

# eigerScript

Programming Language for eigerPanels


## Software Manual for the eigerVirtualMachine **eVM**

English Version

Version 1.10, 19.05.2010  
update 06.11.2013 1\$00

**eigerGraphics**



**FOX** embedded computers   
*the canny swiss solution*

developed and produced by



**S-TEC electronics AG**

Industriestrasse 49  
CH-6300 Zug  
Switzerland  
[www.s-tec.ch](http://www.s-tec.ch)  
[www.eigergraphics.com](http://www.eigergraphics.com)



## Content

<b>Content .....</b>	<b>1</b>
<b>eigerScript .....</b>	<b>3</b>
Short description .....	3
Views .....	3
Syntax coloring .....	4
Data types .....	4
Constants .....	5
Variables .....	5
Registers .....	6
<b>eigerScript Methods .....</b>	<b>7</b>
Class eigerVideoEngine (EVE) .....	7
Class Display .....	8
Class File .....	12
Class String .....	13
The string concept in eigerScript .....	13
Addition of strings .....	13
Class Value .....	24
Class Label .....	27
Description of the eVM registers for a Label .....	27
Fonts on the system .....	31
Class Draw .....	33
Class Math .....	35
Integermath .....	35
Conversions from Long to Integer .....	36
Type conversions .....	37
Calculations .....	39
Class Binary .....	40
Bit functions with 16-bit or 32-bit operands .....	40
Logic functions with 16-bit or 32-bit operands .....	42
Class HotSpot .....	45
HotSpot Groups .....	45
How the HotSpots work .....	45
Class HotKey .....	52
Class Time .....	54
Class Timer .....	57
Timer: EventSection .....	59
Timer: TicSection .....	60
<b>Serial asynchronous Interfaces RS232/RS485 .....</b>	<b>63</b>
Communication modes RS485 .....	64
Class Serial .....	64
Initialize serial interface parameters .....	64
Receive characters .....	65
Send characters .....	65
Send binary data (YMODEM) .....	66
<b>Serial Two Wire Interface I2C .....</b>	<b>69</b>
Class I2C .....	69
Class Colors_15 .....	71
Change colours .....	71
Manipulate colour channels .....	74
Set or get colour values .....	78
Exchange colour channels .....	80
Class InOut .....	83
Class Fill .....	84
Class Load .....	85

Class Transfer .....	87
<b>Program control flow .....</b>	<b>88</b>
<b>Error handling .....</b>	<b>91</b>
<b>Debugging code .....</b>	<b>92</b>
Turn debug mode OFF/ON .....	92
<b>Working with CSV-Files .....</b>	<b>93</b>
What are CSV-Files ? .....	93
Class CSV .....	95
Analyse table .....	95
Retrieve data from the table.....	96
Search data in the table .....	98
Replace data in the table .....	99
<b>Revision History .....</b>	<b>100</b>
<b>Support .....</b>	<b>100</b>



## eigerScript

### Short description

eigerScript is a programming language for the eiger virtual machine eVM. With this virtual machine the eiger System is programmed. An eigerScript program is composed out of comprehensive methods. The integrated development environment IDE is called **eigerStudio**. It features syntax colouring. The source code file \*.EVS however is a plain ASCII-File. With eiger Tip the methods can be quickly selected from a list and the input parameters for the method are shown. The eigerCompiler, which is also part of the eigerStudio, compiles the \*.EVS source file to a binary file (\*.EVI). This binary file is executed from the eVM.

The commands in the eigerScript language are calls to a large library of methods already resident in the ROM of the microcontroller. For example such a command gets an image file from the CompactFlash card (CFC) and displays it on the screen. With just a few bytes this operation can be performed.

The more powerful the instructions are, the shorter is the instruction code and the time for decoding the instruction is neglectable compared to the work done by the method. In other words, the overhead is minimized having powerful instructions. The strict separation of the user program code and the virtual machine needs no painful system integration on the microprocessor. Just plug in the CFC and the system runs. With eigerScript quick responding graphical user interfaces GUI's can be realized in a sensational short time. Because of using an industrial microprocessor the system is energy efficient and long-term available. The typical lifetime of a microprocessor family is +15 years. With eigerScript on the eigerPanel touch screen solutions are interesting even for lower volume applications as exist in building automation.

### Views

The FOX embedded computers need only a small main memory since the single views are acting as a separate program. When changing a view the eVM reads the needed resources from the CFC and renders the code. The great advantage of this concept is that it takes each time the same time to display a view. As in the www the data of the view is requested in the moment it is needed. On the CFC there is nearly no limit of space compared to the file sizes of the compact byte code of the eVM.

If new views are added the system behaves in the same quick manner. Hundreds of views can be stored on the CFC.

Updating a customers system can be done as attached ZIP-File in an email. After unzipping the file and storing it on the CFC it can be plugged in the CFC socket of the FOX embedded computer and the update is done.

## Syntax coloring

The text editor of eigerStudio colors your programming script automatically according to a given syntax coloring scheme. If you like you can customize the syntax coloring scheme any time (*Extras > Options*) .

The default syntax color scheme of eigerStudio is as following:

Colour Keywords	BEGINVIEW
Colour Comments and Remarks	'Comments in Sourcecode
Colour Subroutines and Labels	SubRoutine
Colour VirtualMachine-Functions	Function()
Colour Constants	Constant
Colour Variables	Variable
Colour VirtualMachine-Registers	Register

In this document you find the programming texts colored according to the standard syntax coloring scheme of eigerStudio.

## Data types

In a computer system the data is represented by variables. The variables must be in a suitable data type. In eigerScript there are data types such as INTEGER, LONG, SINGLE (precision floating point) and STRING. The programmer is responsible of using the most suitable data type for the representation of real-world data.

Example: The voltage measurement can be represented by a SINGLE data type.

A counter counts cars in parking for example. So INTEGER is the suitable data type

eigerScript has the following data types:

- Byte range 0..255, ASCII Characters
- Integer range 0..65535 or -32768..32767
- Long range -2147483648..2147483647
- Single Single precision floating-point Number
- String Character strings, up to 65'534 Chars

VarInt	VarInt may be:	Integer number Register starting with eI . * Variable
VarLong	VarLong may be:	Long Integer number Register starting with eL . * (not yet available) Variable
VarSingle	VarSingle may be:	single precision floating point number Register starting with eS . * (not yet available)

		Variable
VarString	VarString may be:	String constant String variable String Register e\$. * (not yet available)

## Constants

Constants are defining values. In the eigerSystem many colours are predefined as system constants. As example the colour "blue" is defined by the value of 0x7C00. The programmer can define his own constants and use them in the source code.

Constants are computed by the compiler of eigerStudio at compile time with floating point accuracy.

Example: In a project buttons are used. Instead of the numeric value of the size you can define constants, for example named by BUTTON\_W and BUTTON\_H, which represent the size of the button. All buttons of the same size are referenced with BUTTON\_W and BUTTON\_H constants. So changes can be done quickly without having to search through the whole source code.

```
CONST      BUTTON_W = 120           ; Width of the button
CONST      BUTTON_H = 36           ; Height of the button

Load.Width_Height (BUTTON_W, BUTTON_H) ; load Width and Height
```

If buttons of the same size are drawn on one view, constants are used. If the size or colour of a button has to be changed, only the value of the constant need to be changed and the program can be recompiled.

Constants can be of different data types. The compiler assigns the correct data type to the variable.

```
CONST      MyInteger    = 50         ; MyInteger has the Startvalue 50
CONST      MyLong       = 150000    ; MyLong has the startvalue 150000
CONST      MySingle     = 3.1415    ; MySingle has the startvalue 3.1415
```

## Variables

Variables are memory locations that could be preassigned with a value. During run time variables hold results of calculations as example. In eigerScript variables are defined as follows:

```
INTEGER    MyInteger    = 50         ; MyInteger has the startvalue 50
LONG       MyLong       = 150000    ; MyLong has the startvalue 150000
SINGLE     MySingle     = 3.1415    ; MySingle has the startvalue 3.1415
```

## Registers

The eVM has a set of registers. The registers have predefined functions and are available without defining them in the user program. For example the registers `eI.Pos_X1` and `eI.Pos_Y1` hold a coordinate pair for drawing graphics. Some instructions read registers and put their result back in the registers.

The contents of registers “survive” a view change (`GotoView()`) but not a project change (`GotoProject(MYPR, 3)`) nor a restart of the system.

### Index of eVM-registers mentioned in this Software Manual:

<code>eI.BackColor</code> 27, 29, 86	<code>eI.Garbage</code> ..... 16	<code>eI.Offset_X10</code> , 11, 46, 86, 87
<code>eI.BackgroundColor</code> ..27	<code>eI.HalfAxis_a</code> ..... 35	<code>eI.Offset_X1</code> ..... 27
<code>eI.BacklightIntensity</code> 9	<code>eI.HalfAxis_b</code> ..... 35	<code>eI.Offset_Y10</code> , 11, 46, 86, 87
<code>eI.BacklightIntensity_</code> <code>MAX</code> .....9	<code>eI.Height</code> . 10, 11, 12, 46, 47, 86, 87	<code>eI.Offset_Y1</code> ..... 27
<code>eI.BacklightIntensity_</code> <code>MIN</code> .....9	<code>eI.HK_Tag</code> ..... 52	<code>eI.Pos_X1</code> ..10, 11, 12, 27, 33, 34, 35, 46, 47, 85, 86, 87
<code>eI.BacklightIntensity_</code> <code>Speed</code> .....9	<code>eI.HorizontalAdjust</code> 28	<code>eI.Pos_X2</code> ..... 33
<code>eI.BacklightIntensity_</code> <code>Timer</code> .....9	<code>eI.HotSpotGroup</code> 45, 46, 49, 51	<code>eI.Pos_Y1</code> ..10, 11, 12, 27, 33, 34, 35, 46, 47, 85, 86, 87
<code>eI.Boolean</code> .....21, 98	<code>eI.HotSpotTag</code> .. 46, 49	<code>eI.Pos_Y2</code> ..... 33
<code>eI.BorderStyle</code> .....28	<code>eI.HOURS</code> ..... 55, 56	<code>eI.Position</code> ..... 28
<code>eI.Brightness</code> .....9	<code>eI.HS_... registers</code> 52	<code>eI.Radius</code> ..... 34
<code>eI.DATE</code> .....55, 56	<code>eI.HS_EventType</code> .... 47	<code>eI.SEC</code> ..... 55, 56
<code>eI.DecimalSeparatorCha</code> <code>r</code> .....20	<code>eI.HS_Group</code> ..... 51	<code>eI.SpaceLeft</code> ..... 28
<code>eI.Display_DumpMode</code> .10	<code>eI.HS_Height</code> ..... 46	<code>eI.SpaceRight</code> ..... 28
<code>eI.Display_WriteMode</code> 10	<code>eI.HS_ID</code> ..... 46	<code>eI.Status</code> ..24, 25, 26, 37, 91
<code>eI.DisplayColor</code> .8, 29, 84	<code>eI.HS_Pos_X</code> ..... 46	<code>eI.TextColor</code> ...27, 29, 84, 86
<code>eI.DOW</code> .....55	<code>eI.HS_Pos_Y</code> ..... 46	<code>eI.VerticalAdjust</code> ...28
<code>eI.ErrorCode</code> ...17, 18, 20, 24, 25, 26, 91	<code>eI.HS_State</code> ..... 46	<code>eI.Width</code> ...10, 11, 12, 46, 47, 86, 87
<code>eI.eVM_DebugMode</code> ....92	<code>eI.HS_Tag</code> .. 45, 46, 49	<code>eI.XSTART</code> ..... 7
<code>eI.FillChar</code> .....19, 20	<code>eI.HS_Width</code> ..... 46	<code>eI.YEAR</code> ..... 55, 56
<code>eI.FillColor</code> ...27, 29, 34, 84, 86	<code>eI.LineColor</code> .. 27, 29, 33, 34, 35, 84, 86	
<code>eI.FontNumber</code> ..27, 31, 32	<code>eI.MIN</code> ..... 55, 56	
	<code>eI.MONTH</code> ..... 55, 56	
	<code>eI.Mouse_X_Down</code> .... 83	
	<code>eI.Mouse_Y_Down</code> .... 83	
	<code>eI.MSEC</code> ..... 55, 56	
	<code>eI.NumericChar</code> ..... 20	

Registers are kind of “super global” variables. An integer value stored in a register survives a view change and even a project change until another value is set and until the `eigerPanel` is shut down respectively.

Most registers are reserved for specific functions and properties. Following registers are for free use:

<code>eI.R00</code>	<code>eI.R05</code>	<code>eI.R10</code>	<code>eI.R15</code>
<code>eI.R01</code>	<code>eI.R06</code>	<code>eI.R11</code>	<code>eI.Garbage</code>
<code>eI.R02</code>	<code>eI.R07</code>	<code>eI.R12</code>	
<code>eI.R03</code>	<code>eI.R08</code>	<code>eI.R13</code>	
<code>eI.R04</code>	<code>eI.R09</code>	<code>eI.R14</code>	



## eigerScript Methods

In this chapter you find the eigerScript methods described. The methods are assigned to different classes according to their function. This is a great help to orientate within the eigerScript. For example if you want to implement a serial interface the methods of class Serial are certainly of first choice.

Some methods are highlighted by blue and some by yellow bars. Yellow bars signalize those methods, which are – in our opinion – the most important.

0\$50

At some methods you find the version number of the eiger virtual machine (eVM) from which on the method is can be used. The actual eVM version on your eigerPanel is indicated at the top of the screen while the system is booting (e.g. eigerVM V1.00).

### Class eigerVideoEngine (EVE)

This methods interact directly with the eigerVideoEngine EVE. With the methods of this class the behaviour of the graphics are controlled.

#### **EVE.Init ( )**

0\$40

The **EVE.Init** method puts the horizontal start position to zero and resets the **eI.XSTART** register. The write mode is set without transparency and not inverted. The transparent mode is set to “transparence disable” and the AVR and the RVR video memory are written simultaneously. After this method the graphics are ready to use. It is recommended to initialize the graphics in the beginning of a new view in order to prevent previous views with different settings to disturb the result of the rendering of a view.

#### **EVE.Load\_XSTART (VarInt:X-Position)**

The video memories AVR and RVR have 1024 pixels in the X-direction (landscape). With this method the start of the displayed region is controlled. If the sum of the start position and the image width is greater than 1024, the image is scrolled.

#### **EVE.WriteEnable\_AVR ( )**

With the method **EVE.WriteEnable\_AVR ( )** the write access to the readable AVR-Video-RAM is enabled. The writing to the RVR is disabled. With this method a view can be prepared in the background. With the **Display.Show ( )**-method the view can be displayed. See also **Display.Prepare ( )**.

**EVE.WriteEnable\_RVR()**

With the method **EVE.WriteEnable\_RVR()** the write access to the RVR-Video-RAM is enabled and the write access to the AVR-Video-RAM is disabled. This method can be used to write pop-ups as controls or warnings on the display without destroying the background. A similar method is **Display.Direct()**.

**EVE.WriteEnable\_AVR\_RVR()**

With the method **EVE.WriteEnable\_AVR\_RVR()** the write access is enabled in both Video-RAM's. This is the normal operating mode after EVE.Init. The EVE graphics engine without time delay doubles the video content. In this way there is an original and a copy of the data. The copy is displayed and if disrupted by a pop-up, it can be restored from the original, which is stored in the AVR video RAM.

**EVE.Load\_Transparency()**

With the method **EVE.Load\_Transparency()** the EVE graphics engine is programmed to load the transparency colour in the video-RAM. This method is used, when not rectangular shapes are used as sprites. Normally this mode is off.

**EVE.Process\_Transparency()**

With the method **EVE.Process\_Transparency()** the EVE graphics engine is programmed that the transparency colour is **not** loaded in the video-RAM when a transparent pixel is written to the EVE. This is the normal configuration after **EVE.Init**. **With this setting it is possible to write on an image background.**



## Class Display

Methods of class "Display" control the behaviour of the display content.

**Display.Clear()**

0\$40

With the method **Display.Clear()** the content of the screen is cleared with the colour stored in the register **eI.DisplayColor**. After a reset the register **eI.DisplayColor** is set to silver as default.

Example:     `Display.Clear()`                     `; Display is cleared`

Remark:

The size of the screen is defined in the firmware, thus the programmer has not to bother about the screen size.

## **Display.ClearColor(VarInt:Color)**

With the method `Display.ClearColor()` the screen is cleared (filled) with the colour given as parameter.

Example:     `Display.Clear(blue)`     `; display is cleared with blue colour`

## **Display.SetBrightness()**

For FOXS computer with CCFL Backlight TFT this method will set the intensity of the backlight stored in the `eI.Brightness` register from 0..0x0F with 0 full dark and 0x0F full bright.

## **Display.BacklightIntensity\_Set(VarInt:Intensity)**

Since eVM V1.01 this method sets the backlight intensity for TFT-panels with LED backlight. The register `eI.BacklightIntensity` holds the current value of the backlight intensity.

- The register `eI.BacklightIntensity_MAX` holds the value for the maximum brightness
- The register `eI.BacklightIntensity_MIN` holds the value for the minimum brightness
- The register `eI.BacklightIntensity_Timer` holds the time in seconds until the backlight is dimmed after no touch event is occurred.
- The register `eI.BacklightIntensity_Speed` holds the value in ms for dimming the display

## **Display.Prepare()**

The video memory consists of two video planes called AVR (accessible video RAM) and RVR (refresh video RAM). While the content of the AVR is visible on the display, the content of RVR is not visible as long it is not copied to the AVR. The microcontroller can read and write the AVR but only perform writes to the RVR. The video controller uses the RVR to refresh the display..

The method `Display.Prepare()` disables the write access to the RVR and only the AVR is written. The method `Display.Show()` copies the content of the AVR to the RVR and the content can be seen on the display. This way you can prepare the layout of your view (e.g. background

picture and buttons and labels etc.) on the “hidden” RVR and than instantly make the whole layout visible on the display by copying to the AVR.

The register `eI.Display_WriteMode` is set to `Write_To_AVR`.

## Display.Direct()

The method `Display.Direct()` disables the write access to the AVR video RAM. All Outputs are **directly visible** on the display. The method is used to show a pop-up without overwriting the original display content in the AVR. With the method `Display.Show()` the content of the AVR is copied to the RVR and the original content is restored. For the user the effect is that the popup disappears.

A `CallSubroutine()` saves the state of the write mode and restores it after returning.

With the register `eI.Display_DumpMode`, which is per default set on `false`, it can be selected whether the method works normal or when set `true` all writes are to the AVR *and* the RVR video RAM. (`eI.Display_DumpMode := true`). This is useful to document views with pop-up menus with the screenshot function. The register `eI.Display_WriteMode` is set to `Write_To_RVR`.

What is AVR and RVR? → `Display.Prepare()`

## Display.Show()

The method `Display.Show()` copies the content of the AVR to the RVR video RAM and makes it visible for the user. This method is used in combination with the methods `Display.Prepare()` or `Display.Direct()`. See also the description for these methods.

With the method the whole screen is copied. The similar method `Display.ShowWindow()` copies only a window of the screen.

## Display.ShowWindow()

The method `Display.ShowWindow()` copies a rectangular window of the AVR to the RVR video RAM and thus makes the window visible to the user. The parameters for the method are written in the following registers:

<code>eI.Pos_Xl</code>	X-position upper left corner
<code>eI.Pos_Yl</code>	Y-position upper left corner
<code>eI.Offset_X</code>	Offset X
<code>eI.Offset_Y</code>	Offset Y
<code>eI.Width</code>	width
<code>eI.Height</code>	height

Tip: When a label is often refreshed to display a changing value, the method can be used to prevent the label from flickering.

**Display.CopyWindow( )**

The method **Display.CopyWindow( )** copies a rectangular window of the display to another place on the display. The parameters for the method are written in the following registers:

<b>eI.Pos_X1</b>	X-position upper left corner
<b>eI.Pos_Y1</b>	Y-position upper left corner
<b>eI.Offset_X</b>	Offset X
<b>eI.Offset_Y</b>	Offset Y
<b>eI.Width</b>	width
<b>eI.Height</b>	height
<b>eI.Pos_X2</b>	X-position upper left corner of the target window
<b>eI.Pos_Y2</b>	Y-position upper left corner of the target window

Example:

```
Load.Geometry_XYWH(300, 200, 50, 800) ; window to be copied
Load.Pos_X2Y2(400, 300) ; target position for copied window
Display.CopyWindow() ; executes the copy process
```



## Class File

With the methods of the class File the content of the CompactFlash card is accessed.

There are specialized methods to read a graphics file and put it in the video RAM or to read a Comma Separated Value file (\*.CSV). This file type can be read and written by Excel and with the \*.CSV it is possible to have access to data stored in spread sheet tables. See class CSV for further information (p. 95).

### **File.Read\_EGI(VarStr:FileName)**

The method **File.Read\_EGI(FileName)** reads a graphic image in the \*.EGI-format and writes it directly to the video RAM. The method is optimized for speed with the file system and the display driver software working closely together.

<b>eI.Pos_X1</b>	X-position upper left corner
<b>eI.Pos_Y1</b>	Y-position upper left corner

As side effect the method puts the width and the height of the image to the following registers:

<b>eI.Width</b>	Width of the image
<b>eI.Height</b>	height of the image

Tip: The registers can be used to draw a border around the picture or to install a hotspot on the picture.

The FileName String can be of the following structures:

```
File.Read_EGI('C:\\TG12\\IMAGE_A.EGI') ; whole path
File.Read_EGI('IMAGE_B.EGI')          ; short form: IMAGE_B is in same folder
File.Read_EGI('C:\\PICT\\IMAGE_C.EGI') ; IMAGE_B is in another folder
File.Read_EGI('C:\\TG12\\PICT\\IMAGE_D.EGI') ; IMAGE_B is in a subfolder
```

Possible errors: The method can indicate the following errors (see error handling for details):

<b>ERR_File_CFC_missing</b>	CFC is missing
<b>ERR_File_not_found</b>	File is not found
<b>ERR_eVM_EGI_corrupt</b>	EGI is corrupt

### **File.DeleteFile(VarStr:FileName)**

The method **File.DeleteFile(FileName)** deletes a file with a given filename.

