



Peripheral Files Programming Commands

[TRACE32 Online Help](#)

[TRACE32 Directory](#)

[TRACE32 Index](#)

TRACE32 Documents	
Peripheral Files	
Peripheral Files Programming Commands	1
History	4
Introduction	4
Peripheral File General Description	4
Passing Arguments	9
Memory Classes	12
Comma-Separated-Values (CSV) File Format for *.per Files	13
Editing a *.per File in CSV Format in a Spreadsheet Editor	14
Mixing Regular and CSV Formats	17
GROUP Commands	18
GROUP	Define read/write GROUP 18
HGROUP	Define read-once/write GROUP 20
RGROUP	Define read-only GROUP 20
WSGROUP	Define write-only and shadow GROUP 21
WGROUP	Define write-only GROUP 22
SGROUP Commands	23
SGROUP	Define sequence GROUP 23
SET	Write constant value to memory 25
SETX	Write SGROUP buffer to memory 26
GETX	Read from memory to the SGROUP buffer 27
CONSTX	Write constant value to the SGROUP buffer 28
VARX	Write expression to SGROUP buffer 29
WRITEBACK	Separate write a part from a read part 30
Other Top Level Commands	32
ASSERT	Abort if condition not met 32
AUTOINDENT	Indent content of peripheral file automatically 32
BASE	Define a base address for following group definitions 39
BASEOUT	Output a value before calculating a base address 39
BASESAVEOUT	Output a value before calculating a base address 41
CONFIG	Configure default access width and line break for BIT 41
CSV	Enables CSV capabilities 43

ELSE	Conditional GROUP display	44
ELIF	Conditional GROUP display	44
ENDIAN	Define little or big endian	44
ENDIF	Conditional GROUP display	44
ENTRY	Assign parameters to macros	44
HELP	Reference online manual	45
IF	Conditional GROUP display	45
INCLUDE	Include another peripheral file	47
PERCMD	Row definition in CSV-formatted *.per file	48
SIF	Conditional interpretation	50
TREE	Define hierarchic display	51
WIDTH	Width of register names and a BIT description	51
WAIT	Wait with PER windows until system is ready	52
Commands within GROUPs		53
ASCII	Display ASCII character	53
BIT	Define bits	53
BITFLD	Define bits individually	54
BUTTON	Define command button	58
COPY	Copy GROUP	59
DECMASK	Define bits for decimal display	60
FLOATMASK	Define bits for decimal floating point display	61
EVENTFLD	Define event flag bits individually	62
HEXFLD	Define hexword individually	63
HEXMASK	Define bits for a hexadecimal display	64
HIDE	Define write-only line	65
IN	Define input field	66
INDEX	Output a value	66
LINE	Define line	68
MUNGING	Translate to little endian mode (PowerPC only)	69
NEWLINE	Line break within detailed register description	69
RBITFLD	Define bits individually (read-only)	70
RHEXMASK	Define bits for a hexadecimal display (read-only)	70
SAVEINDEX	Save original and output a value	71
SAVETINDEX	Save original and output a value	72
SETCLRFLD	Define set/clear locations	72
STRING	Display a string saved in memory	73
SYSCON	SYSCON register (C166/ST10 only)	74
TEXTLINE	Define text header with a new line	74
TEXTFLD	Define text header	74
TINDEX	Output a value	75
Functions		76
History		77

History

- | | |
|-----------|---|
| 09-Apr-20 | New command "RHEXMASK". |
| 27-May-19 | New command "LINE.FLOAT". |
| 17-Jul-18 | Added section " Comma-Separated-Values (CSV) File Format for *.per Files " to describe the use of the commands CSV and PERCMD by way of an example. |
| 11-Jul-18 | Added description and screenshot examples for the new command CSV and description for the new command PERCMD . |
| 29-Jun-18 | Added description and examples for the new statement AUTOINDENT . |

Introduction

This document describes the commands which are used to write peripheral files. This allows to display/manipulate configuration registers and the on-chip peripheral registers at a logical level. Registers and their contents are visible and accessible in the [PER.view](#) window.

Peripherals in MCU can be displayed and manipulated with the [PER](#) commands. TRACE32 offers configurable window for displaying memory or I/O structures. Displaying the state of peripheral components or memory based structures is very comfortable.

User can define 'chip macros' and put them together to generate 'project files'. These files describe the port structure for a specific hardware system.

Examples for different microcontrollers reside in the directory `~/demo/per/`.

Peripheral File General Description

To start writing the peripheral file, please create a file with extension *.per.
".per" is the TRACE32 standard extension for peripheral files.

The syntax of a peripheral file is line oriented. Blanks and empty lines can be inserted to define the structure of the program. Comment lines start with semicolon.

Examples of the peripheral file reside in the directory `~/demo/per`.

At the beginning of the file, the commands **WIDTH** and **CONFIG** should be placed. The next step is to define the base address using **BASE** command. Each implemented module has to be started with **TREE** command and ended with the **TREE.END** command.

A typical peripheral file implementation is showed below:

```
; "dots" mean decimal format
CONFIG 16. 8.

; 0x means hex format
WIDTH 0xb

; "Treeview" of the module
TREE "Module Registers"

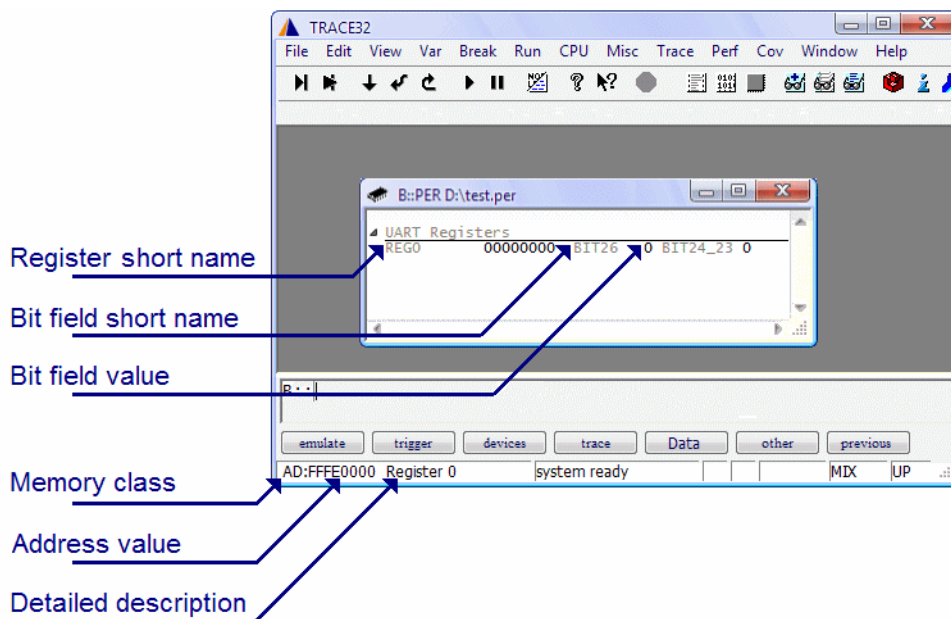
; base address of the module
BASE ad:0xf0000000

; GROUP definition
GROUP.LONG 0x00++0x3
; register definition
LINE.LONG 0x00 "REG0,Register 0"

; one bit filed definition
BITFLD.LONG 0x00 26. " BIT26 ,Bit 26" "0,1"

; 2-bit field definition
BITFLD.LONG 0x00 23.--24. " BIT24_23 ,Bits 24 to 23" "0,1,2,3"

; end of the tree
TREE.END
```



```

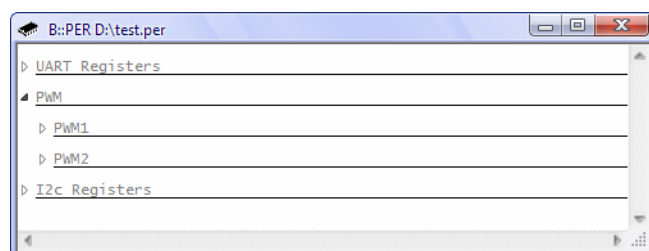
TREE "UART Registers"
  BASE ad:0xfffe0000
  GROUP.LONG 0x00++0x3
    LINE.LONG 0x00 "REG0,Register 0"
      BITFLD.LONG 0x00 26. " BIT26 ,Bit 26" "0,1"
      BITFLD.LONG 0x00 23.--24. " BIT24_23 ,Bits 24 to 23" "0,1,2,3"
      BITFLD.LONG 0x00 26. " BIT17 ,Bit 17" "0,1"
TREE.END

TREE.OPEN "PWM"
  TREE "PWM1"
    BASE ad:0xfffe1000
    GROUP.LONG 0x00++0x3
      LINE.LONG 0x00 "REG1,Register 1"
        BITFLD.LONG 0x00 19. " BIT19 ,Bit 19" "0,1"
        BITFLD.LONG 0x00 14.--15. " BIT15_14 ,Bits 15 to 14" "0,1,2,3"
  TREE.END
  TREE "PWM2"
    BASE ad:0xfffe2000
    GROUP.LONG 0x00++0x3
      LINE.LONG 0x00 "REG2,Register 2"
        BITFLD.LONG 0x00 8. " BIT8 ,Bit 8" "0,1"
        BITFLD.LONG 0x00 5.--6. " BIT6_5 ,Bits 6 to 5" "0,1,2,3"
  TREE.END
TREE.END

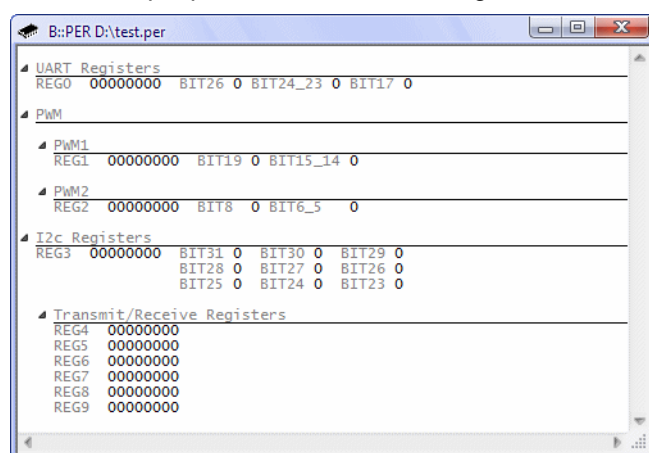
TREE "I2c Registers"
  BASE ad:0xfffe3000
  GROUP.LONG 0x00++0x3
    LINE.LONG 0x00 "REG3,Register 3"
      BITFLD.LONG 0x00 31. " BIT31 ,Bit 31" "0,1"
      BITFLD.LONG 0x00 30. " BIT30 ,Bit 30" "0,1"
      BITFLD.LONG 0x00 29. " BIT29 ,Bit 29" "0,1"
      TEXTLINE " "
      BITFLD.LONG 0x00 28. " BIT28 ,Bit 28" "0,1"
      BITFLD.LONG 0x00 27. " BIT27 ,Bit 27" "0,1"
      BITFLD.LONG 0x00 26. " BIT26 ,Bit 26" "0,1"
      TEXTLINE " "
      BITFLD.LONG 0x00 25. " BIT25 ,Bit 25" "0,1"
      BITFLD.LONG 0x00 24. " BIT24 ,Bit 24" "0,1"
      BITFLD.LONG 0x00 23. " BIT23 ,Bit 23" "0,1"
  TREE "Transmit/Receive Registers"
    GROUP.LONG 0x10++0x17
      LINE.LONG 0x00 "REG4,Register 4"
      LINE.LONG 0x04 "REG5,Register 5"
      LINE.LONG 0x08 "REG6,Register 6"
      LINE.LONG 0x0c "REG7,Register 7"
      LINE.LONG 0x10 "REG8,Register 8"
      LINE.LONG 0x14 "REG9,Register 9"
  TREE.END
TREE.END

```

Peripheral modules are organized in a tree structure.



Contents of peripheral modules is also organized in a tree structure.



Passing Arguments

You can pass arguments from a PRACTICE script to a PER file (peripheral file). These arguments can be strings, hex and decimal values. See below for an [example](#) and an [illustration and explanation of the example](#).

Example

PRACTICE script (*.cmm) - Bold and red are used to highlight the information flow:

```
;Declare four PRACTICE macros and assign values to the PRACTICE macros
LOCAL &addr &reg64bit &name &idx

&addr=0xE0000000    ;Base address of the PER file called with PER.view.
&reg64bit=1.        ;Show the 64bit or the 32bit specific register group.
&name="My Module" ;Module description of the register group.
&idx=35.            ;Show a specific register out of an array of
                    ;memory-mapped registers.

;... your code
SYStem.Up

;View the peripheral file and pass the four arguments
PER.view "per_with_args.per" &addr &reg64bit "&name" &idx "*"

;Open the peripheral file in the built-in TRACE32 editor PER.Program
PER.Program "per_with_args.per"    ;Do not pass arguments here!
;Set a different peripheral file as temporary new default file
PER.ReProgram "per_with_args.per"  ;Do not pass arguments here!
```

The above PRACTICE script (*.cmm) calls this PER file (*.per):

```
CONFIG 16. 8.
WIDTH 10.

;The PER.view command arguments are passed to the ENTRY command arguments
ENTRY &baseaddr=0x0 &reg64bit=0. &modulename="foo" &index=1.

BASE D:&baseaddr

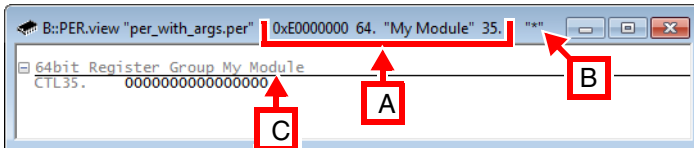
SIF (&reg64bit==1.)
    TREE "64bit Register Group &modulename"
        GROUP.QUAD (0x8*&index)++0x07
            LINE.QUAD 0x00 "CTL&index,Control Register &index"
        TREE.END
ELSE
    TREE "32bit Register Group &modulename"
        GROUP.LONG (0x4*&index)++0x07
            LINE.LONG 0x00 "CTL&index,Control Register &index"
        TREE.END
ENDIF
```

NOTE:

Although the **ENTRY** command arguments may look like PRACTICE macros, they are *not* PRACTICE macros and do *not* behave like PRACTICE macros:

- When you try to create PRACTICE macros with the **LOCAL** command inside a PER file and compile it, you receive the error message “unknown command”.
- When you try to assign an **ENTRY** command argument to another **ENTRY** command argument (`&arg2=&arg1`) inside a PER file and compile it, you also receive the error message “unknown command”.

Our example produces this **PER.view** window:



A The four values passed to the PER file are displayed in the window caption.

B " * " displays all branches. For more information, see **PER.view**.

C Result of the information flow highlighted in bold and red in the above example (see `&name`).

NOTE:

In the PER file, valid default values must be assigned to each **ENTRY** command argument. See highlighted values in the **ENTRY** line.

The default values in the **ENTRY** line ensure that no “syntax error” is reported when a PER file is compiled in the built-in TRACE32 editor **PER.Program**.

```
;Define default values for the ENTRY command arguments
ENTRY &baseaddr=0x0 &reg64bit=0. &modulename="foo" &index=0.
```

As valid default values in a PER file, our example uses:

- `0x0` for hex values.
- `0.` for decimal values.
- `"foo"` for strings.

