



**FOX** embedded computers 

*the canny swiss solution*

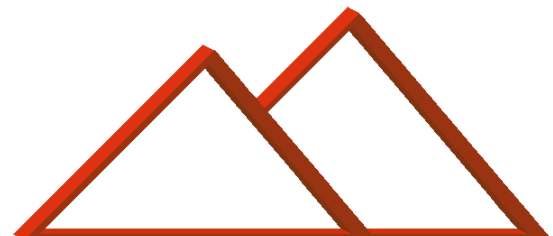
# eigerScript eVM Commands

eiger system Test

Version 1.0, 03.02.2006

update 02.12.2009 0\$70

eigergraphics eigerScript für FOX-Computer



**S-TEC electronics AG**

## Inhalt

Inhalt.....	2
eigerScript .....	3
Kurzbeschreibung .....	3
Views .....	3
Datentypen.....	3
Konstanten.....	4
Variablen.....	5
Register.....	5
Übersicht SystemTests.....	6
Class eigerVideoEngine .....	7
Class Display .....	9
Class File.....	11
Class String .....	12
Class Value .....	21
Class Label.....	23
Class Draw .....	24
Class Math.....	25
Class Binary .....	29
Class HotSpot.....	31
Class HotKey.....	38
Class Timer .....	40
Timer: Eventsektion .....	41
Timer: Ticsektion.....	42
Serielle Schnittstellen /Serial Server.....	44
Class Colors .....	47
Farben verändern .....	47
Farbenwerte setzen/holen.....	52
Software Stack .....	53
Class InOut.....	56
Class Fill/Load/Transfer.....	57
Class Load.....	58
Class Transfer .....	60
Programmfluss-Steuerung.....	61
Fehlerbehandlung.....	62
Debugger.....	63
Arbeiten mit CSV-Files .....	64
Daten aus Tabellen lesen .....	66
Daten in Tabellen suchen .....	66



## eigerScript

### Kurzbeschreibung

eigerScript ist die Programmiersprache für die eiger virtual machine eVM, mit der das eiger System programmiert werden kann. Es stehen in eigerScript leistungsfähige Methoden zur Verfügung aus denen sich ein Programm zusammensetzt. Die Entwicklungsumgebung um eigerScript zu programmieren heisst eigerStudio. eigerStudio ist eine moderne IDE (integrated development environment) mit SyntaxColoring und eigerTIP, der eingebauten Hilfe, mit der Befehle schnell aus Listen ausgewählt werden können. Der eigerCompiler erzeugt den ByteCode (\*.EVI), der von der eVM ausgeführt wird.

Die Befehle kann man sich als Aufrufe von Methoden aus Bibliotheken vorstellen. Ein solcher Befehl besteht zum Beispiel darin, ein Bildfile auf der CompactFlash-Karte zu suchen und es auf dem Display darzustellen. Je mächtiger die Instruktionen sind, desto wirkungsvoller ist das Konzept der virtuellen Maschine, weil die Dekodierzeit und die Parameterladezeit des Befehls gegenüber der eigentlichen Ausführungszeit des Befehls immer mehr in den Hintergrund tritt. Durch die strikte Trennung von Programmcode und Betriebssystem mit virtueller Maschine kann das System gut getestet werden und ist gut wartbar. Das eiger Betriebssystem ist sehr effizient in Assembler programmiert. Dadurch können auf Mikrocontrollern äusserst reaktionsschnelle GUI's realisiert werden. Viele Touch-Applikationen können mit kostengünstiger, energiesparender, langzeitverfügbaren Hardware ausgeführt werden und die Touchscreen-Technologie wird für neue Anwendungen interessant.

### Views

Die FOX embedded Computers benötigen einen relativ kleiner Hauptspeicher, weil die einzelnen Programmteile, in eigerScript Views genannt, immer neu vom CompactFlash geladen werden. Dadurch ergeben sich wesentliche Vorteile gegenüber herkömmlichen Systemen: Ähnlich wie beim WEB müssen nicht alle Daten schon im Hauptspeicher sein. Es können mehrere Programmierer gleichzeitig an einem Projekt arbeiten, da jeder eine vollständige View erstellen kann, die bei der Integration zusammengefasst werden.

Durch Hinzufügen von zusätzlichen Views wird das System nicht langsamer: jeder Viewwechsel dauert gleich kurz. Durch die Verwendung von grossen CFC's sind hunderte von verschiedenen Views möglich. Durch Austausch des CFC's ist das Programm gewechselt: Der neue Code kann per email versandt werden, die Files auf CFC geladen werden und die CFC im FOX eingesteckt werden.

### Datentypen

In einem Computersystem müssen die Daten, die durch Variablen abgebildet sind, in geeignete Datentypen abgelegt werden. Wichtig ist dabei, dass man sich beim Aussuchen einer Variablen Gedanken macht über den Wertebereich einer Zahl und darüber, ob es sich um eine Zahl oder

um ein Wort (String) handelt. Weiter muss man beachten, mit welcher Genauigkeit die Zahl dargestellt werden soll. Es sollte immer der bestgeeignete Datentyp verwendet werden, damit das System optimal funktioniert.

Beispiel: Schulnoten haben einen Wertebereich von 1 bis 6. Werden Achtelnoten erteilt oder ein Notendurchschnitt berechnet, ist es nötig, die Noten mit einer Genauigkeit von 1/1000 darstellen zu können. Die Schulnoten können mit dem Datentyp Single sehr gut dargestellt werden.

Ein Zähler zählt die in ein Parkhaus eingefahrenen Fahrzeuge. Dazu kann je nach Zählperiode ein INTEGER oder ein LONG Datentyp verwendet werden. Single wäre wohl wenig sinnvoll, da kaum halbe Fahrzeuge einfahren.

eigerScript kennt folgende Datentypen:

- |           |                                       |
|-----------|---------------------------------------|
| - Byte    | Bereich 0..255, ASCII Characters      |
| - Integer | Bereich 0..65535, oder -32768..32767  |
| - Long    | Bereich 0..2147483647 -2147483648     |
| - Single  | Single precision floatingpoint Number |
| - String  | Zeichenketten, bis zu 65'000 Zeichen  |

VarInt	VarInt kann sein:	Integerzahl Register Variable	beginnend mit	eI.
VarLong	VarLong kann sein:	Long Integer Zahl Register Variable	beginnend mit	eL.
VarSingle	VarSingle kann sein:	single Zahl Register Variable	beginnend mit	eS.
VarString	VarString kann sein:	String, direkt angegeben String String Register		e\$.

## Konstanten

Konstanten dienen zur Definition von Werten. Im eigerSystem sind zum Beispiel viele Farben mit der im WebDesign verwendeten Namen als Konstanten vordefiniert. Diese Konstanten werden in

einem Includefile in die View eingeschlossen. Es können auch eigene Konstanten definiert werden.

Beispiel: In einem Projekt werden Buttons verwendet. Anstelle die Grösse direkt in Zahlenwerten im Programm einzusetzen, werden Konstanten definiert und die Namen der Konstanten eingesetzt. Dies hat den Vorteil, dass Änderungen schnell ausgeführt werden können.

```
CONST      ButtonWidth = 120           ; Breite des Button
CONST      ButtonHeight = 36          ; Höhe des Button

Load.Width_Height(ButtonWidth,ButtonHeight)      ; Breite und Höhe laden
```

Wenn auf einer Seite mehrere gleichartige Buttons gezeichnet werden sollen, werden die Konstanten verwendet. Soll nun die Grösse der Buttons geändert werden, muss nur der Konstantenwert angepasst werden und das Programm neu kompiliert werden.

Die Konstanten können verschiedenen Datentypen zugeordnet werden. Immer wenn eine Konstante geladen werden soll, evaluiert der Compiler den Wert und weist sie der Variablen zu.

```
CONST      MyInteger      = 50           ; MyInteger hat den Startwert 50
CONST      MyLong         = 150000      ; MyLong hat den Startwert 150000
CONST      MySingle       = 3.1415     ; MySingle hat den Startwert 3.1415
```

## Variablen

Die Variablen sind Speicherzellen, die mit einem Wert vorbelegt werden können und dann im Programm verwendet werden können. Die Definition einer Integer-Variablen sieht so aus:

```
INTEGER    MyInteger      = 50           ; MyInteger hat den Startwert 50
LONG       MyLong         = 150000      ; MyLong hat den Startwert 150000
SINGLE     MySingle       = 3.1415     ; MySingle hat den Startwert 3.1415
```

## Register

Die eVM verfügt über verschiedene RegisterVariablen. Die Register haben oft eine vordefinierte Funktion für die Übergabe von Werten von und zur virtuellen Maschine. Beispielsweise empfangen die Register el.Pos\_X1 und el.Pos\_Y1 ein Koordinatenpaar für das Zeichnen von Grafiken. Es gibt Instruktionen, die Register lesen und solche, die Register beschreiben und den Inhalt verändern. Die Register behalten ihre Werte auch bei einem Viewwechsel. Die Register sind auch streng nach Datentypen getrennt, damit keine unerlaubten Zugriffe möglich sind. Viele Register stehen in direktem Zusammenhang mit Instruktionen, es gibt aber auch Register, die frei verwendet werden können und als Übergabestellen in Subroutinen dienen können.

## Übersicht SystemTests

### **TG01 Toggle-Switches mit Bildern**

Vier Toggle-Switches werden angezeigt. Durch Drücken auf die Hotspots schalten sie um.

### **TG03 Bildschirmschoner**

Bildschirmschoner mit EVEanna und transparentem Sprite, der sich verschiebt. Das System wird komplett ausgelastet. Für einen Sprite muss zuerst ein transparentes Rechteck gezeichnet werden. Darauf wird der Sprite geladen, d. h. ein Bild 160 x 160 mit transparentem Rand. Dieses Bild, das im nicht sichtbaren Teil des AVR gezeichnet wird, ist die Quelle für den Sprite.

In der Schleife wird der Bildausschnitt, auf den der Sprite neu zu liegen kommt, in einen unsichtbaren Arbeitsplatz im AVR kopiert (160x160=25600 Pixel => 51kB). Dann wird der Sprite von der Quelle auf den Arbeitsplatz kopiert (51kB). Das Bild ist nun kombiniert. Im dritten Schritt wird das kombinierte Bild vom Arbeitsplatz an die richtige Stelle im Bild in den RVR kopiert (51kB). Wenn sich der Sprite um einen Pixel verschiebt, sind 153kB verschoben worden.

### **TG04 Farbverlauf zeichnen**

Der Test ist die Übersetzung einer alten View aus dem Assembler-Projekt. Es wird Text ausgegeben und Farbverläufe werden gezeichnet.

### **TG09 Hotkey Inputs**

Der Test zeigt, wie HotKeys verwendet werden können.

### **TG11 Test mit Farben**

Der Test soll die Farben, Label und Borderstyles etc. zeigen.

## Class eigerVideoEngine

Diese Funktionen greifen direct auf die eigerVideoEngine EVE zu. Mit diesen Funktionen wird das Verhalten der Grafikmaschine gesteuert.

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

Das **EVE.Init**-Command schreibt die Helligkeit, die im **eI.Brighthness**-Register gespeichert ist ins entsprechende Register in der EVE. So wird die Helligkeit gesetzt. Beim Reset des FOX wird das **eI.Brighthness**-Register auf den Wert **0x000F** gesetzt, was maximale Helligkeit bedeutet. Achtung: wenn das **eI.Brighthness**-Register mit dem Wert 0 geladen wird, ist das Backlight komplett ausgeschaltet. Wenn das Backlight ein CCFL (FL-Röhren) ist, sollte die Beleuchtung nicht oft ein- und ausgeschaltet werden, weil das der Lebensdauer der Röhren schadet.

Die horizontale Anzeigeposition wird auf 0 gebracht und das Register **eI.XSTART** mit 0 geladen.

Es wird der Schreibmodus ohne Transparenz und nicht invertiert geladen. Der Transparenzmodus wird auf „Transparenz abwehren“ eingestellt und das Schreiben in den AVR und den RVR-Videospeicher ermöglicht.

Nach der Initialisierung ist die Grafikmaschine bereit, neue Befehle entgegenzunehmen.

Es wird empfohlen, den Befehl **EVE\_Init()** am Anfang von jeder View aufzurufen. Damit wird verhindert, dass Überraschungen gibt, wenn eine andere View eine abweichende Einstellung der Grafik benötigte und diese nicht mehr zurückstellte.

### **EVE.Load\_XSTART\_Imm(i)**

Der Befehl **EVE.Load\_XSTART\_Imm(i)** lädt den Immediate-Wert in das XSTART-Register der EVE. Mit diesem Register wird eine horizontale Verschiebung der Anzeige erreicht.

### **EVE.ReLoad\_XSTART()**

Die Methode **EVE.ReLoad\_XSTART()** lädt den Wert des **eI.XSTART**-Registers in das XSTART-Register der EVE. Mit diesem Register wird eine horizontale Verschiebung der Anzeige erreicht. Das **eI.XSTART**-Register muss zuerst vorgeladen werden.

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

Der Methode **EVE.WriteEnable\_AVR()** ermöglicht das Schreiben ins AVR-Video-RAM, das zurückgelesen werden kann. Gleichzeitig wird das Schreiben ins RVR-Video-RAM gesperrt. Diese Methode kann dazu verwendet werden, eine Grafik im Hintergrund vorzubereiten und nach dem Zeichnen den gesamten Inhalt durch die **Display.Show()**-Methode anzuzeigen.

**EVE.WriteEnable\_RVR()**

Die Methode **EVE.WriteEnable\_RVR()** ermöglicht das Schreiben ins RVR-Video-RAM, das auf dem Display angezeigt wird. Gleichzeitig wird das Schreiben ins AVR-Video-RAM gesperrt. Dieser Befehl kann dazu verwendet werden, Warnungen oder ein POP-up anzuzeigen, ohne, dass der rücklesbare Video-Speicher geändert wird. Anschliessend kann vom AVR in den RVR-Speicher geschrieben werden, um den ursprünglichen Bildinhalt wieder zu erhalten.

Eine verwandte Methode ist `Display.Prepare()`.

Mit `Display.Show()` oder `Display.ShowWindow()` kann der Inhalt des AVR-VideoMemory auf dem RVR-VideoMemory sichtbar gemacht werden.

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

Mit dem Befehl **EVE.WriteEnable\_AVR\_RVR()** werden beide Video-RAM-Ebenen gleichzeitig beschrieben. Dies ist die Normaleinstellung nach dem Befehl **EVE.Init**. Die EVE schreibt gleichzeitig in beide Video-RAM, so dass kein Zeitverlust entsteht. Es entsteht ein elektronisches „Original“ und eine Kopie davon. In diesem Zusammenhang steht der Begriff "doublebuffering".

**EVE.Load\_Transparence()**

Mit dem Befehl **EVE.Load\_Transparence()** wird die EVE so konfiguriert, dass die transparente Farbe in den Video-Speicher geladen wird. Beim Kopieren werden transparente Pixel der Quelle nicht mitkopiert, so dass auch nicht rechteckige Formen als „Sprite“ kopiert werden können. Normalerweise ist dieser Modus ausgeschaltet.

**EVE.Process\_Transparence()**

Mit dem Befehl **EVE.Process\_Transparence()** wird die EVE so konfiguriert, dass die transparente Farbe **nicht** in den Video-Speicher geladen wird. Wenn ein transparenter Pixel in die EVE geschrieben wird, werden nur die Pointer entsprechend der programmierten Richtung verschoben. Dies ist die Normaleinstellung nach dem Befehl **EVE.Init**. Mit dieser Einstellung kann eine Schrift mit transparentem Hintergrund auf ein Bild ausgegeben werden.

## Class Display

Mit den Methoden der Klasse Display werden Display-spezifische Einstellungen verändert.

### Display.Clear()

Mit der Methode Display.Clear wird der Bildschirm gelöscht. Die Löschfarbe steht im Register `eI.DisplayColor`. Soll die Löschfarbe geändert werden, kann durch schreiben einer neuen Farbe ins `eI.DisplayColor` Register die Farbe geändert werden. Nach einem Reset des Systems wird das Register `eI.DisplayColor` mit der Farbe silver (hellgrau) vorgeladen.

Beispiel:     `Display.Clear()`     ; Display wird gelöscht

### Display.ClearColor(VarInt:Color

Mit der Methode Display.ClearColor wird der Bildschirm gelöscht. Die Löschfarbe steht als Parameter in den Klammern. Die Löschfarbe kann ein anderes Register, eine Konstante oder eine Variable sein.

Beispiel:     `Display.ClearColor(blue)` ; Display wird gelöscht (mit blau gefüllt).

### Display.SetBrightness()

### Display.Prepare()

Der Bildschirmspeicher hat zwei übereinander liegende Speicherbereiche: das AVR (accessible video RAM) und das RVR (refresh video RAM). Das AVR kann vom Controller gelesen und beschrieben werden, während das RVR nur beschrieben werden kann und vom der Refreshlogik zur Bildschirmanzeige gebraucht wird.