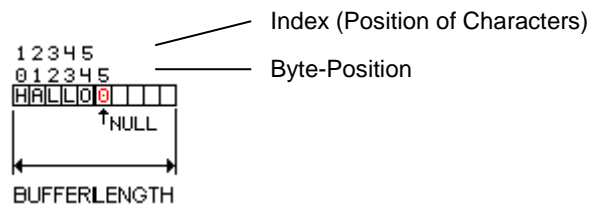
Example:

```
File.DeleteFile('C:\\TG12\\DATA\\TEST.TXT') ; delete file TEST.TXT
```

## Class String

The methods of the class string use arrays of characters to perform their operations. With eigerScript every string has its own buffer length. It is not possible to have strings with more characters than the buffer length stored in a string. Internally the string is terminated with a NUL 0x00 character. The user doesn't have to bother about this string termination. A string can contain up to 65'534 Characters. Thus even small files can be stored in a string.

The methods of the class string are used to manipulate incoming and outgoing strings and to combine texts with numeric representations of variables.



```
STRING [10] MyString.$ = 'HELLO'
```

With this declaration a string is declared with the name MyString.\$ and the buffer length of 10 characters. The string is initialized to the value 'HELLO'. This shows the difference between the string length (=5) and the buffer length(=10). If there is no buffer length specified the buffer length is assigned to 256 by default. Especially when short strings are used this may mean a waste of memory.

For indexing single characters in a string the position of the character is used. Please note that the first character is on position 1 while the last character of the string is at the position, which corresponds to the string length. The character O of the string 'HELLO' is at position 5.

### The string concept in eigerScript

Strings are the starting point of data being rendered on the display screen, transmitted over serial communication lines, protocols or data storage. With the methods of the class Strings the character arrays (strings) can be manipulated, interpreted, cut, chained etc. In this way values transmitted by a connected system can be extracted from a command set or strings for a communication can be formed.

In eigerScript a string has a header, which stores information about the string, as for example the buffer length. The buffer length is used to prevent a memory overflow when a long string is assigned to a short string. The operation is performed on a "best effort" basis. A warning can be analyzed after the method. This feature frees the user of many code lines of error handling although having a robust system.

For extracting numeric values out of a string, please refer to the class Value (p. 24).

### Addition of strings

Strings can simply be merged by adding one String to another with "+" or "&". This operation is limited to two Strings. The addition of three String needs to be done in 2 steps.

Examples:

```
STRING      [32] StringA.$ = 'TextA'
STRING      [32] StringB.$ = 'TextB'
STRING      [32] StringAB.$ = ''
```

```
StringAB.$ := StringA.$ + StringB.$ ; Result: TextATextB
```

or

```
StringAB.$ := StringA.$ & StringB.$ ; Result: TextATextB
```

### **Str.Copy(VarStr:Target,VarStr:Source)**

The method **Str.Copy(TargetString,SourceString)** is used to copy the source string to the target string. If the target string is shorter than the source string, this method copies as many characters as find space in the target string. The method overwrites the previous content of the target string.

### **Str.CopySubstringWord(VarStr:Target,VarStr:Source,VarInt:pos)**

The method **Str.CopySubstringWord(targetstring,sourcestring,position)** appends the word starting at the position of the source string to the target string. If the target string is too short or too full to contain the word, as many characters as possible are copied to the target string.

Example:

```
; Declaration of TargetString:
STRING      [30] Text_1.$ = 'Button 1'
; Deklaration of SourceString :
STRING      [30] Text_2.$ = 'abcd'

; appending part of SourceStr to TargetStr:
Str.CopySubstringWord(Text_1.$,Text_2.$,3)
```

Content of TargetString after using the method above: `Text_1.$ = Button 1cd`

### **Str.Length(VarInt:Length,VarStr:source)**

The method **Str.Length(Length,SourceString)** is used to evaluate the number of characters in a string. All characters except the terminating NUL-Character count for the string length. **Str.Length(Lenght.I,'Hello')** evaluates `Lenght.I = 5`.



**Str.BufferLength(VarInt:Lenght,VarStr:SourceString)**

The method **Str.BufferLength**(Length,SourceString) replies with the buffer length as defined in the string declaration. It corresponds to the maximum number of characters to be stored in this string.

**Str.SpaceInString(VarInt:space,VarStr:SourceString)**

The method **Str.SpaceInString**(Space,SourceString) replies the number of characters that can be stored in the string until it is full. The return value is the difference of the Buffer Length and the String Length.

**Str.Clear(VarStr:TargetString)**

The method **Str.Clear**(TargetString) clears the target string. The first character in the string is NUL, thus the length is 0. The buffer size remains unchanged.

**Str.AddChar(VarStr:TargetString,VarInt:Character)**

The method **Str.AddChar**(TargetString,VarInt) adds a character to the target string. If there is no space left in the string, the character will not be added.

Example 1:

```
Str.AddChar(MyString.$,"A")
```

The character "A" is added to MyString.\$

Example 2:

```
Str.AddChar(MyString.$,0x41)
```

The character "A" is added to MyString.\$ . The ASCII value of A is 0x41.

**Str.Add\_CRLF(VarStr:TargetString)**

The method **Str.Add\_CRLF**(TargetString) adds a carriage linefeed to the targetstring; one Character for CR and one character for LF. If there is no space left in the string for at least two characters, CRLF will not be added.

What is CRLF?

CR = Carriage return > other formats: '\r', 0x0D, 13 in decimal

LF = Line feed > other formats: '\n', 0x0A, 10 in decimal

### Str.FillUpWithChar(VarStr:TargetString,VarInt:Char)

The method **Str.FillUpWithChar(targetstring,VarInt:char)** fills up the target string with the specified character until the string is filled according to its buffer length. This method is useful to fill up a string with space characters.

Example:

**STRING** [10] **MyString.\$** = 'ABC' ; MyString.\$ consists of characters ABC and may contain up to 10 characters

```
Str.FillUpWithChar(MyString.$,"*")
```

**MyString.\$** is after the method: ABC\*\*\*\*\*

### Str.RemoveLastChar(VarInt:RemovedChar,VarStr:String)

The method **Str.RemoveLastChar(String,VarInt)** removes the last character of a string and returns it to the VarInt parameter specified. If the method is performed on a empty string the return value is NUL. If the return value is not of interest, the register **eI.Garbage** can be used to receive the last character. The method is especially useful to implement a delete function that deletes the last character.

```
Str.RemoveLastChar(eI.Garbage,MyString.$)
```

Str.RemoveLastChar(eI.Garbage.TargetString)																			
SourceString	10	1	2	3	4	5	6	7	8	9	0	0							
TargetString old	10	1	2	3	4	5	6	7	8	9	0	0							
TargetString new	9	1	2	3	4	5	6	7	8	9	0								

### Str.Concat(VarStr:TargetString,VarStr:SourceString)

The method **Str.Concat(TargetString,SourceString)** adds the SourceString to the TargetString. The method is used to prepare an output to a label. If the sum of lengths of the two strings (TargetString plus SourceString) is greater than the buffer length of the TargetString, only as many characters as find space are transferred to the TargetString.

```
Str.Concat(TargetString,SourceString) ; Strings are concatenated
```

Str.Concat(TargetString,SourceString)																			
SourceString	6	a	b	c	d	e	f	0											
TargetString old	10	1	2	3	4	5	6	7	8	9	0	0							
TargetString new	16	1	2	3	4	5	6	7	8	9	0	a	b	c	d	e	f	0	

Alternatively you may also write:

```
TargetString := TargetString + SourceString
```

or:

```
TargetString := TargetString & SourceString
```

## Possible errors:

The following errors may occur:

- a) TargetString was full: `eI.ErrorCode == ERR_String_NoSpaceInString`
- b) TargetString has not enough space: `eI.ErrorCode == WAR_String_NotEnoughSpace`

## **Str.UpperCase(VarStr:TargetString)**

The method `Str.UpperCase(TargetString)` converts the characters of the TargetString to capital characters. The codepage used is the ISO Latin-15 respectively the CP1252. Not only the characters from a..z are converted to the capitals A..Z, but also the characters in the codepage with a code above 127 are converted. For example the character ö is converted to Ö. The method is useful to exclude problems with user inputs in advance of a string comparison.

```
Str.UpperCase(TargetString) ; converts a string to capitals
```

## **Str.LowerCase(VarStr)**

The method `Str.LowerCase(TargetString)` converts the TargetString to small letters. The codepage used is the ISO Latin-15 respectively the CP1252. Not only the characters from A..Z are converted to the small letters a..z, but also the characters in the codepage with a code above 127 are converted. For example the character Ö is converted to ö. The method is useful to exclude problems with user inputs in advance of a string comparison.

```
Str.LowerCase(MyString.$) ; MyString.$ is converted to small letters
```

## **Str.Compare(VarInt:Result,VarStr,VarStr)**

The method `Str.Compare(Result,String1,String2)` compares strings 1 and 2 character by character. If the strings are identical, that is of the same characters and the same length, the return integer value is 0. Otherwise the difference of the first differing characters is returned. This can be used for sorting strings. If the result is negative, the string 1 is after string 2 in alphabetical order.

The method is suitable to compare passwords with user input.

## Example:

```
Str.Compare(Result.I,MyString.$,'ABC') ; compare MyString.$ with ABC
```

**Str.Cvt\_ByteHex(VarStr:TargetString,VarInt:Value)**

The method **Str.Cvt\_ByteHex(TargetString,Integer)** adds two ASCII-characters that represent the value of the input parameter, to a string.

```
Str.Cvt_ByteHex(MyString.$,eI.R00)    ; add eI.R00 hexadecimal to  
                                      ; MyString.$
```

Example: 68

Tip: If the representation 0x68 or 68h has to be used, the string is preloaded with 0x or the character „h“ is added.

Possible errors: If there is no space in string, the method doesn't add the characters to the string and the error code is returned in the register **eI.ErrorCode**.

Remark I: Only the lower Byte is output.

Remark II: By this method you *add* the numbers to the string specified, that means you append the numbers at the end of the already existing string content. If you want to be sure that the string exclusively contents the numbers added, you may better clear the whole string content in advance, e.g. by the method **Str.Clear(TargetString)**.

**Str.Cvt\_WordHex(VarStr:TargetString,VarInt:Value)**

The method **Str.Cvt\_WordHex(TargetString,Integer)** adds 4 ASCII-Characters that represent the value, to a string.

```
Str.Cvt_WordHex(MyString.$,eI.R00)    ; eI.R00 is hexadecimal added to  
                                      ; MyString.$
```

Example: FE68

Tip: If the representation 0xFE68 or FE68h has to be used, the string is preloaded with 0x or the character „h“ is added.

Possible errors: If there is no space in string, the method doesn't write to the string and the error code is returned in the register **eI.ErrorCode**.

Remark: By this method you *add* the numbers to the string specified, that means you append the numbers at the end of the already existing string content. If you want to be sure that the string exclusively contents the numbers added, you may better clear the whole string content in advance, e.g. by the method **Str.Clear(TargetString)**.

**Str.Cvt\_LongHex(TargetStr,Long)**

The method **Str.Cvt\_LongHex(TargetString,Long)** adds 8 ASCII-Characters to the string, that represent the hex value of the VarLong.

```
Str.Cvt_LongHex(MyString.$,MyLong.L) ; MyLong.L is added hexadecimal to
                                     ; MyString.$
```

Errors: If there is no space in string, the method doesn't write to the string and the error code is returned in the register **eI.ErrorCode**.

Remark: By this method you *add* the numbers to the string specified, that means you append the numbers at the end of the already existing string content. If you want to be sure that the string exclusively contains the numbers added, you may better clear the whole string content in advance, e.g. by the method **Str.Clear(TargetString)**.

**Str.Cvt\_Long(TargetStr,Long,Int)**

The method **Str.Cvt\_Long(TargetString,Long,Positions)** adds a Long value to the TargetString.

```
Str.Cvt_Long(MyString.$,MyLong.L,8) ; MyLong.L is added decimal
                                     ; to MyString.$ with 8 positions
```

Example: MyLong.L is the number 105'600. The output is with 8 positions. In the example, MyString.\$ contains already: "Counter:"

The register **eI.FillChar** can be preloaded: **eI.FillChar := ""**

MyString after performing the method is then: Counter:\*\*105600

Special case: If the number of positions is 0, then the output is not formatted.

Errors: If there is no space in string, the method doesn't write to the string and the error code is returned in the register **eI.ErrorCode**.

If the number needs more digits than requested positions, the requested positions are filled up with the character preloaded in **eI.NumericChar**.

Remark: By this method you *add* the numbers to the string specified, that means you append the numbers at the end of the already existing string content. If you want to be sure that the string exclusively contains the numbers added, you may better clear the whole string content in advance, e.g. by the method **Str.Clear(TargetString)**.

**Str.Cvt\_LongDeci(VarStr,VarLong,VarInt,VarInt)**

**Str.Cvt\_Integer(TargetString,Long,LeadingDigits,TrailingDigits)** formats an output of a decimal number represented by a Long with leading and trailing decimals. The method is useful to output a number, which is internally in another unit.

Example: For a payment system the base unit is cent. The price of a product should be represented in Swiss Francs or in EURO. Internally all calculations are performed in the base unit  
`MyPrice.L := 13260`

The output should be formatted: 132.60.

The method `LongDeci` consumes the leading and the trailing decimals.

```
Str.Cvt_LongDeci(MyString.$,MyPrice.L,3,2)
```

Example: The value of `MyLong.L` is 260. The output should be with 4 leading and 2 trailing decimals. `MyString.$` is: `Price:`

With the register preloaded as: `eI.DecimalSeparatorChar := ","`

The output then is `Price: 2,60`

Tip: The register `eI.FillChar` is preloaded with the character used to fill up unused leading digits. In the case of preparing a label the register could be preloaded with `0xA0` as `FillChar`. This is a space character with the same glyph width as the numeric glyphs. This has the advantage of the string being always the same string width.

Special case: If the number of leading digits is 0, the output is not formatted.

Possible errors: If there is no space in string, the method doesn't write to the string and the error code is returned in the register `eI.ErrorCode`.

If the number needs more digits than requested positions, the requested positions are filled up with the character preloaded in `eI.NumericChar`.

Remark: By this method you *add* the numbers to the string specified, that means you append the numbers at the end of the already existing string content. If you want to be sure that the string exclusively contents the numbers added, you may better clear the whole string content in advance, e.g. by the method `Str.Clear(TargetString)`.

### **`Str.Cvt_Integer(TargetString,IntegerValue,Digits)`**

The method `Str.Cvt_Integer(TargetString,Integer,Digits)` adds the numeric representation of an integer-number to a string. The number of digits can be set as parameter.

```
Str.Cvt_Integer(MyString.$,MyInteger.I,4) ; MyInteger.$ is appended to
                                           ; MyString.$ in decimal
                                           ; 4 digit representation
```

The behaviour of the method is equal to the method `Str.Cvt_Long`.

Remark: By this method you *add* the numbers to the string specified, that means you append the numbers at the end of the already existing string content. If you want to be sure that the string exclusively contents the numbers added, you may better clear the whole string content in advance, e.g. by the method `Str.Clear(TargetString)`.

**Str.Find(VarStr,VarInt,VarStr,VarInt)**

**Str.Find**(CompareString,StartPosition,SearchString,Returnvalue) scans the CompareString from the StartPosition against the SearchString. If the SearchString is found in the CompareString the position of the next character after the SearchString is returned. If the SearchString is not found in the CompareString the return value is 0.

Example:

```
Str.Find(InputString.$,1,'Booster',StringPosition.I)
IF StringPosition.I > 0 THEN
    ; String 'Booster' found
ELSE
    ; String 'Booster' not found
ENDIF
```

Possible errors: If the requested position is greater than the string length, the return value is 0.

**Str.Match(VarStr,VarInt,VarStr)**

**Str.Match**(VarStr:CompareString,VarInt:StartPosition,VarStr:SearchString) is the method, which compares the SearchString from the StartPosition in the CompareString. If the SearchString matches exactly, the register **eI.Boolean** is set to true; in all other cases it is set to false.

Example:

```
Str.Match('Watertube',6,'tube')
IF eI.Boolean == true THEN
    ; tube found !
ENDIF
```

Possible errors: If the requested position is greater than the string length, the return value is 0.

**Str.GetPosition\_by\_Char(VarInt:FoundPosition,VarInt:Char,VarInt:StartPosition,VarStr:InputString)**

The method **Str.GetPosition\_by\_Char**(FoundPos,Char,StartPos,InpStr) detects the position of a given character. It starts its search at a given position.

Example:

```
INTEGER FoundPosition.I
INTEGER Char.I = "n"
INTEGER StartPosition.I = 5
STRING [32] MyString.$ = 'Find Position of my Char'

Str.GetPosition_by_Char(FoundPosition.I,Char.I,StartPosition.I,B02_Text)
```

Result: FoundPosition.I is 13 (if we set StartPosition.I = 2 then FoundPosition.I would be 3)

**Str.SkipBlank(VarStr,VarInt,VarStr)**

The method **Str.SkipBlank(String,StartPosition,NextPosition)** skips the white-space from the StartPosition until the first non-whitespace-character in the string occurs. If the string end is reached or the StartPosition is greater than the string length the return value is 0.

Example:

```
Str.SkipBlank('    Hello',1,Position.I)
```

after the method the variable **Position.I** has the value 5, since four space characters are skipped.

Following characters are defined as withespace-characters: 0x20 (Space) 0x09 (Tabulator) and 0xA0 (Space).

**Str.Time(VarStr:TargetString,VarInt:FormatTime)**

The Method **Str.Time(TargetString,FormatTime)** is used to fill a string with the current time. You can choose between four format types:

```
FormatTime_XX_XX_XX      ; Example: ??:?:?? with eI.FillChar := "?"
FormatTime_HH_MM_SS      ; Example: 21:45:36
FormatTime_HH_MM_SS_XXX  ; Example: 21:45:06.230
FormatTime_HH_MM         ; Example: 08:05
```

Don't forget to get the current time and date from the Real Time Clock before using **Str.Time(TargetString,FormatTime)**. Cf. **Time.Get()** at p.55.

Example 1:

```
Time.Get()                ; gets Time and Date from the Real Time Clock
eI.FillChar := "?"        ; "?" or 0x3F
Str.Time(Time.$, FormatTime_XX_XX_XX) ; Result: ??:?:??
```

Example 2:

```
Time.Get()                ; gets Time and Date from the Real Time Clock
Str.Time(Time.$, FormatTime_HH_MM_SS) ; Result: 03:58:29
```

**Str.Date(VarStr:TargetString,VarInt:FormatDate)**

The Method **Str.Date(TargetString,FormatDate)** is used to fill a string with the current Date. You can choose between three format types:

```
FormatDate_XX.XX.XXXX    ; Example: ??:?:???? with eI.FillChar := "?"
FormatDate_DD.MM.YYYY    ; Example: 12.08.2010
FormatDate_DD.MM.YY      ; Example: 12.08.10
```



---

```
FormatDate_YYMMDD           ; Example: 100812
FormatDate_ISO8601_YYYYMMDD ; Example: 20100812
FormatDate_ISO8601_YYYY_MM_DD ; Example: 2010-08-12
```

Don't forget to get the current time and date from the Real Time Clock before using **Str.Date(TargetString,FormatDate)**. Cf. **Time.Get()** at p.55 .

## Example 1:

```
Time.Get()           ; gets Time and Date from the Real Time Clock
eI.FillChar := "?"   ; "?" or 0x3F
Str.Date(Date.$, FormatDate_XX.XX.XXXX) ; Result: ??:?:????
```

## Example 2:

```
Time.Get()           ; gets Time and Date from the Real Time Clock
Str.Date(Date.$, FormatDate_ISO8601_YYYY_MM_DD) ; Result: 2010-12-09
```



## Class Value

With the methods of the class Value strings are converted to numeric values of a particular data type. There are methods to read numeric values in decimal and in hexadecimal representation.

### Value.ByteHex(VarInt:Val,VarStr:Str,VarInt:Pos)

The method `Value.ByteHex(VarInt:Val,VarStr:Str,VarInt:Pos)` gets two ASCII-characters from the StartPosition to convert them to a ReturnValue. The characters have to be in the range [0..9] | [A..F] | [a..f]. The range of the ReturnValue is [0x0000..0x00FF].

#### Example:

In `InputString.$` is 'TEST=A9'

```
Value.ByteHex( 'InputValue.I',InputString.$,6)
after this method the value of InputValue.I is: 0x00A9.
```

Possible errors: If there are not two consecutive HEX-Characters, or the position is outside the string or at the specified position the string end is reached an error occurs. The register `eI.Status` := error and the error code is stored in `eI.ErrorCode`.

### Value.WordHex(VarInt:Val,VarStr:Str,VarInt:Pos)

The method `Value.WordHex(VarInt:Val,VarStr:Str,VarInt:Pos)` gets four ASCII-Characters from the StartPosition to convert them to a ReturnValue. The characters have to be in the range [0..9] | [A..F] | [a..f]. The range of the ReturnValue is [0x0000..0xFFFF].

#### Example:

The content of `InputString.$` is: 'TEST=A95e'

```
Value.WordHex( 'InputValue.I',InputString.$,6)
after this method the value of InputValue.I is: 0xA95E.
```

Possible errors: If there are not four consecutive HEX-Characters, or the position is outside the string or at the position the string end is reached an error occurred. The register `eI.Status` := error and the error code is stored in `eI.ErrorCode`.

### Value.LongHex(VarLong:Val,VarStr:Str,VarInt:Pos)

The method `Value.LongHex(VarLong:Val,VarStr:Str,VarInt:Pos)` gets eight ASCII-characters from the StartPosition to convert them to a ReturnValue. The characters have to be in the range [0..9] | [A..F] | [a..f]. The range of the ReturnValue is [0x00000000..0xFFFFFFFF].

#### Example:

The content of `InputString.$` is: 'TEST=A95e76a9'

```
Value.LongHex( 'InputValue.L',InputString.$,6)
after this method the value of InputValue.L is: 0xA95E76A9.
```

Possible errors: If there are not eight consecutive HEX-Characters, or the position is outside the string or at the position the string end is reached an error occurred. The register **eI.Status** := error and the error code is stored in **eI.ErrorCode**.

#### **Value.UInteger (VarInt:Number, VarStr:Str, VarInt:Pos)**

The method **Value.UInteger (VarInt:Number, VarStr:Str, VarInt:Pos)** reads ASCII-characters from the StartPosition and converts them to an unsigned integer number. The characters have to be numeric digits in the range of [0..9]. The return value has a range of [0..65535].

##### Example:

The content of **InputString.\$** is: 'TEST= 675'

```
Value.UInteger ( InputValue.I, InputString.$, 6)
```

after this method the value of **InputValue.L** is: 675.

Possible errors: If a number is too large for the representation by an unsigned integer or the position is outside the string an error occurs. The register **eI.Status** := error and the error code is stored in **eI.ErrorCode**.

