Flexible MyCANIC Script Tool User Manual

www.eepod.com

Copyright EEPod LLC 2018-2025

Copyright EEPod LLC 2018-2025

Revision History

Revision	Release Date	Change Descriptions
1.01	Oct. 4, 2017	First draft.
1.02	Oct. 30, 2017	Numerous updates.
1.03	Dec. 18, 2017	Numerous updates.
1.04	Dec. 29, 2017	Added HLCD, PERIODIC and LOG commands. Added several example scripts.
1.05	Jan. 4, 2018	Added LED, ENDDO and INHALE/EXHALE commands. Numerous other updates.
1.06	June 6, 2018	Added DID MODIFY and WRITE operations. Added SECURITY command..
1.07	July10, 2018	Added XOR and equal to DID MODIFY operation.
1.08	Oct 9, 2018	Add BIT, HEX, and {} detail.
1.09	Oct. 29, 2018	Added more examples of HEX and BIT data types.
1.10	Dec. 6, 2018	Added GETTIME command and TIME data type.
1.11	Jan. 11, 2019	Added EXEC command.
1.12	Jan. 18, 2019	Added PROGNC command.
1.13	Feb. 26, 2019	Added PROGNR command.
1.14	Mar. 27, 2019	Added support for multiplier and offset in LCD commands.
1.15	Apr. 26, 2019	Added firmware update and licensed compiler instructions. Removed security from Exhale command.
1.16	Aug. 1, 2019	Added more installation and usage notes.
1.17	Sep. 8, 2019	Add multiplier and offset to LOG command
1.18	Dec. 11, 2019	Update FW_VERSION to add minor version check.
1.19	June 9, 2020	Added bitwise AND logic operator
1.20	Mar. 18, 2021	Added CAN_FD_HS and CAN_FD_MS protocols
1.21	Nov. 1, 2022	Added ability to set number of debug log file data bytes. Added ability to read voltage of pin 8 and pin12. Added ability to log DTCs.
1.23	Mar. 1, 2023	Added comment about license issues with PCs that are used with a docking station.

Copyright EEPod LLC 2018-2025

1.24	Mar. 16, 2023	Added IOT picture / updates. Added GOTO, REQUEST and SET_VALUE commands. Added LOGADD.EXE utility.
1.25	Mar. 20, 2023	Added P_TIME data type.
1.26	May 17, 2023	Added VERSION update for FSCR_DBG.XXX files.
1.27	Nov. 8, 2023	Added UDATA type.
1.28	Dec.16, 2023	Added numeric data variables and security level 0x61.
1.29	Jan. 29, 2024	Added RETURN from GOTO support.
1.30	Mar. 8, 2024	Changed to only use RETURN with new GOSUB command.
1.31	Mar. 18, 2024	Added support for logical operations with VAR (variables). Added new DAT_STATUS data type to check the status of a saved .DAT file.
1.32	May 23, 2024	Added new UPLOAD, MEM (modify), DOWNLOAD functions. For modifying ECU memory. Minor fixups.
1.33	Sep. 5, 2024	Changed order of functions in manual. Minor fixups.
1.34	Oct. 22, 2024	Added RTC, VOLTS, AMPS, IGN_LINE data types and REQ_FILE command support.
1.35	Nov. 19, 2024	Added FLOAT type.
1.36	Nov. 25, 2024	Update installation instructions.
1.37	Dec. 3, 2024	Added USB and ZPL commands. Added IOT data identifier type.
1.38	Jan. 17, 2025	Fixed remaining curly braces to straight braces in installation instructions.
1.39	Feb. 18, 2025	Added VEHICLE method to EXHALE command. Added DID2MEM command.
1.40	Feb. 26, 2025	Added FREQUEST command.
1.41	Mar. 14, 2025	Added CTRLREQ command.
1.42	Mar. 18, 2025	Added IFLOAT, IDATA and IUDATA types.
1.43	Apr. 28, 2025	Added FUNCTION command.

Copyright EEPod LLC 2018-2025

1 Introduction

The EEPod Flexible MyCANIC Script Tool is designed to create ECU reprogramming and diagnostic script files to be placed on the SD-Card of MyCANIC-IoT, MyCANIC-FD and MyCANIC tools running the EEPod FSCRIPT firmware. The following manual describes the supported commands.

1.1 MyCANIC / MyCANIC-FD /MyCANIC-IOT Firmware (FSCRIPT)

The script processing firmware (FSCRIPT) allows the user to select and run reprogramming script files as well as perform an number of other functions, including reading OBDII parameters, reading/clearing DTCs and running J2534 pass-thru applications. The latest FSCRIPT firmware is always available from the EEPod FTP server. For the MyCANIC (gold keypad), the firmware is updated with a Windows PC console mode application (e.g. FSCRIPTv5291.EXE). Simply connect the MyCANIC to power (OBDII cable) and USB to your PC, then run the application to update the firmware (it only takes about 15 seconds). For the MyCANIC-FD and MyCANIC-IOT (black keypad), connect it to your PC via USB and run the FW.EXE console mode application in the same directory that contains the APP.LDR file to update the firmware. Alternatively, you can copy the APP.LDR file to the SD-Card of the MyCANIC-FD/IOT and then power-cycle to update the firmware.

1.2 FSCRIPT Compiler Installation

The FSCRIPT compiler (FSCRIPT.EXE) is a console-mode Windows PC application that is used to compile script files for use on the MyCANIC or MyCANIC-FD/IOT and is available on the EEPod FTP server as an installation package (e.g. fscript_0.0.27_setup.exe). After running the setup program, the FSCRIPT program will be available on the Windows Start Menu. When you select FSCRIPT from the Start Menu, a console mode command prompt window will appear. Using this window, you can compile scripts using the fscript.exe compiler.

1.3 FSCRIPT Compiler License Setup

The compiler is licensed via the EEPod LLC cloud license server, so you must be connected to the internet when compiling script files. If you need a license or are having any issues with a current license, please send an email to support@eepod.com.

When you receive your license key/token email from the FSCRIPT license server, go to the **FSCRIPT console mode window** (located under your Start menu after installation) and run the following command, but replacing the items in blue with your email, license key/token and company proxy server IP/port address information. Make sure to use the double quotes around the email, license token and proxy server IP/port information.

Note: Use straight quotes (**""**), not curly quotes (**“”**). The curly quotes do not work on some PCs.

For example:

- **Incorrect (Curly Quotes):** **“**example@example.com**”**
- **Correct (Straight Quotes):** **"**example@example.com**"**

Corporate FSCRIPT users that need to use a company proxy server for external internet access:
fscript -l -e "myname@companyname.com" -t "MY1CUSTOM2TOKENXYZABC" -x "123.45.67.89:80"
NOTE: For North America Ford employees, the proxy server IP address is: "19.12.80.237:83"
For All Other FSCRIPT Users:
fscript -l -e "myname@companyname.com"-t "MY1CUSTOM2TOKENXYZABC"

NOTE: Since the license uses the network hardware ID as part of the license verification, PCs (laptops) that are used with and without a docking station may experience issues when trying to use in both cases. Please pick the configuration you would like to use when compiling scripts and enable your license in that configuration.

