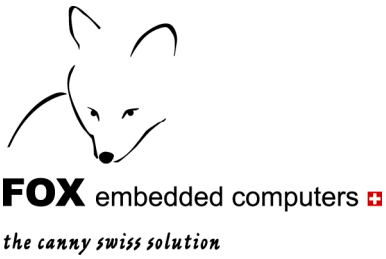


eigerScript – Schnelleinstieg

Letzte Bearbeitung: 27. Mai 2009

Christoph Angst

eiger@s-tec.ch

www.s-tec.ch

www.eigergraphics.com

Dieser eigerScript-Schnelleinstieg ist Teil der eigerDemoCD zum eigerDemoKit. Sie können dieses Dokument auch von der Downloadseite unserer Homepage www.eigergraphics.com herunterladen. Dort finden Sie immer die aktuellste Version. Auf der gleichen Download-Seiten stehen diverse Application Notes bereit zu folgenden Themen:

- **Wie wird eine EGI erstellt**
- **Alternativen zum Erstellen von Layouts**
- **Tasten programmieren mit eigerScript**
- **eigerScript-Symbole**
- **Fonts und ASCII-Codes**
- **Eigene Fonts einbetten**
- **Tabellen auf FOX**
- **Analog- und Digitalsignale einlesen und ausgeben**
- **Serielle Schnittstellen RS232 programmieren**
- **Slideshow auf dem eigerPanel**

Inhaltsverzeichnis

Vorwort

Der vorliegende eigerScript-Schnelleinstieg führt Sie in die Funktionsweise des FOX und in die Programmiersprache *eigerScript* ein. Die kurzen Übungen von 5 bis 60 Minuten ermöglichen es Ihnen, schon nach kurzer Zeit eigene Anwendungen auf dem FOX zu programmieren oder bestehende Anwendungen Ihren Bedürfnissen anzupassen.

Den Einstieg in die Programmiersprache eigerScript und in die Bedienung der virtuellen Maschine des FOX möchte ich Ihnen durch Learning by Doing so leicht und ergiebig wie möglich machen. In den folgenden Übungen dieses Schnelleinstiegs erstellen wir gemeinsam ein einfaches Projekt für den FOX. Die Übungen bauen aufeinander auf. Dadurch bauen wir das Projekt Schritt für Schritt aus und lernen dabei die wichtigsten Regeln und Befehle von eigerScript kennen.

Nach dem Durchgang dieses Schnelleinstiegs ...

- ✓ ... kennen Sie das FOX-Prinzip.
- ✓ ... sind sie vertraut mit der eigerScript Entwicklungsumgebung.
- ✓ ... kennen Sie die wichtigsten Befehle und Konstanten von eigerScript.
- ✓ ... haben Sie den Überblick über die Hilfe-Tools von eigerScript.
- ✓ ... sind Sie in der Lage, eine eigene einfache Bedieneroberfläche für den FOX zu programmieren.
- ✓ ... können Sie sich aufgrund der Hilfe-Tools und der eigerScript Befehlssammlung selbständig nach Ihren Bedürfnissen weitere Kenntnisse über eigerScript erarbeiten.

Zum besseren Verständnis sind manchmal zusätzliche Informationen dienlich. Solche Hinweise sind in dieser Anleitung mit  speziell markiert.


U006S003.EGI

In den Übungen entwickeln wir Views, die wir jeweils auf den FOX übertragen, um sie dort auf dem Bildschirm darzustellen. Die Screenshots dieser View-Darstellungen sind auf der DemoCD in den Übungsordnern unkomprimiert und pixelgenau in den Formaten *.EGI und *.BMP abgelegt, beispielsweise im Ordner

Übung06\Screenshot. Der betreffende Dateiname ist im Piktogramm angegeben – z.B. *U006S003.EGI* –, wobei „U“ für Übung steht und das „S“ für Screenshot.

Ich wünsche Ihnen bei diesem Lehrgang viel Vergnügen und viele Erfolgserlebnisse.

Christoph Angst S-TEC electronics AG

Unterägeri, 12. März 2008

Einleitung

Das FOX-Prinzip

zielorientiert
ressourceneffizient
einfach
praktisch

Die Firma S-TEC electronics AG hat sich zum Ziel gesetzt, ein Gesamtsystem anzubieten, das von der Entwicklung bis zur Anwendung der Soft- und Hardware im Alltag einen höchsteffizienten Einsatz von Zeit und Material garantiert.

Kern dieser Strategie ist das eigerSystem. Die Hardware des eigerPanel besteht aus dem embedded Computer FOX und einem TFT-Flachbildschirm und einem Touchpanel. Auf der Software-Seite steht die Programmiersprache eigerScript mit eigener Entwicklungsumgebung.

Das eigerSystem beweist seine Praxistauglichkeit sowohl in individuellen Einzellösungen wie auch in der Entwicklung von Applikationen kommerzieller Serienprodukte.

Was ist eigerScript?

eigerScript ist die Programmiersprache für die eiger Virtual Machine eVM. Mit eigerScript werden eigerSysteme programmiert. Die Programmiersprache lehnt sich stark an syntaktische Elemente aus Pascal und C an. Es stehen in eigerScript viele leistungsfähige Methoden zur Verfügung, aus denen sich ein Programm zusammensetzt.

Mit eigerScript können Sie am PC mit Hilfe der Entwicklungsumgebung eigerStudio und eigerGraphic Suite eine vollständige Display-Applikation auf dem PC entwickeln und auf den FOX übertragen.

Als Teil des eigerSystems ist eigerScript speziell für industrielle Anwendungen entwickelt worden. Sie ermöglicht die Steuerung von Apparaten und die Verwertung von anfallenden Informationen beispielsweise aus Messungen.

Embedded System

Ein Embedded System verbindet die Flexibilität von Software mit der Leistungsfähigkeit eines technischen Gerätes.

Ein embedded Computer ist quasi das Gehirn eines technischen Gerätes. Ein embedded Computer verfügt über Ein- und Ausgänge zu Sensoren, Aktoren, Tasten, Display etc. (vgl. Abbildung 1). Über diese Schnittstellen werden in einem embedded System die physikalischen bzw. chemischen Abläufe eines Gerätes, beispielsweise eines Kaffeeautomaten, kontrolliert und gesteuert und auch Anweisungen von aussen entgegengenommen. In embedded Systems wird in der Regel auf übermässige Rechenleistung verzichtet zugunsten einer optimalen Energie- und Recourceneffizienz. Die Rechenleistung ist der Applikation angepasst.

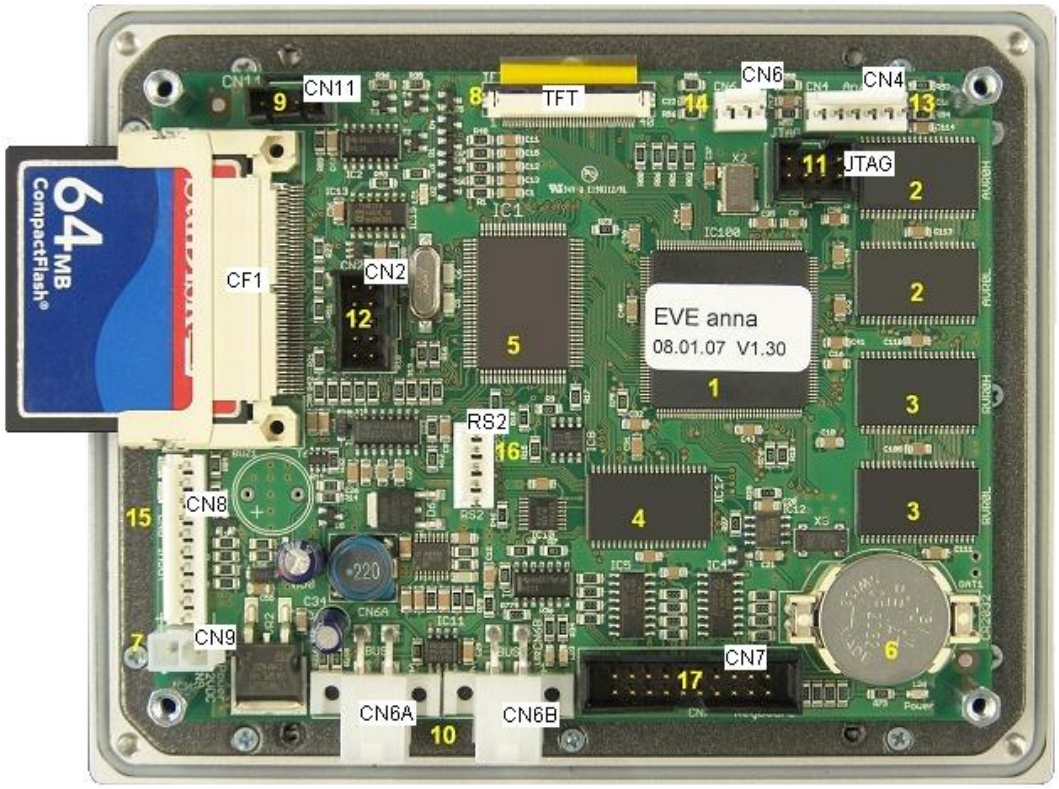


Abbildung 1: FOX embedded computer des eigerPanels 57.

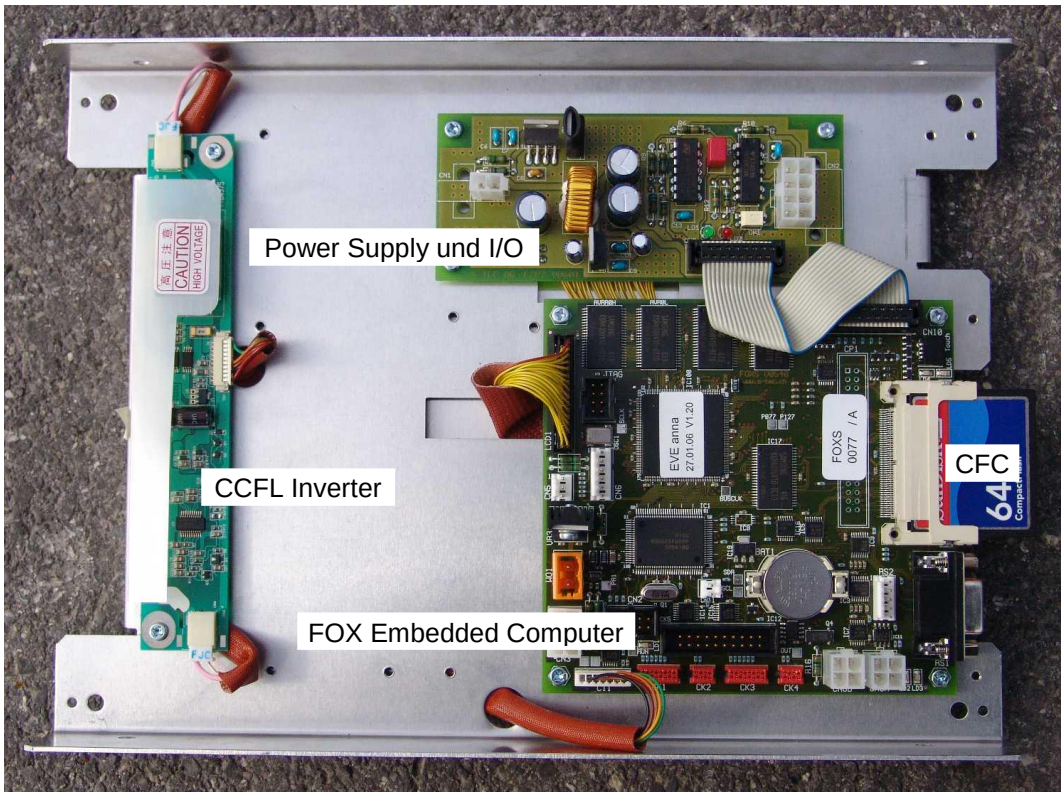
1. Graphic Controller, eigerVideo Engine (EVE anna)
2. Videospeicher Accessible Video Ram (AVR)
3. Videospeicher Refresh Video Ram (RVR)
4. Arbeitsspeicher (RAM)
5. CPU (Micro-Prozessor)
6. RTC-Batterie
7. Power Supply 12V (serielle Schnittstelle RS232)
8. Anschluss Display VGA
9. Anschluss Touchpanel
10. BUS (Serielle Schnittstelle RS485)
11. Programmier-Schnittstelle für EVE anna (JTAG)
12. Programmier-Schnittstelle für Microprozessor (S-PROG10) oder mit Adapterkabel F4337 als FOX-**COM1** (UART1) verwendbar
13. Analog-Eingänge
14. Analogeingang Poti
15. Externe I/O
16. FOX-**COM2** (UART2)
17. 16-KEY Keyboard Input

Low End embedded Systems sind in unserem Alltagsleben weit verbreitet. Wir finden sie grundsätzlich überall, wo ein Computer benutzt wird, ohne dass es der Benutzer merkt. Typische Beispiele sind Kaffeemaschinen, Nähmaschinen, Autos, HiFi-Anlagen, Billetautomaten etc.

Auch der FOX ist als embedded Computer konzipiert, der im Low End Spektrum sehr vielseitig eingesetzt werden kann. Besonders leistungsstark ist der FOX in visuell/graphisch gestützten Touchscreen-Systemen.

Den FOX embedded Computer gibt es in 2 Versionen, den FOX57 für 5.7 Zoll TFT VGA Displays mit LED-Backlight (Abbildung 1) und den FOXS in Verbindung mit VGA Displays mit CFL-Backlight (Abbildung 2).

Abbildung 2:
FOXS auf der Rückseite eines Touchscreen Panels.



Event-driven Programming

Ereignisgesteuerte Software reagiert auf äussere Ereignisse, z.B. Tastendruck, Mausklick oder Timer.

Die ereignisgesteuerte Programmierung (event-driven programming) ist darauf ausgerichtet, bestimmte Benutzeraktivitäten einzufangen und entsprechend zu reagieren. Ein interner Event Manager überwacht die Ereignisse. Er bestimmt, was die Virtuelle Maschine abarbeiten muss. Der Benutzer kann durch Berühren bestimmter Stellen auf dem Touchscreen, durch Tastendruck oder Zahleneingaben den Programmverlauf nach seinen Wünschen steuern. Ein ereignisgesteuertes Programm kann auch auf systeminterne Ereignisse reagieren, beispielsweise auf einmalige oder regelmässige Signale eines programmierten Timers.

„Event-driven“ ist nur ein Fremdwort für eine ganz alltägliche Erfahrung. Unser Verhalten und Handeln ist zu einem grossen Teil ein Reagieren auf laufend eintretende äussere Ereignisse. Die moderne Politik ist hierfür ein besonders eindrückliches Beispiel.

Nehmen wir als weiteres Beispiel den Alltag eines Lehrers. Am frühen Morgen reisst ihn der Wecker aus seinem Schlaf. Dem Lehrer bleibt nichts anderes übrig, als auf dieses Ereignis zu reagieren, aufzustehen und sich für den Tag frisch zu machen. Sein Morgenritual ist genau terminiert. Nach 30 Minuten muss er das Haus verlassen, damit Punkt 8 Uhr der Unterricht beginnen kann. Auf der Fahrt zur Schule passiert er 4 Lichtsignale, welche bei ungünstiger Konstellation die Fahrzeit bedeutend verlängern können. Vor den Fussgängerstreifen muss der Lehrer oft anhalten, weil Fussgänger die Strasse überqueren wollen, und die vielen Verkehrstafeln veranlassen ihn, das Tempo zu drosseln oder zu beschleunigen. Das Unterrichtsprogramm hat der Lehrer zwar vorbereitet, aber er wird immer wieder unterbrochen durch Ereignisse aus seiner Schulklasse. Ein Schüler meldet sich und stellt eine Frage, welche der Lehrer beantworten muss, bevor er weiterfahren kann. Eine weitere Frage führt den Unterricht zu einem ganz anderen Thema. An diesem Morgen ist auch noch Flugtag auf dem nah gelegenen Militärflugplatz. Immer wieder dringt ohrenbetäubender Lärm durchs Fenster und zwingt den Lehrer entweder lauter zu sprechen oder seinen Vortrag kurz zu unterbrechen. Plötzlich öffnet sich die Tür. Ein Knabe hat verschlafen. Auf solche Ereignisse reagiert der Lehrer wie üblich sehr gereizt und der Schüler muss sich eine lange Standpauke anhören. Jede Stunde klingelt die Schulglocke. Das ist immer das Zeichen zum Wechsel des Unterrichtsfachs. Nach der ersten Stunde werden die Deutschbücher weggelegt und die Biologiestunde be-

ginnt mit einem 30 Minuten langen Video über die Zugvögel. So reiht sich während des ganzen Tages Ereignis um Ereignis aneinander. Der Tagesablauf des Lehrers ist zu einem wesentlichen Teil event-driven.

Manchen Menschen ist der Gedanke, event-driven zu sein, ein Horror. Sie tun viel, damit Sie alles im Griff behalten und unvorhergesehene Ereignisse möglichst vermeiden können. Andere lieben die Spontaneität. Sie halten sich offen und flexibel, um auf neue Ereignisse frei reagieren zu können. Völlig vorprogrammierte Freizeit und Ferien könnten sie sich nicht vorstellen.

EigerScript Grund-Ausrüstung

Hardware

- PC (MS Windows XP)
- eigerPanel

  eigerPanel 57, bestehend aus FOX embedded Computer (**FOX 57**), 5,7 Zoll TFT VGA Display, LED-Backlight, Touch und Gehäuse (vgl. Abbildung 1, Abbildung 3)

  oder

  eigerPanel 10.4, bestehend aus FOX embedded Computer (**FOXS**), 10,4 TFT VGA Display, Touch und Gehäuse (vgl. Abbildung 2, Abbildung 4 und Abbildung 5).
- Optional: Keyboard mit bis zu 16 Tasten, angeschlossen an den FOX.
- CompactFlash Card Reader (evtl. im PC eingebaut)
- CompactFlash Card (32MB-2GB)

Abbildung 3: Tischgerät des eigerPanels 57 mit 5.7“ Touch-Display, Potentiometer und Anschlusskabel für COM1 und 2.

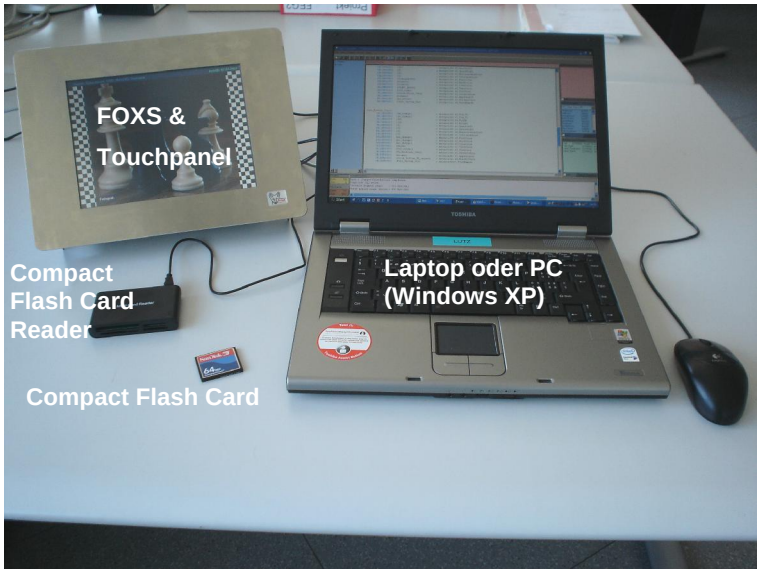


Abbildung 4:
Hardware für die FOX-Programmierung. Im Bild der FOXS mit 10.4“ Display.

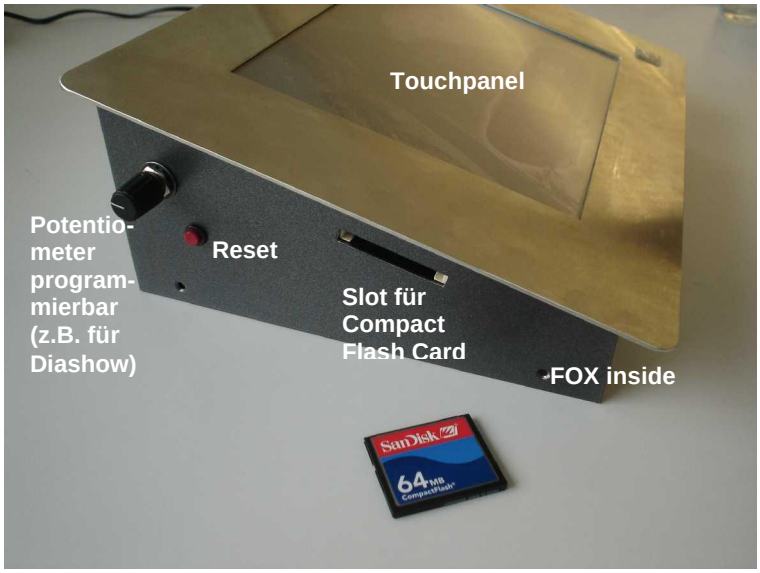


Abbildung 5:
Bedienelemente des FOXS mit 10.4“ Touch-Display und Tischrahmen.

Touchscreen

Im Bereich der embedded Systems haben Touchscreens in den letzten Jahren sehr viel Terrain erobert. Diese Eingabeplattformen sind robust, wartungsarm und mit dem Finger leicht zu bedienen. Sie zeichnen sich auch aus durch eine hohe graphische Gestaltungsfreiheit.

Ein Touchscreen setzt sich zusammen aus einem Bildschirm, Sensoren und einem Controller. Bei Berührung des Bildschirms senden die berührungs- oder lichtempfindlichen Sensoren Signale an den Controller. Der Controller heisst in unserem Fall „FOX 57“ oder FOXS. Dieser berechnet den Berührungsort und verwendet diese Information im ausführenden eigerScript Programm.

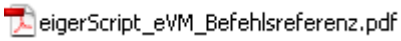
Inhalt der DemoCD

CD – Verzeichnis „Programme“

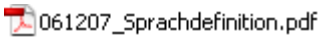
- eigerStudio (Entwicklungsumgebung für eigerScript)
- eigerGraphic Suite (Grafikprogramm)
- .NET Framework Redistributable (.NET Runtimes) v2.0 – Freeware für Windows 98/2000/Me/NT/XP
- Adobe Reader v8.0 – Freeware für Windows 2000/XP/Vista

Siehe auch Kapitel „Installation“ (S.18 ▶) und „eiger Programme“ (S.19 ▶).

CD – Verzeichnis „Dokumentation“



eigerScript-Befehlssammlung und Beschreibung der Befehle.



Erläuterungen zu Struktur und Regeln von eigerScript.

CD – Verzeichnis „Schnelleinstieg“

Im Verzeichnis „Schnelleinstieg“ finden Sie die Verzeichnisse zu den Übungen 1-12. Sie enthalten die vorgefertigten eigerScript-Dateien mit dem fertigen Programmcode im EVS (unkompilierter Code) und im EVI-Format (kompilierter Byte-Code) abgelegt, die wir im Laufe der jeweiligen Übung erstellen werden. Dies erlaubt es Ihnen, bei irgendeiner Übung neu einzusteigen oder gewisse Übungen zu überspringen. Wollen Sie beispielsweise bei Übung 5 einsteigen, können Sie

dafür die Programmdateien inkl. Ordner-Struktur der Übung 4 als Ausgangsbasis verwenden. Voraussetzung dafür sind allerdings die Grundkenntnisse über die Struktur eines eigerProjekts, die in Übung 1 vermittelt werden.

„eigerScript-Schnelleinstieg.pdf“ ist die PDF-Version des Schnelleinstiegs-Tutorials, das Sie gerade lesen.

CD – Verzeichnis „Diaschau“

Die vorgefertigte Diaschau können Sie auf dem FOX laufen lassen. Mehr dazu erfahren Sie in der Übung 4 (S.47 ▶).

CD – Verzeichnis „Application Notes“

Der eigerScript-Schnelleinstieg enthält nur die wichtigsten Grundlagen für das Programmieren in eigerScript. Weiterführende Informationen und anwendungsorientierte Beispiele zu ausgewählten Themen sind auf der CD als Application Notes abgelegt. Diese Rubrik wird zur Zeit laufend erweitert und aktualisiert. Den neuesten Stand finden Sie auf der Download-Seite von www.eigergraphics.com.

CD – Verzeichnis „HeaderDateien“

Das Erstellen eines eiger-Programmscripts im eigerStudio ist nur möglich, wenn der Programmcode mit den Header-Dateien verknüpft ist. Die Header-Dateien enthalten wichtige Funktions-, Register- und Konstantendefinitionen (Tabelle 1). Diese Dateien können auch in der Projektdefinitionsdatei vorhanden sein. Die Information der HeaderDateien wird beim Kompilieren in die ausführbare EVI-Datei integriert (vgl. Erstellen einer ausführbaren Datei für den FOX, S.34). Deshalb müssen diese Dateien nicht auf die CompactFlash Card übertragen werden.

Tabelle 1:
Header-Dateien und deren Inhalt

Datei	Inhalt
DEF_eVM_OpCodes.h	Beinhaltet die Definitionen aller Tokens von eigerScript sowie deren Kodierung. Tokens sind OpCodes, die die eVM veranlassen, eine bestimmte Operation auszuführen.
DEF_eVM_Registers.h	Register der eiger viertuellen Maschine eVM
DEF_eVM_Functions.lib	Funktionsbibliothek, Liste aller Funktionen mit Parametern
DEF_eiger_Colors.INC	Farb-Definitionen
DEF_eiger_Types.INC	Schrift-Typen, die vordefiniert sind
DEF_eiger_StackMachine.INC	

Die Header-Dateien dürfen nicht verändert werden. Sie können mit einem Texteditor (z.B. Microsoft Editor) geöffnet und gelesen werden.

Wichtig: In diesem Schnelleinstieg und den dazugehörenden Übungen arbeiten wir mit Versionen der Headerdateien, die möglicherweise inzwischen nicht mehr dem neusten Stand entsprechen. Den jeweils aktuellsten Stand finden Sie auf der Downloadseite von www.eigergraphics.com.
Das Kompilieren ist nur erfolgreich, wenn die Verweise im Programmscript mit den Namen der betreffenden Headerdateien übereinstimmen.

Installation

Für die Entwicklung von eigerScript-Projekten brauchen Sie folgende Programme:

Tabelle 2:
eigerScript-Programme auf der CD

Programm	Programm-Datei	Pfad auf der CD
eigerStudio	eigerStudio.exe	Programme\eigerStudio
eigerGraphic Suite	EigerGraphicSuite.exe	Programme\eigerGraphicSuite

Die Programme der eigerScript Entwicklungsumgebung brauchen nicht speziell installiert zu werden. Kopieren Sie die Programm-Dateien auf den Desktop oder in ein eigenes Verzeichnis. Durch Doppelklick auf die EXE-Datei öffnet sich das gewünschte Programm und Sie können mit der Arbeit beginnen.

Wichtig: Für den Betrieb der eiger Software muss die Programmschnittstelle **Microsoft .Net Framework** ab Version 1.0 auf dem PC installiert sein. Diese ist ab Windows XP, Service Pack 1 automatisch dabei. Microsoft .Net Framework kann vom Internet gratis heruntergeladen werden: www.microsoft.com .

eiger-Programme

Zur Entwicklung der FOX-Anwendungssoftware stehen die Programme der eiger-Familie zur Verfügung:

EigerStudio	→	programmieren und kompilieren
eigerGraphic Suite	→	Bilder bearbeiten und konvertieren
eigerFont Editor	→	Schriften und Symbole erstellen (im DemoKit nicht vorhanden)

eigerStudio

Die IDE *eigerStudio* unterstützt den Software-Entwickler mit modernen Hilfe-Tools.

eigerStudio ist eine moderne integrierte Entwicklungsumgebung IDE (Abbildung 6). Das Programmieren wird unterstützt durch Syntax-Coloring und eigerTIP, der eingebauten Hilfe, mit der Befehle schnell aus Listen ausgewählt werden können. Im eigerStudio ist auch der eigerCompiler integriert, der den ByteCode (*.EVI) erzeugt, welcher von der virtuellen Maschine eVM ausgeführt wird.

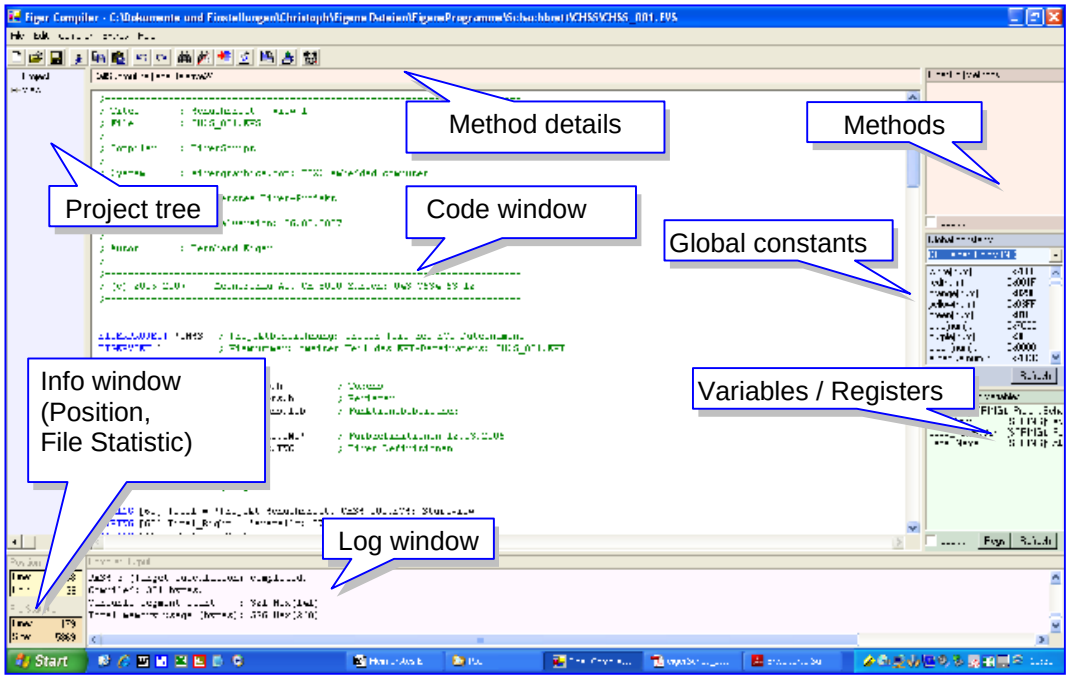


Abbildung 6: Aufbau der Entwicklungsumgebung eigerStudio.

Tabelle 3: Elemente der Benutzeroberfläche von eigerStudio (vgl. Abbildung 6).

Window	Erläuterung
Code window	Editor zum Erstellen, Anzeigen und Bearbeiten von eigerScript-Code. Das Code window färbt Befehle und Schlüsselwörter automatisch ein. Der Sourcecode ist eine Textdatei mit der Endung EVS.
Methods	Auflistung von Methoden für die eigerScript-Klassen. Für die Anzeige dieser Methoden greift eigerStudio auf die LIB-Datei *DEF_eVM_Functions.lib* zu.
Method details	Fokussiert auf die Methode, die im Methods-Fenster aktiviert ist. Hier ist der ganze Ausdruck sichtbar, welcher im Methods-Fenster wegen Platzmangels abgeschnitten sein kann. Die Parameter der Methode sind sichtbar.
Global constants	Liste der Konstanten (Farben, Schriftfonts etc.), die in den Header-Dateien *DEF_eiger_Colors.INC* und *DEF_eiger_Types.INC* definiert sind.
Variables / Registers	Liste der im Programmcode deklarierten Projekt-Variablen bzw. Liste der Register der virtuellen Maschine des FOX, die in der Datei *DEF_eVM_Registers.h* definiert sind.
Log window	Fehleranzeige und Log-Fenster des Compilers. Unter *Extras* > *Options* > *Compiler : Verbosity depth of Compiler Log* kann die Detailtiefe der Aufzeichnungen eingestellt werden. Stufe 3 ist sehr detailliert und macht das System entsprechend langsamer.
Info window	Das Fenster „Position“ dient der Navigation zu einer bestimmten Stelle im Script (*Doppelklick auf Fensterfläche* > *Goto Zeilen-#*). Das Fenster „Statistic“ enthält die Anzahl Zeilen des View-Scripts und Grösse der View-Datei in Kilobyte.
Project tree	Noch nicht aktiv.

eigerGraphic Suite

Die eigerGraphic Suite ist eine Programmsammlung, mit welcher wir Bildressourcen für ein Graphical User Interface GUI bearbeiten können. Sie ist eine Weiterentwicklung des *eigerGraphic File Converters*. Inzwischen vereint die eigerGraphic Suite sieben Applikationen zur Bildbearbeitung und -konvertierung (Abbildung 7 und Tabelle 4).

Gratis Download der eigerGraphic Suite Basisversion: www.eigergraphics.com .

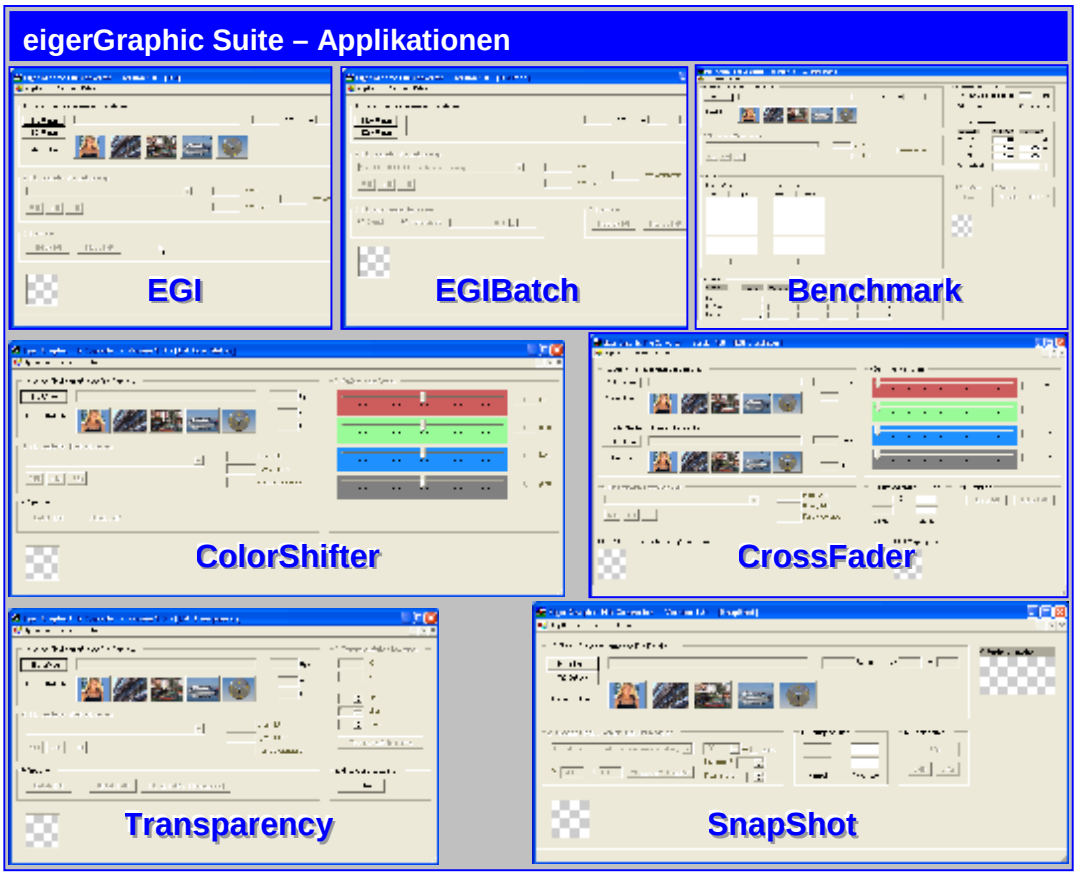


Abbildung 7: Applikationen zur Bildverarbeitung und -konvertierung in der eigerGraphic Suite.

Tabelle 4: Applikationen der eigerGraphic Suite (vgl. Abbildung 7).

Applikation	Erläuterung
EGI	Grafikelemente wie Buttons, Fotos und Grafiken werden mit der EGI-Applikation der *eigerGraphic Suite* in das *.EGI-Format konvertiert und komprimiert. Der Konverter kann *.GIF, *.BMP, *.JPG und *.PNG in das EGI-Format wandeln. Der Konverter kann EGI-Bilder auch zurücklesen und als *.BMP speichern. Die Funktionen der EGI-Applikation sind in der Übung 4 an einem Beispiel genauer erklärt .
EGIBatch	...
Benchmark	...
ColorShifter	...
CrossFader	...
Transparency	...
SnapShot	...