Durch den Aufruf der Methode `Display.Prepare()` wird der Schreibzugriff auf das RVR verhindert und der neue Bildschirminhalt kann im AVR vorbereitet werden. Wenn der Seitenaufbau abgeschlossen ist, kann mit der Methode `Display.Show()` der Inhalt des AVR ins RVR kopiert werden und damit sichtbar gemacht werden.

Das Register `eI.Display_WriteMode` wird auf `Write_To_AVR` gesetzt.

### Display.Direct()

Der Bildschirmspeicher hat zwei übereinander liegende Speicherbereiche: das AVR (accessible video RAM) und das RVR (refresh video RAM). Das AVR kann vom Controller gelesen und beschrieben werden, während das RVR nur beschrieben werden kann und vom der Refreshlogik zur Bildschirmanzeige gebraucht wird.

Durch den Aufruf der Methode `Display.Direct()` wird der Schreibzugriff auf das AVR verhindert. Die Ausgaben werden direkt sichtbar. Diese Methode wird benutzt um ein POPUP darzustellen ohne den Inhalt im AVR zu verändern. Mit der Methode `Display.Show()` wird der Inhalt des AVR ins RVR kopiert werden und damit der alte Zustand wiederhergestellt. Effektiv sieht es aus, als würde das POPUP gelöscht.

Speziell zu erwähnen ist, dass ein Subroutinenaufruf den Zustand wie er vor dem Aufruf war rettet.

Eine weitere Eigenschaft der Methode ist, dass mit dem Register `eI.Display_DumpMode` eingestellt werden kann, ob die Methode normal arbeitet (`eI.Display_DumpMode = false`), d. h. in RVR schreibt oder ob trotzdem auch ins AVR geschrieben werden soll (`eI.Display_DumpMode = true`). Wenn für eine Dokumentation ein Screenshot erstellt werden soll, wird dieses Feature benötigt.

Das Register `eI.Display_WriteMode` wird auf `Write_To_RVR` gesetzt.

## Display.Show()

Mit der Methode `Display.Show()` wird der Inhalt des AVR ins RVR kopiert und damit sichtbar gemacht. Diese Methode wird im Zusammenhang mit den Methoden `Display.Prepare()` oder `Display.Direct()` verwendet. Die Beschreibung findet sich dort.

Die gesamte Bildschirmgröße wird kopiert. Die verwandte Methode `Display.ShowWindow()` kopiert nur einen Ausschnitt.

## Display.ShowWindow()

Mit der Methode `Display.ShowWindow()` wird ein Ausschnitt des AVR ins RVR kopiert und damit sichtbar gemacht. Diese Methode wird im Zusammenhang mit der Methode `Display.Prepare()` verwendet. Die Beschreibung findet sich dort. Die Parameter stehen in den folgenden Registern:

<code>eI.Pos_X1</code>	linke obere Ecke
<code>eI.Pos_Y1</code>	rechte obere Ecke
<code>eI.Offset_X</code>	Offset X
<code>eI.Offset_Y</code>	Offset Y
<code>eI.Width</code>	Breite
<code>eI.Height</code>	Höhe

Tip: Wenn ein Label periodisch geschrieben wird, kann diese Methode verwendet werden, um ein Flackern auf dem Bildschirm zu verhindern.

Diese Methode wird im Zusammenhang mit der Methode `Display.Direct()` verwendet werden, um einen ursprünglichen Bildschirminhalt wieder herzustellen.

`Display.RestoreWriteMode()`

## Class File

Mit den Methoden der Klasse File wird mithilfe des eigerFileSystems auf die CompactFlashCard zugegriffen.

Es stehen Methoden zur Verfügung mit denen ein GrafikFile dargestellt werden kann oder ein Comma Separated Value (\*.CSV) eingelesen werden kann. Dieses File kann von Excel erzeugt werden. Damit ist es möglich, Tabellenwerte ins eiger-System zu übertragen. Weitere Informationen finden sich in der Dokumentation zum CSV-File

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

Mit der Methode `File.Read_EGI()` wird ein Bild im \*.EGI-Format gelesen und direkt in den Videospeicher geschrieben. Der Befehl wurde auf Geschwindigkeit optimiert. Das eiger FileSystem arbeitet direkt mit der Anzeigeroutine zusammen und leitet die Bilddaten zur Dekompression direkt an die EVE. Für die Anzeige eines Bildes müssen folgende Parameter gesetzt werden:

<code>eI.Pos_X1</code>	linke obere Ecke
<code>eI.Pos_Y1</code>	rechte obere Ecke

Die Methode liefert die Höhe und die Breite des Bildes in folgenden Registern zurück.

<code>eI.Width</code>	Breite
<code>eI.Height</code>	Höhe

**Tip:** Die Rückgabeparameter `eI.Width` und `eI.Height` können direkt verwendet werden, um mit der Methode `Draw.Border()` einen Rahmen um das Bild zu zeichnen.

Der FileNameString kann eine der nachfolgenden Formen haben:

```
File.Read_EGI('C:\TG12\IMAGE_A.EGI')           ; Ausführlich
File.Read_EGI('IMAGE_B.EGI')                   ; Kurzform: IMAGE_B ist in Projekt
File.Read_EGI('C:\PICT\IMAGE_C.EGI')           ; In anderem Ordner
File.Read_EGI('C:\TG12\PICT\IMAGE_D.EGI')      ; In UnterOrdner
```

**Fehler:** Die Methode kann folgende Fehler zurückliefern:

Rückgabeparameter `eI.Width` und `eI.Height` können direkt verwendet werden, um mit der Methode `Draw.Border()` einen Rahmen um das Bild zu zeichnen.

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

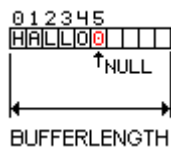
Mit der Methode `File.DeleteFile(FileName)` wird eine Datei mit dem angegebenen `FileName` gelöscht.

Beispiel:

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

## Class String

Mit den Methoden der Klasse `String` werden Zeichenketten (Strings) bearbeitet. Zeichenketten sind Buchstabenfolgen, die zu Wörtern und Sätzen zusammengefügt sind. Ein String ist in `eigerScript` immer mit `NULL = 0x00` terminiert. Das geschieht mit dem `eigerCompiler` automatisch. Ein String kann viel länger sein im Speicher, als die momentane Länge der Zeichenfolge, die mit Null terminiert ist. Diese Länge ist durch die Bufferlänge definiert und kann beim Deklarieren des Strings angegeben werden. Ein String kann bis zu 65'000 Bytes lang sein.



```
STRING [16] CSV_COLUM_1 = 'BARCODE'
```

Mit dieser Deklaration wird ein String mit Namen `CSV_COLUM_1` deklariert und dafür eine Bufferlänge von 16 ASCII-Zeichen reserviert. Der String wird gleichzeitig mit dem Wort `BARCODE` initialisiert. An diesem Beispiel sieht man den Unterschied zwischen Bufferlänge = 16 und der Stringlänge = 7.

Wird bei den Strings die Position angegeben, ist die Position des ersten Zeichens 1, das letzte Zeichen hat die Position, die der Länge entspricht.

Position: Zeichenposition in String Position von O im String 'HALLO' ist 5

### Das Stringkonzept in eigerScript

Die Strings sind die Ausgangslage für die grafische Darstellung von Ergebnissen, die textbasiert vorliegen und die Datenkommunikation mit anderen Systemen. Mit den Methoden der Klasse

String können die Zeichenketten interpretiert und manipuliert werden: Es können Zeichenketten aneinandergereiht werden, Zeichen ausgetauscht werden, Zeichenketten (Wörter) miteinander verglichen werden etc. Damit können Ausgaben formatiert und Eingaben analysiert werden. Im Gegensatz zu anderen Programmiersprachen, hat der String in eigerScript einen Header gespeichert, in dem unter anderem die Bufferlänge gespeichert ist. Die Bufferlänge gibt an, wie lang der String maximal werden darf. Damit lassen sich die Stringoperationen ohne Memoryoverflow realisieren. Die Stringoperationen liefern ein "best effort" Ergebnis. Wird ein zu langer String in einen kürzeren String kopiert, werden so viele Zeichen kopiert, wie im kurzen String Platz haben. Die Methode zeigt eine Warnung an, die ausgewertet werden kann, falls gewünscht. Dieses Vorgehen vermeidet unzählige Zeilen Code für die Fehlerbehandlung. Das System läuft aber trotzdem sicher und zuverlässig.

## **Str.Copy(VarStr:Ziel,VarStr:Quelle)**

Mit der Methode `str.Copy(ZielString,QuelleString)` wird der Quellstring in den Zielstring kopiert. Ist der Zielstring kürzer als der Quellstring, werden genau so viele Zeichen kopiert, wie im Zielstring Platz haben.

## **Str.CopySubstringWord(VarStr:Ziel,VarStr:Quelle,VarInt:Pos)**

Mit der Methode `str.CopySubstringWord(ZielString,QuelleString,Position)` wird ab der Position ein Wort des Quellstrings in den Zielstring kopiert. Ist der Zielstring kürzer als das Wort im Quellstring, werden genau so viele Zeichen kopiert, wie im Zielstring Platz haben.

## **Str.Length(VarInt:Länge,VarStr:Quelle)**

Mit der Methode `str.Length(Länge,QuelleString)` wird die Länge des Quellstrings bestimmt. Es zählen alle Zeichen bis zum Abschluss des Strings mit NULL. `str.Length(Länge,'Hallo')` liefert den Wert Länge=5.

## **Str.BufferLength(VarInt,VarStr)**

Mit der Methode `str.BufferLength(Länge,QuelleString)` wird die Bufferlänge zurückgegeben, d. h. die total mögliche Anzahl Zeichen im String.

**Str.SpaceInString(VarInt:Space,VarStr:QuellString)**

Mit der Methode `str.BufferLength(Platz,QuellString)` wird die Anzahl Zeichen zurückgegeben, die im String noch Platz haben. Der Rückgabewert entspricht der Differenz aus `BufferLength` und `StringLength`.

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

Mit der Methode `str.clear(ZielString)` wird der Zielstrings gelöscht. Das erste Zeichen im String ist NULL. Die Länge ist demzufolge 0. An der Grösse des Buffers ändert sich nichts.

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

Mit der Methode `str.AddChar(ZielString,VarInt)` wird dem Zielstring ein Zeichen angefügt. Die Methode prüft, ob im String noch Platz ist. Falls nicht, wird das Zeichen nicht angefügt.

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

Mit der Methode `str.Add_CRLF(ZielString)` wird dem Zielstring ein Zeilenabschluss (CarriageReturn- und ein LineFeed-Zeichen) angefügt. Die Methode prüft, ob im String noch Platz ist. Falls nicht noch mindestens zwei Zeichen platz haben, wird CRLF nicht angefügt.

**Str.FillUpWithChar(VarStr:ZielString,VarInt:Char)**

Mit der Methode `str.FillUpWithChar(ZielString,VarInt:Char)` wird der Zielstring mit Zeichen aufgefüllt, so dass die ganze `BufferLength` ausgefüllt ist. Diese Funktion kann gut gebraucht werden, um ein Feld mit Leerschlägen aufzufüllen.

Beispiel:

```
STRING [10]  ZielString = 'ABC'           ; ZielString hat die Zeichen ABC und ist 10 lang

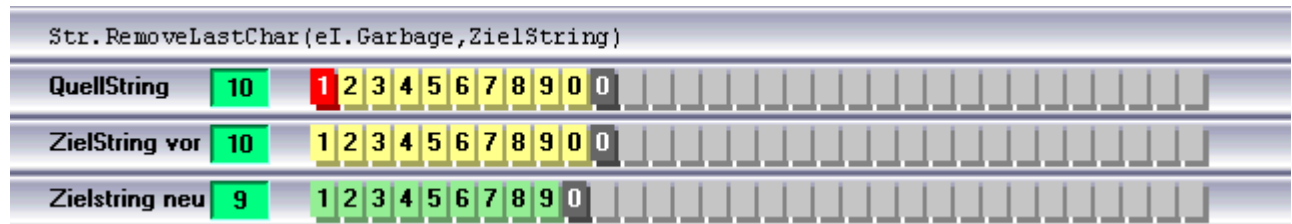
    Str.FillUpWithChar(ZielString,"*")

    ZielString hat nach dem Aufruf der Methode die Zeichen ABC*****
```

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

Mit der Methode `str.RemoveLastChar(String,VarInt)` wird das letzte Zeichen des Strings entfernt und in `VarInt` zurückgeliefert. Wird die Methode auf einen leeren String angewendet, liefert die Methode 0 zurück. Das Register `eI.Garbage` kann als Rückgabewert benutzt werden, wenn das letzte Zeichen nicht von speziellem Interesse ist. Mit der Methode kann gut eine Löschfunktion, die das letzte Zeichen löscht, implementiert werden.

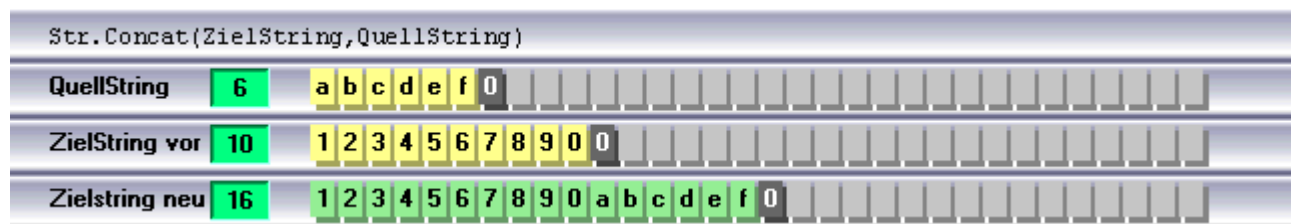
```
Str.RemoveLastChar(eI.Garbage,String)
```



## Str.Concat (VarStr:ZielString,VarStr:QuellString)

Mit der Methode `str.Concat(ZielString,QuellString)` wird dem Zielstring der Quellstring angefügt. Diese Methode wird gebraucht, um die Ausgabe in ein Label vorzubereiten. Wenn die Summe der Länge von `ZielString` und `QuellString` die Bufferlänge des Zielstrings übersteigt, wird der String abgeschnitten und nur so viele Character des Quellstrings kopiert, wie noch Platz haben. Wenn ein String als Zielstring verwendet wird, muss bei der Deklaration eine sinnvolle Länge angegeben werden.

```
Str.Concat(ZielString,QuellString) ; Strings zusammensetzen
```



### Fehler:

Es können folgende Fehler vorkommen:

- a) ZielString war schon zu Beginn voll: `eI.ErrorCode== ERR_String_NoSpaceInString`
- b) ZielString hat zu wenig Platz: `eI.ErrorCode== WAR_String_NotEnoughSpace`

## Str.UpperCase (VarStr:ZielString)

Mit der Methode `str.UpperCase(ZielString)` wird der Zielstring in Grossbuchstaben umgewandelt. Die dabei verwendete Codepage ist Latin-15 bzw. CP1252. Es werden nicht nur die Buchstaben von a..z in die Grossbuchstaben A..Z umgewandelt, sondern auch die sich in der

Codepage über 127 befindenden Zeichen in ihr Grossbuchstabe. So wird z. B. das ö in ein Ö umgewandelt. Die Methode wird gebraucht, um vor einem Stringvergleich eine Benutzereingabe in Grossbuchstaben umzuwandeln, um Probleme mit der Gross-Kleinschreibung auszuschliessen.

```
Str.UpperCase(ZielString) ; String in Grossbuchstaben wandeln
```

## Str.LowerCase(VarStr)

Mit der Methode `str.LowerCase(ZielString)` wird der Zielstring in Kleinbuchstaben umgewandelt. Die dabei verwendete Codepage ist Latin-15 bzw. CP1252. Es werden nicht nur die Buchstaben von A..Z in die Kleinbuchstaben a..z umgewandelt, sondern auch die sich in der Codepage über 127 befindenden Zeichen in ihr Kleinbuchstabe. So wird z. B. das Ö in ein ö umgewandelt. Die Methode wird gebraucht, um vor einem Stringvergleich eine Benutzereingabe in Kleinbuchstaben umzuwandeln, um Probleme mit der Gross-Kleinschreibung auszuschalten.

```
Str.LowerCase(ZielString) ; String in Kleinbuchstaben wandeln
```

## Str.Compare(VarInt,VarStr,VarStr)

Mit der Methode `str.Compare(Ergebnis,String1,String2)` werden die beiden Strings 1 und 2 miteinander zeichenweise verglichen. Wenn die Strings genau gleich sind, d. h. Zeichen genau übereinstimmen und die gleiche Länge haben, wird das Ergebnis 0 zurückgeliefert. Ansonsten wird als Ergebnis die Differenz des ersten ungleichen Zeichens zurückgeliefert. Dieses kann zur Sortierung von Strings benutzt werden. Ist das Ergebnis negativ, ist String 1 in alphabetischer Reihenfolge hinter String 2.