After successfully installing your license, you will be able to use the FSCRIPT compiler whenever you are connected to the internet.

1.4 Automatically Script Run on Power-up

You can automatically run any script file on power-up by renaming the script file to AUTOEXEC.FSF. It is also possible to set the name of the script file that will be run by placing the script file name (e.g. PROGRAM.FSF) in a text file named FSCRIPT.INI. This is helpful in production environments as it will allow for several script files on the SD-Card, but only allow the user to pick the correct one.

1.5 FSCRIPT Compiler Command Line Options

For a list of FSCRIPT compiler command line options, just type “fscript” in the command prompt window and see the following:
FSCRIPT Compiler Version 0.0.4
Usage: fscript <options> <script file>
-h, --help Help
-a, --aes AES encrypt program binary
-v, --verbose Increase debug verbosity
-o, --outfile Set output filename
-l, --license Activate license
-t, --token Set license token
-e, --email Set license email
-x, --proxy Set HTTP proxy server

1.6 FSCRIPT Compiler Example

For most cases, you will want to set the output filename and turn on the AES encryption with a command similar to the following:

fscript -a -osample.fsf sample.scr
NOTE: If the output file is not specified, the default output file is a.out.

Copyright EEPod LLC 2018-2025

2 Reprogramming and Diagnostic Script File Commands

Script files are plain ASCII text files that get compiled into an encrypted binary file that is placed on the SD-Card of the MyCANIC or MyCANIC-FD/IOT. The following describes the commands and features of the scripting language. Note that there are several limitations to the scripting language (e.g. allows only one diagnostic filter and one periodic message at a time) of which the user must be aware when creating new script files.

2.1 Comment Line

Place a comment line in the script file.
Format:
; <comment>
Parameters:
None
Examples:
; This is a comment

2.2 FW_VERSION Command

Add a check in the script of the major and minor firmware version numbers. If the FW_VERSION operation fails (i.e. the MyCANIC/MyCANIC-FD/MyCANIC-IOT firmware major version does not match the command version or the firmware minor version is less than the command version), the script will be terminated.
Format:
FW_VERSION <Version Number>
Parameters:
Version Number - Version number of the firmware.
Examples:
Both examples below would check to make sure the major firmware version is fifty. The second example would also make sure the minor version is at least 11.
FW_VERSION 50
FW_VERSION 50.11

2.3 VERSION Command

Assign a version string to the script file for validation purposes. Note that if the version string has the characters “DBG” anywhere, it will automatically enable the debug log (FSCR_DBG.LOG) feature. If there is a number after the “DBG” characters, it will be used for the maximum number of message data bytes in the debug log. If the VERSION command has the “DBG” characters and is used multiple times in a script,

Copyright EEPod LLC 2018-2025

it will save the FSCR_DBG.LOG file with the next available file extension (FSCR_DBG.000 to FSCR_DBG.999).

Format:

VERSION “<Version String>”

Parameters:

Version String - Any string of text/numbers to describe the script version.

Examples:

VERSION “TEST_v23.125”

VERSION “PROG_1.23DBG12”

2.4 DELAY Command

Delay a number of milliseconds.

Format:

DELAY <Milliseconds>

Parameters:

Milliseconds – Number of milliseconds to delay (must be between 1 and 1,000,000,000).

Examples:

; Delay 100 milliseconds

DELAY 100

2.5 EXIT Command

EXIT the script file.

Format:

EXIT

Parameters:

None

Examples:

EXIT

2.6 GETKEY Command

Check for a key press. Key is stored in the special data value identifier ‘KEY’ for use in logic comparisons. (To refer to special data identifiers click here)

Format:

GETKEY

Parameters:

None

Examples:

GETKEY

Copyright EEPod LLC 2018-2025

2.7 WAITKEY Command

Wait for the user to press a key. Key is stored in a special value KEY. (To refer to special data identifiers click here)

Format:

WAITKEY <Key Type>

Parameters:

Key Type – ENTER, ESCAPE, UP, DOWN, LEFT, RIGHT or ANY

Examples:

```
WAITKEY ANY
WAITKEY ENTER
WAITKEY ESCAPE
WAITKEY UP
WAITKEY DOWN
WAITKEY LEFT
WAITKEY RIGHT
```

2.8 GETTIME Command

Get the current tick count.. Tick count is stored in the special data value identifier ‘TIME’ for use in logic comparisons. (To refer to special data identifiers click here)

Format:

GETTIME

Parameters:

None

Examples:

```
GETTIME
```

2.9 GOTO Command

Goto a label in the script. The label name must end with a colon ‘:’ character.

Format:

GOTO <label>

Parameters:

Label name ending in a colon character

Examples:

```
IF VBAT < 10000
  GOTO VBAT_LOW:
ENDIF
LED GREEN ON
WAITKEY ANY
```

Copyright EEPod LLC 2018-2025

```
EXIT
VBAT_LOW:
LED RED ON
```

2.10 LCD / HLCD Command

Display characters on the LCD screen with or without highlighting. (To refer to special data identifiers click here)

Format:

LCD <Row> <Column> <ASCII text, DATA, HEX, TEXT or VBAT> <Multiplier> <Offset>
HLCD <Row> <Column> <ASCII text, DATA, HEX, TEXT or VBAT> <Multiplier> <Offset>

Parameters:

Row – 1 to 8
Column – 1 to 16
ASCII Text – Any string of printable ASCII characters, encapsulated in double quotes. An empty string (“”) will clear the LCD screen from the row/column specified to the end of the screen.
DATA – Display signed numeric data from a previous READ/REQUEST command.
IDATA – Display signed numeric data (little endian) from a previous READ/REQUEST command.
UDATA – Display unsigned numeric data from a previous READ/REQUEST command.
IUDATA – Display unsigned numeric data (little endian) from a previous READ/REQUEST command.
FLOAT - Display floating point value from a previous READ/REQUEST command.
IFLOAT - Display floating point value (little endian) from a previous READ/REQUEST command.
HEX - Display hexadecimal data bytes from a previous READ/REQUEST command.
FLOAT - Display numeric data from a previous READ command as floating point with a multiplier and offset.
TEXT – Display data from a previous READ/REQUEST command as ASCII text.
VBAT – Display the battery voltage.
Multiplier - Optional floating point multiplier value
Offset - Optional floating point offset value

Examples:

```
; Display a string of characters
LCD 1 1 “Any text”
; Clear everything but the first line
LCD 2 1 “”
; Display the first byte of a CAN / DIAG response message
LCD 2 4 DATA[0, 1]
; Display the first byte of a CAN / DIAG response message in hexadecimal
LCD 2 4 HEX[0, 1]
; Display the first four bytes of a CAN / DIAG response message as a 32-bit value in hexadecimal
LCD 2 4 HEX[0, 4]
; Display the highlighted text in a CAN / DIAG response message, starting at the fifth byte and up to 8 characters long
HLCD 3 0 TEXT[4, 8]
```