EGI – das Bild-Format für FOX

Das EGI-Bildformat (**e**iger**G**raphic **I**mage) wurde von der Firma S-TEC electronics AG, Zug entwickelt. Es ist ein leistungsfähiges highcolor Bildformat, das einen Bildinhalt verlustlos komprimiert. Es unterstützt das LCD-Touchpanel optimal und übernimmt die Vorteile gängiger Formate, ohne deren Nachteile einzugehen (Tabelle 5).

Das EGI-Format bietet:

- 32'768= 3x5bit Farbtiefe (ausreichend für Fotos)
- verlustlose RLE-Kompression
- schnellstmögliche Verarbeitung auf dem eigerGraphics Betriebssystem
- für Fotos und Grafik geeignet
- transparente Bilder möglich

Tabelle 5:
Vor- und Nachteile der gängigen Bildformate JPG, GIF, BMP und PNG).

Bildformat	Vorteile	Nachteile
JPG (Joint Photographic Experts Group)	• relativ kleine Dateien möglich (je nach Komprimierungsgrad)	• Komprimierung verlustbehaftet; für Grafiken deshalb nicht geeignet. • Dekomprimierung erfordert hohe Rechenleistung (oder Hardwaredekompression) • lizenzpflichtig
GIF (Graphics Interchange Format)	• sehr kompakte Komprimierung	• nur 256 Farben; für Fotos deshalb nicht geeignet. • Dekomprimierung erfordert hohe Rechenleistung • bis vor kurzem lizenzpflichtig
BMP (Windows Bitmap)	• verlustfrei (jeder Pixel wird gespeichert)	• benötigt viel Speicherplatz (VGA ca. 0.9 MB pro Bild)
PNG (Portable Network Graphics	• verlustfreie Komprimierung	• Dekomprimierung erfordert hohe Rechenleistung

eigerFont Editor

Standardmässig stehen in eigerScript 18 verschiedene Schriftfonts bzw. Ziffern-Fonts zur Verfügung (vgl. Tabelle 6). Mit Hilfe des eigerFont Editors lässt sich die Schriftenauswahl nach eigenen Wünschen beliebig erweitern (Abbildung 8). Sie können damit entweder einen vorhandenen TrueType-Font importieren – z.B. aus dem Font-Verzeichnis von MS Office – oder auch einen ganz neuen Font gestalten.

Der eigerFont Editor ist das einzige kostenpflichtiges Zusatzprogramm und nicht in der Standardausrüstung enthalten.

Abbildung 8:
Screenshot des FontEditors.

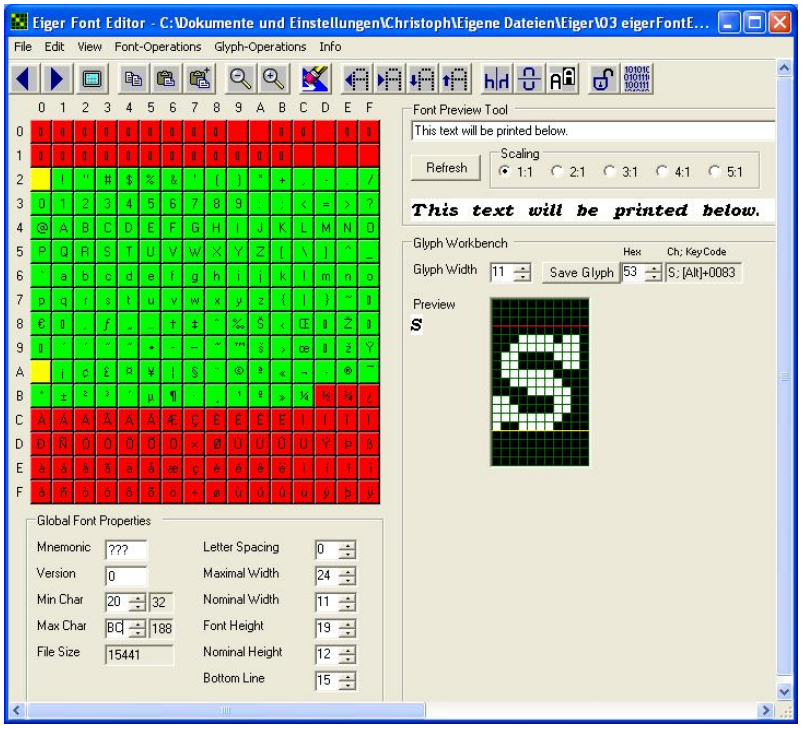


Tabelle 6: Standard-Schriftarten in eigerScript

Konstanten-Name in eiger Script	Font und nähere Bezeichnung
Font_Arial_7	System 7, Mikroschrift
Font_Arial_8	System 8, Kleinschrift
Font_Arial_10n	Arial 10, Normalschrift
Font_Arial_12n	Arial 12, Normalschrift
Font_Arial_14n	Arial 14, Normalschrift
Font_Arial_16n	Arial 16, Normalschrift
Font_Arial_20n	Arial 20, Normalschrift
Font_Arial_24n	Arial 24, normal System Grossschrift
Font_System_9n	System 9, Normalschrift
Font_System_9bd	System 9 bold, Titelschrift fett
Font_System_18bd	System 18 bold, Titelschrift fett
Font_Courier_9n	Courier 9 bold, FixedSpace Schrift
Font_Tekton_bold_24	Tekton 24 bold, Spezialschrift FW
Font_DigitalNumbers_16	Spezialschrift Digitalanzeige nur Ziffern und Punkt.
Font_DigitalNumbers_24	Spezialschrift Digitalanzeige nur Ziffern und Punkt.
Font_DigitalNumbers_32	Spezialschrift Digitalanzeige nur Ziffern und Punkt.
Font_DigitalNumbers_48	Spezialschrift Digitalanzeige nur Ziffern und Punkt.
Font_DigitalNumbers_64	Spezialschrift Digitalanzeige nur Ziffern und Punkt.

In der Application Note „Fonts und ASCII-Codes“ sind die Standard-Schriftarten von eigerScript dargestellt. Eine weitere Application Note „Neuen Font einbetten“ führt Sie in die Funktionen des eigerFont Editors ein und zeigt Ihnen, wie Sie eigene Fonts für Ihre Anwendungen importieren bzw. bearbeiten können. Sie finden die Application Notes auf der eigerDemoCD und auf der Download-Seite von www.eigergraphics.com.

Mein erstes eigerScript-Projekt – Übungen 1-12

In unserem ersten eigerScript-Projekt entwickeln wir eine Startseite (Startview), die zu einer Schachbrett-Seite mit verschiedenen Funktionen führt. Die einzelnen Übungen dieses Schnelleinstiegs bauen aufeinander auf. Sie finden in der beiliegenden CD die Dateien, die für diesen Lehrgang nötig sind. Zu jeder Übung besteht ein eigener Ordner (z.B. Script Übung01). Ein Übungsordner enthält den Projekt-Ausführungsstand vom Ende der betreffenden Übung. Dies bezieht sich auf den Projektordner inkl. alle Dateien. Wenn Sie beispielsweise eine Übung überspringen oder nur durchlesen und die darauffolgende Übung wieder am PC und am FOX nachvollziehen wollen, können Sie auf den bereitgestellten Daten der vorangegangenen Übung aufbauen.

Tabelle 7: Zeitbedarf für die Übungen.

Zeitbedarf für die Übungen (inkl. Aufgaben)

Übung	Zeitbedarf in Minuten
Übung 1: Vorbereitungen	10
Übung 2: Programmieren einer Startview	25
Übung 3: Bild in Startview laden	5
Übung 4: Eigene Bilder für den FOX vorbereiten	15
Übung 5: Titelleiste konfigurieren	15
Übung 6: Kontrollierter View-Aufbau	5
Übung 7: Buttons, HotSpots und HotKeys	30
Übung 8: HotSpot mit Screenshot-Funktion	15
Übung 9: Eine zweite View erstellen	60
Übung 10: Includefile	35
Übung 11: Schachbrett zeichnen	55
Übung 12: Timer	40
Total	**310 Min.**

Übung 1: Vorbereitungen

In dieser Übung lernen Sie

- ✓ Ihren PC für Ihr eigerScript-Projekt optimal vorzubereiten.
- ✓ eine Startdatei für Ihre eigerScript-Anwendung anzulegen.

Zeitbedarf: 10 Minuten

Daten für die Übung: Dateien im Ordner „HeaderDateien“ und FOXLOGOL.EGI aus dem Ordner „Bilder“.

Struktur eines eigerProjekts

Ein eigerScript-Projekt ist in eine bestimmte Ordner- und Dateistruktur eingebettet (vgl. Abbildung 9).

Zunächst erstellen wir einen Projekt-Ordner mit dem Namen „Schachbrett“ und darin den Unterordner: „CHSS“ (dieser Ordnername darf maximal vier Zeichen enthalten; CHSS steht für Chess, engl. Schach).

Innerhalb des CHSS-Ordners legen wir nochmals vier Unterordner an:

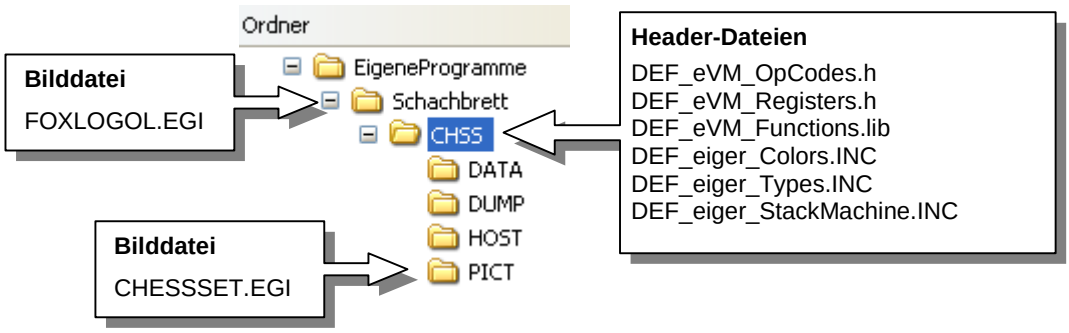
- ➢ DATA
- ➢ DUMP
- ➢ HOST
- ➢ PICT

Die **Header-Dateien** kopieren wir direkt in den Unterordner „CHSS“ (Abbildung 9).

Während des Aufstartens und Ladens unseres Projekts verlangt die virtuelle Maschine des FOX nach der Bilddatei **FOXLOGOL.EGI**. Diese legen wir direkt in den Projekt-Ordner „Schachbrett“ (Abbildung 9 und Abbildung 11).

Für unser erstes eigerScript-Projekt legen wir die Bilddatei „**CHESSSET.EGI**“ im Unterordner CHSS\PICT bereit (Abbildung 9).

Abbildung 9:
Ordnerstruktur für das „Schachbrett-Projekt“.



eiger**G**raphic **I**mage (EGI) ist ein speziell für FOX entwickeltes Bildformat mit verlustfreier Komprimierung. Dieses Format kann mit Hilfe des Konvertierungstools *eigerGraphic Suite* erstellt werden. Umgekehrt können Bilder, die im EGI-Format vorliegen, mit der eigerGraphic Suite wieder in ein PC-Bildformat umgewandelt werden.

Startdatei START.FOX

Beim Aufstarten des FOX wird die eVE initialisiert, dann wird nach einer eingesteckten Compact Compact Flash Card gesucht und die Datei FOXLOGOL.EGI (Format Landscape) bzw. FOXLOGOP.EGI (Format Portrait) dargestellt. Die Datei START.FOX wird eingelesen, in den Projekt-Ordner verzweigt und die View 001.EVI geladen.

Die formatlose Startdatei START.FOX müssen wir erst erstellen. Dafür verwenden wir einen Text-Editor (z.B. Microsoft Editor). START.FOX enthält nichts mehr und nichts weniger als einen Verweis auf unseren CHSS-Ordner (Abbildung 10).

Erstellen von START.FOX:

- Microsoft Editor aufrufen: *Start* > *Programme* > *Zubehör* > *Editor*
- Eingabe: **START=CHSS;** – Semikolon nicht vergessen! (vgl. Abbildung 10)
- Speichern der Datei unter \Schachbrett\START.FOX (vgl. Abbildung 11)

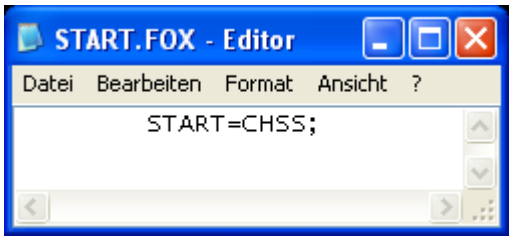


Abbildung 10: Inhalt der Startdatei START.FOX. Wichtig sind ein Tabulator am Anfang und ein Semikolon am Ende.

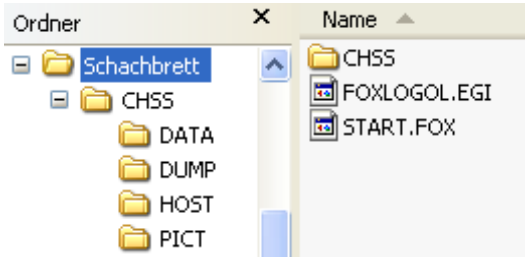


Abbildung 11: Speicherort für FOXLOGOL.EGI und START.FOX

Übung 2: Programmieren einer Startview

In dieser Übung lernen Sie

- ✓ den Grundaufbau der Elemente eines eigerScript-Programmes kennen,
- ✓ die integrierte Entwicklungsumgebung eigerStudio kennen,
- ✓ eine einfache Startview programmieren,
- ✓ ein eigerScript-Programm kompilieren und auf Compact Flash Card übertragen.
- ✓ die Applikation auf dem FOX ausführen.

Zeitbedarf: 25 Minuten

Daten für die Übung: Projektordner „Schachbrett“ aus der Übung 1, den Sie selbst erstellt haben oder vorgefertigt im Ordner „Schnelleinstieg/Übung01“ vorfinden.

Projektdatei und Views

Views sind die Programm-Module von eigerScript.

Ein eigerScript-Projekt besteht aus einer **Projektdatei** und aus einer oder mehreren **Views**.

Die Projektdatei beinhaltet Elemente, welche das ganze Projekt betreffen, d.h. auf welche von jeder View aus zugegriffen werden kann. So werden beispielsweise globale Konstanten und Variablen in der Projektdatei deklariert.

Eine View ist jeweils ein Programm-Modul, welches eine eigene ausführbare Datei darstellt. Eine solche Datei enthält den Programmcode, der das betreffende View und seine Objekte steuert. Es ist immer nur eine View aktiv.

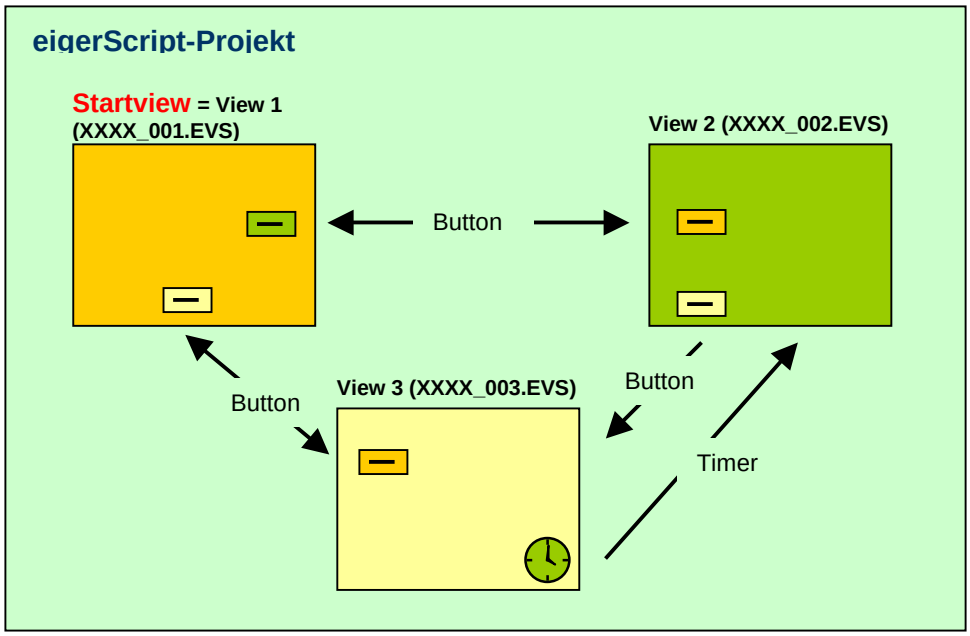
Beim Aufstarten eines eigerScript-Programms wird zuerst die **Startview** aktiviert. Von dieser Startview aus können andere Views des Projekts aktiviert werden. Dies läuft nach folgender Regel ab:

1. Laden der neuen View in den Programmspeicher,

2. ausführen des Hauptprogramms (`BEGINVIEW` bis `ENDVIEW`) der neuen View.

Der Wechsel von einer View zur andern ist entweder programmintern geregelt oder wird durch bestimmte Events (Ereignisse) von aussen ausgelöst. Ein solcher Event ist beispielsweise die Berührung eines Buttons auf der TouchScreen oder ein Timer (vgl. Abbildung 12).

Abbildung 12: Beispiel eines eigerProjekts mit drei Views inkl. Startview. Das Projekt wechselt die aktive View aufgrund der Events Button und Timer.



Den Code für das Beispiel-Projekt von Abbildung 12 finden Sie auf der CD im Ordner „Schnelleinstieg\Beispiel-eigerProjekt“.

View-Bezeichnung

Ein View-Name hat die Struktur XXXX_NNN.EVS

Zum Erstellen des Scripts für die Startview öffnen wir das „eigerStudio“ (Abbildung 6). Im zentralen weissen Editor-Fenster (Code Window) schreiben wir das Programmscript für unsere erste View.

Der Name der View-Datei ist dabei wie folgt strukturiert: **XXXX_NNN.EVS**

XXXX:	beliebiger Projektname (z.B. CHSS), welcher dem Eintrag in der Datei „START.FOX“ entspricht (vgl. Abbildung 10).
NNN:	aufsteigende Viewnummer; erste View *muss* mit „001“ nummeriert sein.
EVS:	steht für **e**iger**V**iew-**S**ource

Unsere Startview heisst demnach: **CHSS_001.EVS**

Unter diesem Namen speichern wir mit „*File > Save as...*“ das noch leere eigerView-Script in den Ordner „CHSS“.

Zur ersten View gehört obligatorisch die Angabe des eiger-Projektnamens (z.B. „CHSS“, genau 4 Zeichen lang) und die Nummer der betreffenden View. Dazu verwenden wir die eigerScript-Schlüsselwörter „EIGERPROJECT“ und „EIGERVIEW“ (Beispiel-Code 1).

Beispiel-Code 1 (Angabe des vierziffrigen Projektnamens und der Viewnummer)

```
EIGERPROJECT 'CHSS'  ; Projektbezeichnung: 1.Teil d. EVI-Dateinamens
EIGERVIEW 1   ; Viewnummer: 2.Teil des EVI-Dateinamens: CHSS_001.EVI
```

Aus diesen Angaben generiert der Compiler beim nächsten Kompiliervorgang den Dateinamen der ausführbaren Datei „**CHSS_001.EVI**“. Die virtuelle Maschine des FOX führt diese EVI-Datei (**e**iger**V**iew **I**nterpretable) aus, die das in Byte-Code übersetzte eigerView-Script enthält.

Technische Angaben am Script-Anfang

Die ersten Zeilen überhaupt dienen uns für einige technische Angaben zum CHSS-Projekt. Dazu fügen wir ganz oben im Code Window (vor Beispiel-Code 1) folgenden Block mittels Copy/Paste ein (Beispiel-Code 2).

Beispiel-Code 2 (Einführender Kommentar mit erklärenden bzw. technischen Projekt- und Viewdaten)

Kommentarzeilen beginnen mit einem Semikolon (;)

```
;-----------------------------------------------------------------
; Titel       : Schachbrett   View 1
; File        : CHSS_001.EVS
;-----------------------------------------------------------------
; Compiler    : eigerScript
;
; System      : eigergraphics.com; FOXS embedded computer
;
; Beschreibung: Mein erstes eiger-Projekt
;
; Version     : Initialversion: 06.03.2007
;
; Autor       : Bernhard Eiger
;
;-----------------------------------------------------------------
; (c) 2005-2007     MeineFirma AG, CH-8000 Zürich; 043 7634 53 12
;-----------------------------------------------------------------
```

Diese Zeilen haben reinen Kommentarcharakter und werden deshalb jeweils am Zeilenanfang mit einem Semikolon (;) eingeführt.

Wir speichern unsere ersten Zeilen mit *File > Save As...* oder mit dem Save-Button  in der Symbolleiste.

Syntax Coloring. Im Code Window erhält jeder Code-Typ seine Farbe.

Während wir den Code eintippen werden wir unterstützt durch das Syntax Coloring. eigerStudio erkennt einzelne Worte, Wortfolgen oder Zeilen als Kommentar, eigerScript-Schlüsselwort etc. und färbt diese automatisch entsprechend ein:

blau:	eigerScript-Schlüsselwort
grün:	Kommentar
rot:	Label (Sprungziel)
	Funktionsname
violett:	eigerScript-Funktion; definiert durch Eintrag in der Funktionsbibliothek (FUNCLIB)
schwarz:	übriger Programmcode

Wenn Sie Programm-Code aus einer anderen Anwendung per Copy/Paste einfügen, kann es vorkommen, dass sich das Syntax Coloring nicht oder nur teilweise auf die eingefügten Zeilen auswirkt. In diesem Fall können Sie mit dem Refresh-Button  in der Symbolleiste den Code wieder korrekt einfärben lassen.

Eine weitere nützliche Eingabehilfe bietet das eigerTip. In den Fenstern am rechten Rand der eigerStudio-Entwicklungsumgebung können wir in den Listen nach den vorhandenen Befehlen, Methoden, Konstanten, Registern und Variablen suchen. Diese lassen sich durch Doppelklick in den Programm-Code einfügen. Zudem wird im Method Details Window die volle Syntax einer Methode angezeigt, wenn wir diese in der Methods Liste anwählen.

Verweis auf eigerScript-Definitionsdateien (Header-Dateien)

Die eigerScript-Definitionsdateien sind fester Bestandteil einer View.

Der eigerCompiler muss auf die eigerScript-Definitionsdateien (Header-Dateien) zugreifen können, damit er ein View-Script in die Maschinensprache übersetzen kann. Mit den entsprechenden Import- und Include-Befehlen binden wir die benötigten Header-Dateien in die View ein (Beispiel-Code 3).

Beispiel-Code 3 (Befehle zur Integration der Headerdateien)

```
IMPORT      'DEF_eVM_OpCodes.h'           ; Tokens
IMPORT      'DEF_eVM_Registers.h'    ; Register
FUNCLIB     'DEF_eVM_Functions.lib'  ; Funktionsbibliothek

INCLUDEFILE 'DEF_eiger_Colors.INC'   ; Farbdefinitionen
INCLUDEFILE 'DEF_eiger_Types.INC'    ; eiger Definitionen
```

Die hier zitierten Header-Dateien haben wir bei der Vorbereitung des Projekts bereits im CHSS-Ordner abgelegt. In diesem Schnelleinstieg arbeiten wir ausschliesslich mit Versionen der Headerdateien, die allenfalls nicht mehr dem neusten Stand entsprechen. Falls Sie über neuere Versionen verfügen – z.B. neuere Versionen von der Downloadseite auf www.eigergraphics.com heruntergeladen haben –, ist es Ihnen natürlich freigestellt, diese zu verwenden. Nur müssen Sie darauf achten, dass Sie im Script die Dateinamen mit dem entsprechenden Versionszusatz verwenden. Sie können diese Headerdateien zugunsten einer besseren Übersichtlichkeit auch in einem separaten Ordner ablegen, z.B. in einem Ordner namens DEF. Der Pfad wäre dann z.B. folgender: Schachbrett\CHSS\DEF\DEF_eiger_Colors.INC. In diesem Fall müssten Sie dann auch die betreffenden Programmscript-Zeilen mit dem **relativen Pfad** ergänzen (Beispiel-Code 4).

Beispiel-Code 4 (Befehle zur Integration der Headerdateien. In diesem Fall sind diese aus Sicht der EVS-Datei im Unterverzeichnis DEF abgelegt)

```
IMPORT      'DEF\\DEF_eVM_OpCodes.h'       ; Tokens
IMPORT      'DEF\\DEF_eVM_Registers.h'     ; Register
FUNCLIB     'DEF\\DEF_eVM_Functions.lib'   ; Funktionsbibliothek

INCLUDEFILE 'DEF\\DEF_eiger_Colors.INC'    ; Farbdefinitionen
INCLUDEFILE 'DEF\\DEF_eiger_Types.INC'     ; eiger Definitionen
```

In den Header-Dateien werden Opcodes (operation codes) und Register einem Klartext-Token zugeordnet. Das Dateiformat der Header-Dateien entspricht dem gewohnten Format, wie es in Assembler-Entwicklungs-systemen verwendet wird.

Hauptprogramm

Das Hauptprogramm der View wird zwischen `BEGINVIEW` und `ENDVIEW` gesetzt.

Alle Aktionen einer View werden vom Hauptprogramm aus gesteuert. Den Rahmen des Hauptprogramms bilden die Schlüsselwörter `BEGINVIEW` und `ENDVIEW` (Beispiel-Code 5). Die Befehle innerhalb des Hauptprogrammes werden nach

dem Starten einer View entsprechend ihrer Reihenfolge „von oben nach unten“ abgearbeitet. Die einzelnen Befehle können vom Hauptprogramm aus **Subroutinen (Prozeduren)** aufrufen, in welchen nähere Anweisungen zu den auszuführenden Aktionen programmiert sind. Durch das Auslagern detaillierter Ausführungsanweisungen kann das Hauptprogramm möglichst schlank gehalten werden.

Den Code für das Hauptprogramm plazieren wir zur leichteren Orientierung am besten immer ganz am Schluss des View-Scripts.

Beispiel-Code 5 (Schlüsselwörter für Beginn und Ende des Hauptprogramms)

```
BEGINVIEW

ENDVIEW
```

Wenn in der View das BEGINVIEW/ENDVIEW, d.h. das Hauptprogramm fehlt, reagiert der eigerCompiler beim nächsten Kompilierbefehl mit einer Fehlermeldung und bricht den Vorgang ab.

Subroutinen (Prozeduren)

Subroutinen werden aus dem Hauptprogramm oder anderen Subroutinen aufgerufen.

Subroutinen (Prozeduren) helfen, das Hauptprogramm möglichst schlank zu halten. Sie stellen kleine Programmelemente dar, die aus dem Hauptprogramm aufgerufen werden können. Subroutinen enthalten in der Regel nähere Anweisungen zu den im Hauptprogramm auszuführenden Aktionen. Der hauptsächliche Einsatz ist die Deklaration von Ereignisprozeduren für Hotspots, Hotkeys und Serial-Events.

Subroutinen besitzen einen Namen, über den sie aufgerufen werden. Ohne einen entsprechenden Aufruf im Hauptprogramm stehen Subroutinen als Waisen da und haben keine Wirkung auf den Programmablauf bzw. auf die Darstellung der View.

Eine Subroutine wird zwischen SUB und ENDSUB gesetzt.

Der Code wird durch die Schlüsselworte SUB und ENDSUB geklammert.

Unsere erste Subroutine werden wir in Übung 5 schreiben ▶.

Initialisieren der Grafikmaschine EVE

Am Anfang des Hauptprogramms steht immer der `EVE.Init`-Befehl.

Am Anfang des Hauptprogramms sollte generell der `EVE.Init`-Befehl stehen (Beispiel-Code 6). Dadurch wird die Grafikmaschine EVE (eigerVideoEngine) frisch initialisiert, bevor irgendwelche andere Aktionen der View ausgeführt werden. EVE wird so von älteren Einstellungen „gereinigt“ und kann unbelastet neue Befehle entgegen nehmen, ohne dass wir Überraschungen aufgrund vorhergehender Einstellungen befürchten müssen.

Beispiel-Code 6 (der Initialisierungsbefehl gehört an den Anfang jedes Hauptprogrammes)

```
BEGINVIEW
    EVE.Init()                          ; EVE ANNA initialisieren

ENDVIEW
```

ANNA ist der Name der Grafikmaschine (vgl. Abbildung 1, S.9).

Der eigerCompiler ist mit einer bequemen Schreibhilfe ausgestattet. Sobald wir im eigerEditor beginnen ein Wort einzutippen, erscheint im rosaroten eigerTip-Fenster eine Liste aller eigerScript-Methoden mit dem gleichen Wortbeginn. Ein Doppelklick auf die gewünschte Methode genügt, um das angefangene Wort syntaktisch korrekt zu vervollständigen.

Erstellen einer ausführbaren Datei für den FOX

Eine kompilierte, ausführbare Datei ist im EVI-Format geschrieben (XXXX_NNN.EVI).

Die virtuelle Maschine des FOX kann die EVS-Datei nicht lesen. Wie bereits angetönt, müssen wir unser eigerView-Script in eine Sprache übersetzen (kompilieren), die von FOX gelesen und ausgeführt werden kann. Diese Übersetzung in Byte-Code nimmt uns der eigerCompiler ab. Er legt den Byte-Code in einer neuen Datei ab, welche in unserem Fall dank Beispiel-Code 1 (S.30) automatisch den Namen CHSS_001**.EVI** erhält.

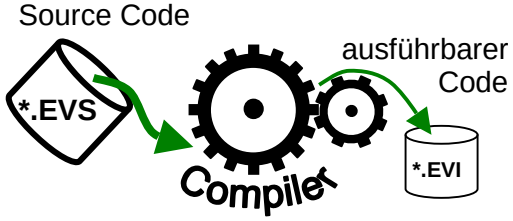


Abbildung 13: Der Compiler übersetzt den in eigerScript geschriebenen Source Code in einen ausführbaren Code für den FOX.

Vor dem ersten Kompilierbefehl füllen wir das Hauptprogramm übungshalber mit einem vorgefertigten Code, der im FOX die Bildschirmanzeige löscht und den Schriftzug „Hello World!“ erzeugt (Beispiel-Code 7). Um die Bedeutung der einzelnen Befehle dieses Codes kümmern wir uns jetzt noch nicht. Wir werden diese im Laufe der folgenden Übungen im einzelnen verstehen lernen.

Beispiel-Code 7 („Hello World!“-Code)

```
BEGINVIEW
    EVE.Init()                ; EVE ANNA initialisieren
    Display.ClearColor(aquamarine)
    Load.Geometry_XYWH(215,180,210,60)
    eI.SpaceLeft := 8         ; Eigenschaften für Hello World!
    eI.SpaceRight := 8
    eI.HorizontalAdjust := 0
    eI.VerticalAdjust := 0
    eI.Position := Pos_center
    eI.Orientation := Orientation_0deg
    eI.Appearance := normal
    eI.BorderStyle := color_button_3D_raised
    eI.FontNumber := Font_Arial_24n
    Label.Color(blue115)
    Label.Text('Hello World!')

    LOOP
    ENDLOOP
ENDVIEW
```

Sofern unser bisheriges eigerView-Script nun dem Beispiel-Code 8 entspricht, steht einem ersten Kompiliervorgang nichts mehr im Wege. Vorher speichern wir aber unsere aktuelle EVS-Datei nochmals.

Beispiel-Code 8 (Script von View 1 vor dem ersten Kompiliervorgang)

```
;-------------------------------------------------------------------------
; Titel       : Schachbrett   View 1
; File        : CHSS_001.EVS
;-------------------------------------------------------------------------
; Compiler    : eigerScript
;
; System      : eigergraphics.com; FOXS embedded computer
;
; Beschreibung: Mein erstes Eiger-Projekt
;
; Version     : Initialversion: 06.03.2007
;
; Autor       : Bernhard Eiger
;
;-------------------------------------------------------------------------
; (c) 2005-2007     MeineFirma AG, CH-8000 Zürich; 043 7634 53 12
;-------------------------------------------------------------------------


EIGERPROJECT 'CHSS'      ; Projektbezeichnung: erster Teil des EVI-Dateinamens
EIGERVIEW 1              ; Viewnummer: zweiter Teil des EVI-Dateinamens: CHSS_001.EVI


IMPORT                   'DEF_eVM_OpCodes.h'          ; Tokens
IMPORT                   'DEF_eVM_Registers.h'        ; Register
FUNCLIB                  'DEF_eVM_Functions.lib'      ; Funktionsbibliothek

INCLUDEFILE              'DEF_eiger_Colors.INC'       ; Farbdefinitionen 12.03.2006
INCLUDEFILE              'DEF_eiger_Types.INC'        ; Eiger Definitionen
```

```
BEGINVIEW
      EVE.Init()                              ; EVE ANNA initialisieren
      Display.ClearColor(aquamarine)
      Load.Geometry_XYWH(215,180,210,60)
      eI.SpaceLeft := 8
      eI.SpaceRight := 8
      eI.HorizontalAdjust := 0
      eI.VerticalAdjust := 0
      eI.Position := Pos_center
      eI.Orientation := Orientation_0deg
      eI.Appearance := normal
      eI.BorderStyle := color_button_3D_raised
      eI.FontNumber := Font_Arial_24n
      Label.Color(blue115)
      Label.Text('Hello World!')

      LOOP
      ENDLOOP

ENDVIEW
```

Der Kompilierbefehl kann im eigerStudio auf verschiedene Arten ausgelöst werden:

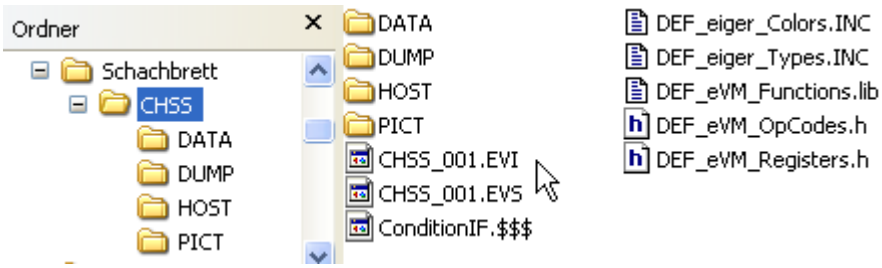
- via Standardleiste: *Compiler* > *Compile Source*
- Durch Mausklick auf den *Compile Source*-Button auf der Symbolleiste (vgl. Abbildung 14)
- mit Shortcut F5

Abbildung 14: Button in der Symbolleiste zum Kompilieren des aktiven eigerScripts

Beim Kompilieren wird die bisherige *.EVI-Datei überschrieben.

Während des Kompilierens generiert der eigerCompiler die ausführbare Datei CHSS_001.EVI am Speicherort der Source-Datei CHSS_001.EVS. Im CHSS-Ordner liegen nun sowohl die Source-Datei (*.EVS) als auch die ausführbare EVI-Datei (vgl. Abbildung 15). Bei jedem weiteren Kompiliervorgang wird die alte EVI-Datei mit der aktuellen Version überschrieben.

Abbildung 15: CHSS-Ordner mit der Source-Datei CHSS_001.EVS und der beim Kompilieren generierten Datei CHSS_001.EVI



Es kann sein, dass der eigerCompiler während des Kompilierens temporäre Ablage-Dateien mit der Erweiterung „**.$$$**“ anlegt. Diese haben keine weitere Bedeutung und können problemlos wieder gelöscht werden.

Übertragen der Startview auf den FOX

U002S001.EGI

Nun laden wir den ganzen CHSS-Ordner inklusive FOXOLOGOL.EGI und START.FOX auf die Compact Flash Card. Die Ordner-Struktur auf der Compact Flash Card sollte dadurch mit derjenigen in Abbildung 16 übereinstimmen.

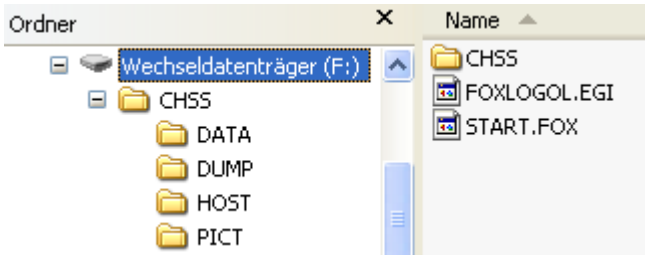


Abbildung 16: Ordner-Struktur auf der Compact Flash Card für die Übertragung auf den FOX.

Die geladene Compact Flash Card fügen wir im FOX ein und drücken Reset. Dies bewirkt einen Neustart des Betriebsystems. Am Ende des Aufstartens sollte nun unser „Hello World!“ auf dem Bildschirm erscheinen (vgl. Abbildung 17)

Abbildung 17: Darstellung von “Hello World!“ auf dem FOX

Übung 3: Bild in Startview laden

In dieser Übung lernen Sie

- ✓ Bilder in Views einbauen.

Zeitbedarf: 5 Minuten

Daten für die Übung: Projektordner „Schachbrett“ mit der Datei CHSS_001.EVS aus der Übung 2, den Sie selbst erstellt haben oder vorgefertigt im Ordner „Schnelleinstieg/Übung02“ vorfinden.