Diese Methode kann zum Vergleich von einem Passwort eingesetzt werden.

```
Str.Compare(Ergebnis,String,'ABC') ; String mit ABC vergleichen
```

## Str.Cvt\_ByteHex(VarStr,VarInt)

Mit der Methode `str.Cvt_ByteHex(ZielString,Integer)` werden dem String 2 ASCII-Zeichen, die den Wert der Zahl repräsentieren, angehängt.

```
Str.Cvt_ByteHex(ZielString,eI.Int_1) ; eI.Int_1 hexadezimal an ZielString  
; anhängen.
```

Bsp: 68

Tip: Für eine Zahlendarstellung 0x68 oder 68h den String vorladen oder einen String anhängen.

Verhalten im Fehlerfall: Sollte es im ZielString keinen Platz mehr haben, wird die Methode nicht ausgeführt. Damit werden Speicherüberläufe vermieden.

Anmerkung: Es wird das niederwertige Byte (LSB) zur Ausgabe verwendet.

## Str.Cvt\_WordHex(VarStr,VarInt)

Mit der Methode `str.Cvt_WordHex(ZielString,Integer)` werden dem String 4 ASCII-Zeichen, die den Wert der Zahl repräsentieren, angehängt.

```
Str.Cvt_WordHex(ZielString,eI.Int_1) ; eI.Int_1 hexadezimal an ZielString  
; anhängen
```

Bsp: FE68

Tip: Für eine Zahlendarstellung 0xFE68 oder FE68h den String vorladen oder einen String anhängen.

Verhalten im Fehlerfall: Sollte es im ZielString keinen Platz mehr haben, wird die Methode nicht ausgeführt. Damit werden Speicherüberläufe vermieden.

## Str.Cvt\_LongHex(VarStr,VarLong)

Mit der Methode `str.Cvt_LongHex(ZielString,Long)` werden dem String 8 ASCII-Zeichen, die den Wert der Zahl repräsentieren, angehängt.

```
Str.Cvt_LongHex(ZielString,MyLong) ; MyLong hexadezimal an ZielString  
; anhängen
```

Verhalten im Fehlerfall: Sollte es im ZielString keinen Platz mehr haben, wird die Methode nicht ausgeführt. Damit werden Speicherüberläufe vermieden.

## Str.Cvt\_Long(VarStr,VarLong,VarInt)

Mit der Methode `str.Cvt_Long(ZielString,Long,Stellen)` wird eine dezimale Zahlenausgabe einer Long-Zahl mit Anzahl Stellen an das Stringende ausgeführt.

```
Str.Cvt_Long(ZielString,MyLong) ; MyLong dezimal an ZielString  
; anhängen
```

Beispiel: In MyLong soll die Zahl 105'600 sein. Die Ausgabe soll mit 8 Stellen erfolgen. In ZielString steht: Counter: 105600

Das Register kann vorgeladen werden: `eI.FillChar := ""`  
 Die Ausgabe wäre dann `Counter:**105600`

Spezialfall: Wird die Anzahl Stellen als 0 angegeben, wird die Zahl unformatiert ausgegeben.

Verhalten im Fehlerfall: Sollte es im ZielString keinen Platz mehr haben, wird die Methode nicht ausgeführt. Damit werden Speicherüberläufe vermieden. Sollte die benötigte Anzahl Stellen grösser sein, als die verlangte Anzahl Stellen, wird Anzahl Stellen mit dem Char in `eI.NumericChar` ausgegeben.

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

Mit der Methode `str.Cvt_LongDeci(ZielString,Long,Feldlänge,Nachkommastellen)` wird eine dezimale Zahlenausgabe einer Long-Zahl mit Anzahl Stellen (Feldlänge) und Nachkommastellen an das Stringende ausgeführt. Die Methode dient dazu, eine Zahl, die intern in einer andern Einheit gespeichert ist, anzugeben.

AnwendungsBeispiel: Für ein Zahlungssystem wird intern mit der Basiseinheit Rappen (oder Cent) gerechnet. Der Preis soll aber in Franken, bzw. EURO angegeben werden. `MyPrice := 3260`

Die Ausgabe soll formatiert erfolgen `32.60`.

Mit der Methode LongDeci kann die Feldlänge und die Anzahl Dezimalstellen angegeben werden.

```
Str.Cvt_LongDeci(ZielString,MyPrice,6,2) ; MyPriceLong dezimal an ZielString
; anhängen mit Nachkommastellen
```

Beispiel: In MyLong soll die Zahl 3'260 sein. Die Ausgabe soll mit 6 Stellen erfolgen. In ZielString steht: `Preis: 32.60`

Das Register kann vorgeladen werden: `eI.DecimalSeparatorChar := ","`

Die Ausgabe wäre dann `Preis: 32,60`

Spezialfall: Wird die Anzahl Stellen als 0 angegeben, wird die Zahl unformatiert ausgegeben.

Verhalten im Fehlerfall: Sollte es im ZielString keinen Platz mehr haben, wird die Methode nicht ausgeführt. Damit werden Speicherüberläufe vermieden. Sollte die benötigte Anzahl Stellen grösser sein, als die verlangte Anzahl Stellen, wird Anzahl Stellen mit dem Char in `eI.NumericChar` ausgegeben.

### **Str.Cvt\_Integer (VarStr,VarInt,VarInt)**

Mit der Methode `str.Cvt_Integer(ZielString,Integer,Stellen)` wird eine dezimale Zahlenausgabe einer Integer-Zahl mit Anzahl Stellen an das Stringende ausgeführt.

```
Str.Cvt_Integer(ZielString,MyInteger,4) ; MyInteger dezimal an ZielString  
; anhängen mit 4 Stellen.
```

Es gilt das bei der Methode Str.Cvr\_Long beschriebene Verhalten.

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

Mit der Methode `str.Find(VergleichString,Startposition,Suchstring,Fundposition)` wird der Suchstring ab der Startposition im Vergleichsstring gesucht. Wenn der SuchString gefunden wurde, wird als Position das nächste Zeichen nach dem gesuchten String angegeben, wenn der String nicht gefunden wurde, wird als Rückgabewert Fundposition=0 geliefert.

### Beispiel:

```
Str.Find(InputString,1,'Booster',StringPosition)  
Jump_IF_Integer_NotZero(Booster_Event,StringPosition)
```

Das Wort Booster wird im InputString gesucht und zum Label BoosterEvent gesprungen, falls das Wort im InputString vorhanden ist.

Verhalten im Fehlerfall: Falls die Position grösser ist, als der String lang, wird Null zurückgeliefert.

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

Mit der Methode `str.Match(VarStr:VergleichString,VarInt:Startposition,VarStr:Suchstring)` wird der Suchstring ab der Startposition im Vergleichsstring verglichen. Wenn der SuchString ab der Startposition genau übereinstimmt, wird `eI.Boolean` auf true gesetzt, andernfalls auf false.

### Beispiel:

```
Str.Match('Gartenzwerg',7,'zwerg')  
IF eI.Boolean == true THEN  
; zwerg gefunden !  
ENDIF
```

Das Wort Booster wird im InputString gesucht und zum Label BoosterEvent gesprungen, falls das Wort im InputString vorhanden ist.

Verhalten im Fehlerfall: Falls die Position grösser ist, als der String lang, wird Null zurückgeliefert.

```
; String auf Übereinstimmung untersuchen ab Position ; Version 0$45 24.11.2006  
;  
Str.Match(VarStr:VergleichString,VarInt:StartPosition,VarStr:SuchString)
```

; in el.Boolean wird true oder false zurückgeliefert.

### **Str.SkipSpace(VarStr,VarInt,VarInt)**

Mit der Methode `str.skipSpace(String,StartPosition,FindPosition)` wird ab der StartPosition WhiteSpace überlesen, bis das erste Non-WhiteSpace-Zeichen im String erscheint. Wird das Stringende erreicht, oder eine Position die grösser ist als die BufferLänge, angegeben, wird als Position der Wert Null zurückgeliefert.

#### Beispiel:

```
Str.SkipSpace('    Hallo',1,Position)
```

nach dem Aufruf hat Position den Wert 5, da vier Spaces überlesen wurden und die Position auf das erste Non-WhiteSpace-Zeichen zeigt.

Die Methode `str.skipSpace` ermöglicht das überlesen von beliebig viel WhiteSpace. Als WhiteSpace-Zeichen gelten: 0x20 (Space) 0x09 (Tabulator) 0xA0 (Space).

```
Str.ASCII_to_ByteHex EQU 0x241D ; Str.ASCII_to_ByteHex
    13.10.2006
;
Str.ASCII_to_ByteHex(VarInt:Number,VarStr:InputString,VarInt:Position)
    ; EI_Status [error | success]
    ; EI_ErrorCode [NoError | eVM_EmptyString_Error |
eVM_OutOfString_Error
    ; eVM_NotInHex_Error]
```

## Class Value

Mit den Methoden der Klasse Value wird ein String in einen Datentyp umgewandelt. Mit diesen Methoden wird ein String in Zahlen gewandelt. Es stehen Methoden zur Verfügung, die Strings in Hexadezimaler Darstellung und in dezimaler Darstellung einlesen.

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

Mit der Methode `value.ByteHex(VarInt:Val,VarStr:Str,VarInt:Pos)` werden ab der StartPosition zwei ASCII-Zeichen eingelesen. Diese müssen im Bereich [0..9] | [A..F] | [a..f] liegen. In Wert wird der konvertierte Rückgabewert zurückgeliefert. Bereich [0x0000..0x00FF].

#### Beispiel:

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

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

nach dem Aufruf hat `InputValue.I` den Wert 0x00A9.

Verhalten im Fehlerfall: wenn nicht zwei aufeinanderfolgende Zeichen HEX-Zeichen sind, die Position ausserhalb des Strings zeigt oder ab der Position das Ende des Strings erreicht wird, wird ein Fehler ausgelöst. `el.Status := error` und in `el.ErrorCode` steht die Fehlernummer.

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

Mit der Methode `value.WordHex(VarInt:Val,VarStr:Str,VarInt:Pos)` werden ab der StartPosition vier ASCII-Zeichen eingelesen. Diese müssen im Bereich [0..9] | [A..F] | [a..f] liegen. In Wert wird der konvertierte Rückgabewert zurückgeliefert. Bereich [0x0000..0xFFFF].

#### Beispiel:

In `InputString.$` steht 'TEST= A95e'

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

nach dem Aufruf hat `InputValue.I` den Wert 0xA95E.

Verhalten im Fehlerfall: wenn nicht vier aufeinanderfolgende Zeichen HEX-Zeichen sind, die Position ausserhalb des Strings zeigt oder ab der Position das Ende des Strings erreicht wird, wird ein Fehler ausgelöst. `el.Status := error` und in `el.ErrorCode` steht die Fehlernummer.

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

Mit der Methode `value.LongHex(VarLong:Val,VarStr:Str,VarInt:Pos)` werden ab der StartPosition acht ASCII-Zeichen eingelesen. Diese müssen im Bereich [0..9] | [A..F] | [a..f] liegen. In Wert wird der konvertierte Rückgabewert zurückgeliefert. Bereich [0x00000000..0xFFFFFFFF].

Beispiel:

In InputString.\$ steht 'TEST= A95e76a9'

```
Value.LongHex(' InputValue.L, InputString.$, 6)
```

nach dem Aufruf hat InputValue.L den Wert 0xA95E76A9.

Verhalten im Fehlerfall: wenn nicht acht aufeinanderfolgende Zeichen HEX-Zeichen sind, die Position ausserhalb des Strings zeigt oder ab der Position das Ende des Strings erreicht wird, wird ein Fehler ausgelöst. el.Status := error und in el.ErrorCode steht die Fehlernummer.

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

Mit der Methode `value.UInteger(VarInt:Number,VarStr:Str,VarInt:Pos)` werden ab der StartPosition ASCII-Zeichen eingelesen. Diese müssen im Bereich [0..9] liegen. In Number wird der konvertierte Rückgabewert zurückgeliefert. Bereich [0..65535].

Beispiel:

In InputString.\$ steht 'TEST= 675'

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

nach dem Aufruf hat InputValue.I den Wert 675.

Verhalten im Fehlerfall: wenn die Zahl zu gross ist, die Position ausserhalb des Strings zeigt oder ab der Position das Ende des Strings erreicht wird, wird ein Fehler ausgelöst. el.Status := error und in el.ErrorCode steht die Fehlernummer.

## Class Label

Mit der Klasse Label werden Strings auf dem Display ausgegeben und Buttons gestaltet. Das Label Objekt ist sehr vielseitig und dadurch etwas komplexer. Es gibt 16 Register der eigerVM, die die Eigenschaften des Objekts beim Aufruf entgegennehmen.

### Beschreibung der eVM Register für Label

#### Position:

Die Register `eI.Pos_X1` und `eI.Pos_Y1` bestimmen die linke obere Ecke des Rechtecks, das das Label aufspannt. Zu beachten ist, dass die absolute X/Y-Position auf dem Display die Summe der Positionsregister und des X/Y-Offsets ist (siehe `eI.Offset_X1` und `eI.Offset_Y1`).

#### Grösse:

Für die Breite und Höhe des Labels werden die Register `eI.Width` und `eI.Heigth` mit den entsprechenden Pixelwerten geladen.

#### Breite:

Für die Farben des Labels werden `eI.FillColor`, `eI.LineColor`, `eI.BackColor` und `eI.TextColor` zuständig.

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

Mit dem Befehl `Label.Color(VarInt:Farbe)` werden die Farbregister für die eVM geladen.

#### Beispiel:

```
Label.Color(light_green) ; Farbe laden
```

Die Register `eI.FillColor`, `eI.LineColor` und `eI.BackColor` werden mit der als Parameter angegebenen Farbe geladen. Für die `eI.TextColor` wird die Methode `Colors.AutoColor` angewendet, was eine schwarze oder weisse Schriftfarbe ergibt, je nach vorgegebener Farbe.

#### Spezialfall:

```
Label.Color(transparent) ; Transparenz laden
```

Die Register `eI.FillColor`, `eI.LineColor` und `eI.BackColor` werden mit Transparenz geladen. Als Eingabewert für die Methode `Colors.AutoColor` wird die Farbe in `eI.DisplayColor` verwendet. Somit erscheint nur die Schrift auf dem Hintergrund. Achtung: wenn das Label ein zweites Mal mit einem anderen Text geschrieben wird, wird der erste Text nicht gelöscht.

**Label.Text (VarStr)**

Mit dem Befehl `Label.Text (VarStr)` wird ein String in ein Label ausgegeben. Dabei wird das Label gemäss den Werten in den eVM-Registern gezeichnet.

Beispiel:

```
Label.Text('Hallo')           ; Hallo in ein Label schreiben
```

Hinweis:

Mit der Methode `Fill.LabelParameter` können die eVM-Register sehr effizient aus einer Struktur geladen werden. Ist eine unterschiedliche Färbung nötig (z.B. Ausgabe eines Hinweises gelb und einer Warnung rot) kann die Methode `Label.Color` angewendet werden oder es können zwei Strukturen definiert werden, die je nach dem geladen werden

## Class Draw

Diese Klasse vereinigt die Methoden zum Zeichnen von Linien und Geometriefiguren. Es können Pixel, Linien, Rechtecke, Kreise, und Ellipsen gezeichnet werden. Die Rechtecke, Kreise und Ellipsen können gefüllt gezeichnet werden.

**Draw.Pixel()**

Mit der Methode `Draw.Pixel()` wird ein Pixel gesetzt. Die Parameter für die Methode müssen in den Registern vorgeladen sein:

```
eI.Pos_X1           : X-Position  
eI.Pos_Y1           : Y-Position  
eI.LineColor        : Farbe des Pixels
```

Beispiel:

Roten Pixel an der Adresse 120/60 zeichnen:

```
eI.Pos_X1 := 120  
eI.Pos_Y1 := 60  
eI.LineColor := red  
Draw.Pixel()           ; Pixel zeichnen
```

## Class Math

In der Klasse Math werden die mathematischen Operationen ausgeführt und Zahlen von einem Typ in den andern umgewandelt (type casting). EigerScript ist eine typenstrenge Sprache, d.h. der Benutzer muss explizit mit einer Methode Variablen eines Typs in einen anderen Typ umwandeln.

### Integermathematik

Die Integermathematik verrechnet ganze 16-Bit Zahlen vorzeichenbehaftet miteinander. Der Anwender muss sich genau überlegen, ob der Wertebereich von ca. +/- 32'000 für seine Anwendung ausreicht, sonst ist auf LONG auszuweichen.

#### **Math.ADD\_Integer (VarInt:Summe,VarInt:B,VarInt:C)**

Mit der Methode **Math.AND\_Integer (VarInt:Summe,VarInt:B,VarInt:C)** die Summe von B und C in die ZielVariable gespeichert. Die Methode kann auf zwei Arten verwendet werden

```
Math.ADD_Integer (Summe,Betrag_1,Betrag_2) ; Beträge zusammenzählen
```

soll nun noch ein dritter Betrag zur Summe addiert werden, kann die Funktion wie folgt genutzt werden:

```
Math.ADD_Integer (Summe,Summe,Betrag_3) ; Betrag3 dazuzählen
```

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

Mit der Methode **Math.MIN\_Integer (VarInt,VarInt,VarInt)** wird als Resultat die kleinere Zahl zurückgeliefert.

```
Math.MIN_Integer (Color,Grau_5B,Blue_5B) ; Minimum suchen
```

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

Mit der Methode `Math.MAX_Integer(VarInt,VarInt,VarInt)` wird als Resultat die grössere Zahl zurückgeliefert.

```
Math.MAX_Integer(Color,Grau_5B,Blue_5B) ; Maximum suchen
```

## Longwerte und Integerwerte umwandeln

**Math.LWRD\_from\_Long(VarInt:Ziel,VarLong:Quelle)**

Mit der Methode `Math.LWRD_from_Integer(VarInt,VarLong)` wird die Integer-Variable mit dem Low-Word (Bits 0..15) der Longvariablen geladen.

**Math.LWRD\_to\_Long(VarLong:Ziel,VarInt:Quelle)**

Mit der Methode `Math.LWRD_to_Integer(VarLong,VarInt)` wird die Integer-Variable als Low-Word (Bits 0..15) in der Longvariablen gespeichert. Das High-Word bleibt unverändert.

**Math.HWRD\_from\_Long(VarInt:Ziel,VarLong:Quelle)**

Mit der Methode `Math.HWRD_from_Integer(VarInt,VarLong)` wird die Integer-Variable mit dem Low-Word (Bits 16..31) der Longvariablen geladen.

**Math.HWRD\_to\_Long(VarLong:Ziel,VarInt:Quelle)**

Mit der Methode `Math.HWRD_to_Integer(VarLong,VarInt)` wird die Integer-Variable als Low-Word (Bits 16..31) in der Longvariablen gespeichert. Das Low-Word bleibt unverändert.

## Longmathematik

**Math.ADD\_Long(VarLong:Summe,VarLong:B,VarLong:C)**

Mit der Methode `Math.AND_Long(VarLong:Summe,VarLong:B,VarLong:C)` die Summe von B und C in die ZielVariable gespeichert. Die Methode kann auf zwei Arten verwendet werden

```
Math.ADD_Long(Summe, Betrag_1, Betrag_2) ; Beträge zusammenzählen
```

soll nun noch ein dritter Betrag zur Summe addiert werden, kann die Funktion wie folgt genutzt werden:

```
Math.ADD_Long(Summe, Summe, Betrag_3) ; Betrag3 dazuzählen
```

## Singlemathematik (Fließkommaarithmetik)

### Typenkonversionen

Die Typenkonversionen dienen dazu, Zahlen von einer Darstellung in die andere zu bringen und umgekehrt.

#### **Math.CVT\_UInteger\_from\_Long(VarInt:Ziel,VarLong:Quelle)**

Mit der Methode **Math.CVT\_UInteger\_from\_Long(VarInt:Integer,VarLong:Long)** wird die Long-Variable in eine unsigned-Integer-Variable umgewandelt. Da der Wertebereich der LONG-Variablen viel grösser ist, als der der Integer-Variablen, kann ein Fehler auftreten bei einer Bereichsüberschreitung. Der Bereich wird von der Methode kontrolliert, und falls eine Bereichsüberschreitung stattgefunden hat, wird der **ERR\_Math\_OutOfRange** geworfen.

typische Codesequenz:

```
eI.Status := success
Math.CVT_UInteger_from_Long(TestInteger,TestLong)
ON_ERROR_GOTO(Label) ; if Fehler
```

#### **Math.CVT\_Integer\_from\_Long(VarInt:Ziel,VarLong:Quelle)**

Mit der Methode **Math.CVT\_Integer\_from\_Long(VarInt:Integer,VarLong:Long)** wird die Long-Variable in eine Integer-Variable umgewandelt. Da der Wertebereich der LONG-Variablen viel grösser ist, als der der Integer-Variablen, kann ein Fehler auftreten bei einer Bereichsüberschreitung. Der Bereich wird von der Methode kontrolliert, und falls eine Bereichsüberschreitung stattgefunden hat, wird der **ERR\_Math\_OutOfRange** geworfen.

typische Codesequenz:

```
eI.Status := success
Math.CVT_Integer_from_Long(TestInteger,TestLong)
ON_ERROR_GOTO(Label)           ; if Fehler
```

### **Math.CVT\_Single\_from\_Integer(VarSingle,VarInt)**

Die Methode **Math.CVT\_Single\_from\_Integer(VarSingle,VarInt)** wird die Integer-Variable in eine Single-Variable umgewandelt.

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

Die Methode **Math.CVT\_Long\_from\_Single (VarLong,VarSingle)** wandelt eine Single-Variable in eine Long-Variable um. Da „Long“ ein Datentyp für ganze Zahlen ist, werden die Nachkommastellen der Single-Variable nicht mitgenommen. Auf eine Rundung wird bei dieser Methode verzichtet

Code-Beispiel:

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

z.B. für Distance.S = 1234.56 erhält Distance.L den Wert 1234; Nachkommastellen werden abgeschnitten.

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

Die Methode **Math.CVT\_Long\_from\_Single\_Round(VarLong,VarSingle)** wandelt eine Single-Variable in eine Long-Variable um. Da „Long“ ein Datentyp für ganze Zahlen ist, werden die Nachkommastellen der Single-Variable nicht mitgenommen, d.h. es wird auf- bzw. abgerundet.

Code-Beispiel:

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

z.B. für Distance.S = 1234.56 erhält Distance.L den aufgerundeten Wert 1235; ab 0.5 wird aufgerundet.

### **Math.CVT\_LongDeci\_from\_Single(VarLong,VarSingle,Stellen)**

Die Methode **Math.CVT\_LongDeci\_from\_Single(VarLong,VarSingle,Stellen)** wandelt eine Single-Variable in eine Long-Variable in der Weise um, dass auch Nachkommastellen mitgenommen werden können. Die Anzahl der Nachkommastellen wird als drittes Argument angegeben.

Code-Beispiel:

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

z.B. für Distance.S = 1234.56 erhält Distance.L den aufgerundeten Wert 1234560. Weil die Single-Variable nur 2 Nachkommastellen hat, wird in diesem Beispiel als Dritte Nachkommazahl eine Null angefügt.

## Class Binary

Diese Klasse beinhaltet die Methoden für die Binären (logischen) Operationen. Enthalten sind Bit-Operationen, Byte- und Wortweise Verknüpfungen, Schiebepfeile etc.

### Bitfunktionen mit 16-Bit Werten

Es existieren Bit-Operationen, mit denen auf einzelne Bits zugegriffen werden kann. Ein Bit kann gesetzt, gelöscht oder gewechselt werden. Die Bits dürfen in Registern oder in Variablen sein.

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

Mit der Methode `Binary.BSET_Integer(VarInt:Q,VarInt:A,VarInt:BitNummer)` wird als das mit BitNummer angegebene Bit gesetzt. Der zulässige Wertebereich für BitNummer ist 0..15.

```
Binary.BSET_Integer(eI.R00,eI.R00,15) ; Vorzeichenbit setzen
```

### Logikfunktionen mit 16-Bit Werten

Auf die Integer-Variablen können Logikfunktionen angewendet werden. Die Inputparameter werden Wortweise miteinander verrechnet und liefern als Ergebnis wieder einen 16-Bit Wert.

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

Mit der Methode `Binary.AND_Integer(VarInt,VarInt,VarInt)` wird als Resultat die logische AND Funktion der Eingangsvariablen zurückgeliefert.

```
Binary.AND_Integer(eI.BackColor,eI.BackColor,0111110000011111B) ; Grünanteil
; wemaskieren
```

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

0\$35

Mit der Methode **Binary.AND\_Integer(VarInt,VarInt,VarInt)** wird als Resultat die logische AND Funktion der Eingangsvariablen zurückgeliefert.

```
Binary.AND_Integer(eI.BackColor,eI.BackColor,0111110000011111B) ; Grünanteil maskieren
```

## Class HotSpot

Die Klasse HotSpot dient zur Erzeugung von rechteckigen Berührflächen, in eiger HotSpots genannt. Es gibt Methoden, mit denen HotSpots erzeugt werden können und auch Methoden, mit denen sie wieder abgeräumt werden können. Im Hintergrund prüft der HotSpot-EventManager des eigerOS, ob bei einem der HotSpots ein Ereignis eingetreten ist. Sollte das der Fall sein, wird der entsprechende EventHandler aufgerufen. Mögliche Ereignisse, die ausgewertet werden sind:

Enter	HotSpot wird erreicht
Leave	HotSpot wird verlassen
Down	Berührung beginnt im Hotspot
Up	Berührung endet im HotSpot

Für ein Touch-System kann im einfachen Fall nur ein EventHandler für das DOWN-Ereignis vorhanden sein, wenn auf eine Animation (z.B. Button wird beim Berühren eingedrückt) verzichtet wird.

Jeder HotSpot kann einen Integer-Wert (Tag) speichern, dieser Tag wird beim Down-Ereignis im Register `eI.HS_Tag` zurückgeliefert.

Die HotSpots werden beim Installieren in eine Tabelle eingetragen. Diese Tabelle wird beim Verlassen einer View automatisch abgeräumt, damit auf der neuen View keine alten HotSpots mehr vorhanden sind.

Die HotSpots werden mit dem Zustand "enabled" installiert. Sollte das für einen HotSpot nicht gewünscht sein, kann mit `HotSpot.Enable_By_ID` (`eI.HS_ID`) unmittelbar auf Install der HotSpot gesperrt werden.

### HotSpot Gruppen:

Jeder HotSpot gehört einer Gruppe an. Die Gruppe wird durch den Wert im Register `eI.HotSpotGroup` bei der Installation bestimmt. Es können beliebig viele HotSpots der gleichen Group angehören. Es existieren Methoden, die auf eine ganze Gruppe von HotSpots wirken, wie z.B. `HotSpot.DeInstallGroup`. Damit können HotSpots gruppenweise referenziert werden.

Die Gruppen haben eine weitere Bedeutung: Wenn ein POP-UP programmiert wird, das vollständig abgeräumt werden soll, kann mit der Methode `HotSpot.GetNextGroupNumber()` die nächste freie Gruppennummer angefordert werden. Beim Abräumen kann die Gruppe vollständig entfernt werden, ohne andere HotSpots zu tangieren.

Das Gruppenkonzept versieht die GruppenNummern mit folgenden Funktionalitäten:

0..15	BasisGruppe, wird beim Verlassen der View abgeräumt. Frei verwendbar.
16..31	BasisGruppe wird beim Verlassen der View abgeräumt. Kann mit <code>DisableOtherGroups</code> nicht deaktiviert werden. In diese Gruppe können Buttons angehören, die auch bei einem ModalDialog aktiv sein müssen.
32..223	POP-UP-Gruppe, wird beim Verlassen der View abgeräumt. Diese Nummern werden von der Methode <code>HotSpot.GetNextGroupNumber()</code> vergeben.
224..239	Diese Gruppen zeigen auf eine Bibliotheksfunktion und werden beim verlassen der View abgeräumt.
240..255	Diese Gruppen zeigen auf eine Bibliotheksfunktion und bleiben beim ViewWechsel aktiv

### Wie funktionieren die HotSpots ?

In der virtuellen Maschine ist ein Speicher reserviert, der die Daten der HotSpots aufnimmt. Der Speicher ist als Array strukturiert. In einer neuen View ist der Speicher noch leer, es sind keine HotSpots installiert. HotSpots können installiert werden im der Methode `HotSpot.Install(Enter,Leave,Down,Up)`. Die Parameter in Klammern stehen für die Namen der Eventhandler, der zu einem HotSpot zugeordnet ist. Beim Installieren des HotSpot ist die Geometrie wichtig, die z.B. mit `Load.Geometry_XYWH ( 0,0, as_DisplayWidth, as_DisplayHeight)` geladen wird. In einer Arrayzeile sind folgende Daten gespeichert:

- Geometrie des HotSpots. 4xINT16
- Offset zur Zeit der Installation: 4xINT16
- Adressen der Events Enter, Leave, Down, Up : 4xAdresse
- Gruppennummer: UINT8 zugehörige Gruppennummer
- Tag: INT16, BenutzerInformation
- State: UINT8 Zustand (enabled/disabled)
- ID: INT16 auf der View einmalige Kennzeichnung des HotSpots, ID

Diese Daten, die für einen HotSpot gespeichert sind, werden in Registern der eVM zurückgeliefert, so dass der Eventhandler diese Daten wiederverwenden kann. Bei einem Keyboard unterscheiden sich die Buchstabentasten eigentlich nur durch ihre Lage und durch den Buchstaben, der ausgelöst wird. Mit den Methoden von eigerScript kann für jede Buchstabentaste derselben HotSpot installiert werden. Die Geometriedaten erhält der EventHandler zurück und im Tag kann der Character zurückgeliefert werden. Durch diese Eigenschaften können gleichartige Buttons in die gleichen EventHandler zusammengefasst werden, was enorm viel Programmcode spart.

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

Mit der Methode `HotSpot.Install(Enter,Leave,Down,Up)` wird ein HotSpot installiert. Dabei müssen die Register `eI.Pos_X1`, `eI.Pos_Y1` sowie `eI.Width` und `eI.Height` mit der Rechteckgröße vorgeladen sein.

#### Beispiel:

```

..
; Eventhandler für animierten 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 )

```

```

; Hier den Funktionscode einfügen ...

ENDSUB
..

SUB   Test_Styles

Test_Style_UP:
    INLINENWORDS (no_change)           ; entspricht eI.Pos_X1
    INLINENWORDS (no_change)           ; entspricht eI.Pos_Y1
    INLINENWORDS (120)                 ; entspricht eI.Width
    INLINENWORDS (40)                  ; entspricht eI.Height
    INLINENWORDS (8)                   ; entspricht eI.SpaceLeft
    INLINENWORDS (8)                   ; entspricht eI.SpaceRight
    INLINENWORDS (0)                   ; entspricht eI.HorizontalAdjust
    INLINENWORDS (0)                   ; entspricht eI.VericalAdjust
    INLINENWORDS (no_change)           ; entspricht eI.FillColor
    INLINENWORDS (as_FillColor)         ; entspricht eI.BackColor
    INLINENWORDS (as_FillColor)         ; entspricht eI.LineColor
    INLINENWORDS (autocolor)           ; entspricht eI.TextColor
    INLINENWORDS (Pos_left)            ; entspricht eI.Position
    INLINENWORDS (Orientation_0deg)     ; entspricht eI.Orientation
    INLINENWORDS (normal)              ; entspricht eI.Appearance
    INLINENWORDS (as_Skin_BtnBorderUP)  ; entspricht eI.BorderStyle
    INLINENWORDS (Font_Arial_14n)      ; entspricht eI.FontNumber
    INLINENWORDS (as_Skin_FormBodyColor) ; entspricht eI.BackgroundColor

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

ENDSUB
..

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

    HotSpot.TableEnable()

```

Nach dem Aufruf von HS1\_LE sind die Register mit der Buttongröße vorgeladen. Die Position wird in der Subroutine geladen, die Breite und die Höhe im Style (3. und 4. Zeile) Ein weiterer Button verwendet meistens denselben Style.

TIPP1: wird der an sich gleiche Button in einer anderen Farbe gewünscht kann das Label mit `Label.Color ( hotpink )` gefärbt werden.

TIPP2: wird der an sich gleiche Button ohne Textbeschriftung gewünscht, kann die Methode `Label.Box ( )` verwendet werden.

TIPP3: wird ein verborgener (unsichtbarer) HotSpot gewünscht, lassen sich die Koordinaten mit `Load.Geometry_XYWH ( 0,0, as_DisplayWidth, as_DisplayHeight )` festlegen. Hier wird ein HotSpot über die ganze Displayfläche gelegt, damit beim Betrachten eines Fotos nichts stört.

TIPP4: ein HotSpot kann um ein Bild gelegt werden. Nach dem Laden eines Bildes mit `File.ReadEGI( FileName )` liegen die Breite und die Höhe des Bildes in den Registern `eI.Width` und `eI.Height`. Somit kann der HotSpot einfach über das Bild gelegt werden.

TIPP5: wenn ein Event nicht ausprogrammiert wird, kann als Parameter `NIL` angegeben werden. `HotSpot.Install ( NIL, NIL, HS1_DN, NIL )` wenn nur der DownEvent ausprogrammiert ist.

TIPP6: Mehrere HotSpots dürfen dieselben EventHandler benutzen. So passiert beim Drücken auf ein Bild oder auf ein Label dasselbe.

### HotSpot.GetNextGroupNumber ( )

0\$50

Mit der Methodel `HotSpot.GetNextGroupNumber ( )` wird eine nächste HotSpotGruppe im Bereich von 32..223 bezogen. Das Ergebnis liegt im Register `eI.HotSpotGroup`. HotSpots die später installiert werden, werden mit dieser Gruppennummer installiert. Die Gruppennummer ist ein Betriebssystem-Wert, der hochgezählt wird und nicht beeinflusst werden kann. Die Gruppen sind nützlich, um Gruppen von HotSpots, wie sie beispielsweise bei einem PopUp-Dialog vorkommen, wieder abzuräumen, und den Programmcode unabhängig zu halten.