Copyright EEPod LLC 2018-2025

```
; Display the battery voltage using highlighted text.
HLCD 4 0 VBAT
; Display an integer value as floating point number with a multiplier of 0.001 and offset of 0.000
LCD 3 0 DATA[7,2] 0.001 0.000
; Display the 4 bytes of CAN / DIAG response message as 32 bit float value
LCD 3 1 FLOAT[7,4]
```

2.11 LOG Command

Create and add to an ASCII text CSV (comma-separated variable) log file. (To refer to special data identifiers click here)

Format:

LOG <Type> <ASCII text, DATA, HEX, FLOAT, TEXT or VBAT> <Multiplier> <Offset>

Parameters:

Type – FILE (creates or opens log file). *NOTE: Only one FILE type command is allowed per script file. Any additional FILE type commands will be ignored.*
NEW (starts a new record with the first field)
ADD (adds another field to the record)

Text – Any string of printable ASCII characters, encapsulated in double quotes.
DATA – Log signed numeric data from a previous READ/REQUEST command.
IDATA – Log signed numeric data (little endian) from a previous READ/REQUEST command.
UDATA – Log unsigned numeric data from a previous READ/REQUEST command.
IUDATA – Log unsigned numeric data (little endian) from a previous READ/REQUEST command.
FLOAT - Log floating point value from a previous READ/REQUEST command.
IFLOAT - Log floating point value (little endian) from a previous READ/REQUEST command.
HEX – Log hexadecimal data from a previous READ/REQUEST command.
DTC - Log DTC information for current DIAG ECU.
TEXT – Log data from a previous READ/REQUEST command as ASCII text.
VBAT – Log the battery voltage.
AMPS - Log the amperage measurement from the FlexStation.
Multiplier - Optional floating point multiplier value
Offset - Optional floating point offset value

Examples:

```
; Start a new log file named VIN.LOG
LOG FILE “VIN.LOG”
; Start a new record in the log file with the first field as the text from a CAN / DIAG response message
LOG NEW TEXT[7,17]
; Add a field to the current log record with the data from a CAN / DIAG response message
LOG ADD DATA[7,2]
; Add a field to the current log record with the vehicle battery voltage.
LOG ADD VBAT
; Add a field to the current log record with a value as floating point number with a multiplier of 0.01 and offset of 1.000
```

Copyright EEPod LLC 2018-2025

```
LOG ADD DATA[7,2] 0.01 1.000
; Log the 4 bytes of CAN / DIAG response message as 32 bit float value
LCD 3 1 FLOAT[7,4]
```

2.12 READ Command

Read a diagnostic response or CAN message. The data is placed in the DATA/FLOAT/HEX/TEXT buffer for logic comparisons.

Format:

READ <Type> <Data>

Parameters:

Type = SID (Service ID).

Type = MSG (Message ID) followed by CAN ID value (11 or 29 bit).

Type = USB

Examples:

```
; Read the last response to a diagnostic WRITE of a SID request
READ SID
; Read the last received CAN message with an ID of 0x123
READ MSG 0x123
; Read command string from USB
READ USB
```

2.13 WRITE Command

Write a diagnostic command or CAN message.

Format:

WRITE <Type> <Data>

Parameters:

Type = SID (Service ID) followed by data bytes (e.g. PID, Routine, etc.)

Type = MSG (Message ID) followed by CAN ID value (11 or 29 bit) and data bytes.

Type = USB

Examples:

```
; Send a SID request for SID 0x31 with three data bytes to the currently selected diagnostic ECU ID
WRITE SID 0x31 0x06 0x11 0x22
; Send a CAN message with an ID of 0x123 and eight data bytes
WRITE MSG 0x123 0x11 0x22 0x33 0x44 0x55 0x66 0x77 0x88
; Send the last LOG string (from LOG NEW to last LOG ADD) to USB
WRITE USB
```

Copyright EEPod LLC 2018-2025

2.14 REQUEST / FREQUEST Commands

Request a diagnostic service and wait for the response. The response data is placed in the DATA/FLOAT/HEX/TEXT buffer for logic comparisons. This command is the equivalent of the WRITE SID, followed by READ SID. The difference between REQUEST and FREQUEST (fast request) is REQUEST will attempt two retries of the request built-in if there is no response or a negative response, FREQUEST will only wait 500msec for a response.

Format:

REQUEST <Data>

FREQEUST <Data>

Parameters:

Data = Request message service ID and data bytes.

Examples:

```
; Request the F188 part number and get the response.
REQUEST 0x22 0xF1 0x88
; Check for an exact part number match
IF TEXT[7,15]=”PC3C-14C199-AAB”
…
ENDIF
```

2.15 CTRLREQ Command

Request a diagnostic control service using a variable as the data and wait for the response. The response data is placed in the DATA/FLOAT/HEX/TEXT buffer for logic comparisons. This command is targeted for use with service 0x31 control routines and service 0x2F output control. The last byte of the Data tells which variable/VAR index (upper nibble) to use for the data and how many bytes of data (lower nibble)

Format:

CTRLREQ <Data>

Parameters:

Data = Request message service ID and data bytes, with the last byte indicating the variable and number of bytes to use

Examples:

```
; Setup a variable to use
VAR[3] = 0x1B8
; Loop to increment the output control channel value by 1 every 100msec
DO
  ; Set output control channel 0x1FEE as a two-byte value using VAR[3]
  CTRLREQ 0x2F 0x1F 0xEE 0x32
  VAR[3] = VAR[3] + 1
  DELAY 100
WHILE VAR[3] < 0x200
```

Copyright EEPod LLC 2018-2025

2.16 SET_VALUE Command

The SET_VALUE command can be used to set several different features in the MyCANIC.

Format:

Security <Type> <Value>

Parameters:

Type = TERMINATION - CAN bus termination for current network. 0 = No termination, 1 = 120ohm termination.

Type = INTERFRAME - CAN interframe spacing. Value in microseconds.

Type = STMIN_TX - ISO15765 STMIN override. Value in milliseconds

Type = BS_TX - ISO15765 BS override. Value in bytes

Type = VPROG - Programmable output voltage (pin 13 of DB15HD connector for MyCANIC, TIP on TRRS connector for MyCANIC-FD and MyCANIC-IOT). Value in millivolts.

Value (see Types above)

Examples:

```
; Turn on CAN_HS network termination to 120ohm
NET CAN_HS 11 500000
SET_VALUE TERMINATION 1
; Set interframe spacing to 100usec
SET_VALUE INTERFRAME 100
; Set the output voltage to 3.5V
SET_VALUE VPROG 3500
; ***NOTE: FlexStation-IOT only***. Set the Vout voltage to 12VDC (0-24000mV range).
SET_VALUE VOLTS 12000
; ***NOTE: FlexStation-IOT only***. Set the Ignition line (Vout-SW) off (0V)
SET_VALUE IGN_LINE 0
; ***NOTE: FlexStation-IOT only.*** Set the Ignition line (Vout-SW) on (same voltage as Vout)
SET_VALUE IGN_LINE 1
; ***NOTE: FlexStation-IOT only.*** Calibrate the current measurement with no load attached
SET_VALUE AMPS 0
```

2.17 NET Command

Initialize a CAN network.