Anstelle von „Hello World!“ laden wir das Bild *CHESSSET.EGI* in die Startview, welches wir in der Übung 1 ◀ im PICT-Ordner bereitgelegt haben. Für die Darstellung dieses Bildes genügen dem FOX zwei Befehlszeilen, welche wir ins Hauptprogramm, d.h. zwischen `BEGINVIEW` und `ENDVIEW` einfügen, und zwar direkt nach dem `EVE.Init`-Befehl (Beispiel-Code 9). Den übrigen „Hello World!“-Code löschen wir wieder.

Beispiel-Code 9 (einbetten eines Bildes im EGI-Format in die View)

```
BEGINVIEW
    EVE.Init()                          ; EVE ANNA initialisieren
    Load.Pos_X1Y1(0,0)                  ; linke obere Ecke
    File.Read_EGI('C:\\CHSS\\PICT\\CHESSSET.EGI')
```

Mit der eigerScript-Funktion `Load.Pos_X1Y1(VarInt:X1,VarInt:Y1)` geben wir die Position für die linke obere Ecke des darzustellenden Bildes an (vgl. Abbildung 19). Die Grösse von CHESSSET.EGI stimmt genau mit den Abmessungen des FOX-Bildschirms überein (640 x 480 Pixel). Wenn wir also X=0 und Y=0 setzen, füllt unser Bild den Bildschirm genau aus. Wir werden in der nächsten Übung näher darauf eingehen, wie wir eigene Bilder für den FOX korrekt vorbereiten ▶.

`File.Read_EGI(VarStr:FileName)` ruft die gewünschte EGI-Bilddatei auf. Dazu ist in Klammern die Angabe des absoluten Pfades auf der Compact Flash Card und des Dateinamens notwendig (vgl. Beispiel-Code 9). Die Compact Flash Card gilt für den FOX als Laufwerk C.

Die Grafikmaschine EVE kann Bilddateien nur im **EGI-Format** lesen (**e**iger**G**raphic**I**mage). Die **eigerGraphic Suite** ist das Werkzeug, mit welchem übliche Bildformate ins EGI-Format konvertiert werden können. Darauf gehen wir in der folgenden Übung näher ein .

Nun komplettieren wir das Hauptprogramm noch mit dem Schleifenbefehl `LOOP` und `ENDLOOP`, unmittelbar vor dem `ENDVIEW` (vgl. Beispiel-Code 10). Dadurch wird verhindert, dass das Programm zur Compilermarke `ENDVIEW` gelangt. In diesem Fall würde das Programm seine Aufgabe als beendet auffassen und die Virtuelle Maschine verlassen. Das wäre gleichbedeutend mit einem System Reset.

Bis zu diesem Zeitpunkt sollte unsere Startview mit Beispiel-Code 10 übereinstimmen.

Beispiel-Code 10 (ganzer Programmcode für die Darstellung des Startview-Bildes CHESSSET.EGI auf dem Bildschirm Bildschirm des eigerPanels)

```
;------------------------------------------------------------------------
; Titel       : Schachbrett   View 1
; File        : CHSS_001.EVS
;------------------------------------------------------------------------
; Compiler    : EigerScript
;
; System      : eigergraphics.com; FOXS embedded computer
;
; Beschreibung: Mein erstes Eiger-Projekt
;
; Version     : Initialversion: 06.03.2007
;
; Autor       : Bernhard Eiger
;
;------------------------------------------------------------------------
; (c) 2005-2007     MeineFirma AG, CH-8000 Zürich; 043 7634 53 12
;------------------------------------------------------------------------


EIGERPROJECT 'CHSS'       ; Projektbezeichnung: erster Teil des EVI-Dateinamens
EIGERVIEW 1               ; Viewnummer: zweiter Teil des EVI-Dateinamens: CHSS_001.EVI


IMPORT          'DEF_eVM_OpCodes.h'          ; Tokens
IMPORT          'DEF_eVM_Registers.h'                ; Register
FUNCLIB         'DEF_eVM_Functions.lib'              ; Funktionsbibliothek

INCLUDEFILE     'DEF_eiger_Colors.INC'               ; Farbdefinitionen 12.03.2006
INCLUDEFILE     'DEF_eiger_Types.INC'                ; Eiger Definitionen


BEGINVIEW
      EVE.Init()                                                ; EVE ANNA initialisieren
      Load.Pos_X1Y1(0,0)                                        ; linke obere Ecke
      File.Read_EGI('C:\\CHSS\\PICT\\CHESSSET.EGI')

LOOP
ENDLOOP

ENDVIEW
```

Mit  (oder F5 oder *Compiler* > *Compile Source*) kompilieren wir unsere bisher programmierte Startview. Damit wird die bereits vorhandene Datei CHSS_001.EVI überschrieben und gleichzeitig auch die Datei CHSS_001.EVS wieder gesichert.

Die neu geladene Compact Flash Card fügen wir wiederum im FOX ein und starten mit Reset das Betriebssystem neu auf. Am Ende des Warmstarts sollte nun unser Schachfiguren-Bild CHESSSET.EGI den Bildschirm füllen (vgl. Abbildung 18).


U003S001.EGI

Abbildung 18:
Schachfiguren-Bild als Startseite View 1 (CHSS_001.EVS)

Übung 4: Eigene Bilder für den FOX vorbereiten

In dieser Übung lernen Sie

- ✓ eigene Bilder für den FOX vorbereiten.
- ✓ Bilder ins EGI-Format konvertieren.
- ✓ das Bildformat EGI kennen.
- ✓ eine Diashow aus eigenen Bildern erstellen.

Zeitbedarf: 15 Minuten

Daten für die Übung: Projektordner „Schachbrett“ mit der Datei CHSS_001.EVS aus der Übung 3, den Sie selbst erstellt haben oder vorgefertigt im Ordner „Schnelleinstieg/Übung03“ vorfinden.

.

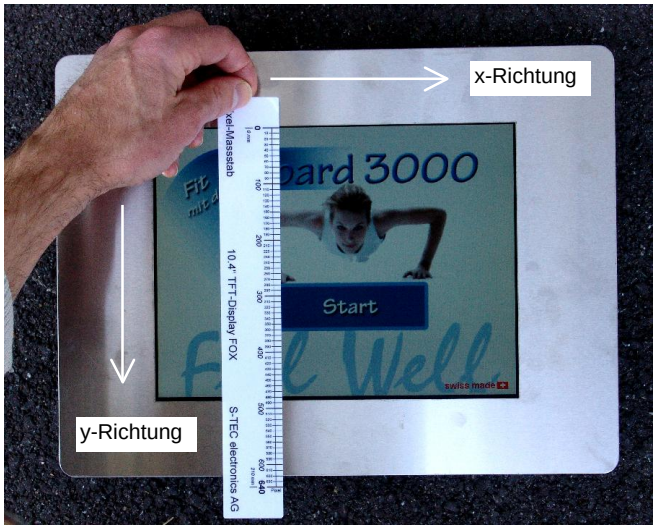


Abbildung 19: Der auf das Touchpanel abgestimmte Pixel-Massstab erleichtert das richtige Dimensionieren und Positionieren von Bildern, Buttons etc.

Bilder am PC richtig dimensionieren mit eiger Graphic Suite

Die virtuelle Maschine des FOX kennt keine Funktion, mit welcher die geladenen EGI-Bilder automatisch an die Bildschirmgrösse angepasst werden könnten. Wir

müssen deshalb die Bilder, die wir mit dem FOX darstellen wollen, zuvor auf dem PC genau dimensionieren. Dazu ist es wichtig, die Dimensionen des FOX-Bildschirms zu kennen.

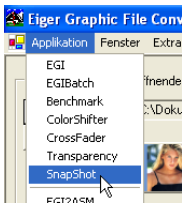
Ein Fenster füllendes Bild hat bei FOX die Grösse 640x480 Pixel (VGA-Format).

Der FOX-Bildschirm misst 640 Pixel in der Breite und 480 Pixel in der Höhe (vgl. Abbildung 19). dieses Format ist unter der Abkürzung VGA bekannt. Mit Hilfe eines Pixel-Massstabs können wir die optimalen Abmessungen und Positionen für Grafiken, Bildern oder Buttons ermitteln. Ein solcher Massstab ist Teil des eigerDemo-Kits.

Um unser Bild Bildschirm füllend auf dem FOX darzustellen, müssen wir es zuvor auf dem PC genau auf 640 x 480 Pixel zuschneiden bzw. vergrössern oder verkleinern. Mit der Applikation *SnapShot* der eiger Graphic Suite kommen wir rasch zum Ziel:

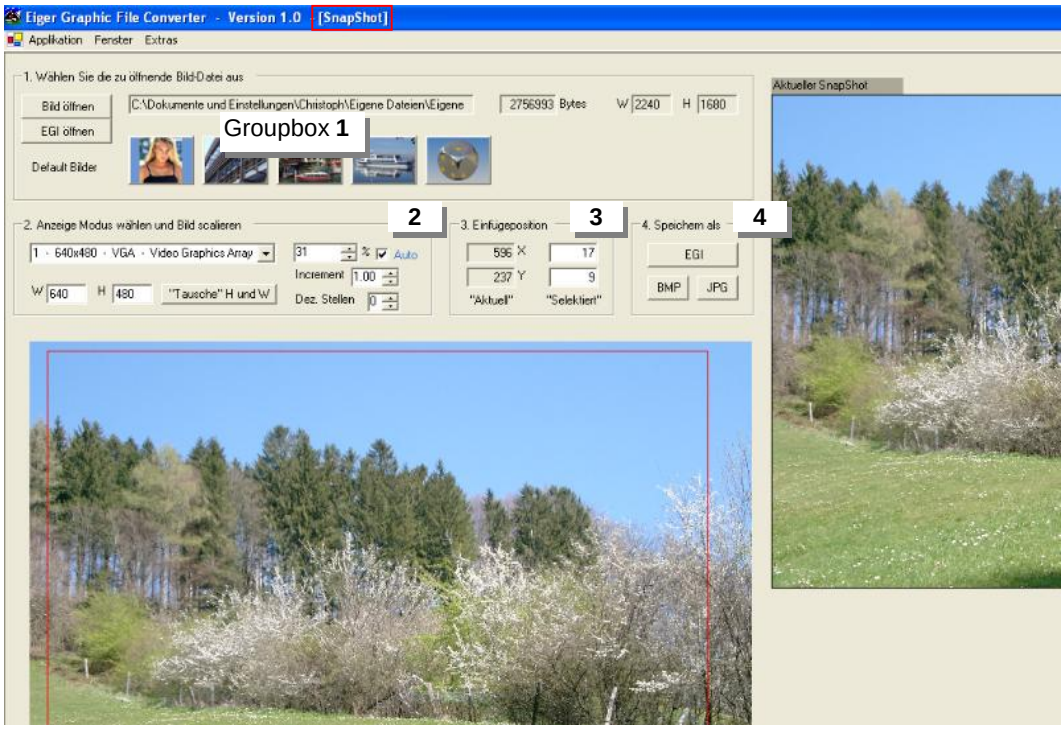
1. eiger Graphic Suite aufrufen.
2. Applikation für die Bild-Dimensionierung wählen: *Applikation* > *SnapShot* (vgl. Abbildung 20).

Abbildung 20:
Weg zur Applikation „SnapShot“ in der eiger Graphic Suite.



3. Bild laden: *Bild öffnen* (Button in der Groupbox 1; vgl. Abbildung 21) > im Dateibrowser gewünschtes Bild suchen > *Öffnen*. Die Datei wird geladen und unterhalb der Bedienelemente angezeigt. Auf der rechten Seite sind Dateigrösse, Breite und Höhe des Bildes (Anzahl Pixel) angegeben. In der Applikation können auch 5 Beispielbilder zum Üben abgerufen werden.

Abbildung 21:
Die Applikation „Shapshot“ in der eiger Graphic Suite dient zum Zuschneiden und Dimensionieren von Bildern.



4. Bildgrösse ändern oder Bild zuschneiden

   Grösse ändern: In *Groupbox 2* (Abbildung 22) vordefiniertes Format des Zielbildes wählen (z.B. für FOX: 640x480 – VGA) oder Breite (W) und Höhe (H) in Pixel eingeben

   > Mit der Maus unten ins Bild klicken, damit der rote Bildrahmen und rechts der SnapShot-Ausschnitt erscheint (die Grösse des roten Bildrahmens und des SnapShots entspricht der zuvor gewählten Breite und Höhe des Zielbildes)

   > in den Feldern „Selektiert“ der *Groupbox 3* die Koordinaten X=0 und Y=0 für die linke obere Ecke des roten Rahmens eingeben

   > die Bildgrösse kann in der *Groupbox 2* mit dem Prozentfeld geändert werden, entweder durch direkte Prozent-Eingabe oder entsprechend der Schrittgrösse, die im Feld „Increment“ gewählt ist. Auf diese Weise wird das Bild in den Zielrahmen eingepasst.

   > Speichern als EGI-Format für den FOX oder als BMP bzw. JPG für andere Anwendungen (Buttons in *Groupbox 4*).

   Bild zuschneiden: Beim Zuschneiden gehen wir ähnlich vor wie beim Ändern der Bildgrösse. Wir wählen die Zielgrösse (WxH) in Pixel, ver-

kleinern das Bild nach Wunsch und schieben den roten Rahmen zum gewünschten Ausschnitt, entweder mit der Maus durch konkrete Angabe von X und Y für die linke obere Ecke. Den SnapShot speichern wir als EGI-Format für den FOX oder als BMP bzw. JPG für andere Anwendungen (Buttons in *Groupbox 4*).

5. Bild oder Ausschnitt verlustfrei speichern als EGI-Format für den FOX oder als BMP bzw. JPG für andere Anwendungen (Buttons in *Groupbox 4*).

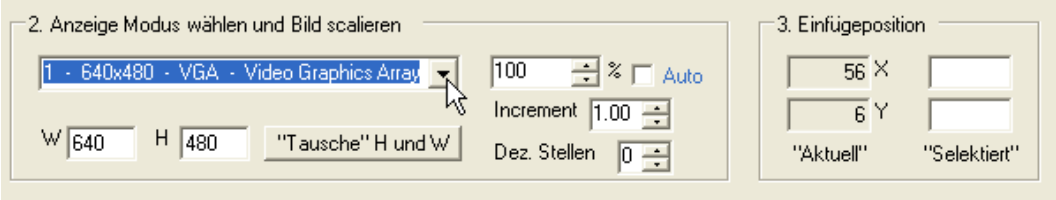


Abbildung 22:
In Groupbox 2 und 3 in der Applikation Snapshot von eiger Graphic Suite werden die Parameter des Bildes bzw. Bildausschnitts gewählt.

Wird ein Bild für den FOX im Format *.EGI gespeichert, so muss dessen Dateiname dem DOS-Format entsprechen: höchstens 8 Zeichen plus 3 Zeichen für die Erweiterung (XXXXXXXX.EGI). Spezialzeichen, wie / , . ; - ? ! oder Umlaute sind auch nicht erlaubt.

Das Bild muss im FOX nicht unbedingt bildschirmfüllend sein. Kleinere Bilder können wir mit der Methode `Load.Pos_X1Y1(0,0)` irgendwo auf dem Bildschirm positionieren.

Vordimensioniertes Bild ins EGI-Format konvertieren

Wenn wir aus einer andern Anwendung bereits über ein Bild in der gewünschten Darstellungsgrösse verfügen, können wir es mit Hilfe der eiger Graphic Suite ins FOX-eigene Bildformat *.EGI umwandeln (vgl. Abbildung 23).

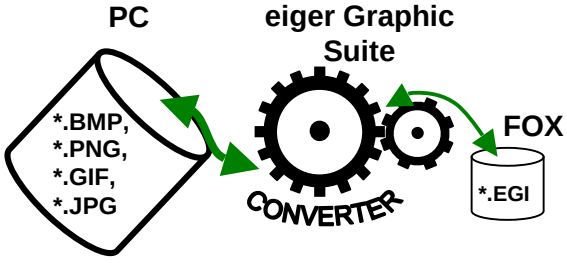


Abbildung 23:
Mit der eiger Graphic Suite werden gängige Windows Bildformate ins FOX-Bildformat *.EGI umgewandelt und umgekehrt.

Mit nur fünf Schritten kommen wir ans Ziel:

1. eigerGraphic Suite aufrufen.
2. Applikation für die Konvertierung wählen: *Applikation* **>** *EGI*. In der Applikation EGI (Abbildung 24) werden die Windows Bildformate ins EGI-Format gewandelt und umgekehrt.
3. Bild laden: *Bild öffnen* (Button in der Groupbox 1; vgl. Abbildung 24) **>** im Dateibrowser gewünschtes Bild suchen **>** *Öffnen*. Die Datei wird geladen und unterhalb der Bedienelemente angezeigt. Auf der rechten Seite sind Dateigrösse, Breite und Höhe des Bildes (Anzahl Pixel) angegeben. In der Applikation können auch 5 Beispielbilder abgerufen werden.
4. Farbreduktion wählen: *5 Bit* (Button in der zweiten Groupbox, vgl. Abbildung 24). Die Reduktion „4 Bit“ und „3 Bit“ kann ausprobiert werden. Die Wirkung auf das Bild wird unten angezeigt, die Auswirkung auf Bildgrösse und Kompression rechts in der Groupbox 2.
5. Bild als EGI speichern: *Bild als EGI* (Button in der dritten Groupbox, vgl. Abbildung 24). Der Browser für das Speichern öffnet sich, und das File kann mit entsprechendem Namen abgelegt werden.

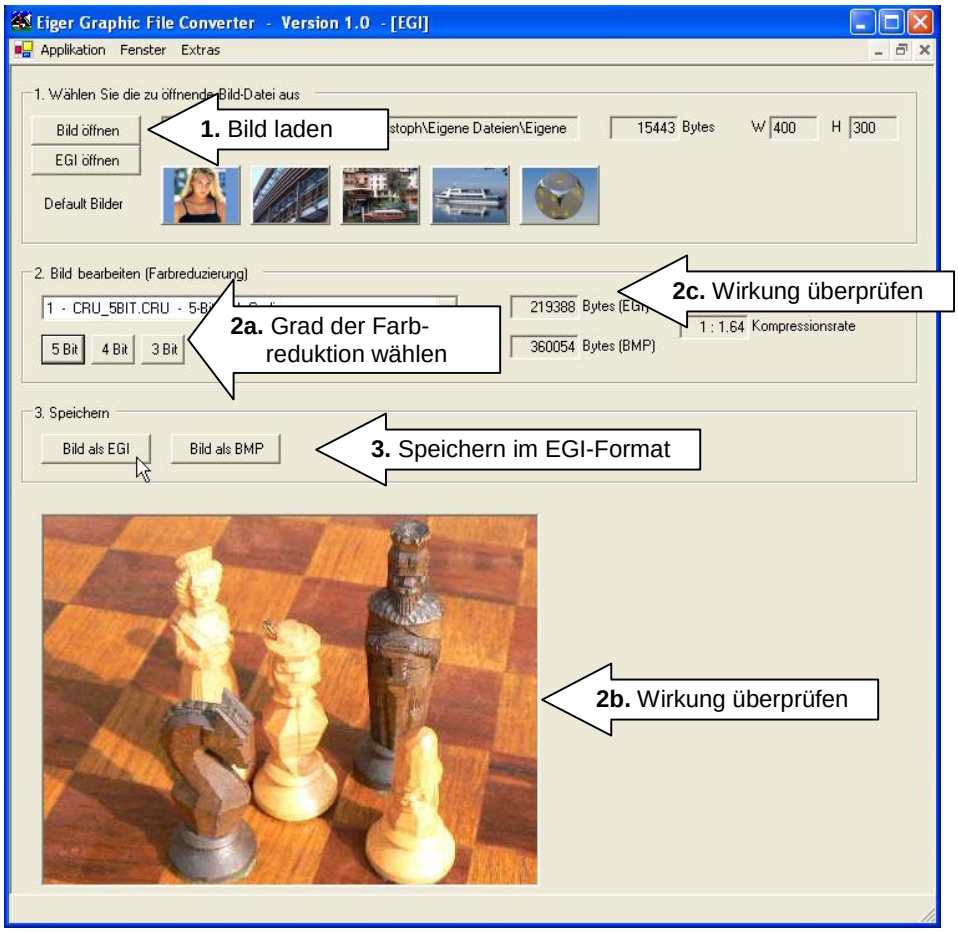


Abbildung 24: Konvertierungs-applikation vom EigerGraphic File Converter und die Schritte für die Konvertierung ins EGI-Format.

eiger-Diaschau

Als Exkurs sei hier noch auf den automatischen Diaschau-Modus von FOX hingewiesen. Beschicken wir die Compact Flash Card nach einer bestimmten Ordnung (vgl. Abbildung 25) mit mehreren Bildern im EGI-Format, dann beginnt nach Einschieben der Compact Flash Card automatisch eine Endlos-Diaschau, sofern keine Datei START.FOX mit gültigem Projekt vorhanden ist (allenfalls vorhandene START.FOX-Datei löschen bzw. umbenennen).

Damit die EGI-Bilder als Diaschau angezeigt werden, müssen wir diese in den Ordner PICT speichern (vgl. Abbildung 25). Zudem müssen die Bilder der Diaschau nach dem Muster BILD_001.EGI mit fortlaufender Numerierung benannt sein. Die Dateinamen bestimmen die Anzeige-Reihenfolge.

Die Taktgeschwindigkeit für den Bildwechsel können wir mit Hilfe des Potentiometers am eigerPanel variieren (vgl. Abbildung 1).

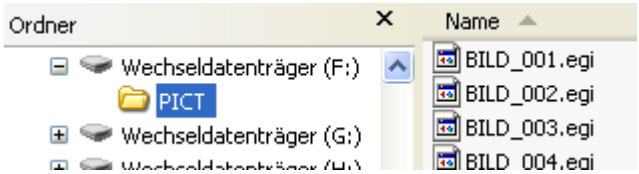


Abbildung 25: Ordner- und Dateistruktur auf Compact Flash Card für automatische FOX-Diaschau.

Auf der CD zum eigerScript-Schnelleinstieg liegt im Ordner *Schnelleinstieg/Diaschau/PICT* eine Beispieldiaschau mit 10 EGI-Bildern bereit. Für eine Demonstration kopieren wir das PICT-Ordner direkt auf die Compact Flash Card (Abbildung 25). Falls auf der Compact Flash Card ein START.FOX vorhanden ist, müssen wir diese Datei löschen oder temporär umbenennen.
Sie können auch komplexere Diashows mit bildspezifischen Übergängen und Anzeigezeiten konfigurieren. Beachten Sie hierzu auch die Application Note „Slideshow auf dem eigerPanel“.

Übung 5: Titelleiste konfigurieren

In dieser Übung lernen Sie

- ✓ Text als Label in eine View integrieren.
- ✓ Textlabels grafisch gestalten (beschriftete Rechtecke).

Zeitbedarf: 15 Minuten

Daten für die Übung: Projektordner „Schachbrett“ mit der Datei CHSS_001.EVS aus der Übung 4, den Sie selbst erstellt haben oder vorgefertigt im Ordner „Schnelleinstieg/Übung04“ vorfinden.

Die Startview soll am oberen Rand des FOX-Fensters eine schwarze Titelleiste enthalten mit dem Projektnamen, dem Namen der View-Datei und dem Erstelldatum (Abbildung 26).

Abbildung 26: Titelleiste der Startseite.

Definition von Text

Die Beschriftung auf der Titelleiste besteht aus zwei eigenständigen Wortketten: einem Titel oben links in hellgelber (lightyellow) Schrift auf schwarzem Grund und einem Titel oben rechts in lachsfarbener (lightsalmon) Schrift.

eigerStudio „kennt“ eine grosse Palette Farbkonstanten, wie z.B. lightyellow oder lightsalmon, die in der Header-Datei DEF_eiger_Colors.INC definiert sind. Nach dem Vorbild dieser Datei ist es auch möglich, eine weitere Farben-Datei mit eigenen Farbton-Definitionen zu erstellen und diese mit `INCLUDEFILE` in die View zu integrieren oder eigene Farben als Konstanten zu deklarieren.

Das eigerScript-Schlüsselwort für die Definition von Wortketten ist `STRING`.

Ein `STRING` wird in Form einer Gleichung definiert (Beispiel-Code 11). Auf der linken Seite der Gleichung stehen das Schlüsselwort, die maximale Anzahl Zeichen des Strings [in Klammern] und der Name des Strings (z.B. `Titel.$` oder `Titel_Right.$`). Die Erweiterung „**.$**“ ist nicht unbedingt notwendig, aber nützlich als Gedankenstütze. Damit ist es beim Lesen des Script auf einen Blick klar, dass es sich bei einem Namen mit „**.$**“ um eine String-Variable handelt. Analog brauchen wir für Integer-Variablen die Erweiterung „.I“ oder für Single „.S“.

Auf der linken Seite der Gleichung geben wir den Titeltext in Hochkommas an (Beispiel-Code 11).

Diese Deklarationen fügen wir im Script der Startview ein, oberhalb des Schlüsselworts `BEGINVIEW` (Beispiel-Code 11). Ein erklärender Kommentar (mit Semikolon) ist hilfreich beim späteren Nachvollziehen des Programm-Scripts.

Beispiel-Code 11 (Text-Definitionen)

```
; Definitionen für Startview -------------------------------------

STRING [60] Titel.$ = 'Projekt Schachbrett, CHSS_001.EVS: Startview'
STRING [60] Titel_Right.$ = 'erstellt: 16.03.2007' ; Titelleiste rechts

BEGINVIEW
```

Wir können einen Namen für eine Variable oder eine Konstante nur dann als Alias verwenden, wenn dieser im Voraus definiert worden ist. Darum sollten Variablen- und Konstanten-Definitionen in der Regel am Anfang eines View-Scripts stehen.

Die Gestaltung einer Titelleiste kann in einem eigenen Projekt selbstverständlich angepasst (oder weggelassen) werden. Es hat sich aber bewährt, eine Statuszeile in die View zu integrieren. So können View (z.B. CHSS_001.EVI) und Source (z.B. CHSS_001.EVS) leichter zusammengehalten werden.

Titelliste formatieren und schreiben

Wir formatieren und schreiben die Titel mittels einer Subroutine, die wir `Draw_Titel` nennen (Beispiel-Code 12). Subroutinen sind definiert mit den

obligatorischen Schlüsselwörtern `SUB` am Anfang und `ENDSUB` am Ende der Subroutinen. Näheres zum Thema Subroutinen (Prozeduren) haben wir bereits in der Übung 2 erfahren .

Mit der Methode `Fill.LabelParameter(labelRelative24)` wird das Titelformat festgelegt. Zwischen die Klammern gehört der Name des Label-Styles, dessen Eigenschaften wir weiter unten spezifizieren werden.

Die eigerScript-Funktion `Fill.LabelParameter(labelRelative24)` verweist auf eine Struktur, in welcher alle Eigenschaften des Label-Objekts (z.B. des Objekts „Titel_Style“) beschrieben sind. `LabelRelative24` steht für eine 24 Bit Pointer-Adresse.

Die Funktion `Label.Text(VarStr)` schreibt jeweils den `STRING` in die View, den wir in Beispiel-Code 11 bereits als „Titel.$“ bzw. „Titel_Right.$“ definiert haben.

Beispiel-Code 12 fügen wir nach den String-Deklarationen und direkt über dem Hauptprogramm ein.

Beispiel-Code 12

```
SUB Draw_Titel
    ; Titelleiste schreiben ---------------------------------------

    Fill.LabelParameter(Titel_Style)
    Label.Text(Titel.$)

    Fill.LabelParameter(Titel_Right_Style)
    Label.Text(Titel_Right.$)

ENDSUB
```

Ohne Aufruf `CallSubroutine` bleibt eine Subroutine verweist.

Damit der Titel dann auch geschrieben wird, braucht es eine Aufrufzeile `CallSubroutine` im Hauptprogramm. Diese positionieren wir im Anschluss an die bisherigen Befehlszeilen (Beispiel-Code 13).

Beispiel-Code 13 (Hauptprogramm ergänzt mit Aufruf zum Zeichnen der Titelleiste)

```
BEGINVIEW
    EVE.Init()                      ; EVE ANNA initialisieren
    Load.Pos_X1Y1(0,0)              ; linke obere Ecke
    File.Read_EGI('C:\\CHSS\\PICT\\CHESSSET.EGI')
    CallSubroutine(Draw_Titel)

    LOOP
    ENDLOOP
```

```
ENDVIEW
```

Mit der Subroutine „Styles“ (Beispiel-Code 14) legen wir nun die Eigenschaften der beiden Titel fest, beispielsweise die Anfangskoordinaten (X,Y) des Titels auf dem FOX-Monitor oder die Hintergrund- und Schriftfarbe. Die Eigenschaftswerte werden mit Zahlenwerten (z.B. Anzahl Pixel für X und Y) oder definierten Konstanten (z.B. `blue181`, vgl. Header-Datei DEF_eiger_Colors.INC) angegeben.

Den Beispiel-Code 14 fügen wir direkt oberhalb des Hauptprogramms ein.

Beispiel-Code 14

```
SUB Styles

Titel_Style:
    INLINEWORDS  (0)          ; entspricht eI.Pos_X1
    INLINEWORDS  (0)          ; entspricht eI.Pos_Y1
    INLINEWORDS  (640)        ; entspricht eI.Width
    INLINEWORDS  (18)         ; entspricht eI.Height
    INLINEWORDS  (20)         ; entspricht eI.SpaceLeft
    INLINEWORDS  (8)          ; entspricht eI.SpaceRight
    INLINEWORDS  (0)          ; entspricht eI.HorizontalAdjust
    INLINEWORDS  (0)          ; entspricht eI.VerticalAdjust
    INLINEWORDS  (black)      ; entspricht eI.FillColor
    INLINEWORDS  (black)      ; entspricht eI.BackColor
    INLINEWORDS  (black)      ; entspricht eI.LineColor
    INLINEWORDS  (lightyellow)       ; entspricht eI.TextColor
    INLINEWORDS  (Pos_left)          ; entspricht eI.Position
    INLINEWORDS  (Orientation_0deg)  ; entspricht eI.Orientation
    INLINEWORDS  (normal)            ; entspricht eI.Appearance
    INLINEWORDS  (no_border)         ; entspricht eI.BorderStyle
    INLINEWORDS  (Font_System_9bd)   ; entspricht eI.FontNumber
    INLINEWORDS  (silver) ; entspr.eI.BackgroundColor

Titel_Right_Style:
    INLINEWORDS  (0)          ; entspricht eI.Pos_X1
    INLINEWORDS  (0)          ; entspricht eI.Pos_Y1
    INLINEWORDS  (640)        ; entspricht eI.Width
    INLINEWORDS  (18)         ; entspricht eI.Height
    INLINEWORDS  (20)         ; entspricht eI.SpaceLeft
    INLINEWORDS  (8)          ; entspricht eI.SpaceRight
    INLINEWORDS  (0)          ; entspricht eI.HorizontalAdjust
    INLINEWORDS  (0)          ; entspricht eI.VerticalAdjust
    INLINEWORDS  (transparent)       ; entspricht eI.FillColor
    INLINEWORDS  (black)             ; entspricht eI.BackColor
    INLINEWORDS  (black)             ; entspricht eI.LineColor
    INLINEWORDS  (lightsalmon)       ; entspricht eI.TextColor
    INLINEWORDS  (Pos_right)         ; entspricht eI.Position
    INLINEWORDS  (Orientation_0deg)  ; entspricht eI.Orientation
    INLINEWORDS  (normal)            ; entspricht eI.Appearance
    INLINEWORDS  (no_border)         ; entspricht eI.BorderStyle
    INLINEWORDS  (Font_System_9bd)   ; entspricht eI.FontNumber
    INLINEWORDS  (silver) ; entspr.eI.BackgroundColor

ENDSUB
```

Der eigerCompiler erkennt die einzelnen Eigenschaften aufgrund ihrer Reihenfolge. Deshalb darf die Reihenfolge und die Anzahl der Eigenschaften auf keinen Fall verändert werden. Der ganze Style ist darum als ein Körper (*Objekt*) zu betrachten. Die Reihenfolge der einzelnen *Register* darf nicht verändert werden. Hingegen können die Eigenschaften dieser Register sehr vielfältig sein. Dieses Objekt ist in der Programmiernomenklatur eine *Klasse*: Label Class.

Für den eigerCompiler und die virtuelle Maschine ist die Anordnung der einzelnen Scriptblöcke im eigerView-Script unwesentlich. Es wird jedoch empfohlen, sich an eine stets gleiche Ordnung zu halten, z.B:

1. Deklarationen
2. Subroutinen
3. Hauptprogramm

U005S001.EGI

Zum Abschluss dieser Übung kompilieren wir das Script der Startview und übertragen das aktualisierte CHSS_001.EGI via Compact Flash Card auf den FOX. Nach einem Warmstart mittels Reset sollte nun auf dem FOX die neue Startview CHSS_001.EVS inklusive Titelleiste erscheinen (Abbildung 27).

Abbildung 27:
Schachfiguren-Bild mit Titelleiste als Startseite View 1 (CHSS_001.EVS)

Übung 6: Kontrollierter View-Aufbau

In dieser Übung lernen Sie

- ✓ das Prinzip des Videospeichers von FOX kennen,
- ✓ die Vorteile der doppelten Videospeicherung nutzen (Doublebuffering).

Zeitbedarf: 5 Minuten

Daten für die Übung: Projektordner „Schachbrett“ mit der Datei CHSS_001.EVS aus der Übung 5, den Sie selbst erstellt haben oder vorgefertigt im Ordner „Schnelleinstieg/Übung05“ vorfinden.

Videospeicher AVR und RVR

Nach einem Reset können wir beobachten, dass der FOX die Startview in der Reihenfolge aufbaut, wie die einzelnen Befehle im Hauptprogramm nacheinander folgen. Zuerst rollt sich das Hintergrundbild ab, als zweites erscheint die Titelleiste und zuletzt wird der Button eingesetzt. Das geht im Normalfall so schnell, dass es kaum auffällt. Bei sehr rechenintensiven Views könnte dieser Prozess des View-Aufbaus aber so viel Zeit beanspruchen, dass diese Aufbauphase störend wirkt.

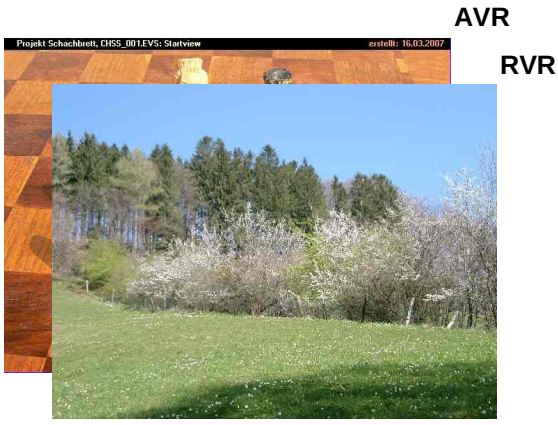


Abbildung 28: EVE kann Bilder auf zwei Ebenen laden. Das Bild im „vorderen“ Videospeicher **RVR** wird auf dem Bildschirm angezeigt. Was ins **AVR** geladen wird, ist verborgen.

In unserem Beispiel können wir den sequentiellen Bildaufbau beobachten, weil die Information, die EVE in den Videospeicher lädt, gleichzeitig auch auf dem Bildschirm angezeigt wird. Dieser „sichtbare“ Videospeicher ist das **RVR** (Refresh Video RAM). EVE verfügt aber zusätzlich noch über einen zweiten Videospeicher, der nicht direkt mit dem Bildschirm verknüpft ist: das **AVR** (Accessible Video RAM). Wenn wir ein Bild nur ins AVR laden, bleibt es verborgen, bis es per Befehl – z.B. mit der Methode `Display.Show` – ins RVR kopiert wird (vgl. Abbildung 29). So kann EVE eine View im Hintergrund im AVR aufbauen und dann als ganzes ins RVR kopieren und auf einmal am Bildschirm anzeigen.