#### Beispiel:

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

### HotSpot.GetCurrGroupNumber ( )

0\$50

Mit der Methodel `HotSpot.GetCurrGroupNumber ( )` wird die aktuelle Gruppennummer angefordert, ohne den internen Zähler hochzuzählen. Das Ergebnis liegt im Register `eI.HotSpotGroup`.

#### Beispiel:

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

<b>HotSpot.DisableGroup (VarInt:Group)</b>	0\$80
--	-------

Mit der Methode `HotSpot.DisableGroup (Group)` wird eine ganze Gruppe von HotSpots deaktiviert. Die HotSpots bleiben in der HotSpotTabelle gespeichert, erzeugen aber keine Events mehr. Die Methode wird angewendet, um HotSpots zeitweilig zu deaktivieren. Die Gruppennummer wird als Parameter übergeben.

Beispiel:

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

<b>HotSpot.EnableGroup (VarInt:Group)</b>	0\$80
---	-------

Mit der Methode `HotSpot.EnableGroup (Group)` wird eine ganze Gruppe von HotSpots reaktiviert. Die Gruppennummer wird als Parameter übergeben.

Beispiel:

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

<b>HotSpot.Disable_By_ID (VarInt:HotSpot_ID)</b>	0\$80
--	-------

Mit der Methode `HotSpot.DisableGroup (Group)` wird eine ganze Gruppe von HotSpots deaktiviert. Die HotSpots bleiben in der HotSpotTabelle gespeichert, erzeugen aber keine Events mehr. Die Methode wird angewendet, um HotSpots zeitweilig zu deaktivieren. Die Gruppennummer wird als Parameter übergeben.

Beispiel:

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

<b>HotSpot.Enable_By_ID (VarInt:HotSpot_ID)</b>	0\$80
---	-------

Mit der Methode `HotSpot.EnableGroup (Group)` wird eine ganze Gruppe von HotSpots reaktiviert. Die Gruppennummer wird als Parameter übergeben.

Beispiel:

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

<b>HotSpot.DeInstallGroup ( )</b>	0\$50
-----------------------------------	-------

Mit der Methode `HotSpot.DeInstallGroup ()` wird eine ganze Gruppe von HotSpots deinstalliert. Die Methode wird angewendet, um einen PopUp-Dialog abzuräumen. Die Gruppennummer muss im Register `eI.HotSpotGroup` übergeben werden. Da in einem PopUpDialog das Abräumen meistens durch Verlassen des Controls mit einem NavigationsButton erfolgt, kann auch die HotSpot-Gruppennummer, die im Register `eI.HS_Group` zurückgeliefert wird, zum Deinstallieren verwendet werden.

Beispiel:

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

### HotSpot.TableEnable()

Mit der Methode `HotSpot.TableEnable()` wird die ganze HotSpotTable für die Auswertung freigegeben. Dieser Befehl ist unbedingt nötig, wenn HotSpots ausgewertet werden sollen.

### HotSpot.TableDestroy()

Mit dem Befehl `HotSpot.TableDestroy()` wird die ganze HotSpotTable für die Auswertung gesperrt und gelöscht. Mit diesem Befehl kann die Auswertung der HotSpots unterbrochen werden. Nachher müssen neue HotSpots installiert werden mit `HotSpot.Install(Enter,Leave,Down,Up)` und die Auswertung der Tabelle mit `HotSpot.TableEnable()` wieder gestartet werden.

### HotSpot.DisableRegion ()

0\$80

Mit der Methode `HotSpot.DisableRegion ()` werden alle HotSpots, die in einer Fläche sind oder die Fläche berühren, gesperrt. Die Fläche wird durch ein Rechteck beschrieben und kann mit `Load.Geometry_XYWH (X,Y,W,H)` geladen werden.

Beispiel:

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

### HotSpot.EnableRegion ()

0\$80

Mit der Methode `HotSpot.EnableRegion ()` werden alle HotSpots, die in einer Fläche sind oder die Fläche berühren, freigegeben.

Beispiel:

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

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

0\$80

Mit der Methode `HotSpot.GetInfo_By_ID (ID)` werden alle in der HotSpot-Tabelle gespeicherten Informationen über einen HotSpot in die Register `eI.HS_...` geschrieben. Damit kann die Geometrie, das Tag, der State etc zurückgelesen werden.

Beispiel:

```
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
```

Die Routine gibt alle Parameter über die Debug-Schnittstelle aus.

## Class HotKey

HotKeys sind Funktionstasten, die an der Seite des Displays angeordnet sind oder ein Keypad, das an den Rechner angeschlossen ist. Die Tasten können beim Drücken und beim Loslassen einen Event erzeugen. Für die Tasten, die ausgewertet werden sollen, muss der EventHandler installiert sein. Mit den HotKeys kann eine Anwendung ohne Touchpanel realisiert werden. Die HotKeys können in jedem Fall verwendet werden, egal ob ein Touchpanel vorhanden ist oder nicht. Für eine Maschinenbedienung kann eine Kombination sinnvoll sein, indem die Tasten für die am häufigsten gebrauchten Funktionen zusätzlich als Hardwaretasten vorhanden sind. Eingaben erfolgen dann mit dem Touchpanel oder mit den HotKeys.

Die Tasten werden am FOXS an den BoxHeader CK5 oder die roten AMP microMatch-Stecker angeschlossen. Hardwaremässig sind 16 Funktionstasten vorhanden. Auf dem BoxHeader sind alle 16 Tastatureingänge vorhanden. An die roten AMP-Stecker können 2x 6 Tasten und 2 x 2 Tasten angeschlossen werden. Die Taste 0 liefert ein Zeichen das ASCII-Zeichen 'A' beim Drücken und beim 'a'. Der BoxHeader und die roten AMP-Stecker belegen die gleichen Eingänge. Die genaue Belegung ist der HardwareDokumentation des FOXS zu entnehmen.

Damit die Tastatureingänge in einer View ausgewertet werden können muss zuerst ein HotKey installiert werden. Wenn die HotKey installiert sind, wird mit `HotKey.InputFlush()` allenfalls noch gespeicherte Tasten gelöscht. Danach muss die HotKeyTable aktiviert werden mit `HotKey.TableEnable()`. Die HotKeys können aber auch auf einer View dynamisch installiert, neu installiert und deinstalliert werden.

Die HotKey können View local oder global installiert werden. Für die globalen Key muss unbedingt auch ein globaler EventHandler vorhanden sein!

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

Mit dem Befehl `HotKey.InstallLocalKey(Key,Event,Tag)` wird ein HotKey lokal installiert. Dabei wird der Methode der Bytewert oder ein ASCII-Zeichen und das Label des dazugehörigen EventHandlers mitgegeben. Zu beachten ist, dass der Wert für den Key im Bereich von 0x00..0x7F sein muss. Andere Werte werden mit einer Maske auf diesen Wertebereich gemappt. Das Tag ist ein beliebiger 16-Bit Wert, der im Register `eI.HK_Tag` zurückgeliefert wird, wenn der Event ausgelöst wird. Dieser Wert kann ideal dazu verwendet werden, die vorgegebenen KeyCodes in spezifische Tastencodes umzuwandeln. Wenn der Key installiert wird, ist er enabled.

```
HotKey.DeInstallKey(VarInt:Key)
```

Mit dem Befehl `HotKey.DeInstallKey(Key)` wird ein einzelner HotKey deinstalliert. Dabei wird der Methode der KeyCode (ein ASCII-Zeichen) übergeben. Der nächste Tastendruck auf die deinstallierte Taste bewirkt nichts mehr. Der Befehl kann für lokale und globale HotKeys angewandt werden.

## **HotKey.DeInstallLocalKeys ( )**

Mit dem Befehl `HotKey.DeInstallLocalKeys()` werden alle lokalen HotKeys deinstalliert. Dieser Befehl wird bei einem Viewwechsel automatisch ausgeführt, da die lokalen Events auf der neuen View nicht mehr gültig sind.

## **HotKey.DisableLocalKeys ( )**

Mit dem Befehl `HotKey.DisableLocalKeys()` werden alle local Keys in der HotKeyTabelle gesperrt. Die globalen Keys können weiterhin verarbeitet werden.

## **HotKey.EnableLocalKeys ( )**

Mit dem Befehl `HotKey.EnableLocalKeys()` werden alle local Keys in der HotKeyTabelle freigegeben.

## **HotKey.InputFlush ( )**

Mit dem Befehl `HotKey.InpuFlush()` wird der Eingangspuffer gelöscht. Dieser Befehl wird angewandt, bevor die HotKeyTable freigegeben wird, damit keine Tastendrucke gespeichert sind.

## **HotKey.InputSelect ( i )**

Mit dem Befehl `HotKey.InpuSelect(i)` wird der Eingangskanal gewählt. Auf dem FOXS wird immer der Treiber aktiviert, der über die Tastaturschnittstelle die Tasten abfragt. Auf dem FOXS muss diese Methode nicht unbedingt aufgerufen werden.

## **HotKey.TableEnable ( )**

Mit der Methode `HotKey.TableEnable()` wird die HotKeyTabelle freigegeben. Ab der Freigabe sind die Tasten aktiv und können von den entsprechend installierten EventHandlern ausgewertet werden.

## **HotKey.TableDisable ( )**

Mit der Methode `HotKey.TableDisable()` wird die HotKeyTabelle gesperrt. Dadurch werden keine HotKeys mehr ausgewertet. Die Tabelleneinträge sind noch vorhanden und durch den

Befehl `HotKey.TableEnable()` kann die Tabelle wieder eingeschaltet werden. Dieser Befehl kann dazu verwendet werden, Tasteneingaben vorübergehend zu sperren.

### HotKey.TableInit()

Mit der Methode `HotKey.TableInit()` wird die HotKeyTabelle komplett initialisiert und gelöscht. Sowol die lokalen als auch die globalen Eventhandler-Einträge werden aus der Tabelle entfernt.

## Anwendungsbeispiel

An den FOX angeschlossene Tasten sollen ausgewertet werden. Die Tasteneingänge D,E,F sollen die Events HK\_Red, HK\_Green und HK\_Blue auslösen. Dazu werden allfällig vorhandene, alte Tasten gelöscht, anschliessend die Tasten als lokale Tasten installiert. In diesem Beispiel wird das Tag nicht gebraucht, deshalb wird 0 geladen.

```
; HotKeys installieren -----  
HotKey.InputFlush()  
HotKey.InstallLocalKey( "D" ,HK_Red,0)  
HotKey.InstallLocalKey( "E" ,HK_Green,0)  
HotKey.InstallLocalKey( "F" ,HK_Blue,0)  
  
HotKey.EnableLocalKeys()  
HotKey.TableEnable()
```

## Class Timer

Timer sind in einem ereignisgesteuerten System eine der wichtigsten Quellen für Events. Das TimerModul der virtuellen Maschine umfasst 8 Timer, die entweder lokal oder global sein können. Im Gegensatz zu den lokalen Timern, die beim verlassen der View deinstalliert werden, bleiben die globalen Timer bei einem Viewwechsel aktiv, was einen globalen Eventhandler bedingt. Die globalen Timer sollten zu einem "nichtzeichnenden", schnellen Event führen.

Die eVM-Timer haben eine TicSektion und eine EventSektion, die mehr oder weniger unabhängig voneinander funktionieren. Die TicSektion setzt ein internes Flag, wenn sie abgelaufen ist und zählt den internen TimerExpiredCounter um eins hoch. Der EventManager prüft periodisch, ob ein Timer abgelaufen ist und erzeugt einen Event, der von der eVM abgearbeitet wird.

Die Timer können zur Zeitüberwachung oder zur periodischen Auffrischung der Anzeige gebraucht werden.

## **Timer.Init()**

Mit dem Befehl `Timer.Init()` wird die gesamte Timerstruktur in einen Grundzustand versetzt, bei dem alle Timer abgeschaltet sind und alle Timer deinstalliert sind. Nach diesem Befehl müssen die Timer zuerst wieder installiert werden.

### Timer: Eventsektion

Die Eventsektion der Timer ist verantwortlich, dass ein abgelaufener Timer einen EventHandlerler findet, der abgearbeitet werden kann.

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

Mit der Methode `Timer.InstallLocal(Timer, Event)` wird ein lokaler Timer installiert.

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

Mit der Methode `Timer.InstallGlobal(Timer, Event)` wird ein globaler Timer installiert. Es wird empfohlen, dass der globale Timer äusserst kurz gehalten wird. Der EventHandlerler des globalen Timers muss im globalen Code sein, da er auch weiterläuft, wenn die View gewechselt wird.

## **Timer.TableDisable()**

Mit der Methode `Timer.TableDisable()` wird die TimerTabelle gesperrt. Dadurch werden keine Events von lokalen Timern erzeugt. Die Tabelleneinträge sind noch vorhanden und durch den Befehl `Timer.TableEnable()` kann die Tabelle wieder eingeschaltet werden. Dieser Befehl kann dazu verwendet werden, TimerEvents vorübergehend zu sperren. Bei einem Viewwechsel wird diese Methode automatisch angewandt, da auf der neuen View die Adressen der Events geändert haben und somit ungültig sind.

## **Timer.TableEnable()**

Mit der Methode `Timer.TableEnable()` wird die TimerTabelle freigegeben. Ab der Freigabe können die Timer Events erzeugen.

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

Mit der Methode `Timer.DeInstall(VarInt:Timer)` wird ein Timer deinstalliert. Falls der Timer vor dem DeInstall einen Event gefeuert hat, wird dieser auch nach dem DeInstall noch

ausgeführt. Falls der Event nicht mehr ausgeführt werden soll, muss die Methode `Timer.Kill(VarInt:Timer)` angewendet werden.

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

Mit der Methode `Timer.Kill(VarInt:Timer)` wird ein Timer deinstalliert und falls der Timer schon einen Event gefeuert hat, dieser gelöscht. Diese Methode empfiehlt sich bei autorepeat-Funktionen, die durch einen HotSpot gesteuert werden.

## Timer: Ticsektion

Die Ticsektion der Timer wird alle Millisekunden bearbeitet. Dabei wird bei jedem Timer geprüft ob er eingeschaltet (RUN) ist, oder ob er ausgeschaltet (STOP) ist. Wenn ein Timer im RUN-Mode ist, wird sein TimerCounter jede ms um 1 vermindert. Beim Erreichen von Null wird der Timer aus dem internen Reload-Register neu geladen und der interne expired Counter um 1 erhöht und das EXP-Flag gesetzt. Dann entscheidet der AUTORELOAD-Mode, ob der Timer weiterhin im RUN-Mode bleibt (continuous) oder auf STOP geht (single).

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

Mit der Methode `Timer.Load()` wird der Timer geladen. Der Wert wird ins RELOAD-Register des betreffenden Timers geschrieben. Beim nächsten Ablauf des Timers oder mit den Befehlen `Timer.StartSingle()` und `Timer.StartContinuous()` wird dieser Wert in den TimerCounter übernommen und die geladene Zeit beginnt abzulaufen.

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

Mit der Methode `Timer.StartSingle()` wird der TimerCounter aus dem Reload-Register geladen, der ExpiredCounter auf Null gestellt und der Timer für einen Zyklus geladen. Nach Ablauf der Zeit, löst der Timer einen Event aus, wenn die Events freigegeben sind.

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

Mit der Methode `Timer.startContinuous()` wird der TimerCounter aus dem Reload-Register geladen, der ExpiredCounter auf Null gestellt und der Timer für periodische Zyklen geladen. Nach jedem Ablauf der Zeit, wird der ExpiredCounter um 1 erhöht und der Timer löst einen Event aus, wenn die Events freigegeben sind.

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

Mit der Methode `Timer.Stop()` wird der betreffende Timer angehalten. Er kann mit `Timer.Continue()` wieder zum Laufen gebracht werden.

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

Mit der Methode `Timer.Continue()` wird der betreffende Timer wieder zum Laufen gebracht, nachdem er mit `Timer.Stop()` angehalten wurde.

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

Mit der Methode `Timer.Reload()` wird der CounterTimer aus dem Reload-Register des betreffenden Timers geladen. Wenn der Timer am Laufen ist, bewirkt die Methode, dass die Zeit wieder von vorn zu laufen beginnt. Diese Methode kann gebraucht werden, um den Ausfall von Ereignissen zu überwachen: jedes zu Überwachende Ereignis ruft die Methode auf. Bleiben die Ereignisse zu lange aus, wird ein Timeout-Event erzeugt.

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

Mit der Methode `Timer.Get_TimerCounter()` wird der TimerCounter, d.h. der Abwärtszähler des betreffenden Timers im Rückgabewert zurückgeliefert. Diese Methode kann benutzt werden, um zu prüfen, wieviel Zeit bis zum Ablauf des Timers noch bleibt.

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

Mit der Methode `Timer.GetTimerCounter()` wird der ExpiredCounter, d.h. der Zähler, der zählt wieviel mal der Timer abgelaufen ist, im Rückgabewert zurückgeliefert. Diese Methode kann benutzt werden, um zu prüfen, ob der Timer abgelaufen ist, oder um nachzuprüfen, ob TimerEvents verloren gegangen sind.

## Serielle Schnittstellen /Serial Server

Die Bedienung der seriellen Schnittstellen erfolgt durch den SerialServer. Der SerialServer hat die Aufgabe, asynchron zum Programmablauf eintreffende Zeichen entgegenzunehmen, zwischenspeichern und falls gewünscht, einen Event auszulösen. Die Zeichen gelangen nach dem Empfang durch die UART in einen Ringpuffer. Der SerialServerTask entnimmt dem Ringpuffer die Zeichen und speichert sie im Linearen InputBuffer zwischen. Für jede Schnittstelle ist ein Monitor vorhanden. Die eigerScript Applikation kann bis zu 16 MonitorSlots installieren. Die MonitorSlots vergleichen die ankommenden Zeichen mit einer Unter- und einer Obergrenze und lösen falls das Zeichen im Intervall liegt einen Event aus. Mit den MonitorSlots ist es möglich, dass ein einzelnes Zeichen wie z. B. CR einen Event auslöst, oder eine Gruppe von Zeichen z.B. "A".. "Z" kann einen Event auslösen. Sollen alle Buchstaben einen Event auslösen, kann ein zweiter MonitorSlot installiert werden, der den gleichen Event auslöst, wenn die Zeichen "a".. "z" eintreffen. Es stehen 16 MonitorSlots mit den Nummern 0..15 zur Verfügung. Der Monitor beginnt mit dem Test bei der Nummer 0. Deshalb muss eine Untermenge z.B. "G".. "M" mit einer tieferen MonitorSlotNummer installiert sein, als die Übermenge "A".. "Z". Die Zeichengrenzen sind immer inklusiv.

Die MonitorSlots können lokal oder global sein. Ein lokaler MonitorSlot wird beim Verlassen der View deinstalliert, da die EventHandler lokal auf der View vorhanden sind. Im Gegensatz dazu werden globale MonitorSlots nicht abgeräumt.

### Schnittstelle initialisieren

<b>Serial.SetBaudrate(VarInt:COMx, VarInt:Baudrate)</b>	<b>0\$40</b>
---	--------------

Mit dem Befehl `Serial.SetBaudrate(VarInt:COMx,VarInt:Baudrate)` wird die Baudrate der entsprechenden Schnittstelle verstellt. Die übrigen Einstellungen sind no parity, 8bit 1 Stoppbit.

Wählbare Baudraten sind: 2'400, 4'800, 9'600, 19'200, 38'400, 57'600, 115'200, 250'000, wobei die bevorzugten Werte unterstrichen sind.

```
Serial.SetBaudrate(COM1, Baud_38400)
```

### Zeichen empfangen

<b>Serial.Rx_Char(VarInt:COMx, VarInt:Char)</b>	<b>0\$40</b>
---	--------------

Mit dem Befehl `Serial.Rx_Char(VarInt:COMx,VarInt:Char)` wird vom entsprechenden Kanal ein Zeichen eingelesen. Wenn an der Schnittstelle kein Zeichen zum Abholen bereit ist, wird der Wert -1 zurückgeliefert. Die Empfangenen Zeichen sind im Bereich von 0 bis 255

```
Serial.Rx_Char ( COM1, MyCharacter.I )
```

**Serial.Rx\_Input\_Flush(VarInt:COMx)****0\$40**

Mit dem Befehl `Serial.Rx_Input_Flush(VarInt:COMx)` wird die entsprechende Schnittstelle gereinigt. Dabei wird der Eingangsringpuffer gelöscht und der Lineare InputBuffer zurückgesetzt. Dieser Befehl ist nützlich, damit sich nach einer Initialisierung nicht alte Zeichen in den Puffern befinden, die falsch ausgewertet werden.

**Serial.Rx\_InBuf\_Clear(VarInt:COMx)****0\$40**

Mit dem Befehl `Serial.Rx_InBuf_Clear(VarInt:COMx)` wird der lineare InputBuffer gelöscht. SerialKey installiert. Dabei wird der Methode der Bytewert oder ein ASCII-Zeichen und das Label des dazugehörigen EventHandlers mitgegeben. Trifft ein Zeichen über die Schnittstelle ein, dessen EventHandler installiert wurde, wird der entsprechende Event ausgeführt.

**Serial.Rx\_InBuf\_to\_String\_Append(VarInt:COMx,VarStr)****0\$40**

Mit dem Befehl `Serial.Rx_InBuf_to_String(VarInt:COMx,VarStr:String)` wird der lineare InputBuffer an einen String angehängt. Damit kann ein String aus der seriellen Schnittstelle eingelesen werden. Er steht zur weiteren Bearbeitung mit den Stringfunktionen der Klasse String zur Verfügung. Der InputBuffer wird bei dieser Methode nicht gelöscht und kann weiterverwendet werden.

**Serial.Rx\_MonitorSlot\_InstallLocal(VarInt:COMx,  
VarInt:Slot,labelRelative24:EventHandler,  
VarInt:CharLO,VarInt:CharHI)****0\$40**

Mit dem Befehl `Serial.Rx_MonitorSlot_InstallLocal(...)` wird ein lokaler MonitorSlot aufgesetzt.

VarInt:COMx           Schnittstellennummer COM1 oder COM2

VarInt:Slot           MonitorSlotNummer [0..15]

labelRelative24      EventHandler, der angesprochen wird, wenn ein Zeichen die Bedingung des MonitorSlot erfüllt.

VarInt:CharLO        untere Grenze des Zeichens (inklusive)

VarInt:CharHI        obere Grenze des Zeichens(inklusive)

Beispiel:

Beim Empfang eines CR-Zeichens, soll ein Event ausgeführt werden:

```
Serial.Rx_Monitor_InstallLocal (COM1,0,Receive_LineEvent,CR,CR)
```

Alles Grossbuchstaben lösen einen Event aus:

```
Serial.Rx_Monitor_InstallLocal (COM1,1,Receive_LineEvent, "A", "Z")
```

### Zeichen senden

#### **Serial.Tx\_String (VarInt:COMx, VarStr)**

Mit der Methode `serial.Tx_string(Kanal,String)` wird ein String über die serielle Schnittstelle ausgegeben. Der Nullterminierte String wird ohne die abschliessende Null gesendet. Muss eine Null gesendet werden, wird die Methode `Serial.Tx_NUL(VarInt)` verwendet. Der Kanal gibt an, über welche Schnittstelle der String ausgegeben werden soll. Die Angabe des Kanals ist für alle Methoden der Klasse nötig.

```
Serial.Tx_String (COM2, 'Hallo') ; UART-Ausgabe auf COM2-Schnittstelle
```

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

Mit der Methode `Serial.Tx_CRLF(Kanal)` wird ein CRLF über die serielle Schnittstelle, die mit der Kanalnummer angegeben wird, ausgegeben, um eine Zeile abzuschliessen.

```
Serial.Tx_CRLF (COM1) ; UART-Ausgabe auf COM1-Schnittstelle
```

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

Mit der Methode `Serial.Tx_NUL(Kanal)` wird ein Null-Character 0x00 über die serielle Schnittstelle, die mit der Kanalnummer angegeben wird, ausgegeben. Diese Funktion kann dazu verwendet werden, damit beim empfangenden System kein Timeout eintritt.

Anmerkung: die terminierenden NULL-Character eines Strings werden nicht übertragen.

```
Serial.Tx_NUL ( COM1 ) ; UART-Ausgabe auf COM1-Schnittstelle
```

#### **Serial.Tx\_Char (VarInt:COMx, VarInt:Char)**

Mit der Methode `Serial.Tx_Char(Kanal,Character)` wird ein Zeichen über die entsprechende Schnittstelle ausgegeben.

```
Serial.Tx_Char ( COM1, "A" ) ; UART-Ausgabe auf COM1-Schnittstelle
```

## Class Colors

Die Class Colors beeinflusst Farbvariablen. Farbanteile können vergrößert oder vermindert werden, Farbanteile bestimmt und die Farben von der internen 1:5:5:5-Darstellung in die 8-Bit Darstellung umgewandelt werden.

### Farben verändern

#### **Colors.AutoColor\_5Bit(VarInt:Ziel,VarInt:Quelle)**

Mit der Methode `Colors.AutoColor_5Bit(Ziel,Quelle)` wird das Ziel mit schwarz oder weiss geladen, je nach Farbe der Quelle. Für eine dunkle Farbe liefert die Funktion weiss und für eine helle Farbe schwarz. Die Funktion dient zur Zuweisung einer Schriftfarbe, wenn die Hintergrundfarbe noch nicht bekannt ist.

```
Colors.AutoColor_5Bit(eI.TextColor,eI.BackColor) ; TextFarbe aus  
Hintergrundfarbe
```

#### Spezialfall:

Ist der Input-Wert transparent, wird die Zielfarbe nicht verändert.

#### **Colors.GreyValue\_5Bit(VarInt:Ziel,VarInt:Quelle)**

Mit der Methode `Colors.GreyValue_5Bit(Ziel,Quelle)` wird das Ziel mit dem der Farbe entsprechenden Grauton geladen. Die Methode kann gut gebraucht werden, um Inaktive Buttons zu zeichnen.

```
Colors.GreyValue_5Bit(eI.BackColor,eI.FillColor) ; Hintergrundfarbe Grauwert zuweisen
```

#### Spezialfall:

Ist der Input-Wert transparent, wird die Zielfarbe transparent geladen.

#### **Colors.BrightenColor\_5Bit(VarInt:Ziel,VarInt:Quelle)**

Mit der Methode `Colors.BrightenColor_5Bit(Ziel,Quelle)` wird das Ziel mit der aufgehellten Farbe zur Quellfarbe geladen. Die Methode kann gut gebraucht werden, zum eine passende Rahmenfarbe zu laden

```
Colors.BrightenColor_5Bit(eI.LineColor,eI.BackColor) ; LinenFarbe aus Hintergrundfarbe
```

#### Spezialfall:

Ist der Input-Wert transparent, wird die Zielfarbe nicht verändert.

#### **Colors.DarkenColor\_5Bit(VarInt:Ziel,VarInt:Quelle)**

Mit der Methode `Colors.DarkenColor_5Bit(Ziel,Quelle)` wird das Ziel mit der abgedunkelten Farbe zur Quellfarbe geladen. Die Methode kann gut gebraucht werden, zum eine passende Rahmenfarbe zu laden

```
Colors.DarkenColor_5Bit(eI.LineColor,eI.BackColor) ; LinenFarbe aus
Hintergrundfarbe
```

#### Spezialfall:

Ist der Input-Wert transparent, wird die Zielfarbe nicht verändert.

### **Colors.ColorMix\_5Bit(VarInt:Ziel,VarInt:Col,VarInt:Col)**

Mit der Methode `Colors.ColorMix_5Bit(Ziel,Quelle,Quelle)` wird das Ziel mit der Mischfarbe der Quellen geladen. Jede Farbe kann mit der Graupalette gemischt werden, um Schattierungen und Farbtöne zu erhalten. Die Farben werden im Verhältnis 1:1 gemixt.

```
Colors.ColorMix_5Bit(eI.LineColor,eI.BackColor,greyscale) ; LinenFarbe aus
; Hintergrundfarbe
```

### **Colors.ColorMix\_3to1\_5Bit(VarInt:Ziel,VarInt,VarInt)**

Mit der Methode `Colors.ColorMix_5Bit(Ziel,Quelle 3 Teile,Quelle 1 Teil)` wird das Ziel mit der Mischfarbe der Quellen geladen. Jede Farbe kann mit der Graupalette gemischt werden, um Schattierungen und Farbtöne zu erhalten. Die Farben werden im Verhältnis 3:1 gemixt.

```
Colors.ColorMix_3to1_5Bit(eI.LineColor,eI.BackColor,greyscale) ; LinenFarbe aus
; Hintergrundfarbe
```

#### Tip:

Vom ersten Wert werden 3 Teile genommen. Ist die Farbe vorn und der Grauton hinten, ergeben sich sattere Farben, umgekehrt erhalten die Grautöne einen Farbstich.

### **Colors.InverseColor(VarInt:Ziel,VarInt:Quelle)**

Mit der Methode `Colors.InverseColor(Ziel,Quelle)` wird das Ziel mit der Komplementärfarbe zur Quellfarbe geladen. Die Methode kann gut gebraucht werden, zum eine passende Rahmenfarbe zu laden.

```
Colors.InverseColor(eI.LineColor,eI.BackColor) ; LinenFarbe aus Hintergrundfarbe
```

#### Spezialfall:

Ist der Input-Wert transparent, wird die Zielfarbe nicht verändert.

### **Colors.MoreRed\_5Bit(VarInt:Farbe)**

Mit der Methode `Colors.MoreRed(Farbe)` wird die Farbe um einen TIC erhöht. Ist die Farbe am Maximum, nimmt sie wieder den Wert 0 an. Die Methode kann gut gebraucht werden, um Farbverläufe zu erstellen.

```
Colors.MoreRed_5Bit(eI.FillColor)           ; mehr rot in FillColor
```

## Spezialfall:

Ist der Input-Wert transparent, wird die Zielfarbe nicht verändert.

## **Colors.MoreRed\_Limit\_5Bit(VarInt:Farbe)**

Mit der Methode `Colors.MoreRed_Limit_5Bit(Farbe)` wird die Farbe um einen TIC erhöht. Ist die Farbe am Maximum, bleibt sie unverändert im Maximum. Die Methode kann gut gebraucht werden, um Farbverläufe zu erstellen.

```
Colors.MoreRed_Limit_5Bit(eI.FillColor)     ; mehr rot in FillColor
```

## Spezialfall:

Ist der Input-Wert transparent, wird die Zielfarbe nicht verändert.

## **Colors.MoreGreen\_5Bit(VarInt:Farbe)**

Mit der Methode `Colors.MoreGreen(Farbe)` wird die Farbe um einen TIC erhöht. Ist die Farbe am Maximum, nimmt sie wieder den Wert 0 an. Die Methode kann gut gebraucht werden, um Farbverläufe zu erstellen.

```
Colors.MoreGreen_5Bit(eI.FillColor)         ; mehr grün in FillColor
```

## Spezialfall:

Ist der Input-Wert transparent, wird die Zielfarbe nicht verändert.

## **Colors.MoreGreen\_Limit\_5Bit(VarInt:Farbe)**

Mit der Methode `Colors.MoreGreen_Limit_5Bit(Farbe)` wird die Farbe um einen TIC erhöht. Ist die Farbe am Maximum, bleibt sie unverändert im Maximum. Die Methode kann gut gebraucht werden, um Farbverläufe zu erstellen.

```
Colors.MoreGreen_Limit_5Bit(eI.FillColor)   ; mehr grün in FillColor
```

## Spezialfall:

Ist der Input-Wert transparent, wird die Zielfarbe nicht verändert.

## **Colors.MoreBlue\_5Bit(VarInt:Farbe)**

Mit der Methode `Colors.MoreBlue(Farbe)` wird die Farbe um einen TIC erhöht. Ist die Farbe am Maximum, nimmt sie wieder den Wert 0 an. Die Methode kann gut gebraucht werden, um Farbverläufe zu erstellen.

```
Colors.MoreBlue_5Bit(eI.FillColor)           ; mehr blau in FillColor
```

#### Spezialfall:

Ist der Input-Wert transparent, wird die Zielfarbe nicht verändert.

### **Colors.MoreBlue\_Limit\_5Bit(VarInt:Farbe)**

Mit der Methode `Colors.MoreBlue_Limit_5Bit(Farbe)` wird die Farbe um einen TIC erhöht. Ist die Farbe am Maximum, bleibt sie unverändert im Maximum. Die Methode kann gut gebraucht werden, um Farbverläufe zu erstellen.

```
Colors.MoreBlue_Limit_5Bit(eI.FillColor)     ; mehr blau in FillColor
```

#### Spezialfall:

Ist der Input-Wert transparent, wird die Zielfarbe nicht verändert.

### **Colors.LessRed\_5Bit(VarInt:Farbe)**

Mit der Methode `Colors.LessRed(Farbe)` wird die Farbe um einen TIC vermindert. Ist die Farbe am Minimum, nimmt sie wieder den Wert 31 an. Die Methode kann gut gebraucht werden, um Farbverläufe zu erstellen.

```
Colors.LessRed_5Bit(eI.FillColor)           ; weniger rot in FillColor
```

#### Spezialfall:

Ist der Input-Wert transparent, wird die Zielfarbe nicht verändert.