When the PRACTICE macro values are passed to the same PER file, the passed values override the default values in the **ENTRY** line of the PER file.

Memory Classes

Format: *<class>:<base_address>*

<class> Appropriate access method to memory class (**D, SD, A, AD, AP, ANC, DC, IC, NC, ED, EAD, VM, P, etc.**)

<base_address> Base address of the peripheral module.

Refer to the [“General Commands Reference Guide D” - Memory Classes](#).

Peripheral files can be formatted as comma-separated values, i.e. the same format as in *.csv files. However, the file extension for peripheral files remains *.per, as usual. The CSV format extends the regular peripheral command set and offers you an alternative way to create and maintain peripheral files more easily in a spreadsheet. Therefore it usually offers better readability. Peripheral files in CSV format can also be generated more easily from binary files (such as netlists, etc.) by automated tools.

Example: Regular *.per file format (excerpt from ~/demo/per/percsv_nocsv.per):

```
TREE "Common Registers"
GROUP 0xE80++0x01
LINE.WORD 0x0 "ADCR1,ADC Control Register 1"
BITFLD.WORD 0 14. "STOP,Stop", "Normal operation,Stop"
BITFLD.WORD 0 13. "START,Start Conversion", "No action,Start"
BITFLD.WORD 0 12. "SYNC,Sync Select", "START bit,sync input or START bit"
GROUP 0xF80++0x01
LINE.WORD 0x0 "ADCR2,ADC Control Register 2"
HEXMASK.WORD.BYTE 0 0.--3. 1. "DIV,Clock Divisor Select"
TREE.END
```

The same register definitions in CSV format and displayed in a spreadsheet editor (excerpt from ~/demo/per/percsv_simple.per):

PERCMD	Address	AccessWidth	Name	Tooltip	From	To	Choices
TREE "Common Registers"							
	0xe80	16.	ADCR1	ADC Control Register 1			
			STOP	Stop	14.	14.	Normal operation,Stop
			START	Start Conversion	13.	13.	No action,Start
			SYNC	Sync Select	12.	12.	START bit,sync input or START bit
	0xf80	16.	ADCR2	ADC Control Register 2			
			DIV	Clock Divisor Select	0.	3.	
TREE.END							

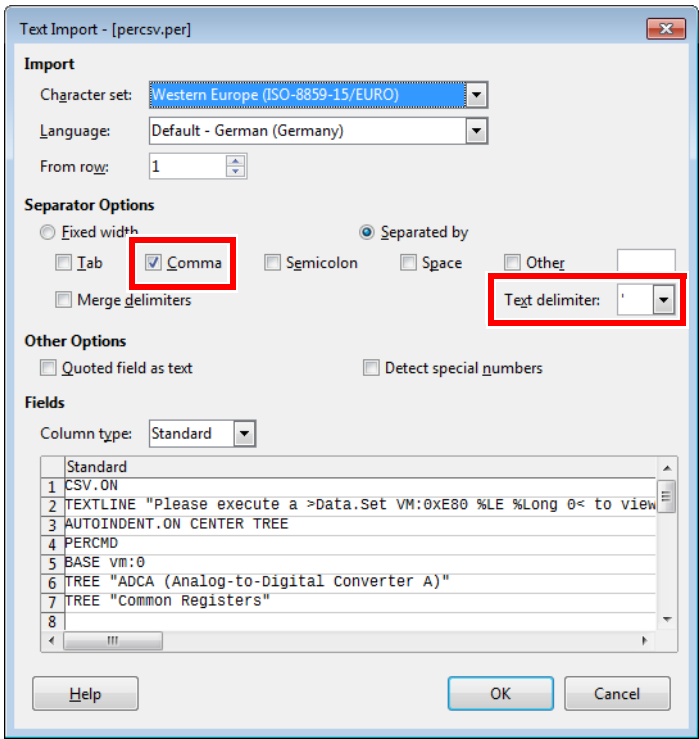
Whenever necessary, you can still mix the regular and CSV file format.

NOTE:

Microsoft Excel is not capable of exporting true comma-separated-values files on machines based in Europe (instead semicolons will be used as separators due to system-wide Region and Language settings). Therefore it is recommended to use LibreOffice Calc or any other spreadsheet editor.

Editing a *.per File in CSV Format in a Spreadsheet Editor

- 1. Do one of the following:
 - Create an empty file, or
 - Open/Import an existing *.per file. Make sure comma is selected as separator and the single quote as text delimiter:



- 2. The first command in the *.per file (except comments) must enable CSV capabilities:

```
CSV.ON
```

- 3. Optional step: Use your preferred auto-indent style (see [AUTOINDENT](#)):

```
AUTOINDENT.ON CENTER TREE
```

- 4. Optional step: Define the columns (see [PERCMD](#)).

- The column name arguments of the [PERCMD](#) command will serve as column headers in your spreadsheet, see [X] below.

CSV.ON									
AUTOINDENT.ON CENTER TREE									
PERCMD	X	Address	AccessWidth	Name	Tooltip		From	To	Choices
BASE x:0									
TREE "ADCA (Analog-to-Digital Converter A)"									

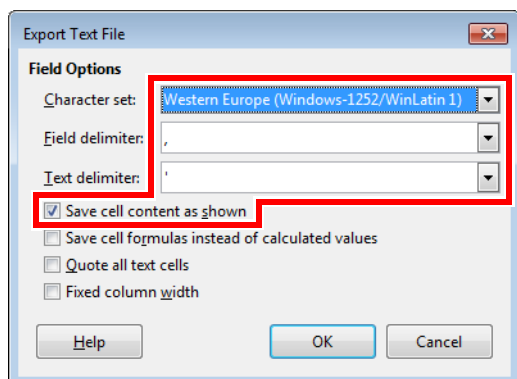
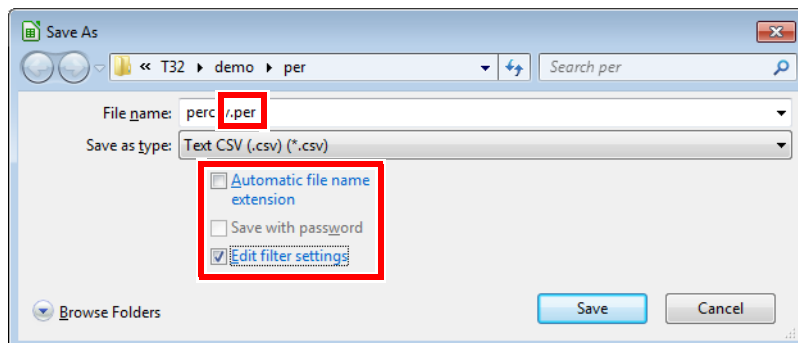
- To freeze the headers, choose **View** menu > **Freeze Rows and Columns**.
- If you omit the [PERCMD](#) command: The first column must always contain peripheral file commands only and must be kept empty otherwise!

5. Optional step: Use **BASE** and **TREE** commands in the subsequent rows to create an environment.
6. Define the registers and bits:

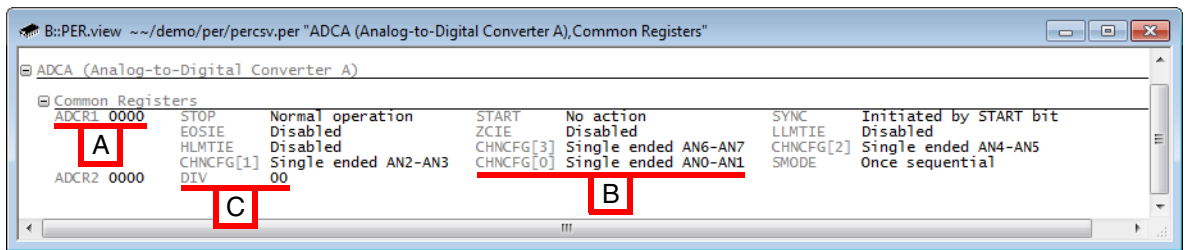
CSV.ON							
AUTOINDENT.ON CENTER TREE							
PERCMD	Address	AccessWidth	Name	Tooltip	From	To	Choices
BASE x:0							
TREE "ADCA (Analog-to-Digital Converter A)"							
TREE "Common Registers"							
	A → 0xe80	16.	ADCR1	ADC Control Register 1			
			STOP	Stop	14.	14.	Normal operation, Stop
			START	Start Conversion	13.	13.	No action, Start
			SYNC	Sync Select	12.	12.	Initiated by START bit
NEWLINE							
			EOSIE	End-of-Scan Interrupt Enable	11.	11.	Disabled, Enabled
			ZCIE	Zero Crossing Interrupt Enable	10.	10.	Disabled, Enabled
			LLMTIE	Low Limit Interrupt Enable	9.	9.	Disabled, Enabled
NEWLINE							
			HLMTIE	High Limit Interrupt Enable	8.	8.	Disabled, Enabled
			CHNCFG[3]	Channel Configure AN6-AN7	7.	7.	Single ended AN6-AN
			CHNCFG[2]	Channel Configure AN4-AN5	6.	6.	Single ended AN4-AN
NEWLINE							
			CHNCFG[1]	Channel Configure AN2-AN3	5.	5.	Single ended AN2-AN
			CHNCFG[0]	Channel Configure AN0-AN1	4.	4.	Single ended AN0-AN
			SMODE	ADC Mode Control	0.	2.	Once sequential, Once
	0xe82	16.	ADCR2	ADC Control Register 2			
			DIV	Clock Divisor Select	0.	3.	
TREE.END							
TREE.END							

A to C For a description, see **Rules** below.

7. When done, save/export the spreadsheet in CSV format as shown below:



Output:



A to C For a description, see [Rules](#) below.

Rules:

- A new register [**A**] will be created if at least one of the following conditions applies:
 - The **Address** value is the first non-empty entry in the spreadsheet.
 - The **Address** value differs from the previous one.
 - The **AccessWidth** value differs from the previous one.
 - **From** and **To** values are empty.
- A new customized bit description [**B**] will be created if the following conditions are all true:
 - The **Address** value does not change, or the entry is empty.
 - The **AccessWidth** value does not change, or the entry is empty.
 - The **Choices** value is not empty.
- A new bit or bit range [**C**] is displayed as hexadecimal if the following conditions are all true:
 - The **Address** value does not change, or the entry is empty.
 - The **AccessWidth** value does not change, or the entry is empty.
 - The **Choices** value is empty.

Mixing Regular and CSV Formats

In order to simplify matters, peripheral files in CSV format do not offer the full functional range of regular *.per files. However, you can easily include regular *per commands in the first column:

Excerpt from ~/demo/per/percsv_mixed.per:

PERCMD	Address	AccessWidth	Name	Tooltip	From	To	Choices
TREE "Common Registers"							
	0xe80	16.	ADCR1	ADC Control Register 1			
IF (Data.Long(vm:0xe80)==0x0)							
			STOP	Stop	14.	14.	Normal operation, Stop
			START	Start Conversion	13.	13.	No action, Start
			SYNC	Sync Select	12.	12.	START bit, sync input or START bit
ELSE							
			EOSIE	End-of-Scan Interrupt Enable	11.	11.	Disabled, Enabled
			ZCIE	Zero Crossing Interrupt Enable	10.	10.	Disabled, Enabled
			LLMTIE	Low Limit Interrupt Enable	9.	9.	Disabled, Enabled
ENDIF							
NEWLINE							
			HLMTIE	High Limit Interrupt Enable	8.	8.	Disabled, Enabled
			CHNCFG[3]	Channel Configure AN6-AN7	7.	7.	Single ended AN6-AN7, AN6 + and AN7 -
			CHNCFG[2]	Channel Configure AN4-AN5	6.	6.	Single ended AN4-AN5, AN4 + and AN5 -
	0xf80	16.	ADCR2	ADC Control Register 2			
			DIV	Clock Divisor Select	0.	3.	
TREE END							

In above example we utilize the regular peripheral commands **TREE**, **IF** and **NEWLINE**. In all other cases, the first column must remain empty!

GROUP Commands

The **GROUP** commands describe how data is basically read or written to/from memory.

GROUP

Define read/write GROUP

Format:	GROUP. <size> <datagr> <fifogroup> ["<name>"]
<datagr>:	<address>++<number_of_read_bytes-1> or <start_address>--<end_address>
<fifogroup>:	<address> <address_range>

The GROUP commands control the debugger access to the target memory.

<size> Size of registers (**Byte**, **Word**, **TByte**, **Long**, **Quad**) or **auto**.

<name> Optional text.

If a name is given, the GROUP is separated from the previous lines and the name is used as headline in the per window. Using numerical values (without memory access class) in address parameter, the address is calculated by the entered value plus the base address (defined by the last **BASE** command). The GROUP can either use normal memory access or fifo access (reads all bytes from the same address). The whole address range of the GROUP command is read at once. Reading from reserved address range may cause a bus error.

Example 1:

```
BASE ud:0x200                ;data bytes at address sd:0x100--0x101
GROUP sd:0x100--0x101 "PortA"

GROUP 0x50--0x51             ;data bytes at address ud:0x250--0x251

GROUP.LONG sd:0x60--0x6f     ;read memory with 32-bit access

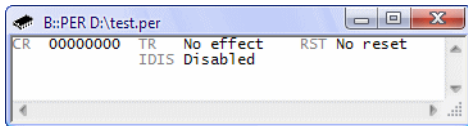
GROUP sd:0x300 0x10          ;fifo at location sd:0x300, 16 bytes
                             ;deep

GROUP 0x10 0x4               ;fifo at ud:0x210, 4 bytes deep
```

```

BASE ad:0x00000000
GROUP 0x00++0x03
    LINE.LONG 0x00 "CR,Control Register"
        BITFLD.LONG 0x00 24. " TR    ,Transfer" "No effect,Transferred"
        BITFLD.LONG 0x00 5.  " RST  ,Software Reset" "No reset,Reset"
        TEXTLINE "
        "
        BITFLD.LONG 0x00 1.  " IDIS ,Interrupt Enable" "Disabled,Enabled"

```

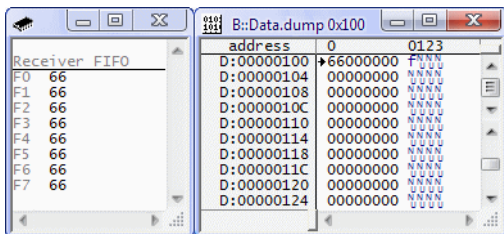


Example 2:

```

BASE ad:0x00000000
GROUP.BYTE 0x100 0x8 "Receiver FIFO"
    LINE.BYTE 0x0 "F0,FIFO position 0"
    LINE.BYTE 0x1 "F1,FIFO position 1"
    LINE.BYTE 0x2 "F2,FIFO position 2"
    LINE.BYTE 0x3 "F3,FIFO position 3"
    LINE.BYTE 0x4 "F4,FIFO position 4"
    LINE.BYTE 0x5 "F5,FIFO position 5"
    LINE.BYTE 0x6 "F6,FIFO position 6"
    LINE.BYTE 0x7 "F7,FIFO position 7"

```



Format:	HGROUP. <size> <datagr> <fifogroup>["<name>"]
<datagr>:	<address>++<number_of_read_bytes-1> or <start_address>--<end_address>
<fifogroup>:	<address> <address_range>

Similar to GROUP, but this definition is useful for ports which are cleared by a read access. Refer to the **GROUP** command description. HGROUP command prevents target memory from the periodic read access and is useful for 'write-only' ports. In hidden GROUPs only hidden elements e.g. **HIDE** command should be used.