Format:

NET <CAN network> <# CAN ID Bits> <CAN Bit Rate>

Parameters:

CAN Network – CAN_HS (pins 6 & 14), CAN_FD_HS (pins 6 & 14), CAN_MS (pins 3 & 11) or CAN_FD_MS (pins 3 & 11)

CAN ID Bits – 11 or 29

CAN Bit Rate – 500000 or 125000

Examples:

```
; Initialize the HSCAN network (OBDII J1962 pins 6 & 14) to use 11-bit identifiers and 500K baud rate
NET CAN_HS 11 500000
; Initialize the FD HSCAN network (OBDII J1962 pins 6 & 14) to use 11-bit identifiers and 500K baud rate
NET CAN_FD_HS 11 500000
; Initialize the MSCAN network (OBDII J1962 pins 3 & 11) to use 29-bit identifiers and 125K baud rate
NET CAN_MS 29 125000
```

2.18 DIAG Command

Initialize ISO15765 diagnostic flow control filter for an ECU. If a different DIAG command were previously run in the script file, this command would supersede the previous command and replace the filter. Note that the DIAG command needs to be preceded by a NET command to select the CAN network to use.

Format:

DIAG <Type> <Request ID> <Response ID>

Parameters:

Type – ISO14229 or KWP2000
Request ID – 11-bit or 29-bit CAN ID value
Response ID – 11-bit or 29-bit CAN ID value

Examples:

```
; Setup 11-bit ISO14229/ISO15765 diagnostic flow control filter for powertrain ECU
DIAG ISO14229 0x7E0 0x7E8
; Setup 29-bit ISO14229/ISO15765 diagnostic flow control filter for powertrain ECU
DIAG ISO14229 0x18DB10F1 0x18DAF110
; Setup 11-bit KWP2000/ISO15765 diagnostic flow control filter for body module ECU
DIAG KWP2000 0x730 0x738
```

2.19 REQ_FILE Command

The **REQ_FILE** command allows setting DID (Data Identifier) bytes by referencing an external file containing byte configurations, simplifying scripts when handling large byte modifications.

Syntax:

REQ_FILE <Filename / DID>

Filename: The external file that contains the byte data for the request or a DID value that will be used to determine the filename.

Example:

```
REQ_FILE “XYZ.REQ”
DELAY 50
```

In this example, the **XYZ.REQ** file contains pre-configured request byte data. The data in the file is similar

to the data you would use in a REQUEST command, but without the 255 byte limit and no “0x” prefix on each byte. Below is an example of a DID DE01 write request that would be the contents of the XYZ.REQ file:

2E DE 01 00 11 22 33 44 55 66 77 88 99 00

When using a DID to specify the file, the DID must return a valid ASCII string to be used for the request data file name. For instance, if the 0xF188 returns a string of “ABC-AA”, that is the name of the data file that will be used.

2.20 DTCVAL (Diagnostic Trouble Code Value) command

The DTCVAL feature in the FScript language allows users to retrieve and compare DTCs stored in the vehicle's ECUs.

Syntax:

```
IF DTCVAL = {byte1, byte2, ..., byteN}
// Statements executed when the specified DTC is found
ELSE
// Statements executed when the specified DTC is not found
```

Example:

```
// Example script to check for a specific DTC and display a message
IF DTCVAL = {0x07, 0x15, 0x00, 0xAF}
  LCD 7 1 "DTC 0715 Found!"
ELSE
  LCD 7 1 "DTC not found!"
```

Use Case: The DTCVAL feature is used in vehicle diagnostic scripts where specific actions need to be taken based on the presence or absence of certain DTCs. It provides a convenient way to handle diagnostic scenarios in FScript.

2.21 DAT_… Commands

Read either the vehicle VIN (read with OBDII mode 9) or a diagnostic DID and create a VIN or ECU specific data file for storing ECU configuration values. On a MyCANIC, due to 8.3 filename limitations, the file created will be the last eight characters of the VIN or DID with a .DAT extension. For the MyCANIC-FD/IOT, the filename will be the full VIN plus CAN ID or DID value with a .DAT extension.

Format:

DAT_INIT <Type> <Data>
DAT_CREATE <Type> <Data>
DAT_CHECK <Type> <Data>
DAT_DELETE <Type> <Data>

Copyright EEPod LLC 2018-2025

Parameters:
Type = VIN
Type = DID followed by DID value
Examples:

```
; Initialize and create a data file based on the vehicle VIN
DAT_INIT VIN
; Check if the data file already is created and complete
IF DAT_STATUS=2
  GOTO USE_EXISTING_DATA:
ENDIF
; Check if the data file is created but not complete
IF DAT_STATUS=1
  DAT_DELETE VIN
ENDIF
DAT_CREATE VIN
; Save DID information in data file
DID SAVE 0xDE00
; Finish the data file and mark it as complete
DAT_CHECK VIN

USE_EXISTING_DATA:

; Continue script…

; Delete data file before exit
DAT_DELETE VIN
EXIT
```

or

```
; Initialize and create a data file base on the ECU serial number
DAT_INIT DID 0xF18C
; Check if the data file already is created and complete
IF DAT_STATUS=2
  GOTO USE_EXISTING_DATA:
ENDIF
; Check if the data file is created but not complete
IF DAT_STATUS=1
  DAT_DELETE DID 0xF18C
ENDIF
DAT_CREATE DID 0xF18C
; Save DID information in data file
```

Copyright EEPod LLC 2018-2025

```
DID SAVE 0xDE00
; Finish the data file and mark it as complete
DAT_CHECK DID 0xF18C

USE_EXISTING_DATA:
; Continue script…
; Delete data file before exit
DAT_DELETE DID 0xF18C
EXIT
```

2.22 VIN_VERIFY Command

Read the vehicle VIN (read with OBDII mode 9) and verify it exists in the VIN list file on the SD-Card. If the VIN is in the list, the script will continue. If the VIN is not in the list, an error message will display and the script will stop and return to the main menu.

Format:

VIN_VERIFY “<VIN File>”

Parameters:

VIN File - Filename of VIN list file on the SD-Card. If the last letter in the filename is an ‘E’ (e.g. VIN.CSE), the file is assumed to be encrypted. Otherwise, it is in ASCII CSV format.

Examples:

VIN_VERIFY “VIN.CSV”
VIN_VERIFY “VIN.CSE”

2.23 DID Command

Command for saving, recalling. comparing, modifying and writing DID values using services 22 and 2E. If a DID COMPARE operation fails (i.e. the saved DID value did not match the current DID value), the script will be terminated.

Format:

DID <Operation> <DID Number> <Data Byte Offset> < Number of Bytes> <Operator> <Operand>

Parameters:

Operation – SAVE, RECALL, COMPARE, MODIFY or WRITE
DID Number – 16-bit DID value in hexadecimal notation
Data Byte Offset - Only for MODIFY operation. Offset into DID data, starting at byte zero (0).
Number of Bytes - Only for MODIFY operation. Number of bytes, starting at the Data Byte Offset.
Operator - Only for MODIFY operation. Can be ‘+’(add), ‘-’(subtract), ‘*’(multiply), ‘/’(divide) , ‘&’(bitwise AND), ‘|’(bitwise OR), ‘^’(bitwise XOR) or ‘=’(equal).
Operand - Only for MODIFY operation. Data value to be used with Operator.