### **Colors.LessRed\_Limit\_5Bit(VarInt:Farbe)**

Mit der Methode `Colors.LessRed_Limit_5Bit(Farbe)` wird die Farbe um einen TIC vermindert. Ist die Farbe am Minimum, bleibt sie unverändert im Minimum. Die Methode kann gut gebraucht werden, um Farbverläufe zu erstellen.

```
Colors.LessRed_Limit_5Bit(eI.FillColor)     ; weniger rot in FillColor
```

#### Spezialfall:

Ist der Input-Wert transparent, wird die Zielfarbe nicht verändert.

### **Colors.LessGreen\_5Bit(VarInt:Farbe)**

Mit der Methode `Colors.LessGreen(Farbe)` wird die Farbe um einen TIC vermindert. Ist die Farbe am Minimum, nimmt sie wieder den Wert 31 an. Die Methode kann gut gebraucht werden, um Farbverläufe zu erstellen.

```
Colors.LessGreen_5Bit(eI.FillColor) ; weniger grün in FillColor
```

### Spezialfall:

Ist der Input-Wert transparent, wird die Zielfarbe nicht verändert.

### **Colors.LessGreen\_Limit\_5Bit(VarInt:Farbe)**

Mit der Methode `Colors.LessGreen_Limit_5Bit(Farbe)` wird die Farbe um einen TIC vermindert. Ist die Farbe am Minimum, bleibt sie unverändert im Minimum. Die Methode kann gut gebraucht werden, um Farbverläufe zu erstellen.

```
Colors.LessGreen_Limit_5Bit(eI.FillColor) ; weniger grün in FillColor
```

### Spezialfall:

Ist der Input-Wert transparent, wird die Zielfarbe nicht verändert.

### **Colors.LessBlue\_5Bit(VarInt:Farbe)**

Mit der Methode `Colors.LessBlue(Farbe)` wird die Farbe um einen TIC vermindert. Ist die Farbe am Minimum, nimmt sie wieder den Wert 31 an. Die Methode kann gut gebraucht werden, um Farbverläufe zu erstellen.

```
Colors.LessBlue_5Bit(eI.FillColor) ; weniger blau in FillColor
```

### Spezialfall:

Ist der Input-Wert transparent, wird die Zielfarbe nicht verändert.

### **Colors.LessBlue\_Limit\_5Bit(VarInt:Farbe)**

Mit der Methode `Colors.LessBlue_Limit_5Bit(Farbe)` wird die Farbe um einen TIC vermindert. Ist die Farbe am Minimum, bleibt sie unverändert im Minimum. Die Methode kann gut gebraucht werden, um Farbverläufe zu erstellen.

```
Colors.LessBlue_Limit_5Bit(eI.FillColor) ; weniger blau in FillColor
```

### Spezialfall:

Ist der Input-Wert transparent, wird die Zielfarbe nicht verändert.

## Farbwerte setzen/holen

Die Farbwerte können gesetzt werden und auch wieder in einzelne Zahlen gesplittet werden. Es existieren Methoden, die die Zahl aus dem 16-Millionen-Farbraum in den eiger-Farbraum (HighColor) umrechnen und umgekehrt. Damit lassen sich die gleichen Zahlendarstellungen verwenden, wie sie auf dem PC und in der Web-Programmierung üblich sind.

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

Mit der Methode `Colors.GetPixelColor(Ziel,X,Y)` wird das Ziel mit der Farbe, die sich auf dem Bildschirm an der absoluten Koordinate X und Y befindet geladen. Der Befehl holt sich die Farbe aus dem AVR-Videospeicher. Man kann sich die Methode als eine Art Farbpipette vorstellen. Wird als X/Y-Wert die Mauskoordinate (`eI.MouseDown_X` und `eI.MouseDown_Y`) angegeben, entsteht der Effekt eines Farbwählers. Zu beachten ist allerdings, dass die Farbe aus dem AVR-Videoram-zurückgelesen wird. Sollte sich im RVR-Videoram ein anderes Bild befinden, wird der Benutzer getäuscht.

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

**Colors.Load\_RGB(VarInt:Ziel,VarInt:R,VarInt:G,VarInt:B)**

Mit der Methode `Colors.Load_RGB(Ziel,R,G,B)` wird das Ziel mit der Farbe, die sich aus Rot-, Grün-, und Blauanteil zusammensetzt. Zu beachten ist, dass nur das niederwertige Byte des Integers berücksichtigt wird. Die Werte sollen also im Bereich [0..255] liegen. Die Funktion ist nützlich, um eine 24-Bit-Farbe in eine eiger-Farbe umzurechnen.

```
Colors.Load_RGB(eI.TextColor,0,0,255) ; TextFarbe ist blau
```

**Colors.GetRed\_5Bit(VarInt:Ziel,VarInt:Color)**

Mit der Methode `Colors.GetRed_5Bit(Ziel,Color)` wird der 5-Bit Rotanteil einer Farbe in die ZielVariable gelegt. Die Werte der Zielvariablen liegen im Bereich [0..31] liegen.

```
Colors.GetRed(eI.Int_0,eI.FillColor) ; Rotanteil von TextColor nach eI.Int_0
```

**Colors.GetGreen\_5Bit(VarInt:Ziel,VarInt:Color)**

Mit der Methode `Colors.GetGreen_5Bit(Ziel,Color)` wird der 5-Bit Grünanteil einer Farbe in die ZielVariable gelegt. Die Werte der Zielvariablen liegen im Bereich [0..31] liegen.

```
Colors.GetGreen(eI.Int_0,eI.FillColor) ; Grünanteil von TextColor nach eI.Int_0
```

**Colors.GetBlue\_5Bit(VarInt:Ziel,VarInt:Color)**

Mit der Methode `Colors.GetBlue_5Bit(Ziel,Color)` wird der 5-Bit Blauanteil einer Farbe in die ZielVariable gelegt. Die Werte der Zielvariablen liegen im Bereich [0..31] liegen.

```
Colors.GetBlue(eI.Int_0,eI.FillColor)           ; Blauanteil von TextColor nach eI.Int_0
```

## Software Stack

Die virtuelle Maschine verfügt über einen Software-Stack auf den Register, Variablen und Werte gelegt (PUSH) und wieder vom Stack geholt (POP). Es können Words oder Longwords gepush und gepopt werden. Der Stackpointer ist im Register `eI.StackPointer` abgelegt. Da der Software-Stack nicht von der Programmausführung beeinflusst wird (Rücksprung-Adressen von Unterprogrammen werden auf einem anderen, nicht zugänglichen Stack abgelegt), kann der Benutzer den Stack sehr frei verwenden. Der Stackpointer darf beliebig manipuliert werden, wenn dies für die Programmausführung erforderlich ist. Der Stackpointer zeigt auf die nächste freie Speicherzelle und wächst nach oben. Wird mit PUSH ein Element auf dem Stack abgelegt, wird der Stackpointer erhöht und mit POP entsprechend vermindert. Ein Überlaufen und Unterlaufen des Stacks wird überwacht und verursacht keine Speicherverletzung. Für den Stack sind 8kB reserviert, was für 4000 Integer- oder 2000 Single- oder Longwerte reicht.

**PUSH\_Word(VarInt:Quelle)**

Mit der Methode `PUSH_Word(eI.Int_0)` wird die Quelle auf den Stack gelegt. Der Stackpointer `eI.StackPointer` wächst um 2 und zeigt auf den nächsten freien Speicherplatz des Stacks.

```
PUSH_Word(eI.Int_0)                           ; eI.Int_0 auf den Stack legen
```

**POP\_Word(VarInt:Ziel)**

Mit der Methode `POP_Word(eI.Int_0)` wird ein Word vom Stack genommen und in die Zielvariable abgelegt. Der Stackpointer `eI.StackPointer` wird um 2 vermindert und zeigt auf den nächsten freien Speicherplatz des Stacks.

```
POP_Word(eI.Int_0)                            ; eI.Int_0 vom Stack holen
```

Anwendung:

Wenn in einem Unterprogramm oder Event-Handler Register verwendet werden, die nicht zerstört werden dürfen, werden sie mit PUSH und POP gerettet:

```

SUB    Timer_1_Expired
    PUSH_Word(eI.Offset_X)           ; X-Offset auf Stack retten
    PUSH_Word(eI.Offset_Y)           ; Y-Offset auf Stack retten
    eI.Offset_X := 220                ; X1
    eI.Offset_Y := 100                ; Y1
    CallSubroutine(MessageBox1)
    POP_Word(eI.Offset_Y)             ; Y-Offset vom Stack holen
    POP_Word(eI.Offset_X)             ; X-Offset vom Stack holen
ENDSUB

```

Bitte unbedingt die Reihenfolge der Register beachten nach dem First in Last out-Prinzip!!

### PUSH\_Offset ( )

Mit der Methode `PUSH_Offset()` werden die Register `eI.Offset_X` und `eI.Offset_Y` auf den Stack gelegt. Diese Methode löst elegant das Problem, den Offset nicht zu zerstören, wenn in einer Subroutine der Offset verändert wird.

```

PUSH_Offset()           ; eI.Offset_X und eI.Offset_Y auf den Stack legen

```

### POP\_Offset ( )

Mit der Methode `POP_Offset()` werden die Register `eI.Offset_X` und `eI.Offset_Y` vom Stack geholt. Diese Methode löst elegant das Problem, den Offset nicht zu zerstören, wenn in einer Subroutine der Offset verändert wird.

```

POP_Offset()           ; eI.Offset_X und eI.Offset_Y vom Stack holen

```

#### Anwendung:

Die Subroutine kann nun so geschrieben werden:

```

SUB    Timer_1_Expired
    PUSH_Offset()                   ; X-Offset und Y-Offset auf Stack retten
    Load_Offset_XY(220,100)         ; neue Werte für Offset laden
    CallSubroutine(MessageBox1)
    POP_Offset(eI.Offset_X)          ; Y-Offset und X-Offset vom Stack holen
ENDSUB

```

### PUSH\_Geometry ( )

Mit der Methode `PUSH_Geometry()` werden die Register `eI.Pos_X1`, `eI.Pos_Y1` sowie `eI.Width` und `eI.Height` auf den Stack gelegt.

```

PUSH_Geometry()           ; Geometrie-Register auf den Stack legen

```

## POP\_Geometry()

Mit der Methode `POP_Geometry()` werden die Registerel.Pos\_X1, el.Pos\_Y1 sowie el.Width und el.Height vom Stack geholt.

```
POP_Geometry() ; Geometrie-Register vom Stack holen
```

## Class InOut

Die Methoden der Class InOut bedienen die Peripherie des Mikrocomputersystems.

### **InOut.Read\_ADC(VarInt,VarInt)**

Mit der Methode `InOut.Read_ADC(Kanal,ZielVariable)` ein Analog-Kanal eingelesen. Der Wert des Kanals wird in der ZielVariablen gespeichert.

```
InOut.Read_ADC(0,eI.Int_2) ; Analogwandler Kanal 0 in Register eI.Int_2 einlesen
```

## Class Fill/Load/Transfer

Die Methoden der Class Fill dienen zum schnellen Abfüllen von häufig gebrauchten Registern. Wenn ein Label-Objekt vollständig beschrieben werden muss, sind 17 Register zuzuweisen. Da vielfach die Objekteigenschaften von verschiedenen Objekten gleich sind, ist die einfachste Lösung, die Eigenschaften in Structures zusammenzufassen. Eine Änderung kann zentral ausgeführt werden. Es ist auch möglich, die Structures in ein Include-File zu legen, und alle Views, die die Structures benutzen, im Batch zu rekompilieren. Die Structures werden mit dem Pointer, der in der Fill-Methode angegeben ist, adressiert. Die Reihenfolge der Werte in den Structures ist unbedingt zu beachten und muss peinlich genau eingehalten werden!!

Spezielle Codes steuern den Ladevorgang in die Register:

no_change	der Registerwert bleibt unverändert
as_FillColor	der Farbwert wird aus dem Register eI.FillColor übernommen.
as_DisplayColor	der Farbwert wird aus dem Register eI.DisplayColor übernommen (Hintergrundfarbe)
autocolor	der Farbwert wird aus eI.FillColor gelesen und die AutoColor-Funktion für besten Kontrast angewendet. Diese Funktion ist vor allem für die automatische Zuweisung von eI.TextColor interessant.
darken_FillColor	der abgedunkelte Wert von eI.FillColor wird geladen. Besonders interessant für das Register eI.LineColor
brighten_FillColor	der aufgehellte Wert von eI.FillColor wird geladen. Besonders interessant für das Register eI.LineColor

### Fill.LabelParameter(labelRelative24)

Mit der Methode `Fill.LabelParameter(labelRelative24)` werden die Register, die für ein Label-Objekt gebraucht werden, geladen.

#### Beispiel:

```
Fill.LabelParameter(AK1_Key_Structure_UP)      ; Eigenschaften
SUB Structures
AK1_Key_Structure_UP:
    INLINERWORDS (no_change)                  ; entspricht eI.Pos_X1
    INLINERWORDS (no_change)                  ; entspricht eI.Pos_Y1
    INLINERWORDS (AK1_Key_Width)              ; entspricht eI.Width
    INLINERWORDS (AK1_Key_Height)             ; entspricht eI.Height
    INLINERWORDS (8)                          ; entspricht eI.SpaceLeft
    INLINERWORDS (8)                          ; entspricht eI.SpaceRight
    INLINERWORDS (0)                          ; entspricht eI.HorizontalAdjust
    INLINERWORDS (0)                          ; entspricht eI.VericalAdjust
    INLINERWORDS (steelblue)                  ; entspricht eI.FillColor
    INLINERWORDS (as_FillColor)                ; entspricht eI.BackColor
    INLINERWORDS (as_FillColor)                ; entspricht eI.LineColor
    INLINERWORDS (autocolor)                  ; entspricht eI.TextColor
    INLINERWORDS (Pos_center)                 ; entspricht eI.Position
    INLINERWORDS (Orientation_0deg)           ; entspricht eI.Orientation
    INLINERWORDS (normal)                    ; entspricht eI.Appearance
```

```

INLINEWORDS (color_button_3D_raised) ; entspricht eI.BorderStyle
INLINEWORDS (Font_Arial_14n) ; entspricht eI.FontNumber
INLINEWORDS (as_Skin_FormBodyColor) ; entspricht eI.BackgroundColor

```

```
ENDSUB
```

## Class Load

Die Methoden der Class Load dienen zum schnellen Laden von häufig gebrauchten Registern. Dabei sind die Werte als Parameter angegeben. Die Reihenfolge der Parameter ist unbedingt zu beachten. Für Register, die oft gleichzeitig beschrieben werden und logisch zusammengehören, gibt es eine Load-Methode.

### Load.Pos\_X1Y1(VarInt:X1,VarInt:Y1)

Mit der Methode `Load.Pos_X1Y1(x,y)` werden die Register `eI.Pos_X1` und `eI.Pos_Y1` geladen. Die Argumente können Konstanten, Variablen oder Register sein.

```
Load.Pos_X1Y1(50,100) ; eI.Pos_X1 und eI.Pos_Y1 laden
```

ist äquivalent zu.

```
eI.Pos_X1 := 50
eI.Pos_Y1 := 100
```

### Load.Pos\_X2Y2(VarInt:X2,VarInt:Y2)

Mit der Methode `Load.Pos_X2Y2(x,y)` werden die Register `eI.Pos_X2` und `eI.Pos_Y2` geladen. Die Argumente können Konstanten, Variablen oder Register sein.

```
Load.Pos_X2Y2(150,300) ; eI.Pos_X2 und eI.Pos_Y2 laden
```

### Load.Width\_Height(VarInt:W,VarInt:H)

Mit der Methode `Load.Width_Height(W,H)` werden die Register `eI.Width` und `eI.Height` geladen. Die Argumente können Konstanten, Variablen oder Register sein.

```
Load.Width_Height(150,300)           ; eI.Width:= 150, eI.Height := 300
```

## **Load.Geometry\_XYWH(VarInt:X,VarInt:Y,VarInt:W,VarInt:H)**

Mit der Methode `Load.Geometry_XYWH(X,Y,W,H)` werden die Register `eI.Pos_X1` und `eI.Pos_Y1` sowie `eI.Width` und `eI.Height` geladen. Die Argumente können Konstanten, Variablen oder Register sein.

```
Load.Geometry_XYWH(200,30,150,300)   ; eI.Pos_X1:= 200, eI.Pos_Y1 := 30  
                                       ; eI.Width:= 150, eI.Height := 300
```

## **Load.Offset\_XY(VarInt:X,VarInt:Y)**

Mit der Methode `Load.Offset_XY(X,Y)` werden die Register `eI.Offset_X` und `eI.Offset_Y` geladen. Die Argumente können Konstanten, Variablen oder Register sein.

```
Load.Offset_XY(200,30)                ; eI.Offset_X := 200, eI.Offset_Y := 30
```

## **Load.Color\_FL(VarInt:FillColor,VarInt:LineColor)**

Mit der Methode `Load.Color_FL(FillColor,LineColor)` werden die Register `eI.FillColor` und `eI.LineColor` mit einem Farbwert geladen. Die Argumente können Konstanten, Variablen oder Register sein.

```
Load.Color_FL(red,crimson)            ; FüllFarbe und LinienFarbe laden
```

## **Load.Color\_BT(VarInt:BackColor,VarInt:TextColor)**

Mit der Methode `Load.Color_BT(BackColor,TextColor)` werden die Register `eI.BackColor` und `eI.TextColor` mit einem Farbwert geladen. Die Argumente können Konstanten, Variablen oder Register sein.

```
Load.Color_BT(red,white)              ; TextHintergrund- und SchriftFarbe laden
```

## **Load.Data\_from\_ARRAY\_Integer(labelRelative24:ArrayBase,VarInt:Index,VarInt:Ziel)**

Mit der Methode `Load.Data_from_ARRAY_Integer(ArrayBase,Index,Ziel)` wird aus einem ARRAY, auf das das Label zeigt, ein Datenwert, der mit dem Integer Index adressiert wird, ausgelesen.

```
Store.Data_to_ARRAY_Integer(labelRelative24:ArrayBase,
VarInt:Index,VarInt:Wert)
```

Mit der Methode `Store.Data_to_ARRAY_Integer(ArrayBase,Index,Wert)` wird in ein ARRAY, auf das das Label zeigt, ein Datenwert, der mit dem Integer Index adressiert wird, geschrieben.

## Class Transfer

Die Methoden der Class Transfer dienen zum schnellen Laden oder verschieben von häufig gebrauchten Registern. Die Werte sind dabei schon in Register vorhanden und werden in andere Register umgeladen. Ebenfalls in der Klasse Transfer finden sich Methoden, mit denen Datentypen in andere, gleich lange Datentypen geladen werden.

```
Transfer.HotSpotGeometry()
```

In einem HotSpot-Tabelleneintrag sind alle Geometriedaten wie Position, Grösse und Offset gespeichert. Zum Neuzeichnen des Buttons (Animation) sind diese Geometriedaten von grösster Nützlichkeit. Mit der Methode `Transfer.HotSpotGeometry()` werden die Geometrieregister mit den Rückgabewerten eines HotSpots geladen. Der Befehl erledigt die folgende Sequenz von Befehlen mit einem Schlag:

```
eI.Width      := eI_HS_Width      ; Breite des HotSpot
eI.Height     := eI_HS_Height     ; Höhe des HotSpot
eI.Pos_X1    := eI_HS_Pos_X      ; X-Position des HotSpot
eI.Pos_Y1    := eI_HS_Pos_Y      ; Y-Position des HotSpot
eI.Offset_X  := eI_HS_Offset_X   ; X-Offset des HotSpot
eI.Offset_Y  := eI_HS_Offset_Y   ; Y-Offset des HotSpot
```

Dieser Befehl wird vor allem bei der Implementierung von lageunabhängigen Controls verwendet.

## Programmfluss-Steuerung

Für die Programmflusssteuerung stehen Sprungbefehle zur Verfügung. Neben dem unbedingten Sprung, der immer ausgeführt wird, gibt es noch die bedingten Sprünge, die eine Bedingung prüfen und je nachdem, ob die Bedingung wahr oder falsch ist, springen oder nicht.

Achtung: Sprünge dürfen nur zu Sprungzielen innerhalb der gleichen Unterprogramm-Ebene führen!

### Jump(labelRelative24)

Der unbedingte Sprung `Jump(Label)` lässt die Programmausführung bei Label weiterfahren.

```
Jump(Weiter)           ; nach Weiter springen
```

Weiter:

### Jump\_IF\_Integer\_Zero(labelRelative24,VarInt)

Der bedingte Sprung `Jump_IF_Integer_Zero(Label,MyInteger)` prüft MyInteger auf Null und springt zu Label, falls MyInteger Null ist.

```
Jump_IF_Integer_Zero(Weiter,MyInteger)           ; nach Weiter springen, wenn  
                                                    ; MyInteger Null ist.
```

Weiter:

### Jump\_IF\_Integer\_NotZero(labelRelative24,VarInt)

Der bedingte Sprung `Jump_IF_Integer_NotZero(Label,MyInteger)` prüft MyInteger auf ungleich Null und springt zu Label, falls MyInteger ungleich Null ist.

```
Jump_IF_Integer_NotZero(Weiter,MyInteger)           ; nach Weiter springen, wenn  
                                                    ; MyInteger ungleich 0 ist.
```

Weiter:

### ON\_ERROR\_GOTO(labelRelative24)

Der bedingte Sprung `ON_ERROR_GOTO(Label)` prüft das Register `el.Status` und springt, falls der Wert `error` ist.

```
ON_ERROR_GOTO(Fehlerbehandlung) ; falls Fehler, Fehlerbehandlung  
                                ; MyInteger ungleich 0 ist.
```

Weiter:

### Jump\_IF\_true(labelRelative24)

Der bedingte Sprung `Jump_IF_true(Label)` prüft, ob nach einer Operation, die das Ergebnis im Register `el.Boolean` speichert, `el.Boolean` den Wert `true` hat. Falls ja, wird der Sprung ausgeführt.

### Jump\_IF\_false(labelRelative24)

Der bedingte Sprung `Jump_IF_false(Label)` prüft, ob nach einer Operation, die das Ergebnis im Register `el.Boolean` speichert, `el.Boolean` den Wert `false` hat. Falls ja, wird der Sprung ausgeführt.

## Fehlerbehandlung

Verschiedene Methoden der eVM können Fehler erzeugen, weil beispielsweise InputParameter nicht gültig sind oder die Methode nicht mit dem gewünschten Resultat beendet werden kann. Beispielsweise Division durch 0. Für die Rückgabe des Fehlerresultats sind zwei eVM-Register zuständig:

`eI.Status` genereller Status: error/success

wenn `eI.Status` error liefert, wird im Register `eI.ErrorCode` der entsprechende Fehlercode zurückgeliefert. Der Fehlercode liefert Information zum Grund des Fehlers. z.B. die angeforderte Stringposition zeigt über das Stringende hinaus.

Für die Abfrage eines Fehlers existiert eine spezielle Sprungfunktion: Mit `ON_ERROR_GOTO(Label)` wird nach Label gesprungen, falls ein Fehler aufgetreten ist. Wenn nach einer Gruppe von Befehlen auf Fehler überprüft werden soll, wird `eI.status` mit success vorgeladen. Tritt ein Fehler auf, kann das Register `eI.ErrorCode` ausgewertet werden.

Die Fehler setzen das `eI.status` auf false, aber nicht auf success. Das bedeutet, dass man vor einer Funktion, deren Fehlerstatus überwacht werden soll, `eI.status` mit success vorladen muss. Danach kann eine Sequenz von Befehlen abgearbeitet werden und am Ende der Sequenz kann der Status überprüft werden. Mit diesem Feature kann einerseits der Code, der im Normalfall abgearbeitet wird, beisammen gelassen werden und andererseits wird die Fehlerbehandlung erheblich vereinfacht.

## Debugger

Die eVM hat einen Debugger eingebaut, mit dem Fehler gefunden werden können. Zusammen mit der Klasse Debug können Fehler gesucht werden.

- |      |                |   |
|------|----------------|---|
| T:   | Trace ON       | Der MDEC (mainDecoder) gibt TraceCode aus, mit dem der Verlauf der Instruktionen mitverfolgt werden kann.               |
| t:   | Trace OFF      | keine Trace Ausgabe   |
| S:   | SingleStep ON  | Der SingleStep-Modus wird aktiviert. Nach jeder Instruktion wartet die EVM auf eine Eingabe an der UART1-Schnittstelle. |
| s:   | SingleStep OFF |   |
| ESC: | SingleStep OFF | Der SingleStep-Modus wird verlassen.  |

## Arbeiten mit CSV-Files

Was sind CSV-Files ?

CSV ist die Abkürzung für comma separated value-File. Grundsätzlich ist ein CSV-File ein ASCII (Text-) File, das mit dem Editor, oder mit dem WordPad, Word oder einem anderen Texteditor bearbeitet werden kann. Die Endung ist meistens \*.CSV. Ein CSV-File besteht aus einzelnen Datenzeilen, in denen die Spalten durch ein Trennzeichen getrennt sind. Das Trennzeichen ist in der von uns verwendeten Variante das Semikolon (;) 0x3B. Wenn in Excel ein File gespeichert wird, trennt Excel die Spalten durch ein Semikolon. Der Zeilenabschluss bildet ein CR LF (0x0A,0x0D). Das CSV-Format ist ein nicht standardisiertes Format, bei der Implementation hielten wir uns deshalb an eine möglichst hohe Kompatibilität zu Excel.

Die eigerVM stellt Methoden zur Verfügung, CSV-Files als kleine Datenbanken zu verwenden, indem Werte aus den CSV-Files gelesen werden können und Werte im CSV-File gesucht werden können.

Die Klasse Files stellt die Methode zum Lesen des CSV-Files zur Verfügung:

```
STRING          [40000] MatchFile = ''  
  
    File.Read_CSV('C:\\TG12\\DATA\\COLORS.CSV',MatchFile)
```

MatchFile wird als String mit 40'000 Zeichen gewählt. Wenn das File kleiner ist, kann die Grösse entsprechend kleiner gewählt werden, die FileGrösse darf jedoch 65'000 Zeichen nicht überschreiten, sonst wird das File nicht vollständig eingelesen.

Beim Einlesen des Files wird das File analysiert, und die Anzahl Spalten und die Anzahl Zeilen gespeichert.

Beispiel eines CSV-Files mit frei erfundenen Testadressen. Excel speichert die Eingaben genau so und liest die Daten auch wieder problemlos ein. Das Einzige was Excel nicht weiss, ist die Breite der Spalte, da das CSV-File nur die Tabelleneinträge, nicht aber die Formatierung beinhaltet.

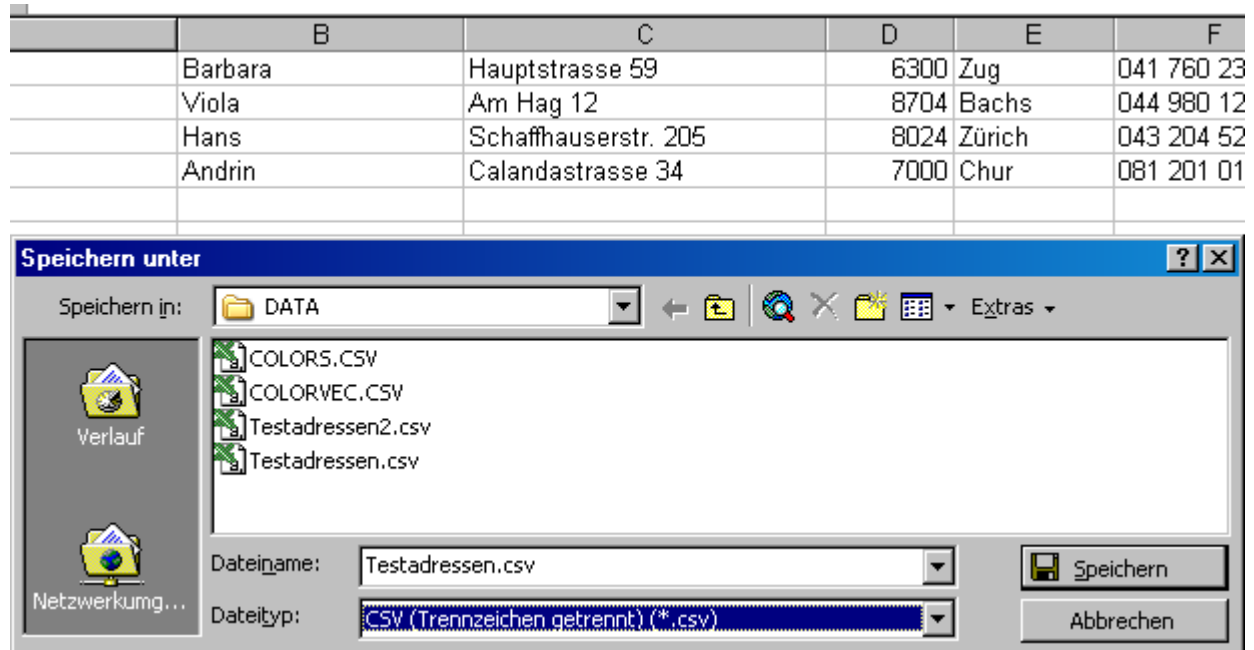
Viele Programme können Daten in CSV exportieren oder importieren. Das Format ist viel kompakter als XML, da nur die rohen Daten, aber keine öffnenden und schliessenden Tags vorhanden sind. Die XML-Files sind um einen Faktor 9-80 grösser als entsprechende CSV-Files. Auf Systemen mit beschränkten Ressourcen ist CSV die beste Wahl.

Allenfalls kann in der ersten Zeile eine Art Titelzeile enthalten sein, die die Spaltenüberschrift enthält. Die Zeile wird von der eigerVM ganz normal als erste Zeile behandelt.

```
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
```

Das CSV-File stellt den Datenaustausch zum PC (Excel) sicher.

TIPP: In Excel File speichern unter Dateityp CSV(Trennzeichen getrennt) \*.CSV auswählen, damit die Sonderzeichen korrekt gespeichert werden.



Einstellungen zum Speichern eines CSV-Files in Excel.

### **CSV.GetMax\_Lines (VarInt:Zeilen,VarStr:CSV\_File)**

Die Methode liefert die Anzahl Linien, die beim Einlesen eines CSV-Files geladen wurden.

```
CSV.GetMax_Lines (MatchFile_LineNoMax,MatchFile)
```

### **CSV.GetMax\_Columns (VarInt:Spalten,VarStr:CSV\_File)**

Die Methode liefert die Anzahl Spalten, eines CSV-Files (Strings). Die Anzahl Spalten eines einzeiligen CSV-Strings kann ebenfalls bestimmt werden.

```
CSV.GetMax_Columns (MatchFile_ColumnsMax,MatchFile)
```

```
TestString2 := 'Aarau;Bern;Chur;Dorf;Eglisau;Flaach;Genf;Hinwil;Illnau;Jona\r\n')
CSV.GetMax_Columns(ColumnsMax,TestString2)
```

liefert ColumnsMax = 10 für die zehn Ortsnamen im String  
Daten aus Tabellen lesen

**CSV.GetFieldData(VarStr:OutputString,VarStr:Matchfile,  
VarInt:Spalte,VarInt:Zeile)**

Die Methode liefert den String, der in der entsprechenden Spalte und Zeile steht. Als Beispiel stehen in der fünften Spalte die Wohnorte.

```
CSV.GetFieldData(OutputString,MatchFile,Spalte,Zeile)
```

wenn das MatchFile die Adresstabelle ist, liefert die Methode

```
CSV.GetFieldData(OutputString,MatchFile,5,2)
```

als OutputString den Wohnort (5.Spalte) der 2. Zeile : OutputSting = 'Bachs'

entsprechend liefert der Aufruf der untenstehenden Methode

```
CSV.GetFieldData(OutputString,TestString2,7,1)
```

als OutputString den Ort (7.Spalte) der CSV-Zeile : OutputSting = 'Genf'

Daten in Tabellen suchen

**CSV.Find\_in\_Column(VarStr:CSV-String,VarInt:StartZeile,  
VarInt:Spalte,VarStr:MatchString,VarInt:ZeilenNummer,  
VarStr:DatenZeile)**

Die Methode sucht den den MatchString, der in der entsprechenden Spalte. Im Beispiel der Adresstabelle kann nach einem Ort, nach dem Vornamen etc. gesucht werden, je nach dem, welche Spalte angegeben wird. Die Methode durchsucht ab der StartZeile bis ein Eintrag, der passt gefunden wurde. Ist dies der Fall, wird das Register el.Boolean auf true gesetzt, andernfalls auf false. Der Matchstring kann nur aus wenigen Anfangsbuchstaben bestehen. Der erste Fundort, der passt wird als Zeilennummer zurückgeliefert, ebenfalls die ganze Datenzeile, damit nicht nocheinmal auf das möglicherweise lange CSV-File zugegriffen werden muss.

Die Methode wandelt intern die Zeichen in Kleinbuchstaben um, so dass die Gross-/Kleinschreibung keine Rolle mehr spielt.

```
CSV.Find_in_Column(MatchFile,1,3,'sch',LineNo,TestString)
```

wenn das MatchFile die Adresstabelle ist, liefert die Methode als LineNo 3 zurück (dritte Zeile) und als TestString:

---

Berger;Hans;Schaffhauserstr. 205;8024;Züich;043 204 52 67;haberzu@green.ch

Der Matchstring hat bei der Schaffhauserstr. eine Übereinstimmung gefunden.  
mit der Methode `CSV.GetFieldData` kann anschliessend auf die einzelnen Datenfelder zugegriffen werden.

TIPP: mit einem Sprung `Jump_IF_false(LabelNoMatch)` kann unmittelbar nach der Suche weggesprungen werden, falls kein Suchergebnis vorliegt. Entsprechend natürlich auch mit `Jump_IF_true(LabelMatch)` an den Code gesprungen werden, der eine erfolgreiche Suche abhandelt.