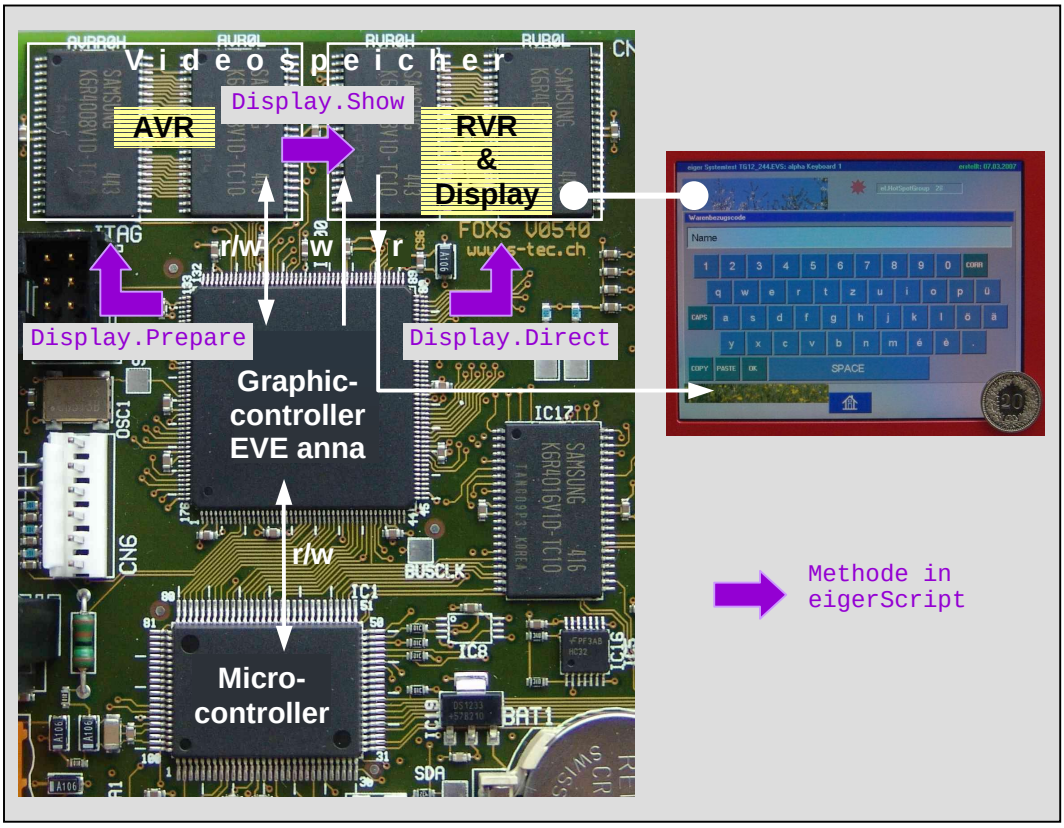


Abbildung 29: Speicherorte und Steuerung der Bildinformation auf der virtuellen Maschine des FOXS. Der Microcontroller (Prozessor) hat via EVE Lese- und Schreibzugriff (r/w) auf den Arbeitsspeicher. Die Information im RVR wird im Display dargestellt.

Bildaufbau im AVR – Anzeige im RVR

Ohne spezielle Anweisung wird die View bzw. die Graphik in beide Videospeicher geladen.

Bisher haben wir in unserem Script bezüglich RVR und AVR keine speziellen Anweisungen gegeben. Unsere Startview wurde deshalb von EVE gleichzeitig in beide Videospeicher geladen.

Mit der Methode `Display.Prepare()` laden wir nun die Startview zuerst vorbereitend ausschliesslich ins AVR. Die entsprechende Code-Zeile positionieren wir im Hauptprogramm direkt nach der Initialisierung von EVE (vgl. Beispiel-Code 15). Zudem fügen wir am Schluss des Hauptprogramms `Display.Show()` ein. Mit dieser Methode kopiert EVE den ganzen Inhalt des AVR ins RVR, was ein augenblickliches Erscheinen der Startview auf dem Bildschirm bedeutet.

Beispiel-Code 15 (`Display.Prepare` am Anfang und `Display.Show` des Bildaufbaus im Hauptprogramms)

```
BEGINVIEW
    EVE.Init()                      ; EVE ANNA initialisieren
    Display.Prepare()
    Load.Pos_X1Y1(0,0)              ; linke obere Ecke
    File.Read_EGI('C:\\CHSS\\PICT\\CHESSSET.EGI')
    CallSubroutine(Draw_Titel)
    Display.Show()

    LOOP
    ENDLOOP

ENDVIEW
```

Zum Test dieser Verfeinerung übertragen wir das kompilierte Script CHSS_001.EVI auf den FOX. Nach Reset stellen wir fest, dass die Startview nun ohne Aufbauphase gleich als Ganzes angezeigt wird.

`Display.Show()` kopiert den ganzen Inhalt vom AVR ins RVR, `Display.ShowWindow()` nur einen bestimmten Ausschnitt.

Nebst diesen beiden Methoden, die wir nun in dieser Übung kennengelernt haben, bietet eigerScript noch eine weitere sehr nützliche Funktion. Mit `Display.ShowWindow()` ist es auch möglich – zusammen mit den entsprechenden Geometrie-Angaben –, nur einen bestimmten Ausschnitt aus dem AVR ins RVR zu kopieren. Der Ausschnitt bezieht sich auf den gegenwärtigen Inhalt der Register `eI.Pos_X1`, `eI.Pos_Y1`, `eI.Width` und `eI.Height`. Diese Methode werden wir beim Erstellen der zweiten View nutzen.

Übung 7: Buttons, HotSpots und HotKeys

In dieser Übung lernen Sie

- ✓ Buttons auf dem Touchpanel plazieren und darstellen.
- ✓ Berührungsempfindliche HotSpots auf dem Touchpanel einrichten.
- ✓ animierte Buttons erstellen.

Zeitbedarf: 30 Minuten

Daten für die Übung: Projektordner „Schachbrett“ mit der Datei CHSS_001.EVS aus der Übung 6, den Sie selbst erstellt haben oder vorgefertigt im Ordner „Schnelleinstieg/Übung06“ vorfinden.

String-Deklarationen für Buttons

Ein Foto ohne Angabe des Autors ist nur halb so viel wert. Deshalb wollen wir in der Startview am unteren Bildrand einen Button „Fotograf“ einfügen, der einen Schriftzug mit dem Namen des Fotografen aufleuchten lässt, wenn wir auf den Button drücken. Dazu definieren wir zwei weitere STRINGS mit den Bezeichnungen „Label_FotoButton.$“ und „Label_Name.$“. Für beide STRINGs genügt der Platz von maximal 15 Zeichen. Den Code fügen wir ordnungsgemäss unterhalb der beiden bereits bestehenden STRING-Definitionen ein (Beispiel-Code 16).

Beispiel-Code 16 (die ersten beiden STRINGS sind bereits aus Übung 5 vorhanden)

```
STRING [60] Titel.$ = 'Projekt Schachbrett, CHSS_001.EVS: Startview'
STRING [60] Titel_Right.$ = 'erstellt: 16.03.2007' ; Titelleiste rechts
STRING [15] Label_FotoButton.$ = 'Fotograf:'
STRING [15] Label_Name.$ = 'Christoph Angst'
```

Strings sind Variablen, deren Wert aus einer Zeichenkette besteht. Die Zeichenkette muss zwischen ’Hochkommas’ stehen. Mehr zu diesem Thema finden Sie im Anhang auf S.116-118 ▶.

Button ‚Fotograf'

Zunächst konzentrieren wir uns auf den Button "Fotograf". Die Eigenschaften des Buttons legen wir als `Button_Style` (Beispiel-Code 17) innerhalb der bereits bestehenden Subroutine `Styles` fest.

Die Position des Buttons ist auf der Startview rechts unten: X=400; Y=420 Pixel. Damit sind die Bildkoordinaten für die linken obere Ecke des Buttons bestimmt.

Die Grösse des Buttons soll 100x50 Pixel betragen (Width x Height).

Als Hintergrundfarbe des Labels wählen wir „red". Diese Farbbezeichnung ist eine Konstante, die in der Datei „DEF_eiger_Colors.INC" definiert ist. EigerScript erkennt diese Konstante, weil wir diese Datei eingangs in unser Projekt importiert haben (vgl. Beispiel-Code 3).

Beispiel-Code 17

```
Button_Style:
    INLINEWORDS  (400)          ; entspricht eI.Pos_X1
    INLINEWORDS  (420)          ; entspricht eI.Pos_Y1
    INLINEWORDS  (100)          ; entspricht eI.Width
    INLINEWORDS  (50)           ; entspricht eI.Height
    INLINEWORDS  (8)            ; entspricht eI.SpaceLeft
    INLINEWORDS  (8)            ; entspricht eI.SpaceRight
    INLINEWORDS  (0)            ; entspricht eI.HorizontalAdjust
    INLINEWORDS  (0)            ; entspricht eI.VerticalAdjust
    INLINEWORDS  (red)          ; entspricht eI.FillColor
    INLINEWORDS  (red)          ; entspricht eI.BackColor
    INLINEWORDS  (red)          ; entspricht eI.LineColor
    INLINEWORDS  (white)              ; entspricht eI.TextColor
    INLINEWORDS  (Pos_center)         ; entspricht eI.Position
    INLINEWORDS  (Orientation_0deg)   ; entspricht eI.Orientation
    INLINEWORDS  (normal)             ; entspricht eI.Appearance
    INLINEWORDS  (color_button_3D_raised)   ; entspr. eI.BorderStyle
    INLINEWORDS  (Font_System_9bd)   ; entsprichteI.FontNumber
    INLINEWORDS  (silver) ; entspr.eI.BackgroundColor
```

Mit einer neuen Subroutine namens `Draw_Button_Fotograf` (Beispiel-Code 18) führen wir den Button-Style (Verweis auf Beispiel-Code 17) und die Aufschrift des Buttons (Verweis auf `STRING`-Deklaration, vgl. Beispiel-Code 16) zusammen.

Beispiel-Code 18 fügen wir unterhalb der Subroutine `Draw_Titel` ein.

Beispiel-Code 18

```
SUB Draw_Button_Fotograf
    Fill.LabelParameter(Button_Style)
    Label.Text(Label_FotoButton.$)
ENDSUB
```

Effektiv zeichnet FOX den Button erst, wenn im Hauptprogramm mit `CallSub-routine(Draw_Button_Fotograf)` der entsprechende Subroutinen-Aufruf eingetragen ist (vgl. Beispiel-Code 19), wobei die Subroutine mit dem Namen `Draw_Button_Fotograf` ausgeführt wird.

Beispiel-Code 19 (Hauptprogramm ergänzt mit Aufruf zum Zeichnen des Button „Fotograf“)

```
BEGINVIEW
    EVE.Init()                      ; EVE ANNA initialisieren
    Display.Prepare()
    Load.Pos_X1Y1(0,0)              ; linke obere Ecke
    File.Read_EGI('C:\\CHSS\\PICT\\CHESSSET.EGI')
    CallSubroutine(Draw_Titel)
    CallSubroutine(Draw_Button_Fotograf)
    Display.Show()

    LOOP
    ENDLOOP
ENDVIEW
```


U006S001.EGI

Nun ist es wieder an der Zeit, die Auswirkungen unserer Ergänzungen am FOX zu testen. Wir kompilieren das aktuelle File „CHSS_EVS“ und übertragen das überschriebene „CHSS_EVI“ via Compact Flash Card auf den FOX. Nach einem Reset des FOX erscheint die Startview gemäss Abbildung 30.

Abbildung 30:
Startview CHSS_001.EVI mit Button „Fotograf“

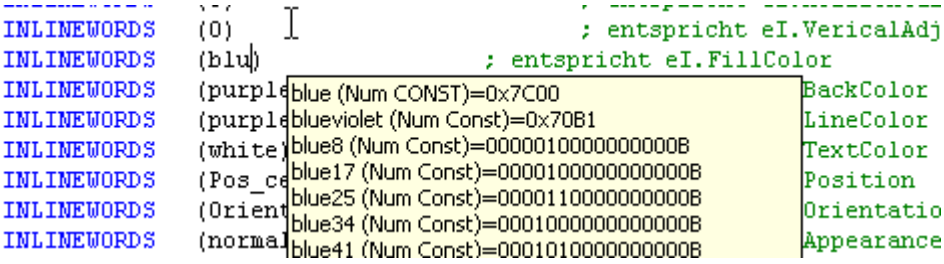
Zum besseren Verständnis der Bedeutung der verschiedenen Farb-Register in Beispiel-Code 17 wählen wir probehalber die Farbkonstante „blue181“ statt „red“ als Füllfarbe (`eI.FillColor`) für den Button. Mit dieser Änderung stellt FOX den Button so dar, wie in Abbildung 31 dargestellt.

Abbildung 31: Button mit geänderter Füllfarbe im Vergleich zu Beispiel-Code 17.


```
eI.FillColor := blue181
eI.BackColor := red
eI.LineColor := red
eI.TextColor := white
```

Während des Schreibens der Farbkonstante blendet eigerStudio unterhalb des Mauszeigers eine Popup-Liste (vgl. Abbildung 32) aller Farbkonstanten ein, die mit den bereits eingetippten Buchstaben beginnen und in der Datei „DEF_eiger_Colors.INC“ vordefiniert sind. Dieser Service ist eine nützliche Hilfe zur Farbwahl und zur Kontrolle der richtigen Schreibweise.

Abbildung 32: Popup-Liste (gelb) während des Schreibens einer Konstante im eigerEditor.



Label ‚Name'

Der Name des Fotografen soll ohne Hintergrundfarbe in weisser Schrift auf dem Startview-Bild erscheinen. Für die Darstellung dieses Label definieren wir wiederum in der Subroutine `Styles` einen speziellen Style mit der Bezeichnung `Name_Style` (Beispiel-Code 20).

Die Position des Labels „Name" ist um die Breite (100 Pixel) des Buttons „Fotograf" plus 10 Pixel nach rechts versetzt: X = 400 + 100 + 10. Die Y-Koordinate ist identisch mit derjenigen des Buttons.

Für die Register `eI.FillColor`, `eI.BackColor`, `eI.LineColor` wählen wir die Konstante „transparent".

Beispiel-Code 20

```
Name_Style:
    INLINEWORDS  (510)          ; entspricht eI.Pos_X1
    INLINEWORDS  (420)          ; entspricht eI.Pos_Y1
    INLINEWORDS  (100)          ; entspricht eI.Width
    INLINEWORDS  (50)           ; entspricht eI.Height
    INLINEWORDS  (8)            ; entspricht eI.SpaceLeft
    INLINEWORDS  (8)            ; entspricht eI.SpaceRight
    INLINEWORDS  (0)            ; entspricht eI.HorizontalAdjust
    INLINEWORDS  (0)            ; entspricht eI.VerticalAdjust
    INLINEWORDS  (transparent)        ; entspricht eI.FillColor
    INLINEWORDS  (transparent)        ; entspricht eI.BackColor
    INLINEWORDS  (transparent)        ; entspricht eI.LineColor
    INLINEWORDS  (white)              ; entspricht eI.TextColor
    INLINEWORDS  (Pos_center)         ; entspricht eI.Position
    INLINEWORDS  (Orientation_0deg)   ; entspricht eI.Orientation
    INLINEWORDS  (normal)             ; entspricht eI.Appearance
    INLINEWORDS  (no_border)          ; entspr. eI.BorderStyle
    INLINEWORDS  (Font_System_9bd)    ; entspr. eI.FontNumber
    INLINEWORDS  (silver) ; entspr.eI.BackgroundColor
```

Die neue Subroutine `Draw_Label_Name` soll dann den Namen des Fotografen mit der Darstellung von Beispiel-Code 20 in die View schreiben (Beispiel-Code 21).

Beispiel-Code 21

```
SUB Draw_Label_Name
    Fill.LabelParameter(Name_Style)
    Label.Text(Label_Name.$)
ENDSUB
```

Der Aufruf zum Schreiben des Namens erfolgt wiederum vom Hauptprogramm aus. Diesen setzen wir gemäss Beispiel-Code 22 anschliessend an die bisherigen Befehlszeilen.

Beispiel-Code 22 (Hauptprogramm ergänzt mit neuem Befehl `CallSubroutine(Draw_Label_Name)`)

```
BEGINVIEW
    EVE.Init()                          ; EVE ANNA initialisieren
    Display.Prepare()
    Load.Pos_X1Y1(0,0)                  ; linke obere Ecke
    File.Read_EGI('C:\\CHSS\\PICT\\CHESSSET.EGI')
    CallSubroutine(Draw_Titel)
    CallSubroutine(Draw_Button_Fotograf)
    CallSubroutine(Draw_Label_Name)
    Display.Show()

    LOOP
    ENDLOOP
ENDVIEW
```

Das Ergebnis eines erneuten Programmtests auf FOX ist in Abbildung 33 dargestellt.

Abbildung 33: Startview CHSS_001.EVI mit Button „Fotograf“ und Label „Name“.

HotSpot ‚Fotograf’

In einem weiteren Schritt hinterlegen wir dem Button „Fotograf“ einen *HotSpot* der auf Berührung das zuvor verborgene Label „Name“ erscheinen lässt.

Der unsichtbare HotSpot soll sich mit dem Button „Fotograf“ decken (vgl. Beispiel-Code 18 und Beispiel-Code 17). Das erreichen wir mit dem Befehl:

`Load.Geometry_XYWH(VarInt:X1,VarInt:Y1,VarInt:W,VarInt:H).`

Diese Befehlszeile wird Teil des Hauptprogramms (gemäss Beispiel-Code 23), wobei die Positions- (X1/Y1) und Grössen-Variablen (Width und Height) identisch sind mit den Abmessungen des Buttons „Fotograf“.

Zusätzlich braucht es noch eine Methode, mit welcher der HotSpot auf dem Fenster installiert wird:

`HotSpot.Install(labelRelative24:Enter,labelRelative24:Leave,labelRelative24:Down,labelRelative24:Up).`

Ein HotSpot kann zwischen vier Ereignissen unterscheiden:

Enter:	HotSpot wird erreicht (schieben des Fingers von aussen in den HotSpot)
Leave:	HotSpot wird verlassen (schieben des Fingers vom HotSpot nach aussen)
Down:	Berührung beginnt im HotSpot
Up:	Berührung endet im HotSpot

Der HotSpot soll nur beim Ereignis „Down“ reagieren. Deshalb setzen wir die drei anderen Ereignisse mit `NIL` ausser Kraft. `NIL` gibt an, dass diese Ereignisroutinen nicht ausprogrammiert sind. An der Stelle des Down-Ereignisses (dritte Variable) muss ein Verweis auf Subroutine `Draw_Label_Name` stehen, die das zuvor „verborgene“ Label „Name“ einblendet (Beispiel-Code 23).

Beispiel-Code 23 (Hauptprogramm mit den neuen Befehlszeilen für den HotSpot Fotograf)

```
BEGINVIEW
    EVE.Init()                      ; EVE ANNA initialisieren
    Display.Prepare()
    Load.Pos_X1Y1(0,0)              ; linke obere Ecke
    File.Read_EGI('C:\\CHSS\\PICT\\CHESSSET.EGI')
    CallSubroutine(Draw_Titel)
    CallSubroutine(Draw_Button_Fotograf)
    CallSubroutine(Draw_Label_Name)

; HotSpot zum Einblenden des Label_Name -------------------------
    Load.Geometry_XYWH(400,420,100,50)
    HotSpot.Install(NIL,NIL,Draw_Label_Name,NIL)

    Display.Show()

    LOOP
    ENDLOOP
ENDVIEW
```

Damit das Label „Name“ zu Beginn nicht sichtbar ist, setzen wir die bisherige Programmzeile `CallSubroutine(Draw_Label_Name)` ausser Kraft, entweder durch Löschen oder durch „Auskommentieren“ mit Hilfe eines Semikolons (der Code färbt sich dadurch grün, vgl. Beispiel-Code 24).

Beispiel-Code 24 (gleich wie Beispiel-Code 23, aber mit auskommentierter CallSubroutine)

```
BEGINVIEW
    EVE.Init()                      ; EVE ANNA initialisieren
    Display.Prepare()
    Load.Pos_X1Y1(0,0)              ; linke obere Ecke
    File.Read_EGI('C:\\CHSS\\PICT\\CHESSSET.EGI')
    CallSubroutine(Draw_Titel)
    CallSubroutine(Draw_Button_Fotograf)
;   CallSubroutine(Draw_Label_Name)

; HotSpot zum Einblenden des Label_Name -------------------------

    Load.Geometry_XYWH(400,420,100,50)
    HotSpot.Install(NIL,NIL,Draw_Label_Name,NIL)

    Display.Show()

    HotSpot.TableEnable()

    LOOP
    ENDLOOP

ENDVIEW
```

Der HotSpot ist erst komplett mit den beiden Befehlen, die in Beispiel-Code 24 am Schluss des Hauptprogramms noch angefügt sind:

- Mit `HotSpot.TableEnable()` wird die ganze HotSpotTable für die Auswertung freigegeben. Erst mit diesem Befehl ist der HotSpot für Events, d.h. für einen Fingerdruck auf der Touchscreen empfänglich. Wir „sensibilisieren“ den HotSpot erst nachdem die Startview auf dem Display erscheint, d.h. nach `Display.Show()`.
- `LOOP`
  `ENDLOOP` : Diese Schleife erhält die Aufmerksamkeit der virtuellen Maschine des FOX gegenüber eintreffenden Ereignissen (Events) aufrecht. Dadurch kann der FOX beispielsweise jederzeit auf Berührungsereignisse im Bereich der installierten HotSpots reagieren.

Nun kompilieren wir das mit dem HotSpot ergänzte eiger-Programm und testen es auf dem FOX. Wir stellen fest, dass nach dem Reset der Button „Fotograf“ auf der Startseite sichtbar ist. Durch Drücken auf den Button erscheint auch der Name des Fotografen: Christoph Angst.


U006S003.EGI

Machen Sie die Übung mit eigenem Foto und Namen.

Anstelle von HotSpots können Events auch durch angeschlossene Funktionstasten ausgelöst werden. Diese Funktion werden wir am Ende im letzten Kapitel dieser Übung einsetzen.

Animierter Button (ButtonDown-Effekt)

Drücken wir auf den Button „Fotograf“, so erscheint zwar wie gewünscht der Name des Fotografen, aber am Button selbst sehen wir keine Auswirkungen. Es wäre schön, wenn der Button wie ein echter Druckknopf unserem Druck „nachgeben“ würde (vgl. Abbildung 34).

Button up Button down

Abbildung 34: Unterschiedliche Darstellung eines Buttons vor und während bzw. nach dem „Knopfdruck-Event“.

Zu diesem Zweck führen wir in unser Startview-Script einen neuen Style ein: den `Button_down_Style` (Beispiel-Code 25). Diesen Style setzen wir innerhalb der Subroutine `Styles` an geeigneter Stelle ein.

Beispiel-Code 25

```
Button_down_Style:
    INLINEWORDS  (400)         ; entspricht eI.Pos_X1
    INLINEWORDS  (420)         ; entspricht eI.Pos_Y1
    INLINEWORDS  (100)         ; entspricht eI.Width
    INLINEWORDS  (50)          ; entspricht eI.Height
    INLINEWORDS  (8)           ; entspricht eI.SpaceLeft
    INLINEWORDS  (8)           ; entspricht eI.SpaceRight
    INLINEWORDS  (1)           ; entspricht eI.HorizontalAdjust
    INLINEWORDS  (1)           ; entspricht eI.VerticalAdjust
    INLINEWORDS  (red)         ; entspricht eI.FillColor
    INLINEWORDS  (red)         ; entspricht eI.BackColor
    INLINEWORDS  (red)         ; entspricht eI.LineColor
    INLINEWORDS  (white)       ; entspricht eI.TextColor
    INLINEWORDS  (Pos_center)         ; entspricht eI.Position
    INLINEWORDS  (Orientation_0deg)   ; entspricht eI.Orientation
    INLINEWORDS  (normal)             ; entspricht eI.Appearance
    INLINEWORDS  (color_button_3D_sunk) ; entspricht eI.BorderStyle
    INLINEWORDS  (Font_System_9bd)    ; entspricht eI.FontNumber
```

```
    INLINEWORDS  (silver) ; entspr.eI.BackgroundColor
```

Um Klarheit zu schaffen, benennen wir den bisherigen `Button_Style` neu mit `Button_up_Style`. Diese Umbenennung müssen wir nicht nur beim Style sondern auch in der Subroutine `Draw_Button_Fotograf` vornehmen.

Der Button_up soll bei Berührung durch den Button_down überschrieben werden. Beide Styles müssen sich deshalb auf den gleichen Bereich beziehen: X = 400, Y = 420, W = 100, B = 50. Die Farbe ist ebenfalls rot. Der einzige farbliche Unterschied betrifft die Umrandung, d.h. die Schattierung des Buttons. Diese Schattierung bewirkt den visuellen 3D-Effekt und wird durch das Register `eI.BorderStyle` geregelt. Die für dieses Register gewählten *Globalen Konstanten* sind in der Header-Datei DEF_eiger_Types.INC definiert (vgl. Abbildung 35).

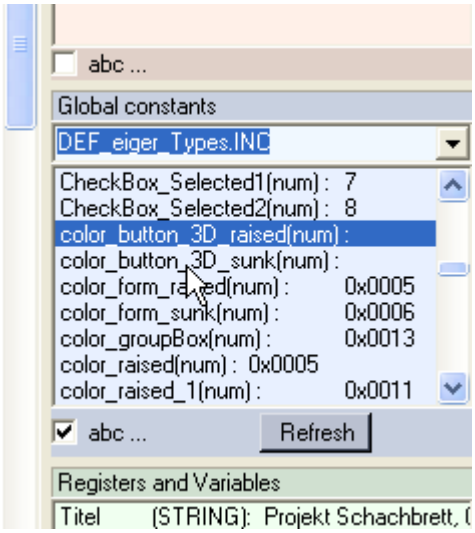


Abbildung 35: Color_button_3D_raised bzw. ..3D_sunk können wie alle globalen EigerScript-Konstanten im **Window Global constants** aufgerufen und durch Doppelklick in den Code eingesetzt werden.

Eine Spezialität des Button_down-Styles liegt noch darin, dass sich gegenüber dem Button_up nebst der Schattierung auch die Schrift leicht (um je 1 Pixel) nach rechts verschiebt. Dadurch wird bei Knopfdruck der visuelle Effekt noch verstärkt. Für diesen Effekt sind die Register `eI.HorizontalAdjust` und `eI.VerticalAdjust` verantwortlich (vgl. Beispiel-Code 25).

Wenn wir auf den HotSpot „Fotograf“ drücken, wird vom Hauptprogramm aus die Subroutine `Draw_Label_Name` aufgerufen. In dieser Subroutine müssen wir nun zwei zusätzliche Code-Zeilen einfügen (vgl. Beispiel-Code 26), damit der Button_down dann auch wirklich auf Knopfdruck den Button_up überschreibt, gleichzeitig mit dem Erscheinen des Fotografen-Namens.

Beispiel-Code 26 (Subroutine ergänzt mit den Zeichenbefehlen für den Button_down)

```
SUB Draw_Label_Name
    Fill.LabelParameter(Button_down_Style)
    Label.Text(Label_FotoButton.$)
    Fill.LabelParameter(Name_Style)
    Label.Text(Label_Name.$)
ENDSUB
```


U007S001.EGI

Wir speichern und kompilieren das aktuelle Startview-Script, um es via Compact Flash Card auf den FOX zu übertragen und zu testen. – Drücken wir auf den Button, so gibt er nach und bleibt „eingedrückt“.

Als weitere Finesse soll nun der Button beim Loslassen wieder in seine ursprüngliche Stellung des Button_up „zurückspringen“. Um diesen Effekt zu erzeugen, genügt eine kleine Änderung im Hauptprogramm.

Die bereits bekannte Funktion `HotSpot.Install` kennt auch ein Argument für den Event „Loslassen“ (= Up):

```
HotSpot.Install(labelRelative24:Enter,labelRelative24:Leave
,labelRelative24:Down,labelRelative24:Up)
```

Dieses Argument war bisher mit `NIL` besetzt, d.h. ohne Wirkung. An dieser Stelle rufen wir nun die Subroutine `Draw_Button_Fotograf` auf, welche dafür sorgt, dass der Button_down beim Loslassen wieder mit dem Button_up überschrieben wird (Beispiel-Code 27).

Beispiel-Code 27 (HotSpot.Install mit geändertem Up-Argument)

```
; Einblenden des Label_Name mit HotSpot
    Load.Geometry_XYWH(25,440,60,30)
    HotSpot.Install(NIL,NIL,Draw_Label_Name,Draw_Button_Fotograf)
```


U007S003.EGI

Wir speichern und kompilieren das aktuelle Startview-Script, um es via Compact Flash Card auf den FOX zu übertragen und zu testen. Der Button bewegt sich ein und aus, wie gewünscht. Manchmal kann es aber passieren, dass wir zwar auf den Button drücken, aber während des Drückens das Button-Feld verlassen. In diesem Fall springt der Button nicht mehr zurück. In der Methode `HotSpot.Install()` heisst dieses Ereignis „Leave“:

```
HotSpot.Install(labelRelative24:Enter,labelRelative24:Leave
,labelRelative24:Down,labelRelative24:Up)
```

Wir wollen, dass der Button auch beim Ereignis „Leave“ zurückspringt. Um das zu erreichen, ersetzen wir an dieser Stelle das `NIL` ebenfalls mit dem Aufruf der Subroutine `Draw_Button_Fotograf` (Beispiel-Code 28). Beim Event „Enter“ soll weiterhin nichts geschehen.

Beispiel-Code 28 (HotSpot.Install mit geändertem Leave-Argument)

```
; Einblenden des Label_Name mit HotSpot
    Load.Geometry_XYWH(25,440,60,30)
    HotSpot.Install(NIL,Draw_Button_Fotograf,Draw_Label_Name,Draw_Bu
tton_Fotograf)
```

Wir speichern und kompilieren das aktuelle Startview-Script, um es via Compact Flash Card auf den FOX zu übertragen und zu testen. Nun sind wir mit dem Ergebnis zufrieden.

HotKey als Alternative zum HotSpot

Den FOX können wir auch über Tasten steuern. Die virtuelle Maschine des FOX registriert und speichert jeden Tastendruck, auch wenn bei einer View kein HotKey installiert ist. Solche nicht abgearbeiteten Tastaturereignisse bleiben im Eingangspuffer gespeichert, bis die Gelegenheit kommt, eine Aktion auszulösen, beispielsweise sobald eine View mit HotKey-Funktion geöffnet wird. Oft geschieht das ohne unseren ausdrücklichen Willen. Wir können solche ungewollten Aktionen verhindern, indem wir im Hauptprogramm vor der eigentlichen Installation des HotKeys die Methode `HotKey.InputFlush()` einbauen (vgl. Beispiel-Code 29). Damit wird der Eingangspuffer geleert.

Die Methode `HotKey.InstallLocalKey(Key,Event,Tag)` ist vergleichbar mit `HotSpot.Install(Enter,Leave,Down,Up)`. Mit „A“ vergeben wir die Funktion an die erste Taste auf dem Keyboard. Als zweites folgt der Aufruf einer Subroutine. An dritter Stelle können wir mit einem beliebigen 16-Bit Wert einen Tag definieren. Damit unterscheiden wir allenfalls verschiedene HotKeys, welche die gleiche Subroutine aufrufen können.

Beispiel-Code 29 (Installation eines HotKeys für den Aufruf des Namens des Fotografen)

```
; HotKey als Alternative für Fotograf-Button ----------------------
    HotKey.InputFlush()       ; Eingabepuffer leeren
    HotKey.InstallLocalKey("A",Draw_Label_Name,0)
```

Es können maximal 16 Tasten angeschlossen werden, entweder an den BoxHeader CK5 oder an die roten AMP microMatch-Stecker CK1 - CK4 (vgl. „Externe I/O“ in Abbildung 4, S.9). Auf dem BoxHeader sind alle 16 Tastatureingänge vorhanden. An die roten AMP-Stecker können 2 x 6 Tasten und 2 x 2 Tasten angeschlossen werden.
Die 16 Tasten sind mit den Buchstaben A-P bzw. a-p belegt. Die Gross-buchstaben stehen für den Event „Taste drücken“, die Kleinbuchstaben für „Taste loslassen“.

Mit dem Befehl `HotKey.EnableLocalKeys()` werden alle Local Keys in der HotKeyTabelle freigegeben und schlussendlich mit der Methode `HotKey.TableEnable()` aktiviert. Dadurch können eintretende Tastenereignisse von den entsprechend installierten EventHandlern ausgewertet werden. Beide Befehle schreiben wir nach `Display.Show()` ins Hauptprogramm (Beispiel-Code 30).

Beispiel-Code 30 (Die Tasten werden freigegeben und aktiviert am Schluss des Hauptprogramms)

```
    Display.Show()

    HotSpot.TableEnable()
    HotKey.EnableLocalKeys()
    HotKey.TableEnable()

    LOOP
    ENDLOOP
ENDVIEW
```

Die Startview kann nun, ergänzt mit Beispiel-Code 29, auch auf Tastendruck den Fotografennamen anzeigen. Die aufgerufene `SUB Draw_Label_Name` zeichnet auf dem Display gleichzeitig auch den Button „Fotograf“ neu als „eingedrückte“ Taste (vgl. Abbildung 36).

Abbildung 36: Anzeige nach Drücken der Taste A auf dem angeschlossenen Keyboard.

Im Weiteren wollen wir beim Loslassen der Taste A den Fotografennamen auf dem Display wieder verschwinden lassen. Der Name soll nur so lange angezeigt bleiben, solange wir auf die Taste drücken. Für den Event „Taste A loslassen“ brauchen wir bekanntlich den Kleinbuchstaben „a“. In diesem Fall soll eine neue Subroutine `SUB Clear_Label_Name` aufgerufen werden, die das Ursprungsbild der Startview wieder anzeigt, auf welcher kein Fotografenname sichtbar ist. Hierfür können wir die FOX-Spezialität der doppelten Videospeicherung nutzen (vgl. Übung 6, S.53 ◀).

Zunächst müssen wir in `SUB Draw_Label_Name` einen Kunstgriff vornehmen, damit sich die Aktionen dieser Subroutine einzig auf das RVR auswirken und die zu Beginn ins AVR geladene Startview unbehelligt bleibt. Das erreichen wir mit `Display.Direct` (Beispiel-Code 31).

Beispiel-Code 31 (Mit `Display.Direct` werden Button und Name nur in das RVR geladen)

```
SUB Draw_Label_Name
    Display.Direct()          ; schreibt direkt ins RVR
    Fill.LabelParameter(Button_down_Style)
    Label.Text(Label_FotoButton.$)
    Fill.LabelParameter(Name_Style)
    Label.Text(Label_Name.$)
ENDSUB
```

Die Subroutine `SUB Clear_Label_Name` ist dann kurz und bündig: mit `Display.Show()` kopieren wir den gesamten ursprünglichen Inhalt der AVR wieder ins RVR (Beispiel-Code 32).

Beispiel-Code 32 (Die Subroutine kopiert den Inhalt der AVR ins RVR)

```
SUB Clear_Label_Name
    Display.Show()            ; kopiert Inhalt AVR ins RVR
ENDSUB
```

Wir testen den aktuellen Code auf dem FOX und stellen zu unserer Freude fest, dass bei Tastendruck der Name des Fotografen erscheint und beim Loslassen wieder verschwindet. Entsprechend reagiert auch der Button „Fotograf“.

Damit der Name auch beim Bedienen des Touchscreen-Buttons erscheint und wieder verschwindet, verbinden wir den Up-Event des entsprechenden HotSpots ebenfalls mit der Subroutine `Clear_Label_Name` ().

Beispiel-Code 33 (Der Up-Event des HotSpots „Fotograf“ wird verbunden mit der Subroutine zum Löschen des Namens)

```
; HotSpot zum Einblenden des Label_Name ------------------------
    Load.Geometry_XYWH(400,420,100,50)
    HotSpot.Install(NIL,Draw_Button_Fotograf,Draw_Label_Name,Clear_L
abel_Name)
```

Übung 8: HotSpot mit Screenshot-Funktion

In dieser Übung lernen Sie

- ✓ HotSpot mit Screenshot-Funktion programmieren,
- ✓ einen FOX-Screenshot durchführen und auf den PC exportieren,
- ✓ ein EGI-Bild in ein gängiges Bildformat umwandeln.

Zeitbedarf: 15 Minuten (inkl. 5 Min. für Aufgabe 1)

Daten für die Übung: Projektordner „Schachbrett“ mit der Datei CHSS_001.EVS aus der Übung 7, den Sie selbst erstellt haben oder vorgefertigt im Ordner „Schnelleinstieg/Übung07“ vorfinden.

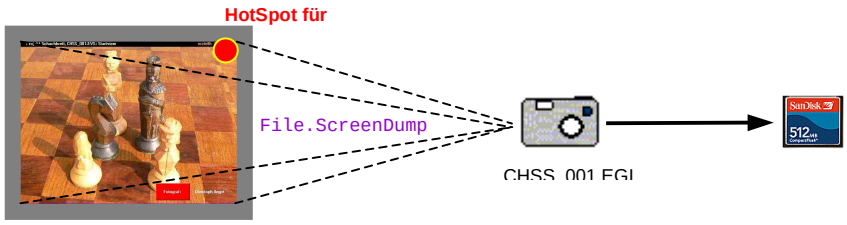


Abbildung 37: Die Methode `File.ScreenDump()` von eigerScript erstellt aus dem am Display dargestellten Inhalt des RVR eine Bilddatei im EGI-Format und speichert diese auf der Compact Flash Card ab.

