, and also resets the scrolling process.

TouchpadON true/false
Makes the animation engines simulator keypad visible/invisible, displaying the words 'Touchpad Keys'. Also clears the keypad state variables in animation engines, and activates the touchpad scanner software. If the Keypad was ON, then it will be disabled, as you can only have one or the other at any one time.

Var+=Var 'A' += 'B'
Adds the 2nd variable to the 1st variable, placing the answer in the 1st variable. As in: A = A + B

Var =Var 'A' = 'B'
Assigns the 2nd variables value to the 1st variable, placing the answer in the 1st variable. As in: A = B

Var-=Var 'A' -= 'B'
Subtracts the 2nd variable from the 1st variable, placing the answer in the 1st variable. As in: A = A - B

Var*=Var 'A' *= 'B'
Multiplies the 1st variable by the 2nd variable, placing the answer in the 1st variable. As in A = A * B

Var/=Var 'A' /= 'B'
Divides the 1st variable by the 2nd variable, placing the answer in the 1st variable. As in: A = A/B

Var%=Var 'A' %= 'B'
Divides the 1st variable by the 2nd variable, placing the remainder of the division in the 1st variable. As in: A = A%B. This can be useful, for example in determining if a number is odd or even, by dividing it by 2. As in: A = A%2.

Var&=Var 'A' &= 'B'
Performs a bit-wise logical AND of the 2nd variable with the 1st variable, placing the answer in the 1st variable. For example, if A = 0101 and B = 0100, then A & B = 0100

Var|=Var 'A' |= 'B'
Performs a bit-wise logical OR of the 2nd variable with the 1st variable, placing the answer in the 1st variable. For example, if A = 0001 and B = 0100, then A | B = 0101

Var^=Var 'A' ^= 'B'

Performs a bit-wise logical XOR of the 2nd variable with the 1st variable, placing the answer in the 1st variable. For example, if A = 0011 and B = 0110, then A ^ B = 0111

VarLimit[L,U] 'A' LL UL

Constrains the variable 'A' to be between lower limit LL and upper limit UL. Values of the variable can be between -32,768 to 32,767. In all cases the value of LL <= UL will be maintained.

WordAtXY X Y

Draws a Word, taken from Text, at X,Y on the output frame. A '~' tilde character can be used in the Text string as a blank word, to create gaps in your phrases. For example, if Text was assigned "Hello There", the Word command would draw this as two consecutive words "Hello" and "There". But if Text was "Hello ~ There" then the Word command would treat this as three separate words, but draw a blank for the "~" character.

WordAtXY=[V] 'X' 'Y'

Draws a Word, taken from Text, at a location on the output frame defined by two variables for X,Y. A '~' tilde character can be used in the Text string as a blank word, to create gaps in your phrases. For example, if Text was assigned "Hello There", the Word command would draw this as two consecutive words "Hello" and "There". But if Text was "Hello ~ There" then the Word command would treat this as three separate words, but draw a blank for the "~" character.

WordCtrXY X Y

Draw a Word, taken from Text, at a location on the output frame centred on X, position Y. The position of the word is based on its length and height. A '~' tilde character can be used in the Text string as a blank word, to create gaps in your phrases. For example, if Text was assigned "Hello There", the Word command would draw this as two consecutive words "Hello" and "There". But if Text was "Hello ~ There" then the Word command would treat this as three separate words, but draw a blank for the "~" character.

WordCtrXY=[V] 'X' 'Y'

Draws a Word, taken from Text, at a location on the output frame centred on variable X, position variable Y. A '~' tilde character can be used in the Text string as a blank word, to create gaps in your phrases. For example, if Text was assigned "Hello There", the Word command would draw this as two consecutive words "Hello" and "There". But if Text was "Hello ~ There" then the Word command would treat this as three separate words, but draw a blank for the "~" character.

WordNext

Get the next word from Text, or cycles back to the first, if the last word has been used. A '~' tilde character can be used as a blank word, to create gaps in your phrases.

WordRhtXY X Y

Draw a Word, taken from Text, at a location on the output frame right justified from X,Y. The position of the word is based on its length and height. A '~' tilde character can be used in the Text string as a blank word, to create gaps in your phrases. For example, if Text was assigned "Hello There", the Word command would draw this as two consecutive words "Hello" and "There". But if Text was "Hello ~ There" then the Word command would treat this as three separate words, but draw a blank for the "~" character.

WordRhtXY=[V] 'X' 'Y'

Draws a Word, taken from Text, at a location on the output frame right justified from the two variables X,Y. A '~' tilde character can be used in the Text string as a blank word, to create gaps in your phrases. For example, if Text was assigned "Hello There", the Word command would draw this as two consecutive words "Hello" and "There". But if Text was "Hello ~ There" then the Word command would treat this as three separate words, but draw a blank for the "~" character.

WordRndXY XLL,XUL YLL,YUL

Draws a Word, taken from Text, at a random location on the output frame, bound by the ranges of XLL,YLL and YLL,YUL.

// Any text

A line with the // characters in column 1 is a comment line. Enter text after the // in the animation list editor to explain elements of your code. Note this should not be confused with the DISABLE function, which shows the // in the column 0, the yellow column, and is in light blue.

.