



Bootloader Awareness Manual GRUB

[TRACE32 Online Help](#)

[TRACE32 Directory](#)

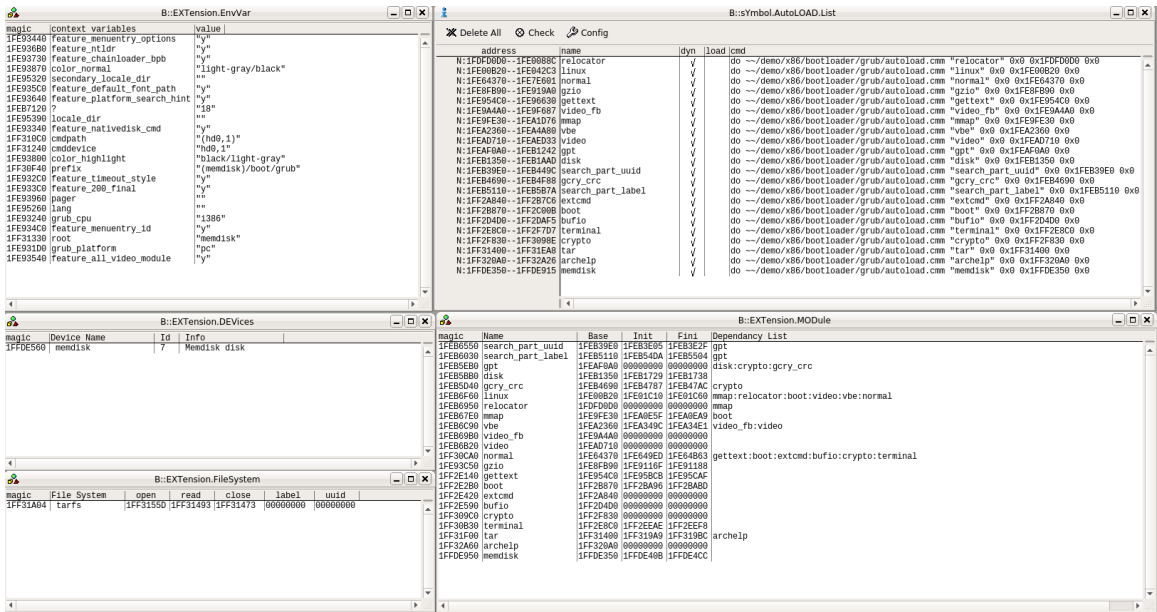
[TRACE32 Index](#)

TRACE32 Documents		
Bootloader Awareness Manuals		
Bootloader Awareness Manual GRUB	1	
History	2	
Overview	2	
Brief Overview of Documents for New Users	3	
Supported Architectures	3	
Configuration	4	
Features	5	
Display of GRUB Resources	5	
GRUB Specific Menu	6	
Symbol Autoloader	7	
GRUB Commands	9	
EXTension.DEVices	Display GRUB boot devices	9
EXTension.EnvVar	Display GRUB environment variables	9
EXTension.FileSystem	Display GRUB boot file system	10
EXTension.MODule	Display GRUB loaded modules	10
EXTension.sYmbol	Symbol management	11
EXTension.sYmbol.DELeMod	Delete module debug symbols	11
EXTension.sYmbol.LOADMod	Load module debug symbols	11
EXTension.sYmbol.ModPATH	Set GRUB module directory path	12
GRUB PRACTICE Functions	13	
TASK.MOD.MAGIC()	Magic of GRUB module	13
TASK.MOD.NAME()	Name of GRUB module	13
TASK.MOD.BASE()	Base address of GRUB module	13
TASK.MOD.INIT()	Init address of GRUB module	14
TASK.MOD.SIZE()	Size of GRUB module	14
TASK.Y.MODP(modpath)	Get path for GRUB modules	14

History

28-Aug-18 The title of the manual was changed from “Bootloader Debugger for <x>” to “Bootloader Awareness Manual <x>”.

Overview



The Bootloader Awareness for GRUB contains special extensions to the TRACE32 Debugger. This chapter describes the additional features, such as additional commands and debugging approaches.

Architecture-independent information:

- **“Debugger Basics - Training”** (training_debugger.pdf): Get familiar with the basic features of a TRACE32 debugger.
- **“T32Start”** (app_t32start.pdf): T32Start assists you in starting TRACE32 PowerView instances for different configurations of the debugger. T32Start is only available for Windows.
- **“General Commands”** (general_ref_<x>.pdf): Alphabetic list of debug commands.

Architecture-specific information:

- **“Processor Architecture Manuals”**: These manuals describe commands that are specific for the processor architecture supported by your debug cable. To access the manual for your processor architecture, proceed as follows:
 - Choose **Help** menu > **Processor Architecture Manual**.
- **“OS Awareness Manuals”** (rtos_<os>.pdf): TRACE32 PowerView can be extended for operating system-aware debugging. The appropriate OS Awareness manual informs you how to enable the OS-aware debugging.
- **“UEFI Awareness Manuals”** (uefi_<x>.pdf): TRACE32 PowerView can be extended for UEFI-aware debugging. The appropriate UEFI manual informs you how to enable the UEFI-aware debugging.

Supported Architectures

Currently GRUB is supported for the following architectures:

- GRUB on Intel® x86/x64 architectures
- GRUB on Arm architectures

Configuration

The Bootloader Awareness for GRUB is configured by loading an extension definition file called “grub.t32” from the demo directory with the **EXTension.CONFIG** command.

Additionally, load the “grub.men” menu file (see “**GRUB specific Menu**”).

A full configuration for all supported architecture can look like this (the path prefix `~~` expands to the system directory of TRACE32):

```
; Load the GRUB symbols
Data.LOAD.Elf <path_to_grub_elf>/kernel.exec /NoCODE

; Load the Bootloader Awareness for GRUB:
EXTension.CONFIG ~/demo/<arch>/bootloader/grub/grub.t32

; Load the additional menu:
MENU.ReProgram ~/demo/<arch>/bootloader/grub/grub.men
```

Features

The Bootloader Awareness supports the following features.

Display of GRUB Resources

The extension defines new commands to display various resources. Information on the following GRUB components can be displayed

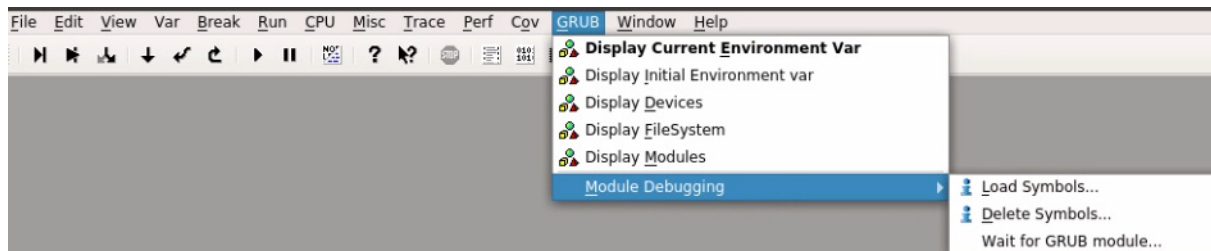
EXTension.MODule	Loaded modules
EXTension.DEVices	Boot devices
EXTension.EnvVar	GRUB environment variables
EXTension.FileSystem	Boot file systems

For a detailed description of each command, refer to chapter “**GRUB Commands**”.

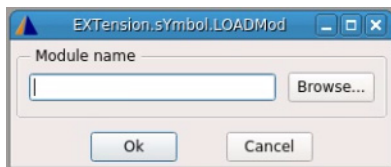
GRUB Specific Menu

The menu file “grub.men” contains a menu with GRUB specific menu items. Load this menu with the **MENU.ReProgram** command.

You will find a new menu called **GRUB**.

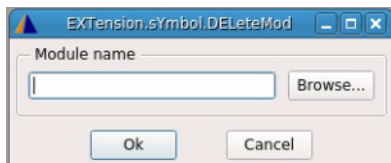


- The **Display** menu items launch the GRUB resource display windows.
- Selecting **GRUB > Module Debugging > Load Symbols** will open the following dialog.



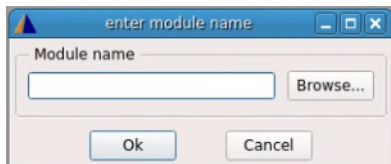
You can enter the GRUB module name, with/without the “.mod” extension and click **OK**, or click the **Browse** button and select the GRUB module directly. The **EXTension.sYmbol.LOADMod** command will be executed with the selected module as parameter.

- Selecting **GRUB > Module Debugging > Delete Symbols** will open the following dialog.



You can enter the GRUB module name, with/without the “.mod” extension and click **OK**, or click the **Browse** button and select the GRUB module directly. The **EXTension.sYmbol.DELeTeMod** command will be executed with the selected module as parameter.

- Selecting **GRUB > Module Debugging > Wait for GRUB module** will open the following dialog.



You can enter the GRUB module name you want to debug with or without the “.mod” extension and click **OK**, or click the **Browse** button and select the GRUB module directly. If the module is already loaded, the Bootloader Awareness will load its debug symbols. If the module is not yet loaded, TRACE32 will wait till GRUB loads this module, then TRACE32 will stop the application execution and load module debug symbols.

Symbol Autoloader

The Bootloader Awareness contains a “Symbol Autoloader”, which automatically loads symbol files corresponding to executed GRUB modules. The autoloader maintains a list of address ranges, corresponding to GRUB modules and the appropriate load command. Whenever the user accesses an address within an address range specified in the autoloader (e.g. via [List](#)), the debugger invokes the command necessary to load the corresponding symbols to the appropriate addresses (including relocation). This is usually done via a PRACTICE script.

In order to load symbol files, the debugger needs to be aware of the currently loaded components. The command [sYmbol.AutoLOAD.CHECK](#) defines *when* these module data structures are read by the debugger (only on demand or after each program execution).

[sYmbol.AutoLOAD.CHECK](#) [ON | OFF | ONGO]

The loaded components can change over time as processes are started and stopped and libraries are loaded or unloaded. The command [sYmbol.AutoLOAD.CHECK](#) configures the strategy when to “check” the kernel data structures for changes in order to keep the debugger’s information regarding the components up-to-date.

without parameters	The sYmbol.AutoLOAD.CHECK command <i>immediately</i> updates the component information by reading the kernel data structures. This information includes the component name, the load address used to fill the autoloader list (shown via sYmbol.AutoLOAD.List).
ON	The debugger <i>automatically</i> reads the component information <i>each time the target stops executing</i> (even after assembly steps), because the debugger has to assume that the component information might have changed. This significantly slows down the debugger, which is inconvenient and often superfluous, e.g. when stepping through code that does not load or unload components.
ONGO	The debugger checks for changed component info like with ON , but <i>not when performing single steps</i> .
OFF	No automatic read is performed. In this case, the update has to be triggered manually when considered necessary by the user.

NOTE: The autoloader covers only modules that are already loaded. Modules that are not yet loaded are not covered.

When configuring the Bootloader Awareness, set up the symbol autoloader with the following command:

```
sYmbol.AutoLOAD.CHECKCoMmanD "<action>"  
<action> to take for symbol load, e.g. "DO autoload.cmm "
```

The command **sYmbol.AutoLOAD.CHECKCoMmanD** is used to define which action is to be taken for loading the symbols corresponding to a specific address. The defined action is invoked with specific parameters (see below). With GRUB, the pre-defined action is to call the script `~/demo/<arch>/bootloader/grub/autoload.cmm`.

NOTE: The action parameter needs to be enclosed in quotation marks (for the parser it is a string).

Note that *defining* this action does not cause its execution. The action is executed on demand, i.e. when the address is actually accessed by the debugger e.g. in the **List** or **Trace.List** window. In this case, the autoloader executes the *<action>* appending parameters indicating the name of the component and the load address.

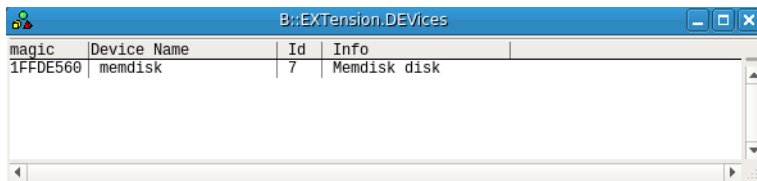
To check the currently active components, use the command **sYmbol.AutoLOAD.List**. Together with the component name, the **sYmbol.AutoLOAD.List** window shows details like the load address, and the command that will be executed to load the corresponding object files with the symbol information. Only components shown in this list are handled by the autoloader.