Die Screenshots in diesem Dokument sind mit Hilfe eines speziellen eiger-Befehls generiert worden in Verbindung mit einem HotSpot (Abbildung 37). Für die Dokumentation eines eigerProjekts ist es oft von Interesse, eine Bildschirm-Darstellung des FOX als Bild festzuhalten und diesen Screenshot in einem gängigen Bildformat auf dem PC abzuspeichern. Zu diesem Zweck wollen wir auf unserer Startview einen HotSpot installieren, der auf Berührung einen Screenshot auslöst. Mit einem solchen HotSpot sind auch die Screenshots in diesem eiger-Schnelleinstieg „aufgenommen“ worden.

Unsichtbaren HotSpot einrichten

Die Screenshot-Methode hat in der eigerScript-Sprache die Syntax `File.ScreenDump(VarStr:FileName)`. Als Variable müssen wir der Methode zwischen den Klammern den Namen der EGI-Datei angeben, welche durch den Screenshot generiert wird (z.B. CHSS_001.EGI). dazu ist auch die genaue absolute Pfadangabe notwendig. Wir haben in der Übung 1 eigens für Screenshots den Ordner „DUMP“ eingerichtet (vgl. Abbildung 9, ◀). Die Screenshot-Funktion formulieren wir als Subroutine mit dem Namen `ScreenDump` (Beispiel-Code 34).

Beispiel-Code 34

```
SUB ScreenDump
    File.ScreenDump('C:\\CHSS\\DUMP\\CHSS_001.EGI')
ENDSUB
```

Als zweites schreiben wir ins Hauptprogramm die Befehlszeilen für die Dimensionierung und Installation eines HotSpots, wie wir sie bereits aus Übung 6 kennen ◀. Der Screenshot-HotSpot soll H x W = 40 x 40 Pixel gross sein und an der rechten oberen Ecke zu liegen kommen (X = 600, Y = 0). Der HotSpot soll nur auf Druck reagieren, und zwar mit dem Aufruf der Subroutine `ScreenDump`. Diese Befehlszeilen setzen wir an den Schluss des Hauptprogramms gemäss Beispiel-Code 35.

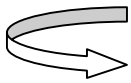
Beispiel-Code 35 (Screenshot-Befehlszeilen am Ende des Hauptprogramms)

```
; Hotspot rechts oben für Screenshot ------------------------
    Load.Geometry_XYWH(600,0,40,40)  ; Pos. & Dimension d. HotSpots
    HotSpot.Install(NIL,NIL,ScreenDump,NIL)

    Display.Show()

    HotSpot.TableEnable()
    HotKey.EnableLocalKeys()
    HotKey.TableEnable()

    LOOP
    ENDLOOP
ENDVIEW
```

Den HotSpot könnten wir auch noch sichtbar darstellen, indem wir an gleicher Stelle einen Button zeichnen, analog dem Button „Fotograf“ in der Übung 6 ◀. Für uns dient der Screenshot-HotSpot aber nur zu Entwicklungszwecken; es

genügt uns zu wissen, dass wir für ein Screenshot rechts oben auf das Touchpanel drücken müssen.

Nach Eingabe des Codes speichern wir das Ganze und kompilieren die View in die ausführbare EVI-Datei.



Aufgabe 1: Sofern an Ihrem FOX ein Keyboard angeschlossen ist, belegen Sie die zweite Taste mit der Screendump-Funktion (vgl. Übung 7, S.67). Einen Lösungsweg finden Sie im Anhang dieses Schnelleinstiegs (S.122). Zeitaufwand: 5 Minuten.

Testen der eingerichteten Screenshot-Funktion

Wir übertragen nun unser aktuelles Projekt mit Hilfe der Compact Flash Card auf den FOX. Wenn nach Reset und Aufstartphase die Startview erscheint, drücken wir auf den unsichtbaren Screenshot-HotSpot in der rechten oberen Ecke der Touchscreen.

Die virtuelle Maschine EVE braucht eine gewisse Zeit, um die Bildschirm-Daten zurück zu lesen und daraus ein EGI-Bild zu generieren. Mit etwa 10 Sekunden Geduld sind wir auf der sicheren Seite. Wenn wir die Compact Flash Card nach Drücken auf den HotSpot zu früh herausziehen, erhalten wir ein schwarzes Bild.
Die Screenshot-Funktion ist vorwiegend für die Entwicklung und Programmierung interessant und deshalb nicht auf Geschwindigkeit getrimmt. Die damit erstellte pixelgenaue EGI-Datei ist nicht komprimiert.

Die Wirkung dieses Berührungs-Events können wir erst überprüfen, wenn wir die Compact Flash Card wieder in den CardReader des PC eingeschoben haben. Nun sollten wir im Ordner „DUMP“ den Screenshot vorfinden mit dem von uns festgelegten Namen „CHSS_001.EGI“.

EGI-Screenshot in ein Windows Bildformat konvertieren

Den Screenshot im EGI-Format öffnen und konvertieren wir mit der eiger Graphic Suite (vgl. Abbildung 24):

1. eiger Graphic Suite aufrufen.

2. Applikation für die Konvertierung wählen: *Applikation* > *EGI* (vgl. Abbildung 20). In der Applikation EGI (Abbildung 24) wird das EGI-Format in BMP-Format gewandelt.
3. Bild laden: *EGI öffnen* (Button in der ersten Groupbox, vgl. Abbildung 24) > im Dateibrowser den Screenshot suchen (Ordner „DUMP“) > *Öffnen*. Die Datei wird geladen und unterhalb der Bedienelemente angezeigt. Auf der rechten Seite steht die Dateigrösse und die Breite und Höhe des Bildes (Anzahl Pixel).
4. Falls nötig Farbreduktion wählen: *5 Bit* (Button in der zweiten Groupbox, vgl. Abbildung 24). Der von EVE generierte Screenshot ist bereits in der 5 Bit-Farb-Codierung. Die Anzahl Farben wird deshalb nur reduziert, wenn wir „4 Bit“ und „3 Bit“ wählen. Die Wirkung der Farbreduzierung auf das Bild wird unten angezeigt, die Auswirkung auf Bildgrösse und Kompression rechts in der Groupbox 2.
5. Bild als BMP speichern: *Bild als BMP* (Button in der dritten Groupbox, vgl. Abbildung 24). Der Browser für das Speichern öffnet sich, und das File kann mit entsprechendem Namen abgelegt werden.
6. Wenn wir statt BMP ein anderes Bild-Format wünschen (z.B. JPEG oder PNG), können wir beispielsweise mit Hilfe des Microsoft Photo Editors eine weitere Konvertierung vornehmen.

Übung 9: Eine zweite View erstellen

In dieser Übung lernen Sie

- ✓ eine zweite View erstellen.
- ✓ einen View-Wechsel programmieren.
- ✓ das Display löschen.
- ✓ eine Hintergrundfarbe für ein View einrichten.

Zeitbedarf: 60 Minuten (inkl. 20 Min. für Aufgabe 2)

Daten für die Übung: Projektordner „Schachbrett“ mit der Datei CHSS_001.EVS aus der Übung 8, den Sie selbst erstellt haben oder vorgefertigt im Ordner „Schnelleinstieg/Übung08“ vorfinden.

Neue View 2 nach Vorlage View 1

In der Regel besteht ein eigerProjekt aus mehreren Views, zwischen welchen hin und her gesprungen werden kann (vgl. Abbildung 12, S.14). Auch wir wollen von unserer Startview aus zu einer weiteren View wechseln können. Unsere View 2 soll ein Schachbrett mit verschiedenen Funktionen enthalten. Zunächst erstellen wir eine simple View 2 mit holzfarbenem Hintergrund.

Für View 2 nehmen wir die Startview (View 1) als Vorlage. Dies macht es uns auch einfacher, eine gewisse Kontinuität von einer View zur andern zu gewährleisten. Wir öffnen unsere Startview aus Übung 8 und speichern sie im gleichen Ordner als CHSS_002.EVS. Das Programm-Script passen wir nun den Bedürfnissen der View 2 an, d.h. wir nehmen die Änderungen vor, die in Tabelle 8 aufgelistet sind. Den Kopfkommentar und einleitende Definitionen, sowie die Importbefehle für die Header-Dateien können wir weitgehend unverändert übernehmen.

Das Info Window links unten im eiger Studio (vgl. Abbildung 6, S.19) gibt an, in welcher Zeile (Line) sich der Cursor aktuell befindet (Abbildung 38). Dies kann für Dokumentationszwecke nützlich sein.

Abbildung 38:
Info Window im eiger Studio gibt die aktuelle Position des Cursors an (Zeile und Kollonne).

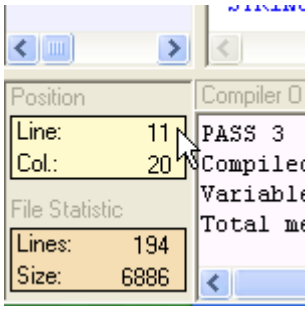


Tabelle 8:
Code-Änderungen für View 2 gegenüber der Vorlage View 1

Zeile(n)*	Subroutine, Style, Haupt-progr. etc.	alter Code	neuer Code (Änderungen gelb markiert)
2	Script-Kopf	`; Titel: Schachbrett View 1`	`; Titel: Schachbrett View 2`
3	Script-Kopf	`; File: CHSS_001.EVS`	`; File: CHSS_002.EVS`
11	Script-Kopf	`Initialversion: 06.03.2007`	**aktuelles Datum**
21	Einleitende Definitionen	`EIGERVIEW 1 ; Viewnummer: zweiter Teil des EVI-Dateinamens: CHSS_001.EVI`	`EIGERVIEW 2 ; Viewnummer: zweiter Teil des EVI-Dateinamens: CHSS_002.EVI`
30	Kommentar	`; Definitionen für Startview ------------`	`; Definitionen für View 2 ---------------`
32	String-Def.	`STRING [60] Titel.$ = 'Projekt Schachbrett, CHSS_001.EVS: Startview'`	`STRING [60] Titel.$ = 'Projekt Schachbrett, CHSS_002.EVS: View 2'`
33	String-Def.	`STRING [60] Titel_Right.$ = 'erstellt: 16.03.2007'`	**aktuelles Datum**
34	String-Def.	`STRING [15] Label_FotoButton.$ = 'Fotograf:'`	**Zeile löschen**
34	String-Def.	`STRING [15] Label_Name.$ = 'Christoph Angst'`	**Zeile löschen**
46-49	Subroutine	`SUB Draw_Button_Fotograf ... ENDSUB`	**Zeilen, d.h. ganze Subroutine löschen**
46-51	Subroutine	`SUB Draw_Label_Name ... ENDSUB`	**Zeilen, d.h. ganze Subroutine löschen**
46-48	Subroutine	`SUB Clear_Label_Name Display.Show ENDSUB`	**Zeilen, d.h. ganze Subroutine löschen**
46-48	Subroutine	`SUB ScreenDump File.ScreenDump('C:\\CHSS\\DUMP\\CHSS_001.EGI') ENDSUB`	`SUB ScreenDump File.ScreenDump('C:\\CHSS\\DUMP\\CHSS_002.EGI') ENDSUB`
51-150	Subroutine	`SUB Styles ... ENDSUB`	**nicht löschen** (wahrscheinlich werden wir diese Styles für neue Buttons in View 2 gebrauchen)
156	Hauptprogr.	`Load.Pos_X1Y1(0,0)`	**Zeile löschen**
156	Hauptprogr.	`File.Read_EGI('C:\\CHSS\\PICT\\CHESSSET.EGI')`	**Zeile löschen**

157	Hauptprogr.	`CallSubroutine(Draw_Button_Fotograf)`	**Zeile löschen**
157	Hauptprogr.	`;CallSubroutine(Draw_Label_Name)`	**Zeile löschen**

158-160	Hauptprogr.	`; HotSpot zum Einblenden des Label_Name --------`
		`Load.Geometry_XYWH(400,420,100,50)`
		`HotSpot.Install(NIL,Draw_Button_Fotograf,Draw_Label_Name,Draw_Button_Fotograf)`
			Zeilen löschen

162	Hauptprogr.	`; HotKey als Alternative für Fotograf-Button ---`	`; HotKey für Screenshot ----`
164	Hauptprogr.	`HotKey.InstallLocalKey("A",Draw_Label_Name,0)`	**Zeile löschen**
164	Hauptprogr.	`HotKey.InstallLocalKey("a",Clear_Label_Name,0)`	**Zeile löschen**
164	Hauptprogr.	`; HotKey für Screenshot -`	**Zeile löschen**

* In Ihrem Script mag die Zeilennummer (vgl. Abbildung 38) differieren. In diesem Fall können Sie sich an den andern Code-Beschreibungen orientieren. Die hier angegebene Zeilennummer berücksichtigt, dass wir während des Durchgangs der Tabelle die nicht benötigten Codezeilen laufend löschen. Deshalb gibt es z.B. zweimal eine Zeile 34, die gelöscht werden muss.

Mit diesen Änderungen haben wir wesentliche Komponenten der Startview bequem übernommen. Die Titelleiste ist damit in der View 2 bereits vorhanden und angepasst. Nachdem wir die Anpassungen vorgenommen haben, kompilieren wir die View 2. Falls der Kompiliervorgang mit einer Fehlermeldung abgebrochen wird, helfen uns die Hinweise des Compiler Outputs im Log Window (vgl. Abbildung 6, S.19). Nach dem Kompilieren sollte der CHSS-Ordner um zwei Dateien reicher sein: **CHSS_002.EVS** und **CHSS_002.EVI** (vgl. Abbildung 39). Leider können wir unser Ergebnis nicht auf dem FOX-Display betrachten, da uns der Link von der Startview zur View 2 noch fehlt.

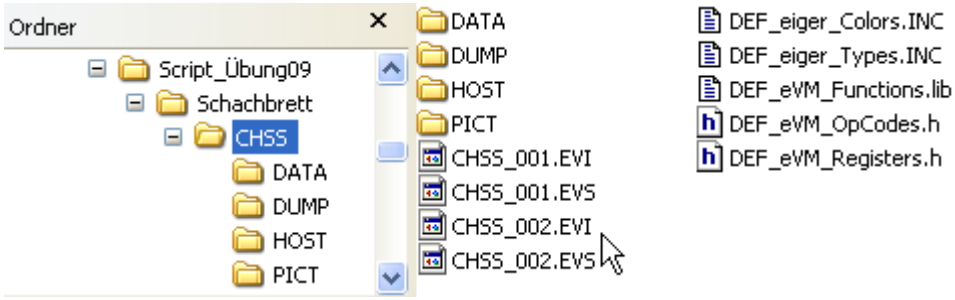


Abbildung 39:
CHSS-Ordner mit den View 2 -Programmdateien CHSS_002.EVS und CHSS_002.EVI

Button für den Link von der Startview zur View 2

Nachdem die View 2 so weit vorbereitet, gespeichert und kompiliert ist, schliessen wir CHSS_002.EVS und öffnen im eigerStudio wieder die Startview (CHSS_001.EVS). Dort wollen wir nun einen Link-Button einrichten, der den Szenenwechsel zu View 2 ermöglicht.

Als erstes zeichnen wir in der Startview einen neuen Button. Die Subroutine `Draw_Button_View2` soll uns den Button zeichnen (vgl. Beispiel-Code 36). Den neuen Subroutinen-Code schreiben wir direkt im Anschluss an die Subroutine `Draw_Button_Fotograf` (ca. Zeile 54). Um uns die Arbeit zu erleichtern, duplizieren wir einfach die Subroutine `Draw_Button_Fotograf` und ändern den Namen zu `Draw_Button_View2`.

Beispiel-Code 36 (Subroutine in der Startview für den Link-Button zu View 2, noch nicht komplett)

```
SUB Draw_Button_View2
    Fill.LabelParameter(Button_up_Style)
    Label.Text(Label_Button_View2.$)
ENDSUB
```

`Fill.LabelParameter(Button_up_Style)` können wir beibehalten, da der neue Button gleich aussehen soll wie der Button „Fotograf“. Dadurch können wir uns auch auf den gleichen `Button_up_Style` beziehen.

Hingegen muss sich die Methode `Label.Text(VarStr)` für die Button-Aufschrift auf einen neuen String beziehen, den wir erst noch definieren müssen. Diesem String werden wir den Alias-Namen `Label_Button_View2.$` vergeben (vgl. Beispiel-Code 38). Darum schreiben wir: `Label.Text(Label_Button_Vi ew2.$)`.

Da sich die Lage der beiden Buttons unterscheidet, müssen wir die Definition für die X- und Y-Lage der beiden Buttons aus dem gemeinsamen `Button_up_Style` auslagern und direkt in den jeweiligen Subroutinen regeln. Zunächst positionieren wir den View2-Button auf dem Touchscreen links unten, auf gleicher Höhe wie der Button „Fotograf“. Beim Suchen einer passenden Lage können wir den Pixel-Massstab zu Hilfe nehmen. Die Koordinaten müssen wir einzeln den entsprechenden Registern zuweisen. Dafür dient uns das Variables/Register Window von eiger Studio (Abbildung 6, S.19). In diesem Window können wir mit dem ersten Button unterhalb des Fensters entweder die Variablen

oder die Register auflisten lassen (Abbildung 40). Wir suchen in der Register-Liste nach `eI.Pos_X1` und `eI.Pos_Y1` und fügen diese Codes mit Doppelklick an gewünschter Stelle ein (Beispiel-Code 37). Mit „**:=**“weisen wir die X- und Y-Werte (X = 20 und Y = 420) den Registern zu. Es ist wichtig, dass wir die speziell definierten Eigenschaften – in unserem Fall X und Y – **vor** den Methoden positionieren, mit welchen der Button dann gezeichnet wird.

In eigerScript wird bei Zuweisungen zu Variablen innerhalb Routinen immer ":=" gebraucht (mit vorgesetztem Doppelpunkt!). Initialwerte bei der Deklaration von Variablen wie auch Konstanten werden mit "=" zugewiesen.

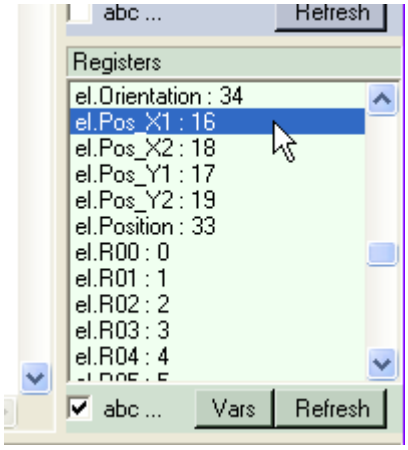


Abbildung 40: Variables/Register Window im eiger Studio. Listet in diesem Beispiel die eiger Register in alphabetischer Reihenfolge auf. Mit dem Button „Vars“ wird zur Variablen-Liste gewechselt. Per Doppelklick auf das gewünschte Register wird dessen Code im Code-Window eingefügt.

Beispiel-Code 37 (komplettierte Subroutine in der Startview für den Link-Button zu View 2)

```
SUB Draw_Button_View2
    eI.Pos_X1 := 20
    eI.Pos_Y1 := 420
    Fill.LabelParameter(Button_up_Style)
    Label.Text(Label_Button_View2.$)
ENDSUB
```

Bei den Stringdefinitionen fügen wir nun die neue Code-Zeile für den String `Label_Button_View2.$` ein (Beispiel-Code 38). Der Button_View2 soll mit „Schachbrett >>“ angeschrieben sein (vgl. Abbildung 41).

Beispiel-Code 38 (Stringdeklaration für den neuen Button_View2, zugefügt in der Startview)

```
; Definitionen für Startview -------------------------------------

STRING  [60] Titel.$ = 'Projekt Schachbrett, CHSS_001.EVS: Startview'
STRING  [60] Titel_Right.$ = 'erstellt: 16.03.2007' ; Titelleiste rechts
STRING  [15] Label_FotoButton.$ = 'Fotograf:'
STRING  [15] Label_Name.$ = 'Christoph Angst'
STRING  [15] Label_Button_View2.$ = 'Schachbrett >>'
```

Unsere Änderungen für den neuen Button haben anderweitig noch zwei wichtige Anpassungen zur Folge:

1. In Beispiel-Code 37 haben wir die Definition der Button-Koordinaten vorweggenommen, damit wir vom bereits vorhandenen `Button_up_Style` profitieren können. Dies bedingt nun auch eine Anpassung bei den X- und Y-Registern des betreffenden Styles (Beispiel-Code 39). Die bisher konkreten Werte für X und Y ersetzen wir mit `no_change`. Das bedeutet, dass der Style an diesen Eigenschaften, die wir für die einzelnen Buttons speziell definiert haben, nichts ändern soll.

2. Die X- und Y- Werte, die wir in Beispiel-Code 39 mit no_change ersetzt haben, betreffen den Fotograf-Button. Diese müssen wir nun direkt in der Subroutine `Draw_Button_Fotograf` den Positionsregistern zuweisen (Beispiel-Code 40).

Beispiel-Code 39 (Auslagerung der Variablen-Zuweisung für X- und Y-Register in der Startview)

```
Button_up_Style:
    INLINEWORDS  (no_change)         ; entspricht eI.Pos_X1
    INLINEWORDS  (no_change)         ; entspricht eI.Pos_Y1
```

Beispiel-Code 40 (Zuweisung der X- und Y-Werte vor Ort in der Startview)

```
SUB Draw_Button_Fotograf
    eI.Pos_X1 := 400
    eI.Pos_Y1 := 420
    Fill.LabelParameter(Button_up_Style)
    Label.Text(Label_FotoButton.$)
ENDSUB
```

Die Button-Installation schliessen wir nun noch mit einem `CallSubroutine` im Hauptprogramm ab (Beispiel-Code 41).

Beispiel-Code 41 (Aufruf der Subroutine Draw_Button_View2, zugefügt im Hauptprogramm der Startview)

```
BEGINVIEW
    EVE.Init()                       ; EVE ANNA initialisieren
    Display.Prepare()
    Load.Pos_X1Y1(0,0)               ; linke obere Ecke
    File.Read_EGI('C:\\CHSS\\PICT\\CHESSSET.EGI')
    CallSubroutine(Draw_Titel)
    CallSubroutine(Draw_Button_Fotograf)
    CallSubroutine(Draw_Button_View2)
;   CallSubroutine(Draw_Label_Name)
```

U009S001.EGI

Wir kompilieren die Startview und begutachten die erweiterte Startview auf dem FOX-Display.

Abbildung 41:
Startview erweitert mit Button „Schachbrett“

HotSpot für den Link von der Startview zur View 2

Alle Views eines Projekts müssen im gleichen Ordner abgelegt sein.

Nun müssen wir dem Schachbrett-Button noch einen HotSpot hinterlegen, damit wir mittels Knopfdruck zur View 2 wechseln können. Im Programmcode der Startview (CHSS_001.EVS) formulieren wir den View-Wechsel als eine weitere Subroutine, die wir `ToView2` nennen (Beispiel-Code 42). Der FOX wechselt die View aufgrund des Befehls `GotoView(VarInt)`. Die geklammerte Variable ist eine dreistellige Zahl, in unserem Beispiel 002. Sie besteht aus den letzten drei Ziffern der *.EVI-Datei (z.B. CHSS_002.EVI), die wir bei Knopfdruck aufrufen möchten. In dieser Datei ist der Programmcode für View 2 gespeichert. Sie muss im gleichen Ordner abgelegt sein wie die Startdatei. Die neue Subroutine können wir dem Reigen der kleinen Subroutinen in der ersten Hälfte des Startview-Scripts beifügen.

Beispiel-Code 42 (neue Subroutine in der Startview für den Wechsel zur View 2 = CHSS_**002**)

```
SUB ToView2
    GotoView(002)
ENDSUB
```

Im Hauptprogramm installieren wir nun den HotSpot für den Schachbrett-Button (Beispiel-Code 43). Position und Geometrie des HotSpots sind deckungsgleich mit dem Schachbrett-Button. Sobald das Ereignis „down“ (= Knopfdruck) eintritt, ruft das Programm die Subroutine `ToView2` auf.

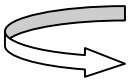
Beispiel-Code 43 (neue HotSpot-Installation im Hauptprogramm der Startview)

```
; HotSpot zum Wechsel zu View 2 -----------------------------
    Load.Geometry_XYWH(20,420,100,50)
    HotSpot.Install(NIL,NIL,ToView2,NIL)
```

Alle unsere Änderungen und Erweiterungen überprüfen wir nun auf dem FOX. Dabei müssen wir nicht nur das neueste CHSS_001.EVI sondern auch die neue CHSS_002.EVI auf den FOX übertragen (vgl. Abbildung 39). Beim Test auf dem FOX erkennen wir den Effekt des Knopfdrucks nur, wenn wir dabei die Titelleiste genau beobachten. Dort ändern sich Dateiname, View-Nummer und Erstelldatum. Ansonsten bleibt alles beim Alten. View 2 besteht zurzeit nur aus der Titelleiste, welche diejenige der Startview überschreibt (Abbildung 42).


U009S002.EGI

Abbildung 42:
Titelleiste von View 2 über der Startview.



Aufgabe 2: Der View2-Button ist noch nicht animiert, d.h. er „bewegt“ sich nicht, wenn wir darauf drücken. Versuchen Sie, den neuen Button analog zum Fotograf-Button zu animieren (vgl. Übung 7, S.64 ◀). Einen Lösungsweg finden Sie im Anhang dieses Schnelleinstiegs (S.122 ▶). Zeitaufwand: 20 Minuten.

Display löschen

Damit der FOX den Bildschirm ganz neu beschreibt, müssen wir einen Löschbefehl vorlagern, mit welchem die Startview ganz aus dem Videospeicher entfernt wird. Dieser Befehl heisst `Display.Clear()`. Mit dieser Methode wird die Startseite ganzflächig mit einer einheitlichen Farbe „übermalt“. Die Default-Farbe ist `silver`.

Beispiel-Code 44 (Löschbefehl für alte View vor dem Aufbau der neuen View)

```
BEGINVIEW
    EVE.Init()                      ; EVE ANNA initialisieren
    Display.Clear()                       ; Bildschirm löschen
    Display.Prepare()
```

Hintergrundfarbe für View 2

Wir wünschen aber für unsere View 2 eine braune Hintergrundfarbe, welche wir vor dem Löschbefehl dem Register `eI.DisplayColor` zuweisen. Die beiden Code-Zeilen setzen wir direkt anschliessend an `EVE.Init()` ins Hauptprogramm ein.

Beispiel-Code 45 (zuweisen einer Hintergrundfarbe für die neue View; default ist „silver“)

```
BEGINVIEW
    EVE.Init()                      ; EVE ANNA initialisieren
    eI.DisplayColor := burlywood
    Display.Clear()                       ; Bildschirm löschen
    Display.Prepare()
```


U009S003.EGI

Wir testen unseren neuesten Programmcode auf dem FOX. Beim Drücken des Schachbrett-Button in der Startview erscheint nun View 2 bildschirmfüllend (Abbildung 43).

Abbildung 43: Screenshot von View 2 mit holzfarbenem Hintergrund.

Übung 10: Includefile

In dieser Übung lernen Sie

- ✓ Includefiles erstellen.
- ✓ Gemeinsame globale Konstanten und Variablen in Includefiles auslagern.
- ✓ Gemeinsame Subroutinen – bzw. Teile davon – in Includefiles auslagern.

Zeitbedarf: 35 Minuten (inkl. 10 Minuten für Aufgabe 3)

Daten für die Übung: Projektordner „Schachbrett“ mit den Dateien CHSS_001.EVS und CHSS_002.EVS aus der Übung 9, den Sie selbst erstellt haben oder vorgefertigt im Ordner „Schnelleinstieg/Übung09“ vorfinden (Variante mit dem animierten Button aus Lösung B, S.124 ▶).

Globale Konstanten und Variablen zentral verwalten

Globale Konstanten und Styles, die mehr als nur eine View betreffen, werden mit Vorteil in einer Include-Datei deklariert und mit `INCLUDEFILE` in den Views genutzt.

Ein eigerProjekt besteht normalerweise aus mehreren Views, die bestimmte Merkmale gemeinsam haben. Sie sind z.B. meist nach einheitlichem Design aufgebaut mit gleicher Titelleiste, gleichem Hintergrundbild, gleicher Button-Gestaltung (Styles) etc. Deklarationen, die mehrere Views betreffen, brauchen nicht in jeder View wiederholt zu werden. Das Projekt ist übersichtlicher und auch leichter zu aktualisieren, wenn wir diese globalen Konstanten und Variablen „zentral“ in einer speziellen Datei verwalten. Wir nennen solche Dateien „Includefiles“. eigerScript kann Includefiles mit dem Schlüsselwort `INCLUDEFILE` in eine View einbetten und dadurch auf alle darin festgelegten Konstanten- und Variablendeklarationen zugreifen. Wir kennen den Befehl bereits aus der Startview (Beispiel-Code 3, S.32), wo wir auf diese Weise wie selbstverständlich die Farbdefinitionen und Schriftfonts benutzen konnten, welche in den Header-Dateien von eigerScript als „Global Constants“ vordefiniert sind.

Beim Programmieren wird zwischen Variablen und Konstanten unterschieden. Dieses Thema und die damit verbundenen Regeln in eigerScript sind im Anhang eingehend beschrieben (S.116-118).

Schreibweise von Zahlen in eigerScript

eigerScript unterstützt keine Gebietsschemata. Das Dezimaltrennzeichen ist der Punkt. Kommas sind bei Fliesskommazahlen nicht erlaubt. Auch Tausender Trennzeichen, wie 1'000 oder 1.000, führen beim Kompilieren zu Fehlern. Hingegen kennt eigerScript die Exponentialschreibweise, z.B. 2.7E15.

Auslagern globaler Konstanten und Variablen in ein Includefile

Unser Projekt weist einige Redundanzen auf, die wir zumindest teilweise vermeiden können. Es gibt Konstanten, Variablen und auch Subroutinen, die sowohl in der Startview als auch in der View 2 vorkommen. Statt diese wiederholt in diverse Views schreiben zu müssen, wollen wir ein Includefile erstellen, in welchem auf gemeinsamen Code von allen Projektviews nach Bedarf zugegriffen werden kann.

Tabelle 9: Gemeinsame Konstanten und Variablen in Startview und View 2

Konstante oder Variable	Code
Erstelldatum	`STRING [60] Titel_Right.$ = 'erstellt: 29.03.2007'`
Style von Titelleiste links	`Titel_Style: ...`
Style von Titelleiste rechts	`Titel_Right_Style: ...`

Wir erstellen im eigerStudio ein Includefile, das wir CHSS_Global.EVS nennen. Diese Datei speichern wir ebenfalls direkt im Ordner CHSS ab (Abbildung 44).

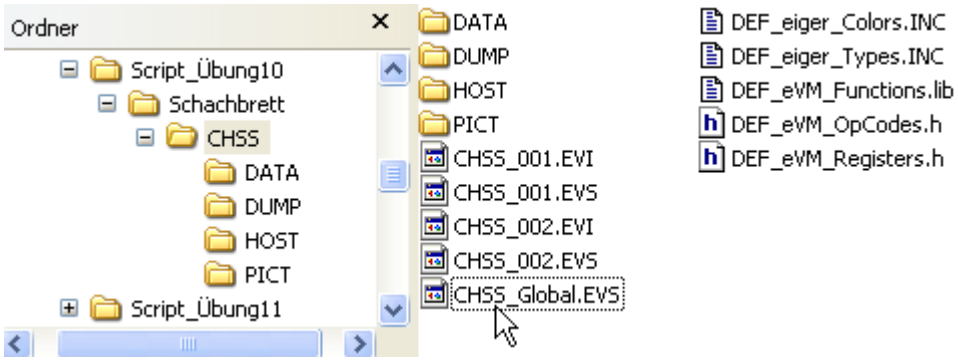


Abbildung 44:
CHSS-Ordner mit dem Includefile CHSS_Global.EVS

Die Dateinamen reiner Includefiles sind nicht auf 8+3 Zeichen beschränkt, weil der Compiler auf PC-Ebene den Inhalt dieser Dateien völlig in die Views einbettet. Die Includefiles erscheinen dann auf der DOS-Ebene des FOX nicht mehr als Datei.

Includefiles werden im gleichen Ordner und auf gleicher Ordner-Ebene gespeichert wie die Views.

Den Kopfkommentar für CHSS_Global.EVS übernehmen wir mit den entsprechenden Anpassungen von der Startview.

Beispiel-Code 46: (Kopfkommentar im Includefile CHSS_Global.EVS)

```
;-------------------------------------------------------------------
; Titel       : Schachbrett   INCLUDEFILE
; File        : CHSS_Global.EVS
;-------------------------------------------------------------------
; Compiler    : eigerScript
;
; System      : eigergraphics.com; FOXS embedded computer
;
; Beschreibung: eigerScript-Schnelleinstieg, Ergebnis von Übung 10
;
; Version     : Initialversion: 3.04.2007
;
; Autor       : Bernhard Eiger
;
;-------------------------------------------------------------------
; (c) 2005-2007     MeineFirma AG, CH-8000 Zürich; 043 7634 53 12
;-------------------------------------------------------------------
```

Im Unterschied zu den Views muss das Includefile nicht kompiliert werden.

Der rechte Teil der Titelliste ist in unserem Projekt für jede View identisch. Das Erstellungsdatum ist für uns identisch mit dem Aktualisierungsdatum, welches wir dank der Auslagerung zentral im Includefile anpassen können. Wir kopieren deshalb als erstes den `STRING Titel_Right.$` in aus der Startview oder aus der View 2 die Includefile (Beispiel-Code 47). Diese Stringdeklaration ist nun in der

Startview und in der View 2 überflüssig – wir können sie dort löschen. Während dieses Transfers kompilieren wir die Views nicht, sondern begnügen uns mit dem Speicherbefehl.

Anstelle der gelöschten Codezeile müssen wir das Includefile mit Hilfe des Schlüsselworts `INCLUDEFILE` in den beiden Views einbetten. Den entsprechenden Code fügen wir ein im Anschluss an die Codezeilen für die Header-Dateien, und zwar bei beiden Views (Beispiel-Code 48).

Beispiel-Code 47 (erste Variable im Includefile, verschoben aus der Startview und der View 2)

```
STRING [60] Titel_Right.$ = 'erstellt: 16.03.2007'   ; Titelleiste rechts
```

Beispiel-Code 48 (neuer Include-Befehl für CHSS_Global.EVS in CHSS_001.EVS und CHSS_002.EVS)

```
INCLUDEFILE 'DEF_eiger_Colors.INC'   ; Farbdefinitionen 12.03.2006
INCLUDEFILE 'DEF_eiger_Types.INC'    ; eiger Definitionen

INCLUDEFILE 'CHSS_Global.EVS' ; Includefile für Schachbrett-Projekt
```

Zum Test setzen wir im Includefile das aktuelle Datum ein (z.B. `'erstellt: 4.04.2007'`) und kompilieren CHSS_001.EVS und CHSS_002.EVS. Während dieses Prozesses integriert der eigerCompiler das Includefile CHSS_Global.EVS in den Byte-Code jeder kompilierten View. Darum ist das separate Kompilieren des Includefiles nicht nötig – und auch nicht möglich, weil es kein Hauptprogramm enthält. Die neuen EVI-Dateien übertragen wir via Compact FlashCard auf den FOX und überprüfen das Ergebnis. – Wir stellen fest, dass die eine Datumsänderung im Includefile sich auf beide Views ausgewirkt hat, weil beide Views nach der Änderung wieder kompiliert worden sind.

Änderungen im Includefile sind erst wirksam, wenn die betroffenen Views neu kompiliert worden sind.
Variablen und Konstanten können im Includefile quasi in einem Arbeitsgang für alle Views deklariert werde, in denen das Includefile integriert ist. Sie gelten aber doch nicht als „global“, weil der Compiler diese Includefile mehrmals kompiliert, d.h. mit jeder einzelnen View, die mit einem Include-Befehl auf dieses Includefile zugreift.

Zentral verwaltete Subroutinen

In Includefiles können auch gemeinsame Subroutinen verwaltet werden.

Nach erfolgreichem Test überführen wir nun auch die anderen gemeinsamen Konstantendeklarationen (vgl. Tabelle 9) von den Views in das Includefile. Es handelt sich dabei um die Styles der Titelleiste. Styles sind nichts anderes als Konstantendeklarationen en block. Ihre Deklaration findet in Form von Subroutinen statt (z.B. `SUB Styles`). Die Views eines Projektes können sich im Includefile auch Subroutinen teilen. Wir überführen darum auch die beiden Titel-Styles ins Includefile. In den Views sind diese Styles Teil der Subroutine `Styles`. Dort nehmen wir diese gemeinsamen Styles heraus und fügen sie als neue Subroutine ins Includefile ein (Beispiel-Code 49). In den Views ist die Subroutine `Styles` nun kleiner geworden, bleibt aber weiterhin bestehen. Darum müssen wir die Subroutine im Includefile anders benennen: `GlobalStyles`.