Examples:

```
; Read DID F188 and save in DAT file
DID SAVE 0xF188
; Recall saved DID F188 from DAT file (put in DATA[ ] array for IF/WHILE commands)
```

```
DID RECALL 0xF188
; Read DID F188 and compare to saved DID F188 in DAT file
DID COMPARE 0xF188
; Read DID DE01 and save in DAT file
DID SAVE 0xDE01
; Remove the LSB of the third data byte of DID DE01 (read-modify-write operation)
DID MODIFY 0xDE01 2 1 & 0xFE
; Write DID 0xDE01 from saved DAT file
DID WRITE 0xDE01
```

2.24 DID2MEM Command

Specialized command for recalling a saved DID value from the current DAT file and writing to a memory address with a specific length. This command is useful when reprogramming causes part and/or serial numbers to be erased/lost.

Format:

DID2MEM <DID Number> <Address> < Number of Bytes>

Parameters:

DID Number – 16-bit DID value in hexadecimal notation

Address - Memory address for service 0x3D write operation.

Number of Bytes - Number of bytes for write operation. If length of saved DID is less than the number of bytes specified, the remaining bytes will be filled with zeroes.

Examples:

```
; Recall DID 0xF18C and write to memory address 0xA00E0000 with a length of 24 bytes
DID2MEM 0xF18C 0xA00E0000 24
```

2.25 GOSUB/RETURN Commands

Goto a label in the script. The label name must end with a colon ‘:’ character. See the RETURN command for returning from a sub-routine.

Format:

GOSUB <label>

RETURN

Parameters:

GOSUB: Label name ending in a colon character

RETURN: None

Examples:

```
DO
  LCD 2 1 ""
  DELAY 1000
  LCD 2 1 "Press Enter for"
  LCD 3 1 "subroutines or *"
  LCD 4 1 "to exit."
```

```
  IF KEY=ENTER
    GOSUB SUB1:
  ENDIF
  WAITKEY ANY
WHILE KEY!ESCAPE
EXIT

SUB1:
LCD 5 1 "SUB 1"
DELAY 500
GOSUB SUB2:
RETURN

SUB2:
LCD 6 1 "SUB 2"
DELAY 500
GOSUB SUB3:
RETURN

SUB3:
LCD 7 1 "SUB 3"
DELAY 500
RETURN
```

2.26 RETURN Command

The RETURN command is for returning from the last GOSUB command (up to 8 levels deep).
Format:
RETURN
Parameters:
None

2.27 DO / WHILE / ENDDO Command

Perform DO / WHILE / ENDDO logic. Special identifiers like ‘DATA’, ‘FLOAT’, ‘TEXT’, ‘VBAT’ and KEY can be used in logic comparisons. (To refer to special data identifiers click here)
Format:
DO
<set of commands>
WHILE <logic comparison>
Parameters:
None
Examples:

```
; Display the battery voltage until a key is pressed or goes above 12.000VDC
DO
   ; Display the battery voltage on line 4
   LCD 4 1 VBAT
   ; Delay between readings to make it readable
   DELAY 200
   ; Check for a key press
   GETKEY
   IF KEY=ESCAPE
      ; Stop the loop if the Escape key was pressed
      ENDDO
   ENDIF
; Go back to beginning of DO loop if the battery is below 12.000VDC
WHILE VBAT<12000
; Display the final battery voltage reading on line 4
LCD 4 1 VBAT
```

2.28 IF / ELSE / ENDIF Command

Perform IF / ELSE logic. Special identifiers like ‘DATA’, ‘FLOAT’, ‘TEXT’, ‘VBAT’ and ‘KEY’ can be used in logic comparisons. Nesting of IF/ELSE/ENDIF commands is only allowed if the ENDIF commands are sequential (see example below). (To refer to special data identifiers click here)

Format:

```
IF <logic comparison>
<set of commands>
ELSE
<set of commands>
ENDIF
```

Parameters:

None

Examples:

```
; Voltage check loop
DO
   ; Display the battery voltage on line 4
   LCD 4 1 VBAT
   IF VBAT>13000
      ; If the voltage is above 13.000VDC, turn off the green LED and start blinking the red LED
      LED GREEN OFF
      LED RED TOGGLE
   ELSE
      IF VBAT<10000
         ; If the voltage is below 10.000VDC, turn off the green LED and turn on the red LED
         LED GREEN OFF
```

```
        LED RED ON
      ELSE
        ; If the voltage is between 10.000-13.000VDC, turn off the red LED and turn on the green LED
        LED GREEN ON
        LED RED OFF
      ENDIF
    ENDIF
    ; Delay a short while so the display of the voltage does not update too fast
    DELAY 500
    GETKEY
; Stay in the loop until the escape key is pressed
WHILE KEY!=ESCAPE
```

2.29 LED Command

Turn on or off the green and red LEDs. For the original MyCANIC, the LEDs are external. For the MyCANIC-FD, the LEDs are in the keypad.

Format:

LED <Selection> <State>

Parameters:

Selection – GREEN or RED

State – ON, OFF or TOGGLE

Examples:

```
; Turn on the red LED
LED RED ON
; Toggle the green LED
LED GREEN TOGGLE
```

2.30 PERIODIC Command

Setup a periodic CAN message. If a previous PERIODIC command occurred in the script file, this message from this command would replace the previous one.

Format:

PERIODIC <Type> <Timing> <Data>

Parameters:

Type = SID (Service ID) followed by Timing value and data bytes (e.g. PID, Routine, etc.)

Type = MSG (Message ID) followed by Timing value, CAN ID value (11 or 29 bit) and data bytes.

Timing = Time between messages, in milliseconds. A value of zero (0) will stop a previously started periodic message.

Examples:

```
; Start a periodic diagnostic request (tester present message) every 5000 milliseconds
PERIODIC SID 5000 0x3E 0x02
; Start a periodic CAN message every 1000 milliseconds with a CAN ID of 0x123 and eight data bytes
```

Copyright EEPod LLC 2018-2025

```
PERIODIC MSG 1000 0x123 0x11 0x22 0x33 0x44 0x55 0x66 0x77 0x88
; Stop the current periodic message
PERIODIC MSG 0
```

2.31 PROG Command

Download a VBF file to an ECU. Note that the LCD lines 3 thru 7 are used during programming to display progress/status.

Format:

PROG “<VBF Filename>” <Security Bytes>

Parameters:

VBF Filename – Any valid VBF file, encapsulated in double quotes.

Security Bytes – 5-byte or 12-byte security seed, only used with SBL files.

Examples:

```
; Download a VBF file (SBL) with a 5-byte security code to the currently selected ECU diagnostic ID
PROG “JL3A-14C273-EA.VBF” 0x1122334455
; Download a VBF file (EXE) to the currently selected ECU diagnostic ID
PROG “JL3A-14C204-BDE.VBF”
```

2.32 PROGNC Command

Download a VBF file to an ECU without a check routine after the download.