The screenshot shows a window titled "B::sYmbol.AutoLOAD.List" with a table of loaded components. The table has columns for address, name, dyn, load, and cmd. The components listed include relocator, linux, normal, gzio, gettext, video_fb, mmap, vbe, video, gpt, disk, search_part_uuid, gcry_crc, search_part_label, extcmd, boot, bufio, terminal, crypto, tar, archelp, and memdisk. Each component has a corresponding command string in the 'cmd' column, such as "relocator" 0x0 0x1FDFD000 0x0.

address	name	dyn	load	cmd
N:1FDFD000--1FE0088C	relocator	√		do ~/demo/x86/bootloader/grub/autoload.cmm "relocator" 0x0 0x1FDFD000 0x0
N:1FE00B20--1FE042C3	linux	√		do ~/demo/x86/bootloader/grub/autoload.cmm "linux" 0x0 0x1FE00B20 0x0
N:1FE64370--1FE7E601	normal	√		do ~/demo/x86/bootloader/grub/autoload.cmm "normal" 0x0 0x1FE64370 0x0
N:1FE8FB90--1FE919A0	gzio	√		do ~/demo/x86/bootloader/grub/autoload.cmm "gzio" 0x0 0x1FE8FB90 0x0
N:1FE954C0--1FE96630	gettext	√		do ~/demo/x86/bootloader/grub/autoload.cmm "gettext" 0x0 0x1FE954C0 0x0
N:1FE9A4A0--1FE9F687	video_fb	√		do ~/demo/x86/bootloader/grub/autoload.cmm "video_fb" 0x0 0x1FE9A4A0 0x0
N:1FE9FE30--1FEA1D76	mmap	√		do ~/demo/x86/bootloader/grub/autoload.cmm "mmap" 0x0 0x1FE9FE30 0x0
N:1FEA2360--1FEA4A80	vbe	√		do ~/demo/x86/bootloader/grub/autoload.cmm "vbe" 0x0 0x1FEA2360 0x0
N:1FEAD710--1FEAD333	video	√		do ~/demo/x86/bootloader/grub/autoload.cmm "video" 0x0 0x1FEAD710 0x0
N:1FEAF9A0--1FEB1242	gpt	√		do ~/demo/x86/bootloader/grub/autoload.cmm "gpt" 0x0 0x1FEAF9A0 0x0
N:1FEB1350--1FEB1AAD	disk	√		do ~/demo/x86/bootloader/grub/autoload.cmm "disk" 0x0 0x1FEB1350 0x0
N:1FEB39E0--1FEB449C	search_part_uuid	√		do ~/demo/x86/bootloader/grub/autoload.cmm "search_part_uuid" 0x0 0x1FEB39E0 0x0
N:1FEB4590--1FEB4F88	gcry_crc	√		do ~/demo/x86/bootloader/grub/autoload.cmm "gcry_crc" 0x0 0x1FEB4590 0x0
N:1FEB5110--1FEB5B7A	search_part_label	√		do ~/demo/x86/bootloader/grub/autoload.cmm "search_part_label" 0x0 0x1FEB5110 0x0
N:1FF2A840--1FF2B7C6	extcmd	√		do ~/demo/x86/bootloader/grub/autoload.cmm "extcmd" 0x0 0x1FF2A840 0x0
N:1FF2B870--1FF2C00B	boot	√		do ~/demo/x86/bootloader/grub/autoload.cmm "boot" 0x0 0x1FF2B870 0x0
N:1FF2D400--1FF2DAF5	bufio	√		do ~/demo/x86/bootloader/grub/autoload.cmm "bufio" 0x0 0x1FF2D400 0x0
N:1FF2E8C0--1FF2F7D7	terminal	√		do ~/demo/x86/bootloader/grub/autoload.cmm "terminal" 0x0 0x1FF2E8C0 0x0
N:1FF2F830--1FF3098E	crypto	√		do ~/demo/x86/bootloader/grub/autoload.cmm "crypto" 0x0 0x1FF2F830 0x0
N:1FF31400--1FF31EA8	tar	√		do ~/demo/x86/bootloader/grub/autoload.cmm "tar" 0x0 0x1FF31400 0x0
N:1FF320A0--1FF32A26	archelp	√		do ~/demo/x86/bootloader/grub/autoload.cmm "archelp" 0x0 0x1FF320A0 0x0
N:1FFDE350--1FFDE915	memdisk	√		do ~/demo/x86/bootloader/grub/autoload.cmm "memdisk" 0x0 0x1FFDE350 0x0

Format: **EXTension.DEVices**

Displays the boot device ID and Info.



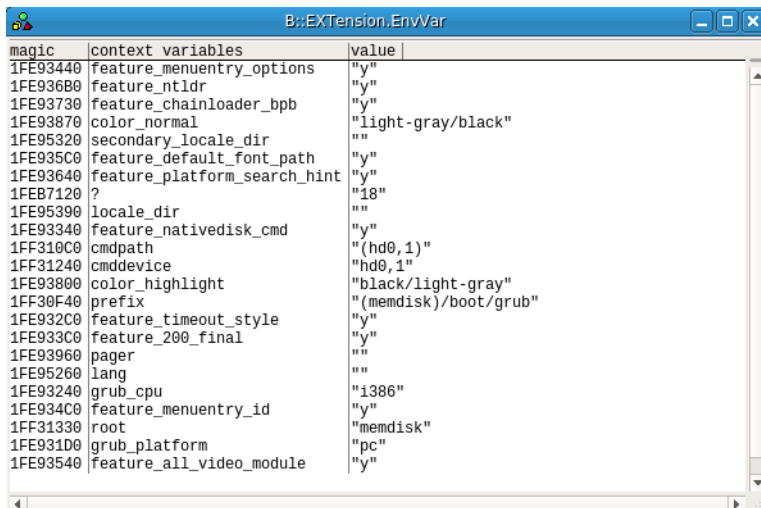
The screenshot shows a window titled "B::EXTension.DEVices" with a table containing one row of data:

magic	Device Name	Id	Info
1FFDE560	memdisk	7	Memdisk disk

“magic” is a unique ID, used by the Bootloader Awareness to identify the boot devices in memory. The “Id”, “Device Name” and “Info” fields are referenced in the GRUB source code `<grub_master>/include/grub/disk.h`

Format: **EXTension.EnvVar**

Displays the GRUB environment variables, the so-called “context”.



The screenshot shows a window titled "B::EXTension.EnvVar" with a table listing GRUB environment variables:

magic	context variables	value
1FE93440	feature_menuentry_options	"y"
1FE936B0	feature_ntldr	"y"
1FE93730	feature_chainloader_bpb	"y"
1FE93870	color_normal	"light-gray/black"
1FE95320	secondary_locale_dir	""
1FE935C0	feature_default_font_path	"y"
1FE93640	feature_platform_search_hint	"y"
1FEB7120	?	"18"
1FE95390	locale_dir	""
1FE93340	feature_nativedisk_cmd	"y"
1FF310C0	cmdpath	"(hd0,1)"
1FF31240	cmddevice	"hd0,1"
1FE93800	color_highlight	"black/light-gray"
1FF30F40	prefix	"(memdisk)/boot/grub"
1FE932C0	feature_timeout_style	"y"
1FE933C0	feature_200_final	"y"
1FE93960	pager	""
1FE95260	lang	""
1FE93240	grub_cpu	"i386"
1FE934C0	feature_menuentry_id	"y"
1FF31330	root	"memdisk"
1FE931D0	grub_platform	"pc"
1FE93540	feature_all_video_module	"y"

“magic” is a unique ID, used by the Bootloader Awareness to identify the address of each environment variable in memory. “context variables” lists the existing variable names, and “value” lists the values of the variables.

Format: **EXTension.FileSystem**

Displays the GRUB boot file system.

magic	File System	open	read	close	label	uuid
1FF31A04	tarfs	1FF3155D	1FF31493	1FF31473	00000000	00000000

“magic” is a unique ID, used by the Bootloader Awareness to identify a boot file system in memory. The “open”, “read” and “close” fields are mouse sensitive. Double-clicking will open a **List** view window of the corresponding function.

EXTension.MODULE

Display GRUB loaded modules

Format: **EXTension.MODULE**

Displays the GRUB loaded modules (module name, base address, Init function address, Fini function address, and module dependency list).

magic	Name	Base	Init	Fini	Dependency List
1FE86550	search_part_u	1FE839E0	1FE83E05	1FE83E2F	gpt
1FE86030	search_part_l	1FE85110	1FE854DA	1FE85504	gpt
1FE85E80	gpt	1FEAF0A0	00000000	00000000	disk:crypto:gcry_crc
1FE85BB0	disk	1FE81350	1FE81729	1FE81738	
1FE85D40	gcry_crc	1FE84690	1FE84787	1FE847AC	crypto
1FE86F60	linux	1FE00B20	1FE01C10	1FE01C60	mmap:relocator:boot:video:vbe:normal
1FE86950	relocator	1FDF00D0	00000000	00000000	mmap
1FE867E0	mmap	1FE9FE30	1FEA0E5F	1FEA0EA9	boot
1FE86C90	vbe	1FEA2360	1FEA349C	1FEA34E1	video_fb:video
1FE869B0	video_fb	1FE9A4A0	00000000	00000000	
1FE86B20	video	1FEAD710	00000000	00000000	
1FF30CA0	normal	1FE64370	1FE649ED	1FE64B63	gettext:boot:extcmd:bufio:crypto:terminal
1FE93C50	gzio	1FE8FB90	1FE9116F	1FE91188	
1FF2E140	gettext	1FE954C0	1FE95B8C	1FE95CAF	
1FF2E2B0	boot	1FF2B870	1FF2BA96	1FF2BADD	
1FF2E420	extcmd	1FF2A840	00000000	00000000	
1FF2E590	bufio	1FF2D4D0	00000000	00000000	
1FF309C0	crypto	1FF2F830	00000000	00000000	
1FF30B30	terminal	1FF2E8C0	1FF2EEAE	1FF2EEF8	
1FF31F00	tar	1FF31400	1FF319A9	1FF319BC	archelp
1FF32A60	archelp	1FF320A0	00000000	00000000	
1FFDE950	memdisk	1FFDE350	1FFDE40B	1FFDE4CC	

“magic” is a unique ID, used by the Bootloader Awareness to identify a module. The “magic” field is mouse sensitive. Double-clicking it will show a variable view of the module GRUB data structure.

Double-clicking “Init” or “Fini” fields will open a **List** window of the Init / Fini module function.

The **EXTension.sYmbol** command group helps to load and unload the symbols of a given GRUB module. The commands are described below.

EXTension.sYmbol.DELeMod

Delete module debug symbols

Format: **EXTension.sYmbol.DELeMod** "<module_name>"

Deletes the module debug symbols from the Bootloader Awareness.

<module_name>

For a parameter description, see [EXTension.sYmbol.LOADMod](#).

EXTension.sYmbol.LOADMod

Load module debug symbols

Format: **EXTension.sYmbol.LOADMod** "<module_name>"

Loads GRUB module debug symbols with the right offset using the Symbol Autoloader feature.

<module_name>

The <module_name> parameter can be a full path to the module and can be entered with or without the file name extension ".mod".

If the full path to a module is entered as a parameter, the used path is automatically set as the default path for the GRUB modules.

If just a module name is used as a parameter, the Bootloader Awareness will load the module from the previously set path.

For more details, refer to [EXTension.sYmbol.ModPATH](#) and [TASK.Y.MODP\(modpath\)](#).

Example 1:

```
EXTension.sYmbol.LOADMod "normal"
```

Example 2:

```
EXTension.sYmbol.LOADMod "normal.mod"
```

Example 3:

```
EXTension.sYmbol.LOADMod "/path/to/normal.mod"
```

EXTension.sYmbol.ModPATH

Set GRUB module directory path

Format: **EXTension.sYmbol.ModPATH** "<module_path>"

Sets the entered module path as a default search for GRUB module when trying to load GRUB module debug symbol.

<module_name>

For a parameter description, see [EXTension.sYmbol.LOADMod](#).

There are special definitions for GRUB specific PRACTICE functions.

TASK.MOD.MAGIC()

Magic of GRUB module

[build 96505 - DVD 09/2018]

Syntax: **TASK.MOD.MAGIC("<module_name>")**

Returns the magic of a given loaded GRUB module.

Parameter Type: [String](#).

Return Value Type: [Hex value](#).

TASK.MOD.NAME()

Name of GRUB module

[build 96505 - DVD 09/2018]

Syntax: **TASK.MOD.NAME(<module_magic>)**

Returns the module name of a given module magic number.

Parameter Type: [Decimal](#) or [hex](#) or [binary value](#).

Return Value Type: [String](#).

TASK.MOD.BASE()

Base address of GRUB module

[build 96505 - DVD 09/2018]

Syntax: **TASK.MOD.BASE("<module_name>")**

Returns the base address of a loaded GRUB module name.

Parameter Type: [String](#).

Return Value Type: [Hex value](#).

Syntax: **TASK.MOD.INIT("<module_name>")**

Returns the address of the "init" function of a loaded GRUB module.

Parameter Type: [String](#).

Return Value Type: [Hex value](#).

TASK.MOD.SIZE()

Size of GRUB module

Syntax: **TASK.MOD.SIZE("<module_name>")**

Returns the size of a loaded GRUB module.

Parameter Type: [String](#).

Return Value Type: [Hex value](#).

TASK.Y.MODP(modpath)

Get path for GRUB modules

Syntax: **TASK.Y.MODP(modpath)**

Returns the previously set module path using command [EXTension.sYmbol.ModPATH](#)

Return Value Type: [String](#).

Example:

```
LOCAL &modpath
&modpath=TASK.Y.MODP(modpath)
IF "&modpath"==" "
(
    PRINT %WARNING "no module path set, use EXTension.sYmbol.ModPATH"
)
```