Beispiel-Code 49 (Neue Style-Subroutine `GlobalStyles` mit den Titel-Styles im Includefile)

```
SUB GlobalStyles
Titel_Style:
    INLINEWORDS  (0)          ; entspricht eI.Pos_X1
    INLINEWORDS  (0)          ; entspricht eI.Pos_Y1
    INLINEWORDS  (640)        ; entspricht eI.Width
    INLINEWORDS  (18)         ; entspricht eI.Height
    INLINEWORDS  (20)         ; entspricht eI.SpaceLeft
    INLINEWORDS  (8)          ; entspricht eI.SpaceRight
    INLINEWORDS  (0)          ; entspricht eI.HorizontalAdjust
    INLINEWORDS  (0)          ; entspricht eI.VericalAdjust
    INLINEWORDS  (black)      ; entspricht eI.FillColor
    INLINEWORDS  (black)      ; entspricht eI.BackColor
    INLINEWORDS  (black)      ; entspricht eI.LineColor
    INLINEWORDS  (lightyellow)       ; entspricht eI.TextColor
    INLINEWORDS  (Pos_left)          ; entspricht eI.Position
    INLINEWORDS  (Orientation_0deg)  ; entspricht eI.Orientation
    INLINEWORDS  (normal)            ; entspricht eI.Appearance
    INLINEWORDS  (no_border)         ; entspricht eI.BorderStyle
    INLINEWORDS  (Font_System_9bd)   ; entspricht eI.FontNumber
    INLINEWORDS  (silver) ; entspr.eI.BackgroundColor


Titel_Right_Style:
    INLINEWORDS  (0)          ; entspricht eI.Pos_X1
    INLINEWORDS  (0)          ; entspricht eI.Pos_Y1
    INLINEWORDS  (640)        ; entspricht eI.Width
    INLINEWORDS  (18)         ; entspricht eI.Height
    INLINEWORDS  (20)         ; entspricht eI.SpaceLeft
    INLINEWORDS  (8)          ; entspricht eI.SpaceRight
    INLINEWORDS  (0)          ; entspricht eI.HorizontalAdjust
    INLINEWORDS  (0)          ; entspricht eI.VericalAdjust
    INLINEWORDS  (transparent)       ; entspricht eI.FillColor
    INLINEWORDS  (black)             ; entspricht eI.BackColor
    INLINEWORDS  (black)             ; entspricht eI.LineColor
    INLINEWORDS  (lightsalmon)       ; entspricht eI.TextColor
    INLINEWORDS  (Pos_right)         ; entspricht eI.Position
    INLINEWORDS  (Orientation_0deg)  ; entspricht eI.Orientation
    INLINEWORDS  (normal)            ; entspricht eI.Appearance
    INLINEWORDS  (no_border)         ; entspricht eI.BorderStyle
    INLINEWORDS  (Font_System_9bd)   ; entspricht eI.FontNumber
```

```
    INLINEWORDS  (silver) ; entspr.eI.BackgroundColor
ENDSUB
```

Ein erneuter Test auf dem FOX zeigt, dass auch das Auslagern von Subroutinen funktioniert. Die Titelleisten der beiden Views erscheinen so, wie wir es uns aus den bisherigen Übungen gewohnt sind.

Unsere Views beinhalten noch eine weitere gemeinsame Subroutine, die wir ins Includefile verschieben können. In beiden ist die gleiche Subroutine `Draw_Titel` für das Zeichnen und Beschriften der Titelleiste zuständig (Beispiel-Code 50). Wir kopieren diese Subroutine ins Includefile und löschen den entsprechenden Code in den Views.

Beispiel-Code 50 (Subroutine für das Zeichnen und Beschriften der Titelleiste wird ins Includefile verlegt)

```
SUB Draw_Titel
    ; Titelleiste schreiben ---------------------------------------

    Fill.LabelParameter(Titel_Style)
    Label.Text(Titel.$)

    Fill.LabelParameter(Titel_Right_Style)
    Label.Text(Titel_Right.$)
ENDSUB
```

Wenn wir nach dieser Aktion die Views kompilieren wollen, erscheint eine Fehlermeldung (vgl. Abbildung 45). Nebst der Fehlermeldung fällt die Markierung im Includefile auf die Methode `Label.Text(Titel.$)`, die in der Subroutine `Draw_Titel` den linken Titeltext schreiben sollte. Zudem gibt uns das Log Window im unteren Teil des eigerStudio den Hinweis, dass der Variablenname `Titel.$` nicht bekannt sei.

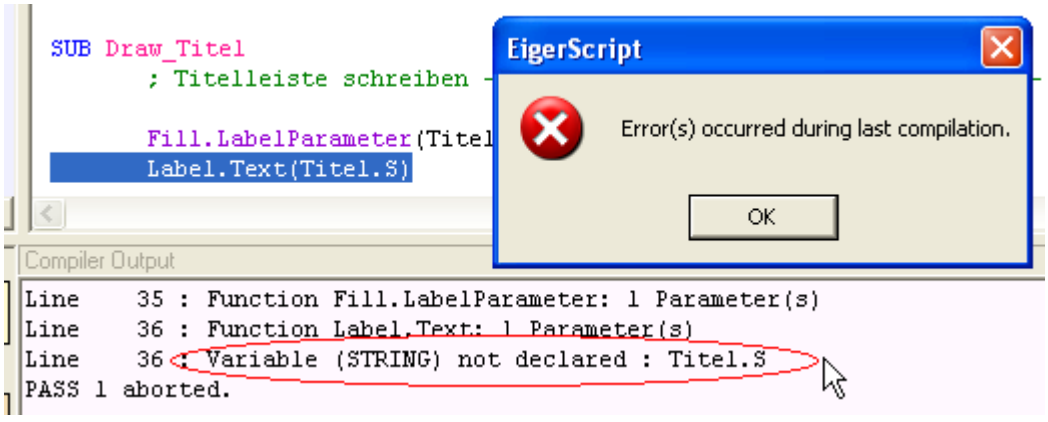


Abbildung 45: Fehlermeldung im Compiler Output beim Kompilieren der Startview nach Auslagerung von SUB `Draw_Titel` ins Includefile.

Die Reihenfolge von Befehlszeilen kann für das Funktionieren des Programms entscheidend sein.

Gehen wir den Programmverlauf in der Startview – oder auch in der View 2 – Schritt für Schritt durch, so fällt uns auf, dass das Includefile zeitlich vor der Deklaration des `Titel.$` in die View eingefügt wird. Zu diesem Zeitpunkt hat also die Subroutine `Draw_Titel` im Includefile noch keine Kenntnis über die Stringvariable `Titel.$`. (Beispiel-Code 51).

Beispiel-Code 51 (das Includefile ist zeitlich **vor** der Stringdeklaration für `Titel.$` eingefügt)

```
INCLUDEFILE 'DEF_eiger_Colors.INC'   ; Farbdefinitionen 12.03.2006
INCLUDEFILE 'DEF_eiger_Types.INC'    ; eiger Definitionen

INCLUDEFILE 'CHSS_Global.EVS' ; Includefile für Schachbrett-Projekt


; Definitionen für Startview -------------------------------------

STRING [15] Label_FotoButton.$ = 'Fotograf:'
STRING [15] Label_Name.$ = 'Christoph Angst'
STRING [15] Label_Button_View2.$ = 'Schachbrett >>'
STRING [60] Titel.$ = 'Projekt Schachbrett, CHSS_001.EVS: Startview'
```

Wir können dieses Problem lösen, indem wir das Includefile erst einführen, wenn die String-Variablen deklariert, d.h. für die weiteren Programmschritte bekannt sind (Beispiel-Code 52).

Beispiel-Code 52 (Startview: Includefile ist zeitlich **nach** der Stringdeklaration für `Titel.$` eingefügt)

```
; Definitionen für Startview -------------------------------------

STRING [15] Label_FotoButton.$ = 'Fotograf:'
STRING [15] Label_Name.$ = 'Christoph Angst'
STRING [15] Label_Button_View2.$ = 'Schachbrett >>'
STRING [60] Titel.$ = 'Projekt Schachbrett, CHSS_001.EVS: Startview'

INCLUDEFILE 'CHSS_Global.EVS' ; Includefile für Schachbrett-Projekt
```

Die gleiche Korrektur führen wir auch bei der View 2 durch.

Beispiel-Code 53 (View 2: Includefile wird zeitlich **nach** der Stringdeklaration für `Titel.$` eingefügt)

```
; Definitionen für View 2 ----------------------------------------

STRING [60] Titel.$ = 'Projekt Schachbrett, CHSS_002.EVS: View 2'

INCLUDEFILE 'CHSS_Global.EVS' ; Includefile für Schachbrett-Projekt
```

Nun können wir unsere Views problemlos kompilieren und auch der Test auf dem eigerPanel verläuft einwandfrei.

Zentral verwalteter Hotspot

Für die Entwicklung unseres Projektes ist der HotSpot mit der Screenshot-Funktion ein nützliches Werkzeug. Diesen Service möchten wir ebenfalls auf jeder View nutzen können – ein typischer Fall für das Includefile. Die Methode `HotSpot.Install()` selbst können wir nicht auslagern, weil sie Teil des Hauptprogramms ist (Beispiel-Code 54). Auch die Subroutine `ScreenDump` sollten wir im Programmcode der View belassen, weil der Dateiname des Screenshots von der betreffenden View abhängt. Hingegen können wir die Geometrie des HotSpots zentral verwalten. Anstelle von konkreten Werten (vgl. Beispiel-Code 54) setzen wir im Hauptprogramm der Views Konstanten ein (Beispiel-Code 93), die wir dann im Includefile deklarieren und dort gegebenenfalls für alle Views auf einmal ändern können.

Konstantennamen für den Screenshot-Hotspot:

- ScreenshotX = 600
- ScreenshotY = 0
- ScreenshotW = 40
- ScreenshotH = 40

Beispiel-Code 54 (bisher: HotSpot für Screenshot im Hauptprogramm der Startview und View 2)

```
; HotSpot für PrintScreen rechts oben
  Load.Geometry_XYWH(600,0,40,40)   ; Position & Dimension des HotSpots
  HotSpot.Install(NIL,NIL,ScreenDump,NIL)
```

Beispiel-Code 55 (neu: Konstanten für die Geometrie des Screenshot-HotSpot, deklariert im Includefile)

```
; Hotspot für PrintScreen (Konstanten in CHSS_Global.EVS deklariert)
  Load.Geometry_XYWH(ScreenshotX,ScreenshotY,ScreenshotW,ScreenshotH)
  ; Position & Dimension des HotSpots
  HotSpot.Install(NIL,NIL,ScreenDump,NIL)
```

Nachdem wir diese Konstantennamen gemäss Beispiel-Code 55 in der Startview und in der View 2 eingeführt und diese Änderungen gespeichert haben,

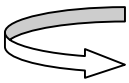
deklarieren wir die Konstanten im Includefile (Beispiel-Code 56). Das tun wir direkt unterhalb der bereits vorhandenen Stringdeklaration.

Beispiel-Code 56 (Deklaration der Konstanten für die Geometrie des Screenshot-HotSpots)

```
; Geometrie für Screenshot-HotSpot-----------------------------
CONST ScreenshotX = 600
CONST ScreenshotY = 0
CONST ScreenshotW = 40
CONST ScreenshotH = 40
```

U010S001.EGI
U010S002.EGI

Wir kompilieren die Views und testen den HotSpot auf dem FOX. Nach dem Reset drücken wir zuerst in der Startview auf den HotSpot und dann auch auf der View 2. In der Folge sollten auf der Compact FlashCard im Ordner CHSS\DUMP die neuen Screenshots CHSS_001.EGI und CHSS_002.EGI liegen.



Aufgabe 3: Von der View 2 gibt es bisher keine Möglichkeit zur Startview zurückzukehren. Installieren Sie in der View 2 einen Back-Button, indem Sie den Code des View2-Buttons von der Startview in die View 2 kopieren und diesen entsprechend anpassen. Lagern Sie wo möglich allfällige Konstanten- und Variablendeklarationen sowie Subroutinen, die aufgrund der Neuerungen in beiden Views identisch sind, in das Includefile aus (vgl. Übung 9, S.78ff und 81ff). Einen Lösungsweg finden Sie im Anhang dieses Schnelleinstiegs (S.126). Zeitaufwand: 10 Minuten.

Übung 11: Schachbrett zeichnen

In dieser Übung lernen Sie

- ✓ Rechtecke zeichnen.
- ✓ den Nullpunkt auf der View verschieben.
- ✓ wiederholte Anweisungen als `FOR/TO/NEXT`-Schleifen programmieren.

Zeitbedarf: 55 Minuten (inkl. 10 Minuten für Aufgabe 4)

Daten für die Übung: Projektordner „Schachbrett“ mit den Dateien CHSS_001.EVS, CHSS_002.EVS und CHSS_Global.EVS aus der Übung 10, den Sie selbst erstellt haben oder vorgefertigt im Ordner „Schnelleinstieg/Übung10“ vorfinden.

Unbeschriftetes Rechteck zeichnen

Auf der View 2 wollen wir Rechtecke zeichnen, die schlussendlich ein Schachbrett darstellen.

Zunächst zeichnen wir ein einzelnes weisses Quadrat. Der Zeichenbefehl von eiger Script für ein Rechteck lautet `Draw.RectangleFilled()`. Diese Methode verpacken wir in eine Subroutine `SUB Draw_Chessboard` (Beispiel-Code 57). Die Farbe für das Rechteck ist bzw. wird im Register `eI.FillColor` festgelegt.

Beispiel-Code 57 (Subroutine zum Zeichnen eines weissen Rechtecks in der View2)

```
SUB Draw_Chessboard
eI.FillColor := white
Draw.RectangleFilled()
ENDSUB
```

Den Einsatz der neuen Subroutine steuern wir vom Hauptprogramm aus (Beispiel-Code 58). Dort legen wir auch die Position und Geometrie des Rechtecks fest, gemäss Abbildung 46.

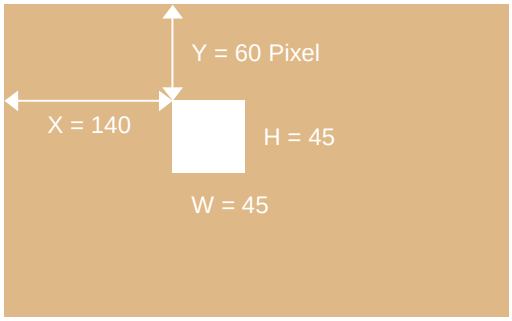


Abbildung 46: Position und Geometrie für das Rechteck auf dem FOX-Display.

Beispiel-Code 58 (Geometrie und Aufruf im Hauptprogramm für die Subroutine `Draw_Chessboard`)

```
BEGINVIEW
  EVE.Init()                          ; EVE ANNA initialisieren
  eI.DisplayColor := burlywood
  Display.Clear()                     ; Bildschirm löschen
  Display.Prepare()
  CallSubroutine(Draw_Titel)
  CallSubroutine(Draw_Button_ChangeView)

  Load.Geometry_XYWH(140,60,45,45)    ;Pos. & Geom. für Rechteck
  CallSubroutine(Draw_Chessboard)     ; Aufruf zum Zeichnen
```


U011S001.EGI

Nach Kompilieren und Übertragen des Codes auf den FOX wird auf der View 2 das weisse Quadrat gezeichnet (Abbildung 47).

Abbildung 47: Weisses Rechteck in der View2.

Um für die weiteren Schritte etwas flexibler zu sein, definieren wir die Lage des Rechtecks über die Konstanten `Chessboard_X` und `Chessboard_Y` und weisen diesen konkrete Pixelwerte zu (Beispiel-Code 59). Auch für die Angabe der Breite und der Höhe des einzelnen Quadrats bzw. Rechtecks benutzen wir Konstanten, nämlich `Field_W` und `Field_H` (Beispiel-Code 59). Die Konstanten deklarieren wir im oberen Teil der View 2 unterhalb der Stringdeklaration.

Beispiel-Code 59 (Deklaration der Schachbrett-Konstanten im oberen Teil der View 2)

```
; Definitionen für View 2 -------------------------------------

STRING   [60] Titel.$ = 'Projekt Schachbrett, CHSS_002.EVS: View 2'
STRING   [15] Label_Button_ChangeView.$ = '<< Back'

CONST    Chessboard_X   = 140
CONST    Chessboard_Y   = 60
CONST    Field_W        = 45
CONST    Field_H        = 45
```

Nun können wir diese Konstanten im Hauptprogramm für die Methode `Load.Geometry_XYWH()` verwenden, die als Grundlage für das Zeichnen der Rechtecke bzw. der Quadrate dient (Beispiel-Code 60). Die Methode `Load.Geometry_XYWH` füllt die Register `eI.Pos_X1`, `eI.Pos_Y1`, `eI.Width` und `eI.Height` mit den Werten dieser Konstanten.

Beispiel-Code 60 (in `Load.Geometry_XYWH()` ersetzen wir die Pixelwerte mit **Konstanten**)

```
BEGINVIEW
  EVE.Init()                          ; EVE ANNA initialisieren
  eI.DisplayColor := burlywood
  Display.Clear()                     ; Bildschirm löschen
  Display.Prepare()
  CallSubroutine(Draw_Titel)
  CallSubroutine(Draw_Button_ChangeView)

  Load.Geometry_XYWH(Chessboard_X,Chessboard_Y,Field_W,Field_H)
                                      ;Pos. & Geom. für Rechteck
  CallSubroutine(Draw_Chessboard)     ; Aufruf zum Zeichnen
```

Es sollte aber in der View 2 nicht bei einem einzigen Quadrat bleiben. Das Ziel ist ein Schachbrett mit 8x8 Quadraten, abwechslungsweise in weiss und schwarz. Dazu könnten wir den Code, den wir im Hauptprogramm neu eingefügt haben, 63 mal kopieren und so – mit entsprechender individueller Positionszu-

weisung – insgesamt 64 einzelne Quadrate aneinander reihen. Diese Lösung wäre allerdings nicht sehr elegant und würde zudem auch das Programmscript unübersichtlich machen.

Für eine schlanke Lösung kombinieren wir im Folgenden zwei recht nützliche eigerScript-Methoden:

- `Transfer.Offset_from_Pos_X1Y1()`
- Schleife `FOR/TO/NEXT`

Nullpunkt verschieben mit Transfer.Offset

Die eigerScript-Methode `Transfer.Offset_from_Pos_X1Y1()` verschiebt den Nullpunkt des Display-Koordinatensystems auf die Position, welche durch die Registereinträge `eI.Pos_X1` und `eI.Pos_Y1` vorgegeben ist (vgl. Abbildung 48. Alle weiteren Positionsangaben beziehen sich dann auf diesen neuen Nullpunkt.

Abbildung 48: Mit `Transfer.Offset_from_Pos_X1Y1()` wird der Nullpunkt der View und damit alle anderen Viewkoordinaten verschoben.

In unserem Fall legen wir fest, dass der neue Nullpunkt der View mit der linken oberen Ecke des weissen Rechtecks und damit auch des Schachbretts zusammenfallen soll (vgl. Abbildung 48). Zu diesem Zweck fügen wir in der Subroutine `SUB Draw_Chessboard` die Methode `Transfer.Offset_from_Pos_X1Y1()` ein (Beispiel-Code 61).

Beispiel-Code 61 (Die Methode Transfer.Offset verschiebt den Nullpunkt der View)

```
SUB Draw_Chessboard
    Transfer.Offset_from_Pos_X1Y1()

    eI.FillColor := white
    Draw.RectangleFilled()
ENDSUB
```

Die Auswirkung dieser Methode testen wir auf dem FOX. `Transfer.Offset()` hat sich der beiden Register `eI.Pos_X1` und `eI.Pos_Y1` bedient, denen zuvor im Hauptprogramm mit der Methode `Load.Geometry_XYWH()` die Werte der Konstanten `Chessboard_X` und `Chessboard_Y` zugewiesen worden sind. Ausgehend vom neuen Nullpunkt hat dann `Draw.RectangleFilled()` die gleichen Positionsregister benutzt um mit Referenz auf den neuen Nullpunkt die Lage für das weisse Quadrat zu bestimmen. Deshalb liegt das Quadrat nun in doppelter Entfernung zur linken oberen Ecke der View 2 (vgl. Abbildung 49).


U011S002.EGI

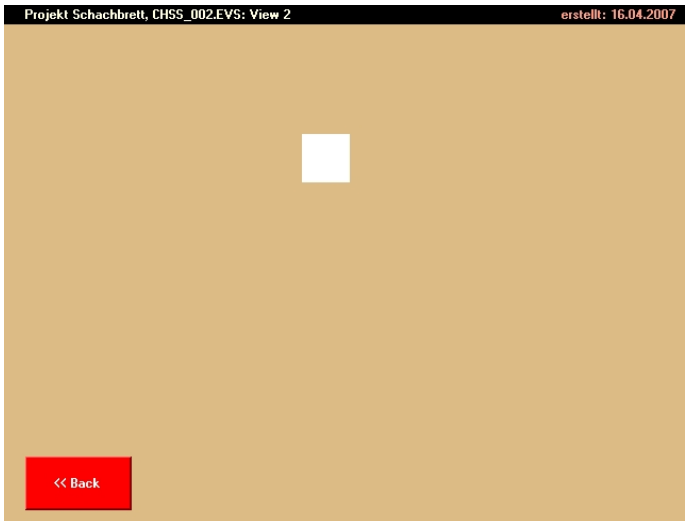


Abbildung 49: Das Quadrat hat gegenüber dem Neuen Nullpunkt die Koordinaten X1 und Y1. Um diese Werte ist auch der Nullpunkt von der linken oberen View-Ecke transferiert.

Um das weisse Quadrat beim neuen Nullpunkt anzusetzen, weisen wir mit der Methode `Load.Pos_X1Y1()` den Registern `eI.Pos_X1` und `eI.Pos_Y1` die Werte X1 = 0 und Y1 = 0 zu (Beispiel-Code 62). Diese Koordinaten beziehen sich nun auf den transferierten Nullpunkt.

Beispiel-Code 62 (Subroutine zum Zeichnen des weissen Quadrats beim transferierten Nullpunkt)

```
SUB Draw_Chessboard
    Transfer.Offset_from_Pos_X1Y1()
```

```
    Load.Pos_X1Y1(0,0)
    eI.FillColor := white
    Draw.RectangleFilled()
ENDSUB
```

Im Folgenden fügen wir dem weissen Quadrat ein schwarzes an. Hierzu duplizieren wir einen Teil der bisherigen `SUB Draw_Chessboard` und weisen dem Register `eI.FillColor` die Füllfarbe Schwarz zu. Das schwarze Quadrat ist um die Breite (`eI.Width` = `Field_W`) des weissen Quadrats in X-Richtung versetzt. Die Y-Koordinate bleibt gleich. Dieser Sachverhalt schreibt sich in eigerScript wie in Beispiel-Code 63 gezeigt.

Beispiel-Code 63 (Subroutine zum Zeichnen des weissen und des schwarzen Quadrats)

```
SUB Draw_Chessboard
    Transfer.Offset_from_Pos_X1Y1()
    Load.Pos_X1Y1(0,0)

; erstes Quadrat (weiss)  ------------------
    eI.FillColor := white
    Draw.RectangleFilled()

; zweites Quadrat (schwarz)  ------------------
    eI.Pos_X1 := eI.Pos_X1 + eI.Width
    eI.FillColor := black
    Draw.RectangleFilled()
ENDSUB
```


U011S003.EGI

Beim Test auf dem FOX erscheinen in der View 2 die beiden gewünschten Quadrate in den gewünschten Positionen (Abbildung 50).

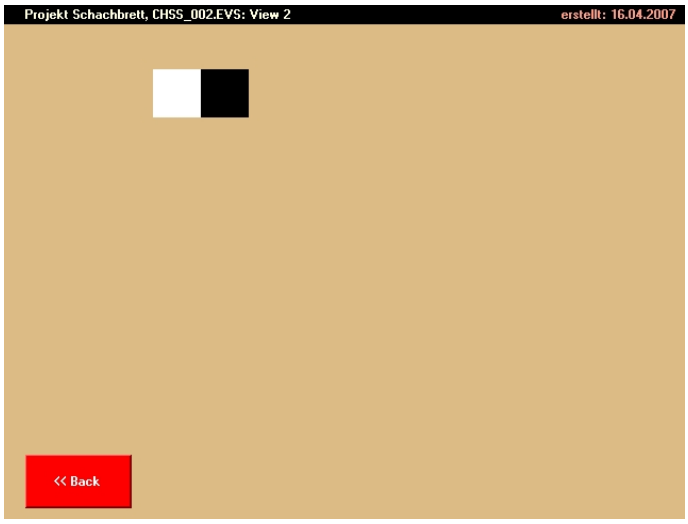


Abbildung 50: Weisses und schwarzes Quadrat nebeneinander, beginnend im transferierten Nullpunkt der View 2.

Damit haben wir den Grundbaustein für unser Schachbrett konstruiert. Dieses Quadratenpaar soll der FOX nun so oft nebeneinander und untereinander anfügen, bis das Schachbrett komplett ist.

Schleife FOR/TO/NEXT

Im Falle wiederholter Programmabläufe erspart uns die `FOR/TO/NEXT`-Schleife das mehrfache Aneinanderfügen derselben Anweisungen. Damit veranlassen wir das Programm, eine Sequenz von Anweisungen mehrere Male zu wiederholen. Wir können auch eine Zählervariable festlegen, die sich nach jedem Umgang um beispielsweise 1 erhöht. Die Schleife wird so oft durchlaufen bis die Zählervariable einen vorgegebenen Wert erreicht hat. Diese Wiederholungsanweisung funktioniert nach dem Schema in Beispiel-Code 64. Das `NEXT` verweist wieder an den Anfang der Schleife. Wenn die Bedingung `FOR/TO` (d.h. `i := b`) erfüllt ist, werden die Anweisungen zum letzten Mal ausgeführt. Danach wird die Schleife verlassen. Die Schleife wird nur durchlaufen, wenn a kleiner oder gleich b ist. Für den Fall, dass a und b identisch sind, wird die Schleife genau ein Mal durchlaufen.

Beispiel-Code 64 (Schema einer `FOR/TO/NEXT`-Schleife)

```
FOR i := a TO b
    <Anweisungen>
NEXT
```

Das Quadratenpaar kommt in der ersten Reihe vier mal vor, d.h. wir müssen den gleichen Code 3x wiederholen. Bei jeder Wiederholung muss die Zählervariable um 1 grösser werden. Auch der X-Wert für die nächsten zwei Quadrate muss jeweils um die doppelte Quadratseite (2 x `Field_W` = 2 x `eI.Width`) zunehmen (vgl. Tabelle 10).

Tabelle 10: Verhalten von Zählervariable und X-Wert, während der Schleifenbefehl `FOR/TO/NEXT` abgearbeitet wird.

	Erster Durchlauf	Erste Wiederholung	Zweite Wiederholung	Dritte Wiederholung
Wert der Zählervariable	0	1	2	3
X-Wert (= `eI.Pos_X1`)	0	2 x `eI.Width`	4 x `eI.Width`	6 x `eI.Width`

In unserem Fall stellt die Zählervariable eine Doppelspalte dar. Vier Doppelspalten ergeben die Breite des Schachbretts. Die Zählervariable nennen wir „`DoubleRowNr.I`“ und deklarieren sie als Integervariable im Anschluss an die bereits bestehenden Deklarationen der View 2 (Beispiel-Code 65). Damit ist für diese Variable ein Speicherplatz von 2 Bytes reserviert (vgl. Tabelle 11, S.116). Mit der Endung „`.I`“ deuten wir an, dass es sich bei `DoubleRowNr.I` um eine Integervariable handelt. Der Variablen müssen wir nicht sofort einen Wert zuweisen; das kann auch erst in der betreffenden Subroutine geschehen.

Beispiel-Code 65 (Deklaration einer Zählervariable vom Datentyp Byte in der View 2)

```
; Definitionen für View 2 -------------------------------------
STRING     [60] Titel.$ = 'Projekt Schachbrett, CHSS_002.EVS: View
2'
STRING [15] Label_Button_ChangeView.$ = '<< Back'

CONST      Chessboard_X = 140
CONST      Chessboard_Y = 60
CONST      Field_W      = 45
CONST      Field_H      = 45

INTEGER DoubleRowNr.I
```

In unserem Fall benutzen wir in der Subroutine `Draw_Chessboard` (vgl. Beispiel-Code 63) das `FOR/TO/NEXT` in der Weise, dass die Anweisungen innerhalb der Schleife dreimal wiederholt werden. Mit andern Worten: In der Variablen `DoubleRowNr` zählen wir von 0 bis 3 (vgl. Tabelle 10). Damit bei der Wiederholung das nächste weisse Quadrat nicht das zuvor gezeichnete schwarze Quadrat überschreibt, muss am Schluss der Schleife vorbereitend die X-Position um die Breite des Quadrats nach rechts verschoben werden. Wie wir diese Überlegungen in eigerScript umsetzen, zeigt Beispiel-Code 66.

Beispiel-Code 66 (`FOR/TO/NEXT`-Schleife für die erste Linie des Schachbretts)

```
SUB Draw_Chessboard
    Transfer.Offset_from_Pos_X1Y1()
    Load.Pos_X1Y1(0,0)

    FOR DoubleRowNr.I := 0 TO 3
```

```
            ; erstes Quadrat (weiss)  -----------------
            eI.FillColor := white
            Draw.RectangleFilled()

            ; zweites Quadrat (schwarz)  ---------------
            eI.Pos_X1 := eI.Pos_X1 + eI.Width
            eI.FillColor := black
            Draw.RectangleFilled()

            eI.Pos_X1 := eI.Pos_X1 + eI.Width
    NEXT
ENDSUB
```


U011S004.EGI

Dank dieser Ergänzungen sollte der FOX nach Kompilieren und Übertragen der Views die erste Linie des Schachbretts zeichnen (vgl. Abbildung 51).

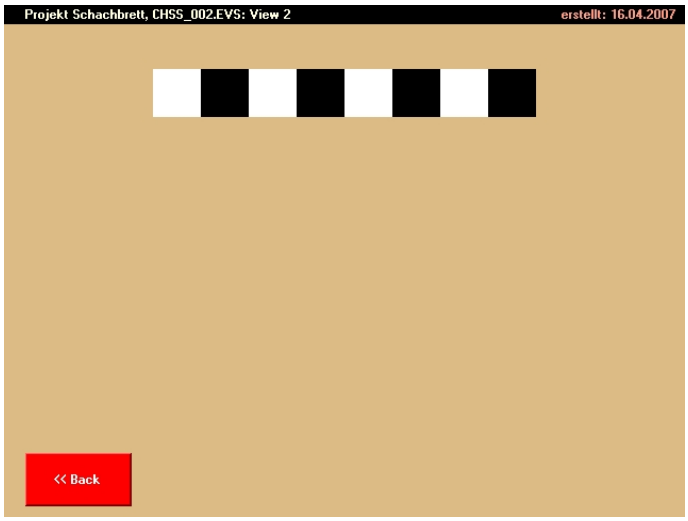


Abbildung 51: Erste Linie des Schachbretts, erstellt mit der `FOR/TO/NEXT`-Schleife in der View 2.

Wir könnten auch die Schleife ohne Zähler anwenden, da in unserem Beispiel ein Zugriff auf die Zählervariable nicht notwendig und auch nicht vorgesehen ist. Statt `FOR DoubleRowNr.I := 0 TO 3` müssten wir also nur schreiben: `FOR 0 TO 3`. Die zählerlose Schleife ist geringfügig schneller als die Variante mit Zähler.
Probieren Sie es aus, ob mit dieser zählerlosen Variante das gleiche Bild dargestellt wird.

Als nächstes wollen wir diese Linie sieben Mal unter der ersten anfügen, damit ein komplettes Schachbrett entsteht. Das ist wiederum ein „Schleifen-Problem“, denn die oben programmierte `FOR/TO/NEXT`-Schleife muss 7 mal wiederholt werden. Mit andern Worten: Die erste `FOR/TO/NEXT`-Schleife muss in eine übergeordnete `FOR/TO/NEXT`-Schleife „verschachtelt“ werden.

Für die übergeordnete Schleife deklarieren wir eine weitere Zählervariable mit dem Namen „`LineNr.I`“ (Beispiel-Code 67).

Beispiel-Code 67 (Zweite Zählervariable für die äussere `FOR/TO/NEXT`-Schleife)

```
INTEGER DoubleRowNr.I
INTEGER LineNr.I
```

Die neue `FOR/TO/NEXT`-Schleife bauen wir um die bisherige herum. Damit die Linien des Schachbretts untereinander angefügt werden, müssen wir für jede neue Linie die X-Position wieder auf Null setzen und die Y-Position um die Höhe eines Quadrats nach unten verschieben. Daraus ergibt sich

Beispiel-Code 68 (Verschachtelte `FOR/TO/NEXT`-Schleifen)

```
SUB Draw_Chessboard
    Transfer.Offset_from_Pos_X1Y1()
    Load.Pos_X1Y1(0,0)

    FOR LineNr.I := 0 TO 7
          FOR DoubleRowNr.I := 0 TO 3

                ; erstes Quadrat (weiss)  ------------------
                eI.FillColor := white
                Draw.RectangleFilled()

                ; zweites Quadrat (schwarz)  ---------------
                eI.Pos_X1 := eI.Pos_X1 + eI.Width
                eI.FillColor := black
                Draw.RectangleFilled()

                eI.Pos_X1 := eI.Pos_X1 + eI.Width
          NEXT

          eI.Pos_Y1 := eI.Pos_Y1 + eI.Height
          eI.Pos_X1 := 0
    NEXT
ENDSUB
```


U011S005.EGI

Die Auswirkungen dieser Erweiterungen überprüfen wir auf dem FOX. Das Ergebnis entspricht noch nicht unseren Vorstellungen (Abbildung 52). Offensichtlich beginnt dass Programm jede Linie immer mit einem weissen Quadrat.

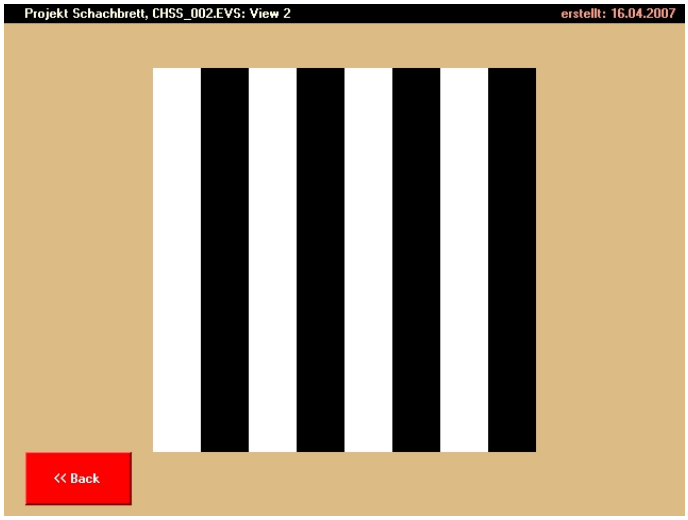


Abbildung 52: Statt eines Schachbretts mit gegeneinander versetzten weissen und schwarzen Quadraten ergibt Beispiel-Code 68 acht Säulen.

Für den Austausch eines Wertes zwischen zwei Variablen braucht es eine Hilfsvariable.

Das Programm sollte bei jedem Durchlauf der äusseren Schleife die Farbe des ersten Quadrats wechseln. Auch diese Aufgabe lösen wir am besten mit Hilfe geeigneter Variablen, die während des Programmablaufs ihren Wert – z.B. die Farbe – ändern können. Die neuen Variablen deklarieren wir als „`Color1.I`“ und „`Color2.I`“ (Beispiel-Code 69). Weshalb wir hier auch noch eine dritte Variable „`Color3.I`“ einführen, wird in den folgenden Schritten erklärt.

Beispiel-Code 69 (Deklaration von Farbvariablen in View 2)

```
INTEGER DoubleRowNr.I
INTEGER LineNr.I
INTEGER Color1.I
INTEGER Color2.I
INTEGER Color3.I
```

Am Anfang der Subroutine `Draw_Chessboard` weisen wir den Variablen `Color1.I` und `Color2.I` die Werte `white` und `black` zu. Dabei handelt es sich um Konstanten, die eigerScript aus der Header-Datei „DEF_eiger_Colors.INC“ als Integer-Zahlen bekannt sind. Deswegen konnten wir diese Variablen oben als Integer-Variablen deklarieren.

Nebst der Wertezuweisung müssen wir dort, wo bisher die Farbkonstante direkt angegeben war, die entsprechende Farbvariable angeben.

Beispiel-Code 70 (Die Farbkonstanten sind durch die zuvor deklarierten Farbvariablen **ersetzt**)

```
SUB Draw_Chessboard
    Transfer.Offset_from_Pos_X1Y1()
    Load.Pos_X1Y1(0,0)
```

```
    Color1.I := white
    Color2.I := black

    FOR LineNr := 0 TO 7
          FOR DoubleRowNr := 0 TO 3

                ; erstes Quadrat ------------------
                eI.FillColor := Color1.I
                Draw.RectangleFilled()

                ; zweites Quadrat -----------------
                eI.Pos_X1 := eI.Pos_X1 + eI.Width
                eI.FillColor := Color2.I
                Draw.RectangleFilled()
```

Nun müssen die Farbvariablen jeweils am Ende des Durchlaufs der äussern Schleife ihren Wert – d.h. ihre Farbe – austauschen. Das geht am besten über eine Hilfsvariable, die wir bereits als `Color3.I` deklariert haben.

