



Application Note 21

Erstellt: 05.08.2011, Christoph Angst, S-TEC electronics AG
Update:



S-TEC electronics AG

Gewerbestrasse 6, CH-6314 Unterägeri, eiger@s-tec.ch, www.eigergraphics.com

Inhalt

Inhalt	2
Einleitung	3
Daten an CSV-File anfügen: File.AppendString()	5
Grund-Applikation CSV0: Messwert anzeigen.....	5
Beschreibung von CSV0	5
Programmcode von CSV0.....	7
Beispiel-Applikation CSV1: Messwert als neuen Datensatz an bestehende Datei anfügen.....	9
Beschreibung von CSV1	9
Vorbereiten einer CSV-Datei für die Beispiel-Applikation CSV1	10
Programmcode von CSV1.....	11
CSV-File generieren: File.Write_TextFile()	14
Beispiel-Applikation CSV2: während Laufzeit CSV-File generieren.....	14
Beschreibung von CSV2	14
Programmcode von CSV2.....	15
CSV-File analysieren	20
Beschreibung von CSV3	22
Programmcode von CSV3.....	24
File.Read_CSV(PathANDName.\$,CSV_String.\$).....	24
CSV.GetMax_Columns(MaxColumns.I,CSV_String.\$)	24
CSV.GetMax_Lines(MaxLines.I,CSV_String.\$).....	25
CSV.DataFieldLength(MyFieldLength.I,CSV_String.\$,LineNumber.I,ColumnNumb er.I).....	25
CSV.Find_in_Column(CSV_String.\$,LineNumber.I,ColumnNumber.I,MatchString. \$,LineNoFound.I,DataLine.\$).....	27
CSV.Get_Integer(MyInteger.I,CSV_String.\$,LineNumber.I,ColumnNumber.I)	28
CSV.Get_Long(MyLong.L,CSV_String.\$,LineNumber.I,ColumnNumber.I).....	29
CSV.Get_LongDeci(MyLongDeci.L,CSV_String.\$, LineNumber.I,ColumnNumber.I,Nachkomma.I).....	30
CSV.Get_HighColor(MyColor.I,CSV_String.\$, LineNumber.I,ColumnNumber.I)	32
CSV.Get_ByteHex(MyByteHex.I,CSV_String.\$, LineNumber.I,ColumnNumber.I)	33
CSV.Get_WordHex(MyWordHex.I,CSV_String.\$, LineNumber.I,ColumnNumber.I)	34
CSV.Get_LongHex(MyLongHex.L,CSV_String.\$, LineNumber.I,ColumnNumber.I)	35
CSV.Get_String(MyString.\$,CSV_String.\$,LineNumber.I,ColumnNumber.I)	37
CSV.Put_String(MyString.\$,CSV_String.\$,LineNumber.I,ColumnNumber.I).....	38

■ Einleitung

Mit eigerScript-Befehlen können Dateien während Laufzeit erstellt, gelöscht, analysiert, gelesen und beschrieben werden. Es handelt sich dabei um formatfreie Dateien, wie *.txt oder *.csv.

Daten während der Laufzeit in einer CSV-Datei abzuspeichern bzw. von dort einzulesen ist für viele Touchpanel-Anwendungen von grosser Bedeutung. Die Vorteile dieser Möglichkeit liegen auf der Hand:

- In CSV-Dateien können beispielsweise Messdaten oder Einstellungen so abgespeichert werden, dass diese bei einem Stromausfall oder Systemabsturz nicht verloren gehen.
- Mittels CSV-Dateien kann eine Benutzerführung in verschiedenen Sprachen angezeigt werden.
- Je nach Situation kann ein Link zu einem passenden Bild aufgerufen werden.
- Manche nutzen CSV-Tabellen auch anstelle von Arrays.
- Mit dem Zugriff auf CSV-Tabellen können Einstellungen so gespeichert werden, dass sie einen System-Neustart oder einen Stromausfall überdauern.

Die vorliegende Application Note zeigt, wie die CSV-Befehle in eigerScript genutzt werden und enthält lauffähige Beispiele, die Sie auf Ihrem eigerPanel anzeigen können. In den ersten Seiten (S. 5 ff) wird die „Grund-Anwendung CSV0“ beschrieben, welche für die eigentlichen CSV-Beispiele als Basis dient. Wenn Sie mit eigerScript bereits vertraut sind, können Sie diese Kapitel verlustlos überspringen.

Zu dieser Application Note gehören die Beispielanwendungen CSV0, CSV1, CSV2, CSV3. Diese können Sie von der Download-Seite www.eigergraphics.com/download.htm herunterladen. Kopieren Sie die gleichnamigen Verzeichnisse auf Ihre CompactFlash Card sehen Sie es sich auf Ihrem eigerPanel an. Das Layout der Beispielanwendungen ist auf die VGA-

Auflösung des eigerPanel57 ausgelegt. Auf dem eigerPanel70 mit WVGA-Auflösung ist deshalb nicht das ganze Bild ausgefüllt.

Auch das Senden oder Empfangen von binären Daten, z.B. Text- und CSV-Dateien oder Bildern, ist möglich. Dies geschieht beim eigerPanel über die RS232 Schnittstelle COM2. Der Dateitransfer basiert auf YMODEM.

■ Daten an CSV-File anfügen: `File.AppendString()`

Wenn Sie bei Laufzeit einer CSV-Datei neue Datensätze anfügen möchten und die entsprechende Datei bereits auf der CompactFlash Card (CFC) existiert, benützen Sie den Befehl

```
File.AppendString(MyLogFile.$,CSV-Datensatz.$)
```

Mit diesem Befehl wird der Inhalt des Strings (hier mit dem Namen `CSV-Datensatz.$`) am Ende der Datei angefügt. Das ist praktisch, wenn Sie z.B. in einem zuvor oder auch während der Laufzeit erstellten LogFile über längere Zeit kontinuierlich Ihre Messwerte aufzeichnen wollen (vgl. eigerScript-Code 6). Wie dieser Befehl im Kontext eingesetzt wird, soll anhand des nachfolgenden Beispiels (CSV 1) gezeigt werden.

Downloadseite für
die eigerPanel-
Projekte CSV0 bis
CSV3:
www.eigergraphics.com/applicationnotes.htm

Grund-Applikation CSV0: Messwert anzeigen

Für die nachfolgenden Beispiele CSV1 bis CSV3 dient die Grund-Applikation CSV0 als Vorlage. Diese finden Sie als Beilage zu dieser Application Note. Laden Sie diese auf Ihre CompactFlash Card und starten Sie das Projekt in Ihrem eigerPanel.

Beschreibung von CSV0

Auf dem Display des eigerPanels erscheint eine Anwendung mit einem blauen Button „Messwert aufzeichnen“ und einer grünen Messwertanzeige (vgl. Abbildung 1).

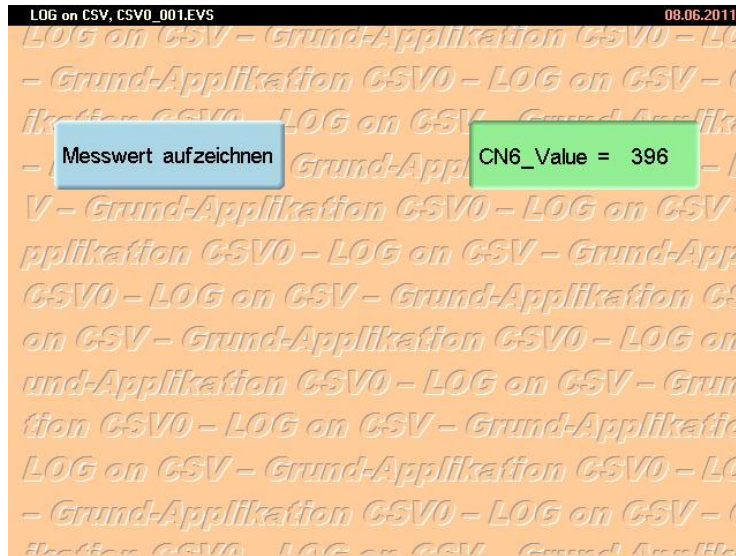


Abbildung 1: PrintScreen der Display-Anzeige des eigerPanel-Projekts CSV1.

Schliessen Sie ein Potentiometer (Teil des eigerPanel StarterKits) an den Stecker des Analog-Eingangs CN6 (vgl. Abbildung 2). Der angezeigte Messwert entspricht der aktuellen Position des angeschlossenen Potentiometers und kann Werte von 0 bis 1023 annehmen (0..3.3V).

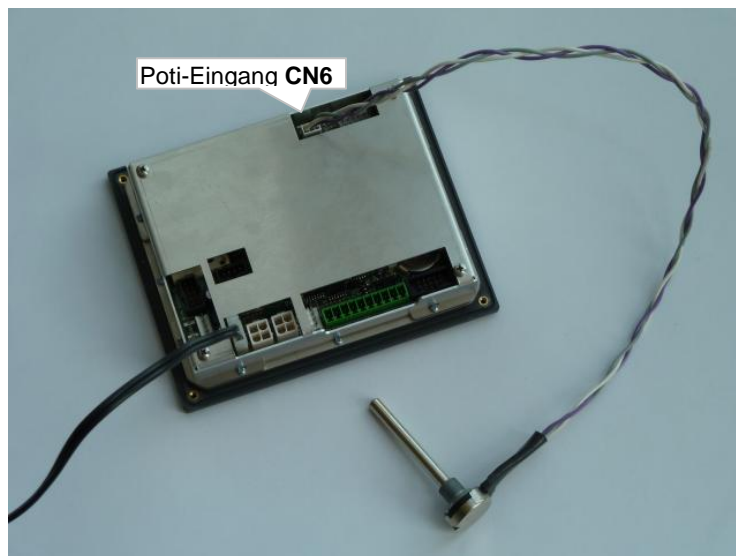


Abbildung 2: Rückseite des eigerPanel57H. Das Potentiometer ist am Analog-Input CN6 angeschlossen.

In der Grund-Applikation CSV0 ist der Button wohl animiert, aber noch mit keiner Funktion ausprogrammiert.

Programmcode von CSV0

Die wesentlichsten Komponenten der Grund-Applikation CSV0 sind der animierte Button, das Anzeigefeld für den Messwert am Potentiometer-Eingang CN6 sowie ein „Continuous Timer“, welcher für die kontinuierliche Abfrage des Messwerts sorgt.

Von den Konstanten und Variablen, welche am Anfang des Programm-Codes definiert sind, seien hier nur zwei Variablen speziell erwähnt: **CN6_Value.I** und **CN6_Value.\$** (vgl. eigerScript-Code 1).

Die **Integervariable CN6_Value.I** benötigen wir für die das eigentliche Auslesen des Integer-Wertes (0..1023) am Analog-Eingang CN6. Diesen Wert müssen wir dann in das String-Format konvertieren, bevor wir damit dann den neuen Datensatz mit „Time“ und „Value“ generieren können.

Die **Stringvariable CN6_Value.\$** benötigen wir nicht unbedingt für das Abspeichern des Messwertes in die CSV-Datei. Sie dient uns lediglich für die Anzeige des Messwertes auf dem Display.

eigerScript-Code 1: Variablen für Messwert-Abfrage beim Analog-Input CN6

```
INTEGER    CN6_Value.I
STRING [20] CN6_Value.$
```

Der **Button** ist mit einem HotSpot hinterlegt, der auf drei Touch-Events, den Down-, Leave- und Up-Event, reagiert. Entsprechend dienen im Programmcode drei Subroutinen als sogenannte „Event-Handler“ (eigerScript-Code 2).