#### **Value.Integer (VarInt:Number, VarStr:Str, VarInt:Pos)**

The method **Value.Integer (VarInt:Number, VarStr:Str, VarInt:Pos)** reads ASCII-characters from the StartPosition and converts them to a signed integer number. The characters have to be numeric digits in the range of [0..9]. The return value has a range of [-32'768..0..32'767].

##### Example:

The content of **InputString.\$** is: 'TEST= -675'

```
Value.Integer ( ' InputValue.I, InputString.$, 6)
```

after this method the value of **InputValue.I** is: -675.

Possible errors: If a number is too large for the representation by a signed integer or the position is outside the string an error occurs. The register **eI.Status** := error and the error code is stored in **eI.ErrorCode**.

#### **Value.Long (VarLong:Number, VarStr:Str, VarInt:Pos)**

The method **Value.Long (VarLong:Number, VarStr:Str, VarInt:Pos)** reads ASCII-characters from the StartPosition and converts them to a unsigned long number. The characters have to be numeric digits in the range of [0..9]. The return value has a range of [-2'147'483'648..0.. 2'147'483'647].

##### Example:

The content of **InputString.\$** is: 'TEST= 100675'

**Value.Integer**( **InputValue.L**, **InputString.\$**, 6 )

after this method the value of **InputValue.L** is: 100675.

Possible errors: If a number is too large for the representation in a unsigned integer or the position is outside the string an error occurred. The register **eI.Status** := error and the error code is stored in **eI.ErrorCode** .

---

## Class Label

With the methods of the class label, Buttons and texts can be rendered on the display. The label object is quite flexible but for this a bit more complex. Sixteen registers of the eVM describe the attributes of the object. Note that all geometric dimensions are given in pixels.

### Description of the eVM registers for a Label

#### Position:

The registers **eI.Pos\_X1** and **eI.Pos\_Y1** define the position of the upper left corner of the rectangle defining the label. The absolute X/Y-position on the display screen is the sum of the position register pair and the X/Y-Offset register pair, that is **eI.Offset\_X1** and **eI.Offset\_Y1**.

#### Size:

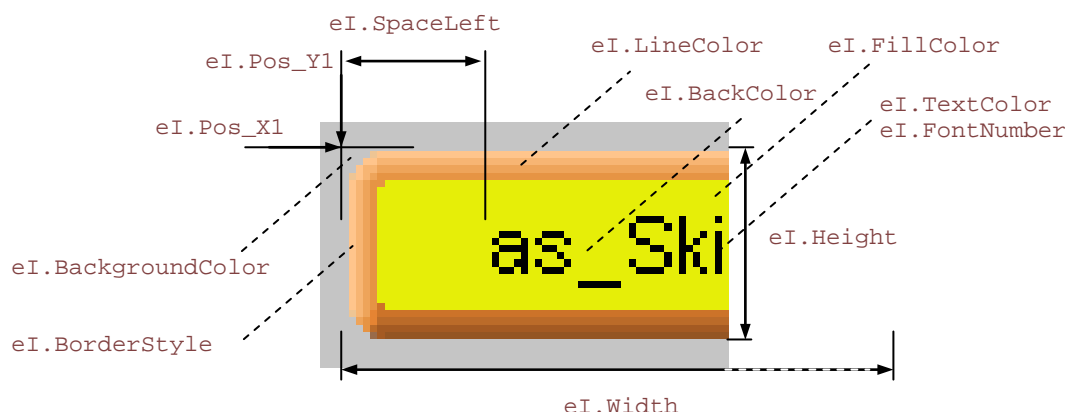
For the width and the height of the label object the registers **eI.Width** and **eI.Height** are preloaded with the appropriate pixel values.

#### Colours:

The label object is characterized by four colours:

<b>eI.FillColor</b>	colour of the body of the label
<b>eI.LineColor</b>	colour of the outline if the label
<b>eI.BackColor</b>	colour of the text background of the label
<b>eI.TextColor</b>	colour of the text on the label
<b>eI.BackgroundColor</b>	colour of the background on which the label is rendered.

normally the BackColor is the same as the FillColor.



#### Font of the text:

The text font can be selected by the register **eI.FontNumber**. For the details of the installed fonts see chapter "Fonts on the system".

Position of the text:

The rendered text can be positioned on the label with the register **eI.Position**. The register can accept the following constants:

```
eI.Position := Pos_topleft      ; text at upper left corner
eI.Position := Pos_top         ; text at the top centre
eI.Position := Pos_topright    ; text at the upper right corner
eI.Position := Pos_left       ; text at centre of left border
eI.Position := Pos_center     ; text is centred
eI.Position := Pos_right      ; text at centre of right border
eI.Position := Pos_bottomleft ; text at lower left corner
eI.Position := Pos_bottom     ; text at lower centre
eI.Position := Pos_bottomright ; text at lower right corner
```

The registers **eI.SpaceLeft** and **eI.SpaceRight** define the distance from the left respectively from the right border.

The registers **eI.HorizontalAdjust** and **eI.VerticalAdjust** define an offset from the calculated position. With these registers it is possible to have the text rendered e.g. one pixel more right and down, when a button is pressed. When **eI.VerticalAdjust** has a large negative value it is possible to write outside a label what is sometimes useful to write titles over a label object.

Border styles:

The label has a rendered border. In the register **eI.BorderStyle** the border style can be selected.

```
eI.BorderStyle := border_color_button_soft_raised_big
eI.BorderStyle := border_color_button_soft_sunk_big
```



```
Load.Geometry_XYWH( 0,320,640,100)
eI.FillColor := silver
Draw.RectangleFilled()

Load.Geometry_XYWH(20,340,140,50)
Label.Color ( darkturquoise )
eI.BorderStyle := border_color_button_soft_raised_big
Label.Box ( )

Load.Geometry_XYWH(180,340,140,50)
Label.Color ( darkturquoise )
eI.BorderStyle := border_color_button_soft_sunk_big
Label.Box ( )
```

Example of two buttons with the two BorderStyles that give the effect of a pressed button.

**Label.Color(VarInt:Color)**

0\$50

The method **Label.Color(VarInt:Color)** preloads the colour registers of the eVM.

Example:

```
Label.Color(light_green)    ; preload registers with colour as parameter
```

The registers **eI.FillColor**, **eI.LineColor** and **eI.BackColor** are loaded with the colour given as parameter.

For **eI.TextColor** the method **Colors\_15.AutoColor(VarInt,VarInt)** is used that gives a black or a white text colour for an optimal contrast, depending of the colour (see p.71).

Special case:

```
Label.Color(transparent)   ; load transparency
```

The registers **eI.FillColor**, **eI.LineColor** and **eI.BackColor** are loaded with transparency. In that case the colour in the **eI.DisplayColor** is used as InputValue for the method **Colors\_15.AutoColor(VarInt,VarInt)**. Thus only the text is written on the background. Attention: if the label is written a second time, the first text is not cleared, as the back colour is transparent.

**Label.GreyColor()**

0\$50

The method **Label.GreyColor()** sets the 4 colours in the registers **eI.FillColor**, **eI.LineColor** and **eI.BackColor** to their corresponding grey values. The method is used to indicate inactive buttons on an application.



**Label.Text(VarStr)**

0\$50

The method **Label.Text(VarStr)** draws a label with a string. The label is rendered according to the setting of the eVM-registers.

Example:

```
Load.Geometry_XYWH( 0,320,640,100)
eI.FillColor := silver
Draw.RectangleFilled()

Load.Geometry_XYWH(20,340,140,50)
Label.Color ( darkturquoise )
eI.BorderStyle := border_color_button_soft_raised_big
eI.Position := Pos_left
eI.HorizontalAdjust := 0
eI.VerticalAdjust := 0
eI.SpaceLeft := 8
eI.FontNumber := Font_Arial_14n
Label.Text ('Hello world')
```



Result of the code above

Tip: With the method **Fill.LabelParameter()** the eVM-register can be efficiently preloaded from a structure called a **style**. If some attributes have to be changed, the registers can be subsequently overwritten with the desired attribute value (see class Fill, p. 84).

Example:

```
Fill.LabelParameter (ButtonUp_Style)
eI.TextColor := red
Label.Text ('Hello world')
```



Result of the code above



**Label.Box( )**

The method **Label.Box( )** renders a label without text but with a border.

Example:

See above

**Label.PrintString(VarStr:Text)**

0\$50

The method **Label.PrintString(VarStr:Text)** renders a string in a label without bordering it. The method is used to draw labels with multiple lines.

**Label.StringWidth(VarInt:Width,VarStr:Text)**

0\$50

The method **Label.StringWidth(VarStr:Text)** computes the width of the text in the specified string and returns the effective width in pixels. The value is dependent of the selected font. Since the fonts are proportional the glyph widths are not the same for different characters the string length doesn't say much about the effective width of a text.

Example:

```
Label.StringWidth(eI.R00,'Colorvectors')
Debug.Print_IntegerHex('width =',eI.R00)
```



Colorvectors

Arial 14  
102 pixel

Application tip:

The method can be used to determine whether a text fits in a label with a fixed width and if the computed width is greater than the label width, a smaller font is chosen until the text fits in the label.

**Fonts on the system**

There are predefined fonts in the firmware of a FOX embedded computer. They can be used when assigning a font in the label structure (**Fill.LabelParameter(labelRelative24)**). Just write the font constant into the register **eI.FontNumber** and the font is set. When next time a label method is used it checks whether the font is already in use; if it is different, the font is installed and the text is rendered with the new font.

The following figure shows the fonts installed on the eigerPanel 57 and the eigerPanel 70. Please note, that on different computers other fonts may be available, depending on the size of the ROM.

Text in Font\_Arial\_7 94 pixels 12  
 Text in Font\_Arial\_8 102 pixels 14  
 Text in Font\_Arial\_10n 124 pixels 16  
 Text in Font\_Arial\_12n 155 pixels 18  
 Text in Font\_Arial\_14n 193 pixels 22  
 Text in Font\_Arial\_16n 207 pixels 24  
 Text in Font\_Arial\_20n 262 pixels 31  
 Text in Font\_Arial\_24n 311 pixels 36  
 Text in Font\_System\_9n 117 pixels 14  
 Text in Font\_System\_9n 134 pixels 14  
**Text in Font\_System\_9bd 279 pixels 28**  
 Text in Font\_Courier\_9n 161 pixels 15  
 0 123456789 100 16  
 0 123456789 140 23  
 0 123456789 240 35  
 0 123456789 400 64

The following list shows the assignments of the register eI.FontNumber to obtain the font probe above. One setting can be used at a time.

```

eI.FontNumber := Font_Arial_7
eI.FontNumber := Font_Arial_8
eI.FontNumber := Font_Arial_10n
eI.FontNumber := Font_Arial_12n
eI.FontNumber := Font_Arial_14n
eI.FontNumber := Font_Arial_16n
eI.FontNumber := Font_Arial_20n
eI.FontNumber := Font_Arial_24n
eI.FontNumber := Font_System_9n
eI.FontNumber := Font_System_9bd
eI.FontNumber := Font_System_18bd
eI.FontNumber := Font_Courier_9n
eI.FontNumber := Font_DigitalNumbers_16
eI.FontNumber := Font_DigitalNumbers_24
eI.FontNumber := Font_DigitalNumbers_32
eI.FontNumber := Font_DigitalNumbers_64
  
```

## Class Draw

The class Draw contains methods to draw geometric objects on the screen. There are methods to draw pixels, lines, rectangles, circles and ellipsoids. The rectangles, circles and ellipsoids can be filled.

### Draw.Pixel()

The method **Draw.Pixel()** sets a pixel. The parameters are given as eVM registers.

<b>eI.Pos_X1</b>	: X-position
<b>eI.Pos_Y1</b>	: Y-position
<b>eI.LineColor</b>	: colour of the pixel

#### Example:

Draw a red pixel at XY-position 120/60: .

```
eI.Pos_X1 := 120
eI.Pos_Y1 := 60
eI.LineColor := red
Draw.Pixel() ; draw pixel at position, colour specified above
```

### Draw.Line()

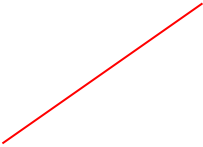
The method **Draw.Line()** sets a line. The parameters are given as eVM registers.

<b>eI.Pos_X1</b>	: X-startposition
<b>eI.Pos_Y1</b>	: Y-startposition
<b>eI.Pos_X2</b>	: X-endposition
<b>eI.Pos_Y2</b>	: Y-endposition
<b>eI.LineColor</b>	: colour of the line

#### Example:

Draw a red line from XY-position 100/50 to XY-position 200/120:

```
eI.Pos_X1 := 100
eI.Pos_Y1 := 50
eI.Pos_X2 := 200
eI.Pos_Y2 := 120
eI.LineColor := red
Draw.Line() ; draw line at startposition, endposition, colour specified above
```



**Draw.Rectangle()**

The method **Draw.Rectangle()** draws a rectangle. The parameters are given as eVM registers.

```
eI.Pos_Xl      : X-Position
eI.Pos_Yl      : Y-Position
eI.Width       : width
eI.Height      : height
eI.LineColor    : colour of the rectangle
```

Example:

Draw a blue rectangle at position X = 10 / Y = 70 with a width of 100 and a height of 40 pixels:

```
Load.Geometry_XYWH(10,70,100,40)
eI.LineColor := blue
Draw.Rectangle() ;draw rectangle with position, geometry, colour specified above
```

**Draw.RectangleFilled()**

The method **Draw.RectangleFilled()** draws a filled rectangle. The parameters are given as eVM registers.

```
eI.Pos_Xl      : X-Position
eI.Pos_Yl      : Y-Position
eI.Width       : width
eI.Height      : height
eI.FillColor    : fill colour of the rectangle
```

Example:

Draw a filled blue rectangle at position X = 10 / Y = 70 with a width of 100 and a height of 40 pixels:

```
Load.Geometry_XYWH(10,70,100,40)
eI.FillColor := blue
Draw.RectangleFilled() ;draw rectangle with position, geometry, colour specified above
```

**Draw.Circle()**

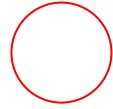
The method **Draw.Circle()** draws a circle. The parameters are given as eVM registers.

```
eI.Pos_Xl      : X-position
eI.Pos_Yl      : Y-position
eI.Radius      : radius
eI.LineColor    : line colour of the circle
```

Example:

Draw a red circle with center at XY-position 120/60 and a radius of 25 pixels:

```
eI.Pos_X1 := 120
eI.Pos_Y1 := 60
eI.Radius := 25
eI.LineColor := red
Draw.Circle() ; draws a circle at position, radius, colour specified above
```

**Draw.Ellipse()**

The method **Draw.Ellipse()** draws a ellipsoid. The parameters are given as eVM registers.

<b>eI.Pos_X1</b>	: X-position
<b>eI.Pos_Y1</b>	: Y-position
<b>eI.HalfAxis_a</b>	: X-halfaxis
<b>eI.HalfAxis_b</b>	: Y-halfaxis
<b>eI.LineColor</b>	: line colour of the ellipsoid

Example:

Draw a red ellipsoid with center at XY-position 120/60, expanse in x = 34 / y=12:

```
eI.Pos_X1 := 120
eI.Pos_Y1 := 60
eI.HalfAxis_a := 34
eI.HalfAxis_b := 12
eI.LineColor := red
Draw.Ellipse() ; draw ellipsoid at position, size, colour specified above
```



## Class Math

The class Math has methods for converting numeric data types from and to each other. Mathematical operations are performed by a stack machine built into the eVM. Since data types are strictly distinguished in EigerScript, type conversions must be explicitly programmed by the programmer.

### Integermath

The integer math works with signed longwords on the stack. This has the advantage that the user can use integers without bothering about overflows.

Special methods for integers:

**Math.MIN\_Integer(VarInt:MIN,VarInt,VarInt)**

The method **Math.MIN\_Integer(VarInt,VarInt,VarInt)** returns the smaller of the two input numbers.

```
Math.MIN_Integer(Result.I,Input1.I,16)
; return minimum from Input1.I and 16
```

**Math.MAX\_Integer(VarInt:MAX,VarInt,VarInt)**

The method **Math.MAX\_Integer(VarInt,VarInt,VarInt)** returns the larger of the two input numbers.

```
Math.MAX_Integer(Result.I,Input1.I,16)
; return maximum from Input1 and the number 16
```

### Conversions from Long to Integer

These methods are used for data manipulation. 16-Bit data words can be packed and unpacked to a long data type.

**Math.LWRD\_from\_Long(VarInt:Target,VarLong:Source)**

The method **Math.LWRD\_from\_Long(VarInt,VarLong)** loads the integer result from the bits 0..15 of the longword.

**Math.LWRD\_to\_Long(VarLong:Target,VarInt:Source)**

The method **Math.LWRD\_to\_Long(VarLong,VarInt)** loads the lower bits 0..15 of the longword with the integer. The high word remains unchanged.

**Math.HWRD\_from\_Long(VarInt:Target,VarLong:Source)**

The method **Math.HWRD\_from\_Long(VarInt,VarLong)** loads the integer result from the bits 16..31 of the longword.

**Math.HWRD\_to\_Long(VarLong:Target,VarInt:Source)**

The method **Math.HWRD\_to\_Long(VarLong,VarInt)** loads the higher bits 16..31 of the longword with the integer. The low word remains unchanged.

**Type conversions**

Data type conversions are used to convert data types from one representation to another representation.

**Math.CVT\_UInteger\_from\_Long(VarInt:Target,VarLong:Source)**

The method **Math.CVT\_UInteger\_from\_Long(VarInt:Integer,VarLong:Long)** converts a LONG variable to an unsigned-integer-variable. Since the range of Long-variables is much greater than unsigned integer, an overflow error can occur. The method checks the range and if it is out of range, the **ERR\_Math\_OutOfRange** is set.

Typical code sequence:

```
eI.Status := success
Math.CVT_UInteger_from_Long(TestInteger.I,TestLong.L)
IF eI.Status == error THEN
    ; out of range
ENDIF
```

**Math.CVT\_Integer\_from\_Long(VarInt:Target,VarLong:Source)**

The method **Math.CVT\_Integer\_from\_Long(VarInt:Integer,VarLong:Long)** converts a LONG variable to a signed-integer-variable. Since the range of Long-variables is much greater than signed integer, an overflow error can occur. The method checks the range and if it is out of range, the **ERR\_Math\_OutOfRange** is set.

typical code sequence:

```
IF eI.Status == success THEN
Math.CVT_Integer_from_Long(TestInteger.I,TestLong.L)
ENDIF
```

```
Math.CVT_Single_from_Integer(VarSingle:Target,VarInt:Source)
```

The method `Math.CVT_Single_from_Integer(VarSingle,VarInt)` converts an Integer-variable to a Single-variable.

```
Math.CVT_Single_from_UInteger(VarSingle:Target,VarInt:Source)
```

The method `Math.CVT_Single_from_UInteger(VarSingle,VarInt)` converts an unsigned Integer-variable to a Single-variable.

```
Math.CVT_Single_from_Long(VarSingle:Target,VarLong:Source)
```

The method `Math.CVT_Single_from_Long(VarSingle,VarLong)` converts a Long-variable to a Single-variable.

```
Math.CVT_Single_from_LongDeci(VarSingle:Target,VarLong:Source,VarInt:Decimals)
```

The method `Math.CVT_Single_from_LongDeci(VarSingle,VarLong,VarInt)` converts a LongDeci-variable to a Single-variable. The decimal position has to be given in the parameter decimals.

```
Math.CVT_Long_from_Single(VarLong,VarSingle)
```

The method `Math.CVT_Long_from_Single(VarLong,VarSingle)` converts a Single-variable to a Long-variable. Since „Long“ is a data type for integer numbers no decimals are converted. The method doesn't round.

Example:

```
Math.CVT_Long_from_Single(Distance.L,Distance.S)
```

e.g. for `Distance.S = 1234.56` the output value for `Distance.L` is `1234`; trailing decimals are neglected.



**Math.CVT\_Long\_from\_Single\_Round(VarLong,VarSingle)**

The method **Math.CVT\_Long\_from\_Single\_Round(VarLong,VarSingle)** converts a Single-variable to a Long-variable. Since „Long“ is a data type for integer numbers no decimals are converted, but the long is rounded up or down.

Example:

```
Math.CVT_Long_from_Single_Round(Distance.L,Distance.S)
```

e.g. for **Distance.S** = 1234.56, **Distance.L** gets the rounded up value 1235.

**Math.CVT\_LongDeci\_from\_Single(VarLong,VarSingle,VarInt:Decimals)**

The method **Math.CVT\_LongDeci\_from\_Single(VarLong,VarSingle,Number of Decimals)** converts a Single-variable to a Long-variable in a way that also decimals are looked after. Relevant decimals are given as parameter.

Example:

```
Math.CVT_LongDeci_from_Single(Distance.L,Distance.S,3)
```

e.g. for **Distance.S** = 1234.56, **Distance.L** the value of 1234560. (factor 1000 greater or 3 decimals greater)

**Calculations****Math.MOD\_Integer(VarInt:Y,VarInt:A,VarInt:B)**

The method **Math.MOD\_Integer(VarInt:Y,VarInt:A,VarInt:B)** returns the remainder on division of A by B.

Example:

```
Math.MOD_Integer(Remainder.I,10,6)
```

---

## Class Binary

The class contains methods for logical bit operations.

### Bit functions with 16-bit or 32-bit operands

There exist methods to manipulate Single bits in Integer and Long operands. A bit can be set, reset or toggled.

#### **Binary.BSET\_Integer(VarInt,VarInt:BitNo)**

0\$35

The method **Binary.BSET\_Integer(VarInt,VarInt:BitNumber)** sets the bit given as parameter. The correct range for the bit number is 0..15.

Example:

```
Binary.BSET_Integer(MyInteger.I,3)           ; set bit 3
If MyInteger.I is 7 (00000000000000111), the method returns 15 (0000000000001111)
to MyInteger.I
```

#### **Binary.BGET\_Integer(VarInt,VarInt:BitNo,VarInt:Result)**

0\$35

The method **Binary.BGET\_Integer(VarInt,VarInt:BitNo,VarInt:Result)** returns the bit at a given position. The correct range for the bit number is 0..15.

Example:

```
Binary.BGET_Integer(MyInteger.I,2,Result.I)   ; get bit 2
If MyInteger.I is 7 (00000000000000111), the method returns 1 to Result.I
```

#### **Binary.BCLR\_Integer(VarInt,VarInt:Bit)**

0\$35

The method **Binary.BCLR\_Integer(VarInt,VarInt:BitNumber)** clears the bit given as parameter. The correct range for the bit number is 0..15.

Example:

```
Binary.BCLR_Integer(MyInteger.I,3)           ; clear bit 3
If MyInteger.I is 15 (0000000000001111), the method returns 7 (0000000000000111)
to MyInteger.I
```

#### **Binary.BNOT\_Integer(VarInt,VarInt:Bit)**

0\$35

The method **Binary.BNOT\_Integer(VarInt,VarInt:BitNumber)** toggles the bit given as parameter. The correct range for the bit number is 0..15.

Example:

```
Binary.BNOT_Integer(MyInteger.I,3)           ; toggle bit 3
If MyInteger.I is 15, the method toggles between 7 (0000000000000111) and 15
(00000000000001111).
```

<b>Binary.BSET_Long(VarLong,VarInt:Bit)</b>
---

<b>0\$35</b>
--------------

The method **Binary.BSET\_Long(VarLong,VarInt:BitNumber)** sets the bit given as parameter. The correct range for the bit number is 0..31.

Example (cf **Binary.BSET\_Integer(..)**):

```
Binary.BSET_Long(MyLong.L,20)                ; set bit 20
```

<b>Binary.BGET_Long(VarLong,VarInt:BitNo,VarInt:Result)</b>
---

<b>0\$35</b>
--------------

The method **Binary.BGET\_Long(VarInt,VarInt:BitNo,VarInt:Result)** returns the bit at a given position. The correct range for the bit number is 0..31.

Example:

```
Binary.BGET_Long(MyLong.I,20,Result.I)       ; get bit 20
```

<b>Binary.BCLR_Long (VarLong,VarInt:Bit)</b>
--

<b>0\$35</b>
--------------

The method **Binary.BCLR\_Long(VarLong,VarInt:BitNumber)** clears the bit given as parameter. The correct range for the bit number is 0..31.

Example:

```
Binary.BCLR_Long(MyLong.L,20)                ; clear bit 20
```

<b>Binary.BNOT_Long (VarLong,VarInt:Bit)</b>
--

<b>0\$35</b>
--------------

The method **Binary.BNOT\_Long(VarLong,VarInt:BitNumber)** toggles the bit given as parameter. The correct range for the bit number is 0..31.

Example:

```
Binary.BNOT_Long(MyLong.L,20)                ; toggle bit 20
```

### Logic functions with 16-bit or 32-bit operands

Integer variables can be logically manipulated. The logic function work bitwise on the operands.

The following table shows how the bits are manipulated. The truth table shows the operation in the same bit position.

Input B	Input A	AND	OR	XOR	NOT
0	0	0	0	0	1
0	1	0	1	1	0
1	0	0	1	1	1
1	1	1	1	0	0

<b>Binary.AND_Integer (VarInt:Q,VarInt:A,VarInt:B)</b>	<b>0\$35</b>
--	--------------

The method **Binary.AND\_Integer (VarInt,VarInt,VarInt)** logically AND's the input operands and stores the result to the output variable. With this method masks can be laid over bit fields.

```
Binary.AND_Integer(eI.BackColor,eI.BackColor,0111110000011111B)
; the green bits are cleared
```

<b>Binary.OR_Integer (VarInt:Q,VarInt:A,VarInt:B)</b>	<b>0\$35</b>
---	--------------

The method **Binary.AND\_Integer (VarInt,VarInt,VarInt)** logically OR's the input operands and stores the result to the output variable. With this method masks can be laid over bit fields.

```
Binary.OR_Integer(eI.BackColor,eI.BackColor,0000001111100000B)
; the green bits are set
```

<b>Binary.XOR_Integer (VarInt:Q,VarInt:A,VarInt:B)</b>	<b>0\$35</b>
--	--------------

The method (exclusive or) **Binary.XOR\_Integer (VarInt,VarInt,VarInt)** logically XOR's the input operands and stores the result to the output variable. With this method masks of bits can selectively be inverted.

```
Binary.XOR_Integer(eI.BackColor,eI.BackColor,0000001111100000B)
; the green bits are inverted
```

<b>Binary.NOT_Integer (VarInt:Q,VarInt:A)</b>	<b>0\$35</b>
---	--------------

The method **Binary.NOT\_Integer (VarInt,VarInt)** logically inverts the input operand and stores the result to the output variable. With this method the bits can be inverted.

```
Binary.NOT_Integer(MyInteger.I,MyInteger.I) ; MyInteger.I is inverted
```

**Binary.SWAP\_BYTE\_Integer (VarInt:Q,VarInt:A)****0\$35**

The method **Binary.SWAP\_BYTE\_Integer (VarInt,VarInt)** changes the low and high byte position of the input parameter to the high and low byte position of the output variable.

**Binary.SWAP\_BYTE\_Integer (MyInteger.I,MyInteger.I)**

Before SWAP	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
After SWAP	7 6 5 4 3 2 1 0 15 14 13 12 11 10 9 8

**Binary.AND\_Long (VarLong:Q,VarLong:A,VarLong:B)****0\$35**

The method **Binary.AND\_Long (VarInt,VarInt,VarInt)** logically AND's the input operands and put the result to the output variable. With this method masks can be laid over bit fields.

**Binary.AND\_Long (eI.BackgroundColor,eI.BackgroundColor,0111110000011111B)**  
 ; the green bits are cleared

**Binary.OR\_Long (VarLong:Q,VarLong:A,VarLong:B)****0\$35**

The method **Binary.AND\_Long (VarInt,VarInt,VarInt)** logically OR's the input operands and put the result to the output variable. With this method masks can be laid over bit fields.

**Binary.OR\_Long (eI.BackgroundColor,eI.BackgroundColor,0000001111100000B)**  
 ; the green bits are set

**Binary.XOR\_Long (VarLong:Q,VarLong:A,VarLong:B)****0\$35**

The method (exclusive or) **Binary.XOR\_Long (VarInt,VarInt,VarInt)** logically XOR's the input operands and put the result to the output variable. With this method masks of bits can be inverted.

**Binary.XOR\_Long (eI.BackgroundColor,eI.BackgroundColor,0000001111100000B)**  
 ; the green bits are inverted

**Binary.NOT\_Long (VarLong:Q,VarLong:A)****0\$35**

The method **Binary.NOT\_Long (VarLong,VarLong)** logically inverts the input operand and stores the result to the output variable. With this method the bits can be inverted.

```
Binary.NOT_Long(MyLong.L,MyLong.L)
```

```
; MyLong.L is inverted
```

```
Binary.SWAP_WORD_Long(VarLong:Q,VarLong:A)
```

```
0$35
```

The method `Binary.SWAP_WORD_Long(VarLong,VarLong)` changes the low and high word position of the input parameter to the high and low word position of the output variable.

```
Binary.SWAP_WORD_Long(MyLong.L,MyLong.L)
```

Before SWAP

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	---	---	---	---	---	---	---	---	---	---

After SWAP

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
----	----	----	----	----	----	---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

## Class HotSpot

The class HotSpot has methods to create the touch screen interface (GUI) to the user. HotSpots are rectangular fields that can throw events, when touched. In the eigerOS the touch panel is permanently scanned for HotSpots that generate events.

In eigerScript there are methods to install HotSpots and methods to remove them. After changing a view all the HotSpots of the previous view are removed. All HotSpots are managed in the HotSpotTable internally to the eigerOS.

If the eigerOS finds a HotSpot that generates an event, the event is executed, when there is an event handler in the eigerScript program. The possible events for a HotSpot are:

Enter	HotSpot is entered
Leave	HotSpot is leaved
Down	Touchdown in the HotSpot
Up	Release of touch action in the HotSpot

The HotSpot events are basically used for two purposes: the animation of the GUI and to take actions on the GUI.

In the simplest use case, only the touchdown event is handled.

Every HotSpot can store an integer value that is called tag. This tag is returned in the **eI.HS\_Tag** register. This feature simplifies the programming of HotSpots that have nearly the same action. Lets think of a numeric keypad. *The numeric keys* have the same functionality except for the numeric value of every key that differs. Using the **eI.HS\_Tag** register, the same event handler can be used for every numeric key.

The geometry of the HotSpot is also returned in special registers.

The HotSpots are stored in the HotSpot table, when installing them. This table is cleared when the view is leaved, to prevent a new view from reacting on old HotSpots.

The HotSpots have two states: they can be enabled or disabled. When disabled, no event is generated, when touched. HotSpots are installed with the state "enabled". If a certain HotSpot should be disabled the method **HotSpot.Enable\_By\_ID(eI.HS\_ID)** immediately after installing it disables the HotSpot.

## HotSpot Groups

Every HotSpot is part of a group. The group is defined with the value of the register **eI.HotSpotGroup** at installation. The number of HotSpots in the same group is not limited. Some methods work with groups as e.g. **HotSpot.DeInstallGroup()**. This permits to reference groups of HotSpots what is especially useful when using controls or popup's.

The method **HotSpot.GetNextGroupNumber()** requests the next free group number and places it into the register **eI.HotSpotGroup**. At removing of the popup the HotSpot group can be removed by addressing the group.

## How the HotSpots work

There is the above mentioned HotSpotTable in the eVM. The tables store information about every HotSpot that is installed on the view. In a new view the HotSpotTable is empty. HotSpots are installed with the method **HotSpot.Install(Enter,Leave,Down,Up)**. The parameters

are names of event handlers defined as subroutines. Before installing the HotSpot the geometry of the HotSpot is placed into the geometry registers **eI.Pos\_X1**, **eI.Pos\_Y1**, **eI.Width** and **eI.Height**. With the method

**Load.Geometry\_XYWH(0,0,as\_DisplayWidth,as\_DisplayHeight)** ; Example

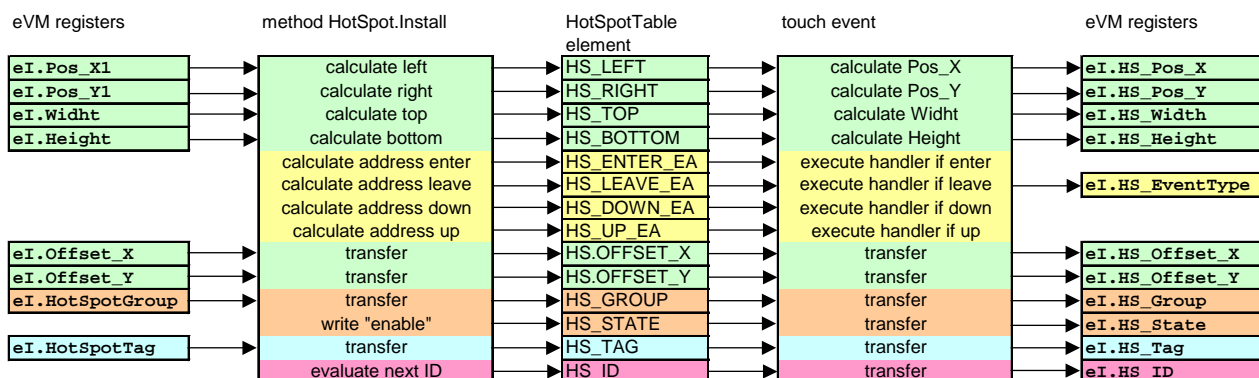
The geometry can be defined.

One element of the HotSpot table has the following form:

HS_LEFT	UINT16	pixels	
HS_RIGHT	UINT16	pixels	
HS_TOP	UINT16	pixels	
HS_BOTTOM	UINT16	pixels	
HS_ENTER_EA	INT24	address of event handler ENTER	
HS_LEAVE_EA	INT24	address of event handler LEAVE	
HS_DOWN_EA	INT24	address of event handler DOWN	
HS_UP_EA	INT24	address of event handler UP	
HS.OFFSET_X	UINT16	pixels	
HS.OFFSET_Y	UINT16	pixels	
HS_GROUP	UINT8	group number	
HS_STATE	UINT8	boolean	
HS_TAG	INT16	tag	
HS_ID	UINT16	hotspot ID	

The geometry at the time of installation is converted in absolute screen coordinates and stored in the structure. The addresses for the event handlers are stored in the structure. If there is no event handler, NIL is stored. At the time of installation offset is also stored to maintain the information of the relative position of the HotSpot. The group number is copied from the register **eI.HotSpotGroup**. The tag can be given by the user and is copied from the register **eI.HS\_Tag**. The HotSpot ID is a unique ID given by the eVM at the time of installation.

Functional description of the eiger HotSpots from left to right:



**Figure 1:** HotSpot table with its eVM registers and methods.

1. The registers have to be set before the HotSpot is installed.
2. The method **HotSpot.Install** is executed and the HotSpot is stored as element in the HotSpot table. Further HotSpots can be installed.



3. With the method **HotSpot.TableEnable** the HotSpot table is enabled for the eigerOS to scan the touch panel for relevant events.
4. If a touch event for one given HotSpot is detected the eigerOS fills up the **eI.HS\_...** registers and executes the appropriate touch event handler.

The register **eI.HS\_EventType** holds the last HotSpot event that occurred. So it is also possible to write an event handler and decide in the handler what to do.

Possible constants in the Register **eI.HS\_EventType** :

```

CONST HS_EVENT_NONE      = 0           ; no event
CONST HS_EVENT_ENTER     = 1           ; Enter event
CONST HS_EVENT_LEAVE     = 2           ; Leave event
CONST HS_EVENT_DOWN      = 3           ; Down event
CONST HS_EVENT_UP        = 4           ; Up event

```

### HotSpot.Install(Enter,Leave,Down,Up)

0\$50

With the method **HotSpot.Install(Enter,Leave,Down,Up)** a HotSpot is installed. As mentioned above the registers **eI.Pos\_X1**, **eI.Pos\_Y1**, **eI.Width** and **eI.Height** have to be preloaded with the size of the rectangle that defines a specific HotSpot.

#### Example:

```

; Event handler for animated button

STRING [32] HS1_Text.$ = 'Label Test'

SUB HS1_LE
  Fill.LabelParameter ( Test_Style_UP )
  Load.Pos_X1Y1 ( 20, 70 )
  Label.Text ( HS1_Text.$ )
ENDSUB

SUB HS1_DN
  Fill.LabelParameter ( Test_Style_DN )
  Load.Pos_X1Y1 ( 20, 70 )
  Label.Text ( HS1_Text.$ )
ENDSUB

SUB HS3_UP
  CallSubroutine ( HS1_LE )
  ; enter the code for the action of the button here...
ENDSUB
..

SUB Test_Styles
Test_Style_UP:
  INLINEWORDS (no_change) ; corresponds to eI.Pos_X1
  INLINEWORDS (no_change) ; corresponds to eI.Pos_Y1
  INLINEWORDS (120) ; corresponds to eI.Width
  INLINEWORDS (40) ; corresponds to eI.Height
  INLINEWORDS (8) ; corresponds to eI.SpaceLeft
  INLINEWORDS (8) ; corresponds to eI.SpaceRight

```

```

    INLINENWORDS (0) ; corresponds to eI.HorizontalAdjust
    INLINENWORDS (0) ; corresponds to eI.VericalAdjust
    INLINENWORDS (cornflowerblue) ; corresponds to eI.FillColor
    INLINENWORDS (as_FillColor) ; corresponds to eI.BackColor
    INLINENWORDS (as_FillColor) ; corresponds to eI.LineColor
    INLINENWORDS (autocolor) ; corresponds to eI.TextColor
    INLINENWORDS (Pos_left) ; corresponds to eI.Position
    INLINENWORDS (Orientation_0deg) ; corresponds to eI.Orientation
    INLINENWORDS (normal) ; corresponds to eI.Appearance
    INLINENWORDS (as_Skin_BtnBorderUP) ; corresponds to eI.BorderStyle
    INLINENWORDS (Font_Arial_14n) ; corresponds to eI.FontNumber
    INLINENWORDS (as_Skin_FormBodyColor) ; corresponds to eI.BackgroundColor

Test_Style_DN:
    INLINENWORDS (no_change) ; corresponds to eI.Pos_X1
    INLINENWORDS (no_change) ; corresponds to eI.Pos_Y1
    INLINENWORDS (120) ; corresponds to eI.Width
    INLINENWORDS (40) ; corresponds to eI.Height
    INLINENWORDS (8) ; corresponds to eI.SpaceLeft
    INLINENWORDS (8) ; corresponds to eI.SpaceRight
    INLINENWORDS (1) ; corresponds to eI.HorizontalAdjust
    INLINENWORDS (1) ; corresponds to eI.VericalAdjust
    INLINENWORDS (seagreen) ; corresponds to eI.FillColor
    INLINENWORDS (as_FillColor) ; corresponds to eI.BackColor
    INLINENWORDS (as_FillColor) ; corresponds to eI.LineColor
    INLINENWORDS (autocolor) ; corresponds to eI.TextColor
    INLINENWORDS (Pos_left) ; corresponds to eI.Position
    INLINENWORDS (Orientation_0deg) ; corresponds to eI.Orientation
    INLINENWORDS (normal) ; corresponds to eI.Appearance
    INLINENWORDS (as_Skin_BtnBorderDN) ; corresponds to eI.BorderStyle
    INLINENWORDS (Font_Arial_14n) ; corresponds to eI.FontNumber
    INLINENWORDS (as_Skin_FormBodyColor) ; corresponds to eI.BackgroundColor

ENDSUB

..
    CallSubroutine ( HS1_LE )
    HotSpot.Install ( NIL, HS1_LE, HS1_DN, HS1_UP )

    HotSpot.TableEnable()

```

After calling the subroutine HS1\_LE the registers are loaded with the position and the size of the button. Then the HotSpot can be installed.

Tip1: if a button with otherwise same attributes should be installed, the button can be coloured with the method `Label.Color(hotpink)`.

Tip2: if a button without text has to be installed the method `Label.Box()` renders a such a button.

Tip3: if an invisible HotSpot is desired the coordinates can be set with the method `Load.Geometry_XYWH(0,0,as_DisplayWidth,as_DisplayHeight)`. A HotSpot over the whole display is installed.

Tip4: a HotSpot can be placed on an image. After loading the image with the method `File.ReadEGI('C:/MYPR/PICT/MYPICT01.EGI')` the width and the height of the image are returned in the registers `eI.Width` and `eI.Height`.

Tip5: if an event needs no event handler, `NIL` (Nothing In List) can be set as parameter. `HotSpot.Install(NIL,NIL,HS1_DN,NIL)` if only the DownEvent needs a handler.

Tip6: several HotSpots can use the same event handler. The action can be selected in the event handler using the value of the register **eI.HS\_Tag**. The value has to be set at installation of the HotSpot.:

Example for assigning a HotSpot-Tag to a HotSpot:

```
Load.Geometry_XYWH(Button_X,Button_Y,ButtonWidth,Button_H)
eI.HotSpotTag := 1 ; you can also assign a CHAR like "A" or 0x41
HotSpot.Install(NIL,Button01_Leave,Button01_Down,Button01_Up)
```

When this HotSpot is touched, it assigns the value of its **eI.HotSpotTag** to the Register **eI.HS\_Tag** (cf. Figure 1, p.46).

### HotSpot.GetNextGroupNumber ( )

0\$50

The method **HotSpot.GetNextGroupNumber ( )** gets the number of the next HotSpotgroup in the range of 32 to 223. The return value is in the register **eI.HotSpotGroup** (see below, "function of the operating system"). All HotSpots that have been installed since calling the method are installed with this group number. The returned group number is counted internally and can not be manipulated. If the group number has reached 223 the next value is 32 again. The group numbers are useful to define groups of HotSpots that are used in a popup dialog. If the popup is removed, its HotSpots have to be removed from the HotSpot table simultaneously. With this feature it is possible to write code for controls in code modules that can be included to the program code.

Function of the operating system:

```
eOS_HS_GroupNumberInternal := eOS_HS_GroupNumberInternal + 1
IF eOS_HS_GroupNumberInternal > 223 THEN
    eOS_HS_GroupNumberInternal := 32
ENDIF
eI.HotSpotGroup := eOS_HS_GroupNumberInternal
```

Example:

```
HotSpot.GetNextGroupNumber()
Debug.Print_IntegerHex('\r\neI.HotSpotGroup = ',eI.HotSpotGroup)
HotSpot.GetNextGroupNumber ( )
Debug.Print_IntegerHex('\r\neI.HotSpotGroup = ',eI.HotSpotGroup)
```

### HotSpot.GetCurrGroupNumber ( )

0\$50

The method **HotSpot.GetCurrGroupNumber ( )** requests the current group number without counting up the internal group number counter. The group number is returned in the register **eI.HotSpotGroup** (see below, "function of the operating system").

Function of the operating system:

```
eI.HotSpotGroup := eOS_HS_GroupNumberInternal
```

Example:

```
HotSpot.GetCurrGroupNumber()
Debug.Print_IntegerHex('\r\n eI.HotSpotGroup = ', eI.HotSpotGroup)
```

(The expression \r\n stands for *Carriage Return* and *New Line*)

**HotSpot.DisableGroup(VarInt:Group)**

0\$80

The method **HotSpot.DisableGroup(Group)** disables a group of HotSpots. The HotSpots stay stored in the HotSpotTable, but don't generate events when touched. The method is used to temporary disable HotSpots. The group number is passed as parameter.

Example:

```
HotSpot.DisableGroup(MyGroup.I)
```

**HotSpot.EnableGroup (VarInt:Group)**

0\$80

The method **HotSpot.EnableGroup(Group)** enables a group of HotSpots. The group number is passed as parameter.

Example:

```
HotSpot.EnableGroup(MyGroup.I)
```

**HotSpot.Disable\_By\_ID(VarInt:HotSpot\_ID)**

0\$80

The method **HotSpot.Disable\_By\_ID(HotSpot\_ID)** disables a HotSpot that is referenced by its unique ID. The HotSpot stays stored in the HotSpotTable, but doesn't generate events. The method is used to temporary disable HotSpots. The HotSpot\_ID is passed as parameter to the method.

Example:

```
HotSpot.Enable_By_ID(MyID.I)
```

**HotSpot.Enable\_By\_ID(VarInt:HotSpot\_ID)**

0\$80

The method **HotSpot.EnableGroup (HotSpot\_ID)** enables a HotSpot that is referenced by its unique ID. The HotSpot that is stored in the HotSpotTable in disabled state is enabled and can generate events. The HotSpot\_ID is passed as parameter to the method.

Example:

```
HotSpot.Enable_By_ID (MyID.I)
```

**HotSpot.DeInstallGroup()**

0\$50

The method **HotSpot.DeInstallGroup()** deinstalls a group of HotSpots. The method is used to clean up a popup dialog. The group number is passed in register **eI.HotSpotGroup** to the method. Since the end of a popup dialog is activated by a navigation button the register **eI.HS\_Group** returns the group number used to remove the controls HotSpots.