<size>	Size of registers (byte, word, tbyte, long, quad).
<name>	Optional text.

RGROUP

Define read-only GROUP

Format:	RGROUP. <size> <datagr> <fifogroup> ["<name>"]
<datagr>:	<address>++<number_of_read_bytes-1> or <start_address>--<end_address>
<fifogroup>:	<address> <address_range>

Similar to GROUP, but this definition is useful for 'read-only' ports. Refer to the **GROUP** command description.

<size>	Size of registers (Byte , Word , TByte , Long , Quad).
<name>	Optional text.

Format: **WSGROUP.<size> <wr_acc_addr> <rd_acc_addr>**

WSGROUP is a specific GROUP command, which forces the debugger to access different registers for read and for write accesses. It is only useful, if the core has write-only registers and their contents are duplicated in shadow registers, which are read- and writable.

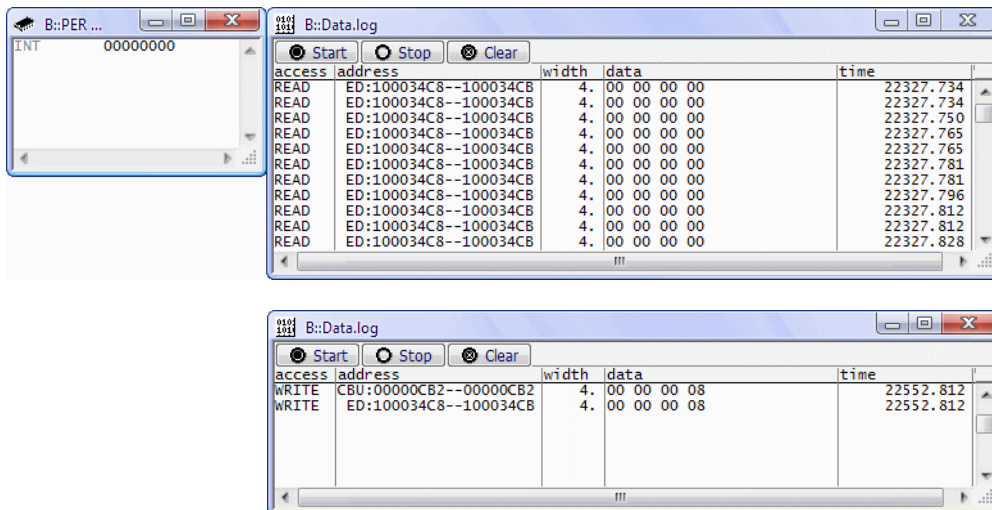
<size> Size of registers (byte, word, tbyte, long, quad).
<wr_acc_addr> Address of the register where data is to be written into.
<rd_acc_addr> Address of the register where data is to be read from.

Read/write accesses have following effects:

- **write access:** Data is written to write-only registers (dataGROUP) as well as to the shadow registers.
- **read access:** Data is read from the shadow registers.

Example:

```
WSGROUP.LONG (ecbu:0x0CB2)++0 (ed:0x100034C8)
LINE.LONG 0x0 "INT,Self-interrupt register"
```



Format:	WGROUP. <size> <datagr> <fifogroup>["<name>"]
<datagr>:	<address>++<number_of_read_bytes-1> or <start_address>--<end_address>
<fifogroup>:	<address> <address_range>

Similar to GROUP command. This definition is useful for 'write-only' ports. The current state of the port is held in the emulation memory (must be mapped at this location). Refer to the **GROUP** command description.

<size>	Size of registers (byte, word, tbyte, long, quad).
<name>	Optional text.

Example:

```
WGROUP sd:0x50--0x51      ;the port at address sd:0x50--0x51
                           ;is a write-only port (e.g. 74xx374)
                           ;but the state can be read via
                           ;dual-port access
```

SGROUP

Define sequence GROUP

Format: **SGROUP** ["<name>"]

Sequence of memory accesses done to get/set the data.

<name> Optional text.

Usually GROUP commands specify the target memory accesses and the following commands e.g. BITFLD, HEXMASK, etc. define how the data are displayed in the per window.

With SGROUP data is not accessed with **SGROUP** itself, but by a sequence of special commands, which transfer data from memory to the “SGROUP data buffer” or from the “SGROUP data buffer” back to memory. The size of the buffer is 256 bytes.

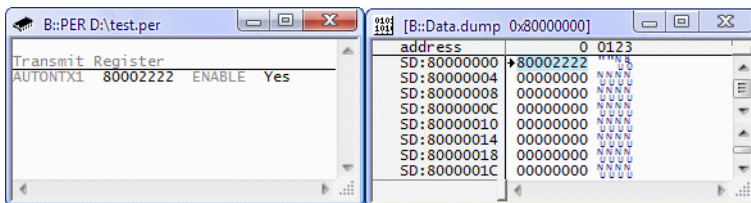
Afterwards this sequence of special commands the data in the buffer can be displayed by following commands e.g. BITFLD, HEXMASK.

To read/write data from/to memory to/from SGROUP buffer you can use the following commands (which are only allowed in **SGROUPs**):

Command	Function
SET <address> %<format> <value>	Constant value --> memory(address)
SETX <address> %<format> <index>	Buffer(index) --> memory(address)
GETX <address> %<format> <index>	Memory(address) --> buffer(index)
CONSTX <index> %<format> <value>	Constant value --> buffer(index)
VARX <index> %<format> <expression>	Variable value --> buffer(index)
WRITEBACK	Separate write part from a read part

Example:

```
SGROUP "Transmit Register"                                ; define sequence GROUP
GETX d:0x80000000 %l 0                                    ; read data at 0x80000000 and store
                                                         ; them in buffer + offset 0
WRITEBACK                                                  ; next commands only done for
CONSTX 2 %w 0x2222                                         ; per.set
                                                         ; write 0x2222 to buffer + offset 2
SETX d:0x80000000 %l 0                                    ; write data from buffer + offset 0
                                                         ; to memory at 0x80000000
LINE.LONG 0x0                                              ; display AUTONTX1 register with
"AUTONTX1,Autonegotiation Next"                           ; contents of buffer[0...3]
Page Transmit Register 1"
BITFLD.LONG 0 31. "ENABLE" "No,Yes"                      ; define bit "Enable"
```



Format: **SET** <address> %<format> <value>

SET command writes data to memory.

The given value is written to the target memory at the specified address or at the base address with added offset. The specified value is written continuously.

<address> Target address.

<format> Defines specific format (**Byte, Word, TByte, Long, Quad, LE, BE**).

<value> Constant value.
The value may be a hexadecimal or mask or binary mask. (E.g.:
0yxxxx10xx)

Command is only allowed in **SGROUP**.

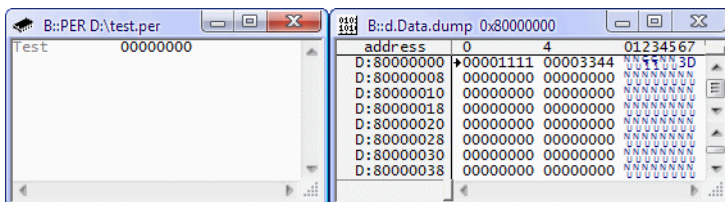
Example:

```

BASE d:0x80000000          ; set base address to d:0x80000000
SGROUP                     ; define sequence GROUP
SET d:0x80000000 %l 0x1111 ; write 0x1111 to d:80000000
SET 4 %l 0x3344             ; write 0x3344 to base address
                           ; (d:80000000) + offset 4

LINE.LONG 0x0 "Test,Test Register"

```



Format: **SETX** <address> %<format> <index>

SETX command writes a buffered value to the memory.

A value stored in a buffer at the given buffer offset is written to the target memory at the specified address or base address with added offset. The value is written only once.

<address> Target address.
 <format> Defines specific format (**Byte, Word, TByte, Long, Quad**).
 <index> Constant value.

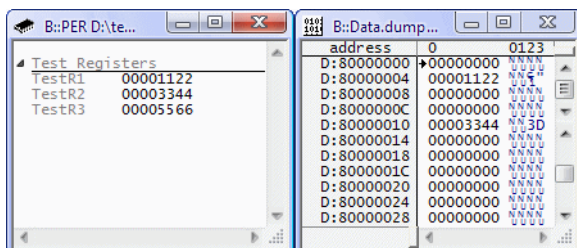
Command is only allowed in **SGROUP**.

Example:.

```
CONFIG 16. 8.
WIDTH 10.
BASE 0x80000000
TREE "Test Registers"
;write into buffer : 0x1122 at offet [0], 0x3344 at [4], 0x5566 at [8]
SGROUP

CONSTX 0 %1 0x1122
CONSTX 4 %1 0x3344
CONSTX 8 %1 0x5566
    LINE.LONG 0x0 "TestR1,Test Register 1"
    LINE.LONG 0x4 "TestR2,Test Register 2"
    LINE.LONG 0x8 "TestR3,Test Register 3"

;write buffer contents into target memory : [0..3] at 0x80000004,...
SETX 4    %1 0
SETX 0x10 %1 4
TREE.END
```



Format: **GETX** <address> %<format> <index>

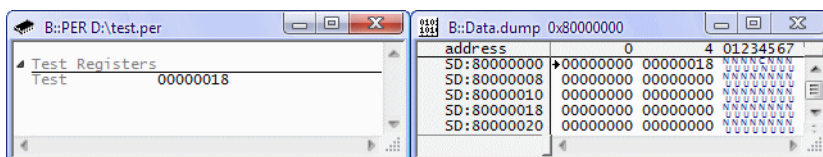
GETX command reads data from the memory and puts it to the buffer. The memory contents from the given address is read using specified access width format. The read data is stored in a buffer at the defined offset.

<address> Target address equals base address + offset.
 <format> Defines specific format (**Byte**, **Word**, **TByte**, **Long**, **Quad**).
 <index> Defines buffer number.

Command is only allowed in **SGROUP**.

Example:

```
BASE d:0x80000000
TREE "Test Registers"
SGROUP                                ; define sequence GROUP
SET d:0x80000004 %l 0x18             ; write value 0x18 to target memory
                                      ; at d:80000004
GETX 4 %l 0                          ; read out target memory at base
                                      ; address d:80000000+offset 4 and
                                      ; store it at buffer+offset 0
LINE.LONG 0x0 "Test,Test Register"   ; display data of buffer[0...3]
TREE.END
```



Format: **CONSTX** <index> %<format> <value>

CONSTX command writes a constant value to the buffer. This data is **not** written to the target memory. The data can be displayed with a following line command.

<index> Defines indexed offset.

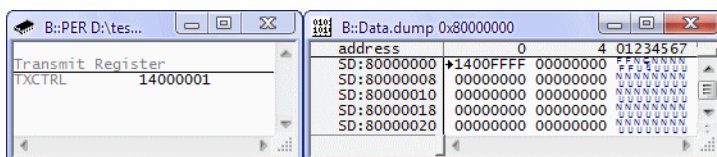
<format> Defines specific format (**Byte, Word, TByte, Long, Quad, LE, BE**).

<value> Defines a constant value.
The value may be a hexadecimal or mask or binary mask. (E.g.: 0yxxxx10xx)

Command is only allowed in **SGROUP**.

Example:

```
SGROUP "Transmit Register"           ; define sequence GROUP
SET 0x80000000 %l 0x1400ffff         ; write value 1400ffff to target
                                     ; memory at d:80000000
GETX d:0x80000000 %l 0x00            ; read out target memory at 80000000
                                     ; and store it at buffer + offset 0
CONSTX 2 %w 0x1                     ; write 0x0001 at buffer + offset 2
LINE.LONG 0x0 "TXCTRL,Transmit      ; display data of buffer[0...3]
Control Register"
```



Format:	VARX <i><index></i> % <i><format></i> <i><expression></i>
---------	--

VARX command writes a variable value to the SGROUP buffer. This data is **not** written to the target memory. The data can be displayed with a following line command.

<i><index></i>	Defines indexed offset.
<i><format></i>	Defines specific format (Byte , Word , TByte , Long , Quad , LE , BE).
<i><expression></i>	Defines a PRACTICE expression. The expression will be parsed whenever the PER window updates and its result will be assigned to the SGROUP buffer

The **VARX** command is very similar to the **CONSTX** command. However the value, which should be assigned to the **SGROUP** buffer may be based on PRACTICE functions, whose values may change during the display of the **PER** window.

With **VARX** you can modify the **SGROUP** buffer in any way you like by using the following PRACTICE functions, which access the **SGROUP** buffer:

PER.Buffer.Byte (<i><index></i>) PER.B.B (<i><index></i>)	Returns a byte at position <i><index></i> from the SGROUP buffer.
PER.Buffer.Word (<i><index></i>) PER.B.W (<i><index></i>)	Returns a 16 bit word at position <i><index></i> from the SGROUP buffer.
PER.Buffer.Long (<i><index></i>) PER.B.L (<i><index></i>)	Returns a 32 bit word at position <i><index></i> from the SGROUP buffer.
PER.Buffer.Quad (<i><index></i>) PER.B.Q (<i><index></i>)	Returns a 64 bit at position <i><index></i> from the SGROUP buffer.

Due to performance reasons you should use **VARX** only, if there is no other solution possible.

Command is only allowed in **SGROUP**.

Example:

```
SGROUP "Dummy Counter"                                ; begin Sequence-GROUP
    varx 0 %quad os.timer()                            ; read timer from OS
    varx 9 %q (PER.B.Q(0)/1000.)                      ; define quad data from
                                                    SGROUP buffer at index 0 by
                                                    1000 and store the result
                                                    at index 9 as quad

    textline ""                                         ; display data at index 0 as
    decmask.quad 0 0--63. 1 " milliseconds:"          decimal
    textline ""                                         ; display data at index 9 as
    decmask.quad 9 0--63. 1 " seconds:"              " decimal
    textline ""                                         ; Newline
```

WRITEBACK

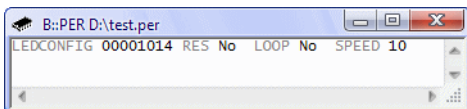
Separate write a part from a read part

Format: **WRITEBACK**

Separates the write part of a sequence from the read part. Command is only allowed in **SGROUP**.

Example 1:

```
SGROUP
    SET 0 %l 0x1014
    GETX 0 %l 0
    WRITEBACK
    CONSTX 2 %w 0x2014
    SETX 0 %l 0
    LINE.LONG 0x0 "LEDCONFIG,LED Configuration Register (20)"
        BITFLD.LONG 0x0 31. "RES ,Reset" "No,Yes"
        BITFLD.LONG 0x0 30. " LOOP ,Loopback" "No,Yes"
        BITFLD.LONG 0x0 29. " SPEED ,Speed" "10,100"
```



The commands after write back are executed only if **PER.Set** command is used. For displaying the data in the PER-window these commands are ignored.