Die Subroutine **B01_Leave** ist für nichts anderes zuständig, als den Button in der Ruhestellung zu zeichnen. Wenn nach einem Down-Event anstelle eines Up-Events ein Leave-Event eintritt, tritt diese Subroutine in Aktion. Sie wird auch beim Einlesen der View für das Zeichnen des Buttons aufgerufen.

Mit der Subroutine **B01_Down** wird der Down-Zustand des Buttons gezeichnet. Das geschieht, wenn jemand auf den Button drückt, d.h. ein Down-Event provoziert.

Manchmal wird in dieser Subroutine zudem auch eine bestimmte Aktion ausgelöst, i.d.R. durch Aufrufen einer weiteren Subroutine.

Die Subroutine `B01_Up` sorgt in unserer Grund-Applikation vorerst nur dafür, dass der Button wieder in die Ruhestellung „zurückspringt“. Das geschieht ganz einfach durch Aufruf der Subroutine `B01_Leave`. Später werden wir mit dieser Subroutine auch weitere Aktionen auslösen, beispielsweise die Aufzeichnung des Messwerts in einer CSV-Datei.

eigerScript-Code 2: Subroutinen für die Darstellung des animierten Buttons B01 mit der Aufschrift „Messwert aufzeichnen“.

```

STRING      [32] B01_Text = 'Messwert aufzeichnen' ; String-Deklaration
für Button-Aufschrift

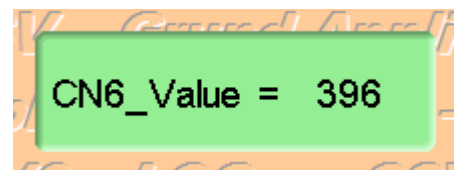
SUB B01_Leave ; Sub wird aufgerufen bei View-Start u Leave-Event
Fill.LabelParameter(Button_UP_Style) ; unberührter Button
Load.Geometry_XYWH(ButtonLeft_X,ButtonRow01_Y,ButtonWidth,ButtonHeight)
Label.Text(B01_Text) ; Button mit Aufschrift zeichnen
ENDSUB

SUB B01_Down ; Subroutine wird aufgerufen bei Down-Ereignis
Fill.LabelParameter(Button_DN_Style) ; gedrückter Button
Load.Geometry_XYWH(ButtonLeft_X,ButtonRow01_Y,ButtonWidth,ButtonHeight)
Label.Text(B01_Text) ; Button mit Aufschrift zeichnen
ENDSUB

SUB B01_Up ; Subroutine wird aufgerufen bei Up-Ereignis
CallSubroutine(B01_Leave) ; Button im Up-Zustand zeichnen
; CallSubroutine(...) ; Aktion aufrufen
ENDSUB

```

Das Anzeigefeld wird mit der Subroutine `ShowValue` dargestellt (eigerScript-Code 3). Zuerst wird mit einem einzigen eigerScript-Befehl der Messwert am Analog-Eingang abgeholt und in eine Integer-Variable eingelesen.



eigerScript-Code 3: Subroutine für die Ermittlung des Potentiometer-Wertes des Analog-Eingangs CN6 und für die Darstellung des Anzeige-Feldes.

```

SUB ShowValue
Display.Prepare()

```

```
; Abfrage des Analog-Eingangs CN6 (= Channel 0):
InOut.Read_ADC(0,CN6_Value.I) ; Wert (0..1023) in CN6_Value.I einlesen

; Integer in String umwandeln:
Str.Clear(CN6_Value.$) ; Stringinhalt löschen für neuen Wert
CN6_Value.$ := 'CN6_Value = ' ; erster Teil des Strings
Str.Cvt_Integer(CN6_Value.$,CN6_Value.I,4) ; Konvertieren

; Anzeige-Feld zeichnen:
Fill.LabelParameter(Button_DN_Style) ; Style für Anzeigefeld
Load.Geometry_XYWH(ButtonRight_X,ButtonRow01_Y,ButtonWidth,ButtonHeight)
Label.Text(CN6_Value.$) ; Wert im Display anzeigen
Display.ShowWindow()
ENDSUB
```

Der **Continuous-Timer** wird in CSV0 mit Hilfe der Subroutine **Timer_GetValue** installiert, geladen und gestartet (eigerScript-Code 4). Diese Timer-Subroutine wird beim View-Start vom Hauptprogramm aus aufgerufen. Der Timer ruft alle 500 Millisekunden die Subroutine **ShowValue** auf, so dass kontinuierlich das Anzeigefeld mit dem aktuell ausgelesenen Messwert neu dargestellt wird.

eigerScript-Code 4: Timer-Subroutine für die kontinuierliche Abfrage und Anzeige des Messwerts.

```
SUB Timer_GetValue
Timer.InstallLocal(0,ShowValue) ; Timer 0 wird installiert, bei
Timer-Ende wird "SUB GetValue" aufgerufen
Timer.Load(0,500) ; Timer-Ende nach 500 msec = 1/2 sec
Timer.StartContinuous(0) ; Timer starten als Continuous-Timer
ENDSUB
```

Beispiel-Applikation CSV1: Messwert als neuen Datensatz an bestehende Datei anfügen

Beschreibung von CSV1

Jedes Mal, wenn der Button „Messwert aufzeichnen“ betätigt wird, soll beim Up-Event der aktuell angezeigten Potentiometer-Wert (z.B. 396) zusammen mit der aktuellen Uhrzeit in eine CSV-Datei abgespeichert werden. Die CSV-Datei „LogFile.CSV“ liegt vorbereitet auf der CompactFlash-Card im Verzeichnis CSV1\HOST. Abbildung 3 zeigt den Inhalt von „LogFile.CSV“ nachdem der Button fünf Mal gedrückt wurde. Es ist eine Tabelle mit den beiden Spaltenüberschriften „Time“ und „Value“ sowie den sieben aufgezeichneten Datensätzen. Die Felder einer Zeile sind mit Semikolons getrennt. Jeder neue Datensatz wird in dieser CSV-Tabelle als neue Zeile unten angefügt.

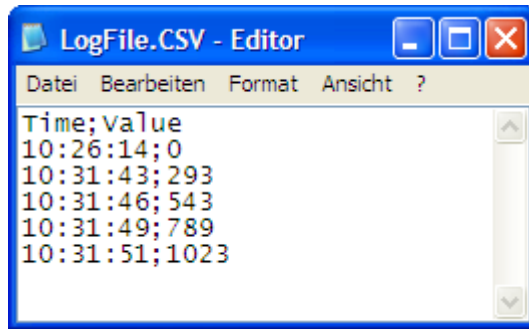


Abbildung 3: PrintScreen eines Text-Editors mit dem Inhalt von „LogFile.CSV“ nach Aufzeichnung einiger Messwerte. Ein neuer Datensatz wird in der Tabelle als neue Zeile unten angefügt.

Vorbereiten einer CSV-Datei für die Beispiel-Applikation CSV1

Auf der CompactFlash Card haben wir im Verzeichnis CSV1\HOST eine CSV-Datei mit dem Namen „LogFile.CSV“ (max. 8 Zeichen für den Filenamen) vorbereitet.

Die Tabellenüberschriften unserer CSV-Tabelle sind „Time“ und „Value“:

Time	Value

Unsere CSV-Datei erstellen wir wie folgt:

Wir öffneten einen Texteditor – z.B. den Editor von Microsoft – und schrieben die Zeile

Time;Value

Eine CSV-Tabelle ist formatfrei; die Spalten werden deshalb in unserem LogFile mit Semikolons (;) voneinander getrennt. Mit Return eröffneten wir eine zweite Zeile (Abbildung 4). Diesen Text mit Zeilenwechsel speicherten wir als formatfreie Datei mit dem Namen „LogFile.CSV“ ins Verzeichnis „HOST“ des Projekts CSV1.



Abbildung 4: CSV-Datei „LogFile.CSV“, die wir mit einem einfachen Text-Editor vorbereitet haben. Sie enthält die Zeilen-Überschriften „Time“ und „Value“ und danach ein Return, so dass der erste Datensatz in eine neue Zeile geschrieben wird.

Programmcode von CSV1

Wir benutzen die „Grund-Applikation CSV0“ als Vorlage. Dazu kopieren wir das ganze Verzeichnis „CSV0“ und geben der Kopie neu die Bezeichnung CSV1. Natürlich müssen wir diese Umbenennung auch bei den Dateinamen und innerhalb des Programm-Codes durchziehen.

1. Definieren wichtiger Variablen

Zunächst definieren wir im Deklarationen-Teil des Sourcecode CSV1_001.EVS einige String- und Integervariablen (vgl. eigerScript-Code 5):

eigerScript-Code 5: Deklarationen von Stringvariablen für den Namen eines LogFiles und für den jeweiligen Mess-Datensatz.

```
; CSV-Deklarationen:  
STRING [50]      CSV_PathAndName.$ = 'C:/CSV1/HOST/LogFile.CSV'  
STRING [15]     CSV_Datensatz.$ = ''
```

Die Stringvariable **CSV_PathAndName.\$** enthält den Pfad unserer CSV-Datei „LogFile.CSV“ auf der CompactFlash Card.

Die Stringvariable **CSV_Datensatz.\$** soll jedes Mal, wenn der blaue Button gedrückt wird, mit dem neuen Datensatz eingelesen werden. Der neue Datensatz besteht jeweils aus der aktuellen Uhrzeit und dem zu diesem Zeitpunkt ausgelesenen Messwert. Diesen Datensatz können wir dann der CSV-Tabelle „LogFile.CSV“ auf der CompactFlash anfügen.

2. Datensatz generieren und an die CSV-Tabelle anfügen

Durch den Programm-Code der Grund-Applikation ist bereits dafür gesorgt, dass der Messwert vom Analog-Input CN6 kontinuierlich neu eingelesen und in der Integervariable **CN6_Value.I** für die weitere Auswertung bereitgelegt wird (vgl. eigerScript-Code 3, S.8), z.B. für dessen Anzeige auf dem Display oder eben auch für die bleibende Speicherung des Messwertes in einer CSV-Tabelle.

Mit Hilfe der Subroutine **SUB LogValue** der wird der eingelesene Messwert in der CSV-Tabelle „LogFile.CSV“ aufgezeichnet. Dazu wird zuerst schrittweise ein Datensatz, d.h. eine neue Tabellen-Zeile aufgebaut, welcher als erstes Spaltenfeld

die aktuelle Uhrzeit enthält und dann im zweiten Spaltenfeld den Messwert zugewiesen erhält. Der neuen Datenzeile wird abschliessend noch ein CRLF (neue Zeile) angefügt. Damit ist gewährleistet, dass ein allenfalls nachfolgender Datensatz – beim erneuten Drücken des Buttons – auch wieder in eine neue Zeile geschrieben wird.