Example:

```
eI.HotSpotGroup := MyGroup.I  
HotSpot.DeInstallGroup()
```

**HotSpot.TableEnable()**

The method **HotSpot.TableEnable()** enables the HotSpotTable for the scanning of touch events that trigger an event handler. The execution of this method is essential after the installation of the HotSpots in a view.

**HotSpot.TableDestroy()**

The method **HotSpot.TableDestroy()** destroys the HotSpotTable and the scanning for events is stopped. After execution of this method, new HotSpots have to be installed with the method **HotSpot.Install(Enter,Leave,Down,Up)** and the scanning of the table has to be restarted with the method **HotSpot.TableEnable()**.

**HotSpot.DisableRegion()**

0\$80

The method **HotSpot.DisableRegion()** disables all HotSpots that are within a rectangular region specified or touch that region. The geometry for the rectangle can be defined with the method **Load.Geometry\_XYWH(X,Y,W,H)**.

Example:

```
Load.Geometry_XYWH(100,100,400,300)  
HotSpot.DisableRegion()
```

**HotSpot.EnableRegion()**

0\$80

The method **HotSpot.EnableRegion()** enables all HotSpots that are within a rectangular region specified or touch that region. The geometry for the rectangle can be defined with the method **Load.Geometry\_XYWH(X,Y,W,H)**.

Example:

```
Load.Geometry_XYWH(100,100,400,300)  
HotSpot.EnableRegion()
```

**HotSpot.GetInfo\_By\_ID(VarInt:ID)**

0\$80

The method **HotSpot.GetInfo\_By\_ID(ID)** starts the transfer of the information that are stored in the HotSpotTable to the **eI.HS\_...** registers.

Example:

```
SUB    HS_Test_Enter
HotSpot.GetInfo_By_ID(eI.HS_ID)
Debug.Print_IntegerHex('r\neI.HS_Pos_X      ',eI.HS_Pos_X )
Debug.Print_IntegerHex('r\neI.HS_Pos_Y      ',eI.HS_Pos_Y )
Debug.Print_IntegerHex('r\neI.HS_Width      ',eI.HS_Width )
Debug.Print_IntegerHex('r\neI.HS_Height     ',eI.HS_Height )
Debug.Print_IntegerHex('r\neI.HS_Offset_X   ',eI.HS_Offset_X )
Debug.Print_IntegerHex('r\neI.HS_Offset_Y   ',eI.HS_Offset_Y )
Debug.Print_IntegerHex('r\neI.HS_Group      ',eI.HS_Group )
Debug.Print_IntegerHex('r\neI.HS_Tag        ',eI.HS_Tag )
Debug.Print_IntegerHex('r\neI.HS_ID         ',eI.HS_ID )
Debug.Print_IntegerHex('r\neI.HS_State      ',eI.HS_State )
Debug.Print_CRLF()
ENDSUB
```

The routine of the example code above prints out all HotSpot information over the COM1 debug interface.

---

## Class HotKey

Some appliances don't use a touch panel but function keys on the side of the display. The class HotKey has methods that can be used to integrate the function keys to the view. HotKeys can generate an event when pressed and when released. For keys that should be scanned, a event handler must be present. The HotKeys can be used in conjunction with a touch panel e.g. as function keys for important functions.

The number of function keys depend on the FOX hardware. There exist models with eight or sixteen keyboard inputs. The hardware manual of the computer used has to be consulted. All of the models have in common that the first key is translated to ASCII 'A' when the key is pressed and to ASCII 'a' when the key is released.

To use the HotKeys in a view, the event handler has to be installed for the view. The method **HotKey.InputFlush()** clears the input buffer. HotKeys are installed with the method **HotKey.InstallLocalKey(Key,EventHandler,Tag)**. Then, the table must be enabled with the method **HotKey.TableEnable()** to receive the events. The HotKeys can be rearranged and reconfigured during the runtime of the view.

The HotKeys can be installed locally for a specific view and globally for all views of a project. For global keys there must be a global event handler in the \*.EPR file.

**HotKey.InstallLocalKey(VarInt:Key,labelRelative24:Event,VarInt:Tag)**

The method **HotKey.InstallLocalKey(Key,EventHandler,Tag)** installs a local HotKey. The method uses an ASCII-Character to identify the HotKey to be installed and the name of the event handler. The tag is an integer value, that is returned in the **eI.HK\_Tag** register,

when the event occurs. The value can be used to convert the HotKey codes to specific key codes or action codes.

Tip: It is possible to use the same event handler for the HotKey event and for a touch event, e.g. If the key should take the same action as a soft button on the display.

## **HotKey.DeInstallKey(VarInt:Key)**

The method **HotKey.DeInstallKey(Key)** deinstalls a single HotKey. The method uses an ASCII character to identify the key that has to be deinstalled. The method works for local and for global HotKeys.

## **HotKey.DeInstallLocalKeys()**

The method **HotKey.DeInstallLocalKeys()** deinstalls **all local** HotKeys. This method is executed automatically when a view change occurs because the local event handlers of previous view are not valid in the new view.

## **HotKey.DisableLocalKeys()**

The method **HotKey.DisableLocalKeys()** disables all local HotKeys in the HotKeyTable while the global HotKeys remain active.

## **HotKey.EnableLocalKeys()**

The method **HotKey.EnableLocalKeys()** enables all local HotKeys of the HotKeyTable.

## **HotKey.InputFlush()**

The method **HotKey.InputFlush()** clears the input buffer the method can be used to ensure that no keys are in the input buffer that would generate events after enabling the scanning of the HotKeys.

## **HotKey.TableEnable()**

The method **HotKey.TableEnable()** enables the scanning for the HotKeyTable. In the enabled status of the HotKeyTable the keys can generate events that are handled by the event handlers.

**HotKey.TableDisable()**

The method **HotKey.TableDisable()** disables the HotKeyTable. No HotKey events are generated afterwards. The HotKeys installed remain in the table and the method **HotKey.TableEnable()** enables the scanning. The method **HotKey.InputFlush()** can be used to disregard user inputs when the scanning was interrupted for prolonged time.

**HotKey.TableInit()**

The method **HotKey.TableInit()** removes all HotKey table entries. The local and the global events are removed from the table and the scanning of the table is stopped.

**Application example:**

Keys that are connected to the FOX computer should be handled. The inputs D,E,F should generate the events HK\_Red (handled by the Subroutine "HK\_Red"), HK\_Green and HK\_Blue. Old keys are cleared and the HotKeys are installed locally. In this example the tag is not used and zero is passed as parameter.

```

-----
HotKey.InputFlush()                ; clear old HotKeys
HotKey.InstallLocalKey("D",HK_Red_Dn,0) ; if pressed go to SUB HK_Red
HotKey.InstallLocalKey("d",HK_Red_Up,0) ; if released go to SUB HK_Red_Up
HotKey.InstallLocalKey("E",HK_Green_Dn,0)
HotKey.InstallLocalKey("F",HK_Blue,0)

HotKey.EnableLocalKeys()
HotKey.TableEnable()

```

**Class Time**

The FOX embedded computer is equipped with a real time clock. Methods of the class Time are used to get the current date and time from the real time clock or to reset the real time clock respectively.

In the context of time and date there are also two useful methods of the class String available (cf. p. 22):

```

Str.Time(VarStrTargetString,VarInt:FormatTime)
Str.Date(VarStrTargetString,VarInt:FormatDate)

```



**Time.Get ( )**

The method **Time.Get ( )** is used to assign the **current time, day of week, month and year** to the corresponding registers (see below). **Time.Get ( )** is a prerequisite for the use of the time registers.

<b>eI.YEAR</b>	:	Calendar year, e.g. 2010, 2011 etc.
<b>eI.MONTH</b>	:	Month of year, 1 .. 12
<b>eI.DOW</b>	:	Day of week, e.g. 1 for Mondays .. 7 for Sundays
<b>eI.DATE</b>	:	Day of month, 1 .. 31
<b>eI.HOURS</b>	:	Hour of day, 0 .. 23
<b>eI.MIN</b>	:	Minute of hour, 0 .. 59
<b>eI.SEC</b>	:	Second of minute, 0 .. 59
<b>eI.MSEC</b>	:	Millisecond of second, 0 .. 999

Example:

```

STRING [9] Weekday.$ = ''           ; String declaration for Weekday
STRING [30] Date.$ = ''           ; String declaration for Date

SUB DayOfWeek
    Time.Get()                     ; gets Time and Date from the Real Time Clock
    IF eI.DOW == 1 THEN
        Weekday.$ := 'Monday'
    ELSIF eI.DOW == 2 THEN
        Weekday.$ := 'Tuesday'
    ELSIF eI.DOW == 3 THEN
        Weekday.$ := 'Wednesday'
    ELSIF eI.DOW == 4 THEN
        Weekday.$ := 'Thursday'
    ELSIF eI.DOW == 5 THEN
        Weekday.$ := 'Friday'
    ELSIF eI.DOW == 6 THEN
        Weekday.$ := 'Saturday'
    ELSIF eI.DOW == 7 THEN
        Weekday.$ := 'Sunday'
    ENDIF

    Date.$ := Weekday.$             ; Thursday
    Str.Concat(Date.$, ', ')       ; Thursday,
    Str.Cvt_Integer(Date.$, eI.YEAR, 4) ; Thursday, 2010
    Str.Concat(Date.$, '-')       ; Thursday, 2010-
    eI.FillChar := "0"           ; Zero ahead of Numbers below 10, e.g. 09 instead of 9
    Str.Cvt_Integer(Date.$, eI.MONTH, 2) ; Thursday, 2010-12
    Str.Concat(Date.$, '-')       ; Thursday, 2010-12-
    Str.Cvt_Integer(Date.$, eI.DATE, 2) ; Thursday, 2010-12-09
ENDSUB

```

**Time.Set ( )**

The method **Time.Set ( )** resets the real time clock according to the parameters you previously assigned to the date and time registers **eI.DATE**, **eI.MONTH** and **eI.YEAR**, **eI.HOURS**, **eI.MIN**, **eI.SEC** and **eI.MSEC**.

Example:

```
SUB Set_Date_and_Time
```

```

eI.DATE      := 10      ; 10th of
eI.MONTH     := 12      ; Dezember
eI.YEAR      := 2010    ; of this year
eI.HOURS     := 14      ; 2 pm
eI.MIN       := 15      ; and a quarter
eI.SEC       := 30      ; plus thirty seconds
eI.MSEC      := 0       ; plus zero milliseconds
Time.Set()
ENDSUB

```

### Time.SetDate()

The method **Time.SetDate()** resets the real time clock according to the parameters you previously assigned to the date registers **eI.DATE**, **eI.MONTH** and **eI.YEAR**.

Example:

```

SUB Set_Date
eI.DATE      := 10      ; 10th of
eI.MONTH     := 12      ; Dezember
eI.YEAR      := 2010    ; of this year
Time.SetDate()
ENDSUB

```

### Time.SetTime()

The method **Time.SetTime()** resets the real time clock according to the parameters you've previously assigned to the date and time registers **eI.HOURS**, **eI.MIN**, **eI.SEC** and **eI.MSEC**.

Example:

```

SUB Set_Date
eI.HOURS     := 14      ; 2 pm
eI.MIN       := 15      ; and a quarter
eI.SEC       := 30      ; plus thirty seconds
eI.MSEC      := 0       ; plus zero milliseconds
Time.SetTime()
ENDSUB

```

### Time.DayOfWeek()

The method **Time.DayOfWeek()** is used to get the day of the week of a certain date (cf. Example). In advance day, month and year must have been assigned to the registers **eI.DATE**, **eI.MONTH** and **eI.YEAR**.

Example (what day of the week was February 19<sup>th</sup> 2010 ?) :

```
eI.DATE      := 19      ; day of Month
eI.MONTH     := 2       ; February
eI.YEAR      := 2010    ; Year
Time.DayOfWeek()

IF eI.DOW == 1 THEN
Weekday.$ := 'Monday'
ELSIF eI.DOW == 2 THEN
Weekday.$ := 'Tuesday'
ELSIF eI.DOW == 3 THEN
Weekday.$ := 'Wednesday'
ELSIF eI.DOW == 4 THEN
Weekday.$ := 'Thursday'
ELSIF eI.DOW == 5 THEN
Weekday.$ := 'Friday'
ELSIF eI.DOW == 6 THEN
Weekday.$ := 'Saturday'
ELSIF eI.DOW == 7 THEN
Weekday.$ := 'Sunday'
ENDIF
Label.Text(Date.$) ; Result: 2010-02-19 was Friday
```

## Class Timer

In an event driven system, timers are one of the most important generators for events. It is possible to generate events periodically or to time out a time. Timers can have handlers that are local or global to the view. Local timers are killed at view change while global timers remain active. Global timers should not draw on the display.

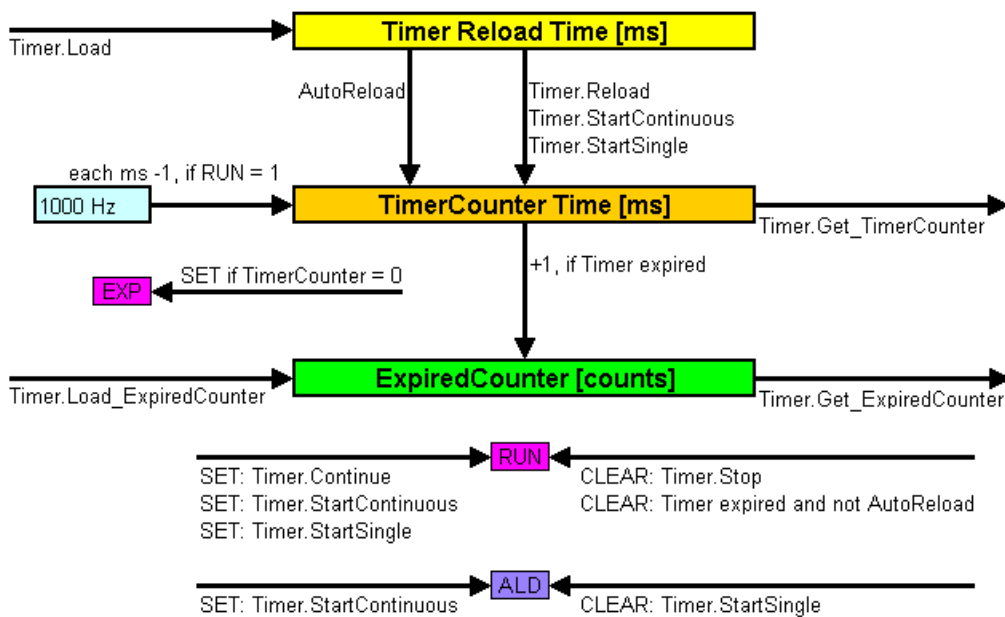
The eVM provides **eight Timers**, Timer 0 to 7.

The eVM-timers have a TicSection and an EventSection that work more or less independently of each other.

The TicSection is responsible for the real-time issues of the timer and sets an internal flag when the timer expires and increments the TimerExpiredCounter.

The TicSection to which the event manager task on the eVM belongs, checks periodically if a timer has expired and generates an event if so.

The timer can be used for periodical updates on the display or to supervise user actions with a time out.

**Structure of timer:**

The diagram above shows the structure of a single timer channel. The timer has a base clock of 1000Hz thus all counts are a multiple of 1ms. With `Timer.Load(Timer,Time)` the reload register is set to the desired value. The timer is deployed with `Timer.StartSingle(Timer)` that sets a one shot functionality or with `Timer.StartContinuous(Timer)` that reloads the timer after expiring. Internally the ALD (auto load) flag is set or cleared. It determines whether the timer is stopped or free running. The TimerCounterTime register can be accessed with the method `Timer.StartContinuous(Timer,Value)`. The timer structure is based on UINT16 registers. So the longest period is about 1 min. When expiring the timer can be programmed to trigger an event with a user event handler. The timer can be stopped and released any time.

**Example of a timer routine:**

```

SUB   Timer_Read_Buffer
      Timer.InstallLocal(0,Read_Buffer)      ; timer 0, call subroutine Read_Buffer
      Timer.Load(0,300)                     ; after 300 ms
      Timer.StartContinuous(0)               ; continuous timer, repeat without end
ENDSUB

```

In addition to the example above, the timer function needs to be enabled with `Timer.TableEnable()`, e.g. in the Mainprogramm (cf. p. 59).

**Timer.Init()**

The method `Timer.Init()` initializes all timers and all timers previously active are stopped. After execution of this method the timers have to be reinstalled.

**Attention:**

This method works on the whole timer section. If global timers are installed, these timers are also initialized.

## Timer: EventSection

The EventSection is responsible that a timer event can be executed.

### **Timer.InstallLocal(VarInt:Timer,labelRel24:Event)**

The method **Timer.InstallLocal(Timer,Event)** is used to install a local event handler. There are eight timers available, timer 0 to timer 7.

#### Example:

```
Timer.InstallLocal(0,Read_Buffer)      ; timer 0, call subroutine Read_Buffer
```

### **Timer.InstallGlobal(VarInt:Timer,labelAbsolute24:Event)**

The method **Timer.InstallGlobal(Timer,Event)** is used to install a global event handler for a timer. We recommend the event handler to be programmed as short as possible and not to write on the display, since all drawing routines consume a lot of CPU power. It is absolutely necessary that the event handler for a global timer is placed into the \*.EPR- file in order to work in all views of a project.

### **Timer.TableDisable()**

The method **Timer.TableDisable()** disables the timer table for generating local events. The table entries are still present and with the method **Timer.TableEnable()** the table can be enabled again. The method is used to temporarily disable timer events. At a view change this method is executed automatically.

### **Timer.TableEnable()**

The method **Timer.TableEnable()** enables the timer table. After the execution of this method timer events can be handled.

### **Timer.DeInstall(VarInt:Timer)**

The method **Timer.DeInstall(VarInt:Timer)** deinstalls a timer. If a timer has expired just before DeInstall, the event is still executed. If the event must not occur, the method **Timer.Kill(VarInt:Timer)** must be used instead.

**Timer.Kill(VarInt:Timer)**

The method **Timer.Kill(VarInt:Timer)** deinstalls a timer. If a timer has just expired the event will not be executed any more. It is strongly recommended to use this method for auto repeat keys or other timers that draw on the display. An other solution to prevent timers to draw, when not needed is to control the timer with a flag.

**Timer: TicSection**

The TicSection of the timer is executed every millisecond. Every timer is checked for its state RUN or STOP. If a timer is in RUN-Mode, the TimerCounter is decremented every ms. When reaching zero the timer is reload from an internal reload register and the expired is incremented and the EXP-Flag is set. Then the AUTORELOAD-Mode defines whether the timer remains in run mode (continuous) or goes to STOP (single).

**Timer.Load(VarInt:Timer,VarInt:Time[ms])**

The method **Timer.Load(Timer,Time)** loads the timer. The time is indicated in ms. The parameter is written to the internal RELOAD-register of the given timer. When the timer expires the next time of with the method **Timer.StartSingle()** or **Timer.StartContinuous()** the value is loaded into the internal TimerCounter register.

**Timer.StartSingle(VarInt:Timer)**

The method **Timer.StartSingle()** loads the TimerCounter from the Reload-register. The expiredCounter is reset and the timer enabled for one cycle. After reaching zero the timer increments the TimerExpired register and generates an event if the events are enabled.

**Timer.StartContinuous(VarInt:Timer)**

The method **Timer.StartContinuous()** loads the TimerCounter from the Reload-register. The expiredCounter is reset and the timer enabled for freely running cycles. After reaching zero the timer increments the TimerExpired and generates an event if the events are enabled.

**Timer.Stop(VarInt:Timer)**

The method **Timer.Stop()** stops a timer. It can be relaunched with the method **Timer.Continue()**.

**Timer.Continue(VarInt:Timer)**

The method **Timer.Continue()** relaunches a stopped timer after executing the method **Timer.Stop()**.

**Timer.Reload(VarInt:Timer)**

The method **Timer.Reload()** reloads the CounterTimer from the Reload-register of the given timer. If the timer is running, the method resets the time to the start. The method can be used to generate a timeout event.

**Timer.Get\_TimerCounter(VarInt:Timer,VarInt:TimerCounter)**

The method **Timer.Get\_TimerCounter()** gets the TimerCounter of the given timer as return value. The method can be used to check how much time is left until the timer expires or when the initial timer value is known, the difference is the duration from start to end.

**Timer.Get\_ExpiredCounter(VarInt:Timer,VarInt:ExpCounter)**

The method **Timer.Get\_ExpiredCounter()** gets the ExpiredCounter of a given timer. This counter is incremented each time the timer is expired. The method can be used to check if TimerEvents were get lost.

**Timer.Load\_ExpiredCounter(VarInt:Timer,VarInt:Value)**

The method **Timer.Load\_ExpiredCounter()** loads the ExpiredCounter of a given timer with a value that is passed as parameter.

**Timer.SUB\_ExpiredCounter(VarInt:Timer,VarInt:Value)**

The method **Timer.SUB\_ExpiredCounter()** subtracts the parameter value from the ExpiredCounter. If precise counts of the expired timers are used this method will perform a secured access to the ExpiredCounter.

Example:

```
Timer.Get_ExpiredCounter(MyTimer.I, MyExpiredCounter.I)  
Math.CVT_Long_from_Integer(MyExpiredCounter.L,MyExpiredCounter.I)  
TotalCounter.L := TotalCounter.L + MyExpiredCounter.L  
Timer.SUB_ExpiredCounter(MyTimer.I, MyExpiredCounter.I)
```

The Totalcounter is incremented by the ExpiredCounter. If in the mean time between Timer.Get\_ExpiredCounter and Timer.SUB\_ExpiredCounter the timer should have expired, the correct numbers of expiries is subtracted and the difference is considered next time.





## Serial asynchronous Interfaces RS232/RS485

The eigerPanels have serial communication lines to connect peripheral devices or to connect a PLC or a customer application to the eigerPanel. With the RS232 interface a point to point communication can be established, while the RS485 interface can form a bus with several bus nodes connected. The controlling of the RS485 bus transceiver is performed automatically by the firmware.

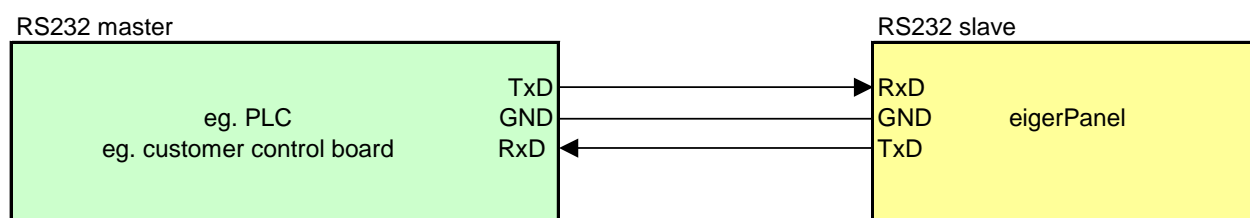
Depending on the system configuration, the connected device is either a master or a slave. On the eigerPanel therefore a slave or a master must be implemented. The master starts the communication and the slave has to respond to requests.

The characters sent by the RS232 asynchronous communication line are always 8 bit characters. The whole range from 0 to 255 is open to the communication. However the serial methods use always integer characters as parameters from and to the routines.

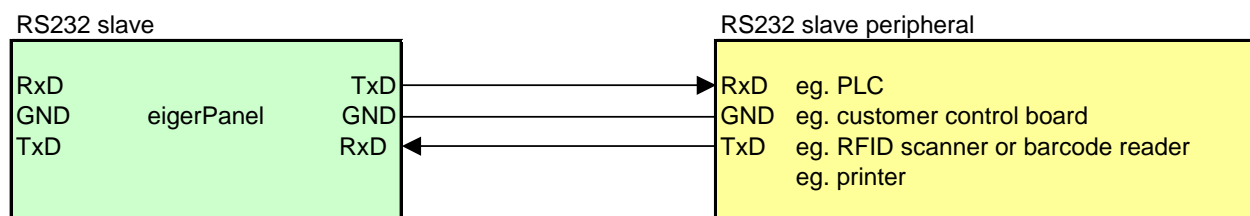
The characters can also be transmitted over RS485. The bus is intended to have several nodes communication with a master node. Please refer to the appropriate documents and literature for further details on RS232 and RS485. Especially the length limitations for both types of communication, the use of the correct cables, the type and configuration of connectors, the shielding, the bus termination and so on.

### Communication modes RS232

The master starts all communication and the slave sends a reply.

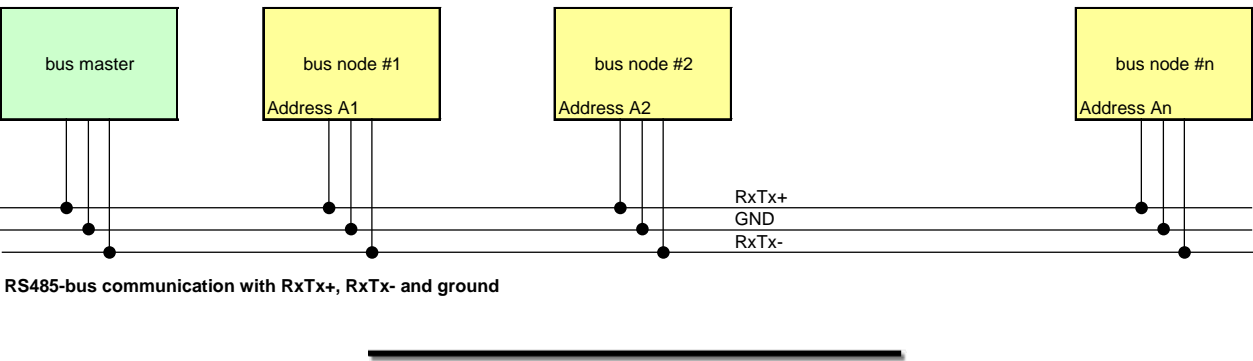


**RS232 3-wire communication with RxD, TxD and ground eigerPanel as slave**



**RS232 3-wire communication with RxD, TxD and ground eigerPanel as master**

Communication modes RS485



Class Serial

Initialize serial interface parameters

**Serial.SetBaudrate(VarInt:COMx,VarInt:Baudrate) 0\$40**

The method **Serial.SetBaudrate(VarInt:COMx,VarInt:Baudrate)** configures the baud rate that is the communication speed in bits/sec of an appropriate communication line. The Rx and the Tx channel always operate at the same speed.

The other parameters are:

- **8bits,**
- **no parity,**
- **one stopbit.**

The selectable baud rates are: 1'200, 2'400, 4'800, 9'600, 19'200, 38'400, 57'600, 115'200, 250'000. The preferred values for RS232 are underlined or highlighted in the table. For RS232 Baud\_250000 is not recommended.

Baudrate	Constant	
1'200	Baud_1200	
2'400	Baud_2400	
4'800	Baud_4800	
9'600	Baud_9600	
19'200	Baud_19200	
38'400	Baud_38400	(Default RS232)
57'600	Baud_57600	
115'200	Baud_115200	
RS485 250'000	Baud_250000	(Default RS485)

Example:

```
Serial.SetBaudrate(COM2,Baud_38400)
```

Receive characters**Serial.Rx\_Char(VarInt:COMx,VarInt:Char)****0\$40**

The method `Serial.Rx_Char(VarInt:COMx,VarInt:Char)` reads one character from the serial interface and removes it from there to the integer variable specified. The return value is an UINT16 valid characters range from 0..255. **If no character is present at the serial interface, the method returns -1.** The method doesn't wait until a character is present in order to prevent deadlocks in the system. If there are several characters in the buffer of the serial interface, the method takes the first character of the queue.

Example:

```
Serial.Rx_Char(COM2, MyCharacter.I)
```

Send characters**Serial.Tx\_Char(VarInt:COMx,VarInt:Char)****0\$40**

The method `Serial.Tx_Char(VarInt:COMx,VarInt:Char)` transmits one character over the serial interface.

```
Serial.Tx_Char(COM2,"A") ; Char A is transmitted over COM2
```

**Serial.Tx\_String(VarInt:COMx,VarStr)****0\$40**

The method `Serial.Tx_String(Kanal,String)` transmits a string over the serial interface. The terminating zero of the string is not transmitted. If it is necessary to transmit a zero, it can be performed by using the method `Serial.Tx_NUL(VarInt:COMx)`.

```
Serial.Tx_String(COM2,'Hello') ; Output Hello on COM2
```

Remark: If large strings are transmitted and the baud rate is not high, the method has to wait until the last character has space in the output buffer. In this case the system is blocked until the method returns.

**Serial.Tx\_CRLF(VarInt:COMx)**

The method `Serial.Tx_CRLF(VarInt:COMx)` sends a CR and a LF character over the indicated serial interface.

```
Serial.Tx_CRLF(COM2) ; Output CRLF on COM2
```

**Serial.Tx\_NUL(VarInt:COMx)**

The method `Serial.Tx_NUL(VarInt:COMx)` sends a Null-Character 0x00 over the serial interface. The method can be used to prevent a timeout on the receiver, when otherwise no communication is necessary.

```
Serial.Tx_NUL(COM2)           ; Output NUL on COM2
```

Remark: the terminating zero of a string is not transmitted. This method could be used to perform this.

**Send binary data (YMODEM)****Serial.SendFromFile(VarInt:COMx,VarStr:Filename)**

The method `Serial.SendFromFile(VarInt:COMx,VarStr:Filename)` outputs any file directly from the CompactFlash Card to a serial interface.

COMx:            either COM0 (serial interface RS485) or COM2 (serial interface RS232).

Filename:      Filename (full path name).

Example:

```
Serial.SendFromFile(COM2,'C:/MYAP/PICT/MyPict.egi').
```

The underlying protocol is the *YMODEM* with CRC checksum. So the file can be received by a PC using the communication program 'HyperTerminal'.

**HyperTerminal settings:**

1. Click **Start > Programs > Accessories > Communications**<sup>1</sup>, and click **HyperTerminal**. The Box "Connection Description" appears.
2. Specify the name for your new connection, e.g. "eigerPanel Transfer".
3. Select an icon by clicking one, e.g. 'open book with phone', and click OK. The box "Phone Number" appears.
4. Select and click in the options box "Connect using" the option **COM 1** or **COM 2**. Click **OK**. The box "COM 1 or COM 2 Properties" appears.
5. Define the following port settings:  
Bits per second: 9600 (according to the baudrate you chose for the eigerPanel)  
Data bits: 8  
Parity: none  
Stop bits: 1  
Flow control: non  
Click **OK**. "HyperTerminal" is activated.

<sup>1</sup> In German: Start > alle Programme > Zubehör > Kommunikation

6. Close "eigerPanel Transfer – HyperTerminal". A question appears.
7. By closing the Hyperterminal window answer the dialog with YES. Answering the following question, if the session should be stored, also with YES and you can use the same terminal properties when you connect the next time to your eigerPanel. To open the next session you can simply select your icon with the text "eigerPanel Transfer" after clicking **Start > Programs > Accessories > Communications > HyperTerminal (Folder)**.

### File transfer from the eigerPanel to your PC (HyperTerminal)

1. Connect your PC and the eigerPanel using a serial cable (e.g. the cable F4339, which is part of the eigerPanel StarterKit).
2. Open the HyperTerminal (Start > Programs > Accessories > Communications > HyperTerminal (Folder) > eigerPanel Transfer).
3. Set the eigerPanel to receive the file, according to your application on the eigerPanel, e.g. by pushing a button, which includes the method `Serial.SendFromFile(COM2, 'C:/MYAP/PICT/MyPict.egi')`. Please note that the file name is of the 8.3-type. Example: "YourName.txt".
4. Then in your PC's HyperTerminal go to "Transfer" menu and "Receive". The dialog "Receive File" appears.
5. In the "Receive file" dialog, set the protocol to YModem and navigate to the folder where you want to store the file to be received. Tap "OK" to receive the file.

### `Serial.ReceiveToFile(VarInt:COMx,VarStr:TargetFolder)`

The method `Serial.ReceiveToFile(VarInt:COMx,VarStr:TargetFolder)` is used for receiving any file transmitted by the YMODEM protocol. The transmitted file is saved in the target folder of the CompactFlash Card.

COMx: either **COM0** (serial interface RS485) or **COM2** (serial interface RS232).

Target Folder: Absolute path of the folder on the Compact Flash Card where you want to save the file transmitted to the eigerPanel by a communication program like HyperTerminal. Example: 'C:/TG12/PICT/'. Ensure that you enclose the path in apostrophes and that the path is terminated by a slash '/'. After the file has been received by the eigerPanel the string file contents the complete path including the file name transmitted. Therefore the String should not be replaced with a constant string.



**Attention:** The folder structure which is indicate as argument of the method `Serial.ReceiveToFile()` **must already exist** on the CompactFlash Card before you transmit any file to the eigerPanel. If the file is transmitted from eigerPanel to eigerPanel using the method `Serial.SendFromFile(...)`, the directory structure of the two Compact Flash Cards need to be identical, concerning source and target folder. In this case the argument 'target folder' can be an empty string ('').

Example:

```
Serial.ReceiveToFile(COM2, FileName.$) ; FileName.$ consists of file
name and full path
```

**File transfer from PC (HyperTerminal) to the eigerPanel**

6. Connect your PC and the eigerPanel using a serial cable (e.g. the cable F4339, which is part of the eigerPanel StarterKit).
7. Open the HyperTerminal (Start > Programs > Accessories > Communications > HyperTerminal (Folder) > eigerPanel Transfer).
8. Then go to "**Transfer**" menu and "**Send**". The dialog "Send File" appears.
9. In the "Send file" dialog, set the protocol to YModem and navigate to the file to be sent (Please note that the file name is of the 8.3-type. Example: "YourName.txt"). Tap "OK" to send the file.
10. Now set the eigerPanel to receive the file. according to your application on the eigerPanel, e.g. by pushing a button, which includes the method `Serial.ReceiveToFile(COM2, FileName.$)`.

**Possible sources of error :**

- *Baud rate:* Ensure that Sender and Receiver work with the same baud rate. In eigerScript the baud rate is set by the method `Serial.SetBaudrate(..)`. Further settings for the HyperTerminal: 8 data bits, 1 stop bit, no parity, no flow control.
- *Filename:* The file name has to be of the 8.3-type. Example: YourName.txt.
- *Directory structure:* Ensure that the target folder and its absolute path you have prompted in the method `Serial.ReceiveToFile(VarInt:COMx,VarStr:target folder)` exists on the CompactFlash Card.
- *Connecting cable:* The connecting cable should be a null modem cable.



## ■ Serial Two Wire Interface I2C

### Class I2C

The class I2C contains methods to communicate with the I<sup>2</sup>C-bus. The eigerPanel is always the (only) master and can't be a slave. The addressing is 7-bit and clock stretching is not supported.

The slave address 0x68 is reserved for the internal real time clock.

The slave address 0x5C is reserved for the capacitive touch.

### Application example

Example for a I<sup>2</sup>C-communication with an analog-to-digital converter.

```
CONST ADS1112_ADDRESS = 0x48

INTEGER byte1.I
INTEGER byte2.I

; read value from analog-to-digital converter
I2C.Start()
I2C.WriteConfigByte(ADS1112_ADDRESS, I2C_DATADIRECTION_READ)
I2C.ReadStreamByte(byte1.I)
I2C.ReadEndByte(byte2.I)
I2C.Stop()
```

### **I2C.Init()**

Initialise the ports for communication with the I<sup>2</sup>C-bus (the firmware does this automatically at start-up of the system).

### **I2C.SetMode(VarInt:I2C\_MODE\_\*)**

Set the data rate of the I<sup>2</sup>C-bus. Up to now only the standard mode with 100kHz is available and this mode is set as default.

Example:

```
I2C.SetMode(I2C_MODE_STANDARD)
```

**I2C.Start()**

Generate a START condition on the I<sup>2</sup>C-bus. It is used to begin a transaction.

**I2C.RepeatedStart()**

Generate a repeated START condition on the I<sup>2</sup>C-bus. The method is functionally identical to **I2C.Start()**.

**I2C.WriteConfigByte(VarInt:SlaveAddress,VarInt:I2C\_DATADIRECTION\_\*)**

Send the slave address and the data direction bit on the I<sup>2</sup>C-bus.

For the parameter I2C\_DATADIRECTION\_\*, the constants **I2C\_DATADIRECTION\_READ** and **I2C\_DATADIRECTION\_WRITE** are possible.

Example:

```
I2C.WriteConfigByte(0x48,I2C_DATADIRECTION_WRITE)
```

**I2C.WriteByte(VarInt:Value)**

Send a data byte on the I<sup>2</sup>C-bus.

Example:

```
I2C.WriteByte(0xAA)
```

**I2C.ReadStreamByte(VarInt:Value)**

Read a data byte on the I<sup>2</sup>C-bus and generate an acknowledgment (ACK). It is used to read more bytes after executing this method.

Example:

```
I2C.ReadStreamByte(value.I)
```

**I2C.ReadLastByte(VarInt:Value)**

Read a data byte on the I<sup>2</sup>C-bus and generate a not-acknowledgment (NACK). This method can be used to read the last or the only byte from a slave.

Example:

```
I2C.ReadLastByte(value.I)
```

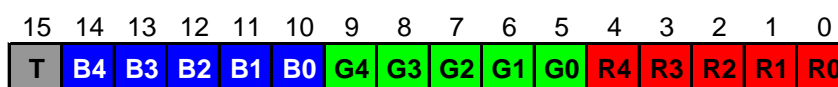


**I2C.Stop()**

Generate a STOP condition on the I<sup>2</sup>C-bus. It is used to terminate a transaction.

**Class Colors\_15**

The Class Colors\_15 is used to manipulate colours on the eigerPanel. The eigerPanel works with 16bit colour variables. The internal representation of the COLOR15 data type is:



The selected data type is a good compromise between colour representation and the bit 15, which is used to encode the transparency or commands in the EGI-Format. The EGI (eiger graphics image format) is a lossless image format used to encode images and graphics on the eigerPanel. The format is converted from and to PC image formats with the eiger graphics suite software. Free download from [www.eigergraphics.com/download.htm](http://www.eigergraphics.com/download.htm).

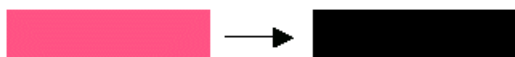
The methods in the class Colors\_15 are used to:

manipulate colours, darken, brighten grayvalue  
convert colours from and to 24bit colour space

**Change colours****Colors\_15.AutoColor(VarInt:Destination,VarInt:Source)**

The method **Colors\_15.AutoColor(Destination,Source)** sets the destination with the colour black or white depending on the colour of the source. For dark colours the method returns white and for bright colours the method returns black. The method is used to set the text colour with maximum contrast automatically.

**Colors\_15.AutoColor(eI.TextColor,eI.BackColor)**

**Special case:**

If the input colour is transparency, the method doesn't change the destination colour.



**Colors\_15.Brighten(VarInt:Dest,VarInt:Source)**

The method `Colors_15.Brighten(Destination,Source)` returns a brightened colour of the source colour.

`Colors_15.Brighten(eI.LineColor,eI.BackColor)`

Special case:

If the input colour is transparency, the method doesn't change the destination colour.

**Colors\_15.Darken(VarInt:Dest,VarInt:Source)**

The method `Colors_15.Darken(Destination,Source)` returns a darkened colour of the source colour.

`Colors_15.Darken(eI.LineColor,eI.BackColor)`

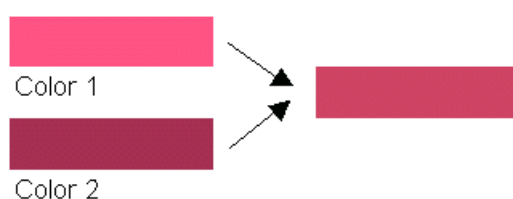
Special case:

If the input colour is transparency, the method doesn't change the destination colour.

**Colors\_15.ColorMix(VarInt:Dest,VarInt:Color1,VarInt:Color2)**

The method `Colors_15.ColorMix(Destination,Color1,Color2)` returns the mixed colour of color1 and color2. Every colour can be mixed with the grey palette to obtain different colours from a base colour. The colours are mixed in a 1:1 ratio.

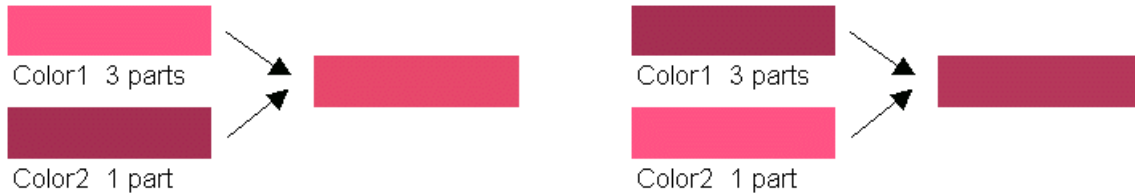
`Colors_15.ColorMix(eI.LineColor,eI.BackColor,greyscale204)`



```
Colors.ColorMix_3to1_5Bit(VarInt:Dest,VarInt:Color1,  
VarInt:Color2)
```

The method `Colors_15.ColorMix_3to1(Destination,Color1 3 parts,Color2 1 part)` returns the mixed colour of the two input colours. The colours are mixed in a 3:1 ratio.

```
Colors_15.ColorMix_3to1(eI.LineColor,eI.BackgroundColor,greyscale204)
```



```
Colors_15.InverseColor(VarInt:Destination,VarInt:Source)
```

The method `Colors.InverseColor(Destination,Source)` returns the complementary colour of the source colour.

```
Colors.InverseColor(eI.LineColor,eI.BackgroundColor)
```



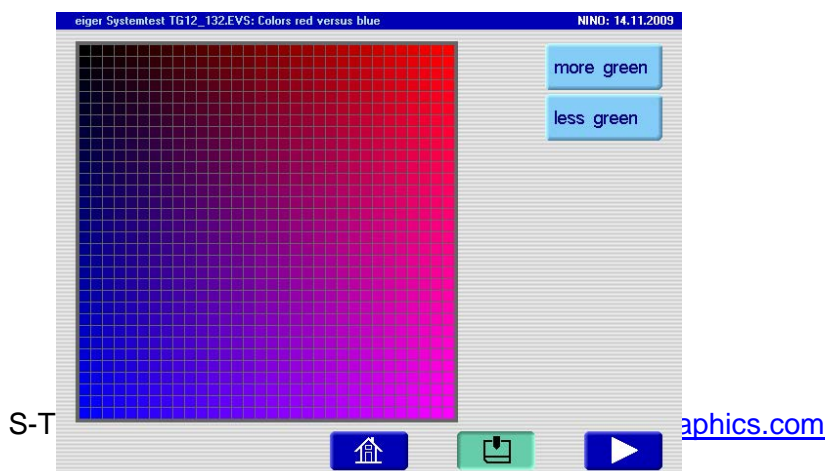
Special case:

If the input colour is transparency, the method doesn't change the destination colour.

## Manipulate colour channels

The methods subsequently described are used to manipulate single colour channels of the colour variable. These methods are perfectly suited to draw colour gradients.

This view shows a rectangular colour flares computed with the methods described below



## **Colors\_15.MoreRed(VarInt:COLOR)**

The method **Colors\_15.MoreRed(Color)** adds one TIC to the red channel. If the colour is already at its maximum value, it becomes zero again.

**Colors\_15.MoreRed(eI.FillColor)**

### Special case:

If the input colour is transparency, no change is performed.

## **Colors\_15.MoreRedLimit(VarInt:COLOR)**

The method **Colors\_15.MoreRedLimit(Color)** adds one TIC to the red channel. If the colour is already at its maximum value, it remains unchanged.

**Colors\_15.MoreRedLimit(eI.FillColor)**

### Special case:

If the input colour is transparency, no change is performed.

## **Colors\_15.MoreGreen(VarInt:COLOR)**

The method **Colors\_15.MoreGreen(Color)** adds one TIC to the green channel. If the colour is already at its maximum value, it becomes zero again.

**Colors\_15.MoreGreen(eI.FillColor)**

### Special case:

If the input colour is transparency, no change is performed.

## **Colors\_15.MoreGreenLimit(VarInt:COLOR)**

The method **Colors\_15.MoreGreenLimit(Color)** adds one TIC to the green channel. If the colour is already at its maximum value, it remains unchanged.

`Colors_15.MoreGreenLimit(eI.FillColor)`

Special case:

If the input colour is transparency, no change is performed.

**`Colors_15.MoreBlue(VarInt:COLOR)`**

The method `Colors_15.MoreBlue(Color)` adds one TIC to the blue channel. If the colour is already at its maximum value, it becomes zero again.

`Colors_15.MoreBlue(eI.FillColor)`

Special case:

If the input colour is transparency, no change is performed.

**`Colors_15.MoreBlueLimit(VarInt:COLOR)`**

The method `Colors_15.MoreBlueLimit(Color)` adds one TIC to the blue channel. If the colour is already at its maximum value, it remains unchanged.