Example 2:

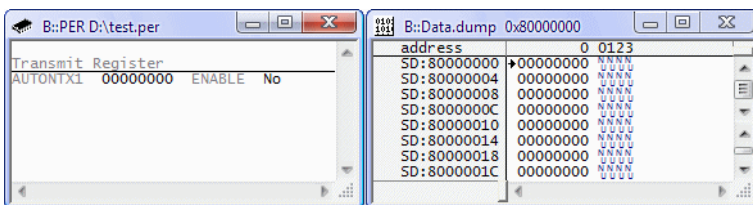
```
SGROUP "Transmit Register" ; define sequence GROUP
GETX d:0x80000000 %l 0 ; read data at 0x80000000 and store
; them in buffer + offset 0

WRITEBACK ; next commands only executed, if a
; write access is done in per-window

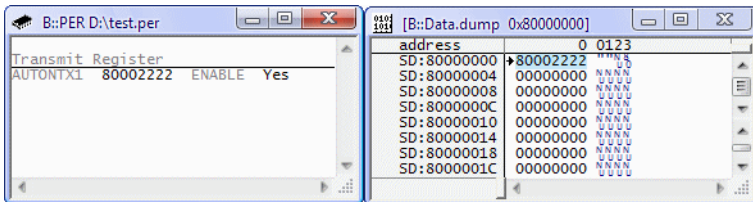
CONSTX 2 %w 0x2222 ; write 0x2222 to buffer + offset 2
SETX d:0x80000000 %l 0 ; write data from buffer + offset 0
; to memory at 0x80000000

LINE.LONG 0x0 "AUTX1,Transmit Reg." ; display AUTX1 register with
BITFLD.LONG 0 31. "ENABLE " "No,Yes" ; contents of buffer[0...3]
; if bit 31 is changed/written
; constx and setx are done
```

Opening the per-window results in displaying data from memory.



Changing state of the ENABLE bit results also in writing constant value 0x2222 to the register.



ASSERT

Abort if condition not met

Format: **ASSERT** *<expression>* [*<string>*]

With **ASSERT** you can ensure that your environment meets a certain condition, before TRACE32 should go on with the parsing of the PER file.

If you omit the optional string with an error message, the following message will be shown instead:
Assertion failed: *<expression>*

- <expression>* Expression which must evaluate to a boolean.
If the result of the expression is FALSE, the parsing of the PER file will be stopped and an error message will be shown.
- <string>* Optional string containing an error message, which will be shown if *<expression>* evaluates to FALSE.

Example: This code line ensures that a PER file is only parsed by “TRACE32 for ARM”

```
ASSERT CPUFAMILY()=="ARM" "Sorry, this PER file is only for ARM cores"
```

AUTOINDENT

Indent content of peripheral file automatically

[\[Examples\]](#)

Format: **AUTOINDENT**.**[ON | OFF]** *<alignment>* *<type>* [*<number>* | *<columns>* *<width>*]

<alignment>: **LEFT | RIGHT | CENTER**

<type>: **TREE | LINE | PROXIMITY | GRID**

Default: OFF

Switches automatic indentation **ON** or **OFF**. Only available for TRACE32 versions >= 97444.

AUTOINDENT ignores all leading and trailing space characters within subsequent definitions and rearranges the contents according to the specified *<alignment>* and *<type>*. It affects all entries within a **TREE** and should therefore only be activated or changed outside of a **TREE**. Otherwise the result may be undefined.

<alignment>	Alignment of the values in relation to their description: LEFT, RIGHT, CENTER . Default: LEFT For examples, see here .
<type>	Indentation type of description-value pairs: TREE, LINE, PROXIMITY, GRID . Default: TREE For examples, see here .
<number>	Proximity range. Only available if <type> = PROXIMITY . Default: 5
<columns>	Number of columns. Only available if <type> = GRID . Default: 5
<width>	Width of a column in characters. Only available if <type> = GRID . Default: 16.

NOTE:

AUTOINDENT affects only the following statements:

- **ASCII**
- **BITFLD, EVENTFLD, RBITFLD, SETCLRFLD**
- **BUTTON**
- **DECMASK, FLOATMASK, HEXMASK**
- **HEXFLD**
- **HIDE**
- **IN**
- **LINE**
- **NEWLINE**

It explicitly does not affect the following statements:

- **BIT**
- **TEXT, TEXTLINE**

It makes the following statements obsolete:

- **WIDTH**
- **CONFIG** (If no BIT command is being used)

Code Example

```

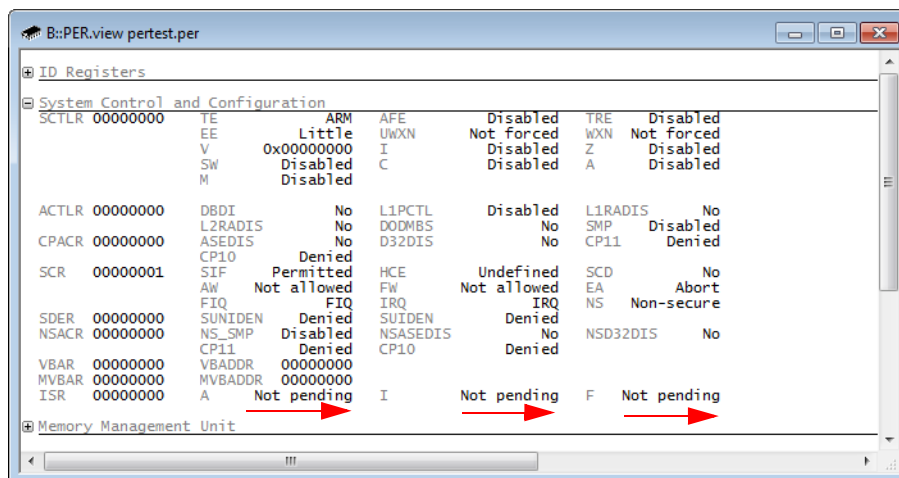
ASSERT version.build()>=97444. "Please update TRACE32"

AUTOINDENT.ON left tree                                ; AUTOINDENT using
TREE "Tree 1"                                           ; <alignment> = left and
    GROUP.LONG ...                                     ; <type> = tree
    LINE.LONG 0, "Reg1,First register"
    BITFLD.LONG 0, 0.--1. "Fld1,First field" "1,2,3,4"
    ...
TREE.END
AUTOINDENT.ON right tree                               ; Second tree looks
TREE "Tree 2"                                           ; better with
    GROUP.LONG ...                                     ; <alignment> = right
    LINE.LONG 0, "Reg32,32nd register"
    BITFLD.LONG 0, 0.--1. "Fld1,First field" "1,2,3,4"
    ...
TREE.END
AUTOINDENT.OFF                                         ; Sometimes you do
TREE "Tree 3"                                           ; not want to use
    GROUP.LONG...                                       ; AUTOINDENT
    LINE.LONG 0, "    Reg99    ,99th register"
    BITFLD.LONG 0, 0.--1. "    Fld1    ,First field" "1,2,3,4"
    ...

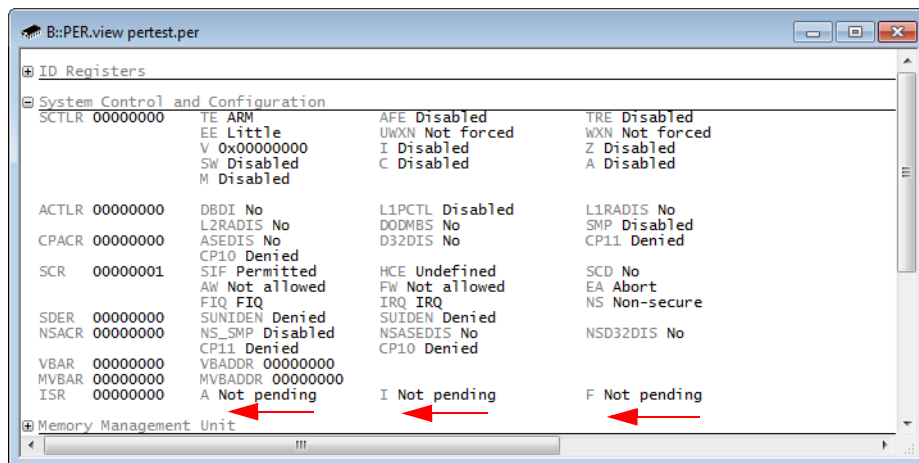
```

<alignment> Examples

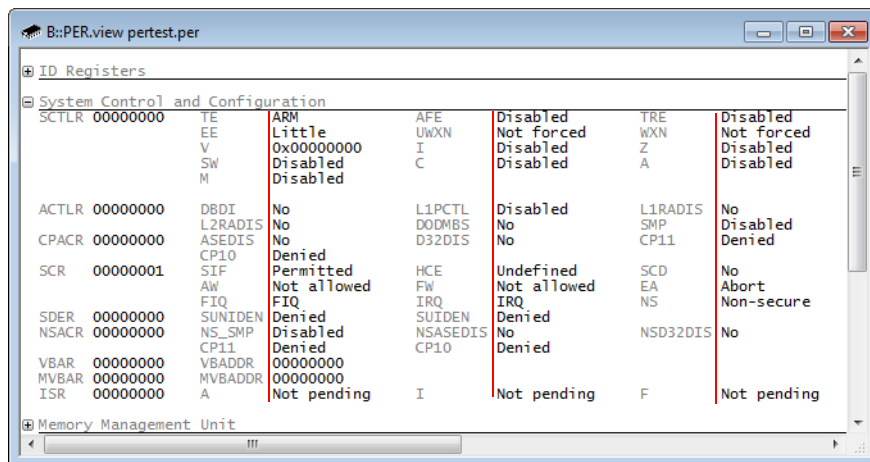
Example 1: AUTOINDENT.ON **RIGHT** TREE aligns all values to the right.



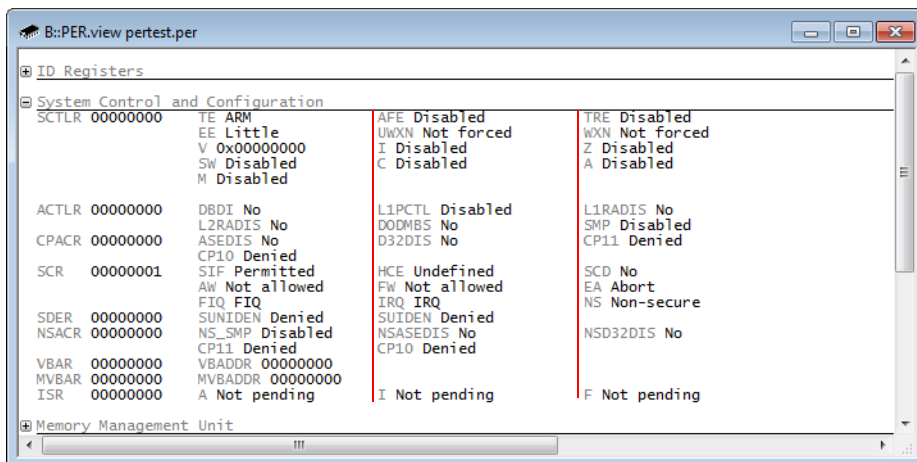
Example 2: AUTOINDENT.ON **LEFT** TREE aligns all values to the left next to their descriptions.



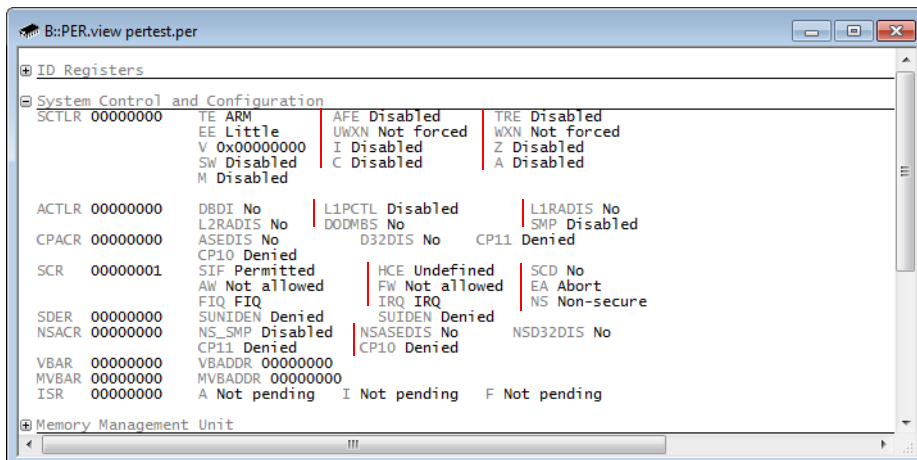
Example 3: AUTOINDENT.ON **CENTER** TREE moves the values somewhere to the middle so they are aligned.



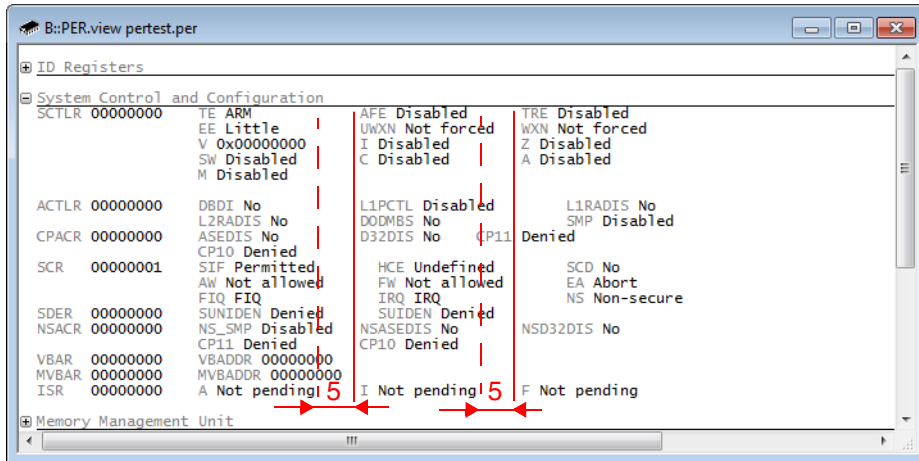
Example 1: AUTOINDENT.ON LEFT **TREE** aligns all description-value pairs within a **TREE**.