Format:

PROGNC “<VBF Filename>”

Parameters:

VBF Filename – Any valid VBF file, encapsulated in double quotes.

Examples:

```
PROGNC “JL3A-14C204-DA.VBF”
```

2.33 PROGNR Command

Download a VBF file to an ECU without a reset before downloading the SBL.

Format:

PROGNR “<VBF Filename>” <Security Bytes>

Parameters:

VBF Filename – Any valid VBF file, encapsulated in double quotes.

Examples:

```
PROGNR “JL3A-14C204-DA.VBF” 0x1122334455
```

2.34 SECURITY Command

The SECURITY command is for entering a security level.

Copyright EEPod LLC 2018-2025

Format:
SECURITY <Level> <Security Bytes>
Parameters:
Level – 1, 3, 5 or 0x61
Security Bytes - Security bytes (a.k.a. fixed bytes)
Examples:
; Enter security level 3 using a 5-byte security value
SECURITY 3 0x1122334455
; Enter security level 0x61 using a 12-byte security value
SECURITY 0x61 0x112233445566778899001122

2.35 EXEC Command

Set the script filename of the next script to execute upon EXIT of the current script.
Format:
EXEC <Filename>
Parameters:
Filename - Next script file to execute.
Examples:
EXEC “NEXTTEST.FSF”
EXIT

2.36 FUNCTION Command

Run internal MyCANIC functions.
Format:
FUNCTION “<Function Name> <Parameter 1> <Parameter 2> <Parameter 3>”
Parameters:
A single string containing the function name and up to 3 parameters separated by spaces.

Note: The following supported internal functions are for the FlexStation-IOT only:
RAMP_VOLTAGE <starting mV> <ending mV> <microseconds>
CI230
CI260A <microseconds>
CI260B <microseconds>
CI260C <microseconds>
CI260D <microseconds>

Examples:
; Ramp the power supply voltage from 12.000V to 13.500V over 0.5 seconds
FUNCTION “RAMP_VOLTAGE 12000 13500 500000”
; Run the CI230 conducted immunity test (sawtooth 8-10V waveform)
FUNCTION “CI230”

Copyright EEPod LLC 2018-2025

; Run the CI260 waveform A conducted immunity pattern for 500 microseconds
FUNCTION “CI260A 500”

2.37 INHALE / EXHALE Commands

The INHALE / EXHALE commands are for saving and restoring vehicle-specific data in an ECU.
Format:
INHALE <Method>
EXHALE <Method>
Parameters:
Method – A (DID DE00-DExx scan), B (Routine 020A), VEHICLE (EXHALE only)
Examples:
; Inhale ECU data using method A
INHALE A
; Exhale ECU data from method A
EXHALE A
; Exhale all VIN-specific DAT files to vehicle
EXHALE VEHICLE

Note: Need to use DAT_INIT and DAT_CREATE to use Inhale/Exhale commands for Methods A or B (only DAT_INIT required for Method VEHICLE). Refer here

2.38 UPLOAD / MEM / DOWNLOAD Commands

The UPLOAD / MEM / DOWNLOAD commands are meant to be used in sequence to modify an area of memory in a module. Since these operations normally require the module to be in programming mode, they are usually preceded by a programming session request and PROG command for the SBL.
Format:
UPLOAD <Address> <Length>
MEM <Offset> <Operator> <Operand>
DOWNLOAD <Address> <Length>
Parameters:
Address - Memory address in the module.
Length - Number of bytes to upload / download.
Offset - Byte offset into the data downloaded.
Operator - Can be ‘+’(add), ‘-’(subtract), ‘*’(multiply), ‘/’(divide) , ‘&’(bitwise AND), ‘|’(bitwise OR), ‘^’(bitwise XOR) or ‘=’(equal).
Operand - The data value to be used with the operator. Can be hexadecimal or text up to 255 bytes.
Examples:
; Connect to the appropriate network
NET CAN_HS 11 500000
; Periodic tester present message to stay in programming session
PERIODIC MSG 2000 0x7DF 0x02 0x3E 0x80 0x00 0x00 0x00 0x00 0x00

Copyright EEPod LLC 2018-2025

```
; Setup the diagnostic filter to the module
DIAG ISO14229 0x7E0 0x7E8
; Download the SBL without doing a reset at the beginning
PROGNR "ML3A-14C273-GA.VBF" 0x112233445566778899001122
; Upload 32 bytes starting at address 0x40008000
UPLOAD 0x40008000 0x20
; Modify the first 3 of the 32 bytes to enable bit 0
MEM 0x00000000 | 0x010101
; Modify the 4 bytes starting a offset 3 to boolean AND with 0xAA
MEM 0x00000003 & 0xAAAAAAAA
; Modify 1 byte at offset 7 to set equal to 0xFE
MEM 0x00000007 = 0xFE
; Modify 14 bytes at offset 8 to set equal to “PRCA-14C599-AC”
MEM 0x0000000 = “PRCA-14C599-AC”
; Erase the 32 byte area in the module and download the modified data
DOWNLOAD 0x40008000 0x20
```

2.39 ZPL Command

The ZPL command is used to send a Zebra Print Language command string to a Zebra compatible printer using the ISO9141/LIN line (@9600baud) of the MyCANIC and an EEPod ZPL printer interface cable. For a complete list of ZPL commands, refer to the list of supported commands for your particular printer and label size.

Format:

ZPL <Print String>

Parameters:

Print String - A string of static text that may have TEXT, HEX or DATA identifiers that will be substituted before the string is sent to the printer. The TEXT, HEX and DATA identifiers use the same format as used in LOG or LCD commands.

Examples:

```
; Label start command
ZPL “^XA”
; Read the F188 part number from a module
REQUEST 0x22 0xF1 0x88
; Add the part number to the label at a position of 50 dots from left edge and 60 dots from top with
; Font 0 with a height of 40 dots
ZPL “^FO50,60^A0,40^FDTEXT[7,24]^FS”
; Add the part number as a 3of9 barcode at a position of 40 dots from left edge and 120 dots from top with
; a barcode height of 60 dots
ZPL “^FO40,120^BY3^BCN,60,,,,A^FDTEXT[7,24]^FS”
; Label finish command
```

Copyright EEPod LLC 2018-2025

ZPL”^XZ”

Copyright EEPod LLC 2018-2025

3 Script File Logic Comparison

3.1 Operators For Use In Logic Comparisons

All comparison operators are a single ASCII character as described below.

Equal Operator: **=**

Not Equal Operator: **!** or **!=**

Less Than Operator: **<**

Greater Than Operator: **>**

Bitwise AND Operator: **&**

Copyright EEPod LLC 2018-2025

3.2 Data Identifier Names For Use In Logic Comparisons, LCD and LOG Functions

The following data identifiers can be used for logic comparisons in the script file:
BIT[x,y] = Bit value from a diagnostic response message or CAN broadcast message, where ‘x’ is the bit offset into the data and ‘y’ is the number of bits and endian direction (-32 to +32, excluding 0).
CH1 = Input voltage (pin 8 of DB15HD connector on MyCANIC, RING1 of TRRS connector on MyCANIC-FD and MyCANIC-IOT) in millivolts.
CH2 = Input voltage (pin 12 of DB15HD connector on MyCANIC, RING2 of TRRS connector on MyCANIC-FD and MyCANIC-IOT) in millivolts.
DAT_STATUS = Data file status (0=Non-existent, 1=Exists but not complete, 2=Exists and complete
DATA[x,y] = Signed data value from a diagnostic response message or CAN broadcast message, where ‘x’ is the offset into the data and ‘y’ is the size of the data in bytes (1-4).
IDATA[x,y] = Signed data value (little endian) from a diagnostic response message or CAN broadcast message, where ‘x’ is the offset into the data and ‘y’ is the size of the data in bytes (1-4).
UDATA[x,y] = Unsigned data value from a diagnostic response message or CAN broadcast message, where ‘x’ is the offset into the data and ‘y’ is the size of the data in bytes (1-4).
IUDATA[x,y] = Unsigned data value (little endian) from a diagnostic response message or CAN broadcast message, where ‘x’ is the offset into the data and ‘y’ is the size of the data in bytes (1-4).
DTC = List of DTCs when logging. Number of DTCs when used in an expression/comparison.
FLOAT[x,4] = 32-bit floating point value from a diagnostic response message or CAN broadcast message, where ‘x’ is the offset into the data and the size of the data is always 4 bytes.
IFLOAT[x,4] = 32-bit floating point value (little endian) from a diagnostic response message or CAN broadcast message, where ‘x’ is the offset into the data and the size of the data is always 4 bytes.
HEX[x,y] = Data bytes from a diagnostic response message or CAN broadcast message, where ‘x’ is the offset into the data and ‘y’ is the size of the data in bytes (1-16). **HEX[x,y]** can only be used with the Equal and Not Equal operators and will be compared byte-by-byte against the byte array on the other side of the logic expression. The ‘y’ parameter must match the length of the array.
KEY = Last key pressed.
P_TIME = Programming time (since start of programming) in milliseconds.
SERNUM = 8-digit MyCANIC-FD/IOT electronic serial number in text format.
TEXT[x,y] = Data from a diagnostic response message or CAN broadcast message, treated at ASCII text, where ‘x’ is the offset into the data and ‘y’ is the maximum length of the text.
TIME = Current tick count in milliseconds since last GETTIME command.
VBAT = Battery voltage (at OBDII J1962 connector) in millivolts.
{ a, b, c, ... } = Array of numbers used for comparisons against **HEX[x,y]**. The commas separating the array values are optional.
VAR[x] = A numeric data variable that can be used for comparisons and counters, where ‘x’ can be 0 thru 8.
VOLTS = ***NOTE: FlexStation-IOT only.*** This variable can be used to read the Vout voltage of a FlexStation in millivolts. When used with the SET_VALUE command it will set the Vout voltage.

Copyright EEPod LLC 2018-2025

AMPS = ***NOTE: FlexStation-IOT only.*** This variable can be used to read the current draw of the ECU(s) attached to a FlexStation in milliamps. When used with the SET_VALUE command, it will perform a zero calibration function with no load.
IGN_LINE = ***NOTE: FlexStation-IOT only***. This variable controls the ignition line state (Vout-SW on FlexStation), allowing you to turn it on or off. A value of 1 turns the ignition line on(save voltage as Vout), while a value of 0 turns it off (approximately 0VDC).
RTC = ***NOTE: Works only with MyCANIC-IOT connected to wireless network.*** Real Time Clock in GMT (format YYYY/MM/DD HH:MM.SS).
IOT = ***NOTE: Works only with MyCANIC-IOT connected to wireless network.*** Number of bytes in queue from IOT server.

Examples:

```
IF HEX[0,4] ={ 0xFF, 0x12, 0x55, 0x24 }
ENDIF
IF DATA[5,1] < -1
ENDIF
IF UDATA[6,1] > 130
IF BIT[48, -3] >= 6
ENDIF
IF FLOAT[7,4]<22971.35
ENDIF
```

```
; Variables example
VERSION "DBG"
HLCD 1 1 "VARIABLES v1.00"
LCD 2 1 SERNUM
FW_VERSION 52.92
LED RED OFF
LED GREEN OFF
GETTIME
VAR[0] = 0
DO
  VAR[0] = VAR[0] + 1
  IF VAR[0] > 10
    EXIT
  ENDIF
  LCD 3 12 TIME
  LCD 4 5 DAC
  LCD 5 5 CH1
  LCD 6 5 CH2
  LCD 7 5 VAR[0]
  DELAY 500
```

Copyright EEPod LLC 2018-2025

```
  GETKEY
WHILE KEY!ESCAPE
EXIT
```

4 Script File Compiler

After the engineer has created the script file, place the script file along with all referenced VBF files in the same directory and run the FSCRIPT.EXE command line script file compiler (e.g. FSCRIPT.EXE <Filename>). Upon successful completion, the output file will have the same base filename as the script file, but with a .FSF extension. Place the .FSF file on SD-Card of the MyCANIC or MyCANIC-FD when complete. Any errors while processing the script file will be noted as a response in the command window and the output (.FSF) file will not be created.

4.1 Example VIN and Module ID Logging Script

```
; Read, log and display the VIN and module ID
HLCD 1 1 "VIN & Module ID"
; Start the log file
LOG FILE “VIN.LOG”
; Initialize the CAN network
NET CAN_HS 11 500000
; Start the diagnostic filter
DIAG ISO14229 0x7E0 0x7E8
; Request the VIN using OBDII Mode $09 Inf $02
WRITE SID 0x09 0x02
; Get the response to the VIN request
READ SID
; Display and log the VIN in a new record
LCD 2 1 TEXT[7,17]
LOG NEW TEXT[7,17]
; Request the module ID using Service $22 and PID $F188
WRITE SID 0x22 0xF1 0x88
; Get the response to the module ID PID request
READ SID
; Display and add the module ID to the current log record
LCD 3 1 TEXT[7,16]
LOG ADD TEXT[7,16]
WAITKEY ENTER
```

Copyright EEPod LLC 2018-2025

4.2 Example Vehicle Battery Voltage Read And Compare Do-While Loop Script

```
; Read/display the vehicle battery voltage and wait for it to go above 13.0V
HLCD 1 1 " VBAT DO/WHILE  "
LCD 2 1 "VBAT < 13V"
DO
   LCD 4 1 VBAT
   GETKEY
   IF KEY=ESCAPE
      ENDDO
   ENDIF
   DELAY 200
WHILE VBAT  < 13000
LCD 4 1 VBAT
LCD 2 1 "VBAT >= 13V"
WAITKEY ENTER
```

4.3 Example Periodic Message Script

```
; Periodic message test
HLCD 1 1 "Periodic Msg Tst"
; Initialize the CAN network
NET CAN_HS 11 500000
; Start the diagnostic filter
DIAG ISO14229 0x7E0 0x7E8
LCD 2 1 "Sending MSG..."
; Send a message every 1000msec
PERIODIC MSG 1000 0x123 0 1 2 3 4 5 6 7
WAITKEY ENTER
; Stop the periodic message
PERIODIC MSG 0
LCD 2 1 "Sending SID..."
; Send a diagnostic request every 1000msec
PERIODIC SID 1000 0x22 0xF1 0x88
WAITKEY ENTER
; Stop the periodic message
PERIODIC MSG 0
```

Copyright EEPod LLC 2018-2025

4.4 Example Module Programming Script With Module ID Comparison Script

```
; Copperhead Programming Script
HLCD 1 1 "   Copperhead   "
; Start the log file
LOG FILE “VIN.LOG”
; Initialize the CAN network
NET CAN_HS 11 500000
; Start the diagnostic filter
DIAG ISO14229 0x7E0 0x7E8
; Request the VIN using OBDII Mode $09 Inf $02
WRITE SID 0x09 0x02
; Get the response to the VIN request
READ SID
; Start a new log record with the VIN
LOG NEW TEXT[7,17]
; Request the module ID using Service $22 and PID $F188
WRITE SID 0x22 0xF1 0x88
; Get the response to the module ID PID request
READ SID
; Add the current module ID to the log record
LOG ADD TEXT[7,15]
; Compare the module ID to determine what file(s) to program
IF TEXT[7,15]="BR3A-SEMAMM-GCO"
   PROG “BC3A-14C273-BD.VBF” 0x1122334455
   PROG “CR3A-14C204-ACC.VBF”
ENDIF
IF TEXT[7,15]="CR3A-14C204-ACC"
   PROG “BC3A-14C273-BD.VBF” 0x1122334455
   PROG “BR3A-SEMAMM-GCO.VBF”
ENDIF
; Reset the module and give it some time to be ready
WRITE SID 0x11 0x01
DELAY 1000
; Request the module ID using Service $22 and PID $F188
WRITE SID 0x22 0xF1 0x88
; Get the response to the module ID PID request
READ SID
; Add the new module ID to the log record
```

Copyright EEPod LLC 2018-2025

```
LOG ADD TEXT[7,15]
LCD 8 1 "Done!"
WAITKEY ENTER
```

4.5 Example Do-While Loop With Timeout Script

```
; Read the current time
GETTIME
; Wait 5 seconds or until the ESCAPE key is pressed
DO
   GETKEY
   IF KEY=ESCAPE
      ENDDO
   ENDIF
WHILE TIME  < 5000
WAITKEY ENTER
```

4.6 Example Variable Testing within Do-While Loop

```
; Testing variables
VERSION "VAR_TEST 1.0"
HLCD 1 1 "VAR_TEST 1.0"
LED RED OFF
LED GREEN OFF
; Initialize counters
VAR[0] = 0 ; Counter for the outer loop
VAR[1] = 0 ; Counter for the inner loop
; Display initial values of counters
LCD 2 1 "BEFORE ENTERING"
LCD 3 1 VAR[0]
LCD 3 5 VAR[1]
DELAY 2000
; Wait for a valid response
DO
  ; Increment Counter 0
  VAR[0] = VAR[0] + 1
  ; Display current state in the outer loop
  LCD 2 1 ""
  LCD 2 1 "INCREMENT VAR[0]"
  ; LCD 4 1 "COUNTER 0:"
  LCD 3 1 VAR[0]
  DELAY 500
  ; Check if Counter 0 is equal to 10
```

Copyright EEPod LLC 2018-2025

```
  IF VAR[0] = 10
    DO
      ; Display a message when inside the inner loop
      LCD 4 1 "INCREMENT VAR[0]"
      ; Increment Counter 0
      VAR[1] = VAR[1] + 1
      LCD 5 1 VAR[1]
      DELAY 500
    WHILE VAR[1] < 15
  ENDIF
WHILE VAR[1] < 10
; Display values of counters
LCD 6 1 "AFTER LOOP"
LCD 7 1 VAR[0]
LCD 7 5 VAR[1]
DELAY 2000
; Exit the script
EXIT
```

The added variable feature allows dynamic counter manipulation, offering flexibility in loop control.

4.7 Example snippet for variables within VOP for 80 Iterations

```
; Initialize variable for the number of Sawtooths
VAR[8] = 0

; Outer loop for multiple sawtooth ramps
DO
  LCD 2 1 "Sawtooth no:"
  LCD 2 15 VAR[8]
  VAR[1] = 0

  ; Inner loop for one sawtooth ramp
  DO
    LCD 3 1 "Sawtooth ITR:"
    LCD 3 15 VAR[1]
    LCD 4 1 "Idle @ 3500RPM  "

    ; Code for ramp execution, including oil pressure and RPM checks

    VAR[1] = VAR[1] + 1
  WHILE VAR[1] < 10

  ; Additional code after completing one sawtooth ramp
```

```
  ; (e.g., returning to idle, releasing overrides)

  VAR[8] = VAR[8] + 1
WHILE VAR[8] < 8
```

Without variables, this logic would involve manual code duplication for each sawtooth iteration, leading to a larger and error-prone script with repeated blocks for each ramp. The absence of variables would also require replicating code for setting oil pressure high, waiting, and checking in each iteration.
Introducing variables, like VAR[8] and VAR[1], enables dynamic control within loops, reducing redundancy and enhancing script readability. With variables, the script becomes concise, adaptable, and easier to maintain.

Moreover, lacking variables required splitting the script into three files due to its extensive length, totaling 64KB. Each file had repeated code sections for different ramps, resulting in redundancy and increased storage needs. However, with variables, the script is consolidated into a single 7.1KB file, optimizing storage and improving overall script organization and management.

Copyright EEPod LLC 2018-2025

5 LOGADD.EXE Windows Console Mode Utility

The LOGADD.EXE Windows console mode program is available for quickly combining script log files via USB (no need to access the SD-Card or plug in the OBDII cable) from several different MyCANIC-FD and MyCANIC-IOT tools into a single consolidated log file. Note that this utility will not work with the classic (gold keypad) MyCANIC.

Simply run the LOGADD program from a console mode window along with the name of the log file to be collected from each MyCANIC-FD/IOT. For example, the following command will read the VIN_LIST.LOG file from the MyCANIC-FD/IOT connected via USB, save it to a sub-directory based on the electronic serial number of the MyCANIC-FD/IOT, add it to the combined VIN_LIST.LOG file on the PC in the current directory and then delete the VIN_LIST.LOG file on the MyCANIC-FD/IOT.

LOGADD.EXE VIN_LIST.LOG

Copyright EEPod LLC 2018-2025

6 References and Acronyms

6.1.1 References

SAE J2534	Recommended Practice For Pass Thru Vehicle Reprogramming

6.1.2 Acronyms

BIN	Binary File
CAN	Controller Area Network
CSV	Comma Separated Variable
DTC	Diagnostic Trouble Code
ECU	Electronic Control Unit
ISO	International Standards Organization
LCD	Liquid Crystal Display
MSG	Message
PID	Parameter ID
OBD	On-Board Diagnostic
SAE	Society of Automotive Engineers
SD	Secure Digital
SID	Service ID
VBF	Volvo Binary Format
VIN	Vehicle Identification Number

Copyright EEPod LLC 2018-2025