`Colors_15.MoreBlueLimit(eI.FillColor)`

Special case:

If the input colour is transparency, no change is performed.

**`Colors_15.LessRed(VarInt:COLOR)`**

The method `Colors_15.LessRed(Color)` subtracts one TIC to the red channel. If the colour is already at its minimum value, it becomes max again.

`Colors_15.LessRed(eI.FillColor)`

Special case:

If the input colour is transparency, no change is performed.

**`Colors_15.LessRedLimit(VarInt:COLOR)`**

The method `Colors_15.LessRedLimit(Color)` subtracts one TIC to the red channel. If the colour is already at its minimum value, it remains unchanged.

**Colors\_15.LessRedLimit(eI.FillColor)**

Special case:

If the input colour is transparency, no change is performed.

**Colors\_15.LessGreen(VarInt:COLOR)**

The method **Colors\_15.LessGreen(Color)** subtracts one TIC to the green channel. If the colour is already at its minimum value, it becomes max again.

**Colors\_15.LessGreen(eI.FillColor)**

Special case:

If the input colour is transparency, no change is performed.

**Colors\_15.LessGreenLimit(VarInt:COLOR)**

The method **Colors\_15.LessGreenLimit(Color)** subtracts one TIC to the green channel. If the colour is already at its minimum value, it remains unchanged.

**Colors\_15.LessGreenLimit(eI.FillColor)**

Special case:

If the input colour is transparency, no change is performed.

**Colors\_15.LessBlue(VarInt:COLOR)**

The method **Colors\_15.LessBlue(Color)** adds subtracts TIC to the blue channel. If the colour is already at its minimum value, it becomes max again.

**Colors\_15.LessBlue(eI.FillColor)**

Special case:

If the input colour is transparency, no change is performed.

**Colors\_15.LessBlueLimit(VarInt:COLOR)**

The method **Colors\_15.LessBlueLimit(Color)** subtracts one TIC of the blue channel. If the colour is already at its minimum value, it remains unchanged.

```
Colors_15.LessBlueLimit(eI.FillColor)
```

#### Special case:

If the input colour is transparency, no change is performed.

### Set or get colour values

The colour values are packed in a UINT16 dataword. The single components red green and blue can be extracted and composed with methods. There are also methods to convert a 24bit colour value to the eiger COLOR15 representation.

```
Colors_15.Load_RGB(VarInt:Destination,VarInt:R,VarInt:G,
VarInt:B)
```

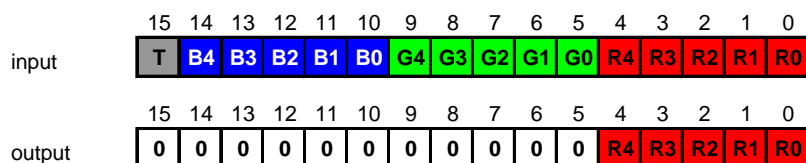
The method `Colors_15.Load_RGB(Destination,R,G,B)` returns the packed COLOR15 representation of the colour composed of the red, green and blue channels. Only the low byte of the RGB input values is used. The input values have to be within the range of [0..255]. The method is useful to convert the 24-bit colour space to the COLOR15 colour space used on the eigerPanel.

```
Colors_15.Load_RGB(eI.TextColor,0,0,255)
```

```
Colors_15.GetRed(VarInt:Destination,VarInt:Color)
```

The method `Colors_15.GetRed(Destination,Color)` returns the 5-bit red component of a colour. The return value is in the range of [0..31].

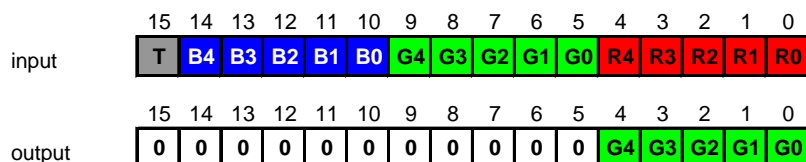
```
Colors_15.GetRed(eI.R00,eI.FillColor)
```



```
Colors_15.GetGreen(VarInt:Destination,VarInt:Color)
```

The method `Colors_15.GetGreen(Destination,Color)` returns the 5-bit green component of a colour. The return value is in the range of [0..31].

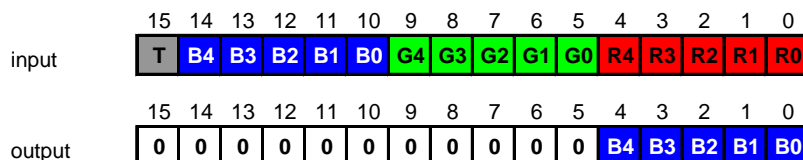
```
Colors_15.GetGreen(eI.R00,eI.FillColor)
```



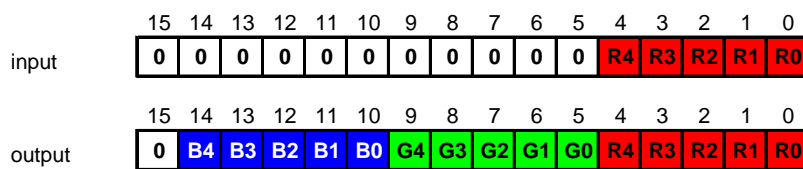


**Colors\_15.GetBlue(VarInt:Destination,VarInt:Color)**

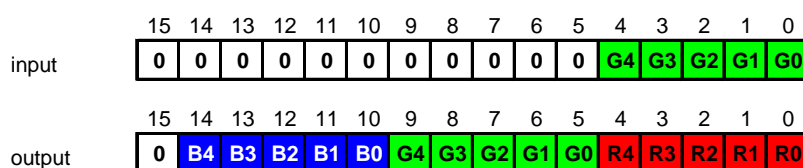
The method **Colors\_15.GetBlue(Destination,Color)** returns the 5-bit green component of a colour. The return value is in the range of [0..31].

**Colors\_15.GetBlue(eI.R00,eI.FillColor)****Colors\_15.PutRed(VarInt:Destination,VarInt:Color)**

The method **Colors\_15.PutRed(Destination,Color)** replaces the 5-bit red component of a colour in the destination colour variable.

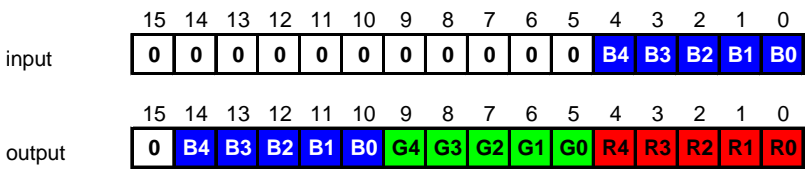
**Colors\_15.PutRed(eI.R00,eI.FillColor)****Colors\_15.PutGreen(VarInt:Destination,VarInt:Color)**

The method **Colors\_15.PutGreen(Destination,Color)** replaces the 5-bit green component of a colour in the destination colour variable.

**Colors\_15.PutGreen(eI.R00,eI.FillColor)****Colors\_15.PutBlue(VarInt:Destination,VarInt:Color)**

The method **Colors\_15.PutBlue(Destination,Color)** replaces the 5-bit blue component of a colour in the destination colour variable.

**Colors\_15.PutBlue(eI.R00,eI.FillColor)**



Exchange colour channels

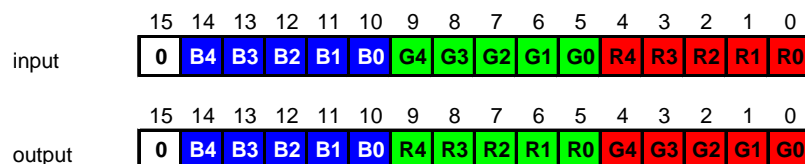
All colours are composed of three colour composites red, green and blue. In eigerScript there are methods to exchange colour components what gives related colours in the colour circle (c.f. colour table below). Starting from hotpink in the left upper corner the consecutive order of colours is shown for this example.

	RGB	COLOR15	WEB
<div>swap R-G</div> 	148, 17,255	0x7C52	#9411FF
<div>swap R-G</div> 	17,148,255	0x7E42	#1194FF
<div>swap R-G</div> 	17,255,148	0x4BE2	#11FF94
<div>swap R-G</div> 	148, 255,17	0x0BF2	#94FF11
<div>swap R-G</div> 	255,148,17	0x0A5F	#FF9411
<div>swap R-G</div> 	255,17,148	0x485F	#FF1194
<div>swap R-G</div> 	148, 17,255	0x7C52	#9411FF
<div>swap G-B</div> 			
<div>swap G-B</div> 			
<div>swap G-B</div> 			
<div>swap G-B</div> 			
<div>swap G-B</div> 			
<div>swap G-B</div> 			
<div>swap G-B</div> 			
<div>swap B-R</div> 			
<div>swap B-R</div> 			
<div>swap B-R</div> 			
<div>swap B-R</div> 			
<div>swap B-R</div> 			
<div>swap B-R</div> 			
<div>swap B-R</div> 			

```
Colors_15.Swap_Red_Green(VarInt:Dest,VarInt:Source)
```

The method Colors\_15.Swap\_Red\_Green(Destination,Source) swaps the red and the green colour component.

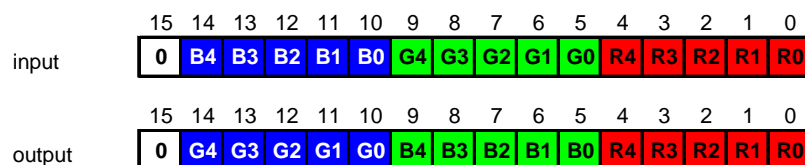
```
Colors_15.Red_Green(eI.FillColor,eI.FillColor)
```



### Colors\_15.Swap\_Green\_Blue(VarInt:Dest,VarInt:Source)

The method `Colors_15.Swap_Green_Blue(Destination,Source)` swaps the green and the blue colour component.

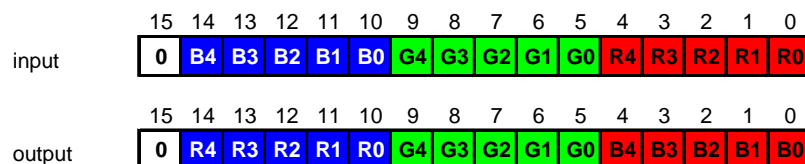
`Colors_15.Swap_Green_Blue(eI.FillColor,eI.FillColor)`



### Colors\_15.Swap\_Blue\_Red(VarInt:Dest,VarInt:Source)

The method `Colors_15.Swap_Blue_Red(Destination,Source)` swaps the blue and the red colour component.

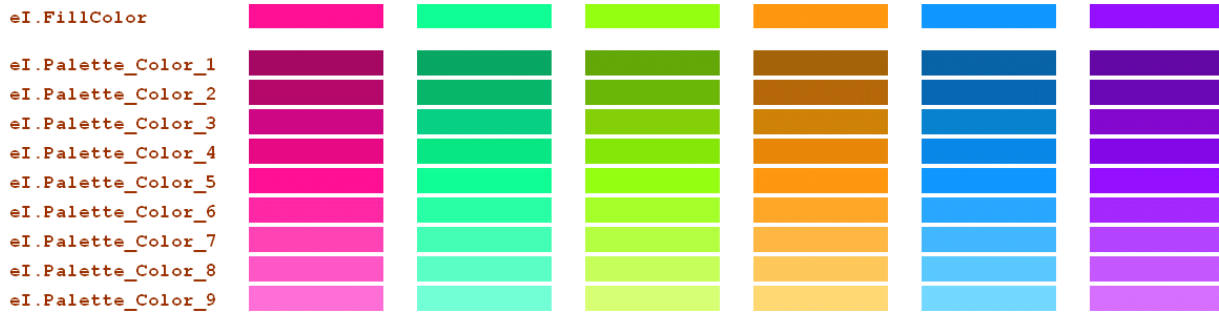
`Colors_15.Swap_Blue_Red(eI.FillColor,eI.FillColor)`



### Colors\_15.SetColorPalette(VarInt:Source)

The method `Colors_15.SetColorPalette(Source)` fills the appropriate colour palette to the `eI.Palette_Color_x` registers. Below an example is given for reference. The method is internally used to compute colour gradients for objects.

`Colors_15.SetColorPalette(eI.FillColor)`



### Colors\_15.Cvt\_ColorComp\_5to8(VarInt:Color8B,VarInt:Color5B)

The method `Colors_15.Cvt_ColorComp_5to8(Destination,Color)` converts a 5-bit colour component in a 8-bit colour component. The conversion is established with a lookup table.

`Colors_15.Cvt_ColorComp_5to8(eI.R00,eI.FillColor)`

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
input	0	0	0	0	0	0	0	0	0	0	0	C54	C53	C52	C51	C50
output	0	0	0	0	0	0	0	0	C87	C86	C85	C84	C83	C82	C81	C80

### Colors\_15.Cvt\_ColorComp\_8to5(VarInt:Color5B,VarInt:Color8B)

The method `Colors_15.Cvt_ColorComp_8to5(Destination,Color)` converts a 8-bit colour component in a 5-bit colour component. The conversion is established with a lookup table.

`Colors_15.Cvt_ColorComp_8to5(eI.FillColor,eI.R00)`

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
input	0	0	0	0	0	0	0	0	C87	C86	C85	C84	C83	C82	C81	C80
output	0	0	0	0	0	0	0	0	0	0	0	C54	C53	C52	C51	C50

#### Application Hint:

The code sequence converts a 8-bit colour component to a 5-bit colour component and back with the effect of having the exact 8-bit colour value as on the eigerPanel. For RGB colours it has to be done with every single colour component.

```
Colors_15.Cvt_ColorComp_8to5(eI.FillColor,eI.R00)
Colors_15.Cvt_ColorComp_5to8(eI.R00,eI.FillColor)
```

**Colors\_15.GetPixelColor(VarInt:Ziel,VarInt:X,VarInt:Y)**

The methods **Colors\_15.GetPixelColor(Target,X,Y)** returns the colour of the pixel at absolute coordinates (X/Y) in the AVR video RAM. It is a good idea to think the method as a pipette in a graphics program. When the registers **eI.Mouse\_X\_Down** and **eI.Mouse\_Y\_Down** are used as coordinates, a colour selector can be programmed.

Example 1:

Get the pixel colour at coordinate (50,355)

```
Colors.GetPixelColor(eI.FillColor,50,355)
```

Example 2:

Get the pixel colour at coordinate (**eI.Mouse\_X\_Down**, **eI.Mouse\_Y\_Down**)

```
Colors.GetPixelColor(eI.FillColor,eI.Mouse_X_Down,eI.Mouse_Y_Down)
```



## Class InOut

The methods of the class InOut serve to handle the peripherals on the FOX embedded computer. Depending on the model of the computer the features may differ.

For the available peripherals on a given computer please refer to the relevant hardware manual available at [www.eigergraphics.com](http://www.eigergraphics.com).

**InOut.Read\_ADC(VarInt:Channel,VarInt:Value)**

The method **InOut.Read\_ADC(Channel,TargetVariable)** reads an analog input channel from an analog to digital converter DAC. The value that is converted is returned in a variable or a register as second parameter.

```
InOut.Read_ADC(0,eI.R02)
```

Remark:

For computers with a 10-bit ADC the input range is from 0 to 1023.

The number of input channels and the sensitivity TIC/V depend on the computer model.

**InOut.Read\_Input(VarInt:InputPort,VarInt:Value)**

The method **InOut.Read\_Input(Channel,TargetVariable)** reads a digital input from a port. The value (0 or 1) is returned in a variable or a register as second parameter.

```
InOut.Read_Input(Input_IP91,eI.R02)
```

Remark:

The number of input channels depend on the computer model.

**InOut.PWM\_Out(VarInt:OutputPort,VarInt:Value)**

The method **InOut.PWM\_Out(Channel,TargetVariable)** sets a PWM capable output channel with a value. The value is in the range of 0..1000.

**InOut.PWM\_Out(Output\_OP76,500)**

Remark:

The number of PWM output channels depend on the computer model.

## Class Fill

The methods of the class Fill are used to fill often used registers in the eVM quickly. A label object for instance is described by 17 properties that have to be filled into the registers. Most often the object properties of different objects are the same. The simplest solution is to store the properties in structures. It is also possible to hold the structures in an include file. So changes are performed quickly and consistent when a batch compile is executed. The structures are addressed by a pointer as parameter to the fill method. It is vital to respect the order of the properties in the structure otherwise unexpected effects may occur.

Special codes are used to control the fill process to the registers:

**no\_change**..... the content of a register remains unchanged.  
**as\_FillColor** ..... the colour is copied from the register **eI.FillColor**.  
**as\_DisplayColor** ..... the colour is copied from the register **eI.DisplayColor** as background colour  
**autocolor**..... the colour is copied from the register **eI.FillColor** and the Autocolour method is used to compute the best contrast. This is specially to assign the colour of the **eI.TextColor**.  
**darken\_FillColor**..... the shaded colour of the colour in the register **eI.FillColor** is assigned. This is interesting for the register **eI.LineColor** used for borders  
**brighten\_FillColor**..... the brightened colour of the colour in the register **eI.FillColor** is assigned.

**Fill.LabelParameter(labelRelative24)**

The method **Fill.LabelParameter(labelRelative24)** loads the registers that are used for a label object. It is very important to know, that with special codes operations can be performed to influence the load process.

Example:

```

Fill.LabelParameter(AK1_Key_Structure_UP)      ; load properties to the registers

SUB Structures

AK1_Key_Structure_UP:

    INLINENWORDS (no_change)                ; corresponds to eI.Pos_X1
    INLINENWORDS (no_change)                ; corresponds to eI.Pos_Y1
    INLINENWORDS (AK1_Key_Width)            ; corresponds to eI.Width
    INLINENWORDS (AK1_Key_Height)           ; corresponds to eI.Height
    INLINENWORDS (8)                        ; corresponds to eI.SpaceLeft
    INLINENWORDS (8)                        ; corresponds to eI.SpaceRight
    INLINENWORDS (0)                        ; corresponds to eI.HorizontalAdjust
    INLINENWORDS (0)                        ; corresponds to eI.VericalAdjust
    INLINENWORDS (steelblue)                ; corresponds to eI.FillColor
    INLINENWORDS (as_FillColor)              ; corresponds to eI.BackgroundColor
    INLINENWORDS (as_FillColor)              ; corresponds to eI.LineColor
    INLINENWORDS (autocolor)                 ; corresponds to eI.TextColor
    INLINENWORDS (Pos_center)                ; corresponds to eI.Position
    INLINENWORDS (Orientation_0deg)           ; corresponds to eI.Orientation
    INLINENWORDS (normal)                   ; corresponds to eI.Appearance
    INLINENWORDS (color_button_3D_raised)    ; corresponds to eI.BorderStyle
    INLINENWORDS (Font_Arial_14n)            ; corresponds to eI.FontNumber
    INLINENWORDS (as_Skin_FormBodyColor)     ; corresponds to eI.BackgroundColor
ENDSUB

```

## Class Load

The methods of the class Load are used to load often used combinations of registers. The values are given as parameters. The order of the parameters is very important in order to have the right result.

### Load.Pos\_X1Y1(VarInt:X1,VarInt:Y1)

The method **Load.Pos\_X1Y1(X-Position,Y-Position)** loads the registers **eI.Pos\_X1** and **eI.Pos\_Y1**.

```
Load.Pos_X1Y1(50,100)
```

is equal to:

```
eI.Pos_X1 := 50
eI.Pos_Y1 := 100
```

### Load.Pos\_X2Y2(VarInt:X2,VarInt:Y2)

The method **Load.Pos\_X2Y2(X-Position,Y-Position)** loads the registers **eI.Pos\_X2** and **eI.Pos\_Y2**.

```
Load.Pos_X2Y2(150,300)
```

**Load.Width\_Height(VarInt:W,VarInt:H)**

The method `Load.Width_Height(Width,Height)` loads the registers `eI.Width` and `eI.Height`.

**Load.Geometry\_XYWH(VarInt:X,VarInt:Y,VarInt:W,VarInt:H)**

The method `Load.Geometry_XYWH(X-Position,Y-Position,Width,Height)` loads the registers `eI.Pos_X1` and `eI.Pos_Y1` and also `eI.Width` and `eI.Height`. The method is useful to span a rectangle for a hotspot or an image.

```
Load.Geometry_XYWH(200,30,150,300)
```

is equal to:

```
eI.Pos_X1  := 200
eI.Pos_Y1  := 30
eI.Width   := 150
eI.Height  := 300
```

**Load.Offset\_XY(VarInt:X,VarInt:Y)**

The method `Load.Offset_XY(x,y)` loads the registers `eI.Offset_X` and `eI.Offset_Y`.

```
Load.Offset_XY(200,30) ; eI.Offset_X := 200, eI.Offset_Y := 30
```

**Load.Color\_FL(VarInt:FillColor,VarInt:LineColor)**

The method `Load.Color_FL(FillColor,LineColor)` loads the registers `eI.FillColor` and `eI.LineColor` with a colour value.

```
Load.Color_FL(red,crimson) ; eI.FillColor and eI.LineColor
```

**Load.Color\_BT(VarInt:BackColor,VarInt:TextColor)**

The method `Load.Color_BT(BackColor,TextColor)` loads the registers `eI.BackColor` and `eI.TextColor` with a colour value.

```
Load.Color_BT(red,white) ; eI.BackColorText and eI.TextColor
```



## Tip:

The method `Label.Color(color)` can be used to set the colours of all four registers.

---

## Class Transfer

The methods of the class Transfer are used to load or copy groups of often used eVM registers. The registers hold values that are transferred into other registers. The class Transfer has also methods to transport variables of same length but different types.

### **Transfer.HotSpotGeometry()**

All geometric data of a HotSpot are stored in the HotSpot-table as position, offset and size. For the animation of a button exactly this information is useful. The method `Transfer.HotSpotGeometry()` loads the geometry registers with the return values of a HotSpots. The method executes the following code sequence:

```
eI.Width      := eI.HS_Width      ; width of the HotSpot
eI.Height     := eI.HS_Height     ; height of the HotSpot
eI.Pos_X1     := eI.HS_Pos_X      ; X-Position of the HotSpot
eI.Pos_Y1     := eI.HS_Pos_Y      ; Y-Position of the HotSpot
eI.Offset_X   := eI.HS_Offset_X   ; X-Offset of the HotSpot
eI.Offset_Y   := eI.HS_Offset_Y   ; Y-Offset of the HotSpot
```

The method is mainly used to implement a position-independent control.

---

## ■ Program control flow

eigerScript uses keywords to implement flow control structures as loops, selections and conditional code sequences.

### Infinite loop

The infinite loop is used to stay in the view while the eVM executes the events that are generated. Between the **LOOP** - **ENDLOOP** statement commands may be executed. The loop can be left by the **EXITLOOP** statement.

```
LOOP
    ---
    EXITLOOP
    ---
ENDLOOP
```

Remark: Up to version 0.67 of eigerStudio **LOOP** - **ENDLOOP** can only be used in the view source files (\*.EVS). In the project definition file (\*.EPR) it does not work.

### FOR-NEXT loop

The **FOR-NEXT** loop runs a code in the loop for a defined number of times. After all of the statements have been executed in the first loop cycle, the program increments the value of the loop counter (e.g. **Counter.I**) by one. The loop can be left by the **EXITFOR** statement when an error has occurred.

```
FOR Counter.I := Start.I TO End.I
    ---
    EXITFOR
    ---
NEXT
```

### REPEAT-UNTIL loop

The **REPEAT-UNTIL** loop runs a loop cycle at least once and at the end checks a condition to decide whether the loop has to be repeated. The loop cycle can be left by the **EXITREPEAT** statement when an error has occurred.

```
REPEAT
    ---
    EXITREPEAT
    ---
UNTIL eI.R00 == 5
```

Example:

```

REPEAT
    y := 2 * y
    IF b == 0 THEN
        EXITREPEAT
    ENDIF
    y := y / b
UNTIL y > 100

```

**WHILE-DO loop**

The **WHILE-DO** loop checks a condition before entering the loop. If the condition is false the code in the loop is never executed. At the loop end the check is executed and the code is skipped if the condition is false and run if the condition is true respectively. The loop can be left by the **EXITWHILE** statement when an error has occurred.

```

WHILE eI.R00 <= 10
    ---
    EXITWHILE
    ---
ENDWHILE

```

Example:

```

WHILE y <= 100
    y := 2 * y
    IF b == 0 THEN
        EXITWHILE
    ENDIF
    y := y / b
ENDWHILE

```

**IF THEN ELSIF ELSE ENDIF conditional code sequence**

The **IF-THEN-ELSIF-ELSE-ENDIF** statement enables conditional code execution. If a condition is true the code is executed. The simplest form is the **IF-THEN-ENDIF** statement that checks a condition that is executed, when the condition is true.

```

IF eI.R00 < 18 THEN
    Label.Text ('cold')
ENDIF

```

The statement **IF-THEN-ELSE-ENDIF** checks a condition and executes the first code block if the condition is true and the second code block if the condition is false.

```
IF eI.R00 < 18 THEN
    Label.Text ('cold')
ELSE
    Label.Text ('not cold')
ENDIF
```

The statement **IF-THEN-ELSIF-ELSE-ENDIF** checks a first condition and executes the first code block if the condition is true, then it checks another condition and executes the associated code block if the condition is true. As many **ELSIF** statements can be placed to check all conditions. This eliminates the need of a switch statement or a case statement as found in other programming languages.

```
IF eI.R00 < 18 THEN
    Label.Text ('cold')
ELSIF eI.R00 < 25 THEN
    Label.Text ('normal')
ELSIF eI.R00 < 32 THEN
    Label.Text ('hot')
ELSE
    Label.Text ('very hot')
ENDIF
```

---

## ■ Error handling

Several methods can end up in an error condition because for example input parameters are invalid or the desired result can not be achieved as in the case of a division by zero. The error is signalled by the eVM in two registers:

**eI.Status** general Status: **error**/**success**

The eVM is programmed so that if a method has an error, it returns **error** in the **eI.Status** register. Before entering a section that is interesting for error checking, the **eI.Status := success** is assigned. Every method of a sequence then may set the **eI.Status** to the **error** state. This feature of the eVM frees the programmer from a bold error handling, when this is not desired.

The method that throws an error writes an error code in the register **eI.ErrorCode**. With this error code the error can further be examined. For example the string methods return error codes that say if a position was not in the string or that there was not enough space in the string.



## ■ Debugging code

The methods of the class Debug serve to find run time errors. When the content of variables has to be traced without writing them on the screen the methods of the class Debug are useful for this task. The debug strings are output on the serial interface **COM1** that is reserved for debugging and programming the system.

Sometimes HotSpots want to be examined. Then it is a good idea to draw a rectangle around the HotSpot to see where the HotSpot was installed.

### Turn debug mode OFF/ON

Too many debug outputs can make an application slow. If this is the case you may turn debug outputs off by assigning the value 0 to the register **eI.eVM\_DebugMode**:

**eI.eVM\_DebugMode := 4** → Debug Mode is ON (default)

**eI.eVM\_DebugMode := 0** → Debug Mode is OFF

Works from Firmware V2.04 (20.06.2013) and higher.

The change of debug mode is temporarily, that means it is only turned off (or on) in the subroutine, where the mode is being changed. If you want to set the debug mode off for a whole view, you need to place the changing code into the main program (between **BEGINVIEW** and **ENDVIEW**).

In a view, where the debug mode is turned off, you still can use a debug method, e.g. for control reason. In that case you need to set **eI.eVM\_DebugMode := 4** before using the debug method. After exiting that subroutine debug mode is automatically turned off (=0) again.



## Working with CSV-Files

### What are CSV-Files ?

CSV is the abbreviation of **comma separated value-files**. Basically a CSV-file is an ASCII text file, which can be edited with an editor or with WordPad. This file is used as a small database as a very simple EXCEL sheet. The file ending is often \*.CSV. A CSV-file consists of data lines. Every data line holds a data record. The columns are separated by a separator character. The separator character we (in Europe) use is the semicolon (;) 0x3B. When in EXCEL a file is stored, EXCEL separates the columns by a semicolon. The end of line is formed by CR LF (0x0A,0x0D). The CSV-format is not standardized. When we implemented the CSV on the eVM we had the design goal to be as compatible as possible to the EXCEL spread sheet program. For this the data exchange with EXCEL works pretty well.

The eigerVM has methods to use CSV-files as small databases. Values can be read from and searched in a CSV-file. **You only can apply the methods of class CSV to a CSV-file after its content is read into a CSV-string** (e.g. `MatchFile.$`). The **class File** incorporates the method to read CSV-files:

```
STRING          [40000] MatchFile.$ = ''  
  
File.Read_CSV('C:/TG12/DATA/COLORS.CSV',MatchFile.$)
```

In this example "MatchFile" is defined as a string with max. 40'000 characters. If the file is smaller of course a smaller size for the string can be used. The limit is however at 65'000 characters due to the addressing with 16 bit pointers. If the file is longer than the string, the file is not read entirely and data is lost.

At the time of loading the file it is analyzed, and checked for the number of columns and lines.

As example there is a CSV-file with nonexistent test addresses. EXCEL stores the input as below and reads the file without problems. The only thing that is not stored in a CSV is the formatting as font attributes, colours and row sizes, as it is normal for a \*.XLS file.

Many programs can handle CSV-files. That makes a data exchange possible. The advantage of the CSV-file is the compact representation of data. It is much more compact than the XML-format, because only the data but not opening and closing tags are stored. The XML-files are from 9 to 80 times larger than the similar CSV-files. On systems with limited resources the CSV-format is the best selection.

It is possible to have a column descriptor in the first line. The first line is handled as standard data record by the eVM.

After edition of a CSV-string the content can be written back as CSV-file. The **class File** incorporates the method to write CSV-files:

```
File.Write_TextFile('C:/TG12/DATA/COLORS.CSV',MatchFile.$)
```

This method overwrites the the current CSV-file on the CompactFlash Card with the new content.

Another method of class **File** can be used to directly append new record at the end of a CSV-file:

```
File.AppendString( 'C:/TG12/DATA/LOG.CSV' , NewRecord.$ )
```

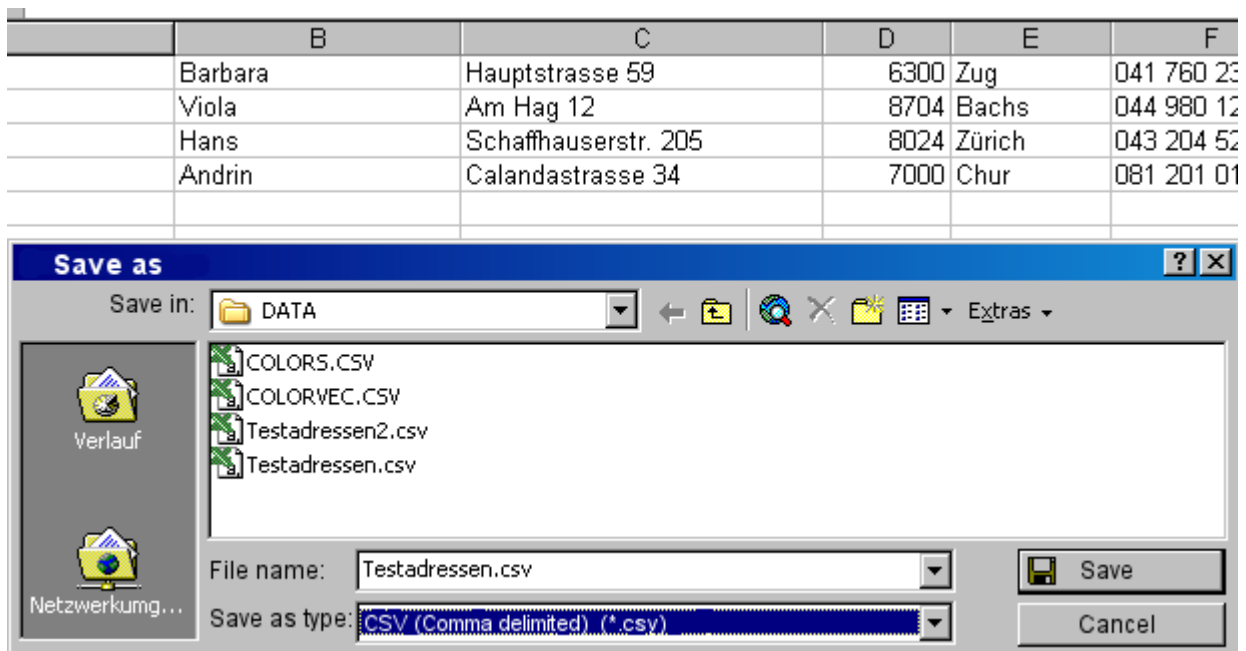
In this case the LOG.CSV on the CompactFlash Card gets a new data line.

#### Example of an adress database:

```
Aregger;Barbara;Hauptstrasse 59;6300;Zug;041 760 23 99;barbara.aregger@gmx.ch
Amsler;Viola;Am Hag 12;8704;Bachs;044 980 12 45;hagivi@bluewin.ch
Berger;Hans;Schaffhauserstr. 205;8024;Züich;043 204 52 67;haberzu@green.ch
Caviezel;Andrin;Calandastrasse 34;7000;Chur;081 201 01 01;canrin@tiscali.ch
Danuser;Carl;Rheinstrasse 174;7000;Chur;081 202 30 69;Carl.Danuser@bluewin.ch
```

The CSV-file enables the data exchange from and to the PC (EXCEL).

TIP: Store the EXCEL file under the data type CSV (separator char) \*.CSV



Settings to store a CSV-files in EXCEL.



## Class CSV

Analyse table

You only can apply the methods of class CSV to a CSV-file after its content is read into a CSV-string (e.g. `MatchFile.$`), cf. p. 95.

**CSV.Analyze\_String(VarStr:CSV\_File)**

The method analyses a CSV-String. It determines the number of Lines and Columns etc. This method is automatically executed when the content of a CSV-file is being read into a CSV-String. If there have been made changes to the CSV-String the method needs to be executed prior to applying further CSV-methods.

```
CSV.Analyze_String(MatchFile.$)
```

**CSV.GetMax\_Lines(VarInt:Lines,VarStr:CSV\_File)**

The method returns the number of lines that are in the CSV-file.

```
CSV.GetMax_Lines(MatchFile_LineNoMax.I,MatchFile.$)
```

Hint 1:

If the CSV has more lines than the method `CSV.GetMax_Lines` returns, there is mostly a not orthogonal structure of the CSV the reason for this. Specially when the file is edited with an editor this is often the problem. Make sure that every line has the correct number of columns.

Hint 2:

The method may return a strange number of Lines when the CSV-String consists of just one line together with a CRLF at its end. Therefore beware that you avoid this combination.

**CSV.GetMax\_Columns(VarInt:Columns,VarStr:CSV\_File)**

The method returns the number of columns in a CSV file or CSV string. The number of columns in a single-line-CSV-string may also be computed (cf. Example below).

```
CSV.GetMax_Columns(MatchFile_ColumnsMax.I,MatchFile.$)
```

```
TestString2 := 'monday;tuesday;wednesday;thursday;friday;saturday;sunday\r\n')  
CSV.GetMax_Columns(ColumnsMax.I,TestString2.$)
```

Returns ColumnsMax = 7 for the seven week days in the string

```
CSV.DataFieldLength(OutputFieldLength.I,CSV_String.$,Line  
Number.I,ColumnNumber.I)
```

The method returns the number of characters of a field addressed by line and column number.

```
CSV.DataFieldLength(OutputFieldLength.I,MatchFile.$,line.I,column.I)
```

If **MatchFile.\$** contains the address table of 94 and **line.I** = 2 and **column.I** = 2 (field content is 'Viola') the method would return **OutputInteger.I** = 5 .

### Retrieve data from the table

You only can apply the methods of class CSV to a CSV-file after its content is read into a CSV-string (e.g. **MatchFile.\$**), cf. p. 95.

```
CSV.Get_Integer(OutputInteger.I,CSV_String.$,LineNumber.  
I,ColumnNumber.I)
```

The method returns an Integer of the field addressed by line and column number.

In the example of our database (cf. p. 94) the third column is the column of postal code.

```
CSV.Get_Integer(OutputInteger.I,MatchFile.$,line.I,column.I)
```

If **MatchFile.\$** contains the address table of 94 and **line.I** = 3 and **column.I** = 4 the method would return **OutputInteger.I** = 8024.

```
CSV.Get_Long(OutputLong.L,CSV_String.$,LineNumber.I,Colu  
mnNumber.I)
```

The method returns a Long of the field addressed by line and column number.

In the example of our database (cf. p. 94) in the third column is the column of postal code.

```
CSV.Get_Integer(OutputLong.L,MatchFile.$,line.I,column.I)
```

If **MatchFile.\$** contains the address table of 94 and **line.I** = 3 and **column.I** = 4 the method would return **OutputLong.L** = 8024.

```
CSV.Get_LongDeci(OutputLongDeci.L,CSV_String.$,LineNumber.I,ColumnNumber.I,DecimalPlaces.I)
```

The method returns a Long of the field addressed by line and column number.

In the example of our database (cf. p. 94) in the third column is the column of postal code.

```
CSV.Get_LongDeci(MyLongDeci.L,CSV_String.$,LineNumber.I,ColumnNumber.I,DecimalPlaces.I)
```

If the value in the field determined by line and column number is 1.23 and **DecimalPlaces.I** = 3 the method would return **OutputLongDeci.L** = 1230. At the subsequent use of this Long you need to always be aware of its number of decimal places. So if you want to convert it to a string in order to display it on a label you would use following method:

```
Str.Cvt_LongDeci(Label_Text.$,MyLongDeci.L,IntegerDigits.I,DecimalPlaces.I)
```

while **DecimalPlaces.I** = 3 , as you determined before.

```
CSV.Get_HighColor(OutputColor.I,CSV_String.$,LineNumber.I,ColumnNumber.I)
```

The method returns a color value of the field addressed by line and column number. The value in the table field must be in HEX (3x8 bit, e.g. #25AAFF) else the method is not executed. The FOX embedded computer converts this HEX value into the closest color value of the 3x5bit color space (e.g. #7EA4) and saves it as integerHEX (e.g. **OutputColor.I** = 7EA4)

```
CSV.Get_ByteHex(OutputByteHex.I,CSV_String.$,LineNumber.I,ColumnNumber.I)
```

The method returns a ByteHEX value of the field addressed by line and column number. The value in the table field must be in ByteHEX (1x8bit, e.g. AF). If there is a number of more than 2 digits the method would read the first two digits (e.g. 1E3F would be read as 1E). In cases where no ByteHEX value can be read the method is not executed.

```
CSV.Get_WordHex(OutputWordHex.I,CSV_String.$,LineNumber.I,ColumnNumber.I)
```

The method returns a WordHEX value of the field addressed by line and column number. The value in the table field must be in WordHEX (2x8bit, e.g. 1E3F). If there is a number of more than 4 digits the method would read the first four digits (e.g. 1A2E22 would be read as 1A2E). In cases where no WordHEX value can be read the method is not executed.

```
CSV.Get_LongHex(OutputLongHex.I,CSV_String.$,LineNumber.I,ColumnNumber.I)
```

The method returns a LongHEX value of the field addressed by line and column number. The value in the table field must be in WordHEX (4x8bit, e.g. 2F04AA3B). If there is a number of more than 8 digits the method would read the first four digits (e.g. 1A2E2234FF would be read as 1A2E2234). In cases where no LongHEX value can be read the method is not executed.

```
CSV.Get_String(OutputString.$,CSV_String.$,LineNumber.I,ColumnNumber.I)
```

The method returns the string that is addressed by line and column number. In the example of our database (cf. p. 94) in the fifth column is the column of the cities.

```
CSV.Get_String(OutputString.$,MatchFile.$,line.I,column.I)
```

when the MatchFile is the address table, the method returns

```
CSV.Get_String(OutputString.$,MatchFile.$,5,2)
```

as OutputString we get the city in the 5<sup>th</sup> column of the 2<sup>nd</sup> line : **OutputString.\$** = 'Bachs'

### Search data in the table

You only can apply the methods of class CSV to a CSV-file after its content is read into a CSV-string (e.g. **MatchFile.\$**), cf. p. 95.

```
CSV.Find_in_Column(VarStr:CSV-String,VarInt:StartLine,VarInt:ColumnNumber,VarStr:MatchString,VarInt:LineNumber,VarStr:DataLine)
```

The method searches for the MatchString, in the indicated column. In our example of the address table (cf. p. 94) you can search for a city or for a first name depending which column you selected. The method searches from the StartLine until the first entry matches. If an entry is found the register **eI.Boolean** is set to **true** (= 1), otherwise to **false** (= 0). The match string may consist of a few starting characters. The first match is reported as line number and also the whole data record to avoid new access to the CSV-file.

The method converts the characters to small letters to avoid problems with small and capital letters.

```
CSV.Find_in_Column(MatchFile.$,1,3,'Hag',LineNo.I,TestString.$)
```

#### Input:

- CSV-String (e.g. **MatchFile.\$**) in which the content of the CSV-File is temporarily stored.
- Start line: Line of the CSV-Table, from where I want to start my top-down search (e.g. Line 1).

- Column number: Column in which the search should be performed (e.g. in the third Column)
- Match string: Search term which I want to find in the column (e.g. "Hag").

**Input:**

- Line Number: Line in which my match string has been found (e.g. "Hag" is part of line number 2 , see database example p.94) in which the content of the CSV-File is temporarily stored. If my MatchFile is the address table (cf. p. 94), "Hag" is part of line 2 and the method returns value 2 as **LineNo.I**.
- Data line: If my MatchFile is the address table (cf. p. 94), the method returns the entire line 2 (the third line) as **TestString.\$**:

```
Amsler;Viola;Am Hag 12;8704;Bachs;044 980 12 45;hagivi@bluewin.ch
```

With the methods **CSV.Get\_Integer** / **CSV.Get\_Long** / **CSV.Get\_String** etc. the specific field data can be accessed.

**Replace data in the table**

You only can apply the methods of class CSV to a CSV-file after its content is read into a CSV-string (e.g. **MatchFile.\$**), cf. p. 95.

```
CSV.Put_String (VarStr:String,VarStr:CSV-String, VarInt:
LineNumber,VarInt:ColumnNumber)
```

The method puts a string into the Cell specified by line and column number.

**Example:**

```
CSV.Put_String('Catharine',MatchFile.$,2,1)
```

If the MatchFile is the address table (cf. p. 94), the method replaces "Viola" with "Catherine".

## ■ Revision History

- 2006: - First version in German called 'eigerScript eVM Commands'. RFO
- 19.05.2010: - First release of the English version called 'eigerScript eVM Software Manual' (V1.10). RFO
- 18.06.2010: - Several corrections concerning orthographies, grammatik and layout.
- List of registers (cf. p.6). CA
- 28.06.2010: - Passage in the Chapter of Class HotSpots added about the constants of the Register **eI.HS\_EventType**.
- List of registers expanded (cf. p.6). CA
- 12.08.2010: - Translation of german remnants in text and images, which have not jet been translated. CA
- 04.10.2010: - Addition: Draw.Line, Draw.Circle and Draw.Ellipse. GmD
- 05.11.2010: - Various corrections concerning layout and spelling.
- Some new examples added. CA
- 10.12.2010: - Addition: Class Time (cf. p.54) and Str.Time and Str.Date (cf. p.22). CA
- 13.01.2011: - Addition: Remarks about Addition among strings (cf. p.13ff). CA
- 13.01.2011: - Addition: Remark concerning Loop - Endloop among strings (cf. p. 88). CA
- 11.05.2011: - Addition: Description of **Display.CopyWindow()** (cf. p. 11),
- Addition: Description of **Str.GetPosition\_by\_Char()** (cf. p. 21). CA
- 25.05.2011: - Addition: Example for installing a HotSpot with **eI.HotSpotTag** (cf. p.49). CA
- 22.07.2011: - Addition: Description of **Math.MOD\_Integer()** (cf. p.39). CA
- 02.09.2011: - Addition: Description of **Binary.BGET\_Integer()** (cf. p.40). CA
- 24.10.2011: - Corrections in Binary-Class. CA
- 31.10.2011: - Addition: Completion of Class CSV (cf. p. 95ff). CA
- 31.10.2011: - Additions: Class CSV (cf. p. 95ff). CA
- 31.10.2011: - Addition of Hint 2 at **CSV.GetMax\_Lines()** (cf. p.95). CA
- 06.11.2013: - Possibility to turn DebugMode off and on (cf. p.92). CA

## ■ Support

Have you got questions concerning programming language eigerScript or products of eigergraphics? We hope you'll find the answer at [www.eigergraphics.com](http://www.eigergraphics.com). Otherwise don't hesitate contacting us:

Phone: +41 41 754 50 10

Email: [eiger@s-tec.ch](mailto:eiger@s-tec.ch)