LAUTERBACH
DEVELOPMENT TOOLS

UEFI Awareness Manual H2O

Release 02.2026

MANUAL

UEFI Awareness Manual H2O

TRACE32 Online Help

TRACE32 Directory

TRACE32 Index

TRACE32 Documents ..... 🗁

UEFI Awareness Manuals ..... 🗁

UEFI Awareness Manual H2O 1

History 4

Overview 5

Brief Overview of Documents for New Users 6

Supported Versions 6

Configuration 7

x86/Atom 32-Bit 7

x64/Atom 64-Bit 7

Hooks & Internals in InsydeH2O 8

Features 9

Display of UEFI Resources 9

Symbol Autoloader 10

Autoloader Configuration 10

Scan the UEFI Module Table 11

Display the Autoloader Table 12

InsydeH2O Specific Menu 13

Debugging UEFI Phases of InsydeH2O 15

Debugging from Reset Vector 15

SEC Phase 15

PEI Phase 16

DXE Phase 17

BDS Phase 17

InsydeH2O Commands 18

EXTension.ConfigTab Display DXE configuration table 18

EXTension.DXEDRiVer Display loaded DXE drivers 18

EXTension.DXEModule Display DXE modules 19

EXTension.FV Display firmware volumes 20

EXTension.HOB Display HOBs 21

EXTension.Option Set awareness options 22

EXTension.PEIModule Display PEI modules 22

EXTension.PEISvc Display PEI services 23

EXTension.POST Display POST code 23
EXTension.PROTocol Display installed protocols 24
EXTension.UCode Display microcodes 24

InsydeH2O PRACTICE Functions 25
EXT.DXEDRV.ENTRY() Entry address for DXE driver 25
EXT.DXEDRV.MAGIC() Magic of DXE driver 25
EXT.DXEDRV.PATH() Build path for DXE driver 25
EXT.DXEFILE.PATH() Build path for DXE module 26
EXT.PEIM.ENTR() Entry addres for PEI module 26
EXT.PEIM.MAGIC() Magic of PEI module 26
EXT.PEIM.PATH() Build path for PEI module 26

