



DaveScript CNC Macros

Code references and examples

Last updated: 05/09/2024

What is DaveScript?

DaveScript is a custom scripting language used within CNC3D Commander that is designed to create complex and dynamic macro scripts or canned cycles that can perform a variety of scripted tasks with a CNC controller.

CNC3D Commander and DaveScript can provide a lot of customized control over GRBL based CNC controllers and advanced GRBL controllers such as Nighthawk, grblHAL and xPROV5 controllers.

The screenshot displays the CNC3D Commander software interface. The top bar shows the IP address 192.168.1.171:23 and a Disconnect button. The main window is divided into several sections. On the left, there's a 'Machine' section with a table showing coordinates for X, Y, Z, and A axes. Below this are buttons for 'Home Machine', 'Go to Zero (XY)', 'Set Job', 'Zero Job', 'My Buttons', 'Probing', 'Coolant Flood', and 'Coolant Mist'. The center section has a 'Stored Macros' tab, which is currently active. It shows a list of macros on the left and a text editor on the right. The text editor contains a sample macro script. At the bottom, there are buttons for 'Import', 'Rename', 'Delete', 'New', 'Edit', and 'Run'. The bottom left corner features a large red 'EMERGENCY STOP' button and a row of colored buttons for X, Y, Z, A, P, D, F, and M.

Machine	Job
X	0.000 -779.000
Y	0.000 -1752.900
Z	0.000 -1.850
A	0.000 0.000

Home Machine Go to Zero (XY) Set Job Zero Job
My Buttons Probing Coolant Flood Coolant Mist

Y+ Z+
X- X+ A- A+
Y- Z-
15 mm 5 mm
5000 mm/m 5000 mm/m

EMERGENCY STOP X Y Z A P D F M
Idle Unlock Pause Keyboard Gamepad

Stored Macros

Easily execute a collection of presaved commands using DaveScript.

Macro Name
Sample macro from user
New test macro
if and pointer macro test
message test

```
@var var1 = 1
:ptr3
%var1% = inputbox{choose a number,\1 or other?|Choose|%var1%}
if{%var1%=1|goto ptr1|goto ptr2}
message{never going to happen} ;This line should never get hit
:ptr1
message{Pointer 1 was hit!\\I like unicorns\\Do you?}
message{I'm also a 2nd line in Pointer 1. Yay!!}
exit
:ptr2
message{Pointer 2 was hit!}
message{I'm also a 2nd line in Pointer 2. Yay!!}
message{Let's do another one in Pointer 2 just because we can!}
goto ptr2
exit
```

Import Rename Delete New Edit Run

Basic structure and Supported Functions

DaveScript supports an unlimited number of user defined variables and has several functions built in for customizing your CNC Macro scripts. All DaveScript macros are case *insensitive* along with any comparisons to keep things simple.

Comment lines

To help you keep track of what is going on in your script you can choose to add comments for your own note keeping. Comment lines must always start with semicolon ; character. Comments can be added on their own line or can be added to the end of a line. Comment lines are in no way used in the processing of your actual script and are designed purely for you to manage your script and what it is doing.

Examples of using Comments

```
;I am a comment line  
X50Y50 ;I am also a comment line but I am at the end of a line.
```

In this example, despite the comments only **X50Y50** will be sent to your CNC controller

User defined Variables

Variables hold values that can be used for creating customized G-code or for use in functions.

All variables MUST be declared before they can be used. To declare a variable, add **@var** to the start of the line and then add the variable name. You can additionally choose to add a default value to the variable at this point as well. If no default value is specified then the default value will be Nothing.

How to declare a variable

Example of declaring a variable without a value:

```
@var MyVariable
```

Example of declaring a variable with a value:

```
@var MyVariable = 20
```

How to assign a user assigned variable after it's declared

Variables being assigned must be enclosed in percentage % characters.

```
%MyVariable% = 50
```

How to use a user defined variable

Variables can be used when compiling Gcode commands or user messages and can also be used for calculations. To use them, make sure they are enclosed in percentage % characters.

```
@var MyVariable = 50  
@var answer  
%answer% = calc{%MyVariable%/2}
```

The code above will assign a value of **25(50/2)** to variable **%answer%**.

Notes regarding user defined variables

It is possible to declare variables at any point of your script. It just needs to be declared somewhere in the script. It is good practice to have all variables listed at the top of your script. Functions, such as Calc{}, can not be used during the assignment of a variable.