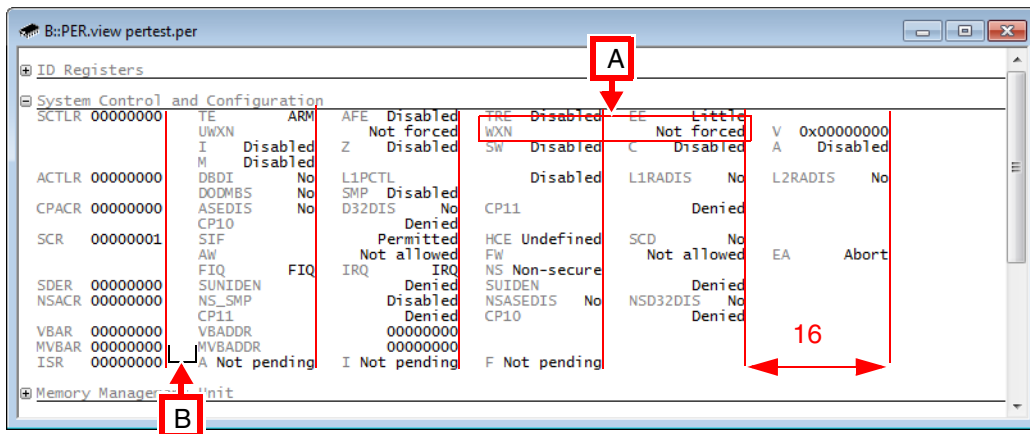
Example 2: AUTOINDENT.ON LEFT **LINE** aligns all description-value pairs within a **LINE**.



Example 3: AUTOINDENT.ON LEFT **PROXIMITY 5** moves all description-value pairs within a **TREE** and the proximity of **<number>** characters to the right in order to align with the right-most description-value pair.



Example 4: AUTOINDENT.ON RIGHT **GRID 5 16**. divides the window into the given number of **<columns>** which are **<width>** characters wide each.



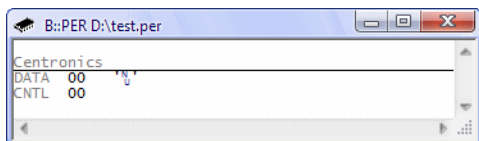
- A** In case a description-value pair does not fit within a column, two (or more) columns will be merged -> see red box above.
- B** When defining the **<width>** of the columns, please take the first 3 separation characters into account.

This command is useful for peripheral files which have been generated automatically and do not contain any **NEWLINE** statements. These will be added automatically if a **LINE** contains more than **<columns>** subentries. **NEWLINE** statements, however, can still be added manually.

Format: **BASE** <address>

This command sets the start address for the peripheral module and refers to simple offset ranges. This expression is permanently recalculated. If the parameters contain functions or symbols, it reflects later changes in the parameters. The BASE command specifies memory class which is responsible for setting appropriate addressing mode. Memory classes are described in [Memory Classes](#) section.

<address> Fixed address or expression which evaluates to the start address of the peripheral groups following the **BASE** command.



Example:

```
// use fixed base
BASE d:0xffff0000
    GROUP.LONG 0x00++0x3
        LINE.LONG 0x00 "Reg_0,Register 0"

// use variable base
BASE (SYStem.BASE()&0x0f)*0x1000

// use variable base
BASE Data.Long(base_pointer)
```

Format: **BASEOUT** <addr_expr> <address> [%<format>] <data>

<format>: **Byte** | **Word** | **Long** | **Quad** | **TByte** | **HByte**
Float. [leee | leeeDbl | leeeXt | <others>]
BE | **LE**

Like the **BASE** command **BASEOUT** defines a start address for the peripheral group definitions following the **BASEOUT** command. This address is usually frequently calculated by the given address expression.

Unlike the **BASE** command **BASEOUT** writes a certain value (*<data>*) to a specified address (*<address>*) before evaluating the expression which sets the start address for the following group definitions. If a bit-mask is used the specified address will be read and modified before it will be written.

NOTE:	If <i><addr_expr></i> is a constant address, no data will be written to <i><address></i> .
--------------	--

<addr_expr> Expression which evaluates to the start address of the peripheral groups following the **BASEOUT** command.

<address> Address which should be written before evaluating the address expression.

<data> Data which should be send to the specified address before evaluating the address expression. This could also be a bit-mask.

Please consider: As the display is refreshed permanently the memory at *<address>* is modified permanently as well.

Example 1: Write 0x01 to address 0x100 before reading the base address from address 0x104. The GROUP command will then read the first three lines at that base address.

```
BASEOUT Data.Long(D:0x104) D:0x100 %Long 0x01
GROUP 0x00++0x3
    LINE.LONG 0x00 "Reg_0,Register 0"
```

Example 2: Set the LSB in address 0x200 before reading the base address from 0x202.

```
BASEOUT Data.Word(D:0x202) D:0x200 %Word 0yXXXXXXXXXXXXXXXXX1
GROUP 0x00++0x3
    LINE.WORD 0x00 "TIMER_CTRL_0,Timer 0 Control register"
```


Format: **BASESAVEOUT** <addr_expr> <address> [%<format>] <data>

<format>: **Byte** | **Word** | **Long** | **Quad** | **TByte** | **HByte**
Float. [**ieee** | **ieeeDbl** | **ieeeXt** | <others>]
BE | **LE**

Outputs a value before calculating a base address with restore. This command is almost the same like **BASEOUT**. However, unlike **BASEOUT** the data on the specified address gets restored after evaluating the address expression.

<addr_expr>	Expression which evaluates to the start address of the peripheral groups following the BASESAVEOUT command.
<address>	Address which should be written before evaluating the address expression. The original content gets saved before evaluating the expression and es restored afterwards.
<data>	Data which should be send to the specified address before evaluating the address expression. This could also be a bit-mask.

Format: **CONFIG** <access_width> [<bits_per_line>]

Configures the default access width used with **GROUP.auto**, aligns the field description after a **LINE** statement, and configures the bits-per-line emitted by the **BIT** statement.

<access_width>	<p>By default the <access_width> is set to 8, which means (a) byte accesses to the memory by GROUP.auto and (b) no extra white space after any LINE statement. The access width in bits configures two things:</p> <ol style="list-style-type: none"> 1. The default data access width in bytes of a GROUP, which does not specify its access width (GROUP.auto). The access width in bytes is calculated as follows: $(\text{access width} + 7) / 8 = \text{result} \quad (\text{max. result: } 8)$ 2. The minimum display width of the hex nibbles of a LINE statement. The minimum width is calculated as follows: $(\text{access width} + 3) / 4 + 1 = \text{result} \quad (\text{max. result: } 17)$
----------------	--

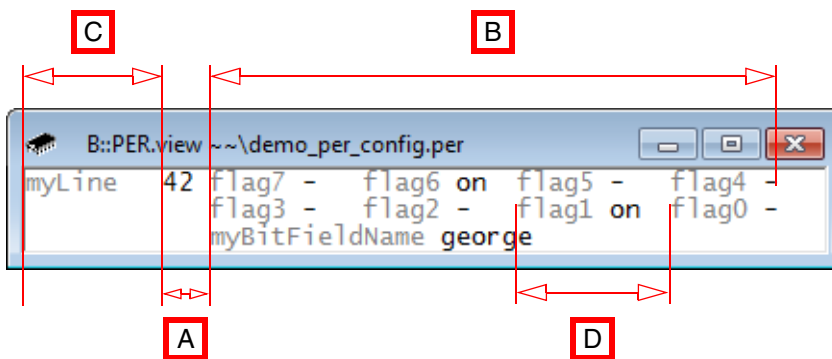
<bits_per_line>

By default *<bits_per_line>* is set to *<access_width>*. The bits per line set the number of bits shown in one line with the **BIT** statement before an automatic line break.

This setting affects only the **BIT** statement, but not the **BITFLD** statement (or others).

Example:

```
WIDTH    9. 10.
CONFIG 16.  4.
GROUP.auto D:0x000++1
LINE.BYTE 0x00 "myLine"
    BIT 7. "flag7" "-,on"
    BIT 6. "flag6" "-,on"
    BIT 5. "flag5" "-,on"
    BIT 4. "flag4" "-,on"
    BIT 3. "flag3" "-,on"
    BIT 2. "flag2" "-,on"
    BIT 1. "flag1" "-,on"
    BIT 0. "flag0" "-,on"
NEWLINE
BITFLD.BYTE 0x00 0--1 "myBitFieldsName " "john,paul,george,ringo"
```



- A** Display width of the hex value emitted by the **LINE** statement. This width is the first parameter of the **CONFIG** statement.
In this example, *<access_width>* is 16 bits, i.e. $(\text{<access_width>} + 3) / 4 + 1 = 5$ characters.
- B** Number of **BIT** items in one single line before an automatic line break. This is configured with the second parameter of the **CONFIG** statement. (here: 4 **BIT** in one line).
- C** Width of the register name emitted by the **LINE** statement. This width is configured with the first parameter of the **WIDTH** statement. (here: 9 characters)
- D** Width of a bit displayed by the **BIT** statement. This width is configured with second parameter of the **WIDTH** statement. (here: 10 characters)

CSV

Enables CSV capabilities

Format: **CSV.[ON | OFF]**

Enables or disables the new CSV file format for *.per files. For more information, see [“Comma-Separated-Values \(CSV\) File Format for *.per Files”](#), page 13.

Refer to the [IF](#) command.

Refer to the [IF](#) command.

Format: **ENDIAN [BE | LE | DEF]**

With DEF parameter the endianness is set due to the configuration of the debugger. With this command the debugger accesses the target data with the specified endianness. This is done independent of the target and the system endianness settings.

Default: **ENDIAN DEF**

Example:

```
ENDIAN.LE                ; little endian
ENDIAN.BE                ; big endian
ENDIAN.DEF               ; target default endian
```

Refer to the [IF](#) command.

Assign parameters used to open the peripheral file to macros, to parametrize the peripheral view (similar to the PRACTICE [ENTRY](#) command).

Refer to [“Passing Arguments”](#), page 9.

Format: **HELP.Winhelp** "<file>,<item>"
 HELP.Online "<item>"

Defines a button in the last GROUP header or tree control. HELP.Online calls the TRACE32 online manual. HELP.Winhelp calls a windows help file (available on Windows only).

IF

Conditional GROUP display

Format: **IF** <condition>
 ELIF <condition>
 ELSE
 ENDIF

<condition>: Condition examples:
 - eval()==<condition_val>
 - %<parameter>==<condition_val>
 - (((data.<size>(<address>))&<bit_mask>)==<condition_val>)

GROUPs can be displayed conditionally using IF...ENDIF commands.

GROUPs defined in different IF and ELIF statements are overlaid at the same place in the window.

Only GROUPs which reside within the fulfilled condition are displayed. The ELSE part is displayed only when no other condition is true. All conditions are dynamically recalculated to reflect the current state of the peripheral.

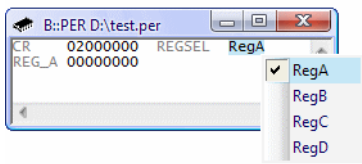
NOTE: The **IF** command cannot be used inside a **GROUP**. (Please use **IF** always before a new **GROUP**).

NOTE: Unlike in the C programming language, the IF statement always evaluates all expressions also for logical operators && and ||.

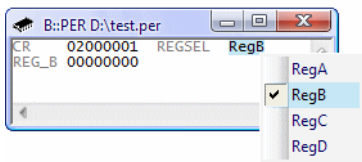
Example:

```
IF (((Data.Long(d:0x00))&0xf)==0x0)
GROUP.LONG d:0x0++0x7
  LINE.LONG 0x0 "CR,Control register"
  BITFLD.LONG 0x0 0.--1. " REGSEL ,Register select" "RegA,RegB,RegC,RegD"
  LINE.LONG 0x4 "REG_A,Register A"
ELIF (((Data.Long(d:0x00))&0xf)==0x1)
GROUP.LONG d:0x0++0x7
  LINE.LONG 0x0 "CR,Control register"
  BITFLD.LONG 0x0 0.--1. " REGSEL ,Register select" "RegA,RegB,RegC,RegD"
  LINE.LONG 0x4 "REG_B,Register B"
ELIF (((Data.Long(d:0x00))&0xf)==0x2)
GROUP.LONG d:0x0++0x7
  LINE.LONG 0x0 "CR,Control register"
  BITFLD.LONG 0x0 0.--1. " REGSEL ,Register select" "RegA,RegB,RegC,RegD"
  LINE.LONG 0x4 "REG_C,Register C"
ELSE
GROUP.LONG d:0x0++0x7
  LINE.LONG 0x0 "CR,Control register"
  BITFLD.LONG 0x0 0.--1. " REGSEL ,Register select" "RegA,RegB,RegC,RegD"
  LINE.LONG 0x4 "REG_D,Register D"
ENDIF
```

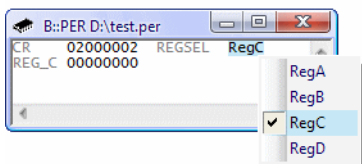
Register REG_A is selected if the value of the REGSEL bit field equals 0.



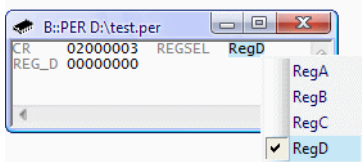
Register REG_B is selected if the value of the REGSEL bit field equals 1.



Register REG_C is selected if the value of the REGSEL bit field equals 2.



Register REG_D is selected if the value of the REGSEL bit field equals 3.



Format	INCLUDE <i><file></i>
--------	------------------------------------

Includes another peripheral file.

<file> Path to another peripheral file

Format: **PERCMD**,<column_list>

<column_list>: **Address,AccessWidth,Name,Tooltip,From,To,Choices[,RW][,Ignore]**

Optional definition of the columns of a peripheral file in CSV format.

- Default: If the **PERCMD** command is omitted in the CSV-formatted *.per file, then the sequence of columns must be: Address,AccessWidth,Name,Tooltip,From,To,Choices
- If the **PERCMD** command is included in a CSV-formatted *.per file, then <column_list> must contain all column names that are flagged as mandatory in the table below. Column names are case sensitive!

NOTE: With the **PERCMD** command included in the CSV-formatted *.per file, you are free to arrange the mandatory and optional columns in any order.

Please also refer to “[Comma-Separated-Values \(CSV\) File Format for *.per Files](#)”, page 13.

Column Names	Meaning in the spreadsheet
Address (mandatory)	Absolute address of a register consisting of access class and value, or the offset from a previously defined BASE command. If empty, the value is assumed to be the same as the last known one. An address different from the previous one corresponds to the LINE command.
AccessWidth (mandatory)	Access width of the register. Valid values are: 8. 16. 32. and 64. If empty, the value is assumed to be the same as the last known one. An access width different from the previous one corresponds to the LINE command.
Name (mandatory)	Name of the register
Tooltip (mandatory)	Tooltip or more meaningful name of the register, e.g. the long form of the register name.
From (mandatory)	Lower boundary of a bit field of a register.
To (mandatory)	Upper boundary of a bit field of a register.

Column Names	Meaning in the spreadsheet
Choices (mandatory)	<ul style="list-style-type: none"> Not empty: Comma-separated list of choices which will appear in the PER.view window in drop-down lists. Corresponds to the BITFLD command. A spreadsheet editor automatically adds the surrounding single quotes when the *.per file is exported in CSV file format. Otherwise the single quotes must be added manually. Empty: Corresponds to the HEXMASK command.
RW (optional)	<p>Access rights to the register or register field. Valid values are:</p> <ul style="list-style-type: none"> RD (read) WR (write) RW (read/write) <p>If empty, WR (write) will be taken as default.</p>
ClearAddress (optional)	<ul style="list-style-type: none"> Not empty: Defines a SETCLRFLD command, see ClearFrom. Empty: Defines a HEXMASK, BITFLD or EVENTFLD command, see ClearFrom.
ClearFrom (optional)	<ul style="list-style-type: none"> Not empty and column ClearAddress empty: Bit(s) of a register which can only be cleared by writing a '1'. Corresponds to the EVENTFLD command. This value must be the same as in the From column while the range is defined as To - From. Not empty and columns ClearAddress, SetAddress and SetFrom not empty: Defines a register status bit with associated set and clear bits. See SETCLRFLD command. Empty: Corresponds to HEXMASK or BITFLD command.
SetAddress (optional)	<ul style="list-style-type: none"> Not empty: Defines a SETCLRFLD command, see ClearFrom. Empty: Corresponds to HEXMASK or BITFLD command.
SetFrom (optional)	<ul style="list-style-type: none"> Not empty: Defines a SETCLRFLD command, see ClearFrom. Empty: Corresponds to HEXMASK or BITFLD command.
Ignore (optional)	<ul style="list-style-type: none"> Ignores a column that is irrelevant for a *.per file, e.g. redundant columns extracted from binaries. User-defined column names will also be ignored in the *.per files.

Example: The two last columns **Ignore** and **myCol1** will not have any effect.

```
PERCMD,Address,AccessWidth,Name,Tooltip,From,To,Choices,Ignore,myCol1
```

SIF

Conditional interpretation

Format: **SIF** (**CPU**()=="<cpu_name>")
 SIF (**CPUIS**("<cpu_name>*"))
 SIF (<logical_comparison>)

According to the condition a block between **SIF** and **ENDIF** (or **SIF** and **ELSE**) will be interpreted when the peripheral file is opened or reparsed. The **SIF** command can be used also inside the **GROUPs**.

Example:

```
SIF (cpu()=="MIPS4KC")
GROUP.LONG CP0:16.++0.
    LINE.LONG 0x0 "Config,Configuration Register"
    BITFLD.LONG 0x00 31. " M ,Config1 register is implemented" "no,yes"
    ...
ELIF (cpu()=="MIPS4KEC")
GROUP.LONG 0x0 "Config,Configuration Register"
    BITFLD.LONG 0x00 31. " M ,Config1 register is implemented" "no,yes"
    ...
ELSE
GROUP.LONG 0x0 "Config,Configuration Register"
    BITFLD.LONG 0x00 31. " M ,Config1 register is implemented" "no,yes"
    ...
ELSE
ENDIF
```

Conventions :

SIF is only to be used to distinguish between CPUs, memory accesses should be avoided (not possible in system.mode down).

Using once a GROUP command inside a **SIF** block, all trees of the **SIF** block must contain GROUP commands. Also the next command after a finished **SIF** block must be a GROUP command then.

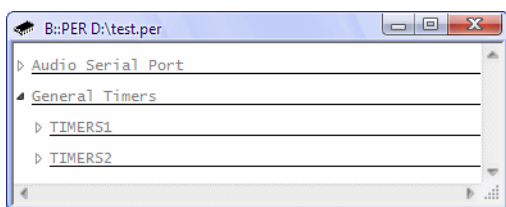
Using the command **PER.TestProgram** the error will be detected.

Format: **TREE** "<name>"
 TREE.OPEN "<name>"
 TREE.END

Defines a "Treeview" of peripheral modules. The tree can be displayed/hidden by a tree control (+/-). It is possible to nest trees.

Example:

```
TREE "Audio Serial Port"           ; tree GROUP displayed closed by
;                                  default
TREE.END                           ; definition of the GROUP members
TREE.OPEN "General Timers"
    TREE "TIMERS1"                 ; tree GROUP displayed opened in the
    ;                              peripheral window
    TREE.END
TREE.END
```



WIDTH

Width of register names and a BIT description

Format **WIDTH** [<register_name>] [<bit_width>]

Configures width of **LINE** register names and a **BIT** description.

<register_name> Sets the width of the register name emitted by the **LINE** statement.
 (default: 6.)

<bit_width> Sets the width reserved for the output of a **BIT** statement. This setting
 (default: 9.) affects only the **BIT** statement, but not the **BITFLD** statement (or others).

Example: For an example, see the **CONFIG** statement.

Format	WAIT [<address> <expression> <boolean_expression>]
--------	---

The **WAIT** command is available for all architectures and **PER** files, but it should only be used when required (i.e. **SIF** with target-dependent values). Most architectures will probably *not* require **WAIT**. But if there is a need to use **WAIT**, then the recommendation is to use **WAIT** at the beginning of a **PER** file.

<address>	Target address which has to be accessible; see example 2 .
<expression>	TRACE32 expression which can be evaluated; see example 3 .
<boolean_expression>	Boolean expression which has to be true; see example 4 .

There are four ways to use the **WAIT** command, see examples 1 to 4.

Example 1: Wait with compilation until the target is up and regular memory can be accessed (this usually means that the target is stopped).

```
WAIT
```

Example 2: Wait with compilation until the target is up and the given memory address can be accessed (it is never really accessed).

```
WAIT ETM:0
```

Example 3: Wait with compilation until the target is up and the expression can be evaluated (the result does not matter).

```
WAIT Data.Long(D:0)
```

Example 4: Wait with compilation until the target is up and the boolean expression evaluates to true.

```
WAIT Data.Long(D:0) != 0
```

Commands within GROUPs

These commands are only useful inside a GROUP (**GROUP**, **RGROUP**, **WGROUP**, **HGROUP**, **SGROUP**).

Beside the commands **INDEX**, **SAVEINDEX** and **BUTTON**, which extend the memory access by a GROUP, the commands define how the data fetched by a GROUP command should be displayed and/or modified.

ASCII

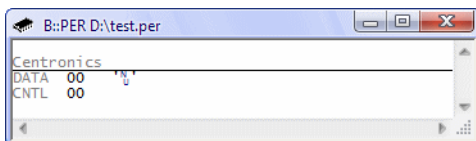
Display ASCII character

Format: **ASCII**

The previously defined byte is displayed as an **ASCII** character.

Example:

```
GROUP.BYTE sd:0x100--0x101 "Centronics"  
  LINE.BYTE 0x0 "DATA,Centronics Data Register"  
    ASCII  
  LINE.BYTE 0x1 "CNTL,Centronics Control Register"
```



BIT

Define bits

Format: **BIT** *<bit>*|*<bitrange>* " *<display_name>*,*<tooltip>*" " *<choices>*"

These fields are in fixed positions in the per window. The bit numbers must be entered from MSB to LSB. The size of a field depends on the number of bits and the size of the name header.

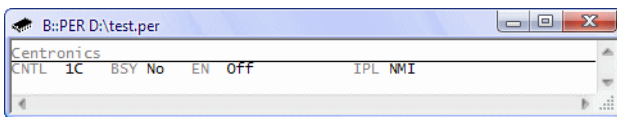
<bit> |
<bitrange>

Defines bit's number and range. LSB is defined as the first, MSB as the second character.

<code><display_name></code>	Short name (abbreviation) of corresponding bit.
<code><tooltip></code>	The sentence accurately describing a bits functionality.
<code><choices></code>	Indicates states with bit field may take. LSB is defined as the first, MSB as the last one. Each state is separated by a comma.

Example:

```
GROUP sd:0x100--0x101 "Centronics"
  LINE.BYTE 0x00 "CNTL,Centronics Control Register"
    BIT 7 "BSY,Centronics Busy" "No,Yes"
    BIT 6 "EN,Centronics Enable" "Off,On"
    BIT 2--4 "IPL,Centronics Interrupt Level" "Off,1,2,3,4,5,6,NMI"
```



BITFLD

Define bits individually

Format: **BITFLD.**`<size> <offset> <bit_range1> [<bit_range2>]`
`"<display_name>,<tooltip>"`
`["<choices>[%d...!%x...|<string>...]"] ...`

BITFLD is used to display the bit field name and its contents in a free format. The fields are chained together in a line. A new line can be created by a **TEXTLINE** command.

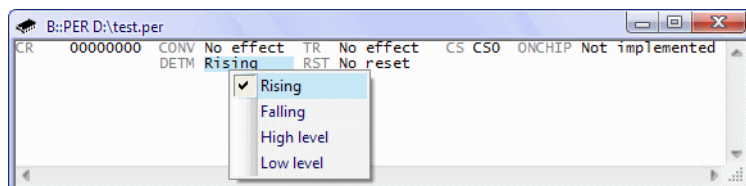
<code><size></code>	Size of register (Byte, Word, TByte, Long, Quad).
<code><offset></code>	The bit field offset refers to the start address of the GROUP command.
<code><bit_range1></code>	Defines a range of bits (or a single bit) that belong to a bit field. The lower bit number has to come before the higher bit number, e.g. 3 . -- 7 .
<code><bit_range2></code>	For disjunct bit fields (= where not all bits are in one block), you can define a second range of bits (or a single bit). Please see examples .
<code><short_name></code>	Short name (abbreviation) of corresponding bit field.
<code><long_name></code>	The sentence accurately describing a bit field functionality.
<code><choices></code>	Defines the possible values (in words) which the bit field may take. LSB is defined as the first, MSB as the last one. Each state is separated by a comma. If you define fewer <code><choices></code> than required for the <code><bit_range></code> , then append %x...

%d...	Placeholder for reserved/unused values at the end of <choices>. The values will be formatted as decimal numbers when displayed in the PER.view window. The field width is defined by the <choices>. If the decimal value is too large to fit into the field, a question mark is displayed. Please see examples .
%x...	Placeholder for reserved/unused values at the end of <choices>. The values will be formatted as hexadecimal numbers when displayed in the PER.view window. The field width is defined by the <choices>. If the hex value is too large to fit into the field, a question mark is displayed.
<string>...	Placeholder for reserved/unused values at the end of <choices>. The values will be displayed as strings in the PER.view window.

```

BASE d:0x00000000
GROUP 0x00++0x03
    LINE.LONG 0x00 "CR,Control Register"
        BITFLD.LONG 0x00 31. " CONV ,Conversion Bit" "No effect,Conv"
        BITFLD.LONG 0x00 24. " TR ,Transfer" "No effect,Transferred"
        BITFLD.LONG 0x00 16.--19. " CS ,Chip Select"
        "CS0,CS1,CS2,CS3,CS4,CS5,CS6,CS7,CS8,CS9,CS10,CS11,CS12,CS13,CS14,CS15"
        BITFLD.LONG 0x00 5. " ONCHIP ,On chip trace implemented" "Not
implemented,Implemented"
    TEXTLINE " "
        BITFLD.LONG 0x00 1. 3. " DETM ,Detection mode"
        "Rising,Falling,High level,Low level"
        BITFLD.LONG 0x00 0. " RST ,Reset mode" "No reset,Reset"

```



Examples

Example for bitranges:

Example 1:

31	...	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
----	-----	----	----	----	----	----	----	---	---	---	---	---	---	---	---	---	---

Example 2:

31	...	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
----	-----	----	----	----	----	----	----	---	---	---	---	---	---	---	---	---	---

Example 3:

31	...	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
----	-----	----	----	----	----	----	----	---	---	---	---	---	---	---	---	---	---

Example 4:

31	...	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
----	-----	----	----	----	----	----	----	---	---	---	---	---	---	---	---	---	---

Example 5:

31	...	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
----	-----	----	----	----	----	----	----	---	---	---	---	---	---	---	---	---	---

Example 6:

31	...	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
----	-----	----	----	----	----	----	----	---	---	---	---	---	---	---	---	---	---

```
;Example 1          <bit_range1>
BITFLD.<size> 0x00    2.

;Example 2          <bit_range1>
BITFLD.<size> 0x00    2.--8.

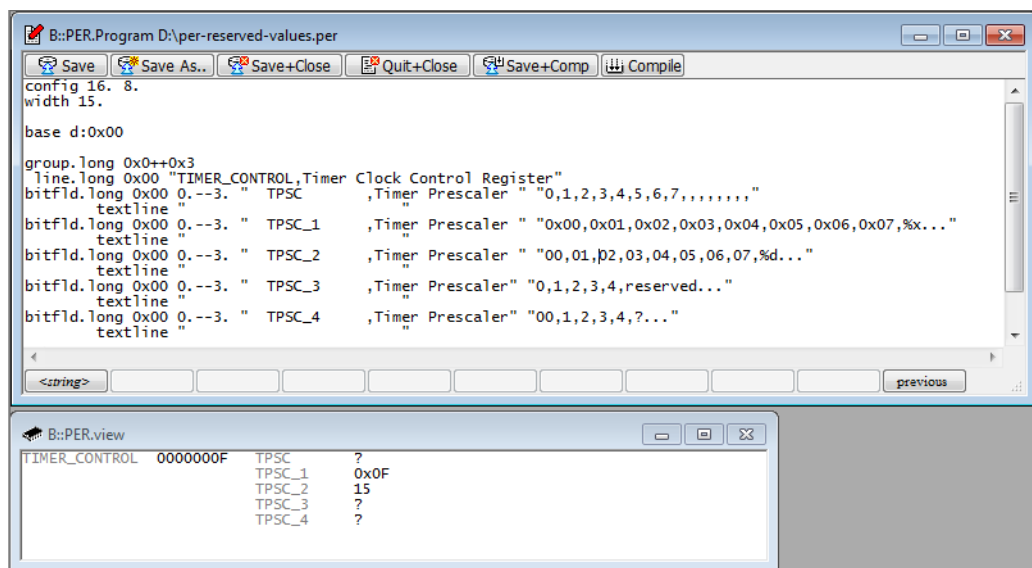
;Example 3          <bit_range1> <bit_range2>
BITFLD.<size> 0x00    2.--8.      14.

;Example 4          <bit_range1> <bit_range2>
BITFLD.<size> 0x00    2.--8.      14.--15.

;Example 5          <bit_range1> <bit_range2>
BITFLD.<size> 0x00    2.          14.

;Example 6          <bit_range1> <bit_range2>
BITFLD.<size> 0x00    2.          14.--15.
```


Example for handling unused/reserved values:



Format: **BUTTON** " <text> " " <cmdline> "

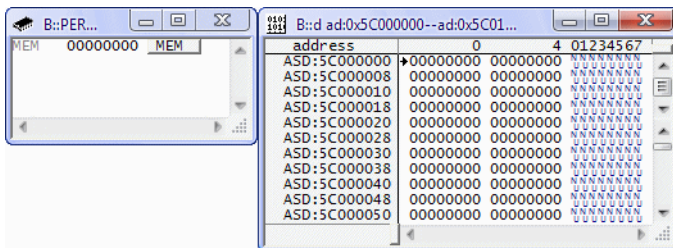
Clicking an input field (button) executes the defined command line. This field can be used to execute input/output commands or open different views (e.g. memory dumps).

<text> Name of the button.

<cmdline> Contains command, address area and an access size.

Example 1: Button with single command.

```
GROUP.LONG 0x00++0x3
LINE.LONG 0x00 "MEM,Memory Array"
BUTTON "MEM " "Data.dump ad:0x5C000000--ad:0x5C01FFFF /Long"
```



Example 2: Button with multiple commands.

```
GROUP.LONG D:0x00++0xFF
LINE.LONG 0x00 "RST_VEC,Reset Vector"
BUTTON "Clear Vector Table"
(
    Data.dump 0x00++0xFF /Long
    Data.set %Long ad:0x5C000000++01FFFF 0
)
```

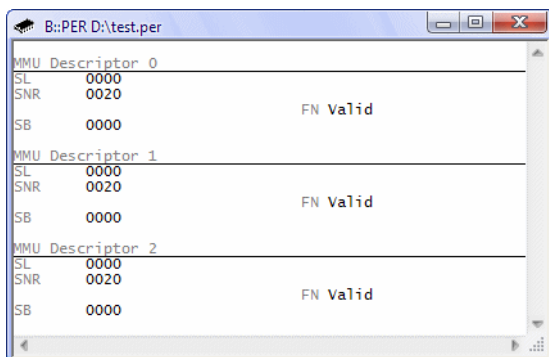
Format: **COPY** [*<number>*]

Copies the last defined **GROUP** to the current **GROUP**. The optional argument defines which **GROUP** should be copied. Number of the **GROUP** is calculated backward form the current one. The command is used to duplicate the definition of **GROUP**s, e.g. for devices with many equal channels.

<number> Optional GROUP number.

Example 1:

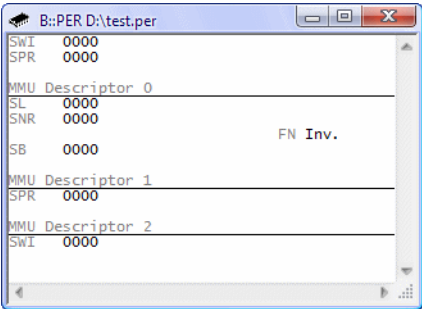
```
GROUP.WORD sd:0x80008038--0x8000803f "MMU Descriptor 0"  
  LINE.WORD 0x0 "SL,Segment Length"  
  LINE.WORD 0x2 "SNR,Segment Number"  
    bit 5 " FN, Flush" "Inv.,Valid"  
  LINE.WORD 0x4 "SB,Segment Base Address"  
GROUP.WORD sd:0x80008048--0x8000804f "MMU Descriptor 1"  
  copy  
GROUP.WORD sd:0x80008050--0x80008057 "MMU Descriptor 2"  
  COPY
```



MMU Descriptor 0		
SL	0000	
SNR	0020	
SB	0000	FN Valid
MMU Descriptor 1		
SL	0000	
SNR	0020	
SB	0000	FN Valid
MMU Descriptor 2		
SL	0000	
SNR	0020	
SB	0000	FN Valid

Example 2:

```
GROUP.WORD sd:0x80008034--0x80008035
  LINE.WORD 0x0 "SWI,Segment Width"
GROUP.WORD sd:0x80008036--0x80008037
  LINE.WORD 0x0 "SPR,Segment Priority"
GROUP.WORD sd:0x80008038--0x8000803f "MMU Descriptor 0"
  LINE.WORD 0x0 "SL,Segment Length"
  LINE.WORD 0x2 "SNR,Segment Number"
    bit 5 " FN, Flush" "Inv.,Valid"
  LINE.WORD 0x4 "SB,Segment Base Address"
GROUP.WORD sd:0x80008048--0x8000804f "MMU Descriptor 1"
  COPY 2
GROUP.WORD sd:0x80008050--0x80008057 "MMU Descriptor 2"
  COPY 4
```



DECMASK

Define bits for decimal display

Format:	DECMASK. <i><access_size></i> [<i>.<display_length></i>] <i><offset></i> <i><bit_range></i> <i><scale></i> [<i><add></i>] " <i><display_name></i> , <i><tooltip></i> "
---------	--

While the similar command **HEXMASK** displays bits as a hexadecimal value, **DECMASK** displays bits as decimal value.

DECMASK defines a set of bits, which should be displayed as decimal value. The bits are extracted from the current buffer at location defined in the bitrange. The result of this extract is multiplied by *<scale>* and increased by the optional *<add>* value.

<i><access_size></i>	Size of register access (Byte, Word, TByte, Long, Quad).
<i><display_length></i>	Length of displayed field (Byte, Word, TByte, Long, PByte, HByte, SByte, Quad).

<offset>	The DECMASK field offset refers to the start address of the GROUP command.
<bit_range>	Defines range of the DECMASK field. LSB is defined as the first, MSB as the second character.
<scale>	Multiplier value. May be a floating point value since build. 46110
<add>	Optional addend - increases value.
<display_name>	Short name (abbreviation) of corresponding DECMASK field.
<tooltip>	The sentence accurately describing a DECMASK field functionality.

FLOATMASK

Define bits for decimal floating point display

Format: **FLOATMASK.**<access_size>[.<display_length>] <offset> <bit_range> <scale> [**<add>**] "<display_name>,<tooltip>"

While the similar command **DECMASK** displays bits only as a decimal value *without* positions after decimal point, **FLOATMASK** displays bits as decimal value *with* positions after decimal point.

FLOATMASK defines a set of bits, which should be displayed as decimal value. The bits are extracted from the current buffer at location defined in the bitrange. The result of this extract is multiplied by <scale> and increased by the optional <add> value.

<access_size>	Size of register access (byte, word, tbyte, long, quad).
<display_length>	Length of displayed field (byte, word, tbyte, long, quad).
<offset>	The FLOATMASK field offset refers to the start address of the GROUP command.
<bit_range>	Defines range of the FLOATMASK field. LSB is defined as the first, MSB as the second character.
<scale>	Multiplier value. Usually a floating point value.
<add>	Optional addend - increases value.
<display_name>	Short name (abbreviation) of corresponding FLOATMASK field.
<tooltip>	The sentence accurately describing a FLOATMASK field functionality.

Example:

```
GROUP D:0x80001204++3 "Timer"
TEXTLINE " "
DECMASK.LONG 0 0--31. 1 " milliseconds: "
TEXTLINE " "
FLOATMASK.LONG 0 0--31. 0.001 " seconds: "
TEXTLINE " "
```

EVENTFLD

Define event flag bits individually

Format: **EVENTFLD.***<size> <offset> <bit_range> "<display_name>,<tooltip>" "<choices>"*

Defines an event bit display in a free format. An event bit can be cleared by writing a '1'. Writing '0' does not affect event bit. The fields are chained together in a line. A new line can be created by a **TEXTLINE** command. The implementation format is the same as a **BITFLD** format.

<i><size></i>	Size of register (byte, word, tbyte, long, quad).
<i><offset></i>	The event bit offset refers to the start address of the GROUP command.
<i><bit_range></i>	Defines range of the bit field. LSB is defined as the first, MSB as the second character. Optionally the third character is bit (or bit range), used if two bit fields are conjuncted.
<i><display_name></i>	Short name (abbreviation) of corresponding event bit field.
<i><tooltip></i>	The sentence accurately describing a event bit field functionality.
<i><choices></i>	Indicates states with bit field may take. LSB is defined as the first, MSB as the last one. Each state is separated by a comma.

Example:

```
GROUP.WORD d:0x100--0x11f "TPU Channels"
TEXTLINE " "
TEXTLINE "CH FUNC PRIO HSF HSR IEF ISF LNK SGL CHS PRM0 PRM1"
TEXTLINE " 0,Channel 0"
BITFLD.WORD 0x1e 0.--1. " " " Off, Low, Mid,High"
BITFLD.WORD 0x16 0.--1. " " " $0, $1, $2, $3"
EVENTFLD.WORD 0x1a 0. " " "No,Yes"
```

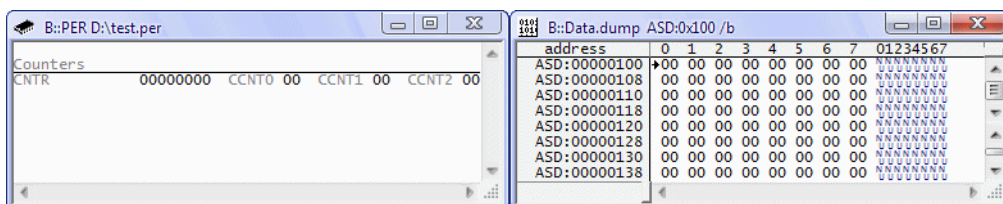
Format: **HEXFLD.<length> <offset> "<display_name>,<tooltip>"**

Defines HEX value in a free format. The fields are chained together in a line. A new line can be created using **TEXTLINE** command. If not the whole value should be displayed. The output size can be limited by the "length" parameter.

<length>	Length of HEX field (Byte , Word , TByte , Long , Quad).
<offset>	The HEX field offset refers to the start address of the GROUP command.
<display_name>	Short name (abbreviation) of corresponding HEX field.
<tooltip>	The sentence accurately describing a HEX field functionality.

Example:

```
GROUP 0x100++0x03 "Counters"
  LINE.LONG 0x00 "CNTR,Channel Counter Register"
    HEXFLD.BYTE 0x00 " CCNT0 ,Channel Counter 0 "
    HEXFLD.BYTE 0x01 " CCNT1 ,Channel Counter 1 "
    HEXFLD.BYTE 0x02 " CCNT2 ,Channel Counter 2 "
```



Format: **HEXMASK.****<access_size>**[**.<display_length>**] **<offset>** **<bit_range>** **<scale>**
[**<add>**] "**<display_name>**,"**<tooltip>**"

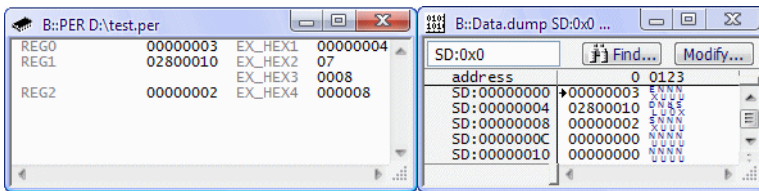
Defines set of bits using HEX value. The bits are extracted from the current buffer at location defined in the bitrange. The result of this extract is multiplied by scale. The <add> value is optional.

<access_size>	Size of register access (Byte, Word, TByte, Long, Quad).
<display_length>	Length of displayed field (Byte, Word, TByte, Long, PByte, HByte, SByte, Quad).
<offset>	The HEX mask field offset refers to the start address of the GROUP command.
<bit_range>	Defines range of the HEX mask field. LSB is defined as the first, MSB as the second character.
<scale>	Multiplier value. <small>May be a floating point value since build. 46110.</small>
<add>	Optional addend - increases Hex mask value.
<display_name>	Short name (abbreviation) of corresponding HEX mask field.
<tooltip>	The sentence accurately describing a HEX mask field functionality.

Example:

```
CONFIG 16. 8.

BASE 0x0
WIDTH 6.
GROUP.LONG 0x00++0xb
LINE.LONG 0x00 " REG0,register 0"
    HEXMASK.LONG 0x00 0.--29. 1. 1. " EX_HEX1  ,Example Hex mask 1"
LINE.LONG 0x04 " REG1,Register 1"
    HEXMASK.LONG.BYTE 0x04 23.--30. 1. 2. " EX_HEX2  ,Example Hex mask 2"
    TEXTLINE "                "
    HEXMASK.LONG.WORD 0x04 4.--15. 8. "  EX_HEX3  ,Example Hex mask 3"
LINE.LONG 0x8 " REG2,Register 2"
    HEXMASK.LONG.TBYTE 0x08 0.--23. 1. 6. " EX_HEX4  ,Example Hex mask 4"
```

HIDE

Define write-only line

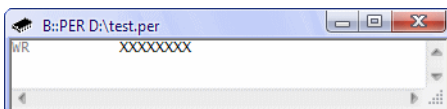
Format: **HIDE**.<size> <offset> "<display_name>,<tooltip>"

This field is used for write-only ports like USART transmitters data registers. **HIDE** command should be used together with **HGROUP** command.

<size>	Size of register (byte, word, tbyte, long, quad).
<offset>	The register offset refers to the start address of the HGROUP command.
<display_name>	Short name (abbreviation) of corresponding register.
<tooltip>	The sentence accurately describing a register functionality.

Example:

```
HGROUP.LONG 0x00++0x3
  HIDE.LONG 0x00 "WR,Write only Register"
```

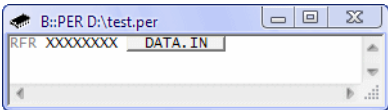


Format:	IN
---------	-----------

An input-field (key) is displayed for the previously defined byte. Clicking that field results in reading data from previously defined location. To execute a read cycle **IN** command must be used along with a **HIDE** definition. It is used for destructive-read ports (i.e. data port of serial interface).

Example:

```
BASE d:0xA00F0000
HGROUP.LONG 0x00++0x3
  HIDE.LONG 0x00 "RFR,Receive FIFO Register"
  IN
```



Format:	INDEX <address> [%<format>] <dataread> <datawrite> OUT (deprecated)
<format>:	Byte Word Long Quad TByte HByte Float. [ieee ieeeDbl leeeeXt <others>] BE LE

Sends specified data to the port. **INDEX** command must be placed after a **GROUP** definition. The data is sent to the port prior to the port access or modification. If two bytes are defined, the second byte is used for writing to the specified port (different indices for reading and writing). It is useful for ports which must be selected first.

Please consider: As the display is refreshed permanently the index register is modified as well.

NOTE:	The INDEX command has no effect inside an SGROUP command.
--------------	---

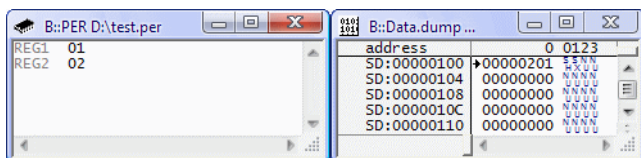
<address>	Destination address.
<dataread>	Data send to the specified address before fetching the data shown by the group definition.
<datawrite>	Data send to the specified address before executing a write to a member of the group definition.

Example 1:

```

GROUP sd:0x100--0x100                                ; select register 1
  INDEX sd:0x100 0x01
  LINE.BYTE 0x0 "REG1,Register index 1"
GROUP sd:0x101--0x101                                ; select register 2
  INDEX sd:0x101 0x02
  LINE.BYTE 0x0 "REG2,Register index 2"

```

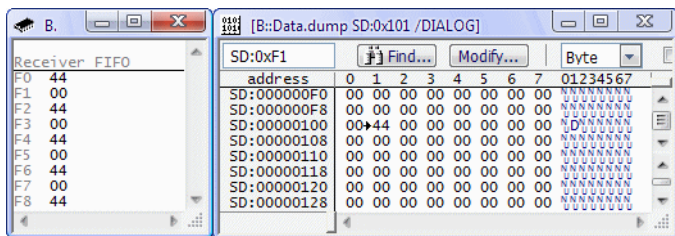


Example 2:

```

GROUP sd:0x101 0x10 "Receiver FIFO"
INDEX sd:0x100 0 0x80 0
  LINE.BYTE 0x0 "F0,FIFO position 0"
  LINE.BYTE 0x1 "F1,FIFO position 1"
  LINE.BYTE 0x2 "F2,FIFO position 2"
  LINE.BYTE 0x3 "F3,FIFO position 3"
  LINE.BYTE 0x4 "F4,FIFO position 4"
  LINE.BYTE 0x5 "F5,FIFO position 5"
  LINE.BYTE 0x6 "F6,FIFO position 6"
  LINE.BYTE 0x7 "F7,FIFO position 7"
  LINE.BYTE 0x8 "F8,FIFO position 8"

```



Format: **LINE.**[<size> | **FLOAT.**<format>] <offset> "<display_name>,<tooltip>"

The **LINE** command defines registers short name and its long name. The value of the offset is added to the address defined in the previous **GROUP** command. The **CONFIG** command affects the displayed format of the **LINE** command.