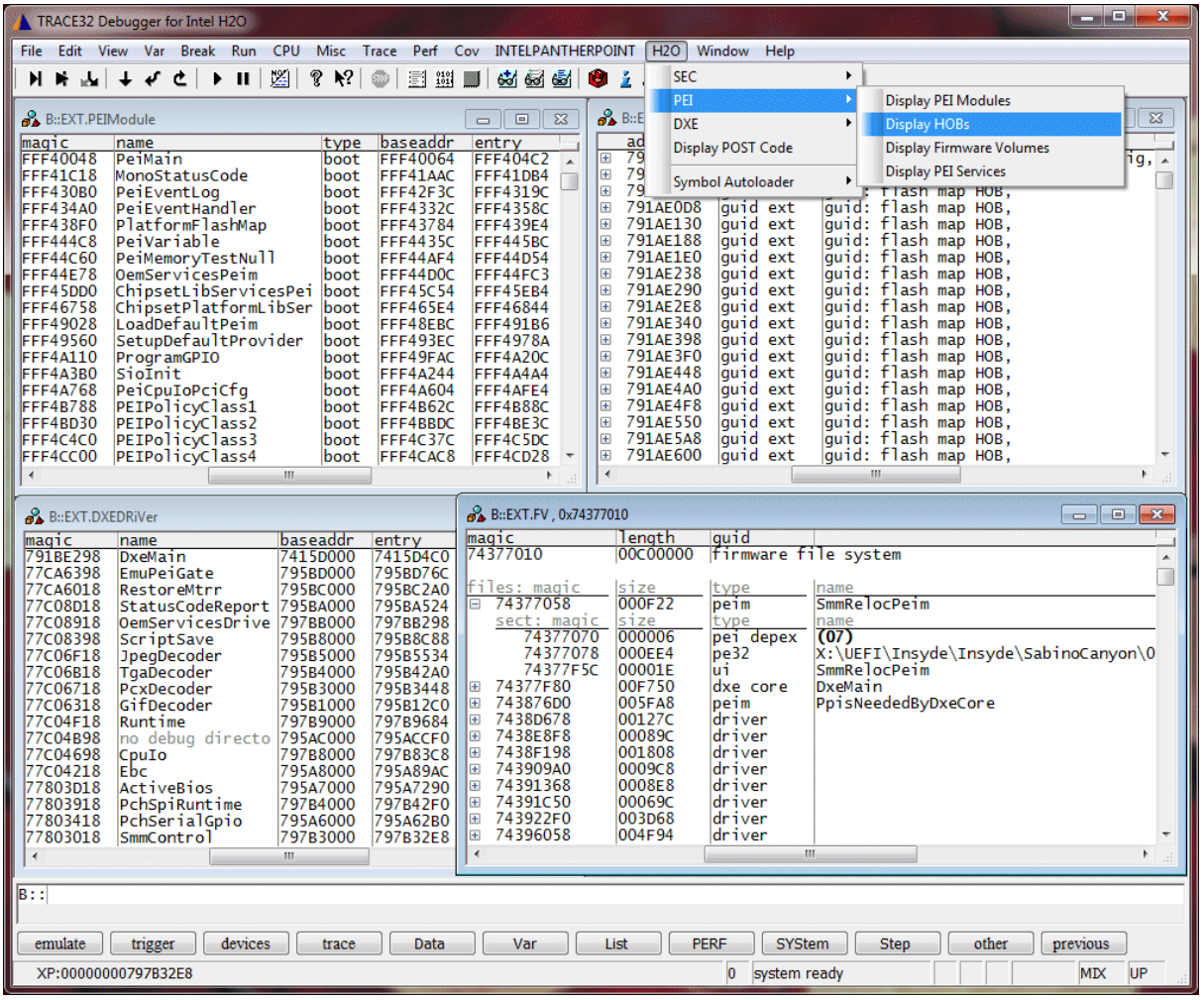
UEFI Awareness Manual H2O

Version 03-Apr-2026

History

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

Overview



The UEFI Awareness for InsydeH2O contains special extensions to the TRACE32 Debugger. This manual describes the additional features, such as additional commands and debugging approaches.

Brief Overview of Documents for New Users

Architecture-independent information:

- **"Debugger Tutorial"** (debugger_tutorial.pdf): Get familiar with the basic features of a TRACE32 debugger.
- **"General Commands"** (general_ref_<*x*>.pdf): Alphabetic list of debug commands.
- **"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.

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**.
- **"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 Versions

Currently InsydeH2O is supported for the following versions:

- 32-bit version 03.61.18 on OakTrail (Atom Z6xx)
- 64-bit version 03.72.51 on SabinoCanyon (Ivy Bridge Dual)

Configuration

The UEFI Awareness for InsydeH2O is configured by loading an extension definition file called “h2o.t32” from the demo directory with the **EXTension.CONFIG** command. Additionally, load the “h2o.men” menu file (see “**H2O specific Menu**”) and configure the **Symbol Autoloader**.

x86/Atom 32-Bit

A full configuration for x86/Atom **32-bit** can look like this (the path prefix ~~ expands to the TRACE32 installation directory.):

```
; Load the InsydeH2O awareness
EXTension.CONFIG ~~/demo/x86/bootloader/uefi/h2o/h2o.t32

; Configure symbol Autoloader
sYmbol.AutoLOAD.CHECKUEFI "DO ~~/demo/x86/bootloader/uefi/h2o/autoload.cmm"

; Program the additional menu
MENU.ReProgram ~~/demo/x86/bootloader/uefi/h2o/h2o.men
```

See also the example scripts in the directory `~~/demo/x86/bootloader/uefi/h2o`.

x64/Atom 64-Bit

A full configuration for x64/Atom **64-bit** can look like this (the path prefix ~~ expands to the TRACE32 installation directory.):

```
; Load the InsydeH2O awareness
EXTension.CONFIG ~~/demo/x64/bootloader/uefi/h2o/h2o.t32

; Configure symbol Autoloader
sYmbol.AutoLOAD.CHECKUEFI "DO ~~/demo/x64/bootloader/uefi/h2o/autoload.cmm"

; Program the additional menu
MENU.ReProgram ~~/demo/x64/bootloader/uefi/h2o/h2o.men
```

See also the example scripts in the directory ~~/`demo/x64/bootloader/uefi/h2o`.

When using the debug build of the InsydeH2O (which is recommended), the build system may automatically insert software breakpoints into the images when loading new modules. TRACE32 does **not** use these software breakpoints. Either remove them from the debug build, or simply continue when halting there.

Features

The UEFI Awareness for InsydeH2O supports the following features.

Display of UEFI Resources

The extension defines new TRACE32 commands to display various UEFI resources. Information on the following UEFI components can be displayed:

EXTension.POST	POST code

SEC phase:

EXTension.UCode	Available microcodes

PEI phase:

EXTension.FV PEI	PEI firmware volumes
EXTension.PEIModule	PEI modules in FVs
EXTension.HOB PEI	PEI HOBs
EXTension.PEISvc	PEI services

DXE phase:

EXTension.FV DXE	DXE firmware volumes
EXTension.DXEModule	DXE modules in FVs
EXTension.DXEDRiVer	Loaded DXE drivers
EXTension.HOB DXE	DXE HOBs
EXTension.PROTocol DXE	Installed DXE protocols
EXTension.ConfigTab	DXE configuration table

For a description of the commands, refer to chapter "**InsydeH2O Commands**".

Since the x86/x64/Atom architecture does not allow to read memory while the program execution is running, the information can only be displayed, if the program execution is stopped.

Symbol Autoloader

The UEFI code is provided by the boot FLASH, but debugging becomes more comfortable when debug symbols are available.

TRACE32 contains an “Autoloader”, which can be set up for automatic loading of symbol files. The Autoloader maintains a list of address ranges, corresponding UEFI components and the appropriate load command. Whenever the user accesses an address within an address range known to the Autoloader, the debugger invokes the load associated command. The command is usually a call to a PRACTICE script, that handles loading the symbol file.

The TRACE32 Autoloader has to be set up. This includes the following steps:

1. Autoloader configuration.
2. Scan of the UEFI module table to the Autoloader table.
3. Display of the Autoloader table.

Autoloader Configuration

The command **sYmbol.AutoLOAD.CHECKUEFI** <*load_command*> specifies the command that is automatically used by the Autoloader to load the symbol information. Typically the script `autoload.cmm` provided by Lauterbach is called.

The command **sYmbol.AutoLOAD.CHECKUEFI** implicitly also defines the parameters that TRACE32 uses internally for the Autoloader.

The script is provided in the TRACE32 demo directory:

- 32-bit: ~~/`demo/x86/bootloader/uefi/h2o/autoload.cmm`.
- 64-bit: ~~/`demo/x64/bootloader/uefi/h2o/autoload.cmm`.

Example:

```
; Configure symbol Autoloader for 32-bit InsydeH2O
sYmbol.AutoLOAD.CHECKUEFI "DO ~~/demo/x86/bootloader/uefi/h2o/autoload.cmm"
```

Scan the UEFI Module Table

When the Autoloader is configured, the command **sYmbol.AutoLOAD.CHECK** can be used to scan the UEFI module table into the Autoloader table and to activate the Autoloader.

Since the UEFI module table is updated by UEFI a re-scan might be necessary.

The point of time at which the UEFI module table is re-scanned can be set very flexibly:

sYmbol.AutoLOAD.CHECK [**ON** | **OFF** | **ONGO**]

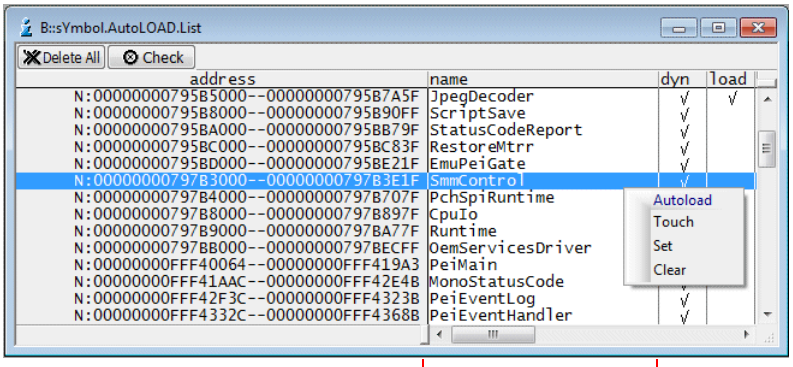
The default setting is **sYmbol.AutoLOAD CHECK OFF.** With this setting TRACE32 re-scans the UEFI module table only on request by using the **sYmbol.AutoLOAD.CHECK** command.

With **sYmbol.AutoLOAD.CHECK ON**, TRACE32 re-scans the UEFI module table after every single step and whenever the program execution is stopped. This significantly slows down the speed of TRACE32.

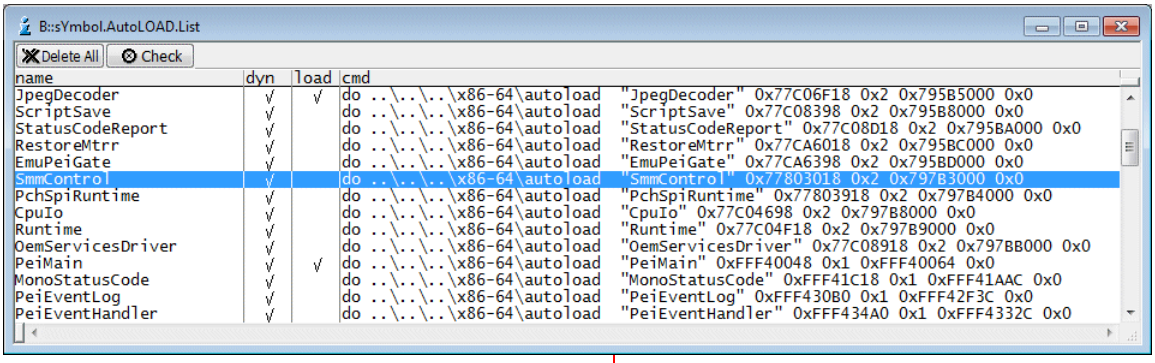
With **sYmbol.AutoLOAD.CHECK ONGO**, TRACE32 re-scans the UEFI module table whenever the program execution is stopped.

NOTE:	The Autoloader can load the symbol information for the SecCore, the PeiCore, all PEI modules and the DXE core as soon as the memory mode (e.g. 32-bit protected mode) used by UEFI is activated. The Autoloader can only load symbol information for DXE modules that are already loaded.

The command "**sYmbol.AutoLOAD.List**" shows a list of all known address ranges/components and their symbol load commands.



Module address range | Module name | Module status
dyn: (no meaning)
load: symbols for module are loaded



Load command | Parameters for load command

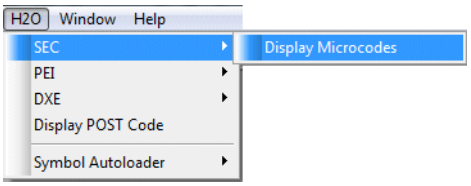
Autoload context menu	
Touch	Advise TRACE32 to load the symbols for the selected module now.
Set	Mark selected module as loaded.
Clear	Delete symbols for the selected module in TRACE32.

InsydeH2O Specific Menu

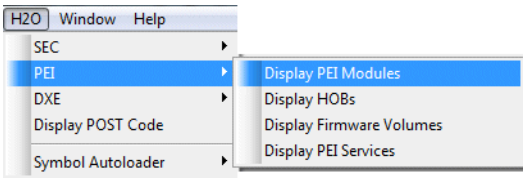
The menu file “h2o.men” allows to program a menu with InsydeH2O specific menu items. Load this menu with the **MENU.ReProgram** command.

You will find a new menu called **H2O**.