Internal Variables

Internal variables are fixed variables that provide environmental information about your CNC such as Machine and Work Offsets. They are always enclosed in hash # characters

A list of current internal Variables available

#[x-c]mpos# OR #[x-c]xm#	Specified axis Machine position for [X,Y,Z,A,B,C]. E.g: #xmpos# or #xm#
#[x-c]wpos# OR #[x-c]xw#	Specified axis Job position for [X,Y,Z,A,B,C]. E.g: #xmpos# or #xm#
#zmpos# OR #zm#	Current Z axis machine position
#[x-c]max#	Specified axis Maximum travel for [X,Y,Z,A,B,C]. E.g: #xmax#
#rand#	Random number generator

How to use an internal variable

Variables can be used when compiling Gcode commands or user messages and can also be used for calculations. To use them, make sure they are enclosed in percentage % characters.

```
@var MyVariable = 50
@var answer
%answer% = calc{%MyVariable%*#xw#}
```

In this example, we have declared a user variable called **MyVariable** and assigned it a default value of **50**. We have then made another variable called **answer**. %answer% then gets assigned the value of: **MyVariable * #xw#** (Current X work/job position).

Notes regarding Internal variables

The values of internal variables are updated before the execution of the next line of the script is executed. It is not possible to assign internal variables custom values. Their values are automatically controlled.

Functions

Functions in DaveScript are all single line in their formatting and share a common layout for parameters. Let's take a look at the basic structure of a DaveScript function.

All Functions are encased in parentheses { } and all parameters of functions are split by a pipe character |.

CALC{ }

Number of parameters: 1

The Calc{ } function is a very powerful calculator function. It can do extremely complex calculations involving multiple numbers and mathematical operators. Calc{ } MUST be the target of an assignment, meaning that it cannot be called on its own. It must be used when assigning a variable.

Basic Usage:

```
@var number
%number% = calc{(3*4)/(18*-38)}
```

First we assign a variable we can use to target the calc{ } function output by using the @var declaration. This function will assign variable **%number%** the mathematical answer to (3*4)/(18*-38) which is; -0.0175.

MESSAGE{

Number of parameters: 4

The Message{ } function is used to display information about what is happening in the script. It will pause current execution of the script while the message is displayed. It can be the target of an assignment or it can be called on its own. It has a few options relating to how it is displayed and can also be configured to provide the user with an option of choosing Yes or No rather than just an "OK" option.

Basic Overview

```
Message{ Prompt | Title | Type (0-2) | Yes/No (0-1) }
```

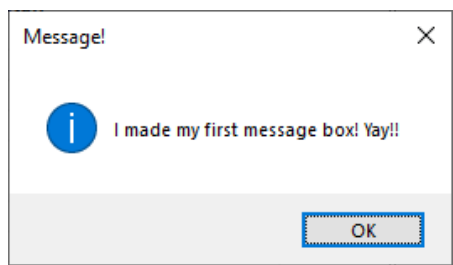
Parameter breakdown

Parameter	Optional	Description
Prompt	NO	The text to be displayed in the message.
Title	Yes	The title of the message box.
Type	Yes	The type of message to display, (0-2) [0 = Info, 1 = Alert, 2 = Error]
Yes/No	Yes	Display Yes/No options to the user (0-1) [0 = OK, 1 = Yes/No]

Basic Usage:

```
Message{ I made my first message box! Yay!! }
```

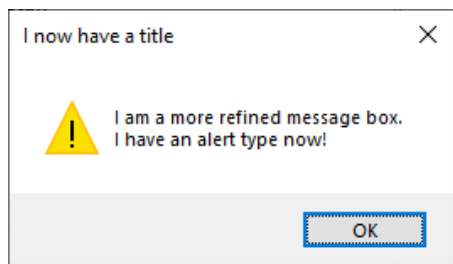
This will display a message box with the Text "I made my first message prompt! yay!" in the main message body.



Basic usage with parameters

```
Message{ I am a more refined message box.\nI have an alert type now! | I now have a title | 1 | 0 }
```

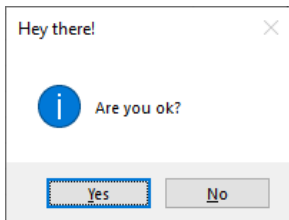
This will display a message box with the required prompt and display a title. It also has the "Type" parameter set to 1 so it shows as an Alert. Note the use of "\n" to create a new line. You can also use the traditional "\n" for a new line as well.



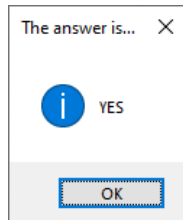
Advanced usage when assigned to a variable

```
@var result  
%result% = message{ Are you ok? | Hey there! | 0 | 1 }  
Message{ %result% | The answer is... }
```

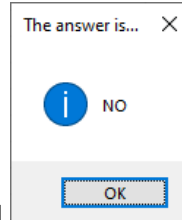
In this example, %result% is being assigned the result of a **Yes/No** message box. Then immediately after, we are showing another message box with the actual result. The responses to either Yes, No or OK are verbatim. Yes returns "YES", No returns "NO", Ok returns "OK".



Returns



if "Yes" is pressed and

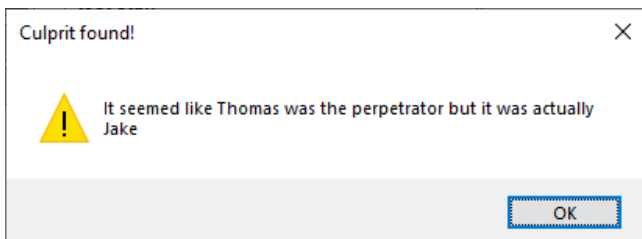


if "No" is pressed.

Using variables in messages

```
@var suspect1 = Thomas  
@var suspect2 = Jake  
Message{ It seemed like %suspect1% was the perpetrator but it was actually %suspect2% | Culprit found! | 1 }
```

In this example we declare 2 variables and give them default values. We then display the message to the user with the variables included in the prompt. You may note, variables do not need to be numeric, they can be text strings too.



INPUTBOX{}

Number of parameters: 3

The InputBox{} function is used to prompt the user for an input. It will pause current execution of the script while the Input is displayed. It MUST be the target of an assignment and can NOT be called on its own.

Basic Overview

```
InputBox{ Prompt | Title | Default Value }
```

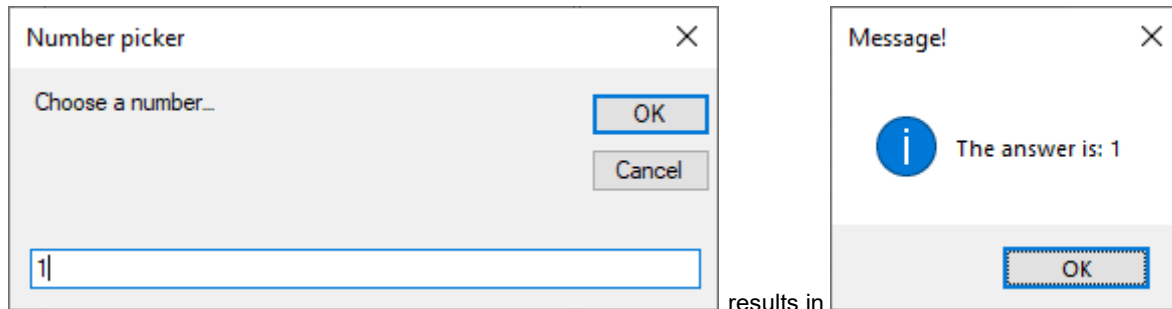
Parameter breakdown

Parameter	Optional	Description
Prompt	NO	The text to be displayed in the Input box.
Title	Yes	The title of the Input box.
Default value	Yes	A default value to display the user in the user entry field

Basic Usage:

```
@var result  
%result% = InputBox{ Choose a number... | Number picker | 1}  
Message{The answer is: %result%}
```

This will display a user Input box asking them to pick a number. Afterwards, a message will appear confirming their number.



The InputBox{} function allows you to change variables and outcomes in the middle of your script. This is handy for making slight adjustments to your Gcode or change what commands are sent.

IF{} Number of parameters: 3

The IF{} function can be used to make a selection based on a certain condition. It is made up of 3 parts. A condition to compare, a value indicating if that condition is True and an optional provision to do something if the condition is False.

Basic Overview

```
If{ Condition | IsTrue | IsFalse }
```

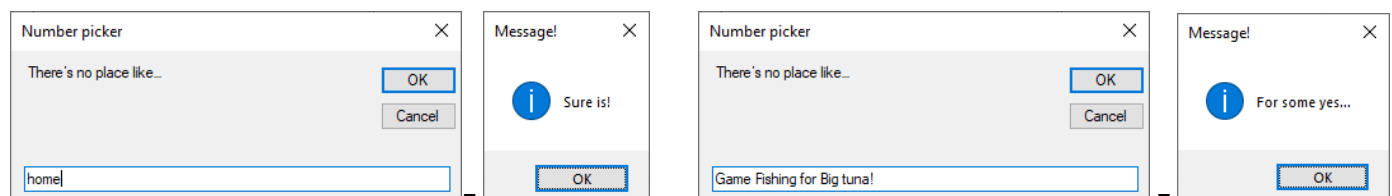
Parameter breakdown

Parameter	Optional	Description
Condition	NO	You are able to compare 2 things are equal or one is greater than or smaller than another.
Is True?	NO	The title of the Input box.
IsFalse?	Yes	A default value to display the user in the user entry field

Basic Usage:

```
@var prompt = home  
%prompt% = InputBox{ There's no place like... | Number picker | %prompt%}  
If{ %prompt% = home | Message{Sure is!} | Message{For some yes...} }
```

This will display an Input box asking the user to choose a place. Then do a comparison of the value of %prompt% against the text "home". If the condition is true, we show a message reading "Sure is!".