Danach genügt ein Befehl, um den neu generierten Datensatz der CSV-Tabelle anzufügen:

```
File.AppendString(CSV_PathAndName.$,CSV_Datensatz.$).
```

eigerScript-Code 6: Subroutine erstellt Datensatz und fügt diesen als weitere Zeile der CSV-Tabelle im LOG-File 'LOG_01.CSV' auf der CFC an.

```
SUB LogValue
  CSV_Datensatz.$ := ' ' ; String für neuen Datensatz leeren
  Str.Time(CSV_Datensatz.$, FormatTime_HH_MM_SS) ; 1. Feld „Time“
  Str.Concat(CSV_Datensatz.$, ';') ; Trennzeichen 1. und 2. Feld
  Str.Cvt_Integer(CSV_Datensatz.$, CN6_Value.I, 4) ; Messwert
  Str.Add_CRLF(CSV_Datensatz.$) ; CRLF als Zeilen-Abschluss
  File.AppendString(CSV_PathAndName.$,CSV_Datensatz.$) ; Datensatz an
  das CSV-File auf der CFC anfügen
ENDSUB
```

Mit einem entsprechenden Subroutinen-Aufruf beim Up-Event des Buttons „Messwert aufzeichnen“ sorgen wir dafür, dass die Aufzeichnung beim Betätigen des Buttons auch ausgeführt wird.

eigerScript-Code 7: Button-Subroutine für Up-Ereignis mit Aufruf der Subroutine **LogValue**, welche den Messwert in die CSV-Tabelle 'LOG_01.CSV' aufzeichnet (vgl. eigerScript-Code 6).

```
SUB B01_Up ; Subroutine wird aufgerufen bei Up-Ereignis
  CallSubroutine(B01_Leave) ; Button im Up-Zustand zeichnen
  CallSubroutine(LogValue) ; Messwert in CSV-Tabelle aufzeichnen
ENDSUB
```



Wir könnten den Subroutinen-Aufruf `CallSubroutine(LogValue)` auch am Ende der Subroutine `ShowValue` (vgl. eigerScript-Code 3, S.8) platzieren. In diesem Fall würde bei jedem Ablauf des ContinuousTimers zusammen mit dem Refresh der Messwert-Anzeige der entsprechende Wert automatisch auch in der CSV-Tabelle 'LOG_01.CSV' aufgezeichnet.



Das Aufzeichnen eines Datensatzes auf die CompactFlash Card (CFC) dauert eine gewisse Zeit; in unserem Beispiel etwa 2 Sekunden(!). Oft gibt es Applikationen, bei welchen diese Zeit für eine Aufzeichnung auf die CFC nicht zur Verfügung steht. In diesen Fällen müssen die Datensätze im RAM

behalten und beispielsweise mit `Str.Concat()` nacheinander an einen String angefügt werden. Dieser String kann dann später oder von Zeit zu Zeit als ganzes in eine CSV-Datei auf die CFC übertragen werden.

■ CSV-File generieren: `File.Write_TextFile()`

eigerScript ermöglicht auch die Bildung eines CSV- oder TXT-Files während der Laufzeit. So kann z.B. in der Applikation ein Event genutzt werden, um bei Laufzeit ein neues LOG-File zu bilden, etwa bei Betätigung eines Buttons oder bei Ablauf eines Timers etc.

Beispiel-Applikation CSV2: während Laufzeit CSV-File generieren

Beschreibung von CSV2

Die „Beispiel-Applikation CSV2“ enthält im Vergleich zu „CSV1“ einen zweiten Button und ein zweites Anzeigefeld (Abbildung 5). Drückt man während Laufzeit den Button, welcher die Aufschrift „Neue CSV-Datei“ trägt, dann soll eine neue CSV-Datei erstellt und auf der CompactFlash Card gespeichert werden. Die so generierten CSV-Dateien werden mit einem Datei-Namen, welcher eine laufende Nummer enthält, benannt: *Log_001.CSV*, *Log_002.CSV* etc. Der Name inkl. Pfad der neusten CSV-Datei wird ins Anzeigefeld geschrieben.

Mit Hilfe des Buttons „Messwert aufzeichnen“ wird der aktuelle Messwert aus CN6 in die neuste CSV-Datei gespeichert, deren Name auch im Anzeigefeld steht. Die Aufzeichnung soll jedoch nur erlaubt sein, wenn auch tatsächlich schon eine CSV-Datei erstellt worden ist.

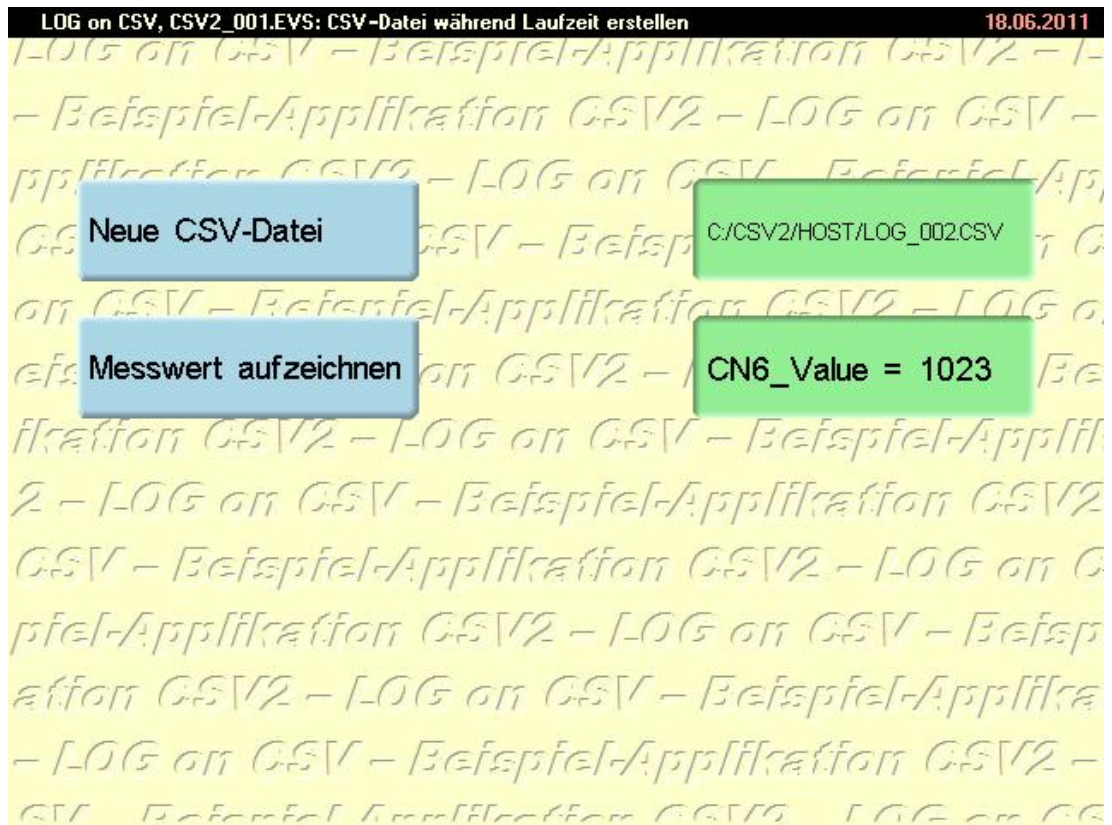


Abbildung 5: ScreenShot der Beispiel-Applikation CSV2.

Programmcode von CSV2

Für die Entwicklung der Beispiels-Applikation CSV2 starten wir mit dem Stand von „CSV1“.

1. Definieren wichtiger Variablen

Vorbereitend ergänzen wir die bereits aus CSV1 vorhandenen CSV-Deklarationen (eigerScript-Code 8).

eigerScript-Code 8: Deklarationen von Stringvariablen für den Namen eines LogFiles und für den jeweiligen Mess-Datensatz.

```
; CSV-Deklarationen:
STRING [15]      CSV_Path.$ = 'C:/CSV1/HOST/'
STRING [15]      CSV_Name.$ = 'LOG_'           ; Stamm für Datei-Name
INTEGER         CSV_Number.I = 0             ; fortlaufende Nr im Datei-Namen
STRING [50]      CSV_PathAndName.$ = ''       ; String Pfad/Datei-Name
STRING [50]      CSV_Ueberschriftenzeile.$ = ''
STRING [15]      CSV_Datensatz.$ = ''
INTEGER         CSV_Flag.I = 0 ; 0 = CSV noch nicht erstellt / 1 = erstellt
```

2. Subroutine für die Erstellung eines CSV-Datei während Laufzeit

Mit der Subroutine `SUB BuildCSV` (eigerScript-Code 9) wird ein LOG-File gebildet, dessen Datei-Name eine laufende Nummer enthält. Pfad und Datei-Name dieses LOG-Files wird zunächst im String `CSV_PathAndName.$` aufgebaut. Danach wird mit „Time“ und „Value“ die Überschriftenzeile der CSV-Tabelle gebildet. Die Spalten einer CSV-Tabelle sind mit Semikolon (;) getrennt. Im String `CSV_Ueberschriftenzeile.$` wird die Überschrift durch anfügen eines CRLF abgeschlossen, um in der CSV-Datei zu einer neuen Zeile zu wechseln. Damit ist das LOG-File dann bereit für die Aufnahme eines ersten Datensatzes.

Die Erstellung einer neuen Datei wird dann abgeschlossen mit dem Befehl

```
File.Write_TextFile(VarStr:FileName,VarStr:Buffer)
```

Nach Ausführung der Subroutine `SUB BuildCSV` (eigerScript-Code 9) ist das LOG-File auf der CompactFlash Card bereit für die Aufzeichnung periodisch eingelesener Werte, beispielsweise wie es im eigerScript-Code 6 dargestellt ist.

eigerScript-Code 9: Subroutine, in welcher aus Pfad, Datei-Name und Spalten-Überschriften ein neues LOG-File gebildet wird.

```
SUB BuildCSV
; FileNamen zusammenstellen:
  CSV_Number.I := CSV_Number.I + 1
  CSV_PathAndName.$ := CSV_Path.$           ; "C:/EPRO/HOST/"
  Str.Concat(CSV_PathAndName.$, CSV_Name.$) ; "C:/EPRO/HOST/LOG_"
  eI.FillChar := 0x30                       ; ASCII-Wert für 0 damit 1 = 001
  Str.Cvt_Integer(CSV_PathAndName.$, CSV_Number.I, 3) ; "C:/../LOG_001"
  Str.Concat ( CSV_PathAndName.$, '.CSV' )     ; Dateinamen-Endung "

; Überschriften-Zeile zusammenstellen:
  CSV_Ueberschriftenzeile.$ := 'Time;Value' ; Überschr. für CSV-Tab
  Str.Add_CRLF(CSV_Ueberschriftenzeile.$) ; Zeilenabschluss mit CRLF

; Neue Datei mit Überschriften-Zeile abspeichern:
  File.Write_TextFile(CSV_PathAndName.$, CSV_Ueberschriftenzeile.$)
ENDSUB
```



Existiert auf der CompactFlash Card bereits eine Datei mit gleichem Pfad und Namen, so wird diese durch den Befehl `File.Write_TextFile()` mit der neuen Datei überschrieben.

3. Button „Neue CSV-Datei“

Mit einem zweiten Button soll während Laufzeit eine neue CSV-Datei erstellt werden. Dafür kopieren wir den Code des Button01 („Messwert aufzeichnen“), so dass im Programm-Code neu zwei Buttons nacheinander stehen. Bei den Subroutinen des zweiten Buttons ändern wir die Button-Nr. von 01 auf 02. Zudem weisen wir dem Button02 die Y-Koordinate `ButtonRow02_Y` zu. Neu hat der Button02 die Funktion „Messwert aufzeichnen“. Entsprechend müssen wir auch im Hauptprogramm den Button-Aufruf inkl. HotSpot duplizieren.