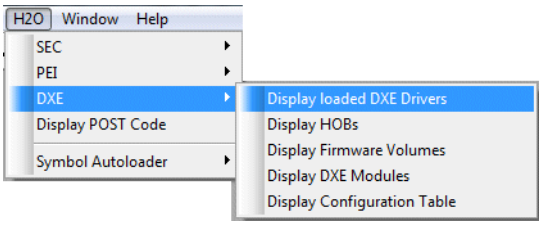
- The **SEC** submenu provides a list of the available microcodes.



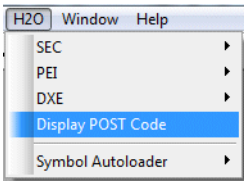
- Use the **PEI** submenu to launch windows displaying PEI specific resources.



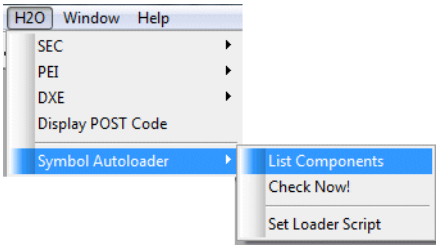
- Use the **DXE** submenu to launch windows displaying DXE specific resources.



- The **Display POST Code** menu item allows to display the current POST (Power On SelfTest) code.



Use the **Symbol Autoloader** submenu to configure the Autoloader. See also chapter "**Symbol Autoloader**".



- **List Components** opens a **sYmbol.AutoLOAD.List** window showing all components currently active in the Autoloader.
- **Check Now!** performs a **sYmbol.AutoLOAD.CHECK** and reloads the Autoloader list.
- **Set Loader Script** allows you to specify the script that is called when a symbol file load is required. You may also set the automatic Autoloader check.

Debugging UEFI Phases of InsydeH2O

UEFI runs in several “phases”. It starts with the “Security” (SEC) phase which immediately switches to the “Pre-EFI Initialization Environment” (PEI) phase. After this phase ended, control is given to the “Driver Execution Environment” (DXE) phase. Shortly, before the OS is booted, the “Boot Device Selection” (BDS) phase is running.

Each of this phases needs a different debugging environment. See below for a detailed description of each phase.

Debugging from Reset Vector

TRACE32 is a JTAG-based debugging tool and, as such, allows the user to start debugging their Atom/x86 system right from the reset vector (normally at BP:0xF000:0xFFF0). It is possible to walk through the very first steps of the start-up to detect FLASH problems or faulty reset behavior.

Shortly after reset, the system switches into the SEC phase.

SEC Phase

At the start of the SEC phase there is no RAM accessible. Variables, in particular the call stack, are stored in “internal memory”, e.g. the cache.

The JTAG debugger has no access to this “internal memory”. And even worse, the CPU might crash when the debugger tries to access this memory. To be especially careful it is recommended to block accesses to the “internal memory” by the following TRACE32 command:

```
MAP.DenyAccess <address_range>
```

As soon RAM is initialized the access can be unblocked by the TRACE32 command:

```
MAP.NoDenyAccess <address_range>
```

If symbol information is available for the SecCore this information has to be loaded by the following command as long as the cores are in real-mode.

Data.LOAD.eXe *<file>* **BP:***<address>* **/NoCODE /NoMMU**	Load exe file. - **BP:***<address>* advises TRACE32 to relocate the symbol addresses the specified real-mode address - **NoCODE** advises TRACE32 to load only the debug symbols, since code is already in FLASH - **NoMMU** advises TRACE32 to ignore the segment information

```
Data.LOAD.eXe SecCore.pdb BP:0xf000:0x0f0e0 /NoCODE /NoMMU
```

As soon as UEFI memory mode is initialized, the Autoloader can be used to reload the symbol information for the SecCore.

```
sYmbol.AutoLOAD.TOUCH "SecCore"
```

Before the microcode is patched in the SEC phase, you can inspect the available microcodes by the following command:

EXTension.UCode

PEI Phase

If you want to debug the PEI phase right from the start, stop the program execution in the SEC phase. Then load the symbols of the PEI core module (“PeiCore”) with the Autoloader, and go until “PeiCore”:

```
sYmbol.AutoLOAD.CHECK
sYmbol.AutoLOAD.Touch "PeiCore"
Go PeiCore
```

InsydeH2O starts the PeiCore several times with different settings. The first time after the SEC phase, code runs from Flash and data is in the small RAM initialized in the SEC phase. PeiCore then initializes the full external RAM and calls itself, starting PeiCore a second time, now with code in Flash and data in the larger external RAM. In some configurations, PeiCore may be called a third time, now executing code and data both in external RAM. If code runs from external RAM, check and touch the module in the Autoloader again, to trigger a reload of the PeiCore symbols, now to new RAM addresses.

Inspect the PEI resources with the menu items in the “PEI” submenu.

For debugging a dynamic PEI module from its entry point, a special script “go_peim” is available in the ~~/demo directory. Call this script with the name of the PEI module *before* the module is started. E.g. to debug the PEI module “PeiVariable”:

```
DO go_peim PeiVariable
```

This script sets a breakpoint in the code of the PEICore and waits until the specified PEI module is loaded. Then it sets a breakpoint onto the module entry point and halts there. You can then start debugging the module from scratch.

DXE Phase

After PEI phase completed, it hands off control to the DXECore. To debug the DxeCore from start, load the symbols of “DxeCore” just before PEI jumps into the DxeCore and set a breakpoint at “DxeMain”. DxeMain then starts the DXE dispatcher.