If you are comparing numeric values, the IF{} function's condition parameter supports the use of =, >, <, >=, <=, <> compare options. If the comparison is text based and is non numeric, then only = is allowed to be used to compare.

Let's compare some numbers to see how it can vary the outcome.

```
@var num1 = 30
@var num2 = 50
@var result

%num1% = InputBox{ Choose the first number to compare... | Number 1 | %num1% }
%num2% = InputBox{ Choose the second number to compare... | Number 2 | %num2% }

If{ %num1% >= %num2% | %result% = IS | %result% = IS NOT }

Message{ The first number %result% bigger than or equal to the second number. | There is the answer | 0 }
```

In this example, we declare 3 variables, 2 are numbers and the 3rd is to hold the result of our IF{} statement. We present the user with 2 separate Input boxes to choose a first and second number. We then compare that number to see if number 1 is bigger than or equal to the second number and display the appropriate answer.

Compare options

=	Compare if both values are equal
>	Compare if the first number is bigger than the second
<	Compare if the first number is smaller than the second
>=	Compare if the first number is bigger than or equal to the second
<=	Compare if the first number is smaller than or equal to the second
<>	Compare if both values do not equal the same value

Pointers

Pointers are used to tell your script where to execute next. They are great for having blocks of code that perform a specific task without needing to execute other commands in your script.

In order for Pointers to work, they are made up of 3 important parts; the declaration and location of the pointer, a GOTO function that tells your script to execute from the specified pointer and an EXIT statement which terminates your entire script when you want it to

To declare a Pointer, add colon character ":" to the start of your line and give it a name. The name must not have any spaces and must be unique. See example below:

:mypointer

To go to a pointer, use the GOTO function and add the pointer name, See example below:

GOTO mypointer

You can use the EXIT statement to terminate out of your script at any time. This will completely stop the script and any further executions in place. See example syntax:

EXIT

Let's look at some sample code of how to use Pointers to make your Macro code more dynamic and interactive.

```
@var var1 = 1
@var goagain = yes
:ptr3
%var1% = inputbox{choose a number,\1 or other? | Choose | %var1% }
if{%var1%=1 | goto ptr1 | goto ptr2}
message{ never going to happen } ;This line should never get hit because of the IF{} statement conditions above
:ptr1
message{ Pointer 1 was hit!!\I like unicorns\\\Do you? }
message{ I'm also a 2nd line in Pointer 1. Yay!! }
%goagain% = message{ Do you want to do this again? | Repeat | 0 | 1 }
if{ %goagain% = yes | goto ptr3 | exit }
exit
:ptr2
message{ Pointer 2 was hit! }
message{ I'm also a 2nd line in Pointer 2. Yay! }
message{ Let's do another one in Pointer 2 just because we can! }
%goagain% = message{ Do you want to do this again? | Repeat | 0 | 1 }
if{ %goagain% = YES | goto ptr3 }
exit
```

Internal Properties

Internal properties are internal settings to determine how the script is executed.

Timed property

The “Timed” property dictates whether or not Commander needs to wait for motion to complete before executing the next macro line or not.

Timed = True

This tells your script that you want to wait for all motions to stop before executing the next line.

Timed = False

This tells your script to immediately execute and send any commands to your controller as soon as it possibly can.

How to use “Timed”

```
Timed = True
;Do multiple lines of things...
;And more things...
Timed = False
```

Please Note: To keep things easy and simple for users, the setting of Timed can be:

True: True, Yes, Y, 1

False: False, No, N, 0(zero)

When should the “Timed” property be used?

Because the Nighthawk and other GRBL controllers process G-code data directly with a planned command buffer, many commands may be executed before the first command is actually processed motion wise. If you are planning on using Internal variables such as #XMPOS# it is recommended to set timed = true on the last motion line before the request to access #XMPOS is needed. This will ensure all motions have stopped before recording that value.

Safety Features

With the use of Pointers, it can be easy to find yourself in an endless loop between a GOTO and a Pointer.

There is an automatic script kill function built in to DaveScript that if the number of total executions exceeds a very large number, it will automatically kill the script.

PRO TIP!

If you are unlucky enough to have caused an endless loop with a Message{} and InputBox{} function, you can simply hold down the [**ALT**] key on your keyboard and dismiss the pop up on your screen to stop the current script from running. This also works if you are in an endless loop and do not want to wait for it to time out.