Button 1 erhält nun die Aufschrift „Neue CSV-Datei“. Wird dieser Button gedrückt, so soll beim Up-Event die Subroutine `SUB BuildCSV` aufgerufen und damit eine neue CSV-Datei erstellt werden (vgl. eigerScript-Code 10).

eigerScript-Code 10: Button-Subroutine für Up-Ereignis mit Aufruf der Subroutine `BuildCSV`, welche den eine neue CSV-Tabelle erstellt (vgl. eigerScript-Code 9).

```
SUB B01_Up                ; Subroutine wird aufgerufen bei Up-Ereignis
  CallSubroutine(B01_Leave) ; Button im Up-Zustand zeichnen
  CallSubroutine(BuildCSV) ; neue CSV-Datei erstellen
ENDSUB
```

4. Anzeige für aktuellen Datei-Namen der Log-Datei

Damit wir wissen, in welche Datei der aktuelle Messwert jeweils gespeichert wird, wollen wir den Namen der „aktiven“ CSV-Datei in einer zweiten Anzeige sichtbar machen.

Dazu duplizieren wir die Anzeige-Subroutine `SUB ShowValue` und benennen die kopierte Subroutine mit `SUB ShowFileName`. Der „alten“ Subroutine weisen wir die Y-Koordinate `ButtonRow02_Y` zu. Damit rückt die Messwert-Anzeige hinunter auf die Höhe des zuvor ebenfalls versetzten Buttons „Messwert aufzeichnen“. Die neue Subroutine behält die Y-Koordinate `ButtonRow01_Y`. In `SUB ShowFileName`. Löschen wir nun alle Befehle, welche für die Abfrage von CN6 zuständig sind, so dass nur noch die für die Anzeige benötigten Befehle übrigbleiben. Damit soll nun in dieser Subroutine anstelle der Messwertes der Pfad und der Name des aktuellen CSV-Datei, welcher in der Stringvariable gespeichert ist, angezeigt werden.

eigerScript-Code 11: Subroutine für die Anzeige des Datei-Namens mit Pfad in einem Anzeige-Feld. Da der Datei-Name inkl. Pfad etwas lang ist, wird nach dem Laden des Styles eine kleinere Schriftart gewählt.

```
SUB ShowFileName
  Display.Prepare()
  ; Anzeige-Feld zeichnen:
  Fill.LabelParameter(Button_DN_Style) ;Anzeige im Style d.Up-Button
  Load.Geometry_XYWH(ButtonRight_X,ButtonRow01_Y,ButtonWidth,ButtonH
eight) ; Position und Geometrie des Anzeigefeldes
  eI.FontNumber := Font_Arial_12n
  Label.Text(CSV_PathAndName.$) ; Datei-Namen in Label anzeigen
  Display.ShowWindow()
ENDSUB
```

Entsprechend müssen wir auch im Hauptprogramm den Anzeige-Aufruf duplizieren. Die Anzeige des Namens der neusten CSV-Datei soll nicht nur bei View-Start angezeigt, sondern immer aktualisiert werden, sobald mit Hilfe des Buttons „Neue CSV-Datei“ eine neue CSV-Datei erstellt worden ist. Deshalb wird in der Subroutine für das Up-Ereignis auch ein Verweis auf die Subroutine `SUB ShowFileName` platziert (eigerScript-Code 12).

eigerScript-Code 12: Subroutine des Up-Events des Buttons „Neue CSV-Datei“. Unmittelbar nach dem Erstellen einer neuen CSV-Datei wird auch die Subroutine `ShowFileName` aufgerufen, um den neuen Datei-Namen anzuzeigen.

```
SUB B01_Up ; Subroutine wird aufgerufen bei Up-Ereignis
  CallSubroutine(B01_Leave) ; Button im Up-Zustand zeichnen
  CallSubroutine(BuildCSV) ; neue CSV-Datei erstellen
  CallSubroutine(ShowFileName) ; Anzeige d.File-Namens aktualisieren
ENDSUB
```

5. Sicherheit für die Aufzeichnung von Daten

Nun müssen wir noch eine Sicherheit einbauen, damit man mit dem Button „Messwert aufzeichnen“ nur Datensätze abspeichern kann, wenn schon eine CSV-Datei erstellt worden ist. Dafür haben wir vorsorglich die Integervariable `CSV_Flag.I` als Flag definiert (vgl. eigerScript-Code 8). Diese hat zu Beginn, wenn noch keine CSV-Datei erstellt worden ist, den Wert 0. Sobald eine neue CSV-Datei erstellt ist, soll `CSV_Flag.I` den Wert „1“ erhalten. Diesen Wechsel bewirkt eine entsprechende Programmzeile in der Subroutine für das Up-Ereignis des Buttons „Neue CSV-Datei“.

eigerScript-Code 13: Subroutine des Up-Events des Buttons „Neue CSV-Datei“. Wenn eine neue CSV-Datei erstellt worden ist, wechselt das CSV_Flag auf 1 (letzte Zeile).

```
SUB B01_Up           ; Subroutine wird aufgerufen bei Up-Ereignis
  CallSubroutine(B01_Leave)      ; Button im Up-Zustand zeichnen
  CallSubroutine(BuildCSV)      ; neue CSV-Datei erstellen
  CallSubroutine(ShowFileName)  ; Anzeige d. File-Namens aktual.
  CSV_Flag.I := 1              ; 1 = neue CSV-Datei erstellt
ENDSUB
```

Bei Betätigung des Buttons „Messwert aufzeichnen“ soll die Subroutine `SUB LogValue` nur aufgerufen werden, wenn schon eine neue CSV-Datei erstellt worden ist, d.h. wenn `CSV_Flag.I` den Wert 1 aufweist. Das erreichen wir mit einer `IF`-Bedingung in der Subroutine für das Up-Ereignis des Buttons „Messwert aufzeichnen“ (eigerScript-Code 14).

eigerScript-Code 14: Subroutine des Up-Events des Buttons „Messwert aufzeichnen“. Der aktuelle Datensatz wird nur aufgezeichnet, wenn CSV_Flag den Wert 1 aufweist.

```
SUB B02_Up           ; Subroutine wird aufgerufen bei Up-Ereignis
  CallSubroutine(B02_Leave)      ; Button im Up-Zustand zeichnen
  IF CSV_Flag.I == 1 THEN      ; 1 = CSV-Datei existiert
    CallSubroutine(LogValue)    ; Messwert in CSV-Tab. aufzeichnen
  ENDIF
ENDSUB
```

■ CSV-File analysieren

Die Klasse CSV von eigerScript stellt eine ganze Reihe von Befehlen zur Verfügung, mit welchen eine CSV-Datei ausgewertet werden kann (vgl. Tabelle 1).

Tabelle 1: eigerScript-Befehle zur Auswertung von CSV-Tabellen

EigerScript-Befehl	Kurzbeschreibung
<code>CSV.GetMax_Columns</code> (VarInt:MaxColumns,VarStr:CSV-String)	Ermittelt die maximale Anzahl Spalten einer CSV-Tabelle.
<code>CSV.GetMax_Lines</code> (VarInt:MaxLines,VarStr:CSV-String)	Ermittelt die maximale Anzahl Zeilen einer CSV-Tabelle.
<code>CSV.Find_in_Column</code> (VarStr:CSV-String,VarInt:StartLine,VarInt:Colum,VarStr:MatchString,VarInt:LineNoFound,VarStr:DataLine)	Sucht in einer bestimmten Spalte (VarInt:Colum) nach einem bestimmten Ausdruck (VarStr:MatchString) und schreibt als Resultat die ganze Zeile mit dem gesuchten Ausdruck in einen String (VarStr:DataLine). Zusätzlich wird auch die Nummer der betreffenden Zeile ausgegeben (VarInt:LineNoFound). Mit VarInt:StartLine wird die Nummer der Zeile angegeben, ab welcher der Suchvorgang laufen soll.
<code>CSV.Analyze_String</code> (VarStr:CSV-String)	Wird ein eingelesener CSV-String in einen anderen String-Buffer kopiert, dann muss der zweite String-Buffer zuerst mit diesem Befehl analysiert werden, bevor er mit den anderen CSV-Befehlen ausgewertet werden kann.
<code>CSV.Get_Integer</code> (VarInt:Number,VarStr:CSV-String,VarInt:Zeile,VarInt:Spalte)	Liest aus einem mit Zeilen- und Spalten-Nummer bestimmten Feld den Integerwert aus.
<code>CSV.Get_UIInteger</code> (VarInt:Number,VarStr:CSV-String,VarInt:Zeile,VarInt:Spalte)	Wird durch den Compiler des eigerStudio 1.03 nicht unterstützt
<code>CSV.Get_Long</code> (VarLong:Number,VarStr:CSV-String,VarInt:Zeile,VarInt:Spalte)	Liest aus einem mit Zeilen- und Spalten-Nummer bestimmten Feld den Longwert aus.
<code>CSV.Get_LongDeci</code> (VarLong:Number,VarStr:CSV-String,VarInt:Zeile,VarInt:Spalte,VarInt:Nachkomma)	Liest aus einem mit Zeilen- und Spalten-Nummer bestimmten Feld eine Dezimalzahl als Longwert aus. Dabei muss die Anzahl der gewünschten Nachkommastellen angegeben werden.
<code>CSV.Get_String</code> (VarStr:String,VarStr:CSV-String,VarInt:Zeile,VarInt:Spalte)	Liest den betreffenden Feldinhalt im Stringformat ein (auch für Zahlen möglich, die nur angezeigt werden müssen). Der bisherige Stringinhalt wird ersetzt.
<code>CSV.Get_HighColor</code> (VarInt:Color,VarStr:CSV-String,VarInt:Zeile,VarInt:Spalte)	Holt den Farbwert aus dem betreffenden Tabellenfeld. Der Farbwert muss im HexFormat vorliegen, z.B. #25AAFF (= 025,170,250 RGB)
<code>CSV.Get_ByteHex</code> (VarInt:Number,VarStr:CSV-String,VarInt:Zeile,VarInt:Spalte)	Liest den maximal 2 stelligen Hexwert (Datentyp Byte Hex, z.B. 7B) aus dem betreffenden Tabellenfeld.

<code>CSV.Get_WordHex(VarInt:Number,VarStr:CSV-String, VarInt:Zeile, VarInt:Spalte)</code>	Liest eine maximal 4 stellige HEX-Zahl (Datentyp Word Hex, z.B. 2F03) aus dem betreffenden Tabellenfeld.
<code>CSV.Get_LongHex(VarLong:Number,VarStr:CSV-String, VarInt:Zeile,VarInt:Spalte)</code>	Liest eine maximal 8 stellige HEX-Zahl (Datentyp Long Hex, z.B. 2F04AA3B) aus dem betreffenden Tabellenfeld.
<code>CSV.DataFieldLength(VarInt:Length,VarStr:CSV-String, VarInt:Zeile,VarInt:Spalte)</code>	Zählt die Anzahl Zeichen im betreffenden Tabellenfeld.
<code>CSV.Get_String(VarStr:String,VarStr:CSV-String, VarInt:Zeile,VarInt:Spalte)</code>	Holt betreffenden Feldinhalt im Stringformat (auch für Zahlen möglich, die nur angezeigt werden müssen). Der bisherige Stringinhalt wird ersetzt – nicht hinten angefügt.
<code>CSV.Put_String(VarStr:String,VarStr:CSV-String,VarInt:Zeile, VarInt:Spalte)</code>	Fügt einen String in das mit Zeilen- und Spalten-Nummer bestimmte Tabellenfeld ein. Der bisherigen Feldinhalt wird dadurch ersetzt. Die Änderung wirkt sich erst auf die CSV-Datei auf der CompactFlash Card (CFC) aus wenn diese mit <code>File.Write_TextFile()</code> C auf der CFC überschrieben wird.

Diese Befehle können jedoch nicht direkt auf die CSV-Datei, die auf der CFC liegt, angewendet werden. **Zuerst muss der ganze Inhalt einer CSV-Datei ins RAM, d.h. in einen String-Buffer eingelesen werden**, und danach wird dieser String analysiert, z.B. die Anzahl Zeilen ermittelt oder der Inhalt des Feldes in Zeile 3, Spalte 2 ausgelesen. Je nach Datentyp eines Feldinhalts muss dieser als String, Integer, Hex-Wert etc. ausgelesen werden.

Beschreibung von CSV3

In der Beispiel-Applikation finden Sie Script-Beispiele für alle Befehle, welche in der Tabelle 1, S.20 kurz aufgelistet sind. Auf der CompactFlash Card sind zwei CSV-Tabellen gespeichert (vgl. Abbildung 6), deren Inhalte in Abbildung 7 und Abbildung 8 gezeigt sind.

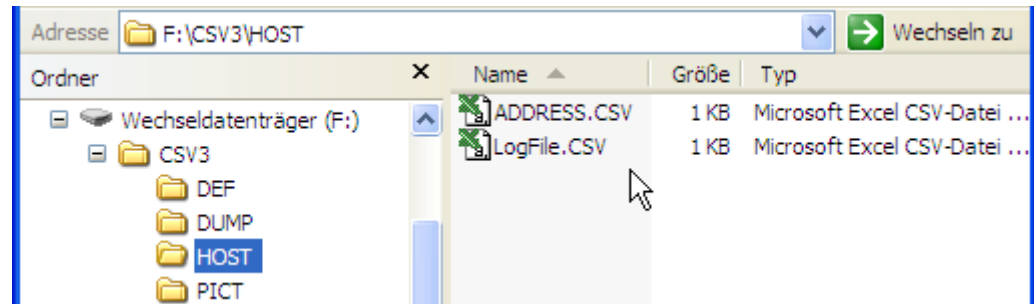


Abbildung 6: In der Beispiel-Applikation CSV3 sind im HOST-Verzeichnis zwei CSV-Dateien gespeichert.

	1	2	3	4	5	6	7	8	9	10	11	12
	A	B	C	D	E	F	G	H	I	J	K	L
1	first name	last name	address	zip	place	phone	salary CHF	tax rate	favorite color	Character	WordHex	WordHex
2	Peter	Steiger	Biberweg 4	9434	Au	041 52 52	20230	0.22	#FFFFFF	AF	12AB	12ABF21C
3	Claudine	Moser	Rigiblick	6415	Arth	053 12 25	70320	0.35	#EE66FF	7B	1C3D	1C3DB890
4	Karin	Trepp	Seestrasse 2	6466	Bauen	074 16 98	45060	0.31	#25AAFF	33	2DDD	2DDD067C
5	Franz	Reber	Mattenweg 1	3067	Boll	054 89 75	90350	0.42	#AA66FF	43	2F03	2F04AA3B

Abbildung 7: Die CSV-Datei ADDRESS.CSV ist eine Tabelle mit 5 Zeilen und 12 Spalten inkl. Überschrift. In der Abbildung ist die Tabelle in der Excel-Ansicht gezeigt, deshalb sind die Semikolons (;) als Spaltenseparator nicht sichtbar.

	A	B
1	Time	Value
2	10:26:14	0
3	10:31:43	293
4	10:31:46	543
5	11:31:49	789
6	10:31:51	1023
7	10:33:54	112
8	10:34:22	1023

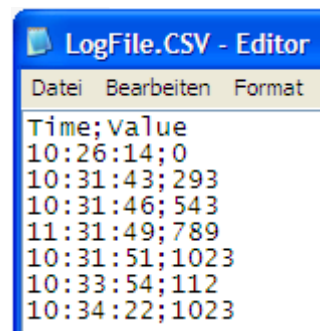


Abbildung 8a und b: Die CSV-Datei LogFile.CSV ist eine Tabelle mit 8 Zeilen und 2 Spalten inkl. Überschrift. In der linken Abbildung ist die Tabelle in der Excel-Ansicht gezeigt, deshalb sind die Semikolons (;) als Spaltenseparator nicht sichtbar. In der rechten Abbildung ist die Datei im Text-Editor geöffnet. Das Semikolon trennt die beiden Spalten.

Die beiden CSV-Tabellen werden wahlweise mit Hilfe des obersten Buttons in eine Stringvariable eingelesen. Danach kann der eingelesene CSV-String mit Hilfe weiterer Buttons analysiert, z.B. auf einen bestimmten Begriff durchsucht oder der Inhalt einer bestimmten Zelle abgefragt werden.



Abbildung 9: ScreenShot von View 1 aus der Beispiel-Applikation CSV3. Mit dem oberen breiten Button wird eine der beiden CSV-Dateien – LogFile.CSV oder ADDRESS.CSV – in eine Stringvariable eingelesen. In diesem Beispiel ist jedem Button eine bestimmte eigerScript-Methode aus der Klasse CSV hinterlegt, womit der eingelesene CSV-String analysiert werden kann.



Abbildung 10: ScreenShot von View 2 aus der Beispiel-Applikation CSV3. Analog zu Abbildung 9 finden Sie in der View 2 je ein Beispiel für die übrigen eigerScript-Methoden der Klasse CSV.

Programmcode von CSV3

In der Folge werden die einzelnen CSV-Methoden anhand von eigerScript-Beispielen des Programmcodes aus den Views CSV3_001 und CSV3_002 erklärt.

`File.Read_CSV(PathANDName.$,CSV_String.$)`

Im eigerScript-Code 15 wird eine CSV-Datei in die zuvor deklarierte String-Variable `CSV_to_analyze.$` eingelesen. Achten Sie darauf, dass die String-Variable genügend gross dimensioniert ist, damit diese die erwartete Anzahl Zeichen der CSV-Datei aufnehmen kann. Ein String sollte maximal 32'000 Zeichen gross sein.

eigerScript-Code 15: Befehl für das Einlesen des gesamten Inhalts einer CSV-Datei (Beispiel-Code – Zeile aus der Beispiel-Applikation CSV3). Die Variable `CSV_String1.$` enthält Pfad und Dateiname der einzulesenden

```
File.Read_CSV(CSV_PathANDName.$,CSV_to_analyze.$)
```

`CSV.GetMax_Columns(MaxColumns.I,CSV_String.$)`

Diese Methode ermittelt die maximale Anzahl Spalten der in eine Stringvariable eingelesenen CSV-Tabelle. Die ermittelte Anzahl Spalten wird in eine Integervariable, z.B. `MaxColumns.I` ausgegeben (vgl. eigerScript-Code 16).

Input für die Methode

- `CSV_String.$`: Name der String-Variablen (z.B. `CSV_to_analyze.$`), in welche die CSV-Datei eingelesen worden ist.

Output der Methode

- `MaxColumns.I`: In diese Integer-Variable (z.B. `MaxColumns.I`) gibt der FOX embedded Computer die Anzahl Spalten der CSV-Tabelle an.

In CSV3 (View 1) wird die Anzahl Spalten mit Hilfe des [Button12](#) ermittelt. Aus Abbildung 9, S.23 ist ersichtlich, dass `LogFile.CSV` aus 2 Spalten besteht (vgl. Abbildung 8, S.22).

eigerScript-Code 16: Subroutine aus CSV3_001.EVS für die Ermittlung der maximalen Anzahl Spalten einer CSV-Tabelle, die zuvor als String in die Stringvariable `CSV_to_analyze.$` eingelesen wurde. Die Anzahl-Zeilen wird hier in die Integervariable `MaxColumns.I` ausgegeben.

```
SUB GetMaxColumns           ;CSV-Analyse: Maximale Anzahl Spalten:
  CSV.GetMax_Columns(MaxColumns.I,CSV_to_analyze.$)
  CallSubroutine(Label112)
ENDSUB
```

CSV.GetMax_Lines(MaxLines.I,CSV_String.\$)

Diese Methode ermittelt die maximale Anzahl Zeilen der in eine Stringvariable eingelesenen CSV-Tabelle (Beispiel: eigerScript-Code 17).

Input für die Methode

- `CSV_String.$`: Name der String-Variablen (z.B. `CSV_to_analyze.$`), in welche die CSV-Datei eingelesen worden ist.

Output der Methode

- `MaxLines.I`: In diese Integer-Variable (z.B. `MaxLines.I`) gibt der FOX embedded Computer die Anzahl Zeilen der CSV-Tabelle an.

In CSV3 (View 1) wird die Anzahl Zeilen mit Hilfe des [Button13](#) ermittelt. Aus Abbildung 9, S.23 ist ersichtlich, dass `LogFile.CSV` aus 8 Zeilen besteht (entsprechend Abbildung 8, S.22).

eigerScript-Code 17: Subroutine aus CSV3_001.EVS für die Ermittlung der maximalen Anzahl Spalten einer CSV-Tabelle, die zuvor als String in die Stringvariable `CSV_to_analyze.$` eingelesen wurde. Die Anzahl-Zeilen wird hier in die Integervariable `MaxColumns.I` ausgegeben.

```
SUB GetMaxLines           ;CSV-Analyse: Maximale Anzahl Zeilen:
  CSV.GetMax_Columns(MaxLines.I,CSV_to_analyze.$)
  CallSubroutine(Label113)
ENDSUB
```

CSV.DataFieldLength(MyFieldLength.I,CSV_String.\$,LineNumber.I,ColumnNumber.I)

Diese Methode ermittelt die Feldlänge, d.h. die Anzahl Zeichen eines mit Zeilen- und Spalten-Nummer bestimmten Datenfeldes und schreibt die Feldlänge in eine Integer-Variablen.

Input für die Methode

- CSV_String.\$: Name der String-Variablen (z.B. `CSV_to_analyze.$`), in welche die CSV-Datei eingelesen worden ist.
- LineNumber.I: Zeilen-Nummer direkt als Ziffer oder Name der Integer-Variablen (z.B. `LineNumber.I`), welche die Zeilen-Nummer des gewünschten Tabellenfeldes enthält.
- ColumnNumber.I: Spalten-Nummer direkt als Ziffer oder Name der Integer-Variablen (z.B. `ColumnNumber.I`), welche die Spalten-Nummer des gewünschten Tabellenfeldes enthält.

Output der Methode

- MyFieldLength.I: In diese Integer-Variablen (z.B. `MyFieldLength.I`) gibt der FOX embedded Computer die Feldlänge (Anzahl vorhandene Zeichen) des mit Zeilen- und Spalten-Nummer bestimmten Tabellen-Feldes aus.

In der Beispiel-Applikation CSV3 (View 2) wird mit dem [Button14](#) die Feldlänge des Datenfeldes (Zeile 2 / Spalte 1) in der der CSV-Datei ADDRESS.CSV ermittelt. Die dahinterliegenden eigerScript-Zeilen sind in eigerScript-Code 18 wiedergegeben. Als Resultat dieser Methode wird in unserem Beispiel im Anzeigefeld die Feldlänge 5 (Zeichen) ausgegeben (vgl. Abbildung 11).

eigerScript-Code 18: Subroutine aus CSV3_002.EVS für die Ermittlung der Feldlänge (Anzahl Zeichen) eines bestimmten Datenfeldes einer CSV-Tabelle, die zuvor als String in die Stringvariable `CSV_to_analyze.$` eingelesen wurde. Die Feldlänge wird hier in die Integer-Variablen `MyFieldLength.I` ausgegeben (vgl. Abbildung 11) .

```
SUB DataFieldLength ; CSV-Analyse: Anzahl Zeichen im Tabelle-Feld:
  LineNumber.I := 2 ; Zeile 2
  ColumnNumber.I := 1 ; Spalte 1
  CSV.DataFieldLength(MyFieldLength.I,CSV_to_analyze.$,LineNumber.I,
  ColumnNumber.I)
  CallSubroutine(Label126)
ENDSUB
```

Abbildung 11: ScreenShot von CSV3 (View 1) mit dem Output von `CSV. DataFieldLength ()` aufgrund von eigerScript-Code 18.



```
CSV.Find_in_Column(CSV_String.$,LineNumber.I,ColumnNumber.I,MatchString.$,LineNoFound.I,DataLine.$)
```

Diese Methode sucht im eingelesenen CSV-String nach einem bestimmten Ausdruck.

Input für die Methode

- CSV_String.\$: Name der String-Variablen (z.B. `CSV_to_analyze.$`), in welche die CSV-Datei eingelesen worden ist.
- LineNumber.I: Zeilen-Nummer direkt als Ziffer oder Name der Integer-Variablen (z.B. `CSV_to_analyze.$`), welche die Zeilen-Nummer enthält. Ab dieser Zeilen-Nummer wird der Suchlauf im CSV-String gestartet.
- ColumnNumber.I: Spalten-Nummer direkt als Ziffer oder Name der Integer-Variablen (z.B. `ColumnNumber.I`), welche die Spalten-Nummer enthält. In dieser Spalte wird nach dem Suchbegriff gesucht.
- MatchString.\$: Suchbegriff als Wort in Hochkomma (z.B. 'Franz') oder Name der String-Variablen (z.B. `MatchString.$`), welche den Suchbegriff enthält. Der Suchbegriff kann auch aus mehreren Wörtern bestehen (z.B. 'Seestrasse 2').

Output der Methode

- LineNoFound.I: In diese Integer-Variable (z.B. `LineNoFound.I`) gibt der FOX embedded Computer die Nummer der Zeile aus, in welcher der Suchbegriff als erstes gefunden worden ist (vgl. Abbildung 12). Wenn keine Übereinstimmung gefunden wird, gibt der Rechner den Wert 0 zurück.
- DataLine.\$: In diese String-Variable (z.B. `DataLine.$`) wird die ganze Zeile ausgegeben, welche den Suchbegriff enthält (vgl. Abbildung 12).

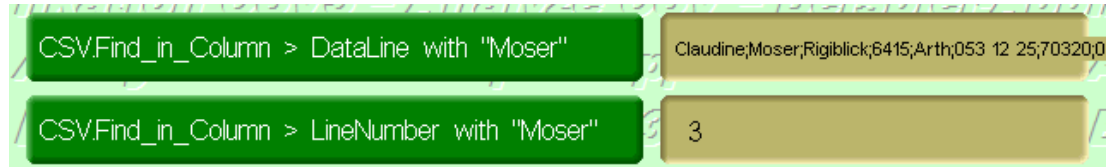
In der Beispiel-Applikation CSV3 (View 1) wird mit den beiden [Buttons 15 und 16](#) nach dem Suchbegriff 'Moser' gesucht. Die dahinterliegenden eigerScript-Zeilen sind in eigerScript-Code 19 wiedergegeben. Als Resultat dieser Suche wird in unserem Beispiel die Zeile 3 (`LineNoFound.I`) und deren Inhalt (`DataLine.$`) angezeigt (vgl. Abbildung 12).

eigerScript-Code 19: Subroutine aus CSV3_001.EVS für die Begriffsuche im eingelesenen CSV-String. Die Methode sucht in der Spalte 2 nach dem Begriff 'Moser' und beginnt die Suche bei der Zeile 1.

```
SUB A_Find_in_Column ; Begriff-Suche in CSV-String
  MatchString.$      := 'Moser' ; Suchbegriff
  LineNumber.I       := 1 ; Suchvorgang starten in Zeile 1
  ColumnNumber.I     := 2 ; Suchen in Spalte 2
  CSV.Find_in_Column(CSV_to_analyze.$,LineNumber.I,ColumnNumber.I,MatchString.$,LineNoFound.I,DataLine.$)
```

```
CallSubroutine (Label14)
ENDSUB
```

Abbildung 12: ScreenShot von CSV3 (View 1) mit dem Output von `CSV.Find_in_Column()` aufgrund von eigerScript-Code 19.



```
CSV.Get_Integer(MyInteger.I,CSV_String.$,LineNumber.I,ColumnNumber.I)
```

Diese Methode kopiert einen Integer-Wert eines mit Zeilen- und Spalten-Nummer bestimmten Feldes in eine Integer-Variable.

Input für die Methode

- `CSV_String.$`: Name der String-Variablen (z.B. `CSV_to_analyze.$`), in welche die CSV-Datei eingelesen worden ist.
- `LineNumber.I`: Zeilen-Nummer direkt als Ziffer oder Name der Integer-Variablen (z.B. `LineNumber.I`), welche die Zeilen-Nummer des gewünschten Tabellenfeldes enthält.
- `ColumnNumber.I`: Spalten-Nummer direkt als Ziffer oder Name der Integer-Variablen (z.B. `ColumnNumber.I`), welche die Spalten-Nummer des gewünschten Tabellenfeldes enthält.

Output der Methode

- `MyInteger.I`: In diese Integer-Variable (z.B. `MyInteger.I`) gibt der FOX embedded Computer den Inhalt des mit Zeilen- und Spalten-Nummer bestimmten Tabellen-Feldes als Integer-Wert aus. Wenn das Feld keinen Integer kompatiblen Wert enthält, wird die Integer-Variable nicht verändert.

In der Beispiel-Applikation CSV3 (View 1) wird mit dem [Button17](#) ein Integer aus der Spalte 2 in der Zeile 4 herausgelesen. Die dahinterliegenden eigerScript-Zeilen sind in eigerScript-Code 20 wiedergegeben. Als Resultat dieser Methode wird in unserem Beispiel der Wert 543 aus der CSV-Datei `LogFile.CSV` gelesen (vgl. Abbildung 13).

eigerScript-Code 20: Subroutine aus CSV3_001.EVS für das Auslesen eines Integer-Wertes aus dem Tabellenfeld (Zeile = 4 / Spalte = 2). Der Integer-Wert wird hier in die Integer-Variablen `MyInteger.I` ausgelesen (vgl. Abbildung 13). Falls das Feld keinen Integer kompatiblen Wert enthält, wird der Inhalt von `MyInteger.I` nicht verändert; in diesem Beispiel bleibt dann der vordefinierte Wert 9999 erhalten.

```

SUB Get_Integer           ; Integer-Wert aus Tabellen-Feld auslesen:
MyInteger.I := 9999 ; Wenn Feld kein Long enthält, dann Wert = 9999
LineNumber.I := 4      ; Zeile 4
ColumnNumber.I := 2    ; Spalte 2
CSV.Get_Integer(MyInteger.I, CSV_to_analyze.$, LineNumber.I, ColumnNumber.I)
CallSubroutine(Label16)
ENDSUB

```

Abbildung 13: ScreenShot von CSV3 (View 1) mit dem Output von `CSV.Get_Integer()` aufgrund von eigerScript-Code 20.



CSV.Get_Long(MyLong.L, CSV_String.\$, LineNumber.I, ColumnNumber.I)

Diese Methode kopiert einen Long-Wert eines mit Zeilen- und Spalten-Nummer bestimmten Feldes in eine Long-Variablen.

Input für die Methode

- `CSV_String.$`: Name der String-Variablen (z.B. `CSV_to_analyze.$`), in welche die CSV-Datei eingelesen worden ist.
- `LineNumber.I`: Zeilen-Nummer direkt als Ziffer oder Name der Integer-Variablen (z.B. `LineNumber.I`), welche die Zeilen-Nummer des gewünschten Tabellenfeldes enthält.
- `ColumnNumber.I`: Spalten-Nummer direkt als Ziffer oder Name der Integer-Variablen (z.B. `ColumnNumber.I`), welche die Spalten-Nummer des gewünschten Tabellenfeldes enthält.

Output der Methode

- `MyLong.L`: In diese Long-Variablen (z.B. `MyLong.L`) gibt der FOX embedded Computer den Inhalt des mit Zeilen- und Spalten-Nummer bestimmten Tabellen-Feldes als Long-Wert aus. Wenn das Feld keinen Long kompatiblen Wert enthält, wird die Long-Variablen nicht verändert.

In der Beispiel-Applikation CSV3 (View 1) wird mit dem [Button18](#) ein Long-Wert aus der Spalte 7 in der Zeile 5 herausgelesen. Die dahinterliegenden eigerScript-Zeilen sind in eigerScript-Code 21 wiedergegeben. Als Resultat dieser Methode

wird in unserem Beispiel der Wert 90350 aus der CSV-Datei ADDRESS.CSV gelesen (vgl. Abbildung 14).

eigerScript-Code 21: Subroutine aus CSV3_001.EVS für das Auslesen eines Long-Wertes aus dem Tabellenfeld (Zeile = 5 / Spalte = 7). Der Long-Wert wird hier in die Long-Variable `MyLong.L` ausgelesen (vgl. Abbildung 14). Falls das Feld keinen Long kompatiblen Wert enthält, wird der Inhalt von `MyLong.L` nicht verändert; in diesem Beispiel bleibt der dann vordefinierte Wert 9999 erhalten.

```

SUB Get_Long           ; Long-Wert aus Tabellen-Feld auslesen:
  MyLong.L := 9999     ; Wenn Feld keinen Long enthält, dann 9999
  LineNumber.I := 5    ; Zeile 5
  ColumnNumber.I := 7  ; Spalte 7
  CSV.Get_Long(MyLong.L, CSV_to_analyze.$, LineNumber.I,
  ColumnNumber.I)
  CallSubroutine(Label17)
ENDSUB

```

Abbildung 14: ScreenShot von CSV3 (View 1) mit dem Output von `CSV.Get_Integer()` aufgrund von eigerScript-Code 20.



```

CSV.Get_LongDeci(MyLongDeci.L, CSV_String.$,
  LineNumber.I, ColumnNumber.I, Nachkomma.I)

```

Diese Methode kopiert einen LongDeci-Wert eines mit Zeilen- und Spalten-Nummer bestimmten Feldes in eine LongDeci-Variable des Typs Long.

Input für die Methode

- `CSV_String.$`: Name der String-Variablen (z.B. `CSV_to_analyze.$`), in welche die CSV-Datei eingelesen worden ist.
- `LineNumber.I`: Zeilen-Nummer direkt als Ziffer oder Name der Integer-Variablen (z.B. `LineNumber.I`), welche die Zeilen-Nummer des gewünschten Tabellenfeldes enthält.
- `ColumnNumber.I`: Spalten-Nummer direkt als Ziffer oder Name der Integer-Variablen (z.B. `ColumnNumber.I`), welche die Spalten-Nummer des gewünschten Tabellenfeldes enthält.
- `Nachkomma.I`: Anzahl der gewünschten Nachkomma-Stellen direkt als Ziffer oder Name der Integer-Variablen (z.B. `ColumnNumber.I`), welche die Anzahl Nachkommastellen enthält.

Output der Methode

- `MyLongDeci.L`: In diese Long-Variable (z.B. `MyLong.L`) gibt der FOX embedded Computer den Inhalt des mit Zeilen- und Spalten-Nummer

bestimmten Tabellen-Feldes als LongDeci-Wert aus. Wenn das Feld keinen Long kompatiblen Wert enthält, wird die Long-Variable nicht verändert.

In der Beispiel-Applikation CSV3 (View 1) wird mit dem [Button22](#) der Inhalt eines Tabellen-Feldes (Zeile 3 / Spalte 8) als „LongDeci-Wert“ in eine Long-Variable ausgelesen. Die dahinterliegenden eigerScript-Zeilen sind in eigerScript-Code 22 wiedergegeben. Bei 3 Nachkommastellen als Vorgabe wird in die Long-Variable der Wert 350 geschrieben (vgl. Tabelle 2, S.32). Bei der Weiterverwendung dieses Long-Werts muss die zugehörige Anzahl Nachkommastellen mitberücksichtigt werden. So wird für die Konvertierung des Long-Werts in einen String auch wieder die gleiche Anzahl Nachkommastellen vorgegeben (vgl. eigerScript-Code 23), damit schlussendlich auf dem Display der richtige Wert angezeigt wird (vgl. Abbildung 15).

eigerScript-Code 22: Subroutine aus CSV3_002.EVS für das Auslesen eines Long-Wertes aus dem Tabellenfeld (Zeile = 5 / Spalte = 7). Der Long-Wert wird hier in die Long-Variable `MyLong.L` ausgelesen (vgl. Abbildung 14). Falls das Feld keinen Long kompatiblen Wert enthält, wird der Inhalt von `MyLong.L` nicht verändert; in diesem Beispiel bleibt dann der vordefinierte Wert 9999 erhalten.

```
SUB Get_LongDeci           ; LongDeci-Wert aus Tabellen-Feld auslesen:
  LineNumber.I           := 3           ; Zeile 3
  ColumnNumber.I         := 8           ; Spalte 8
  Vorkomma.I             := 3           ; drei Vorkommastellen
  Nachkomma.I            := 3           ; drei Nachkommastellen
  MyLongDeci.L := 9999
  CSV.Get_LongDeci(MyLongDeci.L,CSV_to_analyze.$, LineNumber.I,
  ColumnNumber.I, Nachkomma.I)
  CallSubroutine(Label18)
ENDSUB
```

eigerScript-Code 23: eigerScript-Methode, welche einen als Long gespeicherten „LongDeci“-Wert zu einer Dezimalzahl in einen String konvertiert, z.B. für die Anzeige auf dem Display (vgl. Abbildung 15 mit den Vorgaben aus eigerScript-Code 22).

```
Str.Cvt_LongDeci(Label_Text.$, MyLongDeci.L,Vorkomma.I,Nachkomma.I)
```

Abbildung 15: ScreenShot von CSV3 (View 2) mit dem Output von `CSV.Get_Integer()` aufgrund von eigerScript-Code 20.



Tabelle 2: Ausgabe-Werte von `CSV.Get_Integer ()` bei verschiedenen Nachkommastellen.

Input (Line 3 / Column 8)	Nachkomma- stellen	Ausgabewert (LongHex)	Ausgabewert (LongDez)
0.35	3	0000015E	350
0.35	2	00000023	35
0.35	1	00000004	4

```
CSV.Get_HighColor(MyColor.I,CSV_String.$,  
LineNumber.I,ColumnNumber.I)
```

Diese Methode kopiert einen Farbwert vom Web-Format (z.B. #25AAFF) eines mit Zeilen- und Spalten-Nummer bestimmten Feldes in eine Integer-Variable.

Input für die Methode

- `CSV_String.$`: Name der String-Variablen (z.B. `CSV_to_analyze.$`), in welche die CSV-Datei eingelesen worden ist.
- `LineNumber.I`: Zeilen-Nummer direkt als Ziffer oder Name der Integer-Variablen (z.B. `LineNumber.I`), welche die Zeilen-Nummer des gewünschten Tabellenfeldes enthält.
- `ColumnNumber.I`: Spalten-Nummer direkt als Ziffer oder Name der Integer-Variablen (z.B. `ColumnNumber.I`), welche die Spalten-Nummer des gewünschten Tabellenfeldes enthält.

Output der Methode

- `MyColor.I`: In diese Integer-Variable (z.B. `MyColor.I`) gibt der FOX embedded Computer den Inhalt des mit Zeilen- und Spalten-Nummer bestimmten Tabellen-Feldes als Integer-Wert aus. Liegt der Feldinhalt nicht im Web-Format vor, so wird kein Wert in die Variable geschrieben. Der WEB-Farbwert ist 3x8 bit, d.h. für Rot-, Grün-, Blau-Anteil je 8 bit (3 x 256). Das eigerPanel komprimiert den 3x8 bit Farbraum (z.B. #25AAFF) in einen 3x5 bit Farbraum (z.B. #7EA4) und schreibt diesen Wert in die Integer-Variable.



Das Web-Format umfasst 3x8 bit, d.h. für Rot-, Grün-, Blau-Anteil je 8 bit (3 x 256). Das eigerPanel komprimiert den 3x8 bit Farbraum (z.B. #25AAFF) in einen 3x5 bit Farbraum (z.B. #7EA4) und schreibt diesen Wert in die Integer-Variable.

In der Beispiel-Applikation CSV3 (View 2) wird mit dem [Button23](#) ein Farbwert aus der Spalte 9 in der Zeile 4 der CSV-Datei ADDRESS.CSV herausgelesen (Eingabewert = #25AAFF entspricht einer blauen Farbe mit RGB 025/170/256). Die dahinterliegenden eigerScript-Zeilen sind in eigerScript-Code 24 wiedergegeben.

Als Resultat dieser Methode wird in unserem Beispiel das Anzeigefeld mit der entsprechenden blauen Farbe gefärbt. (vgl. Abbildung 16).

eigerScript-Code 24: Subroutine aus CSV3_002.EVS für das Auslesen eines Farb-Wertes aus dem Tabellenfeld (Zeile = 4 / Spalte = 9). Der Farb-Wert wird hier in die Integer-Variable `MyColor.I` ausgelesen und das Label22 entsprechend blau gefärbt (vgl. Abbildung 16). Falls das Feld keinen Farbwert im Web-Format enthält, wird der Inhalt von `MyColor.I` nicht verändert.

```
SUB Get_HighColor      ; Farbwert aus Tabellen-Feld auslesen:
  LineNumber.I        := 4          ; Zeile 4
  ColumnNumber.I      := 9          ; Spalte 9
  CSV.Get_HighColor(MyColor.I,CSV_to_analyze.$, LineNumber.I,
  ColumnNumber.I)
  CallSubroutine(Label22)
ENDSUB
```

Abbildung 16: ScreenShot von CSV3 (View 2) mit dem Output von `CSV.Get_HighColor()` aufgrund von eigerScript-Code 24.



```
CSV.Get_ByteHex(MyByteHex.I,CSV_String.$,
LineNumber.I,ColumnNumber.I)
```

Diese Methode kopiert einen Wert eines mit Zeilen- und Spalten-Nummer bestimmten Feldes in eine Integer-Variable. Der Wert muss im ByteHex-Format vorliegen, d.h. mit max. 2 Ziffern (z.B. AF).

Input für die Methode

- `CSV_String.$`: Name der String-Variablen (z.B. `CSV_to_analyze.$`), in welche die CSV-Datei eingelesen worden ist.
- `LineNumber.I`: Zeilen-Nummer direkt als Ziffer oder Name der Integer-Variablen (z.B. `LineNumber.I`), welche die Zeilen-Nummer des gewünschten Tabellenfeldes enthält.
- `ColumnNumber.I`: Spalten-Nummer direkt als Ziffer oder Name der Integer-Variablen (z.B. `ColumnNumber.I`), welche die Spalten-Nummer des gewünschten Tabellenfeldes enthält.

Output der Methode

- `MyByteHex.I`: In diese Integer-Variable (z.B. `MyByteHex.I`) gibt der FOX embedded Computer den Inhalt des mit Zeilen- und Spalten-Nummer bestimmten Tabellen-Feldes als ByteHex-Wert aus. Enthält das Feld eine Zahl mit mehr als 2 Ziffern, dann liest die Methode die ersten beiden Ziffern

aus, z.B. aus 12AB wird 12. Enthält das Feld keinen Wert im 2-ziffrigen Hex-Format, so wird kein Wert in die Variable geschrieben.

In der Beispiel-Applikation CSV3 (View 2) wird mit dem [Button24](#) ein ByteHex-Wert aus der Spalte 10 in der Zeile 2 der CSV-Datei ADDRESS.CSV herausgelesen (Eingabewert = AF). Die dahinterliegenden eigerScript-Zeilen sind in eigerScript-Code 25 wiedergegeben. Als Resultat dieser Methode wird in unserem Beispiel im Anzeigefeld der Wert AF ausgegeben (vgl. Abbildung 17).

eigerScript-Code 25: Subroutine aus CSV3_002.EVS für das Auslesen eines Wertes im 2-ziffrigen ByteHex-Format aus dem Tabellenfeld (Zeile = 2 / Spalte = 10). Der Wert wird hier in die Integer-Variable `MyByteHex.I` ausgelesen (vgl. Abbildung 17). Falls die Methode keinen ByteHex-Wert auslesen kann, wird der Inhalt von `MyByteHex.I` nicht verändert.

```
SUB Get_ByteHex          ; ByteHex-Wert aus Tabellen-Feld auslesen:
  LineNumber.I          := 2           ; Zeile 2
  ColumnNumber.I        := 10         ; Spalte 10
  CSV.Get_ByteHex(MyByteHex.I,CSV_to_analyze.$, LineNumber.I,
  ColumnNumber.I)
  CallSubroutine(Label123)
ENDSUB
```

Abbildung 17: ScreenShot von CSV3 (View 2) mit dem Output von `CSV.Get_ByteHex()` aufgrund von eigerScript-Code 25.



```
CSV.Get_WordHex(MyWordHex.I,CSV_String.$,
LineNumber.I,ColumnNumber.I)
```

Diese Methode kopiert einen Wert eines mit Zeilen- und Spalten-Nummer bestimmten Feldes in eine Integer-Variable. Der Wert muss im WordHex-Format vorliegen, d.h. mit max. 4 Ziffern (z.B. 1C3D).

Input für die Methode

- `CSV_String.$`: Name der String-Variablen (z.B. `CSV_to_analyze.$`), in welche die CSV-Datei eingelesen worden ist.
- `LineNumber.I`: Zeilen-Nummer direkt als Ziffer oder Name der Integer-Variablen (z.B. `LineNumber.I`), welche die Zeilen-Nummer des gewünschten Tabellenfeldes enthält.

- ColumnNumber.I: Spalten-Nummer direkt als Ziffer oder Name der Integer-Variablen (z.B. `ColumnNumber.I`), welche die Spalten-Nummer des gewünschten Tabellenfeldes enthält.

Output der Methode

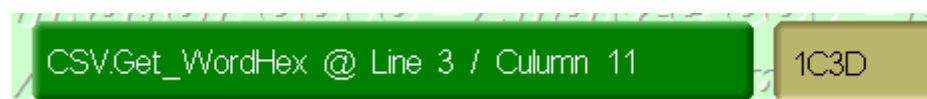
- MyWordHex.I: In diese Integer-Variable (z.B. `MyWordHex.I`) gibt der FOX embedded Computer den Inhalt des mit Zeilen- und Spalten-Nummer bestimmten Tabellen-Feldes als WordHex-Wert aus. Enthält das Feld eine Zahl mit mehr als vier Ziffern, dann liest die Methode die ersten vier Ziffern aus, z.B. aus 12ABEF wird 12AB. Enthält das Feld keinen Wert im 4-ziffrigen Hex-Format, so wird kein Wert in die Variable geschrieben.

In der Beispiel-Applikation CSV3 (View 2) wird mit dem [Button25](#) ein WordHex-Wert aus der Spalte 11 in der Zeile 3 der CSV-Datei ADDRESS.CSV herausgelesen (Eingabewert = 1C3D). Die dahinterliegenden eigerScript-Zeilen sind in eigerScript-Code 26 wiedergegeben. Als Resultat dieser Methode wird in unserem Beispiel im Anzeigefeld der Wert 1C3D ausgegeben (vgl. Abbildung 18).

eigerScript-Code 26: Subroutine aus CSV3_002.EVS für das Auslesen eines Wertes im 4-ziffrigen WordHex-Format aus dem Tabellenfeld (Zeile = 3 / Spalte = 11). Der Wert wird hier in die Integer-Variable `MyWordHex.I` ausgelesen (vgl. Abbildung 17). Falls die Methode keinen WordHex-Wert auslesen kann, wird der Inhalt von `MyWordHex.I` nicht verändert.

```
SUB Get_WordHex           ; WordHex (4 stellige HEX-Zahl ) auslesen:
  LineNumber.I           := 3           ; Zeile 3
  ColumnNumber.I         := 11          ; Spalte 11
  CSV.Get_WordHex(MyWordHex.I, CSV_to_analyze.$, LineNumber.I,
  ColumnNumber.I)
  CallSubroutine(Label124)
ENDSUB
```

Abbildung 18: ScreenShot von CSV3 (View 2) mit dem Output von `CSV.Get_WordHex()` aufgrund von eigerScript-Code 26.



```
CSV.Get_LongHex(MyLongHex.L, CSV_String.$,
LineNumber.I, ColumnNumber.I)
```

Diese Methode kopiert einen Wert eines mit Zeilen- und Spalten-Nummer bestimmten Feldes in eine Long-Variable. Der Wert muss im LongHex-Format vorliegen, d.h. mit max. 8 Ziffern (z.B. 2F04AA3B).

Input für die Methode

- CSV_String.\$: Name der String-Variablen (z.B. `CSV_to_analyze.$`), in welche die CSV-Datei eingelesen worden ist.
- LineNumber.I: Zeilen-Nummer direkt als Ziffer oder Name der Integer-Variablen (z.B. `LineNumber.I`), welche die Zeilen-Nummer des gewünschten Tabellenfeldes enthält.
- ColumnNumber.I: Spalten-Nummer direkt als Ziffer oder Name der Integer-Variablen (z.B. `ColumnNumber.I`), welche die Spalten-Nummer des gewünschten Tabellenfeldes enthält.

Output der Methode

- MyLongHex.I: In diese Long-Variable (z.B. `MyLongHex.I`) gibt der FOX embedded Computer den Inhalt des mit Zeilen- und Spalten-Nummer bestimmten Tabellen-Feldes als LongHex-Wert aus. Enthält das Feld eine Zahl mit mehr als acht Ziffern, dann liest die Methode die ersten acht Ziffern aus, z.B. aus 12ABEF39D1 wird 12ABEF39. Enthält das Feld keinen Wert im 8-ziffrigen Hex-Format, so wird kein Wert in die Variable geschrieben.

In der Beispiel-Applikation CSV3 (View 2) wird mit dem [Button26](#) ein LongHex-Wert aus der Spalte 12 in der Zeile 5 der CSV-Datei ADDRESS.CSV herausgelesen. Die dahinterliegenden eigerScript-Zeilen sind in eigerScript-Code 27 wiedergegeben. Als Resultat dieser Methode wird in unserem Beispiel im Anzeigefeld der Wert 2F04AA3B ausgegeben (vgl. Abbildung 19).

eigerScript-Code 27: Subroutine aus CSV3_002.EVS für das Auslesen eines Wertes im 8-ziffrigen LongHex-Format aus dem Tabellenfeld (Zeile = 5 / Spalte = 12). Der Wert wird hier in die Long-Variable `MyLongHex.I` ausgelesen (vgl. Abbildung 19). Falls die Methode keinen WordHex-Wert auslesen kann, wird der Inhalt von `MyLongHex.I` nicht verändert.

```
SUB Get_LongHex           ; LongHex (8 stellige HEX-Zahl ) auslesen :
  LineNumber.I           := 5           ; Zeile 5
  ColumnNumber.I         := 12          ; Spalte 12
  CSV.Get_LongHex(MyLongHex.L, CSV_to_analyze.$, LineNumber.I,
  ColumnNumber.I)
  CallSubroutine(Label25)
ENDSUB
```

Abbildung 19: ScreenShot von CSV3 (View 2) mit dem Output von `CSV.Get_LongHex()` aufgrund von eigerScript-Code 27.



```
CSV.Get_String(MyString.$,CSV_String.$,LineNumber.I,ColumnNumber.I)
```

Diese Methode kopiert den Inhalt eines mit Zeilen- und Spalten-Nummer bestimmten Datenfeldes als Zeichenfolge (String) in eine String-Variablen.

Input für die Methode

- CSV_String.\$: Name der String-Variablen (z.B. `CSV_to_analyze.$`), in welche die CSV-Datei eingelesen worden ist.
- LineNumber.I: Zeilen-Nummer direkt als Ziffer oder Name der Integer-Variablen (z.B. `LineNumber.I`), welche die Zeilen-Nummer des gewünschten Tabellenfeldes enthält.
- ColumnNumber.I: Spalten-Nummer direkt als Ziffer oder Name der Integer-Variablen (z.B. `ColumnNumber.I`), welche die Spalten-Nummer des gewünschten Tabellenfeldes enthält.

Output der Methode

- MyString.\$: In diese String-Variablen (z.B. `MyString.$`) gibt der FOX embedded Computer den Inhalt des mit Zeilen- und Spalten-Nummer bestimmten Tabellen-Feldes als Zeichenfolge aus.

In der Beispiel-Applikation CSV3 (View 2) wird mit dem [Button27](#) der Feldinhalt als Zeichenfolge (String) aus der Spalte 2 in der Zeile 4 herausgelesen. Die dahinterliegenden eigerScript-Zeilen sind in eigerScript-Code 28 wiedergegeben. Als Resultat dieser Methode wird in unserem Beispiel der Wert 543 aus der CSV-Datei LogFile.CSV gelesen (vgl. Abbildung 20).

eigerScript-Code 28: Subroutine aus CSV3_002.EVS für das Auslesen des Inhalts eines Datenfeldes (Zeile = 3 / Spalte = 1). Der Inhalt wird als Zeichenfolge in eine String-Variablen – z.B. `MyString.$` – ausgelesen (vgl. Abbildung 20).

```
SUB Get_String                ;CSV-Analyse: Maximale Anzahl Spalten:
  LineNumber.I                := 3                ; Zeile 3
  ColumnNumber.I              := 1                ; Spalte 1
  CSV.Get_String(MyString.$,CSV_to_analyze.$, LineNumber.I,
  ColumnNumber.I)
  CallSubroutine(Label127)
ENDSUB
```

Abbildung 20: Screenshot von CSV3 (View 2) mit dem Output von `CSV.Get_Integer()` aufgrund von eigerScript-Code 28.



```
CSV.Put_String(MyString.$,CSV_String.$,LineNumber.I,ColumnNumber.I)
```

Diese Methode kopiert den Inhalt eines mit Zeilen- und Spalten-Nummer bestimmten Datenfeldes als Zeichenfolge (String) in eine String-Variable.

Input für die Methode

- CSV_String.\$: Name der String-Variablen (z.B. `CSV_to_analyze.$`), in welche die CSV-Datei eingelesen worden ist.
- LineNumber.I: Zeilen-Nummer direkt als Ziffer oder Name der Integer-Variablen (z.B. `LineNumber.I`), welche die Zeilen-Nummer des gewünschten Tabellenfeldes enthält.
- ColumnNumber.I: Spalten-Nummer direkt als Ziffer oder Name der Integer-Variablen (z.B. `ColumnNumber.I`), welche die Spalten-Nummer des gewünschten Tabellenfeldes enthält.

Output der Methode

- MyString.\$: Der Inhalt dieser String-Variable (z.B. `MyString.$`) fügt die Methode in das mit Zeilen- und Spalten-Nummer bestimmten Datenfeld ein. Der bisherige Feldinhalt wird dadurch ersetzt.



Die Methode `CSV.Put_String` wirkt sich nur auf den eingelesenen CSV-String aus – die entsprechende CSV-Datei auf der CompactFlash Card (CFC) bleibt unverändert. Erst wenn die CSV-Datei durch `File.Write_TextFile` mit dem veränderten CSV-String neu überschrieben wird, ist die Änderung bleibend – ausserhalb des RAM – auf der CFC abgespeichert.

In der Beispiel-Applikation CSV3 (View 2) wird mit dem [Button28](#) der Inhalt der Stringvariablen `MyString.$` in das Datenfeld Spalte 1 / Zeile 3 eingefügt und damit anstelle von „Claudine“ abwechslungsweise die Vornamen „Kurt“ und „Sylvie“ eingefügt. In den eigerScript-Zeilen in eigerScript-Code 29 ist nur ein Teil der betreffenden Subroutine wiedergegeben; der Wechsel zwischen „Kurt“ und „Sylvie“ ist hier weggelassen. Um die Wirkung der Methode anzuzeigen, wird im Anzeigefeld (Label28) jeweils die betreffende Tabellenzeile neu eingeblendet (vgl. Abbildung 21).

eigerScript-Code 29: Subroutine aus CSV3_002.EVS für das Auslesen des Inhalts eines Datenfeldes (Zeile = 3 / Spalte = 1). Der Inhalt wird als Zeichenfolge in eine String-Variable – z.B. **MyString.\$** – ausgelesen (vgl. Abbildung 21).

```
SUB Put_String           ;CSV-Analyse: Maximale Anzahl Spalten:
  LineNumber.I          := 3           ; Zeile 3
  ColumnNumber.I        := 1           ; Spalte 1
  MyString.$ := 'Kurt'
  CSV.Put_String(MyString.$,CSV_to_analyze.$,LineNumber.I,
  ColumnNumber.I)
  CallSubroutine(Label128)
ENDSUB
```

Abbildung 21: ScreenShot von CSV3 (View 2) mit dem Output von **CSV.Get_Integer()** aufgrund von eigerScript-Code 29.