Inspect the DXE resources with the menu items in the “DXE” submenu.

For debugging a DXE driver from its entry point, a special script “go_dxedrv” is available in the ~~/demo directory. Call this script with the name of the DXE module *before* the module is started. E.g. to debug the DXE driver “EventLog”:

```
DO go_dxedrv EventLog
```

This script sets a breakpoint in the DXECore code and waits until the specified DXE module is loaded. Then it sets a breakpoint onto the module entry point and halts there. You can then start debugging the module from scratch.

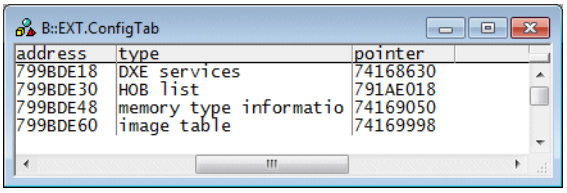
BDS Phase

InsydeH2O implements the BDS phase as DXE driver. To debug the BDS phase, debug the “Bds” module like shown in “**DXE Phase**”.

EXTension.ConfigTab — Display DXE configuration table

Format: **EXTension.ConfigTab**

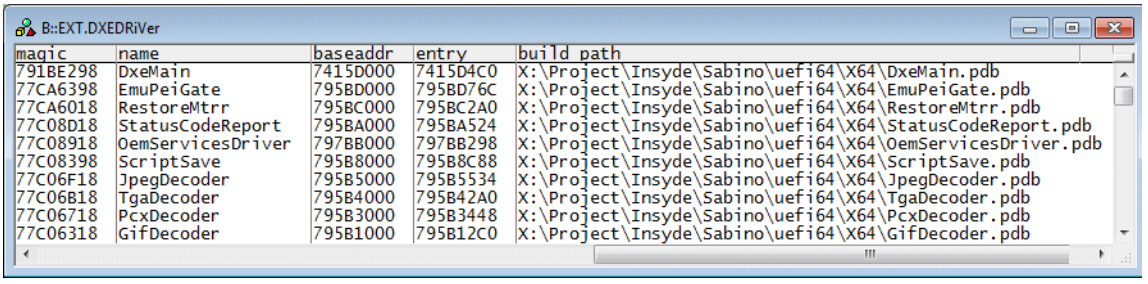
Displays the DXE configuration table.



EXTension.DXEDRiVer — Display loaded DXE drivers

Format: **EXTension.DXEDRiVer**

Displays a table with all DXE drivers that DxeCore already loaded into the system.

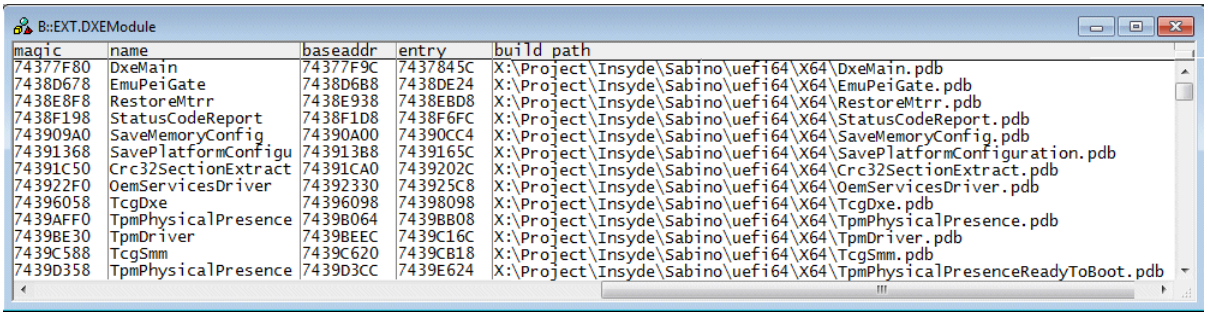


You can sort the window to the entries of a column by clicking on the column header.

“magic” is a unique ID, used by the UEFI Awareness to identify a specific driver.

Format:	**EXTension.DXEModule**

Displays a table with all DXE modules found in the system (firmware volumes or HOBs).



B::EXT.DXEModule

magic	name	baseaddr	entry	build path
74377F80	DxeMain	74377F9C	7437845C	X:\Project\Insyde\Sabino\uefi64\X64\DxeMain.pdb
7438D678	EmuPeiGate	7438D6B8	7438DE24	X:\Project\Insyde\Sabino\uefi64\X64\EmuPeiGate.pdb
7438E8F8	RestoreMtrr	7438E938	7438EBD8	X:\Project\Insyde\Sabino\uefi64\X64\RestoreMtrr.pdb
7438F198	StatusCodeReport	7438F1D8	7438F6FC	X:\Project\Insyde\Sabino\uefi64\X64\StatusCodeReport.pdb
743909A0	SaveMemoryConfig	74390A00	74390CC4	X:\Project\Insyde\Sabino\uefi64\X64\SaveMemoryConfig.pdb
74391368	SavePlatformConfigu	743913B8	7439165C	X:\Project\Insyde\Sabino\uefi64\X64\SavePlatformConfiguration.pdb
74391C50	Crc32SectionExtract	74391CA0	7439202C	X:\Project\Insyde\Sabino\uefi64\X64\Crc32SectionExtract.pdb
743922F0	OemServicesDriver	74392330	743925C8	X:\Project\Insyde\Sabino\uefi64\X64\OemServicesDriver.pdb
74396058	TcgDxe	74396098	74398098	X:\Project\Insyde\Sabino\uefi64\X64\TcgDxe.pdb
7439AFF0	TpmPhysicalPresence	7439B064	7439BB08	X:\Project\Insyde\Sabino\uefi64\X64\TpmPhysicalPresence.pdb
7439BE30	TpmDriver	7439BEEC	7439C16C	X:\Project\Insyde\Sabino\uefi64\X64\TpmDriver.pdb
7439C588	TcgSmm	7439C620	7439CB18	X:\Project\Insyde\Sabino\uefi64\X64\TcgSmm.pdb
7439D358	TpmPhysicalPresence	7439D3CC	7439E624	X:\Project\Insyde\Sabino\uefi64\X64\TpmPhysicalPresenceReadyToBoot.pdb

You can sort the window to the entries of a column by clicking on the column header.

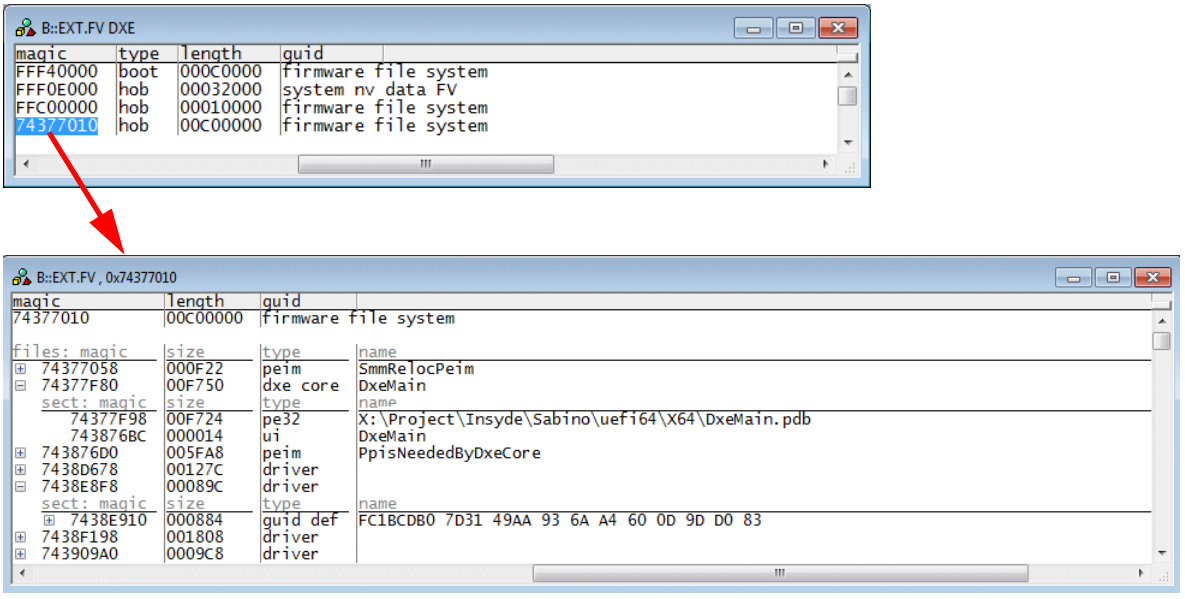
“magic” is a unique ID, used by the UEFI Awareness to identify a specific module.

The “magic” fields are mouse sensitive. Right-click on them to get a local menu. Double-clicking on them opens appropriate windows.

Format:	**EXTension.FV** [**PEI** | **DXE** [<*fv_address*>]]

Displays a table with the firmware volumes of the PEI or DXE phase.

If an address of a firmware volume is specified, the command displays the contents of this FV.



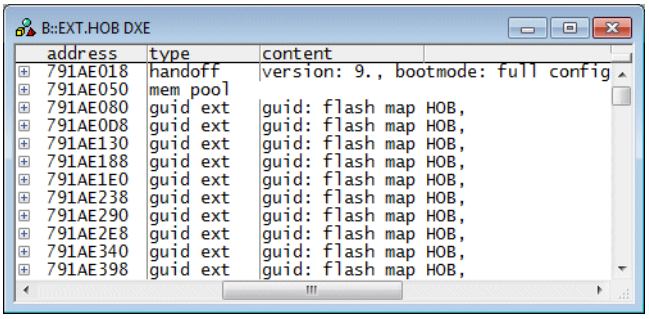
“magic” is a unique ID used by the UEFI Debugger to identify a specific firmware volume or file.

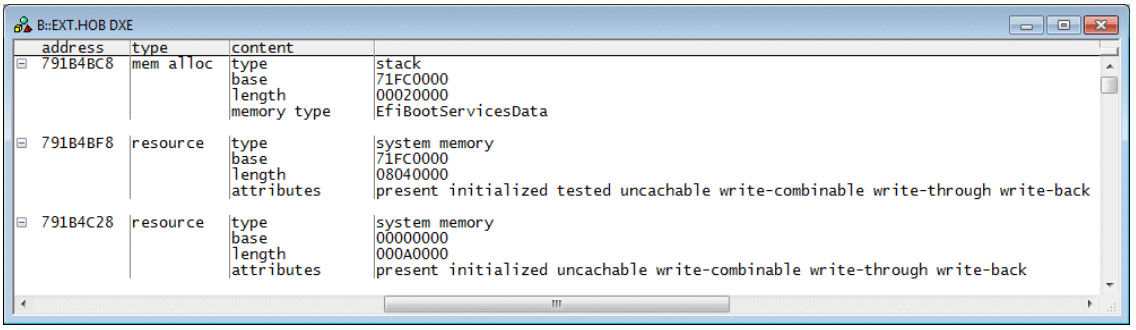
The “magic” fields are mouse sensitive, double clicking on them opens appropriate windows. Right-clicking on them will show a context menu.