<size> Size of register (**Byte, Word, TByte, Long, Quad**).

<format> Display register content as floating point number. Currently the following formats are supported:

- IEEE: 32 bit IEEE-754 single
- IEEE DBL: 64 bit IEEE-754 double

<offset> The register offset refers to the start address of the GROUP command.

<code><display_name></code>	Short name (abbreviation) of corresponding register.
-----------------------------------	--

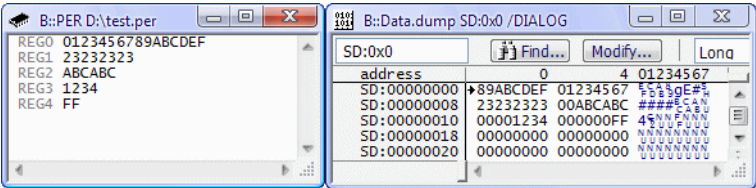
<tooltip> Register long name (a sentence accurately describing the register functionality).

Example:

```

BASE 0x0
WIDTH 6.
GROUP.QUAD 0x00++0x7
    LINE.QUAD 0x00 " REG0,Register 0"
GROUP.LONG 0x08++0x3
    LINE.LONG 0x00 " REG1,Register 1"
GROUP.TBYTE 0x0c++0x2
    LINE.TBYTE 0x00 " REG2,Register 2"
GROUP.WORD 0x10++0x1
    LINE.WORD 0x00 " REG3,Register 3"
GROUP.BYTE 0x14++0x0
    LINE.BYTE 0x00 " REG4,Register 4"

```



Format: **MUNGING** *<belle>*

Usually byte ordering is either little endian or big endian mode. For PPC additional munging little endian and munging big endian modes are provided. For a detailed description refer to PPC documentation.

Special address translation for PowerPC little endian mode.

```
MUNGING.LE
```

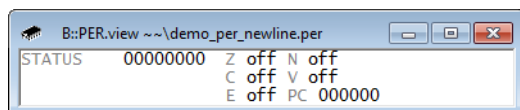
NEWLINE

Line break within detailed register description

Format: **NEWLINE**

Creates a line break for the detailed description of the fields of a peripheral register. The indentation of the new line can be configured with the first parameter of **WIDTH** and **CONFIG**.

```
CONFIG 32.
WIDTH 10.
GROUP.LONG D:0x100++3
LINE.LONG 0x00 "STATUS,Status Register"
    BITFLD.LONG 0x00 31. " Z ,Zero Flag"      "off,on"
    BITFLD.LONG 0x00 30. " N ,Negative Flag"    "off,on"
    NEWLINE
    BITFLD.LONG 0x00 29. " C ,Carry Flag"      "off,on"
    BITFLD.LONG 0x00 28. " V ,Overflow Flag"    "off,on"
    NEWLINE
    BITFLD.LONG 0x00 27. " E ,Interrupt Mask"  "off,on"
    HEXMASK.LONG.TBYTE 0x00 0.--23. 4 " PC ,Program Counter"
```



Format: **RBITFLD.<size> <offset> <bit_range> " <display_name>,<tooltip>"**
" <choices>"

RBITFLD is identical to **BITFLD** with the difference that the defined bits are read-only. It can be used to visualize that certain settings within a read-write register are read-only.

<size>	Size of register (Byte, Word, TByte, Long, Quad).
<offset>	The bit field offset refers to the start address of the GROUP command.
<bit_range>	Defines range of the bit field. LSB is defined as the first, MSB as the second character. Optionally the third character is bit (or bit range), used if two bit fields are conjuncted.
<short_name>	Short name (abbreviation) of corresponding bit field.
<long_name>	The sentence accurately describing a bit field functionality.
<choices>	Defines the possible values (in words) which the bit field may take. LSB is defined as the first, MSB as the last one. Each state is separated by a comma.

```
BASE D:0xF0001234
GROUP 0x00++0x03
    LINE.LONG 0x00 "CSR,Control and Status Register"
        RBITFLD.LONG 0x00 1. " RSTST ,Reset status" "Reset inactive, Reset
active"
        BITFLD.LONG 0x00 0. " RST ,Reset" "No reset,Reset"
```

RHEXMASK

Define bits for a hexadecimal display (read-only)

Format: **RHEXMASK.<access_size>[.<display_length>] <offset> <bit_range> <scale>**
[<add>] " <display_name>,<tooltip>"

Same as **HEXMASK** but bits are read-only.

Format: **SAVEINDEX** <address> [%<format>] <dataread> <datawrite>
SAVEOUT (deprecated)

<format>: **Byte** | **Word** | **Long** | **Quad** | **TByte** | **HByte**
Float. [**ieee** | **ieeeDbl** | **ieeeXt** | <others>]
BE | **LE**

Sends the specified data to the port. The current values at the port are read before the access is made and are restored after the access. The byte is sent to the port prior to the port access or modification.

SAVEINDEX command must be placed after a GROUP definition. If two bytes are defined, the second byte will be used for writing to the specified port (different indices for reading and writing). This is useful for ports which are selected by another port when the index register can be read back.

<address> Destination address.

<dataread> Data send to the specified address before fetching the data shown by the group definition.

<datawrite> Data send to the specified address before executing a write to a member of the group definition.

NOTE: **SAVEINDEX** command has no effect inside an **SGROUP** command.

```
GROUP d:0x11--0x11 "SERIAL CONTROL 80196"
  SAVEINDEX d:0x14 %byte 0x00 0x0f           ;index 0 for read,
                                              ;15 for write
  LINE.BYTE 0 "SCN,Serial Control Register"
```

Format: **SAVETINDEX** <address> [%<format>] <dataread> <datawrite>

<format>: **Byte | Word | Long | Quad | TByte | HByte**
Float. [ieee | ieeeDbl | leeeXt | <others>]
BE | LE

Similar to **SAVEINDEX**, uses however a different sequence for write accesses: the data value is first written to the address and the index is written to trigger/transfer the write operation.

SETCLRFLD

Define set/clear locations

Format: **SETCLRFLD.**<size> <offset1> <bit1> <offset2> <bit2> <offset3> <bit3>
 "<display_name>,<tooltip>" "<choices>"

Defines a bit display in a free format. The fields are chained together in a line. A new line can be created by a **TEXTLINE** command.

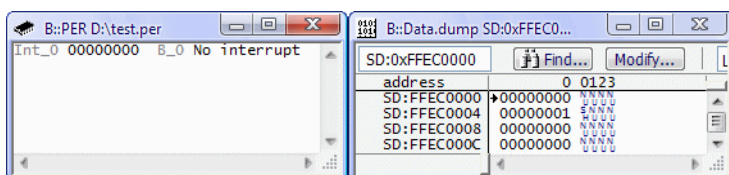
<size>	Size of register (Byte, Word, TByte, Long, Quad).
<offset1> <bit1>	Status register offset and corresponding bit number.
<offset2> <bit2>	Set register offset and corresponding bit number.
<offset3> <bit3>	Clear register offset and corresponding bit number.
<display_name>	Short name (abbreviation) of corresponding set/clear bits.
<tooltip>	The sentence accurately describing a set/clear bits functionality.
<choices>	Indicates states with bit field may take. The first state is responsible for clearing, the second one for setting corresponding set/clear bits. Each state is separated by a comma.

The command is an extension of the **BITFLD** command. Additionally to the BITFLD command two further locations must be entered. The first parameter pair offset1 - bit1 is the location where the data is read from. The second parameter pair offset2 - bit2 is the set location. The third parameter pair offset3 - bit3 is the clear location.

Usually the SETCLRFLD-command is used if the read location is a status register, which shows the status of I/O ports and other (not static) registers exist to enable and disable ports. If the port is enabled, the value of '1' is set to the corresponding bit in the register addressed by location 2 (other bits are cleared). If the port is disabled, the value of '1' is set at the corresponding bit position in the register addressed by location 3 (the other bits are cleared).

```
BASE sd:0xffec0000
GROUP.LONG 0x00++0x3
  LINE.LONG 0x00 "Int_0,Interrupt Register 0"
    SETCLRFLD.LONG 0x0 0. 0x4 0. 0x8 0. " B_0 ,Bit 0"
    "No Interrupt,Interrupt"

;writing 1 sets the bit in the Set Register
;writing 0 sets the bit in the Clear Register
;the result is read from the Status register
```



STRING

Display a string saved in memory

Format: **STRING** <display_width> <offset> <string>

Defines a field to display an ASCII encoded string, which is saved in target memory.

Example:

```
BASE sd:0xff000000
WIDTH 8.
GROUP.LONG 0x00++0x03
  LINE.LONG 0x00 "KEYREG, "
  STRING 4. 0. "KEY "
  STRING 3. 0. " KEY "
  STRING 3. 1. " KEY "
```

<width>	Number of bytes/characters.
<offset>	Offset to group start address.
<string>	Field name.

Format: **SYSCON**

Special block for C166 bondout CPUs (ICE).

TEXTLINE

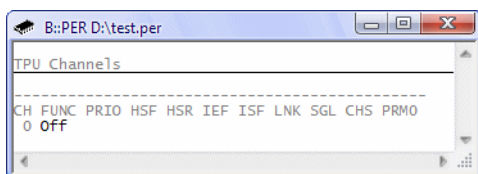
Define text header with a new line

Format: **TEXTLINE "<text>"**

The text can either be used as general comment or as a header to **BITFLD** or **HEXFLD** fields. **TEXTLINE** creates a new line.

<text> Optional text.

```
GROUP d:0x0e00--0x0fff "TPU Channels"
TEXTLINE " "
TEXTLINE "-----"
TEXTLINE "CH FUNC PRIO HSF HSR IEF ISF LNK SGL CHS PRM0"
TEXTLINE " 0,Channel 0"
BITFLD.WORD 0x1e 0.--1. " " "Off,Low,Mid,High"
```



TEXTFLD

Define text header

Format: **TEXTFLD "<text>"**

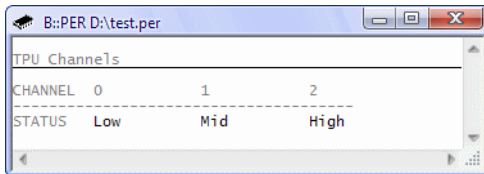
Defines text without creating a new line.

<text> Optional text.

```

GROUP d:0x80000000--0x80000fff "TPU Channels"
TEXTLINE " "
TEXTLINE "CHANNEL "
TEXTFLD " 0,Channel 0"
TEXTFLD " 1,Channel 1"
TEXTFLD " 2,Channel 2"
TEXTLINE "-----"
TEXTLINE "STATUS ,Status"
BITFLD.WORD 0x0 0.--1. " " "Off,Low,Mid,High"
BITFLD.WORD 0x0 2.--3. " " "Off,Low,Mid,High"
BITFLD.WORD 0x0 4.--5. " " "Off,Low,Mid,High"

```



TINDEX

Output a value

Format: **TINDEX** <address> [%<format>] <dataread> <datawrite>

<format>: **Byte | Word | Long | Quad | TByte | HByte**
Float. [ieee | ieeeDbI | ieeeXt | <others>]
BE | LE

Similar to **INDEX**, uses however a different sequence for write accesses: the data value is first written to the address and the index is written to trigger/transfer the write operation.

The table below shows an extract of functions useful for writing PER files.

For a complete list of available functions please see:

- [PowerView Function Reference](#)
- [General Function Reference](#)
- [Stimuli Generator Function Reference](#)

<int>	CONVert.INTTOBOOL(<bool>)	<p>Converts a boolean value to an integer. TRUE becomes 1, FALSE becomes 0</p> <p>This function allows you to write conditional base statements e.g.:</p> <pre>base VM: (0x1010*conv.booltoint(d.l(vm:0))==42) 0x1070*conv.booltoint(d.l(vm:0)!=42)</pre>
<int>	PER.ARG(<index>) and PER.ARG.ADDRESS() (deprecated)	<p>We recommend that you no longer use these two deprecated functions. Instead, use the method described in “Passing Arguments”, page 9.</p> <p>Returns the (optional) argument of the Per.view command. The parameter is currently not used. Only useful inside peripheral definition files.</p>
<int>	PER.Buffer.Byte(<index>)	Returns a byte from the SGROUP buffer. Only useful within a SGROUP of a PER file.
<int>	PER.Buffer.Word(<index>)	Returns a 16 bit word from the SGROUP buffer. Only useful within a SGROUP of a PER file.
<int>	PER.Buffer.Long(<index>)	Returns a 32 bit word from the SGROUP buffer. Only useful within a SGROUP of a PER file.
<int>	PER.Buffer.Quad(<index>)	Returns a 64 bit from the SGROUP buffer. Only useful within a SGROUP of a PER file.
<address>	PER.EVAL(<index>)	<p>Returns the value of a expression (defined with BASE) inside a peripheral definition file (PER file), which was defined after BASE, IF, ELIF or ELSE command.</p> <p>The parameter defines which expression is returned (0=first one).</p> <p>Note: The function returns only the last evaluated value of the expression. It will not evaluated the expression again. Expressions after BASE, will be evaluated by a GROUP command after the BASE command in a PER file.</p>

For information about the availability of certain commands, refer to the build numbers.

Build 98464 11. Jul. 2018	New command CSV to support comma-separated file format.
Build 97778 19. Jun. 2018	New command AUTOINDENT to automatically indent names and values.
Build 88576 08. Sep. 2017	New command NEWLINE to get a line break with indentation inside a detailed register description.
Build 72790 04. May. 2016	New command STRING to view strings from target memory.
Build 57080 07. Oct. 2014	New command WAIT to delay peripheral file interpretation.
Build 53657 16. May. 2014	New command INCLUDE to include one peripheral file in another one. New command ENTRY to allow parameterization.
Build 50692 30. Jan. 2014	Command TEXTLINE may be used outside GROUPs
Build 49992 20. Dec. 2013	New commands: TINDEX and SAVETINDEX Command renamed: OUT to INDEX , SAVEOUT to SAVEINDEX
Build 46110 31. Jul. 2013	New command FLOATMASK , to show floating point values. Allow floating point values in HEXMASK and DECMASK for <i><scale></i> .
Build 41022 11. Dec. 2012	New command RBITFLD to define read-only bit field..
Build 21955 27. Feb. 2010	New commands BASEOUT and BASESAVEOUT
Build 21439 28. Jan. 2010	New function CONV.BOOLTOINT() .
Build 21331 20. Jan. 2010	New command ASSERT .
Build 21299 19. Jan. 2010	New command VARX for SGROUP .
Build 20627 24. Nov. 2009	Enhanced SGROUP commands SET and CONSTX to accept also bitmasks and hexmasks for <i><value></i> .
Build 18839 28. Jul. 2009	New command DECMASC , to show decimal numbers.
Build 13442 28. Mai. 2008	PER programming enhanced to support nested IFs.