Beispiel-Code 71 (Zweiter Teil von `SUB Draw_Chessboard`. Mit Hilfe der Variablen `Color3` werden die Farbwerte in den Variablen `Color1` und `Color2` vertauscht)

```
                eI.Pos_X1 := eI.Pos_X1 + eI.Width
          NEXT

          eI.Pos_Y1 := eI.Pos_Y1 + eI.Height
          eI.Pos_X1 := 0

          Color3.I := Color1.I
          Color1.I := Color2.I
          Color2.I := Color3.I
    NEXT
ENDSUB
```

U011S006.EGI

Mit diesen Ergänzungen zeichnet uns der FOX ein wunderschönes schwarz/weisses Schachbrett in die View 2 (vgl. Abbildung 53).

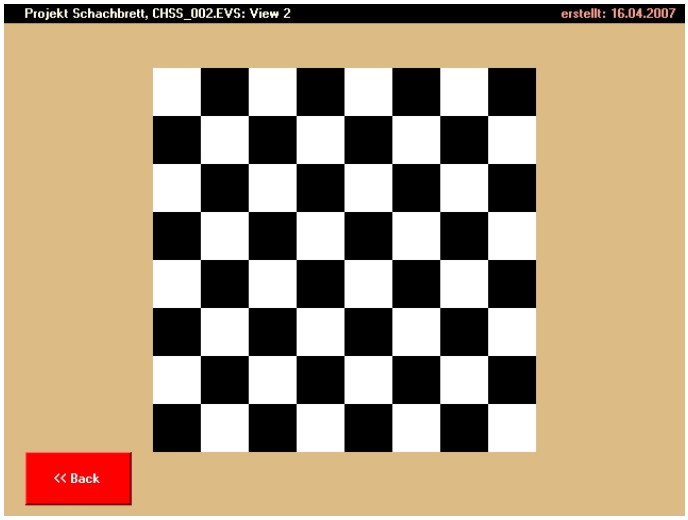


Abbildung 53: Komplette Darstellung eines Schachbretts in der View 2.

Eine kleine Spielerei

Der Aufruf zum Zeichnen des Schachbretts geht vom Hauptprogramm aus (vgl. Beispiel-Code 60, S.95). Wenn wir diesen Aufruf kopieren und ein zweites Mal im Hauptprogramm einfügen, zeichnet der FOX das Schachbrett zweimal. Wir können dazu beim zweiten Aufruf die Werte für das zweite Schachbrett verändern – nach Belieben auch mit unterschiedlicher Breite und Höhe (z.B. Beispiel-Code 72) – und dadurch auf der View 2 zwei verschiedene Schachbretter darstellen (vgl. Abbildung 54).

U011S007.EGI

Beispiel-Code 72 (Hauptprogramm mit einem zweiten Aufruf zum Zeichnen eines Schachbretts)

```
BEGINVIEW
   EVE.Init()                          ; EVE ANNA initialisieren
   eI.DisplayColor := burlywood
   Display.Clear()                     ; Bildschirm löschen
   Display.Prepare()
   CallSubroutine(Draw_Titel)
   CallSubroutine(Draw_Button_ChangeView)

   Load.Geometry_XYWH(Chessboard_X,Chessboard_Y,Field_W,Field_H)
                                       ;Pos. & Geom. für Rechteck
   CallSubroutine(Draw_Chessboard)     ; Aufruf zum Zeichnen

Aufruf für ein zweites Schachbrett ------------------------
   Load.Geometry_XYWH(470,320,10,15)   ;Pos. & Geom. für Rechteck
   CallSubroutine(Draw_Chessboard)     ; Aufruf zum Zeichnen
```

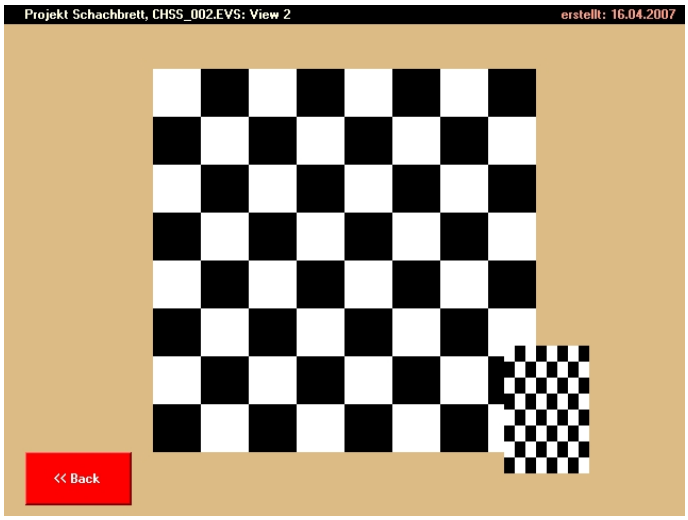


Abbildung 54: Zwei „Schachbretter“ in der View 2 aufgrund von Beispiel-Code 72.

Aufgabe 4: Richten Sie in der View 2 einen zusätzlichen Button mit Hotspot ein, der das kleine Schachbrett einblendet. (vgl. Übung 7, S.56ff). Einen Lösungsweg finden Sie im Anhang dieses Schnelleinstiegs (S.128). Zeitaufwand: 10 Minuten.

Übung 12: Timer

In dieser Übung lernen Sie

- ✓ einen Timer als Countdown programmieren (Single Timer).
- ✓ einen Timer für einen Sekundenzähler benützen.

Zeitbedarf: 40 Minuten (inkl. 7 Minuten für Aufgabe 5)

Daten für die Übung: Projektordner „Schachbrett“ mit den Dateien CHSS_001.EVS und CHSS_002.EVS aus der Übung 11, den Sie selbst erstellt haben oder vorgefertigt im Ordner „Schnelleinstieg/Übung11“ vorfinden.

Timer sind in einem ereignisgesteuerten System eine der wichtigsten Quellen für Events. Die virtuelle Maschine EVE verfügt über 8 Timer, die unabhängig voneinander benutzt werden können.

eigerScript unterscheidet zwischen einfachem und kontinuierlichem Timer. Der einfache oder „Single“ Timer stoppt, nachdem die programmierte Zeit abgelaufen ist, während der kontinuierliche oder „Continuous“ Timer den Countdown automatisch wieder von neuem beginnt.

Single Timer

In dieser Übung wollen wir unser Schachbrett-Projekt mit einem Single Timer ausrüsten, der nach einer bestimmten Zeit den Sprung von der View 2 zurück zur Startview veranlasst. Den Single Timer installieren wir in der View 2. Sobald die View 2 aufgerufen ist, soll er automatisch aktiviert werden.

Um das Hauptprogramm möglichst schlank zu halten, legen wir für den Timer eine spezielle Subroutine `SUB TimerToStartview` an. Diese Subroutine rufen wir beim Öffnen der View 2 vom Hauptprogramm aus auf (Beispiel-Code 73).

Beispiel-Code 73 (Neuer Befehl im Hauptprogramm zum Aufruf der Subroutine `TimerToStartview`)

```
BEGINVIEW
    EVE.Init()                      ; EVE ANNA initialisieren
    eI.DisplayColor := burlywood
    Display.Clear()                 ; Bildschirm löschen
    Display.Prepare()
    CallSubroutine(Draw_Titel)
    CallSubroutine(Draw_Button_ChangeView)
    CallSubroutine(Draw_Button_SmallChessboard)
    CallSubroutine(TimerToStartview)
```

Wie ein HotSpot oder ein HotKey muss auch der Timer zuerst in der View installiert werden. Dazu dient der Befehl `Timer.InstallLocal()`, den wir in die neue Subroutine schreiben (Beispiel-Code 74). Dem Befehl fügen wir in Klammern die Nummer des Timers (0 bis 7) bei, sowie den Namen der Subroutine, welche nach Ablauf des Countdowns aufgerufen werden soll. Wir wählen den Timer Nr. 0 und die Subroutine `ToStartview`.

Beispiel-Code 74 (Installation des Timers in einer neuen Subroutine der View 2)

```
SUB TimerToStartview
    Timer.InstallLocal(0,ToStartview)

ENDSUB
```

Als zweites muss der Timer mit der Methode `Timer.Load()` geladen werden. Auch hier wird wieder die Nummer des betreffenden Timers verlangt sowie die Zeitdauer in Millisekunden [ms], beispielsweise 7000 ms = 7 Sekunden (Beispiel-Code 75). Dieser Wert wird ins RELOAD-Register des betreffenden Timers geschrieben.

Beispiel-Code 75 (Im Timer 0 wird die Zeitspanne von 7000 Millisekunden geladen)

```
SUB TimerToStartview
    Timer.InstallLocal(0,ToStartview)
    Timer.Load(0,7000)
```

Nun muss der Timer 0 noch als Single Timer gestartet werden. Dafür sorgt eine dritte Befehlszeile (Beispiel-Code 76). Mit diesem Befehl wird der TimerCounter aus dem Reload-Register geladen, der ExpiredCounter auf Null gestellt und der

Timer für einen Zyklus geladen. Nach Ablauf des Zyklus springt der FOX zurück zur Startview.

Beispiel-Code 76 (Mit der dritten Befehlszeile wird der Timer 0 als Single Timer gestartet)

```
SUB TimerToStartview
    Timer.InstallLocal(0,ToStartview)
    Timer.Load(0,7000)
    Timer.StartSingle(0)
ENDSUB
```

Nun fehlt nur noch der obligate letzte Schritt: Der Timer kann nur Aktionen auslösen, wenn die Timertabelle freigegeben ist. Auch darin sind sich HotSpot, HotKey und Timer gleich. Die Methode zur Freigabe der Timertabelle können wir am Schluss unserer Subroutine plazieren oder im Hauptprogramm. Wir tun es am Ende des Hauptprogramms (Beispiel-Code 77).

Beispiel-Code 77 (Mit `Timer.TableEnable` geben wir die Timertabelle(n) frei, z.B. im Hauptprogramm)

```
    Display.Show()

    Timer.TableEnable()
    HotSpot.TableEnable()
    HotKey.EnableLocalKeys()
    HotKey.TableEnable()

    LOOP
    ENDLOOP

ENDVIEW
```

Testen wir unsere Neuerungen auf dem FOX, so stellen wir fest, dass die View 2 nach dem Öffnen während 7 Sekunden angezeigt wird und danach wieder die Startview erscheint.

In unserer View 2 ist es unerheblich, an welcher Stelle wir den Timer mit `Timer.StartSingle()` starten, bzw. mit `Timer.TableEnable()` für Aktionen freigeben. In komplexeren Views mit zeitrelevanten Funktionsabläufen sollte jedoch darauf geachtet werden, dass der Timer nicht wegen falscher Befehlsplazierung schon halb abgelaufen ist, wenn er eigentlich erst starten sollte.

Timer stoppen

Das automatische Zurückspringen in die Startview ist vielleicht nicht unbedingt erwünscht. Es sollte deshalb noch eine Möglichkeit geben, diese Funktion, d.h. den ablaufenden Timer zu stoppen. Hierzu bedienen wir uns der Methode `Timer.Stop(VarInt:Timer)` und verpacken sie in eine neue Subroutine `SUB Stop_TimerToStartview` (Beispiel-Code 78).

Beispiel-Code 78 (Subroutine in View 2 zum Anhalten des Timers, der zurück zur Startview führt)

```
SUB Stop_TimerToStartview
    Timer.Stop(0)
ENDSUB
```

Die Stop-Methode soll durch einen Button und HotSpot aktiviert werden können.

Den Button zeichnen wir als Duplikat des Back-Buttons, dessen Subroutine `Draw_Button_ChangeView` wir aus dem Includefile CHSS_Global.EVS ins Script der View 2 kopieren. Diese Subroutine verändern wir so, dass in der View 2 direkt oberhalb des Back-Buttons ein Stay-Button gezeichnet wird.

Beispiel-Code 79 (Douplikat der `SUB Draw_Button_ChangeView` , **angepasst** für den Stay-Button)

```
SUB Draw_StayButton
    eI.Pos_X1 := 20
    eI.Pos_Y1 := 350
    Fill.LabelParameter(Button_up_Style)
    Label.Text(Label_StayButton.$)
ENDSUB
```

Die neue Variable `Label_StayButton.$` zur Beschriftung des Stay-Buttons deklarieren wir bei den anderen Stringdeklarationen der View 2. Wir weisen ihr den Text '> Stay <' zu (vgl. Beispiel-Code 80).

Beispiel-Code 80 (Deklaration der Stringvariablen für die Beschriftung des Stay-Buttons)

```
; Definitionen für View 2 ------------------------------------
STRING [60] Titel.$ = 'Projekt Schachbrett, CHSS_002.EVS: View 2'
STRING [15] Label_Button_ChangeView.$ = '<< Back'
STRING [20] Label_SmallChessboard.$ = 'kl.Schachbrett'
STRING [8] Label_StayButton.$ = '> Stay <'
```

Die neue Subroutine wird vom Hauptprogramm aus aufgerufen (vgl. Beispiel-Code 81).

Beispiel-Code 81 (Hauptprogramm mit dem neuen Subroutinenaufruf für den Stay-Button)

```
BEGINVIEW
    EVE.Init()                          ; EVE ANNA initialisieren
    eI.DisplayColor := burlywood
    Display.Clear()                     ; Bildschirm löschen
    Display.Prepare()
    CallSubroutine(Draw_Titel)
    CallSubroutine(Draw_Button_ChangeView)
    CallSubroutine(Draw_Button_SmallChessboard)
    CallSubroutine(TimerToStartview)
    CallSubroutine(Draw_StayButton)
```


U012S001.EGI

Zwischendurch empfiehlt es sich, die Änderungen auf dem FOX zu testen. Das Ergebnis ist in Abbildung 55 zu sehen.

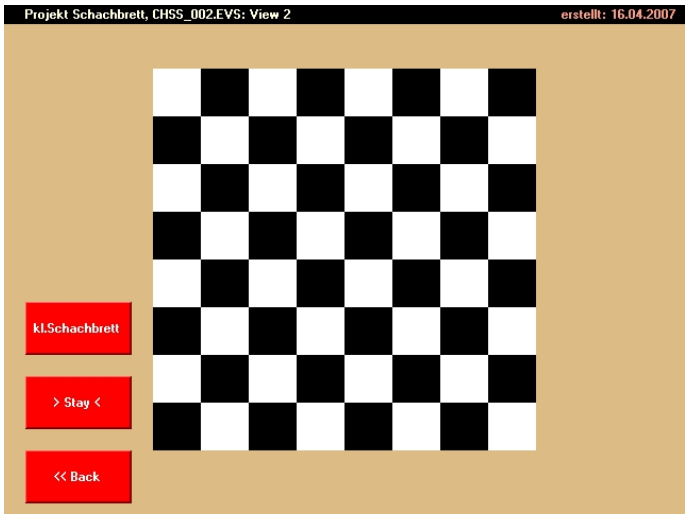


Abbildung 55: Neuer Stay-Button in der View 2.

Nun fehlt noch der HotSpot für den Stay-Button. Als Vorlage bieten sich die Codezeilen des HotSpots an, der dem Back-Button zugrunde liegt. Diese kopieren wir und fügen sie gleich unter halb wieder ein. Die notwendigen Anpassungen sind Beispiel-Code 82 in gezeigt.

Beispiel-Code 82 (kopierter HotSpot mit Anpassungen für den Stay-HotSpot)

```
; HotSpot zum Anhalten des Timers um in View 2 zu bleiben --------
    Load.Geometry_XYWH(20,350,100,50)
    HotSpot.Install(NIL,Draw_StayButton,Stop_TimerToStartview,Draw_S
tayButton)
```

Nun ist der Stay-Button aber noch nicht wirklich animiert. Zwar wird beim Event „Down“ die Subroutine `Stop_TimerToStartview` aufgerufen, aber der Button lässt sich nicht „eindrücken“. Dafür bedarf es noch einiger Ergänzungen in der Subroutine `Stop_TimerToStartview`. Was noch fehlt, kopieren wir einfach aus der Subroutine `SUB Draw_StayButton` und ändern den Style zum `Button_down_Style` (vgl. Beispiel-Code 83).

Beispiel-Code 83 (Subroutine zum Anhalten des Timers, ergänzt mit Animationsstyle für den Stay-Button)

```
SUB Stop_TimerToStartview
    Timer.Stop(0)
    eI.Pos_X1 := 20
    eI.Pos_Y1 := 350
    Fill.LabelParameter(Button_down_Style)
    Label.Text(Label_StayButton.$)
ENDSUB
```

Continuous Timer

EigerScript bietet noch eine weitere Timer-Funktion. Sie heisst „Continuous Timer“ und ist im Prinzip ein Endlos-Timer, der nach Ablauf der vorgegebenen Zeit immer wieder von neuem beginnt. Jedes Mal, wenn die Timerzeit durchlaufen ist, zählt der interne TimerExpiredCounter um eins hoch. Zweitens kann auch jedes Mal eine Aktion ausgelöst, d.h. eine Subroutine aufgerufen werden (vgl. Abbildung 56).

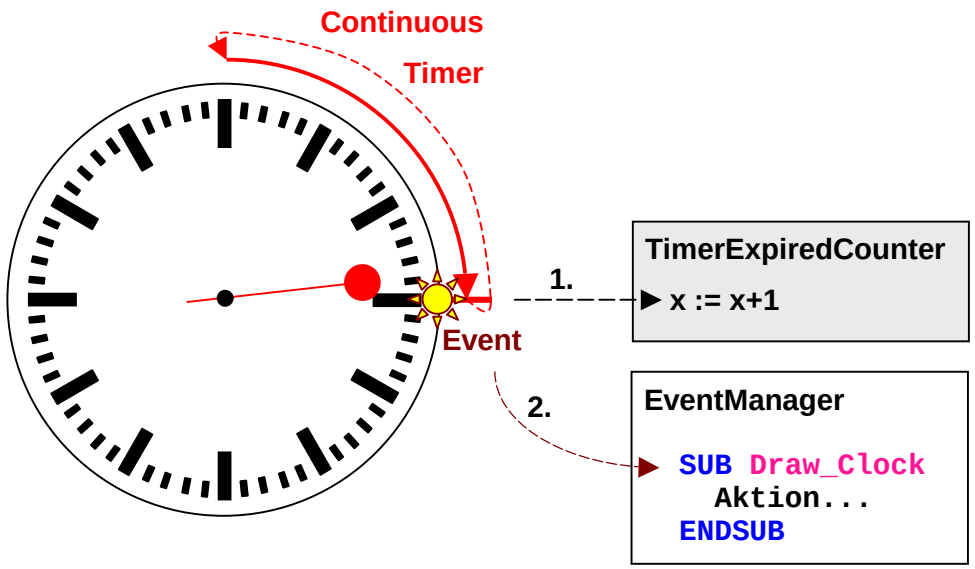
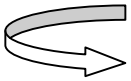


Abbildung 56: Funktionsweise des Continuous Timer. Der Timer durchläuft in einer endlosen Schleife die vorgegebene Zeit. Jeweils bei Ablauf der Zeit geht eine Meldung an den TimerExpiredCounter und an den EventManager.

Mit Hilfe des TimerExpiredCounters können wir eine Uhr programmieren, die beispielsweise die Sekunden zählt.

Wir machen es uns einfach und leiten das Design der Uhr von den Buttons ab, die wir bereits erstellt haben (vgl. z.B. der Übung 7, S.56 ◀).



Aufgabe 5: Bereiten Sie die Grundlage für unsere Uhr vor, indem Sie in der View 2 einen Button positionieren nach folgenden Vorgaben:

- Name der Subroutine: ..... `Draw_Clock`
- Position des Buttons: ..... **links oben** (z.B. X=20 und Y=100)
- Buttonfarbe: ..... **grün** (z.B. green140)
- Style für Button: ..... `Button_down_Style`
- Schriftgrösse: ..... **ca. 20** (z.B. `Font_DigitalNumbers_24`)
- Name der Stringvariablen für die Beschriftung: ..... **`Time.$`**
- Code für die Variablendeklaration: ..... **`STRING [8] Time.$`**

Testen Sie Ihr Ergebnis auf dem FOX.

Einen Lösungsweg finden Sie im Anhang dieses Schnelleinstiegs (S.131 ▶).

Zeitaufwand: 7 Minuten.

Die Timereinheit ist Millisekunden (1/1000 Sek.).

Nachdem nun aufgrund von Aufgabe 5 der Button für die Uhr eingerichtet ist (vgl. Abbildung 61, S.133), fehlt uns noch die Uhr selbst, d.h. die Ziffern, welche uns die Sekundenzahl angeben. Vom Single Timer wissen wir bereits, dass die Zeit in Millisekunden gezählt wird. Für eine Uhr, welche die Sekunden zählt, muss der Zähler des TimerExpiredCounter jede Sekunde um eins erhöht werden, d.h. der Continuous Timer muss seinen Turnus alle 1000 Millisekunden von neuem beginnen (vgl. Abbildung 56). Mit dieser Vorgabe installieren wir mit Hilfe einer neuen Subroutine `SUB ClockTimer` einen zweiten Timer (Timer 1). Für diesen Zweck können wir die bereits bestehende Subroutine `SUB TimerToStartview` als Schema übernehmen und entsprechend anpassen (Beispiel-Code 84).

Beispiel-Code 84 (Kopierte `SUB TimerToStartview` , **angepasst** für den Continuous Timer)

```
SUB ClockTimer
    Timer.InstallLocal(1,Draw_Clock)
    Timer.Load(1,1000)
    Timer.StartContinuous(1)
ENDSUB
```

Die Timer-Tabelle ist im Hauptprogramm mit `Timer.TableEnable()` bereits freigegeben (vgl. Beispiel-Code 77). Es fehlt im Hauptprogramm aber noch der Aufruf für die `SUB ClockTimer`.

Beispiel-Code 85 (Neuer Subroutinenaufruf für den Continuous Timer)

```
    CallSubroutine(Draw_StayButton)
    CallSubroutine(Draw_Clock)
    CallSubroutine(ClockTimer)
```

Sobald der Continuous Timer (Nr.1) gestartet ist, erhält der TimerExpiredCounter jede Sekunden einen Zählimpuls. Ebenfalls im Sekundenabstand wird auch die Subroutine `Draw_Clock` aufgerufen, d.h. neu gezeichnet. Die Uhr soll nun jede Sekunde mit der aktuellsten Sekundenzahl des TimerExpiredCounter gezeichnet werden. Diese Zahl ist durch `Timer.Get_ExpiredCounter(Timer,ExpCounter)` zugänglich. In Klammern geben wir der Methode erstens die Nummer des interessierenden Timers an und zweitens den Namen der Integervariable, in welche der Zählwert des Counters ausgegeben werden soll. Diese zweite Variable deklarieren wir mit dem Namen `Seconds` im Bereich der Deklarationen von View 2 (Beispiel-Code 86).

Beispiel-Code 86 (Deklaration der Integervariablen „`Seconds`“ in der View 2)

```
INTEGER Color2.I
INTEGER Color3.I
INTEGER Seconds.I
```

Die Methode `Timer.Get_ExpiredCounter()` benötigen wir in der Subroutine `Draw_Clock`, welche die Uhr zeichnet. Wir können aber den Inhalt von `Seconds.I` nicht direkt in das Label des Clock-Buttons schreiben, weil `Seconds.I` eine Integervariable und nicht ein String ist. Die Sekundenzahl von `Seconds.I` müssen wir erst noch mit der Methode `Str.Cvt_Integer(VarStr:String,VarInt:Zahl,VarInt:Feldlänge)` in einen String konvertieren. Die Variable für den String haben wir bereits in Aufgabe 5 vorbereitet: `Time.$`. Die zweite Variable in dieser Konvertierungsmethode ist `Seconds.I` und als Feldlänge wählen wir 5 – mehr als 5 Zeichen macht wenig Sinn, da der Wert für eine Integervariable in eigerScript auf maximal +32767 (-32768 bis +32767) beschränkt ist (vgl. Tabelle 11). Beispiel-Code 87 zeigt `SUB Draw_Clock` mit den beiden neuen Methoden.

Beispiel-Code 87 (Subroutine, welche die Uhr inkl. Sekundenzahl darstellt)

```
SUB Draw_Clock
    Timer.Get_ExpiredCounter(1,Seconds.I)
    Str.Cvt_Integer(Time.$,Seconds.I,5)

    eI.Pos_X1 := 20
    eI.Pos_Y1 := 100
    Fill.LabelParameter(Button_down_Style)
    eI.FillColor := green140
    eI.BackColor := green140
    eI.LineColor := green140
    eI.FontNumber := Font_DigitalNumbers_24
    Label.Text(Time.$)
ENDSUB
```

Diese neu programmierte Uhr testen wir nun auf dem FOX ...


U012S003.EGI

... Es tut sich etwas im Uhrenfeld. Das können wir bei genauer Beobachtung aufgrund des zeitweiligen Flackerns feststellen. Allerdings sollte die Uhr die Sekunden zählen und nicht nur eine weisse Null anzeigen (Abbildung 57).

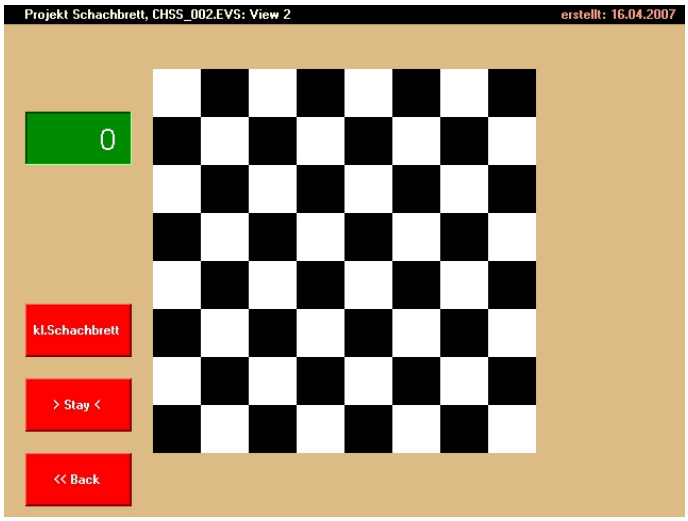


Abbildung 57:
View 2 mit Uhr, welche allerdings die Sekunden noch nicht sichtbar zählt.

Fragen wir einen Experten nach der Lösung unseres Problems, so erhalten wir den Rat, den String `Time.$` vor jedem Konvertierungsbefehl zu leeren. Sonst fügt die Methode `Str.Cvt_Integer()` in der Stringvariablen dem bisherigen Inhalt gemäss Beispiel-Code 87 fünf neue Zeichen an, ohne den bisherigen String zu löschen. Dies ist eine Spezialität der Convert-Routinen. Weil das grüne Feld und der String zu kurz dimensioniert ist, sehen wir während dieses Vorgangs allerdings nur immer die Null. Was sich dahinter abspielt, können wir nur erahnen.

Wir nehmen den Expertenrat zu Herzen und erweitern die Subroutine SUB Draw_Clock an geeigneter Stelle mit einer Zeile, welche den Inhalt von Time.$ löscht, bevor die neue Sekundenzahl abgerufen wird (Beispiel-Code 88).

Beispiel-Code 88 (In der Subroutine wird nun jede Sekunde zuerst der Inhalt von Time.$ gelöscht)

```
SUB Draw_Clock
    Time.$ := ''
    Timer.Get_ExpiredCounter(1,Seconds.I)
```

Ein weiterer Test auf dem FOX ist nun erfolgreich. Die Uhr beginnt zu zählen, sobald wir zur View 2 wechseln. Doch das zeitweilige Flackern ist noch als kleiner Schönheitsfehler geblieben. Dieses Problem können wir beheben, indem wir mit Display.Prepare() die Uhr zuerst im Hintergrund auf dem AVR aufbauen und erst dann als fertiges Bild ins RVR bzw. auf den Bildschirm kopieren (vgl. Übung 6, S.53 ◀). Die Methode Display.ShowWindow() beschränkt sich dabei im Unterschied zu Display.Show() auf den im Programmablauf aktuellen Inhalt der Register eI.Pos_X1, eI.Pos_Y1, eI.Width und eI.Height. Das ist in unserem Fall die Uhr (Beispiel-Code 89).

Beispiel-Code 89 (Mit den Methoden aus der Klasse Display wird die Uhr erst als Ganzes angezeigt)

```
SUB Draw_Clock
    Display.Prepare()
    Time.$ := ''
    Timer.Get_ExpiredCounter(1,Seconds.I)
    Str.Cvt_Integer(Time.$,Seconds.I,5)

    eI.Pos_X1 := 20
    eI.Pos_Y1 := 100
    Fill.LabelParameter(Button_down_Style)
    eI.FillColor := green140
    eI.BackColor := green140
    eI.LineColor := green140
    eI.FontNumber := Font_DigitalNumbers_24
    Label.Text(Time.$)
    Display.ShowWindow()
ENDSUB
```

Nun läuft die Uhr zuverlässig und ohne zu „flackern“.

Anhang

Ein wenig Theorie

Variablen

Eine Variable ist im Prinzip ein *Platzhalter* oder eine *Unbekannte*. Sie ist im Unterschied zu einer Konstanten veränderlich d.h. variabel. Sie kann während des Programmablaufs verschiedene Werte annehmen und beliebig oft einen anderen Wert zugewiesen erhalten. In der Naturwissenschaft ist beispielsweise die Temperatur eine typische Variable.

In Computerprogrammen stellen Variablen statische Speicherbereiche dar, die Lese- und Schreibzugriff haben. Die Struktur dieser Speicherbereiche hängt ab vom Datentyp und von der Länge der Variablen.

Die Variablen werden zur Kompilierzeit erzeugt und bei Bedarf auch initialisiert. eigerScript kennt keine dynamische Speicherverwaltung; somit können zur Laufzeit keine Speicherbereiche alloziert oder freigegeben werden.

Je nach Inhalt der Variablen muss für deren Deklaration ein geeigneter Datentyp gewählt werden. eigerScript kennt die wichtigsten Datentypen, die mit den betreffenden Schlüsselwörtern (z.B. `STRING` oder `INTEGER`) deklariert werden (vgl. Tabelle 11). Die Deklaration einer Variable geschieht immer nach dem Muster:

Variablentyp (Länge) Bezeichner = Initialisierungswert

Tabelle 11: Datentypen in Eigerscript.

Datentyp (benötigter Speicherplatz)	**Beispiel**	**Bemerkungen**
String (je nach Anzahl reservierter Zeichen)	`STRING [15] MyText = 'Dies ist eine Zeichenkette'`	Zeichenketten, bis zu 65'000 Zeichen. Die Angabe einer Zahl in der Klammer veranlasst den Compiler, den gewünschten Speicherplatz in Anzahl Zeichen (Bytes) zu reservieren. Falls der reservierte Platz kleiner ist als der zugewiesene String, wird dieser entsprechend abgeschnitten.
Byte (1 Byte)	`BYTE MyByte = 35` **Für spezielle Funktionen verwendet!**	Ganze Zahlen von -128 bis +127 bzw. von 0 bis 255. Falls die Initialisierung jenseits dieses Zahlbereichs liegt, wird dieser per MODULO-Operation korrigiert.

Integer (2 Bytes)	`INTEGER MyInt = 365`	Ganze Zahlen von -32768 bis +32767 bzw. von 0 bis 65535. Falls die Initialisierung jenseits dieses Zahlbereichs liegt, wird dieser per MODULO-Operation korrigiert.
Long (4 Bytes)	`LONG MyLong = 123456`	Ganze Zahlen von -2‘147‘483‘648 bis +2‘147‘483‘647 bzw. von 0 bis 4'294'967'295.
Single (4 Bytes)	`SINGLE MySingle = 1.234e-4`	Fliesskomma-Zahlen von 1.175×10^{-38} bis 3.403×10^{38}. Die Genauigkeit liegt bei 7 bis 8 Dezimalstellen (einfache Genauigkeit).
Double (8 Bytes)	`DOUBLE MyDouble = 1.1234e-104` **Noch nicht implementiert!**	Fliesskomma-Zahlen von 5.0×10^{-324} bis 1.7×10^{308}. Die Genauigkeit liegt bei 15 bis 16 Dezimalstellen (doppelte Genauigkeit).
Font	`FONT MyFont = 'Helvetica.EFF'`	Definition einer einzubindenden Fontdatei. Die angegebene Fontdatei muss im gleichen Verzeichnis liegen wie das Projekt.

Konstanten

Eine Konstante steht für einen festen Wert. Der Wert kann eine Zahl oder ein Text sein. In der Naturwissenschaft kennen wir beispielsweise die Konstante Pi (π). Wer „Pi“ sagt, meint damit unmissverständlich die Zahl 3.14159... Eine Konstante wird einmal bei der Initialisierung bestimmt und kann danach nicht mehr verändert werden.

Neben den selbst deklarierten Konstanten, gibt es eine Reihe von vordefinierten Konstanten, beispielsweise die Farbkonstanten oder die Fonts in den Header-Dateien „DEF_eiger_Colors.INC“ bzw. „DEF_eiger_Types.INC“.

In eigerScript bestimmen wir Konstanten mit Hilfe des Schlüsselworts `CONST`. Dem Schlüsselwort muss ein Bezeichner, d.h. ein Konstantennamen folgen. Danach folgt ein Gleichheitszeichen und schlussendlich die vertretene Zahl oder Zeichenkette (vgl. Beispiel-Code 90). Zeichenketten müssen zwischen ’Hochkommas’ stehen:

Beispiel-Code 90 (Schema für die Deklaration von Konstanten)

```
CONST Bezeichner = 35
CONST Text = ’Dies ist ein Text’
```

Je nach Position von `CONST` werden globale oder lokale Konstanten erzeugt:

- Im Viewrumpf: globale Konstanten; sie gelten für die ganze View.
- In einer Subroutine (zwischen `SUB` und `ENDSUB`): lokale Konstanten; sie gelten nur innerhalb der betreffenden Subroutine.

Unterschied zwischen Variablen und Konstanten

Variablen können innerhalb eines Projektes oder während des Programmablaufs beliebig oft einen anderen Wert zugewiesen erhalten. Durch die Variablendeklaration erzeugt der Compiler einen Speicherplatz, in dem der jeweilige Wert der Variablen abgelegt wird. Die Variable wird dabei im Byte-Code durch ihre Speicheradresse referenziert. Bei Konstanten wird der Konstantenname beim Kompilieren definitiv mit dem zugewiesenen Wert ersetzt. Dadurch ist der Konstantenname im Byte-Code nicht mehr vorhanden. Der Compiler enthält den Wert, der einer Konstanten zugewiesen worden ist, in allen möglichen Datentypen. Deshalb ist die Typsicherheit jederzeit gewährleistet.

In eigerScript wird bei Zuweisungen zu Variablen innerhalb Routinen immer ":=" gebraucht (mit vorangesetztem Doppelpunkt!). Initialwerte bei der Deklaration von Variablen wie auch Konstanten werden mit "=" zugewiesen.

Projektdefinitionsdatei für globale Variablen und Konstanten

Ab der eigerStudio-Version 0.40 (9.5.2007) können Sie auch globale Variablen und Konstanten einsetzen, die während des ganzen Applikationsablaufs, d.h. ihren Wert auch beim Wechsel der View beibehalten. Das Gefäss für globale Variablen und Konstanten ist die Projektdefinitionsdatei. Diese benennen Sie mit Ihren vier Projekt-Buchstaben und der Endung EPR, also z.B. TEST.EPR (vgl. Beispiel-Code 91). Auch Subroutinen, Styles und Arrays, welche Sie in der Projektdefinitionsdatei plazieren, stehen jeder View des Projekts zur Verfügung.

Der Code der Projektdefinitionsdatei wird beim Systemstart zusammen in das RAM des eigerPanels geladen und verbleibt dort als Überbau, während darunter der View-Code bei jedem View-Wechsel ausgetauscht wird (vgl. Abbildung 58, S.121). Mit dem Schlüsselwort `Globalbase` wird der Platzbedarf für den auswechselbaren View-Code bestimmt. Dieser richtet sich nach der grössten View und sollte grosszügig bemessen sein.

Die Projektdefinitionsdatei ist nach dem Kompilieren im Bytecode der View 1 enthalten. Dadurch ist die „EVI-Datei“ der View 1 i.d.R. grösser als diejenige der höher numerierten Views. Beim Applikationsstart wird bekanntlich die View1 (Startview) aufgerufen und gleichzeitig auch die darin integrierte Projektdefinitionsdatei initialisiert. Damit sind die globalen Variablen, Konstanten, Subroutinen etc. für die ganze Applikation eingeführt. Dies bedeutet aber auch, dass bei jedem Aufruf der Startview, alle globalen Variablen etc. wieder neu definiert und damit auch mit dem initialen Wert überschrieben werden! Wenn Sie das unkontrollierte Überschreiben Ihrer globalen Variablen verhindern wollen, empfiehlt es sich, die Startview exklusiv für die Startinitialisierung zu verwenden.