The debugger tries to detect the address of the boot firmware volume automatically. If this fails, specify the address of the boot FV manually with the **EXTension.Option BOOTFV** command.

Format:	**EXTension.HOB PEI | DXE**

Displays a table with the hand off blocks of the PEI or DXE phase.





The “address” fields are mouse sensitive, double clicking on them opens appropriate windows. Right clicking on them will show a context menu.

Format:	**EXTension.Option** *<option>*
<option>:	**BOOTFV** *<address>* **PEIHOBS** *<address>* **SYSTABLE** *<address>* **UCODE** *<address>*

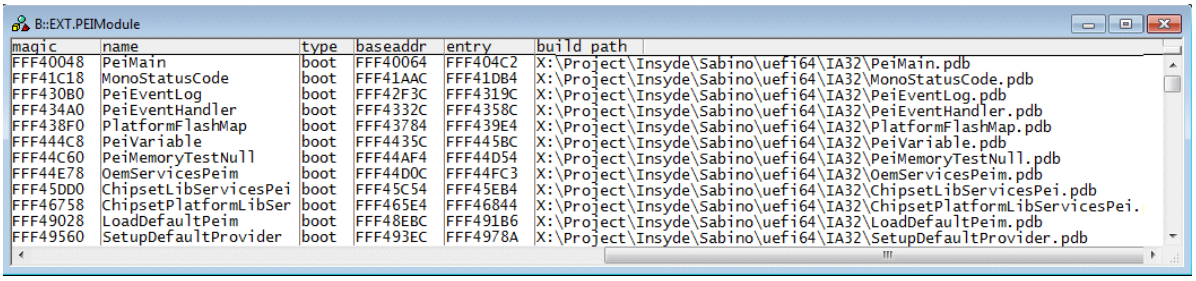
Sets various options to the awareness.

BOOTFV	Set the base address of the boot firmware volume.
PEIHOBS	Set the base address of the HOB list in PEI phase.
SYSTABLE	Set the base address of the EFI System Table
UCODE	Set the base address of the microcode table.

EXTension.PEIModule Display PEI modules

Format:	**EXTension.PEIModule**

Displays a table with all PEI modules found in the system.



magic	name	type	baseaddr	entry	build path
FFF40048	PeiMain	boot	FFF40064	FFF404C2	X:\Project\Insyde\Sabino\uefi64\IA32\PeiMain.pdb
FFF41C18	MonoStatusCode	boot	FFF41AAC	FFF41DB4	X:\Project\Insyde\Sabino\uefi64\IA32\MonoStatusCode.pdb
FFF430B0	PeiEventLog	boot	FFF42F3C	FFF4319C	X:\Project\Insyde\Sabino\uefi64\IA32\PeiEventLog.pdb
FFF434A0	PeiEventHandler	boot	FFF4332C	FFF4358C	X:\Project\Insyde\Sabino\uefi64\IA32\PeiEventHandler.pdb
FFF438F0	PlatformFlashMap	boot	FFF43784	FFF439E4	X:\Project\Insyde\Sabino\uefi64\IA32\PlatformFlashMap.pdb
FFF444C8	PeiVariable	boot	FFF4435C	FFF445BC	X:\Project\Insyde\Sabino\uefi64\IA32\PeiVariable.pdb
FFF44C60	PeiMemoryTestNull	boot	FFF44AF4	FFF44D54	X:\Project\Insyde\Sabino\uefi64\IA32\PeiMemoryTestNull.pdb
FFF44E78	OemServicesPeim	boot	FFF44D0C	FFF44FC3	X:\Project\Insyde\Sabino\uefi64\IA32\OemServicesPeim.pdb
FFF45DD0	ChipsetLibServicesPei	boot	FFF45C54	FFF45EB4	X:\Project\Insyde\Sabino\uefi64\IA32\ChipsetLibServicesPei.pdb
FFF46758	ChipsetPlatformLibSer	boot	FFF465E4	FFF46844	X:\Project\Insyde\Sabino\uefi64\IA32\ChipsetPlatformLibServicesPei.
FFF49028	LoadDefaultPeim	boot	FFF48EBC	FFF491B6	X:\Project\Insyde\Sabino\uefi64\IA32\LoadDefaultPeim.pdb
FFF49560	SetupDefaultProvider	boot	FFF493EC	FFF4978A	X:\Project\Insyde\Sabino\uefi64\IA32\SetupDefaultProvider.pdb

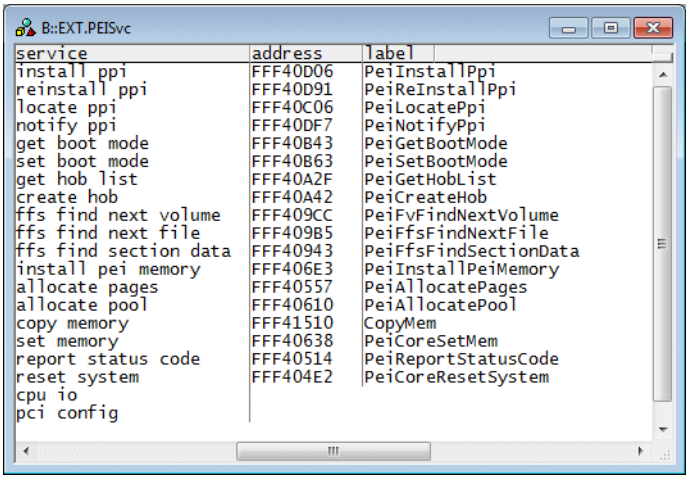
You can sort the window to the entries of a column by clicking on the column header.

“magic” is a unique ID, used by the UEFI Awareness to identify a specific module.

The “magic” fields are mouse sensitive. Right-click on them to get a context menu. Double-clicking on them opens appropriate windows.

Format: **EXTension.PEISvc**

Displays a table with all available PEI services.

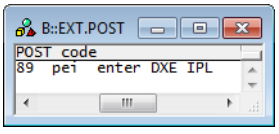


EXTension.POST Display POST code

Format: **EXTension.POST**

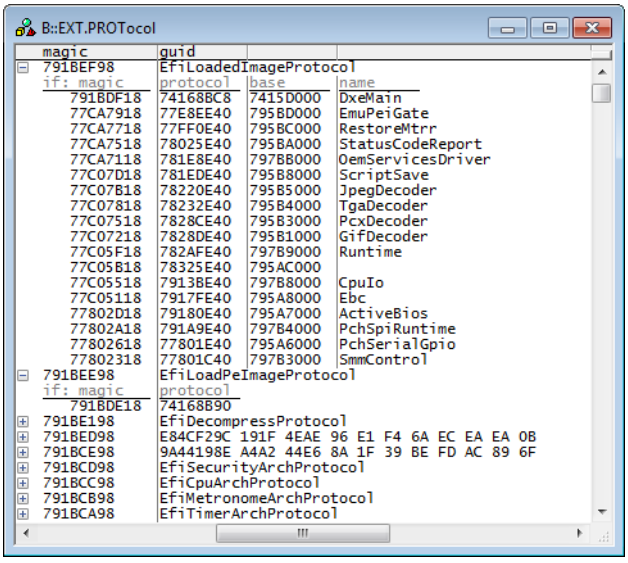
(Only available on x86/x64 targets.)

Displays the Power-On Self-Test code.



Format: **EXTension.PROTocol**

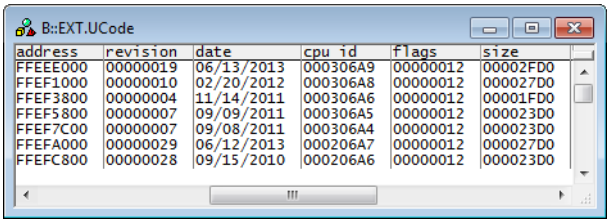
Displays the list of installed DXE protocols.



EXTension.UCode Display microcodes

Format: **EXTension.UCode**

Displays the list of available microcodes.



The debugger tries to detect the address of the microcode list automatically. If this fails, specify the address of the first microcode manually with the command **EXTension.Option UCODE**.

InsydeH2O PRACTICE Functions

There are special definitions for InsydeH2O specific PRACTICE functions.

EXT.DXEDRV.ENTRY() Entry address for DXE driver

Syntax: **EXT.DXEDRV.ENTRY(***<dxedrv_magic>***)**

Returns the entry address for the specified DXE driver.

Parameter Type: Decimal or hex or binary value.

Return Value Type: Hex value.

EXT.DXEDRV.MAGIC() Magic of DXE driver

Syntax: **EXT.DXEDRV.MAGIC("***<dxedrv_name>***")**

Returns the “magic” of the specified loaded DXE driver.

Parameter Type: String (*with* quotation marks).

Return Value Type: Hex value.

EXT.DXEDRV.PATH() Build path for DXE driver

Syntax: **EXT.DXEDRV.PATH(***<dxedrv_magic>***)**

Returns the build path for the specified DXE driver.

Parameter Type: Decimal or hex or binary value.

Return Value Type: String.

EXT.DXEFILE.PATH() Build path for DXE module

Syntax: **EXT.DXEFILE.PATH(**<*dxem_magic*>**)**

Returns the build path for the specified DXE module.

Parameter Type: Decimal or hex or binary value.

Return Value Type: String.

EXT.PEIM.ENTR() Entry addres for PEI module

Syntax: **EXT.PEIM.ENTRY(**<*peim_magic*>**)**

Returns the entry address for the specified PEI module.

Parameter Type: Decimal or hex or binary value.

Return Value Type: Hex value.

EXT.PEIM.MAGIC() Magic of PEI module

Syntax: **EXT.PEIM.MAGIC("**<*peim_name*>**")**

Returns the “magic” of the specified PEI module.

Parameter Type: String (*with* quotation marks).

Return Value Type: Hex value.

EXT.PEIM.PATH() Build path for PEI module

Syntax: **EXT.PEIM.PATH(**<*peim_magic*>**)**

Returns the build path for the specified PEI module.

Parameter Type: Decimal or hex or binary value.

Return Value Type: String.