Die Verweise auf Headerdateien und Includedateien Ihres Projekts plazieren sie mit Vorteil ebenfalls in der Projektdefinitionsdatei. Damit wird Ihr Code schlanker und Ihre Applikation schneller, weil bei einem View-Wechsel keine zusätzlichen Includedateien geladen werden müssen.

Als Alternative zu den globalen Variablen können Sie auch Register nutzen. Was einem Register zugewiesen wird, bleibt so lange erhalten bis es überschrieben wird oder ein Neustart des Systems. Viele Register sind für bestimmte Funktionen reserviert. Die Register `eI.R00` bis `eI.R15` sowie `eI.Garbage` stehen zur freien Verfügung.

Beispiel-Code 91 (Inhalt einer Projektdefinitionsdatei „TEST.EPR“)

```
;+------------------------------------------------------------------+
;| Titel       : Neues Projekt eigerPanel 57                        |
;| File        : TEST.EPR                                           |
;+------------------------------------------------------------------+
;| Compiler    : EigerScript                                        |
;|                                                                  |
;| System      : eigergraphics.com; FOXS embedded computer          |
;|                                                                  |
;| Beschreibung: SystemTest                                         |
;|                                                                  |
;| Version     : Initialversion: 07.03.2009                         |
;|                                                                  |
;| Aenderungen :                                                    |
;|                                                                  |
;|                                                                  |
;|                                                                  |
;|                                                                  |
;+------------------------------------------------------------------+
;|(c) 2005-2009 S-TEC electronics AG, CH-6300 Zug; 041 760 30 88    |
;|             Weitnauer Messtechnik, CH-8752 Näfels; 055 612 51 31|
;+------------------------------------------------------------------+


EIGERPROJECT 'TEST'

IMPORT     'EIGER\\DEF_eVM_OpCodes_0$70.h'         ; Tokens
IMPORT     'EIGER\\DEF_eVM_Registers_0$70.h'       ; Register
FUNCLIB    'EIGER\\DEF_eVM_Functions_0$70.lib' ;Funktionsbibliothek
```

```
INCLUDEFILE 'EIGER\\DEF_eiger_Colors_0$70.INC'; Farbdefinitionen
INCLUDEFILE 'EIGER\\DEF_eiger_Types_0$70.INC' ; Eiger Definitionen

INCLUDEFILE 'EIGER\\NEUB.INC' ; Register des I/O-Boards "NEUB"


GLOBALBASE 100000

VIEWBASE   0


; Globale Konstanten und Variabeln ..........................


; Styles ....................................................
```

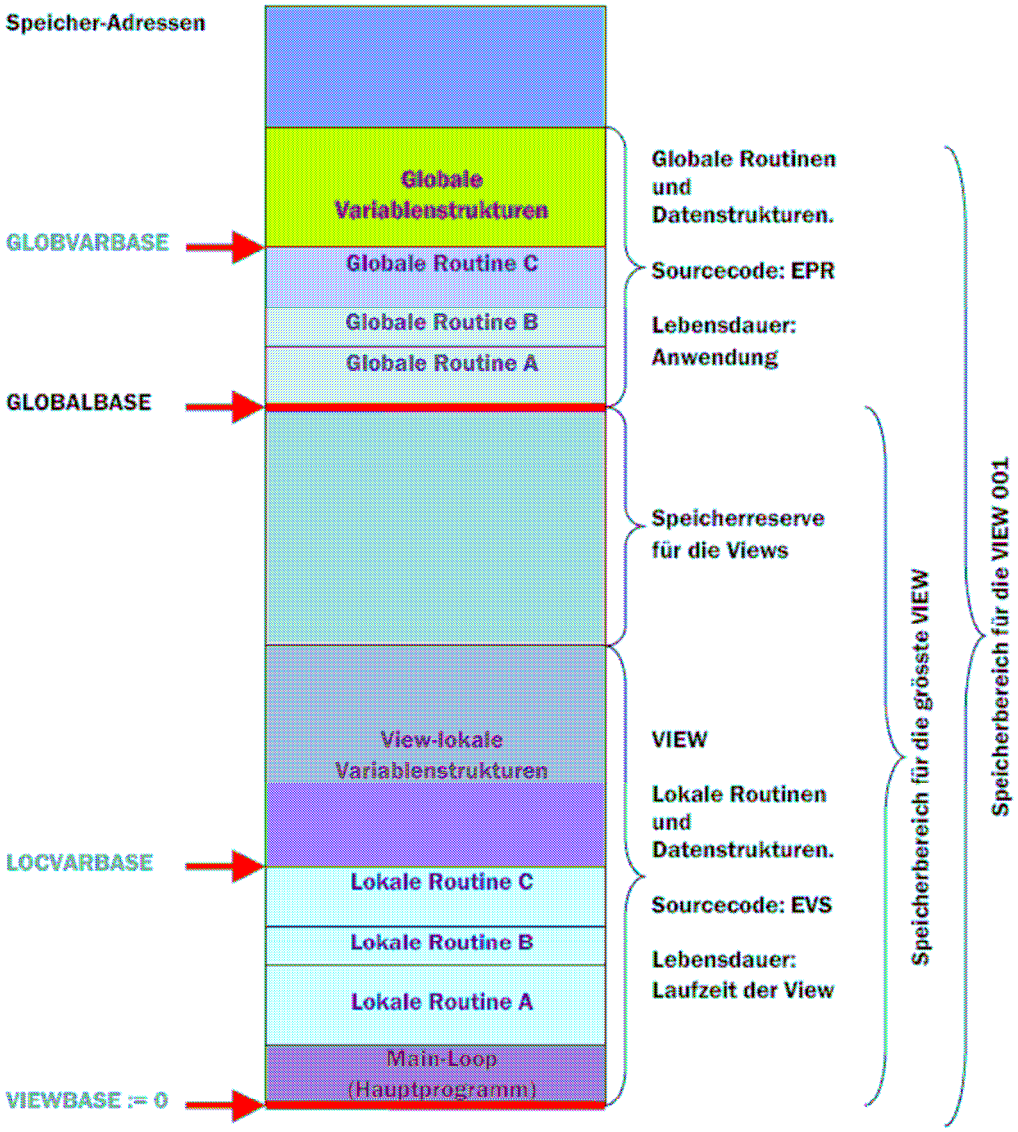
Das eiger-Speichersystem

Der eigerCompiler des eigerStudios bildet und adressiert den Bytecode und legt diesen nach einem bestimmten System in den ausführbaren Dateien ab. Das eigerSpeichermodell ist in Abbildung 58 dargestellt.

Abbildung 58: eiger-Speichersystem nach Weitnauer (2009)[1].

Der Benutzer muss den Wert für GLOBALBASE selber definieren. Wenn der Wert zu klein ist, wird er zwar gewarnt, aber eine weitere Hilfe existiert noch nicht. (Dies würde einen zweifachen Build-Vorgang benötigen)

Die anderen drei Adressen, diejenigen, die grau gesetzt sind, stehen für den Benutzer nicht zur Verfügung (obwohl sie der Compiler bereits beachtet...)



[1] A. Weitnauer, 2009: Speichersystem eigerPanel. PDF-Datei.
Link: http://www.eigergraphics.com/Download/docs/Speichermodell_eigerPanel.pdf

Lösungen zu den Aufgaben

Übung 8, Aufgabe 1, S.73

Sofern an Ihrem FOX ein Keyboard angeschlossen ist, belegen Sie die zweite Taste mit der Screendump-Funktion.

Lösung:

Die Subroutine zum Aufnehmen eines Screenshot ist bereits im Code der Startview vorhanden. Diese `SUB ScreenDump` können wir mit einem zweiten Hotkey aufrufen (Beispiel-Code 92).

Beispiel-Code 92 (zweiter HotKey im Hauptprogramm für Screenshot)

```
; HotKey als Alternative für Fotograf-Button ---------------------
    HotKey.InputFlush()        ; Eingabepuffer leeren
    HotKey.InstallLocalKey("A",Draw_Label_Name,0)
    HotKey.InstallLocalKey("a",Clear_Label_Name,0)

; HotKey für Screenshot ---------------------------------------
    HotKey.InstallLocalKey("B",ScreenDump,0)

    HotKey.EnableLocalKeys()
    HotKey.TableEnable()
```

Übung 9, Aufgabe 2, S.82

Der View2-Button ist noch nicht animiert, d.h. er „bewegt“ sich nicht, wenn wir darauf drücken. Versuchen Sie, den neuen Button analog zum Fotograf-Button zu animieren.

Lösung A:

Die Methode `HotSpot.Install(Enter,Leave,Down,Up)` kann auf vier Ereignisse reagieren. Wenn das Down-Ereignis den View-Wechsel auslösen soll, braucht es einzig für dieses Ereignis eine Button-Animation (vgl. Tabelle 12). Die Ereignisse „Leave“ bzw. „Up“ treten erst nach dem Down-Ereignis und damit nach dem View-Wechsel ein. Beim Down-Ereignis soll der View2-Button mit der Aufschrift „Schachbrett >>“ dem Druck „nachgeben“, d.h. den `Button_down_Style` annehmen.

Tabelle 12:
Events der Methode `HotSpot.Install()` und die Aktionen, welche im Beispiel von diesen Ereignissen ausgelöst werden.

Ereignis (Event)	Animation des Buttons und ausgelöste Aktion
Enter, d.h. der Druck verschiebt sich auf dem Touchscreen von ausserhalb in das HotSpotfeld herein).	Button soll nicht „reagieren“. Keine Aktion.
Leave, d.h. der Druck verschiebt sich auf dem Touchscreen vom Innenbereich des HotSpotfeldes hinaus in dessen Umgebung.	Animation und Aktion aufgrund dieses Ereignisses erübrigen sich, da bei einem allfälligen Eintreffen dieses Ereignisses der View-Wechsel längst vollzogen ist.
Down, d.h. es wird direkt auf das HotSpotfeld gedrückt.	Button wird „eingedrückt“ sobald auf den HotSpot gedrückt wird (`Button_down_Style`). **FOX wechselt zur View 2.**
Up, d.h. der Druck auf den HotSpot hört auf.	Animation und Aktion aufgrund dieses Ereignisses erübrigen sich, da bei dessen Eintreffen der View-Wechsel längst vollzogen ist.

Für die Methode `HotSpot.Install()` des View2-Buttons wird die Zusammenstellung in Tabelle 12 keine direkten Auswirkungen haben (Beispiel-Code 93). Wir belassen die Variable NIL für alle Ereignisse ausser für Down.

Beispiel-Code 93 (View-Wechsel im Falle des **Down**-Ereignisses – „alles“ bleibt beim Alten)

```
; HotSpot für den Wechsel zu View 2 ----------------------------
    Load.Geometry_XYWH(20,420,100,50)
    HotSpot.Install(NIL,NIL,ToView2,NIL)
```

Die Animation regeln wir in der Subroutine `ToView2` mit Hilfe des `Button_down_Style` (vgl. Beispiel-Code 94). Dieser Style stellt den Button als „eingedrückt“ dar. Auf den `Button_down_Style` bezieht sich auch schon der Fotograf-Button. Um den Style gemeinsam nutzen zu können, müssen wir – wie schon beim `Button_up_Style` – die Positionsregister im Style mit `no_change` neutralisieren (Beispiel-Code 1). Die X- und Y-Werte definieren wir direkt bei den Zeichen-Subroutinen der einzelnen Buttons (Beispiel-Code 94 und Beispiel-Code 96). Die Subroutine `ToView2` bewirkt nun, dass der Button eingedrückt wird und dann View 2 erscheint.

Beispiel-Code 94 (vor dem View-Wechsel wird der Button im `Button_down_Style` gezeichnet)

```
SUB ToView2
    eI.Pos_X1 := 20
    eI.Pos_Y1 := 420
    Fill.LabelParameter(Button_down_Style)
    Label.Text(Label_Button_View2.$)
    GotoView(002)
ENDSUB
```

Beispiel-Code 95 (die bisher konkreten X- und Y-Werte sind nun auch durch no_change ersetzt)

```
Button_down_Style:
    INLINEWORDS  (no_change)         ; entspricht eI.Pos_X1
    INLINEWORDS  (no_change)         ; entspricht eI.Pos_Y1
```

Beispiel-Code 96 (X- und Y-Werte werden direkt beim Zeichenbefehl für den Fotograf-Button definiert)

```
SUB Draw_Label_Name
    eI.Pos_X1 := 400    ; X-Koordinate für den Fotograf-Button
    eI.Pos_Y1 := 420    ; Y-Koordinate für den Fotograf-Button
    Fill.LabelParameter(Button_down_Style)  ; zeichnet den Foto-
                        ; Button als "eingedrückten" Button
    Label.Text(Label_FotoButton.$)
    Fill.LabelParameter(Name_Style)  ; schreibt den Namen des
                        ; Fotografen auf den View-Hintergrund rechts
                        ; neben den Button
    Label.Text(Label_Name.$)
```

Der Test dieser Neuerungen auf dem FOX zeigt, dass die Animation des View2-Buttons beim Down-Ereignis kaum erkennbar ist.

Lösung B:

Als Alternative zu Lösung A können wir den View-Wechsel auch erst auf das Up-Ereignis legen. Damit kommt eine Animation deutlicher zum Ausdruck, denn damit kann der Button auch auf die Ereignisse „Enter“ und „Leave“ reagieren (vgl. Tabelle 13). Diese letzteren und auch das Up-Ereignis sind bisher mit der Variablen „NIL“ inaktiv gesetzt. In der alternativen Lösung wollen wir nun auch die Ereignisse „Leave“ und „Up“ mit entsprechenden Subroutinen-Aufrufen belegen. Das Enter-Ereignis belassen wir auch weiterhin inaktiv. Mit der Aktivierung des Leave-Ereignisses geben wir die Möglichkeit noch „auszusteigen“, wenn man fälschlicherweise auf den Button drückt.

Tabelle 13: Events der Methode `HotSpot.Install()` und die Aktionen, welche im Beispiel von diesen Ereignissen ausgelöst werden.

Ereignis (Event)	Animation des Buttons und ausgelöste Aktion
Enter, d.h. der Druck verschiebt sich auf dem Touchscreen von ausserhalb in das HotSpotfeld herein).	Button reagiert nicht, wenn der Druck ins HotSpotfeld eintritt. Auch sonst keine Aktion.
Leave, d.h. der Druck verschiebt sich auf dem Touchscreen von innerhalb aus dem HotSpotfeld hinaus).	Button ist wegen des vorangegangenen Down-Ereignisses „eingedrückt“ (`Button_down_Style`) und „springt zurück“ bei Verlassen des HotSpotfeldes (`Button_up_Style`) Sonst keine Aktion.
Down, d.h. es wird direkt auf das HotSpotfeld gedrückt.	Button wird „eingedrückt“ sobald auf den HotSpot gedrückt wird und bleibt „eingedrückt“ solange der Druck im HotSpotfeld besteht (`Button_down_Style`). Sonst keine Aktion.
Up, d.h. der Druck auf den HotSpot hört auf.	Keine Button-Animation. **FOX wechselt zur View 2.**

Die Überlegungen in Tabelle 13 setzen wir nun in der Methode HotSpot.Install() um (Beispiel-Code 97).

Beispiel-Code 97 (View-Wechsel im Falle des **Up**-Ereignisses)

```
; HotSpot zum Wechsel zu View 2 ----------------------------
    Load.Geometry_XYWH(20,420,100,50)
    HotSpot.Install(NIL,Draw_Button_View2,Draw_Button_down_View2,
ToView2)
```

Für das Down-Ereignis brauchen wir eine Soubroutine, die mit dem `Button_down_Style` den „eingedrückten“ Button zeichnet. Sonst soll nichts geschehen (Beispiel-Code 98).

Beispiel-Code 98 (zeichnet den eingedrückten View2-Button)

```
SUB Draw_Button_down_View2
    eI.Pos_X1 := 20
    eI.Pos_Y1 := 420
    Fill.LabelParameter(Button_down_Style)
    Label.Text(Label_Button_View2.$)
ENDSUB
```

Da sich schon der Fotograf-Button auf den `Button_down_Style` bezieht, dürfen wir die X- und Y-Koordinaten nicht im „Style-Körper“ definieren. Analog zu Lösung A ersetzen wir die Werte des Fotograf-Button im Style mit no_change (Beispiel-Code 95) und weisen diese direkt in der Subroutine `Draw_Label_Name` dem Fotograf-Button zu (Beispiel-Code 96).

Beispiel-Code 99 (die bisher konkreten X- und Y-Werte sind nun auch durch no_change ersetzt)

```
Button_down_Style:
    INLINEWORDS  (no_change)        ; entspricht eI.Pos_X1
    INLINEWORDS  (no_change)        ; entspricht eI.Pos_Y1
```

Im Up-Ereignis animieren wir den Button nicht, sondern wechseln mit der unveränderten Subroutine `ToView2` (Beispiel-Code 100) direkt zur View 2.

Beispiel-Code 100

```
SUB ToView2
    GotoView(002)
ENDSUB
```

Übung 10, Aufgabe 3, S.92

Von der View 2 gibt es bisher keine Möglichkeit zur Startview zurückzukehren. Installieren Sie in der View 2 einen Back-Button, indem Sie den Code des View2-Buttons von der Startview in die View 2 kopieren und diesen entsprechend anpassen. Lagern Sie wo möglich allfällige Konstanten- und Variablendeklarationen sowie Subroutinen, die aufgrund der Neuerungen in beiden Views identisch sind, in das Includefile aus.

Lösung:

Die Subroutine `Draw_Button_View2` in der Startview kann auch für den Back-Button in View 2 gebraucht werden. Wir exportieren diese deshalb aus der Startview ins Includefile CHSS_Global.EVS und benennen sie mit einem etwas allgemeineren Namen, z.B. `Draw_Button_ChangeView`. (vgl. Beispiel-Code 101). Entsprechende Anpassungen müssen wir auch bei der Methode `Label.Text` (vgl. Beispiel-Code 101) und in der Startview bei der Stringdeklaration (vgl. Beispiel-Code 102) vornehmen.

Beispiel-Code 101 (**Anpassungen** in der Subroutine `Draw_Button_View2` ; sie wird von der Startview ins Includefile verschoben)

```
SUB Draw_Button_ChangeView
    eI.Pos_X1 := 20
    eI.Pos_Y1 := 420
    Fill.LabelParameter(Button_up_Style)
    Label.Text(Label_Button_ChangeView.$)
ENDSUB
```

Beispiel-Code 102 (**Anpassungen** in den Stringdeklarationen der Startview)

```
; Definitionen für Startview -------------------------------------

STRING [60] Titel.$ = 'Projekt Schachbrett, CHSS_001.EVS: Startview'
STRING [15] Label_FotoButton.$ = 'Fotograf:'
STRING [15] Label_Name.$ = 'Christoph Angst'
STRING [15] Label_Button_ChangeView.$ = 'Schachbrett >>'
```

Wie mit `Draw_Button_View2` gehen wir auch bei der Subroutine `Draw_Button_down_View2` vor.

Beispiel-Code 103 (**Anpassungen** in der Subroutine `Draw_Button_down_View2` ; sie wird von der Startview ins Includefile verschoben)

```
SUB Draw_Button_down_ChangeView
    eI.Pos_X1 := 20
    eI.Pos_Y1 := 420
    Fill.LabelParameter(Button_down_Style)
    Label.Text(Label_Button_ChangeView.$)
ENDSUB
```

Im Hauptprogramm der Startview passen wir den Code zum Aufruf der Subroutine Draw_Button_View2 dem neuen Namen an (vgl. Beispiel-Code 104). Den gleichen Befehl fügen wir mit Copy/Paste auch ins Hauptprogramm der View 2 ein.

Beispiel-Code 104 (**Anpassung** am Subroutinenaufruf im Hauptprogramm der Startview)

```
    CallSubroutine(Draw_Button_ChangeView)
```

In der Startview müssen wir die HotSpot-Methode anpassen, welche sich auf die umbenannte Subroutine bezieht (Beispiel-Code 105). Auch diese beiden Codezeilen kopieren wir zusätzlich ins Hauptprogramm von View 2.

Beispiel-Code 105 (**Anpassungen** an der HotSpot-Methode im Hauptprogramm der Startview)

```
; HotSpot für den Wechsel zu View 2 ----------------------------
    Load.Geometry_XYWH(20,420,100,50)
    HotSpot.Install(NIL,Draw_Button_ChangeView,Draw_Button_down_Chan
geView,ToView2)
```

In der View 2 soll der HotSpot die neue Subroutine `ToStartview` (Beispiel-Code 106) aktivieren, analog zu `ToView2` in der Startview. Dies hat eine entsprechende Anpassung in der HotSpot-Methode zur Folge, die wir bereits in der View 2 eingefügt haben (vgl. Beispiel-Code 107).

Beispiel-Code 106 (neue Subroutine für den Wechsel zurück zur Startview)

```
SUB ToStartview
    GotoView(001)
ENDSUB
```

Beispiel-Code 107 (**Anpassung** an der HotSpot-Methode im Hauptprogramm der View 2)

```
; HotSpot für den Wechsel zu View 2 ----------------------------
```

```
    Load.Geometry_XYWH(20,420,100,50)
    HotSpot.Install(NIL,Draw_Button_ChangeView,Draw_Button_down_Chan
geView,ToStartview)
```

Die Aufschrift des neuen Buttons in der View 2 soll „<< Back“ lauten. Dafür sorgt eine neue Stringdeklaration, die wir in der View 2 an geeigneter Stelle vornehmen, analog zur Stringdeklaration in der Startview (vgl. Beispiel-Code 102).

Beispiel-Code 108 (neue Stringdeklaration für den Button, der zur Startview zurück wechselt)

```
; Definitionen für View 2 -------------------------------------
STRING [60] Titel.$ = 'Projekt Schachbrett, CHSS_002.EVS: View 2'
STRING [15] Label_Button_ChangeView.$ = '<< Back'
```

Als letztes können wir noch die beiden Styles `Button_up_Style` und `Button_up_Style` von der Startview die Subroutine `SUB Styles` des Includefiles auslagern. In der View 2 löschen wir die ganze Subroutine `SUB Styles`.

Nach dem Kompilieren der Views und Übertragung auf den FOX können wir nun dank des neuen Back-Buttons (vgl. Abbildung 59) beliebig oft zwischen Startview und View 2 wechseln.

U010S003.EGI

Abbildung 59: View 2 mit Back Button.

Übung 11, Aufgabe 4, S.73

Richten Sie in der View 2 einen zusätzlichen Button mit Hotspot ein, der das kleine Schachbrett einblendet.

Lösung:

Dem neuen Button geben wir die Aufschrift „kl. Schachbrett“ (Beispiel-Code 109).

Beispiel-Code 109 (Stringdeklaration für Buttonbeschriftung zum Zeichnen des kleinen Schachbretts)

```
; Definitionen für View 2 -------------------------------------
STRING [60] Titel.$ = 'Projekt Schachbrett, CHSS_002.EVS: View 2'
STRING [15] Label_Button_ChangeView.$ = '<< Back'
STRING [20] Label_SmallChessboard.$ = 'kl.Schachbrett'
```

Um den Button zu zeichnen, schreiben wir im Script der View 2 eine neue Subroutine `Draw_Button_SmallChessboard`. Diese generieren wir z.B. von einer Kopie der Subroutine `Draw_Button_ChangeView` aus dem Includefile CHSS_Global. Die entsprechenden Anpassungen sind in Beispiel-Code 110 gezeigt.

Beispiel-Code 110 (Kopierte `Draw_Button_ChangeView` mit entsprechenden **Anpassungen** in View 2)

```
SUB Draw_Button_SmallChessboard
    eI.Pos_X1 := 20
    eI.Pos_Y1 := 350
    Fill.LabelParameter(Button_up_Style)
    Label.Text(Label_SmallChessboard.$)
ENDSUB
```

Den gleichen Button brauchen wir auch im Button_down_Style (

Beispiel-Code 111 (Subroutine für den „eingedrückten“ Button in der View 2)

```
SUB Draw_Button_down_SmallChessboard
    eI.Pos_X1 := 20
    eI.Pos_Y1 := 350
    Fill.LabelParameter(Button_down_Style)
    Label.Text(Label_SmallChessboard.$)
ENDSUB
```

Im Hauptprogramm muss ein Aufruf auf die Subroutine `Draw_Button_SmallChessboard` verweisen (Beispiel-Code 112), damit der Button in der View 2 gezeichnet wird.

Beispiel-Code 112 (Aufruf der Subroutine **`Draw_Button_SmallChessboard`** im Hauptprogramm)

```
BEGINVIEW
    EVE.Init()                      ; EVE ANNA initialisieren
    eI.DisplayColor := burlywood
    Display.Clear()                 ; Bildschirm löschen
    Display.Prepare()
    CallSubroutine(Draw_Titel)
    CallSubroutine(Draw_Button_ChangeView)
    CallSubroutine(Draw_Button_SmallChessboard)
```

Im Hauptprogramm kopieren wir einen bereits vorhandenen HotSpot-Befehlsblock und passen das Duplikat dem neuen Button an (Beispiel-Code 113). Dabei soll das kleine Schachbrett beim Event „Up“ durch die Subroutine `Draw_SmallChessboard` (vgl. Beispiel-Code 114) gezeichnet werden. Beim Event „Enter“ soll der Button nicht reagieren (NIL), beim Event „Leave“ soll der Button ohne weitere Konsequenzen vom „eingedrückten“ Zustand, der durch den Event „Down“ verursacht wird, in den Normalzustand zurückspringen.

Beispiel-Code 113 (Neuer Hotspot im Hauptprogramm zum Zeichnen des kleinen Schachbretts)

```
; HotSpot zum Zeichnen des kleinen Schachbretts ------------------
    Load.Geometry_XYWH(20,350,100,50)
    HotSpot.Install(NIL,Draw_Button_SmallChessboard,Draw_Button_down
_SmallChessboard,Draw_SmallChessboard,)
```

Die Position und die Masse des kleinen Schachbretts geben wir in der Subroutine `Draw_SmallChessboard` an (Beispiel-Code 114). Diese Subroutine bedient sich dann – unter Berücksichtigung unserer Postions- und Massvorgaben – der eigentlichen Schachbrett-Subroutine `Draw_Chessboard` zum Zeichnen des kleinen Schachbretts. Weil das kleine Schachbrett erst in Verbindung mit dem „Up“-Event gezeichnet wird (vgl. Beispiel-Code 113), muss gleichzeitig auch wieder der Button in den Normalzustand zurückspringen.

Beispiel-Code 114 (Subroutine enthält Positions- und Massangaben und Zeichenbefehl für Schachbrett sowie den Zeichenbefehl für den Butten im Up-Zustand)

```
SUB Draw_SmallChessboard
    Load.Geometry_XYWH(520,60,10,10) ;Pos. & Geom. für Rechteck
    CallSubroutine(Draw_Chessboard)  ; Aufruf zum Zeichnen
    CallSubroutine(Draw_Button_SmallChessboard)
ENDSUB
```

U011S008.EGI

Mit den hier eingesetzten Positions- und Massvorgaben zeigt sich die neue View 2 wie in Abbildung 60 dargestellt.

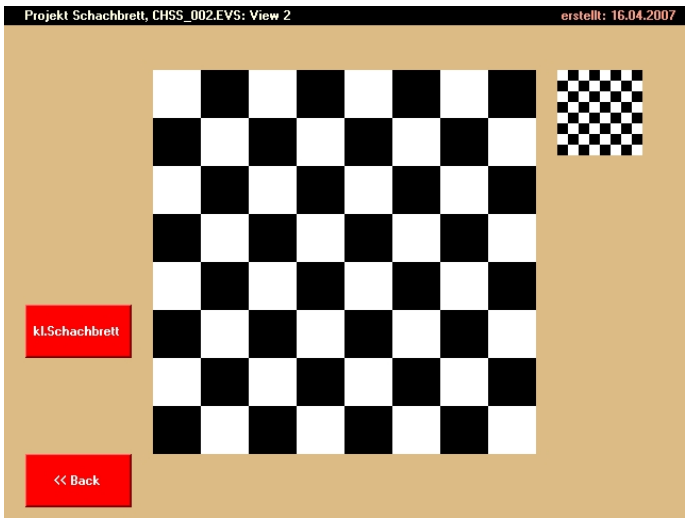


Abbildung 60:
View 2 mit Button, der das kleine Schachbrett aufruft.

Übung 12, Aufgabe 5, S.112

Bereiten Sie die Grundlage für unsere Uhr vor, indem Sie in der View 2 einen Button positionieren nach folgenden Vorgaben::

Name der Subroutine:	`Draw_Clock`
Position des Buttons:	**links oben** (z.B. X=20 und Y=100)
Buttonfarbe:	**grün** (z.B. green140)
Style für Button:	`Button_down_Style`
Schriftgrösse:	**ca. 20** (z.B. `Font_DigitalNumbers_24`)
Name der Stringvariablen für die Beschriftung:	`Time.$`
Code für die Variablendeklaration:	`STRING [8] Time.$`

Testen Sie Ihr Ergebnis auf dem FOX.

Lösung:

Als Vorlage dient uns die Subroutine `SUB Draw_Button_down_ChangeView` im Includefile CHSS_Global.EVS. Diese Subroutine kopieren wir in die View 2, beispielsweise direkt oberhalb von `SUB Draw_Chessboard`. An der Subroutine nehmen wir die erforderlichen Anpassungen vor (vgl. Beispiel-Code 115).

Beispiel-Code 115 (ursprüngliche `SUB Draw_Button_down_ChangeView` kopiert in die View 2 und **angepasst** für eine Uhr)

```
SUB Draw_Clock
    eI.Pos_X1 := 20
```

```
    eI.Pos_Y1 := 100
    Fill.LabelParameter(Button_down_Style)
    eI.FillColor := green140
    eI.BackColor := green140
    eI.LineColor := green140
    eI.FontNumber := Font_DigitalNumbers_24
    Label.Text(Time.$)
ENDSUB
```

Im oberen Teil des Scripts der View 2 deklarieren wir die Stringvariablen `Time`. Sie wird uns den Text, d.h. heisst die Uhrzeit liefern (vgl. Beispiel-Code 116).

Beispiel-Code 116 (Deklaration der Stringvariablen `Time` für die Beschriftung der Uhr)

```
; Definitionen für View 2 ---------------------------------------
STRING [60] Titel.$ = 'Projekt Schachbrett, CHSS_002.EVS: View 2'
STRING [15] Label_Button_ChangeView.$ = '<< Back'
STRING [8] Time.$
```

Im Hauptprogramm positionieren wir den entsprechenden Subroutinenaufruf (vgl. Beispiel-Code 117).

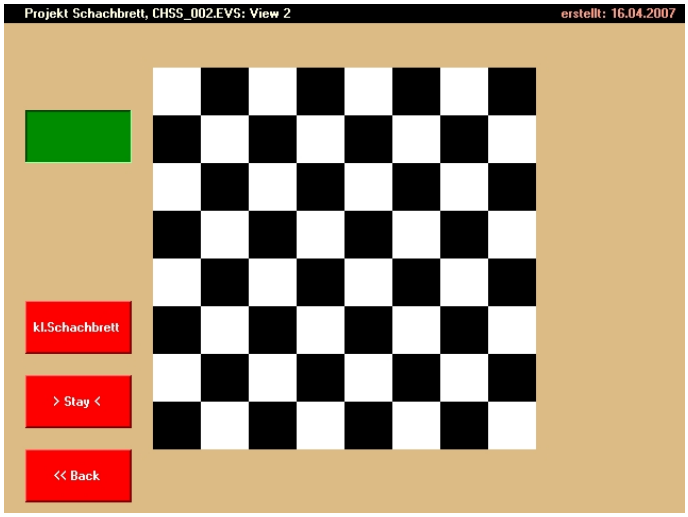
Beispiel-Code 117 (Hauptprogramm der View 2 mit Aufruf der Subroutine zum Zeichnen der Uhr)

```
BEGINVIEW
    EVE.Init()                      ; EVE ANNA initialisieren
    eI.DisplayColor := burlywood
    Display.Clear()                 ; Bildschirm löschen
    Display.Prepare()
    CallSubroutine(Draw_Titel)
    CallSubroutine(Draw_Button_ChangeView)
    CallSubroutine(TimerToStartview)
    CallSubroutine(Draw_StayButton)
    CallSubroutine(Draw_Clock)
```


U012S002.EGI

Damit ist die Aufgabe gelöst, und wir testen unser Ergebnis auf dem FOX. Wir sehen in der View 2 links oben das neue grüne Feld, in welchem im weiteren Verlauf der Übung 12 die Uhr in weisser Schrift die Sekunden zählen wird (vgl. Abbildung 61).

Abbildung 61:
View 2 mit grünem Feld für die Uhr.



Abkürzungen und Fachbegriffe

AVR	**A**ccessible **V**ideo **R**AM. Dieser Speicher wird vom Prozessor gelesen und geschrieben. Das „Bild“ auf dem AVR kann mit einem COPY-Befehl ins RVR geladen und dadurch auf dem FOX-Bildschirm sichtbar gemacht werden.
eI.	**e**iger**I**nteger-Register. Es sind virtuelle Maschinenregister, die als globale Variablen betrachtet werden können.
eiger	Kombination aus FOX-Rechner und Display mit einem Touchpanel (→ HMI).
EVE	**e**iger**V**ideo**E**ngine ist die Grafikmaschine bzw. der Grafik-Chip von FOX. EVE verwaltet das Video-Memory, macht das Interface zum Microcontroller und ist zuständig für den Display-Refresh.
EVI	**e**iger**V**iew **I**nterpretable; Format einer für die virtuelle Maschine FOX ausführbaren Datei. Sie enthält das Programm eines eigerView-Scripts im Byte-Code. Eine EVI-Datei ist eine kompilierte, d.h. in Byte-Code „übersetzte“ EVS-Datei. Hierzu dient der eigerCompiler, ein Element von eigerStudio.
EVS	**e**iger**V**iew**S**ource; Dateiformat des eigerView-Scripts, das mit der integrierten Entwicklungsumgebung eigerStudio geschrieben und abgespeichert wird. Es ist ein Textformat.
Globale Konstanten	*Siehe Konstanten.*
GUI	**G**raphical **U**ser **I**nterface (grafische Benutzerschnittstelle). Grafische Benutzeroberfläche, welche dem Benutzer die Programmbedienung erleichtert.
HMI	**H**uman-**M**achine **I**nterface (Mensch-Maschine-Schnittstelle, Benutzeroberfläche). Beim FOX gehören v.a. der Touchscreen, allfällige Tasten und Potentiometer dazu, beim Auto beispielsweise das Steuer, das Armaturenbrett etc.
IDE	**I**ntegrated **D**evelopment **E**nvironment (Integrierte Entwicklungsumgebung). Anwendungsprogramm zur Entwicklung von Software. EigerStudio ist eine IDE.
Konstanten	*Globale Konstanten* sind innerhalb einer ganzen View bekannt. Sollen globale Konstanten innerhalb eines ganzen Projekts verwendet werden, empfiehlt es sich, diese in einer eigenen Datei zu deklarieren. Diese können dann in den einzelnen Views mit Hilfe des Schlüsselwortes `INCLUDEFILE` genutzt werden. *Lokale Konstanten* werden nur temporär in Subroutinen (Prozeduren) definiert. *Stringkonstante:* eine Stringkonstante enthält primär eine Zeichenkette. Wird die Konstante als Wertparameter

	eingesetzt, versucht der Compiler den Zahlenwert zu ermitteln. *Wertkonstante:* Eine Wertkonstante (z.B. `CONST Max = 3`) enthält primär einen ganzzahligen Wert. Wird die Konstante als Stringparameter verwendet (z.B. `CONST Max = 'Test'`), ersetzt der Compiler den Wert durch seine Stringrepräsentation.
Lokale Konstanten	*Siehe Konstanten.*
RVR	**R**efresh **V**ideo **R**AM. In diesem Speicher ist das ausgegebene Bild gespeichert.
Stringkonstanten	*Siehe Konstanten.*
VGA	**V**ideo **G**raphics **A**rray steht für einen bestimmten Computergrafik-Standard. Eine Bildauflösung von 640 x 480 Pixel gilt als VGA-Auflösung und ist auch das Format von FOX.
Wertkonstanten	*Siehe Konstanten.*

Wie kann ich...

eine Übersicht über die **eiger-internen Fonts** auf dem Display des eigerPanels anzeigen?	▶ Application Note „Fonts und ASCII-Codes“
eigene Fonts in mein eigerProjekt einbauen? ...	▶ Application Note „Neuen Font einbetten“
mit eigerScript die **seriellen Schnittstellen RS232** programmieren?	▶ Application Note „Serielle Schnittstelle RS232 programmieren“

Stichwortverzeichnis

Notizen: