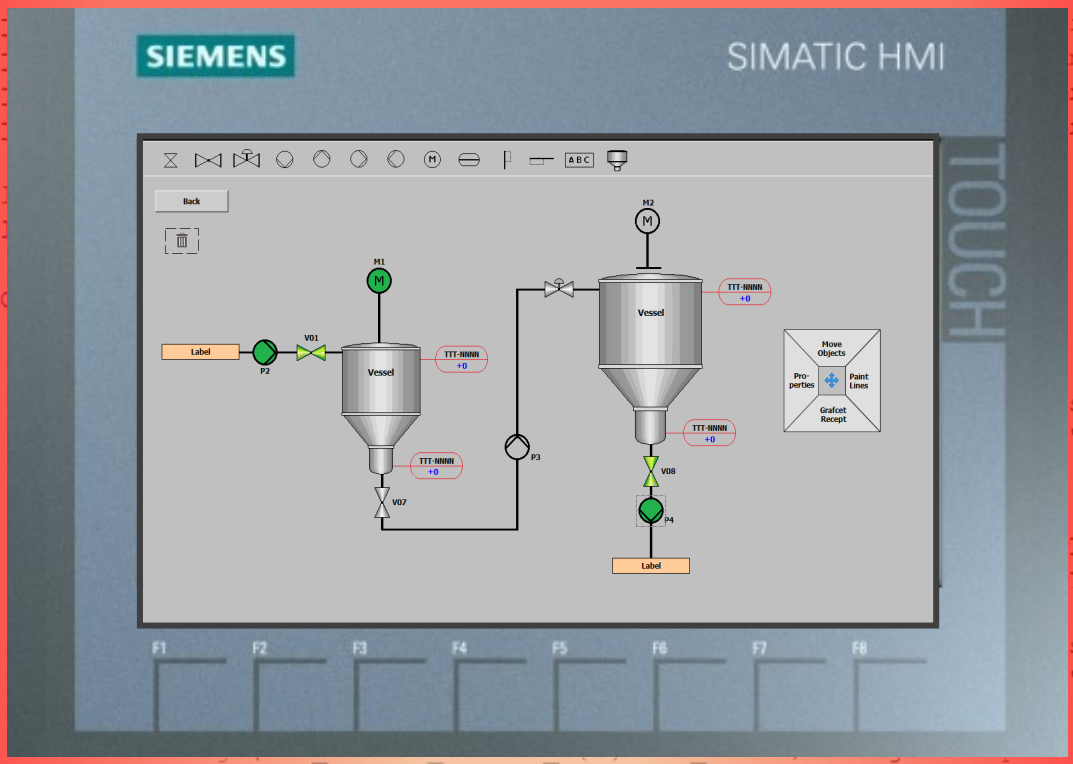
Johannes Hofer



TIA Portal

Objekte mit der Maus bewegen

Einführung in Skripte schreiben mit WinCC Advanced

Eine WinCC-Applikation in Runtime, wie eine App im Smartphone zu benutzen

Inkl. TIA Portal Projekte

Johannes Hofer

Objekte mit der Maus bewegen

TIA Portal

Objekte mit der Maus bewegen

Einführung in Skripte schreiben mit WinCC Advanced

Eine neue Anwendung mit WinCC Advanced. Die Idee:

> Eine WinCC-Applikation in Runtime, wie eine App im Smartphone zu benutzen.

1. Auflage (Februar 2017)

Rechtliches:

Haftungsausschluss: Die Beispielskripts werden im gegenwärtigen Zustand und ohne jegliche Gewährleistung zur Verfügung gestellt. Der Autor schließt darüber hinaus jede Gewährleistung aus, die für einen bestimmten Zweck wie auch immer aus der Verwendung oder Ausführung der Beispielskripts verwendet werden. Der Autor kann in keinem Fall durch die Bereitstellung der Skripts irgendwelche Schäden ohne Einschränkung haftbar gemacht werden. Ebenso ist eine Haftung für die Benutzung der CD ausgeschlossen.

ISBN 978-84-617-8556-8

Warenzeichen:
STEP®, SIMATIC®, TIA-Portal®, S7-300®, S7-400®, S7-1200®, S7-1500®, PLCSIM® und IOT2000® sind eingetragene Warenzeichen der SIEMENS Aktiengesellschaft.

Vorwort

Diskontinuierliche Prozesse werden traditionell so visualisiert, dass die Steuerung (SPS) und die Visualisierung eine feste Einheit darstellen. Die SPS steuert und regelt durch ihre proprietäre Programmierung den Prozess und die Visualisierung wird dazu meist als Bediener-Schnittstelle genutzt (HMI). Der Anwender bestimmt zur Projektphase was und wie visualisiert werden soll und hat nach der Programmierung und Abnahme seiner Anlage, ohne zusätzlich hohen Kostenaufwand, keinen Einfluss mehr auf sein Projekt.

Mit der neuen Methode Bildbausteine und deren Funktionen mit der Maus wie in einem Smartphone zu nutzen, entsteht eine moderne und flexible Möglichkeit, auch in der Runtime entsprechend erlaubte Anpassungen vorzunehmen. Das hat nicht nur Einfluss auf die Gestaltung des Prozesses, sondern auch auf die Wiederverwendung von bereits getesteter Software. Dadurch entsteht eine völlig neue Sicht zwischen Prozess und Realisierung und bietet verschiedene Alternativen an, welche in der traditionellen Vorgehensweise nicht erreicht werden können.

In der chemischen Fertigung z. B. geradezu ideal geeignet, wenn Prozesse in Runtime jederzeit neu entstehen oder geändert werden können. Besonders im Laborbereich ist eine solche Flexibilität nicht nur gewünscht, sondern manchmal zwingend notwendig.

Beispielerweise Lagerstellen unterhalb einer automatischen Krananlage, können so den täglichen Bedürfnissen der Produktion und Materialanlieferung angepasst werden, ohne auch nur ein Stück zu programmieren.

Ein besonders zu erwähnendes Beispiel sind Werkzeug-Rundschalt-Tische, welche typischen Ablaufsteuerungen unterliegen, erlauben so über das Bediengerät ohne Expertenwissen das Auswechseln der Werkzeuge und damit auch deren Funktionalität (Programm).

Da gibt es eine ganze Menge neuer Ideen, Objekte mit der Maus in Runtime zu bewegen. Dazu gehören nicht nur Bildobjekte, sondern auch komplette Planstrukturen, wie z.B. in P&ID- oder GRAFCET-Pläne. Die starre Konstruktion und das Festhalten an eingebrannte Strukturen bezüglich der traditionellen Automatisierung erfährt mit dieser Technologie eine neue, zeitgerechte Dimension.

Ich wünsche allen Lesern viel Spaß mit dieser neuen Idee!

Johannes Hofer

1 Inhaltsverzeichnis

2 Einige Grundlagen zu Wincc-Objekte und deren Eigenschaften

Die SIMATIC HMI[1] bietet unter anderem einzelne Komponenten an, welche standardisierte Schnittstellen besitzen, die es erlauben über die Skript-Technik (VBS[2]) deren Eigenschaften in Runtime (RT[3]) nicht nur zu lesen, sondern auch schreibend zu verändern. Beispielhaft hat der Kreis (*Circle*) u. a. die in der **Tabelle 2.1** gezeigten Eigenschaften.

Eigenschaft	Beschreibung	Datentyp	Zugriff in RT
BackColor	Hintergrundfarbe	Color	Read & Write
BorderColor	Linienfarbe	Color	Read & Write
BorderWidth	Linienstärke	Long	Read & Write
Visible	Sichtbarkeit	Boolean	Read & Write
Width	Legt die Weite (Breite) fest	Long	Read & Write
Height	Höhe des Objektes	Long	Read & Write
ObjectName	Name des Objektes	String	Read
Top	Y-Koordinate des Objektes	Double	Read & Write
Left	X-Koordinate des Objektes	Double	Read & Write

Tabelle 2.1 Eigenschaften von "Circle"

Die Eigenschaften eines Elementes können über einen Skript nur verändert werden, wenn der Zugriff auf die Eigenschaft mit *Write* gekennzeichnet ist.

Zum Beispiel kann bei *Circle* die Eigenschaft *ObjektName* nur gelesen und nicht geschrieben werden. Objekte welche in RT bewegt werden sollen, müssen für die Eigenschaften *Top* und *Left* den Zugriff *Write* besitzt. Soll die Größe des Objektes verändert werden, müssen die Eigenschaften *Width* und *Height* den Zugriff *Write* besitzen.

Objekte	Eigenschaften Left, Top	Eigenschaften Width, Height
Linie	Read & Write	Read
Circle	Read & Write	Read & Write
Ellipse	Read & Write	Read & Write
Polyline	Read & Write	Read
Polygon	Read & Write	Read
Rectangle	Read & Write	Read & Write
TextField	Read & Write	Read
IOField	Read & Write	Read
Button	Read & Write	Read

Tabelle 2.2 Auswahl einiger Objekte mit den Eigenschaften Bewegen und Größe in RT verändern

[1] [HMI] Human Machine Interface
[2] [VBS] Visual Basic Script ist eine von Microsoft© entwickelte Scriptsprache
[3] [RT] Runtime

Im Beispiel *Circle* ist beides möglich. In **Tabelle 2.2** sind einige Objekte aufgelistet, welche in RT bewegt werden können (*Left, Top*). Die Änderung der Gösse (*Width, Height*) ist ebenfalls aufgelistet, da diese sich von den Eigenschaften der Bewegung unterscheiden kann. Die aktuellen Informationen aller Objekte können aus dem WinCC-Hilfesystem entnommen werden.

Noch eine wichtige Eigenschaft ist *Visible* vom Typ *Boolean*.

Die in einem Projekt verwendeten Objekte, müssen die Eigenschaft *Visible* vom Typ *Boolean* mit *Read & Write* besitzen, wenn diese wie in einer App auf dem Smartphone eingefügt und gelöscht werden sollen (siehe **Kapitel 6**).

2.1 Erstes praktisches Beispiel zu „Move Objects“

Vorausgesetzt wird ein TIA-Projekt mit einer HMI. Eine PLC ist vorerst nicht notwendig. Das TIA-Projekt zu diesem und allen folgenden Kapiteln sind mit der TIA-Version V13, SP1, UpD8 übersetzt worden. Für dieses Beispiel werden geringe VBS-Kenntnisse und die Bedienung zur Oberfläche TIA Portal vorausgesetzt. Im folgenden **Kapitel 2.1.1** wird der Zeichenvorgang und die Programmierung mit dem TIA Portal detailliert dargestellt und kann so nebenbei auch als kurze Einführung in das TIA Portal genutzt werden. In den weiteren Kapiteln, wird dann auf die detaillierten Hinweise verzichtet.

2.1.1 Objekt *Circle* zeichnen und per Skript bewegen

Zuerst werden die Objekte Kreis und Schaltfläche aus *Werkzeuge/Basisobjekte/Elemente* in das Grundbild gezogen (**Bild 2.1**, **Punkt 1**). Danach wird noch eine leere VB-Skript-Datei (**Punkt 2**) mit der Bezeichnung *MoveCircle* (*Rechter Mausklick auf VB-Skripte, dann Neue VB-Funktion hinzufügen*) eingefügt.

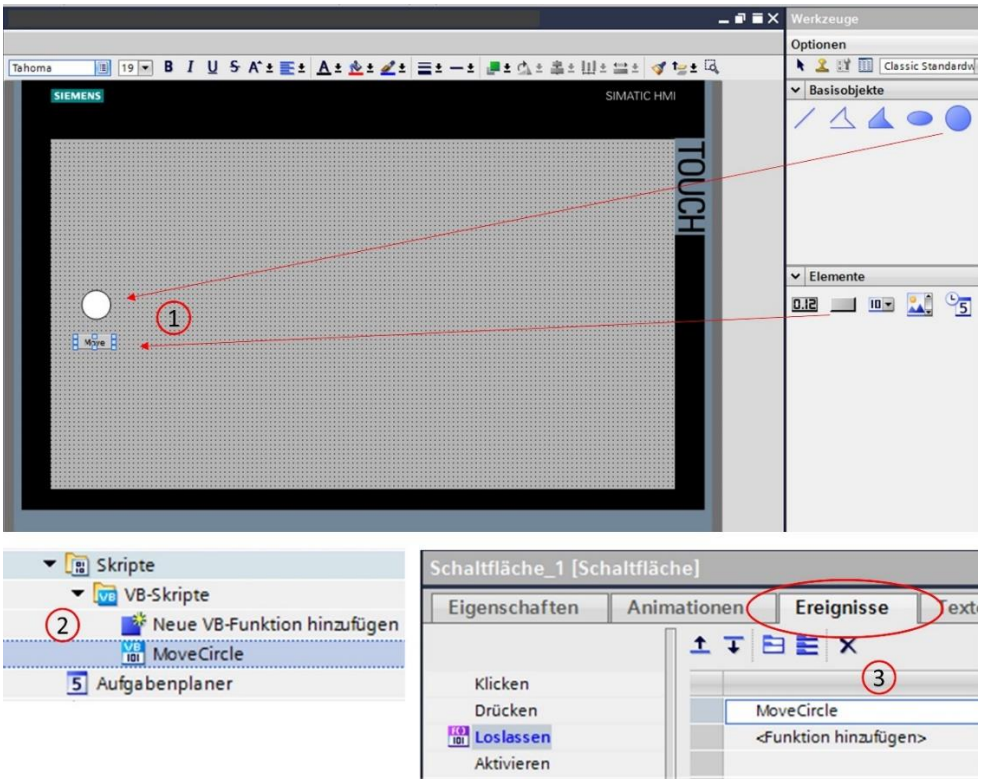


Bild 2.1 Grafische Objekte zeichnen und Ereignis „MoveCircle“ der „Schaltfläche“ zuordnen

Für die Schaltfläche wird das Ereignis *Loslassen* mit dem Skript *MoveCircle* in die Funktionsliste eingetragen (*Klick auf Schaltfläche, Ereignis, Loslassen und MoveCircle in <Funktion hinzufügen> eintragen*.

Mit dem Doppelklick auf den Skript *MoveCircle* gelangen wir in den VBS-Editor (**Bild 2.2**). Im ersten, kleinen Test soll der Kreis auf die X-Koordinate 100 und auf die Y-Koordinate 50 bewegt werden. Zur Eingabe kann die Hilfeunterstützung für die Auffindung der Skript-Befehle und der Objekte benutzt werden, welche nachfolgend in 5 Punkten kurz erläutert werden:

Punkt 1: In Zeile 4 (VBS-Programm) mit *Strg + Leerstelle HmiRuntime* auswählen. Der Text *HmiRuntime* wird eingefügt.

Punkt 2: Nach Eingabe des Punktes hinter *HmiRuntime* wird *Screens* selektiert.

Punkt 3: Nach Eingabe der „Klammer-Auf“ hinter *Screens*, wird mit *Strg + j* der Dialog zur Suche des Bildes ausgegeben und *Grundbild* ausgewählt.

Punkt 4: Mit Eingabe des Punktes nach *Grundbild* wird *ScreenItems* selektiert.

Punkt 5: Schließlich kann nach Eingabe der „Klammer-Auf“ mit *Strg + j* der Dialog zur Suche des Objektes mit *Circle* selektiert werden.

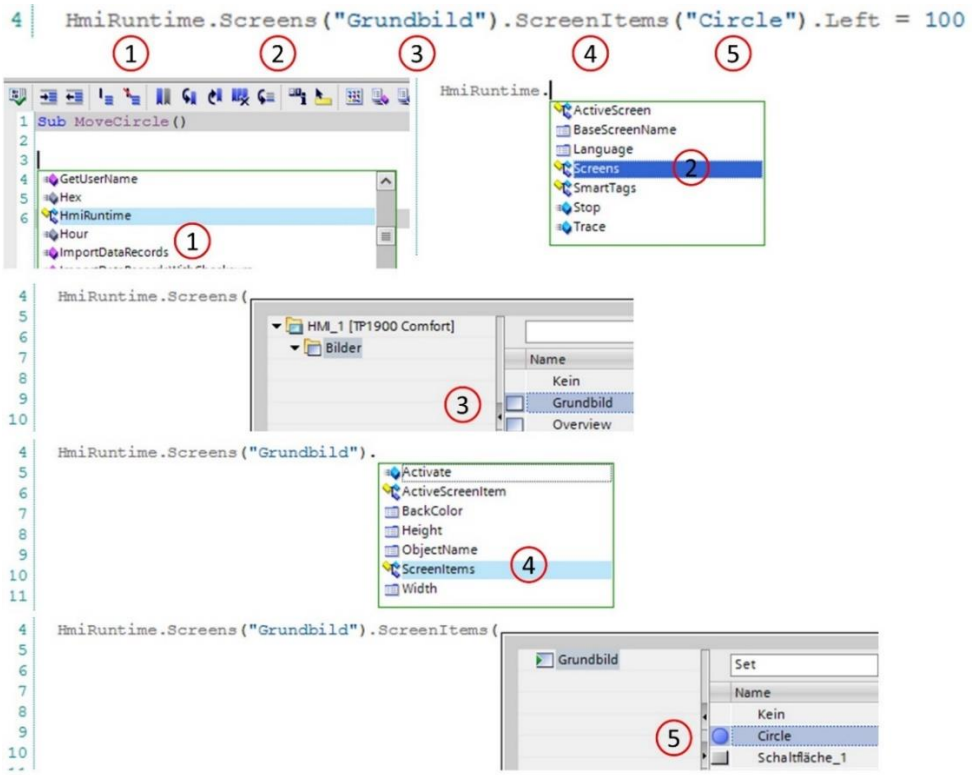


Bild 2.2 Objekt über HmiRuntime suchen und editieren

Die Hilfeunterstützung mit „Strg+Leerstelle“ und „Strg+j“ ist sehr hilfreich und wird besonders zu Beginn der Programmierung häufiger genutzt. Bei Erzeugung eines neuen Skriptes wird der Hilfe-Hinweis als Text in Kommentarform in den Skript kopiert (siehe **Bild 2.3**, Zeilen 2-7). Wenn das nicht gewünscht wird, dann kann die Option (*Extra/Einstellungen/Visualisierung/Runtime Scripting*), wie in **Bild 2.3** ersichtlich, abgeschaltet werden.

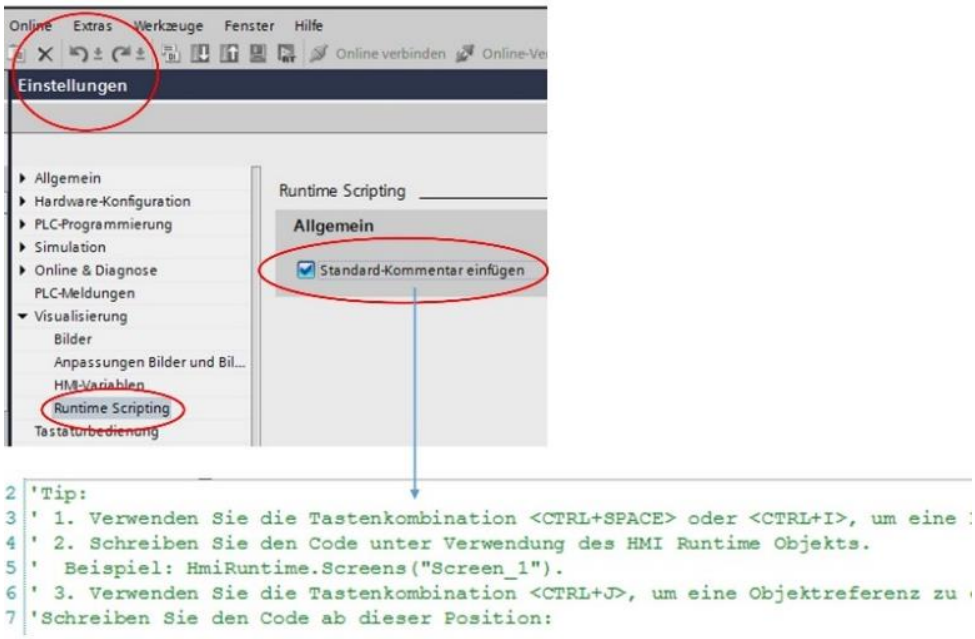


Bild 2.3 Hilfeoption als Kommentar an/ausschalten

Der erste kleine VBS ist nun fertig (**Bild 2.4**). Mit dem Punktoperator nach *ScreenItems(Circle)* wird die Eigenschaft *Left* bzw. *Top* selektiert und die Werte 100 und 50 zugewiesen, welche durch das Ereignis der Schaltfläche *Move* ausgelöst werden.

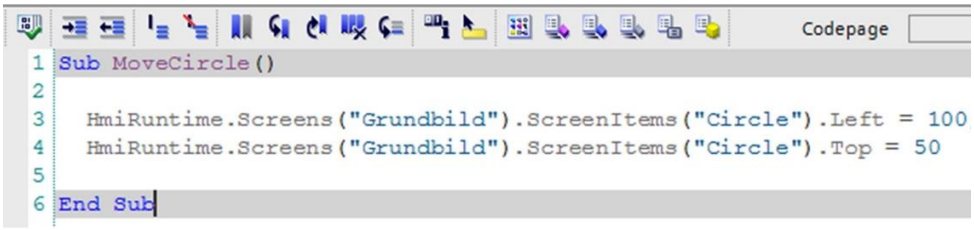


Bild 2.4 VBS-„MoveCircle“

Betrachten wir nach der Kompilierung mit (*rechter Mausklick auf HMI_1/Übersetzen/Software (komplett übersetzen)*) das Ergebnis in der HMI-Simulation (**Bild 2.5**), dann können wir die gezeichnete Position des Kreises und die Position nach Betätigung der Schaltfläche *Move* erkennen.

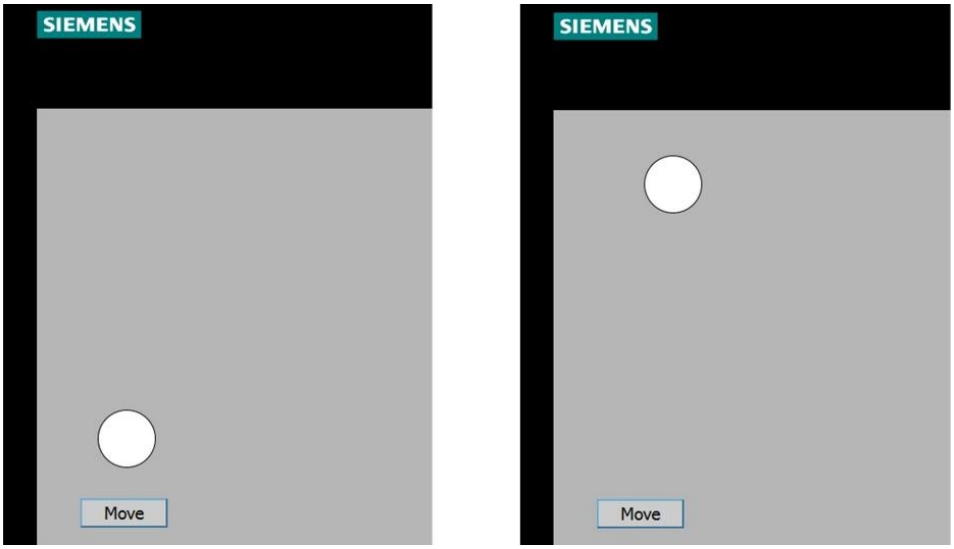


Bild 2.5 *MoveCircle* in Aktion

Der Kreis hat sich auf die Position (100, 50) positioniert und zeigt, dass der Skript funktioniert hat. Nun wird das Projekt mit einer zusätzlichen Bildseite *Bild_1* und je einer Schaltfläche erweitert, welche vom *Grundbild* in das *Bild_1* und aus *Bild_1* wieder zum *Grundbild* verzweigt (**Bild 2.6**). Das Ergebnis bezüglich des Bildwechsels wird im folgenden Kapiteln erläutert.

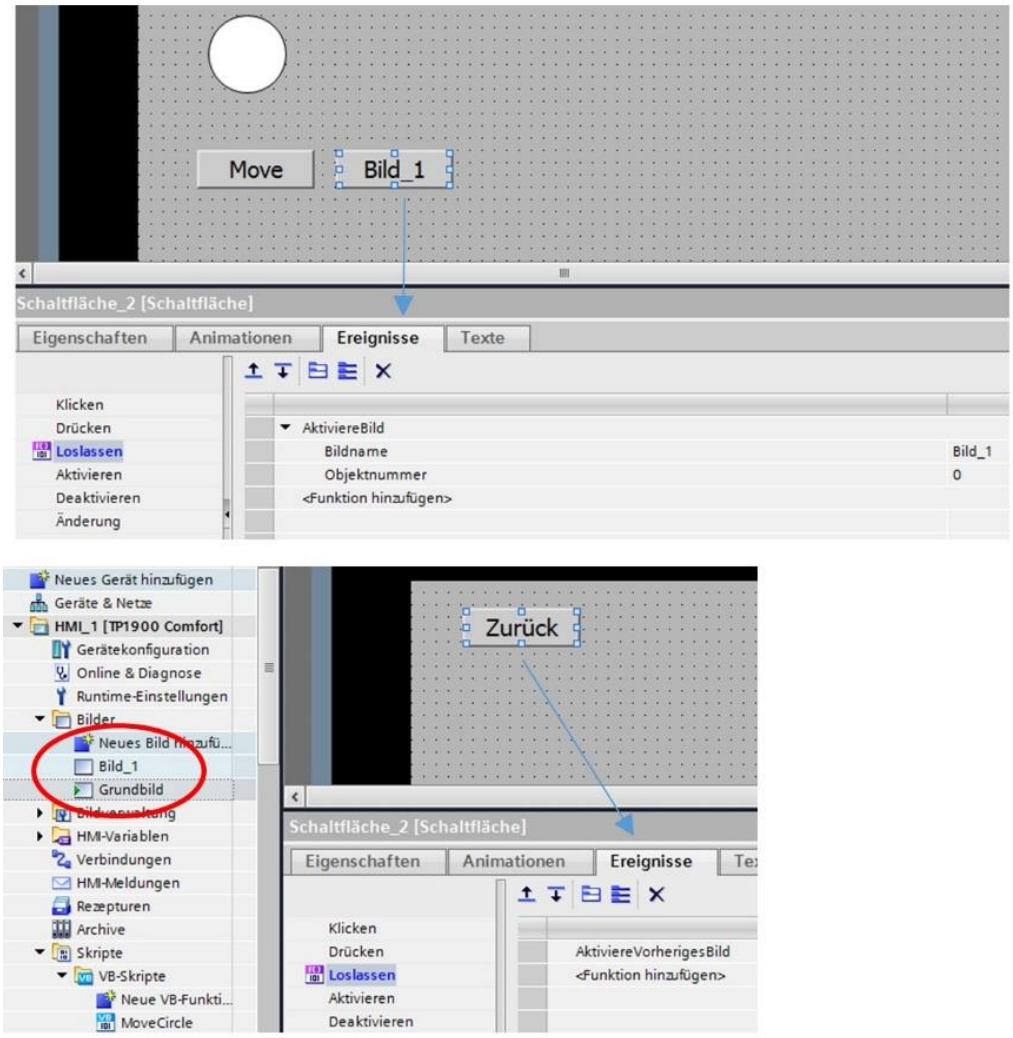


Bild 2.6 *Bild_1* hinzufügen

2.1.2 Bildwechsel zerstört Position

Das HMI-Projekt wird nach dem Hinzufügen der neuen Elemente wieder komplett neu kompiliert, die HMI-Simulation gestartet und vor dem Bildwechsel nach *Bild_1* die Taste *Move* gedrückt. Es kann beobachtet werden, dass sich der Kreis wie erwartet auf die Koordinaten 100, 50 bewegt. Danach erfolgt der Bildwechsel nach *Bild_1* und wieder zurück zum *Grundbild*.

Überraschenderweise befindet sich die Kreisfläche wieder in der gezeichneten Position, obwohl dieser zuvor verschoben wurde.

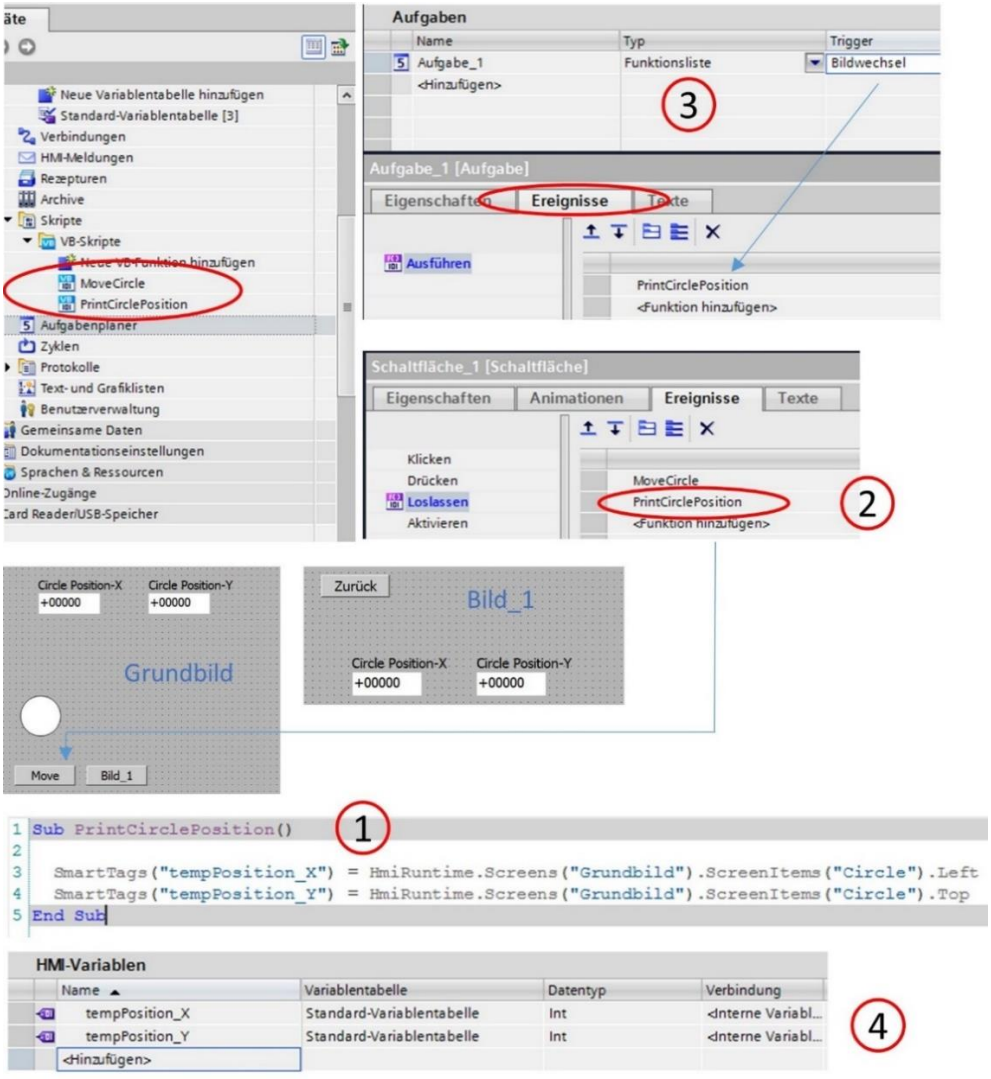


Bild 2.7 Anzeige der Koordinaten

Um der Ursache auf den Grund zu gehen, wurden das *Grundbild* und *Bild_1* mit der Anzeige der Koordinaten erweitert (**Bild 2.7**), welche durch einen neuen Skript *PrintCirclePosition* (**Punkt 1**) angezeigt werden. Mit der Schaltfläche *Move* wird der neue Skript *PrintCirclePosition* nach dem Skript *MoveCircle* aufgerufen (**Punkt 2**) Zudem wurde im Aufgabenplaner die Aufgabe *Bildwechsel* aktiviert (**Punkt 3**), welche in Ereignisse bei einem Bildwechsel ebenfalls den Skript *PrintCirclePosition* aufruft. Die Variablen für die X-Y-Positionen der Kreisfläche sind in der HMI-Variablen-Tabelle ersichtlich (**Punkt 4**).

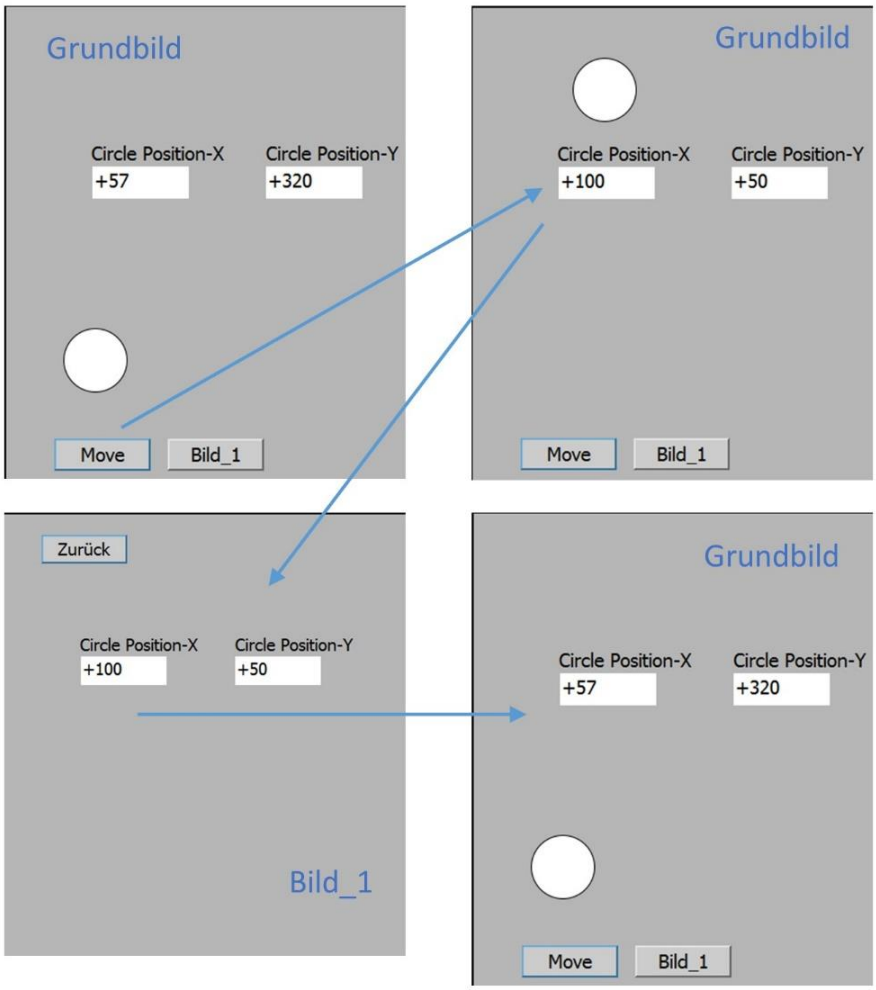


Bild 2.8 Koordinaten nach dem Bildwechsel

Die Schaltfläche *Move* im *Grundbild* wurde, wie schon erwähnt, mit dem Aufruf des Skriptes *PrintCirclePosition* erweitert, sodass gut zu beobachten ist, in welcher Position sich der Kreis befindet, wenn die Schaltfläche betätigt wird. Bei einem Bildwechsel mit der Schaltfläche *Bild_1* wird nun durch den Aufgabenplaner ebenfalls der Skript *PrintCirclePosition*

aufgerufen und so die noch gültigen Werte in die Variablen kopiert. Diese bleiben im *Bild_1* noch erhalten, jedoch mit dem Bildwechsel in das *Grundbild*, werden wieder die alten, gezeichneten Daten angezeigt (**Bild 2.8**).

Diese Erkenntnis ist sehr wichtig, da die Elemente in einem Bild immer so dargestellt werden, wie diese gezeichnet wurden. Mit dem Grundsätzlichen Aufbau, Elemente mit der Maus zu bewegen, muss dies bei einem Bildwechsel berücksichtigt werden.

2.2 Zusammenfassung

Das Beispiel zeigt deutlich, dass es ein Problem darstellt, Objekteigenschaften bezüglich der Koordinaten (*Left*, *Top*) in Skripten zu verändern,

da diese nach einem Bildwechsel wieder so abgebildet werden, wie sie ursprünglich gezeichnet wurden.

Außerdem muss darauf geachtet werden, dass

Änderungen in Skripten zur Position des Objektes nur unter „HmiRuntime.**Screens**“ funktionieren.

Also nicht unter „HmiRuntime.**ActiveScreen**.

Ein Skript welcher Bilddaten verändert, funktioniert in der RT nur dann, wenn das betroffene Bild dazu, gerade aktiv angezeigt wird.

3 Zyklisches Skript-Intervall

Wie in **Kapitel 2.1.1** erläutert, können Elemente mit einem Skript über dessen Eigenschaften bewegt werden. Um nun die Position eines oder mehrere Elemente kontinuierlich zu verändern, muss dieser Script zyklisch aufgerufen werden. Das gilt besonders dann, wenn die Koordinaten des Elementes über die Maus-Koordinaten kontinuierlich angepasst werden sollen.

Ein Skript in der HMI kann bei Wertänderung einer **SPS-Variablen**, durch die RT aufgerufen werden.

Diese Tatsache soll nun genutzt werden, ca. mindestens 500-mal innerhalb einer Sekunde einen Skript zu aktivieren, welcher dann ein oder mehrere Objekte mit den aktuellen Mausdaten nachführen soll. Das entspricht einer zeitlichen Auflösung, welche für die flüssige Darstellung der bewegten Objekte am Monitor, für das menschliche Auge völlig ausreichend ist. Je nach Hardwaretyp der SPS und Typ des Bediengerätes, ist die Auflösung tatsächlich wesentlich höher. Wie das nun praktisch umgesetzt wird, zeigt das folgende **Kapitel 3.1**.

3.1 Request UserAction

Das folgende TIA-Projekt zu diesem Kapitel (**Bild 3.1**) zeigt wie ein zyklischer Aufruf für einen Skript über die Änderung eines Wertes der SPS-Variablen erfolgt. Die Variable befindet sich in einem DB innerhalb der SPS. Ausgelöst wird der Skript *UserAction*, sobald mit der linken Maustaste auf ein Bildobjekt geklickt wird.

In der SPS wird der Baustein *DB_Request_HMI* (*DB2*) mit der Variablen *_requestLoop* vom Typ *Int* (**Punkt 1**) angelegt. Diese Variable wird in der HMI als HMI-Variable mit der entsprechenden Verbindung zur SPS registriert (**Punkt 2**). Zusätzlich wird die interne HMI-Variable *requestUserAction* vom Typ *Bool* deklariert (**Punkt 3**).

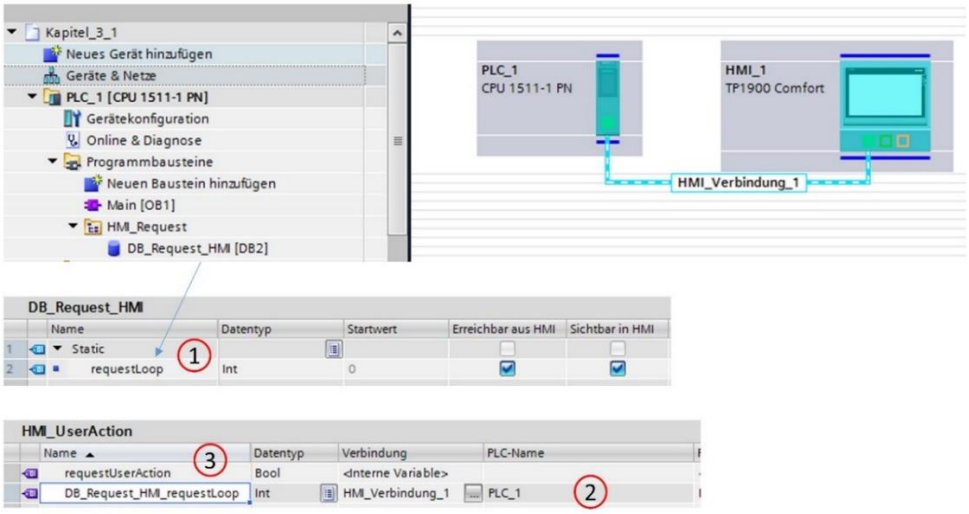


Bild 3.1 Projekt CPU1500 mit HMI-TP1900 Comfort

Damit der Skript *UserAction* immer dann aufgerufen wird, wenn sich die Variable *DB_Request_HMI_requestLoop* ändert, muss die Wertänderung in der HMI durch ein Ereignis bekannt gemacht werden. Das ist in **Bild 3.2** ersichtlich.

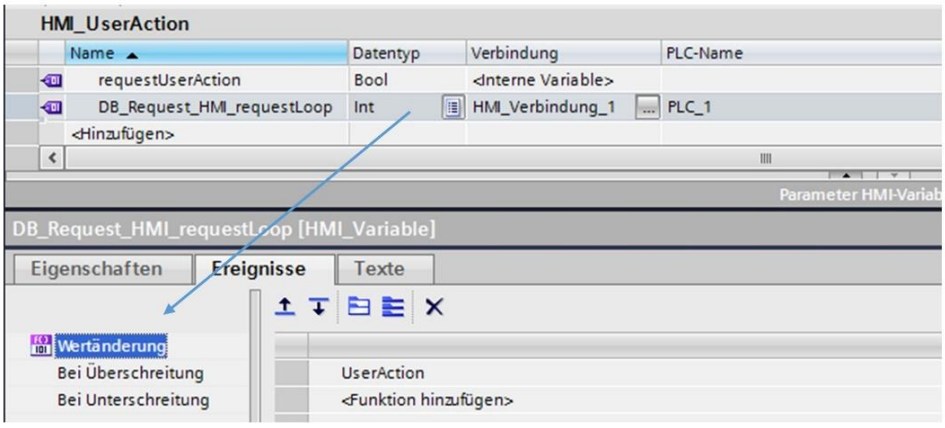


Bild 3.2 Eigenschaft für den Request-Event festlegen

Die HMI_Variablen können über dessen Ereignisse, z.B. bei einer Wertänderung, eine Funktion ausführen (Funktionsliste).

In unserer Anwendung ist das der Aufruf des Skriptes *UserAction*. Der Trick für einen zyklischen Aufruf liegt darin, dass der erste Aufruf von *UserAction* durch den Klick mit der linken Maus-Taste erfolgt und dann im Skript *UserAction* die Variable

DB_Request_HMI_requestLoop bei jedem Aufruf verändert wird. Die Änderung wird in der SPS gespeichert und löst wiederum in der HMI einen neuen *Event* aus. Dieser *Event* ist in **Ereignisse** der Variablen *DB_Request_HMI_requestLoop* unter **Wertänderung** mit dem Skript *UserAction* eingetragen. Der Skript wird aufgerufen und dort ändert sich wieder die Variable *DB_Request_HMI_requestLoop* und der Kreislauf schließt sich, bis der Zyklus abgebrochen wird. Das ist dann der Fall, wenn die linke Maustaste wieder losgelassen wird. Betrachten wir zum besseren Verständnis den Skript *UserAction* in **Kapitel 3.1.1**.

3.1.1 Der Skript UserAction

Über einem Kreis-Element befindet sich eine Schaltfläche, welche ein Ereignis bei Wertänderung auslösen kann (**Bild 3.3**). Beim Drücken der linken Maustaste werden 2 Ereignisse eingetragen (**Punkt 1**). Die interne Variable *requestUserAction* vom Typ *Bool* wird mit *SetzeBit* auf *True* gesetzt und danach der Skript *UserAction* aufgerufen.

Das Ereignis wird **nur einmal** ausgeführt, auch wenn der Event **Drücken** heißt, bedeutet das **nicht**, dass diese Aktion nun solange ausgeführt wird, solange die linke Maustaste gedrückt wird.

Jetzt weiß der Skript *UserAction* über die Variable *requestUserAction*, dass die Variable *DB_Request_HMI_requestLoop* bei jedem Aufruf verändert werden muss, bis request*UserAction* auf *False* steht.

Wird die linke Maustaste wieder losgelassen (**Punkt 2**), so wird durch das Ereignis die Variable *requestUserAction* wieder auf *False* gesetzt und im Skript die Variable *DB_Request_HMI_requestLoop* nicht mehr verändert, sodass der Loop beendet wird.

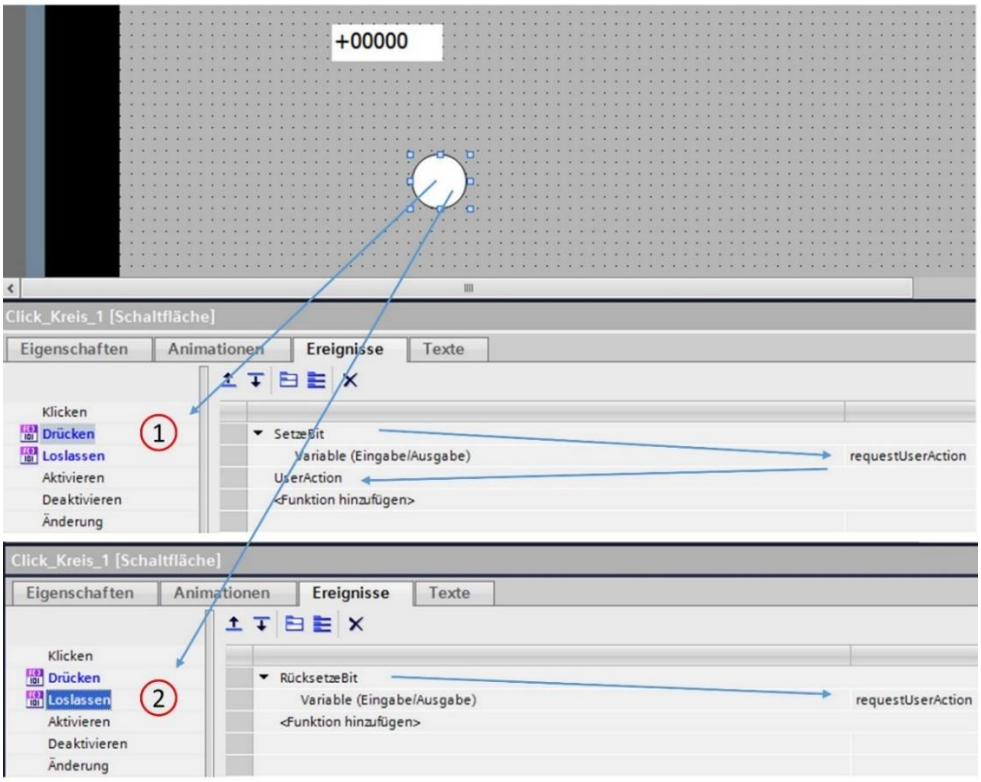


Bild 3.3 Ereignisse durch Mausklick

Betrachten wir nun den Skript *UserAction* in **Bild 3.4**. Mit Drücken der linken Maustaste ist die Variable *requestUserAction* auf *True* gesetzt (**Punkt 1**). So ist das *IF-Statement* in Zeile 4 erfüllt und es wird **Punkt 2** ausgeführt. Die Variable *DB_Request_HMI_requestLoop* wird um eine Ziffer erhöht (*Inkrement*). Da sich dieser Vorgang innerhalb einer Sekunde mindestens 500-mal wiederholt, wird die Variable auf die Zahl 5000 begrenzt um dann wieder mit Null zu beginnen (**Punkt 3**). Danach wird ein Skript (im späteren Buchverlauf sichtbar) aufgerufen (Zeile 11) und der Skript wird verlassen. Die Zahl 5000 ist beliebig gewählt worden.

Nach dem Loslassen der linken Maustaste wird die Variable *requestUserAction* auf *False* gesetzt (**Bild 3.3**, *Rücksetze Bit*). Das *IF-Statement* in Zeile 4 verzweigt in den *Else-Teil* nach Zeile 14 und die Variable *DB_Request_HMI_requestLoop* wird (nur der ordnungshalber) auf Null gesetzt (**Punkt 4**). Danach wird der Skript nicht mehr aufgerufen, da es keine Wertänderung mehr gibt und der Loop ist beendet.

```
1  Sub UserAction()
2
3  ' event loop
4    If(SmartTags("requestUserAction") ) Then
5
6    SmartTags("DB_Request_HMI_requestLoop") = _
7                    SmartTags("DB_Request_HMI_requestLoop") + 1
8      If( SmartTags("DB_Request_HMI_requestLoop") > 5000 )Then
9        SmartTags("DB_Request_HMI_requestLoop") = 0
10     End If
11
12     'DO anything
13
14   Else
15     SmartTags("DB_Request_HMI_requestLoop") = 0
16   End If
17
18 End Sub
```

Bild 3.4 Skript "UserAction"

In **Bild 3.5** ist bei gedrückter Maustaste die Variable *DB_Request_HMI_requestLoop* gerade auf 1129 hochgezählt. Wird die Maustaste losgelassen, hat die Variable wieder den Wert Null.

Bild 3.5 Der Loop bei gedrückter und nicht gedrückter Maustaste

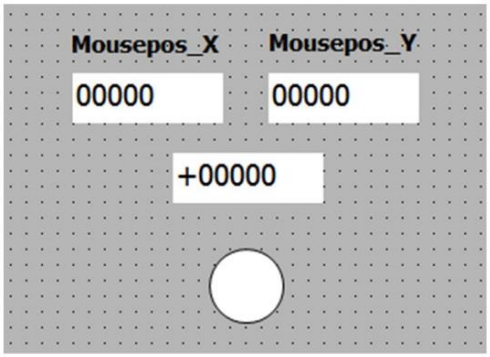
3.2 Die Mauskoordinaten anzeigen

In Skript *UserAction* wird nun der Kommentar *Do anything* in Zeile 11 durch den Skript *GetMousePosition* ersetzt. Zunächst soll die Verarbeitung und Anzeige der Mausposition vorgestellt werden, um erst danach Objekte zu bewegen. Im TIA-Projekt zu diesem Kapitel wird das *Grundbild* um die Anzeige der Maus-Koordinaten erweitert.

In **Bild 3.6** zu sehen, wurde die Position der Maus für die X- und Y-Koordinate hinzugefügt. Damit diese auch angezeigt werden, ist die Variablentabelle erweitert worden.

Im Skript *UserAction* (**Bild 3.7**) wird nun die Skriptfunktion *GetMousePosition* aufgerufen (**Punkt 1**). Diese liefert bei Erfolg ein *True* und verzweigt dann in das *IF-Statement* nach Zeile 12. Hier gibt es vorerst ein weiteres *Do anything.*

Wie in der Projektnavigation (**Punkt 3**) zu sehen (Gruppe *MouseProgram*), wurden die Skripte *GetMousePosition*, *RequestMousePos* und *CheckRequestHMI* hinzugefügt. Die Skripte *CheckRequestHMI* und *RequestMousePos* werden von *GetMousePosition* aufgerufen. Diese Skripte überprüfen, ob neue Mausdaten ordnungsgemäß registriert wurden.



HMI_UserAction

Name	Datentyp	Verbindung
DB_Request_HMI_requestLoop	Int	HMI_Verbindung_1
requestUserAction	Bool	<Interne Variable>
requestMousePos	Bool	<Interne Variable>
userActionActiv	Bool	<Interne Variable>
statusGetMousePosition	Int	<Interne Variable>
timeOutGetMousePos	Int	<Interne Variable>
mouseXpos	Int	<Interne Variable>
mouseYpos	Int	<Interne Variable>

Bild 3.6 Mouse-Position im Grundbild hinzufügen

```
1  Sub UserAction()
2
3  ' event loop
4    If(SmartTags("requestUserAction") ) Then
5
6      SmartTags("DB_Request_HMI_requestLoop") = _
7                  SmartTags("DB_Request_HMI_requestLoop") + 1
8      If( SmartTags("DB_Request_HMI_requestLoop") > 5000 )Then
9        SmartTags("DB_Request_HMI_requestLoop") = 0
10     End If
11
12     If( GetMousePosition ) Then                    (1)
13       'Do anything
14     End If
15   Else 'No request bit
16     SmartTags("statusGetMousePosition") = 0
17     SmartTags("userActionActiv") = 0                    (2)
18     SmartTags("DB_Request_HMI_requestLoop") = 0
19   End If
20
21 End Sub
```

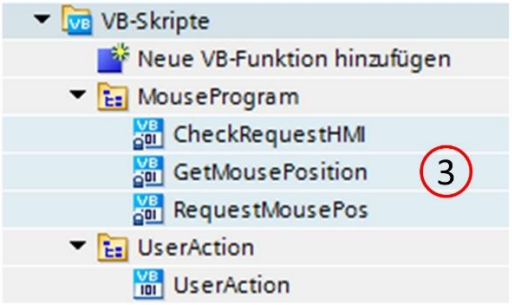


Bild 3.7 *UserAction* wird erweitert

Im Else-Teil (**Punkt 2**) wurden 2 interne Variablen hinzugefügt, welche erst später eine Bedeutung bekommen. Wird dieses Projekt komplett übersetzt und gestartet, dann wird durch den Skript *GetMousePosition* die aktuelle Mausposition in die internen Variablen *mouseXpos* und *mouseYpos* gespeichert (**Bild 3.6**).

Es muss also ein *Maus-Add-on* gestartet werden, damit die Mausposition aus dem Betriebssystem gelesen und in HMI-Variablen geladen wird.

Die Mauskoordinaten werden nur angezeigt, wenn das *Maus-Add-on* gestartet wurde. Dazu muss auf dem verwendeten PC im Pfad:

C:/TIA-Expert/MoveObjects/MoveObjects_Basic/Request

das Add-on ***MausAddOn.EXE*** bereits gestartet worden sein!

Das TIA-Projekt ***MoveObjects_Basic* muss** unter den o.g. Pfad kopiert sein. Zumindest muss die Datei ***MausAddOn.EXE*** im Verzeichnis *../Request* vorhanden sein, denn dort werden die Mausdaten ausgetauscht. Die zum Buch gelieferte Software und die geschützten Skripte stellen eine Basis-Software dar und können vom Leser beliebig genutzt werden.

Das Programm *MausAddOn.EXE* ist in dieser Starter-Version begrenzt einsetzbar. Es zählt die Mausbewegungen und beendet sich nach einer festgelegten Anzahl automatisch. Danach muss das Programm neu gestartet werden. Sonst gibt es keine weiteren Einschränkungen. Auf der Webseite www.*tia-expert.com* kann ein TIA-Projekt kostengünstig ohne geschützte Skripte erworben werden.

Jedoch wird in **Kapitel 11** eine C++-Schnittstelle vorgestellt, die es ermöglicht, dass der Leser sich seine eigene Schnittstelle programmieren kann.

Nachdem das TIA-Projekt zu diesem Kapitel fehlerfrei komplett neu übersetzt und der Treiber ***MausAddOn.EXE*** gestartet wurde, werden die Maus-Koordinaten kontinuierlich über *GetMousePosition* in die internen Variablen *mouseXpos* und *mouseYpos* kopiert.

Achten Sie darauf, dass auch PLCSIM gestartet wurde!

Mit dem Click auf die Schaltfläche *Click_Kreis_1* wird der Skript *UserAction* gestartet und die Maus-Koordinaten angezeigt (**Bild 3.8**). Im Bild sind die momentanen Koordinaten zu sehen. Was nicht erfolgt, ist die Bewegung des Kreis-Objektes, denn das ist ja noch gar nicht programmiert und wird in **Kapitel 3.3** vorgestellt.

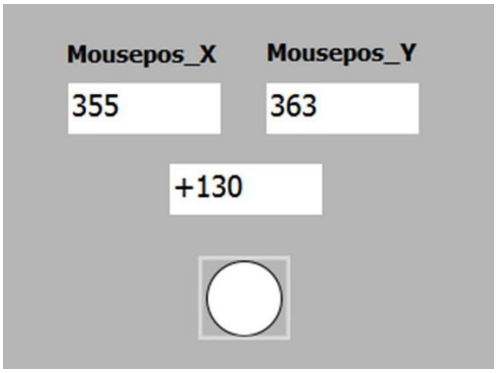


Bild 3.8 Die Maus-Koordinaten werden angezeigt

3.3 Das Objekt Kreis bewegen

Das TIA-Projekt zu diesem Kapitel zeigt, wie das Bildobjekt Kreis mit seiner Schaltfläche über die Mauskoordinaten auf dem Bediengerät (Panel) bewegt werden kann. Wie in **Kapitel 3.2** erläutert wurde, werden die Maus-Koordinaten in den Variablen *mouseXpos* und *mouseYpos* gespeichert. Nun besteht die Aufgabe darin, im zyklischen Aufruf des Skriptes *UserAction* die X- und Y-Position des Kreises an die Maus-Koordinaten anzupassen, damit sich das Bildobjekt synchron mit der Maus bewegt.

Da die Maus-Koordinaten aus dem Betriebssystem keinen Bezug auf das Bildobjekt im TIA-Projekt haben, muss die momentane Position der Maus mit dem Start der linken Maustaste gespeichert und der Offset zum Kreis-Objekt berücksichtigt werden. Es reicht nicht aus, die Maus-Koordinaten aus den Variablen *mouseXpos* und *mouseYpos* in die Koordinaten des Bildobjektes zu kopieren.

In **Bild 3.9** ist die Programmierung des zyklischen Ablaufes als Komponentenplan dargestellt. In der Komponente *UserAction* wurde der Script *SelectObject* hinzugefügt. Der zyklische *Event*, wie in **Kapitel 3.2** vorgestellt wurde, ruft bei Übergabe der korrekten Mausdaten den Skript *SelectObject* auf. Hier werden nun die Objekte selektiert, welche zum Zeitpunkt des Maus-*Events* bewegt werden sollen (**Bild 3.10**, **Punkt 1**).

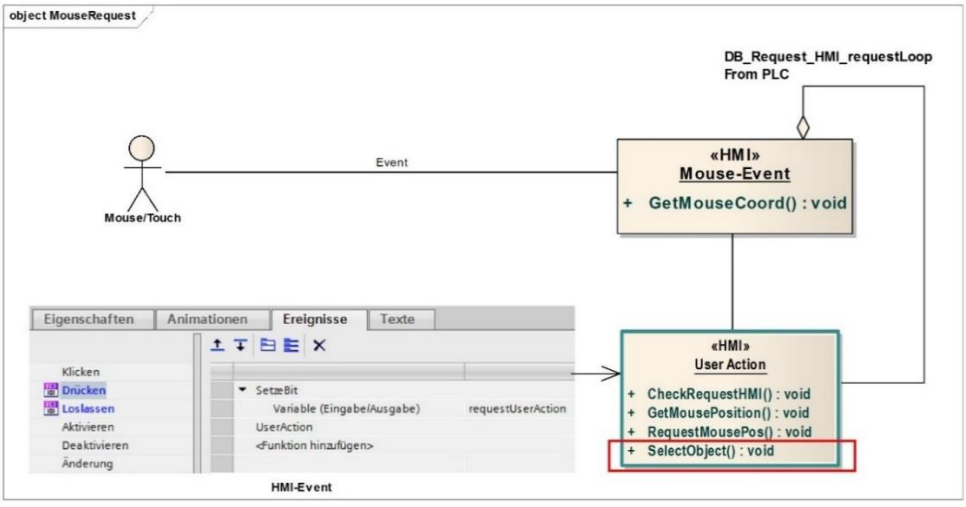


Bild 3.9 Komponentenplan zum Mouse-Event

```
1  Sub UserAction()
2
3  ' event loop
4    If(SmartTags("requestUserAction") ) Then
5
6      SmartTags("DB_Request_HMI_requestLoop") = _
7                SmartTags("DB_Request_HMI_requestLoop") + 1
8      If( SmartTags("DB_Request_HMI_requestLoop") > 5000 )Then
9        SmartTags("DB_Request_HMI_requestLoop") = 0
10     End If
11
12     If( GetMousePosition ) Then
13       SelectObject
14     End If
15   Else 'No request bit
16     SmartTags("statusGetMousePosition") = 0
17     SmartTags("userActionActiv") = 0
18     SmartTags("DB_Request_HMI_requestLoop") = 0
19   End If
20
21 End Sub
```


Bild 3.10 SelectObject wird zyklisch über den Maus-Event aufgerufen

3.3.1 Der Skript *SelectObject*

Die Objekte, welche bewegt werden sollen, erhalten eine Objekt-Nummer, welche den Objekt-Typ unterscheiden soll (Kreise, Vierecke, Motoren, Ventile usw.). Auch wird ein Datentyp für die Anzahl der Objekte festgelegt, da auch mehrere Objekte existieren können.

Damit der Skript die Mauskoordinaten an das entsprechenden Objekt zuordnen kann, muss das Objekt in der SPS vorhanden sein.

Zuerst wird ein Datentyp *UDT_CIRCLE* (**Bild 3.11**, **Punkt 2**) und ein DB von diesem Datentyp (**Punkt 1**) erstellt. Da es mehrere Datentypen geben soll und wir den Datentyp klassifizieren müssen, legen wir die Konstanten in *PLC-Variablen* fest (**Punkt 4**). Hier ist folgendes zu bemerken:

Das Array *of UDT_CIRCLE* wird mit den Konstanten *MIN* und *MAX_CIRCLE* deklariert (**Punkt 3**). In *PLC-Variablen/Anwenderkonstanten* werden in diesem Beispiel die Array-Konstanten immer so angelegt, wie diese auch in der Programmierung verwendet werden.

Bei zwei Kreisobjekte wird also nicht die Konstante *MAX_CIRCLE* auf 2 gesetzt, sondern auf 1. Wie im DB zu erkennen (**Punkt 3**) besteht das Array aus dem Index 0 und 1. Das ist dann auch in der Programmierung so. Man spart sich so die Indizierung mit z. B. ***MAX_CIRCLE-1*** und kann so direkt mit der Konstanten arbeiten (z.B. bei einer Schleifenbildung).

Falsche Indizierung führt zum Stop der PLC. Das Risiko kann durch die Anwendung und Festlegung über Konstanten verringert werden.

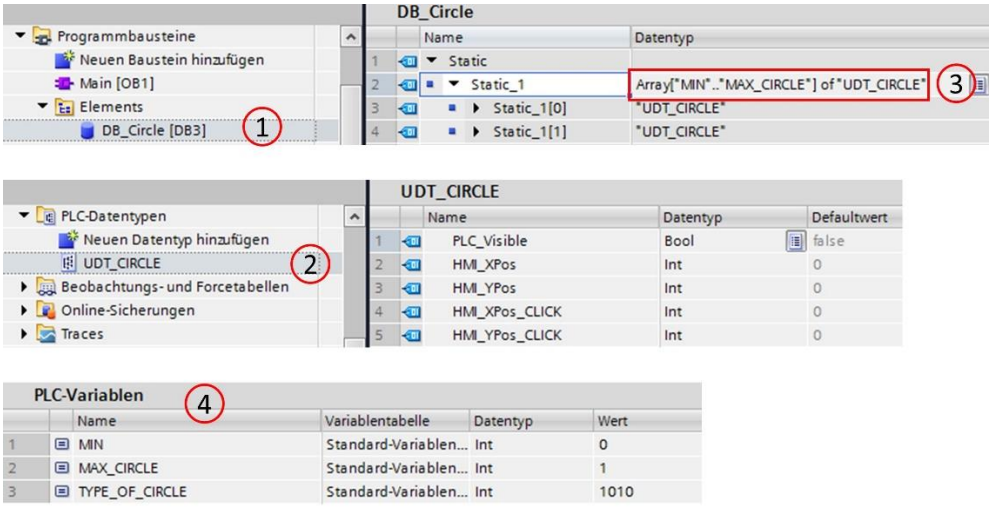


Bild 3.11 Baustein in der PLC anlegen

Der Aufruf für *UserAction* muss nun spezifiziert werden, wenn auf den Kreis ein Click erfolgt.

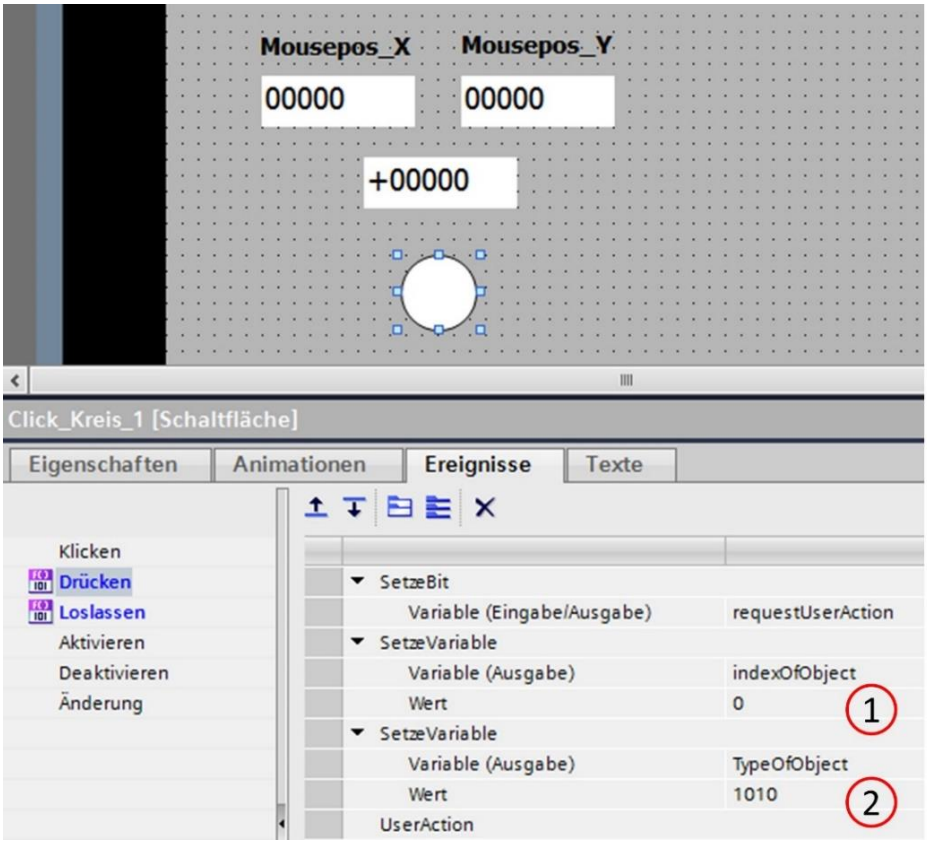


Bild 3.12 Der Aufruf *UserAction*

In **Bild 3.12** ist der Aufruf und das Ereignis zum Kreis ersichtlich. Die neue Variable *indexOfObject* wird auf 0 gesetzt, da es sich um den ersten Kreis im Grundbild handelt. Der Objekt-Typ mit der neuen Variablen *TypeOfObject* wird auf den Wert 1010 gesetzt. Hier kann leider nicht die Konstante (**Punkt 4** in **Bild 3.11**) verwendet werden (TIA-Version V13 SP1 UPD 8). Am Ende der Liste wird nun *UserAction* aufgerufen und damit auch über den Skript *UserAction* der Skript *SelectObject*.

In der PLC-Variablentabelle (**Bild 3.13** oben) werden die neuen Variablen angelegt. Die bereits erwähnten Offsets für die Mauskoordinaten X und Y sind als Array deklariert und erlauben somit insgesamt 32 Objekte auf einmal zu bewegen.

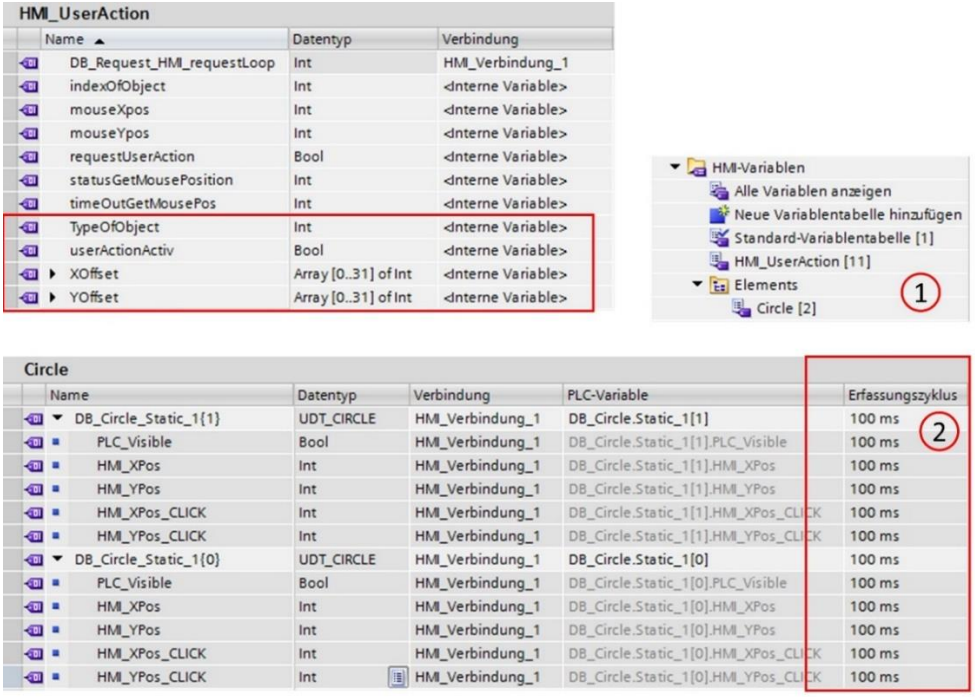


Bild 3.13 Erweiterung der PLC-Variablen

Für den Zugriff auf den *DB_Circle* (**Bild 3.11**, **Punkt 1**) wird in *HMI-Variablen-Circle* der *DB* unter der Gruppe *Elements* deklariert. Da diese Variablen zyklisch sehr schnell geändert werden, ist darauf zu achten, dass die Erfassungszeit je Zyklus auf 100 ms steht (**Punkt 2**).

Im **Bild 3.14** ist der Skript *SelectObject* bis Zeile 21 dargestellt. Im ersten Abschnitt werden die *Screen-Limits* berücksichtigt, damit das Objekt nicht über die erlaubten Grenzen geschoben werden kann (Zeilen 3-20).

Wird dem Objekt eine X- bzw. Y-Koordinate zugewiesen, welche außerhalb der Screen-Limits liegt, kommt es zu unerwünschten Konflikten in WinCC.

```
1  Sub SelectObject()
2
3  Const BorderX = 2 ' 2% from the screen width
4  Const BorderY = 2 ' 2% from the screen height
5
6  Dim ObjectScreen_Width, ObjectScreen_Height
7  Dim ObjectScreen_Left, ObjectScreen_Top
8  Dim Object, ClickObject
9
10 'Screen limits
11   ObjectScreen_Width = HmiRuntime.Screens("Grundbild").Width _
12                      - ((HmiRuntime.Screens("Grundbild").Width _
13                      / 100) * BorderX)
14   ObjectScreen_Height = HmiRuntime.Screens("Grundbild").Height _
15                      - ((HmiRuntime.Screens("Grundbild").Height / 100)_
16                      * BorderY)
17   ObjectScreen_Left = (HmiRuntime.Screens("Grundbild").Width / 100)_
18                       * BorderX
19   ObjectScreen_Top = (HmiRuntime.Screens("Grundbild").Height / 100)_
20                      * BorderY
21
```

Bild 3.14 Der Skript *SelectObject-Teil1*

Mit der *Select Case SmartTags("TypeOfObject")* Anweisung in Zeile 23 (**Bild 3.15**) kann nun der Objekt-Typ erkannt werden. Im Beispiel war das im Aufruf die Typbezeichnung 1010 als Konstante für den Kreis und dem Index 0 für die Variable *indexOfObject*.

Die aktuellen Werte für die X- und Y-Koordinaten werden aus der *HMI* in den *DB* gespeichert (Zeilen 32 bis 37). Da nicht nur das Bild *Kreis_1* sondern auch die Schaltfläche *Click_Kreis_1* verschoben werden sollen, müssen alle beteiligten Objekte kopiert werden.

Erst danach wird der Script *Move2Objects* in Zeile 43 aufgerufen. Dieser Skript verschiebt die angegebenen Objekte *Object* und *ClickObject* innerhalb der erlaubten Grenzen.

Das Schreiben der Objekt-Koordinaten in den *DB* ist nur deswegen notwendig, damit durch einen Bildwechsel in der *HMI* die Objekte wieder über die Daten der SPS korrekt gezeichnet werden können. Denn sonst zeichnet *WinCC* die Objekte in der Position, in der diese ursprünglich in der Entwicklungsumgebung gezeichnet wurden.

Das ist eine wichtige Erkenntnis, welche noch im weiteren Buchverlauf erwähnt wird. Im folgenden Kapitel betrachten wir nun wie der Skript *Move2Objects* funktioniert.

```
'call user functions
  Select Case SmartTags("TypeOfObject")

    Case 1010 'Circle
      Select Case SmartTags("indexOfObject")
        Case 0
          Set Object = HmiRuntime.Screens("Grundbild")._
                                  ScreenItems("Kreis_1")
          Set ClickObject = HmiRuntime.Screens("Grundbild")._
                                  ScreenItems("Click_Kreis_1")
          SmartTags("DB_Circle_Static_1{0}.HMI_XPos") = Object.Left
          SmartTags("DB_Circle_Static_1{0}.HMI_YPos") = Object.Top
          SmartTags("DB_Circle_Static_1{0}.HMI_XPos_CLICK")_
                      = ClickObject.Left
          SmartTags("DB_Circle_Static_1{0}.HMI_YPos_CLICK")_
                      = ClickObject.Top
        Case 1
          'and so on
      Case Else
        Exit Sub
      End Select 'indexOfObject
      Move2Objects  Object,ClickObject,ObjectScreen_Width,_
                    ObjectScreen_Left,ObjectScreen_Height,_
                    ObjectScreen_Top
      Exit Sub
  Case Else
    Exit Sub
  End Select 'TypeOfObject

End Sub
```

Bild 3.15 Die Zuordnung der Objekt-Daten für die SPS –Teil 2

3.3.2 Der Skript *Move2Objects*

In **Kapitel 3.3.1** wird der Skript *SelectObject* vorgestellt. Dort besteht das Kreis-Objekt aus 2 Elementen (Kreisbild und Schaltfläche). Beide Objekte müssen mit den Mauskoordinaten nachgeführt werden. Bei 3 Objekten muss der entsprechenden Skript *Move3Objects* erstellt werden.

```
Sub Move2Objects(ByRef Object_0, ByRef Object_1, ByRef Max_X, ByRef Min_X, By
' const and dim
  Dim X_Obj_0, Y_Obj_0, X_Obj_1, Y_Obj_1

' first call init (1)
  If( Not SmartTags("userActionActiv") ) Then
    SmartTags("XOffset")(0) = (SmartTags("mouseXpos") - Object_0.Left)
    SmartTags("YOffset")(0) = (SmartTags("mouseYpos") - Object_0.Top)
    SmartTags("XOffset")(1) = (SmartTags("mouseXpos") - Object_1.Left)
    SmartTags("YOffset")(1) = (SmartTags("mouseYpos") - Object_1.Top)
    SmartTags("userActionActiv") = True
    Exit Sub
  End If

' control move (2)
  X_Obj_0 = SmartTags("mouseXpos") - SmartTags("XOffset")(0)
  X_Obj_1 = SmartTags("mouseXpos") - SmartTags("XOffset")(1)
  If( (X_Obj_0 < (Max_X - Object_0.Width)) And (X_Obj_0 > Min_X) )Then
    If( (X_Obj_1 < (Max_X - Object_1.Width)) And (X_Obj_1 > Min_X) )Then
      Object_0.Left = X_Obj_0
      Object_1.Left = X_Obj_1
    End If
  End If

  Y_Obj_0 = SmartTags("mouseYpos") - SmartTags("YOffset")(0)
  Y_Obj_1 = SmartTags("mouseYpos") - SmartTags("YOffset")(1)
  If( ((Y_Obj_0 + Object_0.Height) < Max_Y) And (Y_Obj_0 > Min_Y) )Then
    If( ((Y_Obj_1 + Object_1.Height) < Max_Y ) And (Y_Obj_1 > Min_Y) )Then
      Object_1.Top = Y_Obj_1
      Object_0.Top = Y_Obj_0
    End If
  End If

End Sub
```

Bild 3.16 Der Skript *Move2Objects*

Betrachten wir den Skript *Move2Objects* in **Bild 3.16**, dann ist dieser in zwei Abschnitte unterteilt. Im ersten Abschnitt *first call init* (**Punkt 1**) werden die Mauskoordinaten in die interne Variable *XOffset* und *YOffset* gespeichert. Im Abschnitt *control move* (**Punkt 2**) werden die Mauskoordinaten mit dem Offset verrechnet und in die Objekte unter Berücksichtigung der *Screen-Limits* geschrieben. Das Objekt bewegt sich nun in RT, wie im **Bild 3.17** zu sehen ist.

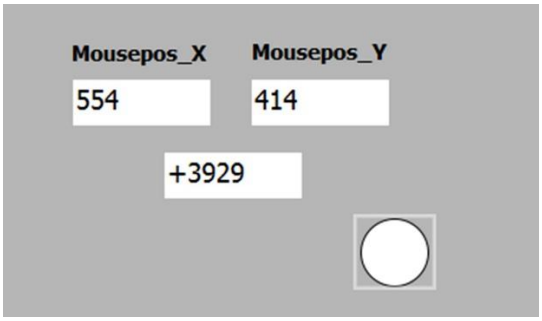


Bild 3.17 Das Bildobjekt wird mit der Maus verschoben

In **Bild 3.18** sind die bis jetzt notwendigen Skripte dargestellt, die eine Basis in WinCC bilden, um Objekte in RT mit der Maus zu bewegen.

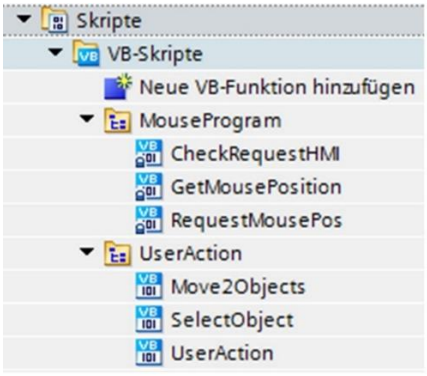


Bild 3.18 Basis-Skripte zu Objekte bewegen mit der Maus

3.3.2.1 Die Mauskoordinaten anpassen

Zum Listing *Move2Objects* gibt es noch einiges zu erklären. In **Bild 3.19** sind die Maus-Koordinaten im Verhältnis zu den Objekt-Koordinaten in einem Beispiel dargestellt. Würde man die Maus-Koordinaten direkt in das Bildobjekt kopieren, wird das Bildobjekt einen Sprung durchführen. Aus diesem Grund müssen die Maus-Koordinaten zu den Koordinaten des Bildobjektes in *WinCC* durch dessen Differenz angepasst werden. D. h. der Maus-Cursor wird gespeichert und die Differenz zum Objekt-Cursor berücksichtigt.

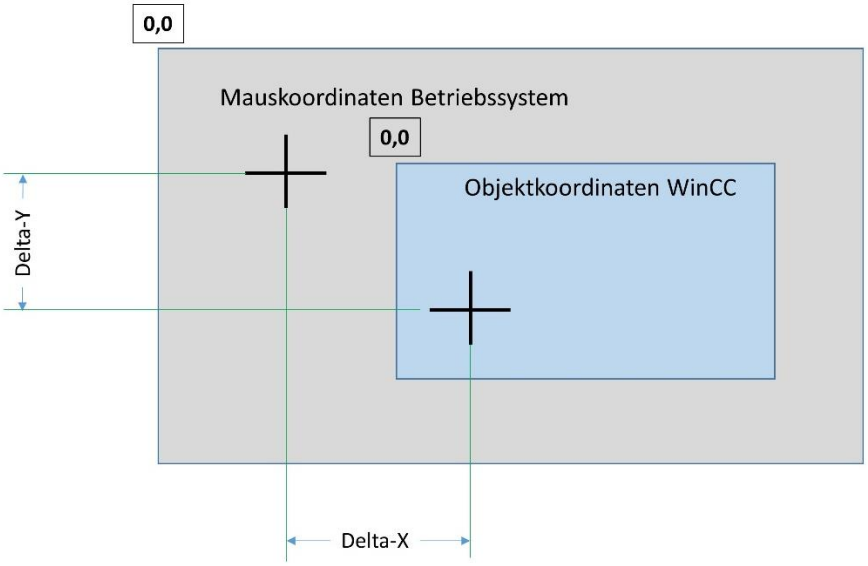


Bild 3.19 Die Mauskoordinaten sind nicht identisch

In **Bild 3.20** wird die interne Variable *XOffset (0)* für *Object_0.Left* genutzt. Hier handelt es sich um das Bildobjekt *Kreis_1* (Zeile 23, **Bild 3.15**), welches als Übergabeparameter in Object_0 gespeichert ist. Der so errechnete Offset mit *(SmartTags(„mouseXpos“) – Object_0.Left)* wird in *(SmartTags(„XOffset“)(0)* gespeichert. Das gilt analog dazu auch für die Y-Koordinate aus *Object_0*.

Diese Differenz, zwischen dem Maus-Cursor im Betriebssystem und der tatsächlichen Koordinate in Monitor *WinCC*, wird danach vom Maus-Cursor abgezogen, bevor die Zuordnung des Maus-Cursors zum Objekt eingetragen wird.

```
5  ' first call init
6  If( Not SmartTags("userActionActiv") ) Then
7    SmartTags("XOffset")(0) = (SmartTags("mouseXpos") - Object_0.Left)
8    SmartTags("YOffset")(0) = (SmartTags("mouseYpos") - Object_0.Top)
9    SmartTags("XOffset")(1) = (SmartTags("mouseXpos") - Object_1.Left)
10   SmartTags("YOffset")(1) = (SmartTags("mouseYpos") - Object_1.Top)
11   SmartTags("userActionActiv") = True
12   Exit Sub
13 End If
```

Bild 3.20 Offset zur HMI-Situation bestimmen

Danach wird mit der Mausbewegung nur noch die Differenz zum Bild-Objekt übertragen. Analog gilt das für alle Objekte, wie hier im Beispiel der Kreis und seine Schaltfläche. Die Variable *userActionActiv* verhindert beim folgenden Aufruf des Skriptes, dass die

Mausposition erneut in den *Offset* kopiert wird. Zurückgesetzt wird *userActionActiv*, wenn die linke Maustaste wieder losgelassen wird (siehe **Bild 3.7**, **Punkt 2**).

```
15 ' control move
16   X_Obj_0 = SmartTags("mouseXpos") - SmartTags("XOffset")(0)
17   X_Obj_1 = SmartTags("mouseXpos") - SmartTags("XOffset")(1)
18   If( (X_Obj_0 < (Max_X - Object_0.Width)) And (X_Obj_0 > Min_X) )Then
19     If( (X_Obj_1 < (Max_X - Object_1.Width)) And (X_Obj_1 > Min_X) )Then
20       Object_0.Left = X_Obj_0
21       Object_1.Left = X_Obj_1
22     End If
23   End
```

Bild 3.21 Koordinaten zum Objekt eintragen

In den Zeilen 20 und 21 in **Bild 3.21** wird nun die Differenz des neuen Maus-Cursors zum gespeicherten Offset (Zeilen 16, 17) unter Berücksichtigung von *Max*_X und *Max*_Y in das Objekt eingetragen. Damit bewegt sich das Objekt entsprechend der Positions-Differenz zum Maus-Cursor.

Die Objekt-Koordinaten sind in WinCC gespeichert. Findet jedoch ein Bildwechsel in das Bild mit dem Bildobjekt statt, gehen die Bild-Koordinaten leider verloren und es werden wieder die ursprünglichen Bild-Koordinaten eingetragen! (siehe auch **Kapitel 2.1.2**)

Wie die Objekte bei einem Bildwechsel wieder ordnungsgemäß gezeichnet werden, zeigt das folgende Kapitel.

4 Objekte nach einem Bildwechsel neu zeichnen

In **Kapitel 3** wurde gezeigt, wie Bild-Objekte mit der Maus in RT bewegen werden können. Aus dieser neuen Idee entstehen nun viele Möglichkeiten derer Anwendungen, wie z.B. in der chemischen Industrie zum Zeichnen eines P&ID[4] oder die Erstellung eines GRAFCET-Plan. In der praktischen Umsetzung gibt es allerdings kleine Probleme, welche das WinCC so mit sich bringt. Die Entwickler von WinCC haben damals sicherlich nicht daran gedacht, das Bediengerät wie den Monitor von einem Smartphone zu betrachten?

Bei einem Bildwechsel werden die Objekte wieder so gezeichnet (siehe auch **Kapitel 2.1.2**), wie diese ursprünglich in der Entwicklungsumgebung TIA Portal erstellt wurden. D.h. alle Bewegungen der Objekte gehen verloren?

Demnach müssen die Bildobjekte nach einem Bildwechsel mit den letzten X-Y-Koordinaten aus der SPS neu gezeichnet werden. Das ist auch der Grund, warum die Koordinaten der Bildobjekte in einem DB gespeichert wurden.

Eine weitere, kleine Hürde besteht noch, falls große Bildobjekte wie z.B. Behälter oder viele Elemente auf einer Bildseite vorhanden sind, denn WinCC benötigt ein wenig Zeit die Bilder zu zeichnen um diese neu darzustellen.

Auch wenn die Bildobjekte neu gezeichnet werden, sind diese mit den alten Koordinaten noch eine kurze Weile im Monitor sichtbar!

Zuerst schaltet WinCC auf das Bild um und die Bildobjekte werden sofort mit den ursprünglichen Koordinaten sichtbar. Das führt dann zu einem kurzen „Flackern“, bis alle Bildobjekte mit den neuen Koordinaten gezeichnet sind.

4.1 Der Skript *InitObjects* und das MausAddOn.EXE

Achten Sie darauf, das Maus-Add-on **MausAddOn.EXE** zu starten**,** sonst wird der Skript *UserAction* keine Mausdaten bekommen!

(***C:/TIA-Expert/MoveObjects/MoveObjects_Basic/Request*).**

Das folgende TIA-Projekt zu diesem Kapitel zeigt die Anwendung zur Initialisierung der Bildobjekte nach einem Bildwechsel in der HMI, mit den gespeicherten Koordinaten aus der SPS. Der Skript *InitObjects* kann über den Aufgabenplaner mit dem Trigger Bildwechsel aufgerufen werden. **Bild 4.1** zeigt die Einstellungen im Aufgabenplaner. Vorteil des Aufgabenplaners ist, dass sich der Programmierer nicht darum kümmern muss, wann *InitObjects* aufgerufen werden muss.

[4] [P&ID] Piping and Instrumentation Diagram

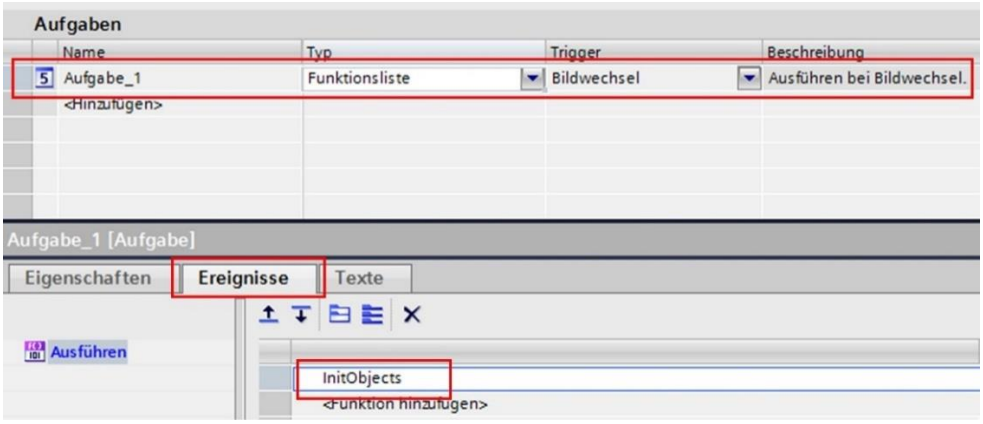


Bild 4.1 In Aufgabenplaner den Bildwechsel mit Skript *InitObjects* aktivieren

Nachteil ist, dass auch bei einem Bildwechsel in Bildseiten ohne Bildobjekte, welche mit der Maus verschoben worden sind, der Skript ebenfalls unnötigerweise aufgerufen wird.

Auch liegt der Gedanke nahe, den Skript *InitObjects* in der Bildseite direkt über die *Skriptliste* mit dem Ereignis *Aufgebaut* aufzurufen. Das erscheint sinnvoller, da keine Ressourcen bei jedem Bildaufbau verbraucht werden (**Bild 4.2**). Ist aber für den ersten Bildaufruf nach dem Einschalten des PCs nicht geeignet.

Leider gibt es noch ein Problem, egal welche Methode man wählt, denn beim ersten Start von WinCC wird der Bildwechsel oder das Ereignis *Aufgebaut* aufgerufen, ohne dass die Treiber schon die Daten der SPS initialisiert haben. Das hat zur Folge, dass beim ersten Bildaufruf alle Elemente in der X-Y-Position „0,0“ gezeichnet werden.

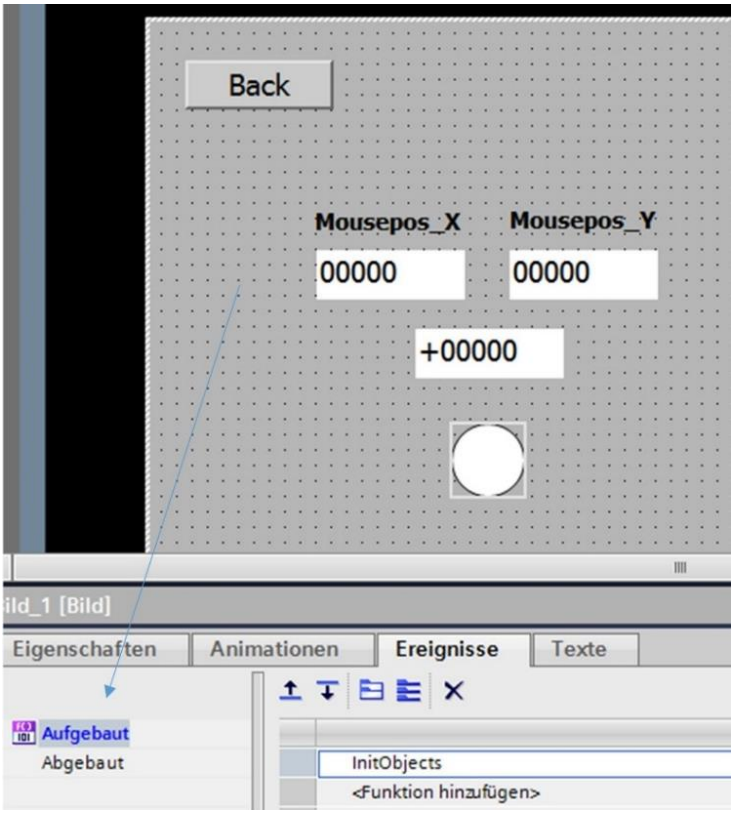


Bild 4.2 Über das Ereignis des Skript *InitObjects* aufrufen

Um hier nicht viel Aufwand mit einer automatischen Korrektur zu betreiben, ist es sinnvoll, Bildseiten mit beweglichen Objekten **nicht** in das Grundbild zu zeichnen. Aus diesem Grund wird im Grundbild nur die Schaltfläche für den Bildwechsel nach *Bild _1* eingefügt (**Bild 4.3**) und im *Bild_1* können wir dann mit der Schaltfläche *Back* wieder zurück in das Grundbild schalten.

Außerdem wird der Skript *InitObjects* über den Aufgabenplaner aufgerufen.

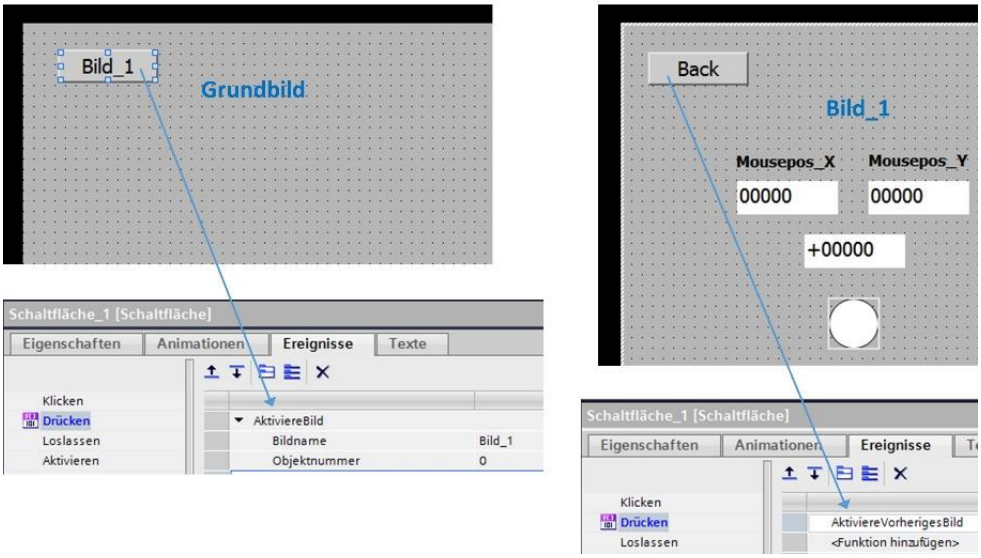


Bild 4.3 Bildwechsel zum Test für den Skript *InitObjects*

Betrachten wir nun den Skript *InitObjects* in **Bild 4.4**, dann fällt zuerst auf, dass das *IF-Statement* in den Zeilen 5 und 14 noch kommentiert sind. Alle Bildobjekte werden normalerweise erst dann sichtbar, wenn diese über ein Menü auf den Monitor gezogen werden. Beim Bildaufbau ist es demnach nur sinnvoll, Bildobjekte neu zu zeichnen, wenn diese sichtbar sind. Das ist hier noch nicht berücksichtigt und wird in den Kapiteln zu den Anwendungen dann noch nachgeholt.

```
1  Sub InitObjects()
2
3  'Circle
4    '0
5      'If(SmartTags("DB_Circle_Static_1{0}.PLC_Visible"))Then
6        HmiRuntime.Screens("Bild_1").ScreenItems("Kreis_1").Left = _
7                              SmartTags("DB_Circle_Static_1{0}.HMI_XPos")
8        HmiRuntime.Screens("Bild_1").ScreenItems("Kreis_1").Top = _
9                              SmartTags("DB_Circle_Static_1{0}.HMI_YPos")
10       HmiRuntime.Screens("Bild_1").ScreenItems("Click_Kreis_1").Left = _
11                             SmartTags("DB_Circle_Static_1{0}.HMI_XPos_CLICK")
12       HmiRuntime.Screens("Bild_1").ScreenItems("Click_Kreis_1").Top = _
13                             SmartTags("DB_Circle_Static_1{0}.HMI_YPos_CLICK")
14     'End If
15
16 End Sub
```

Bild 4.4 Der Skript *InitObjects*

Hinweis: Starten Sie das *Bild_1* in der Entwicklungsumgebung durch die Runtime-Simulation, dann wird der Kreis in die Position „0,0“ gezeichnet. Das gilt dann auch für alle Objekte dieser Gruppe, also Kreis und der Click dazu und hat zur Folge, dass der Click und

der Kreis übereinander liegen, obwohl diese gar nicht so gezeichnet wurden. Dieser Effekt soll uns vorerst nicht stören, da nach dem Freigeben des Kommentares (Zeilen 5 und 14) die Objekte nicht mehr initialisiert werden, da diese unsichtbar sind. Erst nach dem verschieben mit der Maus in den sichtbaren Bereich werden sie initialisiert, sodass in der SPS die Objekt-Daten einzeln, richtig abgelegt werden.

Wird das Kreisbild nun mit der Maus weit von der ursprünglich gezeichneten Position verschoben (**Bild 4.5**) und über einen Bildwechsel wieder neu aufgerufen, dann kann man die alte Position des Kreises kurz erkennen. Im folgenden Kapitel soll dies durch eine Abdeckung vertuscht werden.

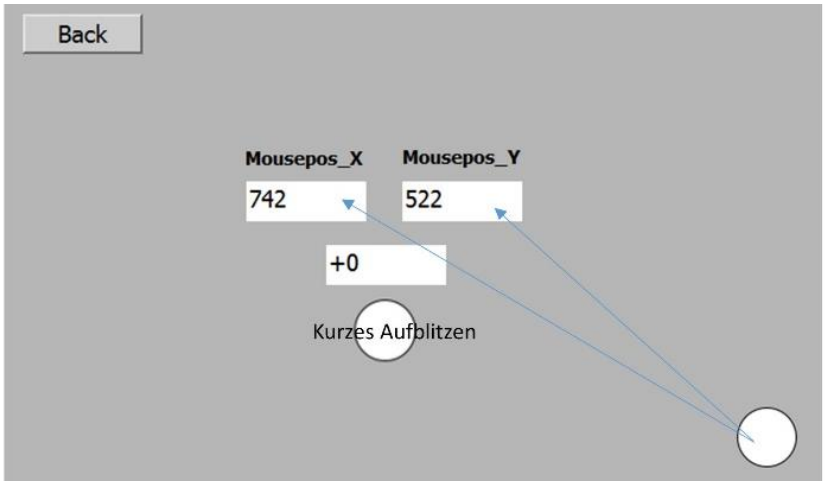


Bild 4.5 Kurzes Aufflackern des Kreises bei einem Bildwechsel

Hinweis: Die Tatsache, dass die Objekte in der Position „0,0“ gezeichnet werden, wenn die SPS neu initialisiert wurde, bleibt bestehen, da der entsprechende *DB* die Werte Null haben wird. Das Problem ist für die jetzige Einzeldarstellung noch tragbar und kann vorerst übersehen werden. Im weiteren Verlauf des Buches werden die gezeichneten Objekte auf den Datenträger gespeichert und erhalten so nach dem Laden der Datei die richtigen Position, auch wenn die SPS neu initialisiert wurde.

4.2 Das Bildflackern verhindern

Wie in **Kapitel 4.1** bereits angesprochen, werden die Bildobjekte nach einem Bildwechsel in der ehemaligen, gezeichneten Position wie in der Entwicklungsumgebung festgelegt, dargestellt. Danach müssen die Bildobjekte mit dem Skript *InitObjects* neu gezeichnet werden. Der Bildaufbau erfolgt demnach zweimal (alte Position und neue Position). Bei genauem Hinsehen fällt dieser Effekt negativ auf und könnte unerwünscht sein. Eine einfache Bildabdeckung (*Cover*) kann hier sehr hilfreich sein. Der *Cover* ist ein Bildobjekt

(Rechteck) und hat die gleiche Hintergrundfarbe wie die Hintergrundfarbe des Bildes und deckt durch die Anordnung in einen höheren Layer alle sichtbaren Bildobjekte ab.

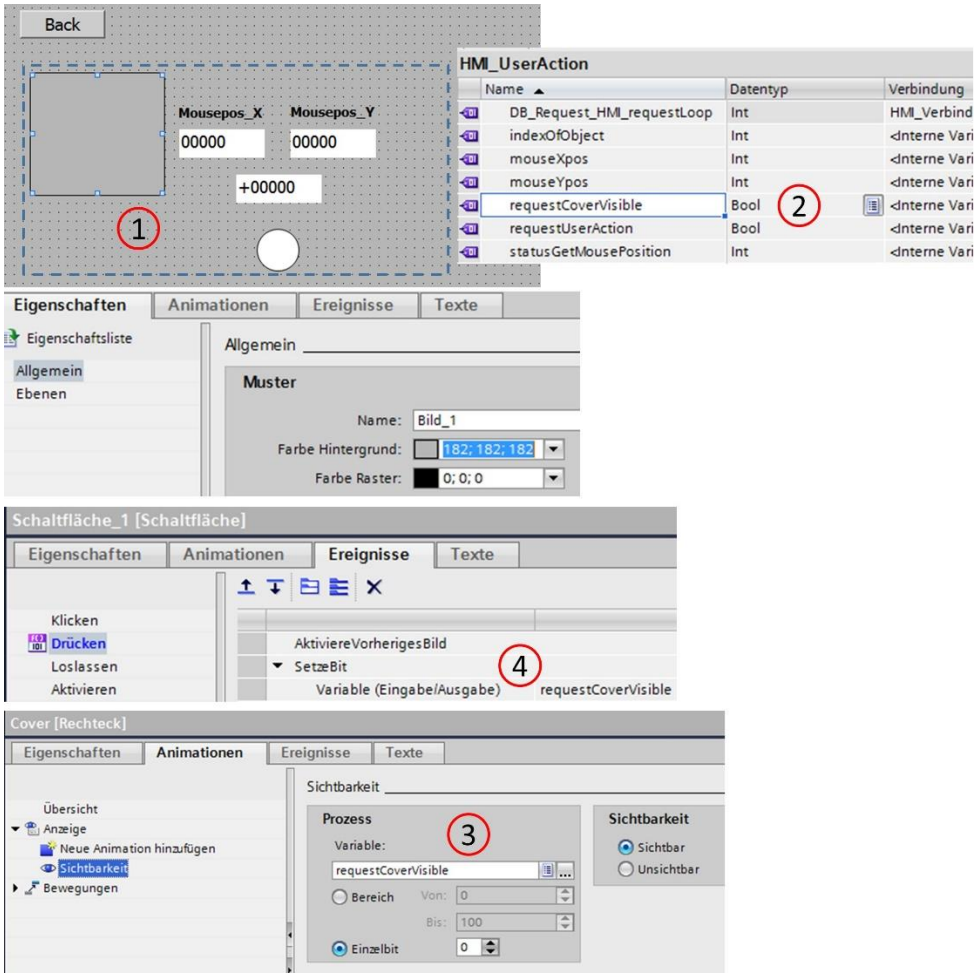


Bild 4.6 Cover deckt bei einem Bildwechsel kurzzeitig die Bild-Objekte ab

Im TIA-Projekt zu diesem Kapitel ist diese Aktion umgesetzt (**Bild 4.6**). Der Cover (**Punkt 1**) wird mit der gleichen Bildfarbe wie das Bild selbst gezeichnet (hier 182,182,182). Das Rechteck bekommt in seinen Eigenschaften die Sichtbarkeit mit der neuen internen Variablen *requestCoverVisible* (**Punkt 2**) zugeordnet (**Punkt 3**)**.**

Im **Bild 4.6** ist der Cover nur zur besseren Sichtbarkeit mit einem Rand und kleiner als die Bildfläche dargestellt. Ob Sie den Cover über das gesamte Bild spannen oder nur über die gezeichneten Objekte, bleibt unbedeutend. Sinnvoll ist eine gesamte Abdeckung über das gesamte Bild.

Die Schaltfläche (*Back*) zum Verlassen des Bildes, wird über ein Ereignis erweitert und setzt die HMI-Variable *requestCoverVisible* auf *True* (**Bild 4.6, Punkt 4**).

In **Bild 4.7** ist der Skript *InitObjects* mit der Erweiterung in Zeile 14 dargestellt. Wenn demnach alle Objekte mit den Daten aus der SPS initialisiert sind, kann der *Cover* wieder unsichtbar geschaltet werden (*requestCoverVisible=False)*. Zudem wurde der Cover auf die Bildebene 30 (**Bild 4.7** unten) gesetzt, damit seine Abdeckung mehr Priorität gegenüber anderen Bildobjekten hat.

Dieses Verfahren funktioniert sehr gut. Alternativ könnte man die HMI-Variable *requestCoverVisible* auch in die SPS verlegen. So hätte man im SPS-Programm für eventuelle Programm-Anwendungen einen automatischen Hinweis, dass das Bild mit den Bildobjekten nicht aktiv ist.

```
1  Sub InitObjects()
2
3  'Circle
4    '0
5      'If(SmartTags("DB_Circle_Static_1{0}.PLC_Visible"))Then
6        HmiRuntime.Screens("Bild_1").ScreenItems("Kreis_1").Left
7        HmiRuntime.Screens("Bild_1").ScreenItems("Kreis_1").Top
8        HmiRuntime.Screens("Bild_1").ScreenItems("Click_Kreis_1'
9        HmiRuntime.Screens("Bild_1").ScreenItems("Click_Kreis_1'
10     'End If
11
12
13   'final clear requestCoverVisible
14   SmartTags("requestCoverVisible")   = False
15
16 End Sub
```

Bild 4.7 Der Skript *InitObjects wurde erweitert*

Nachdem nun der Bildwechsel in das Bild mit bewegten Objekten geklärt ist, soll im nächsten Kapitel das Platzieren der Bildobjekte auf ein Raster vorgestellt werden. Der Grund ist in den Applikationen zu sehen, denn diese fordern meistens ein geordnetes Bild wie z. B. bei Ventilen mit Leitungen oder Steuerelemente mit Verbindungen wie GRAFCET oder

logische Schaltgatter. Auch in der Darstellung von pneumatischen oder hydraulischen Elementen erleichtert eine Rasterung die Zeichenarbeit.

5 Raster als Fangpunkt für Bild-Objekte

Im TIA-Projekt zu diesem Kapitel sollen die Bildobjekte zur Zeichenerleichterung auf ein Raster gelegt werden. Gleichzeit soll dieses Kapitel zeigen, wie Objekte vervielfacht und welche Skripte angepasst werden müssen. Ausgehend vom TIA-Projekt aus **Kapitel 4.2** werden 1 Kreis und 2 Rechtecke hinzugefügt.

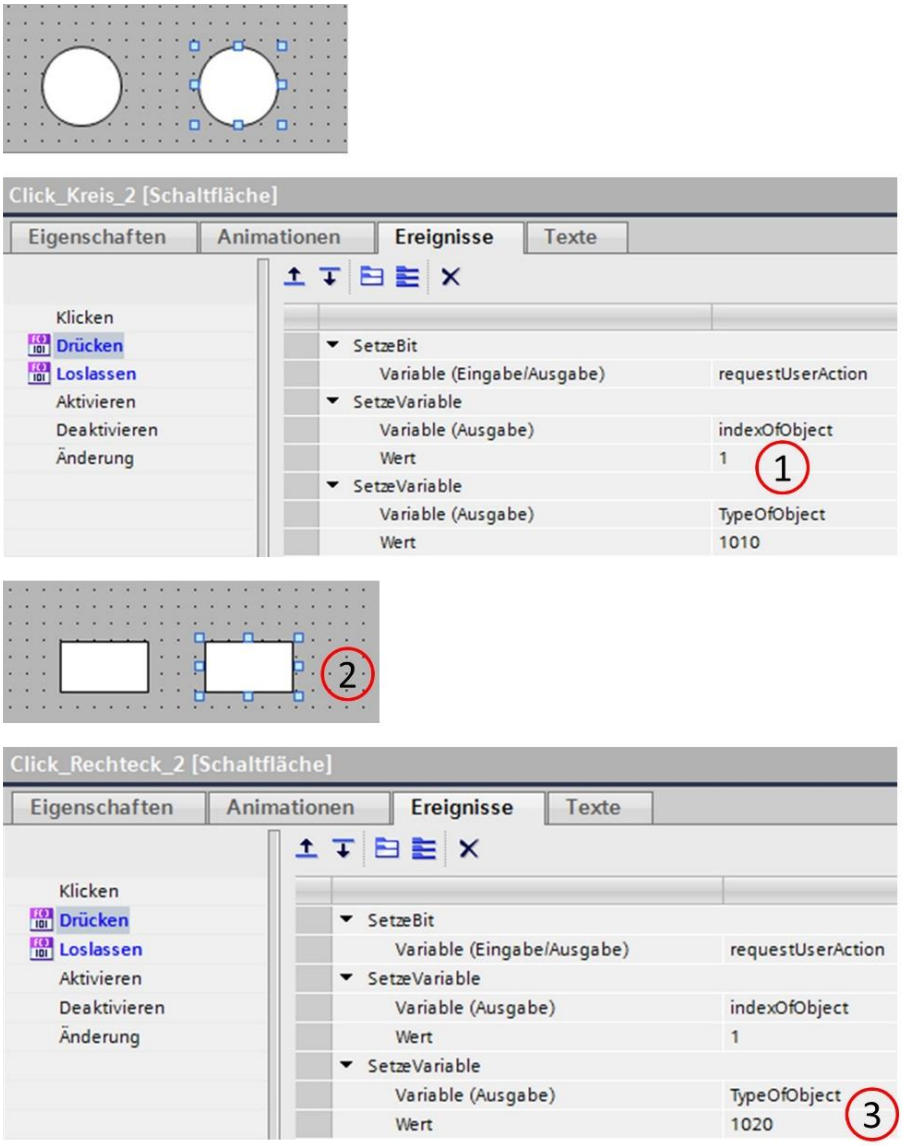


Bild 5.1 Beim Kopieren muss auch der Index in der Skript-Liste angepasst werden

Das kann durch Kopieren des Grundelementes z.B. *Kreis_1* sehr elegant durchgeführt werden, da die Namensvergebung automatisch durch den Index am Ende der Bezeichnung folgt (aus *Kreis_1* wird *Kreis_2* usw.). Zu beachten ist die Anpassung des *Index* auf 1 in der *Skriptleiste* (**Bild 5.1**) (*indexOfObject)* für *Click_Kreis_2* (**Punkt 1**). Die zwei neu eingefügten Rechtecke (**Punkt 2**) bekommen eine neue Kennzeichnung, damit der Skript *SelectObject* das neue Objekt erkennt. Die Variable *TypeOfObject* wird im Beispiel zum *Click_Rechteck_2* auf den Wert 1020 gesetzt (**Punkt 3**).

Mehrere Objekte haben natürlich auch zur Folge, dass die SPS um diese Objekte erweitert werden muss (**Bild 5.2**). Der neue Objekt-Typ, eingetragen mit 1020 für das Rechteck, wird in die *PLC-Variablen/Anwenderkonstanten* nachgetragen (**Punkt 1**). Dazu gehört auch die Anzahl der maximalen Objekte (jetzt mit 1 für den Indexbereich 0 bis 1), welche im *DB_Circle* und dem neuen *DB_Rectangle* als Indizes verwendet werden. Der neue PLC-Datentyp *UDT_RECTANGLE* (**Punkt 2**) wird für den neuen *DB_Rectangle* (**Punkt 3**) benötigt.

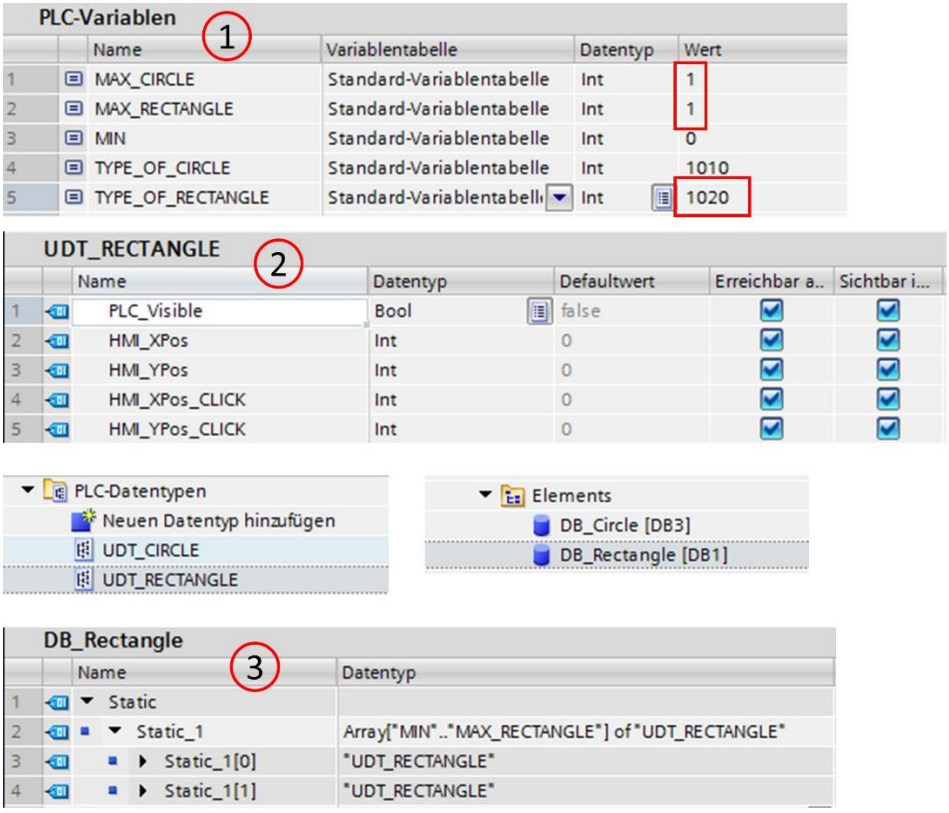


Bild 5.2 Neue Bildelemente fordern dessen Erweiterung in der SPS

Im Script *SelectObjects* (**Bild 5.3** und **Bild 5.4**) werden die neuen Objekte hinzugefügt. Durch kopieren der Texte und Änderung der Indizes ist das Anpassen recht schnell und einfach zu gestalten.

Damit der Skript ohne Fehler übersetzt werden kann, muss zuvor in den HMI-Variablen der neue Datentyp hinzugefügt werden. In **Bild 5.5** wird der Kopiervorgang dargestellt. Zuerst wird eine neue Variablentabelle *Rectangle* angelegt. Der neue *DB_Rectangle* wird angewählt, sodass dieser in der *Detailansicht* sichtbar wird. Nun können die beiden Datensätze in der Detailansicht gekennzeichnet werden und so direkt über *Drag&Drob* in die Variablentabelle kopiert werden (**Punkt 2**).

Vergessen Sie nicht die neuen HMI-Variablen mit der Zykluszeit von 100ms zu versehen (**Punkt 1**).

```
1  Sub SelectObject()
2
3  Const BorderX = 2 ' 2% from the screen width
4  Const BorderY = 2 ' 2% from the screen height
5
6  Dim ObjectScreen_Width, ObjectScreen_Height, ObjectScreen_Left, ObjectScreen_Top
7  Dim Object, ClickObject
8
9
10 'Screen limits
11   ObjectScreen_Width = HmiRuntime.Screens("Bild_1").Width - ((HmiRuntime.Screens("Bil
12   ObjectScreen_Height = HmiRuntime.Screens("Bild_1").Height - ((HmiRuntime.Screens("B
13   ObjectScreen_Left = (HmiRuntime.Screens("Bild_1").Width / 100) * BorderX
14   ObjectScreen_Top = (HmiRuntime.Screens("Bild_1").Height / 100) * BorderY
15
16 'call user functions
17   Select Case SmartTags("TypeOfObject")
18
19     Case 1010 'Circle
20       Select Case SmartTags("indexOfObject")
21         Case 0
22           Set Object = HmiRuntime.Screens("Bild_1").ScreenItems("Kreis_1")
23           Set ClickObject = HmiRuntime.Screens("Bild_1").ScreenItems("Click_Kreis_1")
24           SmartTags("DB_Circle_Static_1{0}.HMI_XPos") = Object.Left
25           SmartTags("DB_Circle_Static_1{0}.HMI_YPos") = Object.Top
26           SmartTags("DB_Circle_Static_1{0}.HMI_XPos_CLICK") = ClickObject.Left
27           SmartTags("DB_Circle_Static_1{0}.HMI_YPos_CLICK") = ClickObject.Top
28         Case 1
29           Set Object = HmiRuntime.Screens("Bild_1").ScreenItems("Kreis_2")
30           Set ClickObject = HmiRuntime.Screens("Bild_1").ScreenItems("Click_Kreis_2")
31           SmartTags("DB_Circle_Static_1{1}.HMI_XPos") = Object.Left
32           SmartTags("DB_Circle_Static_1{1}.HMI_YPos") = Object.Top
33           SmartTags("DB_Circle_Static_1{1}.HMI_XPos_CLICK") = ClickObject.Left
34           SmartTags("DB_Circle_Static_1{1}.HMI_YPos_CLICK") = ClickObject.Top
35         Case 2
36           'on so on
37       Case Else
38         Exit Sub
39       End Select 'indexOfObject
40       Move2Objects Object,ClickObject,ObjectScreen_Width, ObjectScreen_Left,ObjectScr
```

Bild 5.3 Das Bild-Objekt *Kreis_2* wird mit seinem Click-Element hinzugefügt

```
43      Case 1020 'Rectangle
44        Select Case SmartTags("indexOfObject")
45          Case 0
46            Set Object = HmiRuntime.Screens("Bild_1").ScreenItems("Rechteck_1")
47            Set ClickObject = HmiRuntime.Screens("Bild_1").ScreenItems("Click_Rechteck_1")
48            SmartTags("DB_Rectangle_Static_1{0}.HMI_XPos") = Object.Left
49            SmartTags("DB_Rectangle_Static_1{0}.HMI_YPos") = Object.Top
50            SmartTags("DB_Rectangle_Static_1{0}.HMI_XPos_CLICK") = ClickObject.Left
51            SmartTags("DB_Rectangle_Static_1{0}.HMI_YPos_CLICK") = ClickObject.Top
52          Case 1
53            Set Object = HmiRuntime.Screens("Bild_1").ScreenItems("Rechteck_2")
54            Set ClickObject = HmiRuntime.Screens("Bild_1").ScreenItems("Click_Rechteck_2")
55            SmartTags("DB_Rectangle_Static_1{1}.HMI_XPos") = Object.Left
56            SmartTags("DB_Rectangle_Static_1{1}.HMI_YPos") = Object.Top
57            SmartTags("DB_Rectangle_Static_1{1}.HMI_XPos_CLICK") = ClickObject.Left
58            SmartTags("DB_Rectangle_Static_1{1}.HMI_YPos_CLICK") = ClickObject.Top
59          Case 2
60            'on so on
61        Case Else
62          Exit Sub
63        End Select 'indexOfObject
64        Move2Objects Object,ClickObject,ObjectScreen_Width, ObjectScreen_Left,ObjectScreen_
65        Exit Sub
```

Bild 5.4 Der neue Datentyp *Rectangle* wird hinzugefügt

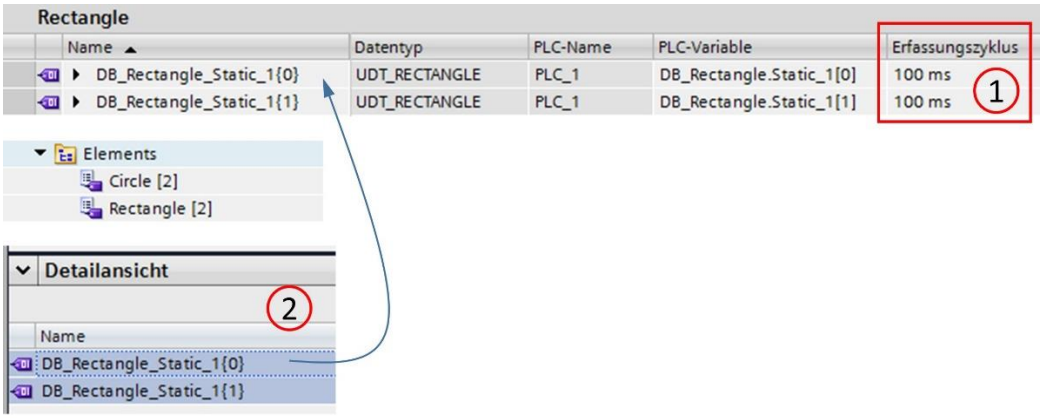


Bild 5.5 Die PLC-Variablen über Detailansicht in die HMI-Variablen kopieren

Da der neue Datentyp *Rectangle* ebenfalls aus nur zwei Objekten besteht, kann der Skript *Move2Objects* auch für das Rechteck übernommen werden (Zeile 64, **Bild 5.4**). Für die Initialisierung zum Bildwechsel, werden die neuen Objekte im Skript *InitObjects* (**Bild 5.6** und **Bild 5.7**) hinzugefügt.

```
Sub InitObjects()

'Circle
  '0
    'If(SmartTags("DB_Circle_Static_1{0}.PLC_Visible"))Then
      HmiRuntime.Screens("Bild_1").ScreenItems("Kreis_1").Left = _
                        SmartTags("DB_Circle_Static_1{0}.HMI_XPos")
      HmiRuntime.Screens("Bild_1").ScreenItems("Kreis_1").Top = _
                        SmartTags("DB_Circle_Static_1{0}.HMI_YPos")
      HmiRuntime.Screens("Bild_1").ScreenItems("Click_Kreis_1").Left = _
                        SmartTags("DB_Circle_Static_1{0}.HMI_XPos_CLICK")
      HmiRuntime.Screens("Bild_1").ScreenItems("Click_Kreis_1").Top = _
                        SmartTags("DB_Circle_Static_1{0}.HMI_YPos_CLICK")
    'End If
  '1
    'If(SmartTags("DB_Circle_Static_1{1}.PLC_Visible"))Then
      HmiRuntime.Screens("Bild_1").ScreenItems("Kreis_2").Left = _
                        SmartTags("DB_Circle_Static_1{1}.HMI_XPos")
      HmiRuntime.Screens("Bild_1").ScreenItems("Kreis_2").Top = _
                        SmartTags("DB_Circle_Static_1{1}.HMI_YPos")
      HmiRuntime.Screens("Bild_1").ScreenItems("Click_Kreis_2").Left = _
                        SmartTags("DB_Circle_Static_1{1}.HMI_XPos_CLICK")
      HmiRuntime.Screens("Bild_1").ScreenItems("Click_Kreis_2").Top = _
                        SmartTags("DB_Circle_Static_1{1}.HMI_YPos_CLICK")
    'End If

```

Bild 5.6 Der Skript *InitObjects* wird um das Bildelement *Kreis_2* erweitert

```
'Rectangle
  '0
    'If(SmartTags("DB_Rectangle_Static_1{0}.PLC_Visible"))Then
      HmiRuntime.Screens("Bild_1").ScreenItems("Rechteck_1").Left = _
                        SmartTags("DB_Rectangle_Static_1{0}.HMI_XPos")
      HmiRuntime.Screens("Bild_1").ScreenItems("Rechteck_1").Top = _
                        SmartTags("DB_Rectangle_Static_1{0}.HMI_YPos")
      HmiRuntime.Screens("Bild_1").ScreenItems("Click_Rechteck_1").Left = _
                        SmartTags("DB_Rectangle_Static_1{0}.HMI_XPos_CLICK")
      HmiRuntime.Screens("Bild_1").ScreenItems("Click_Rechteck_1").Top = _
                        SmartTags("DB_Rectangle_Static_1{0}.HMI_YPos_CLICK")
    'End If
  '1
    'If(SmartTags("DB_Rectangle_Static_1{1}.PLC_Visible"))Then
      HmiRuntime.Screens("Bild_1").ScreenItems("Rechteck_2").Left = _
                        SmartTags("DB_Rectangle_Static_1{1}.HMI_XPos")
      HmiRuntime.Screens("Bild_1").ScreenItems("Rechteck_2").Top = _
                        SmartTags("DB_Rectangle_Static_1{1}.HMI_YPos")
      HmiRuntime.Screens("Bild_1").ScreenItems("Click_Rechteck_2").Left = _
                        SmartTags("DB_Rectangle_Static_1{1}.HMI_XPos_CLICK")
      HmiRuntime.Screens("Bild_1").ScreenItems("Click_Rechteck_2").Top = _
                        SmartTags("DB_Rectangle_Static_1{1}.HMI_YPos_CLICK")
    'End If

  'final clear requestCoverVisible
  SmartTags("requestCoverVisible")  = False

End Sub
```

Bild 5.7 Der Skript *InitObjects* wird um die Bildelemente *Rechteck_1* und *Rechteck_2* erweitert

Bei der Erweiterung durch Bild-Objekte müssen alle beteiligten Skripte ebenfalls erweitert werden. Es ist mit Kopieren schon bestehender Programmierzeilen darauf zu achten, dass die Indizes richtig zugeordnet werden!

Nachdem nun alle Objekte richtig ergänzt wurden, kann für das Loslassen der linken Maustaste ein neuer Skript Move*2LeftGrid* in die Skriptleiste eingetragen werden (**Bild 5.8**). Dieser soll das soeben verschobene Objekt auf das geplante Raster fixieren.

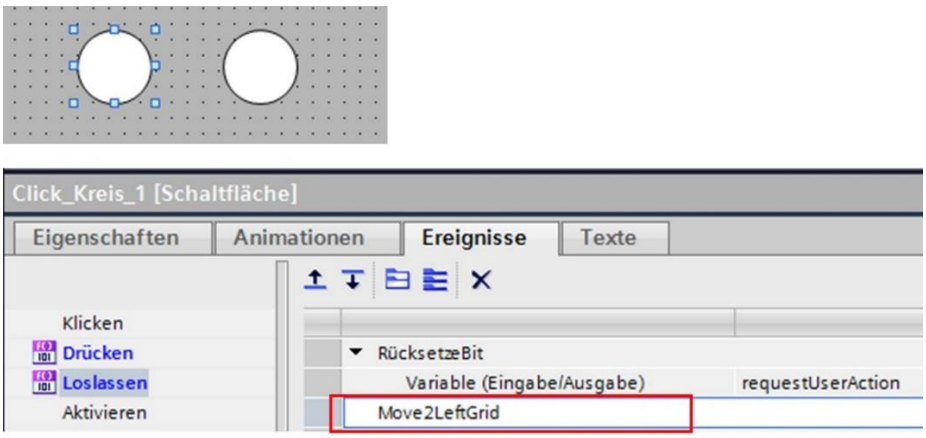
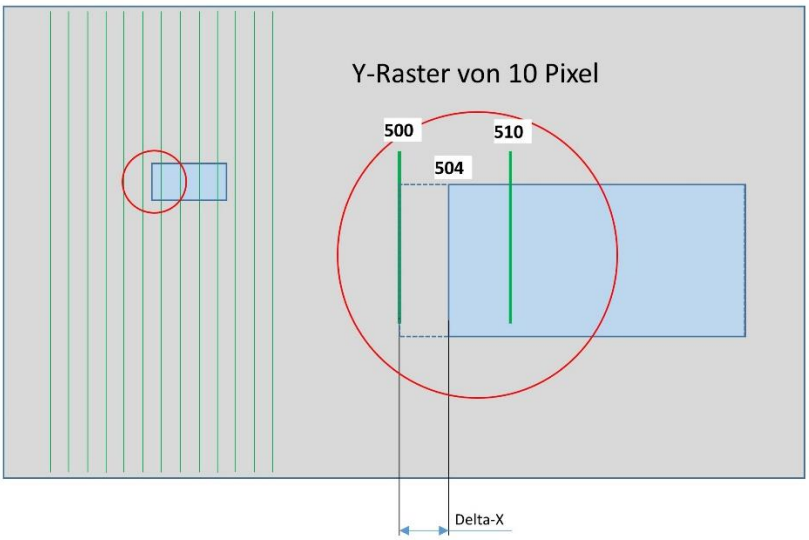


Bild 5.8 Beispiel für *Click_Kreis_1* mit dem Aufruf des Skriptes *Move2LeftGrid*

5.1 Der Skript *Move2LeftGrid*

Betrachten wir das Raster mit einer Rasterbreite von 10 Pixel für die X-Achse (**Bild 5.9**). Das Objekt soll nach dem Loslassen der linken Maustaste auf die nächste linke Rasterlinie springen. Im Beispiel befindet sich das Objekt noch auf der X-Achse mit dem X-Maß 504. Nun ergibt sich aus dem Formelabschnitt *CInt(Object.Left/CInt(10));* 504/10 der Wert 50.4, welcher auf 50 durch den Integer-Cast *CInt* abgeschnitten wird (Nachkommastellen werden gelöscht). So ergibt sich aus dem Rest der Formel ein Delta-X von 4 Pixel, welches im Skript dann vom *Object.Left* abgezogen wird. Das Ergebnis lautet dann 504-4 und ergibt die gewünschte X-Position 500, welches dem linken Raster aus der Position 504 entspricht. Der Skript *Move2LeftGrid* wird hier für das Rechteck und den Kreis verwendet. Beide haben zwei Elemente (Bildobjekt und der Klick zum Bildobjekt). Damit Elemente mit mehr als 2 Objekten auf das Raster geschoben werden, wird im Skript *Move2LeftGrid* die Subroutine *SUB_Grid_Element_2* (**Bild 5.10**) aufgerufen, welche die Berechnung für die im Parameter angegebenen zwei Objekte ausführt. Später wird es dann einen Skript *SUB_Grid_Element_3 usw.* geben.



Delta-X = Object.Left - CInt((CInt(Object.Left)/ CInt(10)))*CInt(10)

Delta-X = 504 - CInt((CInt(504/ CInt(10)))*CInt(10)

Delta-X = 504 – 50 * CInt(10) = 504-500 = 4

Bild 5.9 Versatz zum linken Raster berechnen

```
1  Sub SUB_Grid_Element_2(ByRef Object, ByRef ClickObject)
2
3  Dim deltaX, deltaY
4
5  deltaX = Object.Left - CInt((CInt(Object.Left)/CInt(10)))*CInt(10)
6  deltaY = Object.Top - CInt((CInt(Object.Top)/CInt(10)))*CInt(10)
7
8  Object.Left = Object.Left - deltaX
9  Object.Top = Object.Top - deltaY
10 ClickObject.Left = ClickObject.Left - deltaX
11 ClickObject.Top = ClickObject.Top - deltaY
12
13 End Sub
```

Bild 5.10 Der Skript *SUB_Grid_Element-2* bewegt für 2 Objekte den Sprung zum linken *Grid*

Im (**Bild 5.11**) wird der erste Teil zum Skript *Move2LeftGrid* dargestellt. In Zeile 11 ist der Aufruf des SUB-Skripts für den ersten Kreis zu sehen. Nach diesem Aufruf werden die Koordinaten in die SPS gespeichert.

```
1  Sub Move2LeftGrid()
2
3  Dim object, clickObject
4
5  Select Case SmartTags("TypeOfObject")
6    Case 1010 'Circle
7      Select Case SmartTags("indexOfObject")
8        Case 0
9          Set object = HmiRuntime.Screens( "Bild_1").ScreenItems("Kreis_1")
10         Set clickObject = HmiRuntime.Screens("Bild_1").ScreenItems("Click_Kreis_1")
11         SUB_Grid_Element_2 object,clickObject
12         SmartTags("DB_Circle_Static_1{0}.HMI_XPos") = object.Left
13         SmartTags("DB_Circle_Static_1{0}.HMI_YPos") = object.Top
14         SmartTags("DB_Circle_Static_1{0}.HMI_XPos_CLICK") = clickObject.Left
15         SmartTags("DB_Circle_Static_1{0}.HMI_YPos_CLICK") = clickObject.Top
16       Case 1
17         Set object = HmiRuntime.Screens( "Bild_1").ScreenItems("Kreis_2")
18         Set clickObject = HmiRuntime.Screens("Bild_1").ScreenItems("Click_Kreis_2")
19         SUB_Grid_Element_2 object,clickObject
20         SmartTags("DB_Circle_Static_1{1}.HMI_XPos") = object.Left
21         SmartTags("DB_Circle_Static_1{1}.HMI_YPos") = object.Top
22         SmartTags("DB_Circle_Static_1{1}.HMI_XPos_CLICK") = clickObject.Left
23         SmartTags("DB_Circle_Static_1{1}.HMI_YPos_CLICK") = clickObject.Top
24       Case 2
25         'and so on
26
27     End Select
28
29     ' muelleimer
30     '   If( (object.Left <= 100) And (object.Top <= 170) )Then
31     '     DeleteObject
32     '     Exit Sub
33     '   End If
34     Exit Sub
35
```

Bild 5.11 Der erste Teil vom Skript *Move2LeftGrid*

Das Speichern in die SPS in den Skript *SUB_Grid_Element_2* zu verlagern, wäre gedanklich richtig. Ist aber nicht zu empfehlen, da die *SmartTags* mit der Zuordnung über Parameter so nicht problemlos funktionieren.

Ab Zeile 29 ist ein Teil des Listings kommentiert. Hier soll später das Objekt wieder gelöscht werden, falls es sich beim Loslassen der Maustaste im Bereich eines Mülleimers befindet.

```
36  Case 1020 'Rectangle
37    Select Case SmartTags("indexOfObject")
38      Case 0
39        Set object = HmiRuntime.Screens( "Bild_1").ScreenItems("Rechteck_1")
40        Set clickObject = HmiRuntime.Screens("Bild_1").ScreenItems("Click_Rechteck_1")
41        SUB_Grid_Element_2 object,clickObject
42        SmartTags("DB_Rectangle_Static_1{0}.HMI_XPos") = object.Left
43        SmartTags("DB_Rectangle_Static_1{0}.HMI_YPos") = object.Top
44        SmartTags("DB_Rectangle_Static_1{0}.HMI_XPos_CLICK") = clickObject.Left
45        SmartTags("DB_Rectangle_Static_1{0}.HMI_YPos_CLICK") = clickObject.Top
46      Case 1
47        Set object = HmiRuntime.Screens( "Bild_1").ScreenItems("Rechteck_2")
48        Set clickObject = HmiRuntime.Screens("Bild_1").ScreenItems("Click_Rechteck_2")
49        SUB_Grid_Element_2 object,clickObject
50        SmartTags("DB_Rectangle_Static_1{1}.HMI_XPos") = object.Left
51        SmartTags("DB_Rectangle_Static_1{1}.HMI_YPos") = object.Top
52        SmartTags("DB_Rectangle_Static_1{1}.HMI_XPos_CLICK") = clickObject.Left
53        SmartTags("DB_Rectangle_Static_1{1}.HMI_YPos_CLICK") = clickObject.Top
54      Case 2
55        'and so on
56
57    End Select
58    ' muelleimer
59    '   If( (object.Left <= 100) And (object.Top <= 170) )Then
60    '     DeleteObject
61    '     Exit Sub
62    '   End If
63    Exit Sub
64
65 End Select 'type
66
67 End Sub
```

Bild 5.12 Der zweite Teil vom Skript *Move2LeftGrid*

In **Bild 5.12** ist der zweite Teil für das Rechteck ersichtlich. Auch dieser Teil beinhaltet schon die Vorbereitung das Objekt zu löschen (ab Zeile 58). **Bild 5.13** zeigt das Ergebnis, wenn mit der Maus die Objekte in die Nähe der Line positioniert werden, sind danach alle Objekte auf die Position mit X=500 durch den Skript *Move2LeftGrid* fixiert.

Damit sind die Grundstrukturen für die Bewegung mit der Maus ausführlich vorgestellt worden. Da es sich hier um Einzelobjekte handelt, ist der praktische Einsatz noch nicht direkt erkennbar. Auch gibt es noch das bekannte Problem, dass die Objekte beim Erstaufruf noch auf die Koordinaten „0,0“ geschoben werden, da der Skript *InitObjects* noch nicht ganz fertig ist.

Im nächsten Kapitel wird eine Menüleiste vorgestellt, die es erlaubt Objekte wie auf einem Smartphone auf die Oberfläche des Monitors zu ziehen und zu positionieren. Diese Technik zeigt dann schon einen praktischen Bezug, wie z. B. für die Erstellung eines P&IDs.

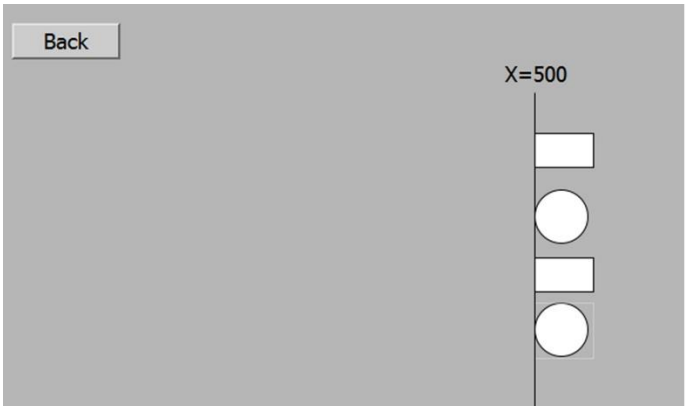


Bild 5.13 Die Bildobjekte auf die X-Achse 500 fixiert

6 Die Menüleiste

Objekte können wie bei einem Smartphone aus einer Menüleiste auf die Bild-Oberfläche gezogen und positioniert werden. Soll das Objekt wieder gelöscht werden, so kann man dieses in einen Mülleimer schieben um es damit wieder verschwinden zu lassen.

6.1 Objekte in der Menüleiste anbieten

Das TIA-Projekt zu diesem Kapitel zeigt den Ausschnitt einer Menüleiste für die Grundelemente eines P&ID. Im oberen Teil (**Bild 6.1**) befindet sich eine Menüleiste mit Bild-Objekten (*Grafikanzeige*) und je einer darüber befindlichen Schaltfläche.

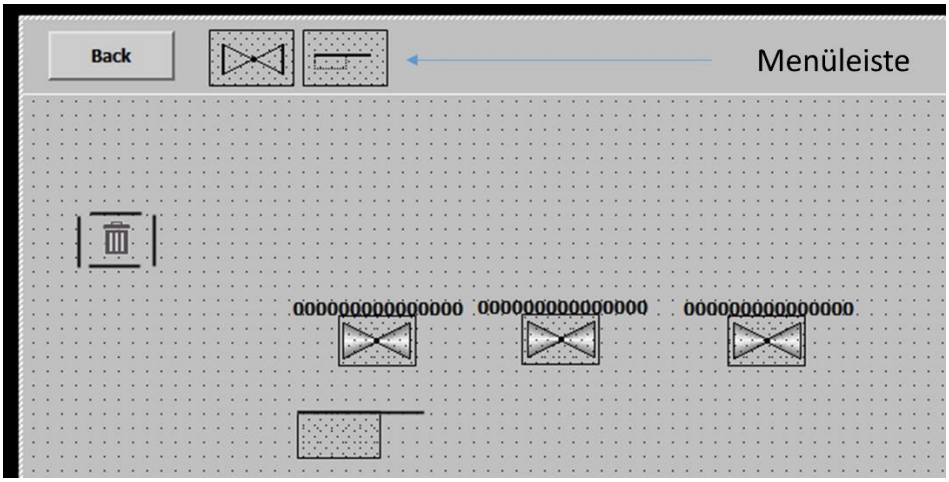


Bild 6.1 Die Menüleiste für zwei Objekte

Die Aufgabe besteht nun darin, aus der Menüleiste eines der unten gezeichneten Bild-Objekte (grafisches *E/A-Feld*) auf die Oberfläche zu ziehen und zu positionieren. Werden diese nicht mehr gebraucht, dann können die Objekte zur Löschung auf den Mülleimer gezogen werden.

Wie das Zusammenspiel der Menüleiste mit den Bild-Objekten auf der Oberfläche und der SPS funktionieren, zeigen die folgenden Kapitel.

6.1.1 Bild-Objekte mit der *Grafikanzeige*

Für die Menüleiste werden Grafik-Objekte vom Typ *Grafikanzeige* verwendet. Im Beispiel haben die Grafik-Objekte die Größe von 55x35 Pixel (Beispiel horizontales Ventil). Das Bild für die Darstellung in der Menüleiste konstruieren wir mit dem Paint-Tool (**Bild 6.2**). Hier legen wir die gleiche Größe mit 55x35 Pixel fest. So kann die Grafik ohne Verzerrungen in die *Grafikanzeige* eingefügt werden. Zur Menüleiste gehören die gezeichneten Bild-Objekte Valve-Horizontal und Linie-Horizontal.

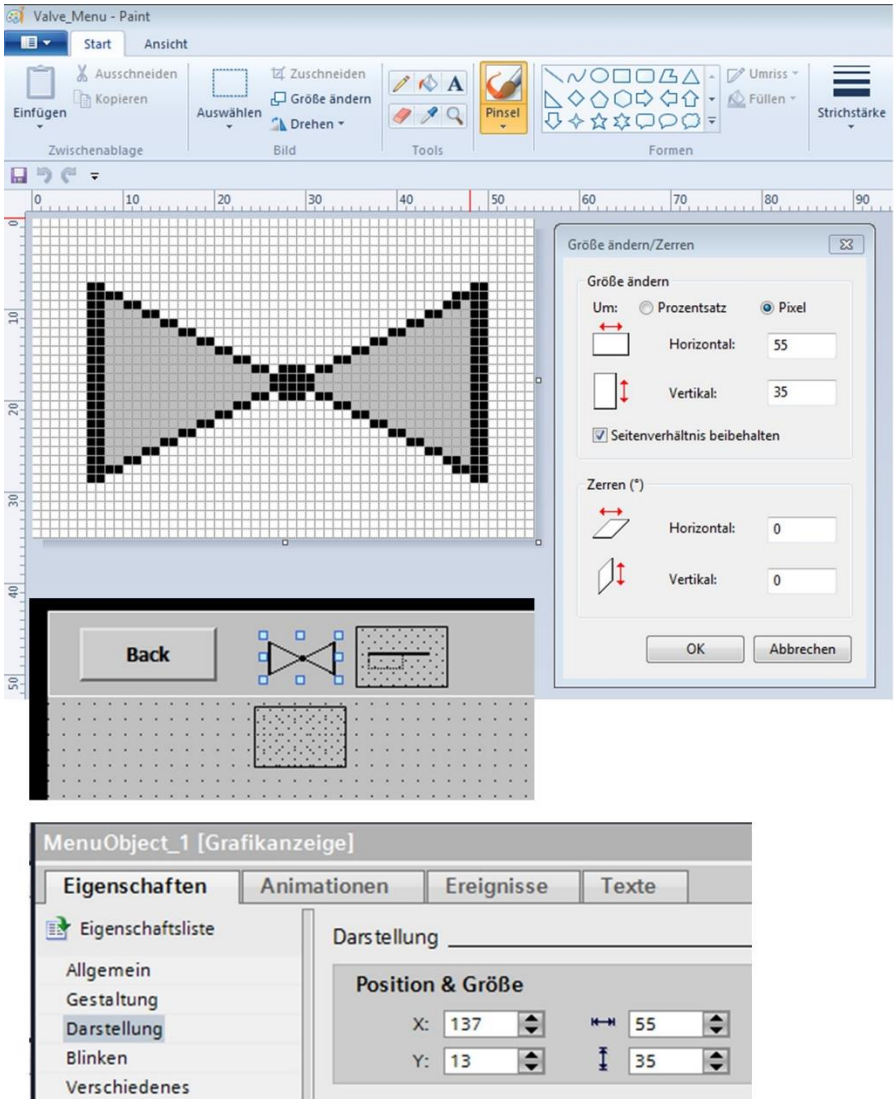


Bild 6.2 Grafik-Objekt für die Menüleiste sind mit *Paint* gezeichnet

6.1.2 Die Bild-Objekte mit dem *Grafischen E/A-Feld*

Damit die in der Menüleiste angewählten Objekte auf die Bildoberfläche gezogen werden können, müssen diese auch vorhanden sein. In **Bild 6.3** ist ein Beispiel zum horizontalen Ventil gezeigt. Das Objekt besteht aus den 3 Komponenten *ValveName_Hor_0*, *Click_Valve_Hor_0* und *Valve_Hor_0*.

Das grafische *E/A-Feld* zum Ventil-Bild beträgt 50x30 Pixel (**Punkt 1**) und wird ebenfalls über das Tool *Paint* mit der gleichen Pixelgröße gezeichnet. Damit diese Objekte mit der Maus

bewegt werden können, müssen einige neue Skripte geschrieben werden, da es sich hier um 3 und nicht wie bereits schon angewendet, um 2 Elemente handelt.

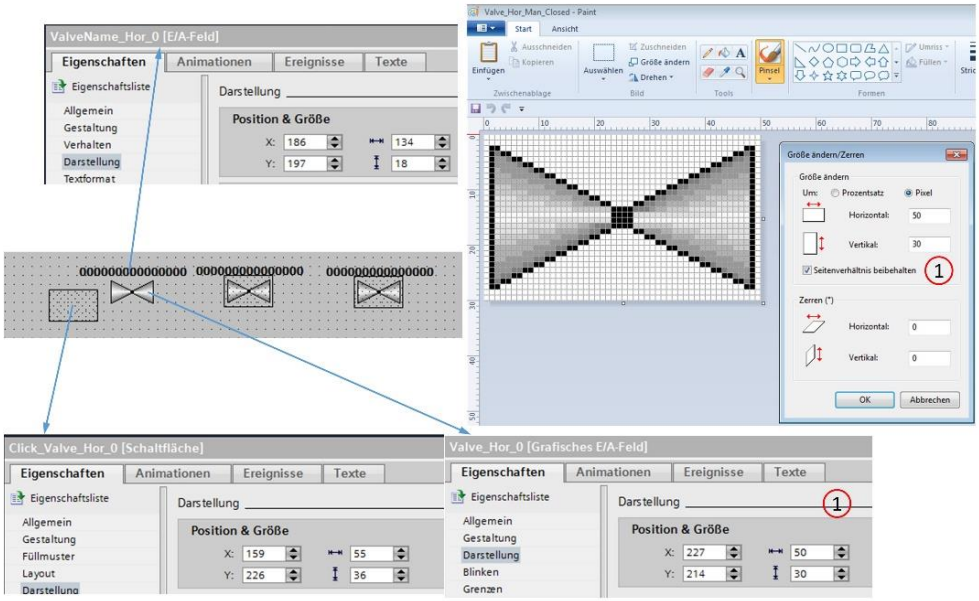


Bild 6.3 Beispiel am Bildobjekt *Valve_Hor_0*

In der PLC werden die entsprechenden *DBs* für die Ventile und die horizontale Linie mit den PLC-Datentypen *UDT_LINE* und *UDT_VALVE* deklariert (**Bild 6.4**, **Punkt 1**). Die UDTs (**Punkte 2+3**) beinhalten die Positionen der Elemente und die notwendigen Informationen zum Objekt. Die Linie speichert in *HMI_Length* die Länge der Linie und in *HMI_Name* die Bezeichnung des Ventils. Die *DBs* (**Punkte 4+5**) zeigen die jeweiligen Speicherbereiche, welche über die Anwenderkonstanten *MAX_VALVE* und *MAX_LINE* deklariert wurden (**Punkt 6**).

Im PLC-Datentyp für das Ventil und die Linie wurde eine zusätzliche Variable (*Image*) verwendet, damit die Grafikliste in der HMI eine Zuordnung erhält. Diese Deklaration ist bei Grafiklisten notwendig, soll aber hier nicht gesondert behandelt werden.

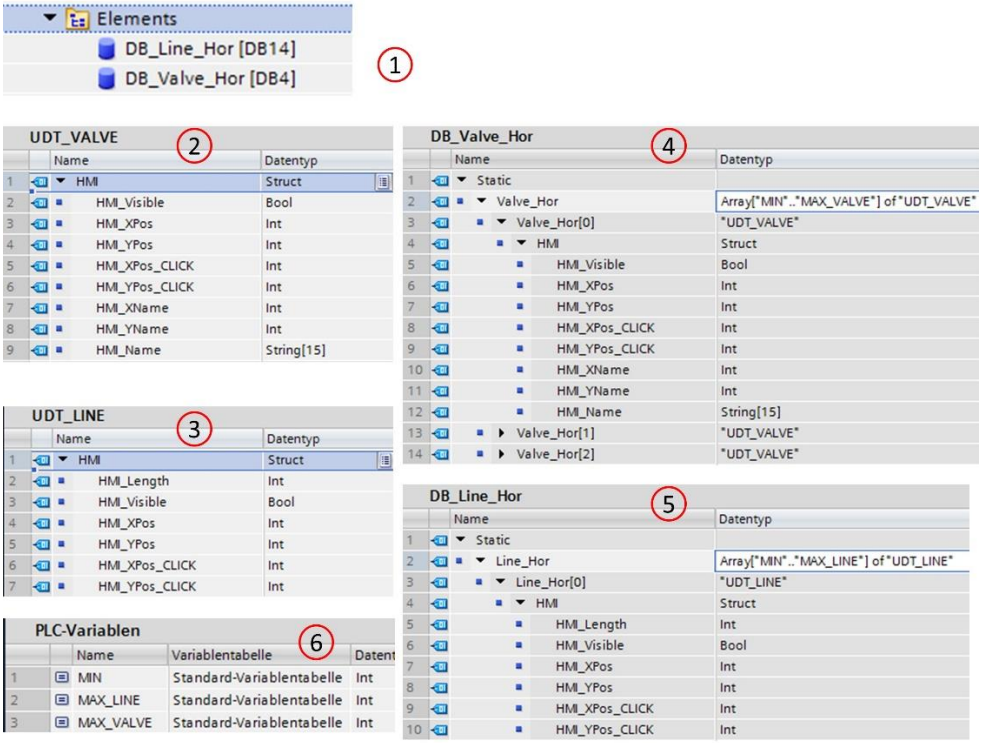


Bild 6.4 PLC-Deklarationen der neuen Elemente

6.2 Programmierung der Menüleiste

Wie kommen nun die Objekte *HMI_Valve_Hor* und *HMI_Line* auf die Oberfläche, wenn in der Menüleiste ein Klick auf das gewünschte Objekt ausgeführt wird?

Wie in **Kapitel 3** bereits erklärt wurde, ist der zyklische Request *UserAction* dafür verantwortlich, den Skript *SelectObjects* aufzurufen (siehe **Bild 3.9**). Bisher wurde direkt auf das zu bewegende Bild-Objekt über dessen Schaltfläche geklickt. Jetzt allerdings klicken wir auf die Schaltfläche in der Menüleiste und möchten ein freies Bild-Objekt sichtbar machen und zunächst auf eine feste Position setzen.

Der Skript *NewObject* wertet die Schaltfläche in der Menüleiste aus und sucht sich in der SPS ein freies Bild-Objekt, welches dann abhängig von der Position in der Menüleiste positioniert wird.

Danach ruft *NewObject* den Skript *UserAction* auf und befindet sich damit wieder in der zyklischen Schleife und das Objekt kann nun von der fixen Position aus, lückenlos auf die Oberfläche verschoben werden.

6.2.1 Der Skript *NewObject* Teil-1

In **Bild 6.5** ist im oberen Teil die Schaltfläche für das horizontale Ventil gedrückt worden. Es wird in der Skript-Liste das Ereignis *Drücken* aktiviert (**Punkt 1**). Die Variable *TypeOfObject* mit 1020 finden wir im Listing zu *NewObject* in den Zeilen 5-7 wieder.

Da wir das Bild-Objekt zuerst auf eine feste Position setzen möchten, wird die Position der gedrückten Schaltfläche in der Menüleiste benötigt.

In Zeile 8 wird die Variable *MenuButton* mit *Click_MenuObject_1* gesetzt. Damit sind die Daten für die gedrückte Schaltfläche in *MenuButton* zugänglich und können später verwendet werden.

In einem Projekt sind meistens mehrere Bildobjekte gezeichnet. In der Menüleiste wird das nächste freie Objekt benötigt. In Zeile 9 wird die Variable *_requestActual_Index_Valve_Hor* auf -1 geprüft. Das Ergebnis dieser Variablen wird vom *FB_ObjectControl* ermittelt (Kommentar in Zeile 10). Eine -1 bedeutet, dass kein freies Bild-Objekt gefunden wurde und der Skript wird in Zeile 11 mit *Exit Sub* verlassen.

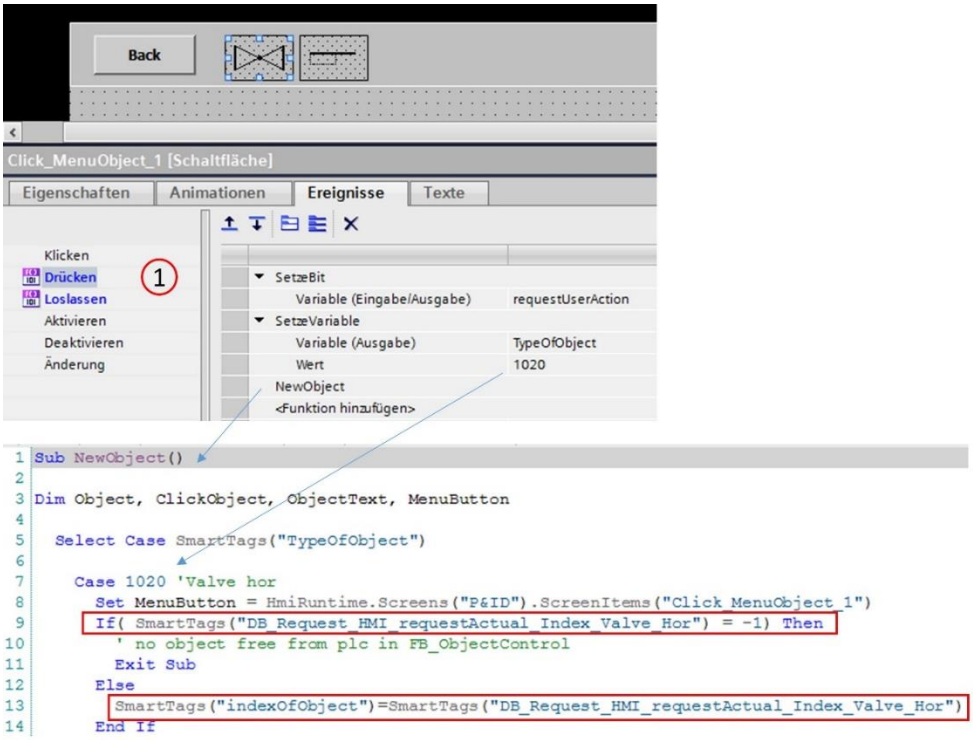


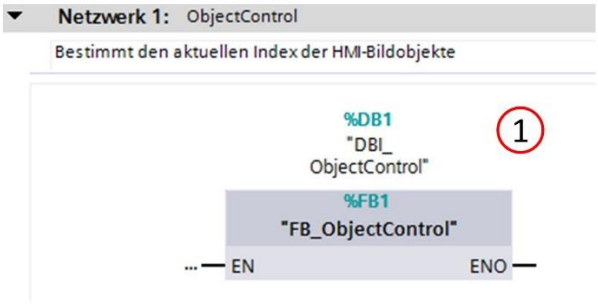
Bild 6.5 Der Klick auf die Menüleiste löst den Skript *NewObject* aus

Ist dagegen noch ein freies Objekt vorhanden, dann wird der Index dieses Objektes in *indexOfObject* gespeichert (Zeile 13) und mit diesem Index im Skript weiter gearbeitet werden.

6.2.1.1 Der *FB_ObjectControl*

Der *FB_ObjectControl* (**Bild 6.6**) wird in der PLC in *Main* (OB1) zyklisch aufgerufen (**Punkt 1**) und kontrolliert ständig, ob noch ein freies Element zur Verfügung steht. Die Verfügbarkeit wird über die Sichtbarkeit des Objektes entschieden.

Das *IF-Statement* in Zeile 3 (Beispiel für die horizontalen Ventile) prüft, ob *#static_Index_Valve_Hor* innerhalb der Grenzen von „MIN“ und „MAX_VALVE“ (**Punkt 2**) befindet. Wenn nicht, wird die Variable im *Else-Teil* auf Null gesetzt (Zeile 15). Es handelt sich hier um eine sogenannte offene Schleife, welche bei jedem Aufruf mit der Schleifenvariable *#static_Index_Valve_Hor* um eins inkrementiert wird (Zeile 12) bis ein Ventil mit der Sichtbarkeit (Zeilen 6+7) gefunden wird. Der Index dieses Objektes wird dann in die Variable *requestActual_Index_Valve_Hor* gespeichert (**Punkt 3**).




```
1  // FB_ObjectControl
2    // Valve hor
3    IF ((#static_Index_Valve_Hor >= "MIN")
4      AND
5      (#static_Index_Valve_Hor <= "MAX_VALVE")) THEN
6      IF (NOT "DB_Valve_Hor".Valve_Hor[#static_Index_Valve_Hor].
7        HMI.HMI_Visible) THEN
8        "DB_Request_HMI".requestActual_Index_Valve_Hor :=
9          #static_Index_Valve_Hor;
10       #static_Index_Valve_Hor := 0;
11     ELSE
12       #static_Index_Valve_Hor := #static_Index_Valve_Hor + 1;
13     END_IF;
14   ELSE
15     #static_Index_Valve_Hor := 0;
16     "DB_Request_HMI".requestActual_Index_Valve_Hor := -1;
17   END_IF;
18
```

2

3

Bild 6.6 Der *FB_ObjectControl*

6.2.2 Der Skript *NewObject* Teil-2

Wird im *FB_ObjectControl* ein Index gefunden (*requestActual_Index_Valve_Hor* ungleich -1), so kann nun das entsprechende Objekt sichtbar geschaltet werden.

```
15      Select Case SmartTags("indexOfObject")
16        Case 0
17          SmartTags("DB_Valve_Hor_Valve_Hor{0}_HMI_HMI_Visible") = True
18          Set Object = HmiRuntime.Screens("P&ID")._
19                                  ScreenItems("Valve_Hor_0")
20          Set ClickObject = HmiRuntime.Screens("P&ID")._
21                                  ScreenItems("Click_Valve_Hor_0")
22          Set ObjectText = HmiRuntime.Screens("P&ID")._
23                                  ScreenItems("ValveName_Hor_0")
24          NewObject3Elements Object,ClickObject,ObjectText, MenuButton
25          UserAction
```

Bild 6.7 Das Ventil wird auf die feste Koordinate sichtbar geschaltet

Der *indexOfObject* (**Bild 6.7**, Zeile 15) wurde durch den *FB_ObjectControl* gesetzt (siehe **Bild 6.5**, Zeile 13). Im Beispiel wird das *Valve_Hor_0{0}* in Zeile 17 auf sichtbar geschaltet. In den Zeilen 18-23 werden die Objekte für den Aufruf der Skripte *NewObject3Elements* vorbereitet. Dieser Skript positioniert das Bild-Objekt in Abhängigkeit des *MenuButton* (Zeile 24). Danach wird der schon bekannte Skript *userAction* aufgerufen (Zeile 25) und das Bild-Objekt kann nun mit der Maus weiter bewegt werden.

Beide Skripte *NewObject* und *NewObject3Elements* sorgen dafür, dass das Objekt in der Nähe von der gedrückten Schaltfläche in der Menüzeile positioniert und sichtbar gemacht wird, um dann ganz elegant durch den Aufruf des Skriptes *UserAction* in der Zyklusschleife (**Kapitel 3**) zu landen. Das Objekt kann dann weiter mit der Maus bewegt werden.

6.2.3 Der Skript *NewObject3Elements*

Im Skript *NewObject* wird der Skript *NewObject3Elements* (**Bild 6.8**) für die Ventile aufgerufen, da es sich bei diesem Objekt um 3 Elemente handelt (Bild-Objekt, Schaltfläche-Klick und Text).

Mit dem ersten Parameter *HMI_Element_0* (Bild-Objekt Valve) kann die Differenz *difX* errechnet werden (**Bild 6.8,** Zeile 6). So können nun alle Objekte (Parameter *HMI_Element_0* bis *HMI_Element_2*), wie beispielhaft in Zeile9 ersichtlich, auf eine feste Position verschoben werden.

```
Sub NewObject3Elements(ByRef HMI_Element_0, ByRef HMI_Element_1,
                       ByRef HMI_Element_2, ByRef HMI_MenuButton)
Dim difX, difY

'main object
  difX = (HMI_Element_0.Left - HMI_MenuButton.Left)
  difY = (HMI_Element_0.Top - HMI_MenuButton.Top)

  HMI_Element_0.Left = HMI_Element_0.Left - difX + HMI_MenuButton.Width/3
  HMI_Element_0.Top = HMI_Element_0.Top - difY  + HMI_MenuButton.Height/2

' move all other elements from the objects by the same distance
  HMI_Element_1.Left = HMI_Element_1.Left - difX + HMI_MenuButton.Width/3
  HMI_Element_1.Top = HMI_Element_1.Top - difY  + HMI_MenuButton.Height/2

  HMI_Element_2.Left = HMI_Element_2.Left - difX + HMI_MenuButton.Width/3
  HMI_Element_2.Top = HMI_Element_2.Top - difY  + HMI_MenuButton.Height/2

End Sub
```

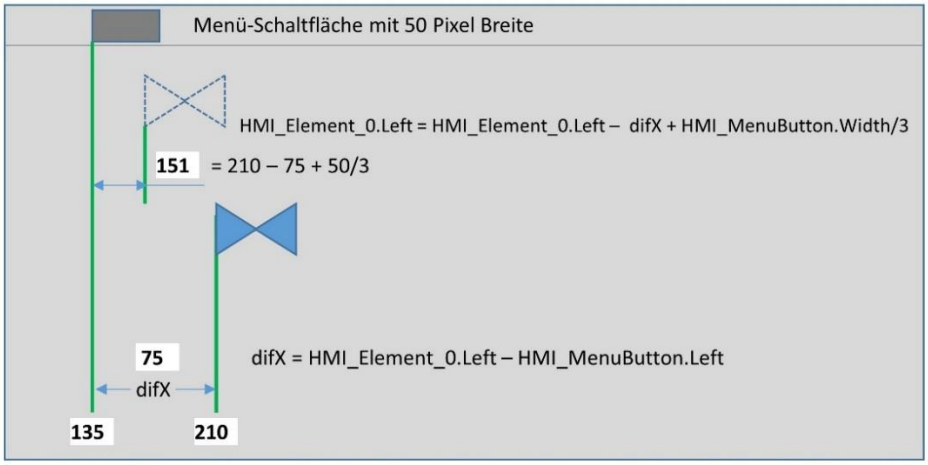


Bild 6.8 Berechnung der Positionen der einzelnen Elemente in Skript *NewObject3Elements*

Es ist darauf zu achten, dass der so errechnete Fixpunkt in Abhängigkeit der Menü-Schaltfläche innerhalb der Bildgrenzen liegt, da sonst unerwünschte Effekte entstehen.

Wenn das der Fall ist, weil z.B. die Bild-Objekte sehr groß sind (Behälter), muss der Skript *NewObject3Elements* entsprechend angepasst werden.

6.2.4 Der Skript *NewObject* Teil-3

In **Bild 6.9** ist die Positionierung der horizontalen Linie dargestellt. Im Prinzip identisch mit dem Ventil-Objekt bis auf den Skript *NewObject2Elements,* da die Linie nur aus 2 Elementen (Bild-Objekt und Click-Schaltfläche) besteht.

```
46      Case 1060 'Pipe hor
47        Set MenuButton = HmiRuntime.Screens("P&ID")._
48                                 ScreenItems("Click_MenuObject_2")
49        If( SmartTags("DB_Request_HMI_requestActual_Index_Line_Hor")_
50             = -1) Then
51          ' no object free from plc in FB_ObjectControl
52          Exit Sub
53        Else
54          SmartTags("indexOfObject")=_
55                      SmartTags("DB_Request_HMI_requestActual_Index_Line_Hor")
56        End If
57        Select Case SmartTags("indexOfObject")
58          Case 0
59            SmartTags("DB_Line_Hor_Line_Hor{0}_HMI_HMI_Visible")=True
60            Set Object = HmiRuntime.Screens("P&ID").ScreenItems("LineHor_0")
61            Set ClickObject = HmiRuntime.Screens("P&ID")._
62                                                ScreenItems("Click_LineHor_0")
63            NewObject2Elements Object,ClickObject, MenuButton
64            UserAction
65          Case 1
66            'and so on
```

```
1  Sub NewObject2Elements(ByRef HMI_Element_0, ByRef HMI_Element_1, ByRef HMI
2
3  Dim difX, difY
4
5  'main object
6    difX = (HMI_Element_0.Left - HMI_MenuButton.Left)
7    difY = (HMI_Element_0.Top - HMI_MenuButton.Top)
8
9    HMI_Element_0.Left = HMI_Element_0.Left - difX + HMI_MenuButton.Width/3
10   HMI_Element_0.Top = HMI_Element_0.Top - difY + HMI_MenuButton.Height/2
11
12 ' move all other elements from the objects by the same distance
13   HMI_Element_1.Left = HMI_Element_1.Left - difX + HMI_MenuButton.Width/3
14   HMI_Element_1.Top = HMI_Element_1.Top - difY + HMI_MenuButton.Height/2
15
16
17 End Sub
```

Bild 6.9 Die horizontale Linie wird durch den Skript *NewObject2Elements positioniert*

6.2.5 Test für Objekte aus der Menüleiste einfügen

Für den Test wurde im Beobachtungsmodus mittels *Operand steuern* dem ersten Ventil der Name „Valve-1“ zugeordnet (**Bild 6.10**, **Punkt 1**). Somit kann das Verschieben mit drei Elementen sichtbar nachvollzogen werden. Nachdem alle Objekte eingefügt worden, gibt es mit einem Klick auf die Menüleiste keine Reaktion mehr, da der *FB_ObjectControl* als Indexergebnis eine -1 liefert und so der Skript *NewObject* ohne weitere Tätigkeit verlassen wird (siehe z.B. **Bild 6.5**, Zeile 11).

Das Bild-Objekt Linie wird aufgrund der Layertechnik hinter das Ventil abgebildet. Natürlich besteht jetzt der Wunsch die Linie zu vergrößern um darauf die Ventile zu setzen. Wie das

funktioniert und welche Bedingungen dazu notwendig sind, wird in **Kapitel 7** erläutert. Wie die Bild-Objekte wieder gelöscht werden können, zeigt **Kapitel 6.3**.

DB_Valve_Hor

	Name	Datentyp	Offset	Startwert	Beobachtungswert
1	Static				
2	Valve_Hor	Array["MIN"..."MAX_...	0.0		
3	Valve_Hor[0]	"UDT_VALVE"	0.0		
4	HMI	Struct	0.0		
5	HMI_Visible	Bool	0.0	false	TRUE
6	HMI_XPos	Int	2.0	0	196
7	HMI_YPos	Int	4.0	0	130
8	HMI_XPos_CLICK	Int	6.0	0	193
9	HMI_YPos_CLICK	Int	8.0	0	127
10	HMI_XName	Int	10.0	0	155
11	HMI_YName	Int	12.0	0	113
12	HMI_Name	String[15]	14.0	''	'Valve-1'

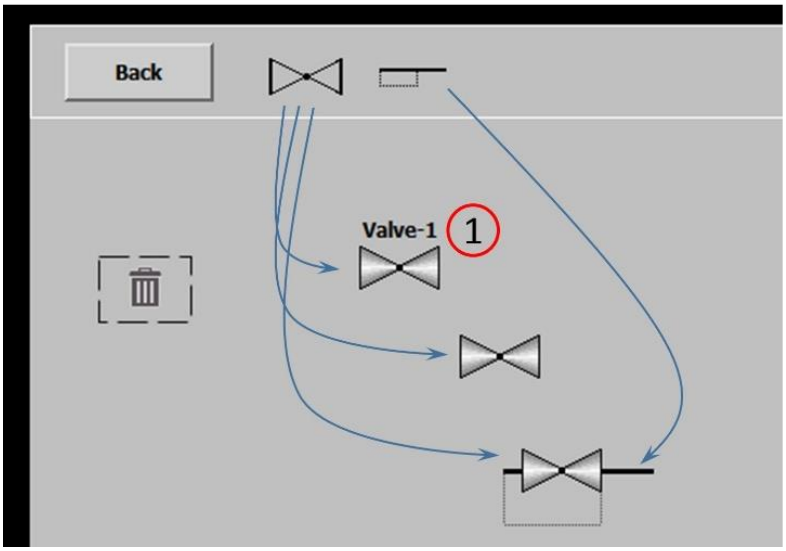


Bild 6.10 Das Einfügen der Bildobjekte von der Menüleiste

6.3 Objekte löschen

Werden die über den Skript *NewObject* eingefügten Bild-Objekte wieder unsichtbar geschaltet, stehen diese wieder zur Anwahl in der Menüleiste zur Verfügung. Der Löschbereich wird durch die Lage des Bild-Objektes *MenuObject_Delete* (**Bild 6.11**) bestimmt. Befindet sich das Objekt innerhalb dieses Bild-Objektes (Mülleimer), dann wird die Eigenschaft *Visible* im entsprechenden DB durch den Skript *DeleteObject* auf *False* gesetzt und das Bild-Objekt ist nicht mehr sichtbar.

In den Zeilen 6-8 (**Bild 6.12**) des Skriptes *DeleteObject* werden die Limits berechnet. Im Beispiel zum ersten Ventil wird die Sichtbarkeit auf *False* gesetzt, wenn die Bedingung dass sich das Objekt innerhalb des Mülleimers befindet (Zeile 17), erfüllt ist. *DeleteObject* wird in der Skriptleiste bei der Eigenschaft *Loslassen* aufgerufen (Beispiel beim ersten Ventil im Bild unten).

Die komplette Darstellung des Skriptes *DeleteObject* ist im Bild aufgrund der Zeilenlänge nicht mehr lesbar darstellbar. Dazu sollte der Skript im TIA-Projekt direkt am Monitor betrachtet werden.

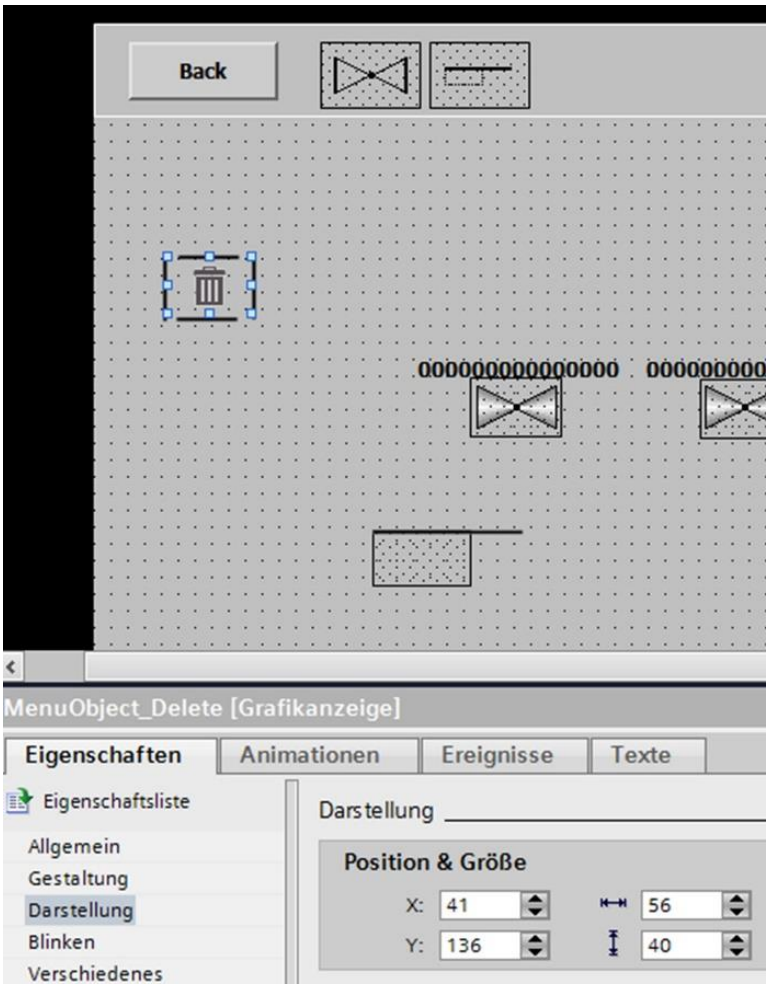


Bild 6.11 Die Lage des Mülleimers *MenuObject_Delete*

```
1  Sub DeleteObject()
2
3  Dim XLeft, YTop, YDown
4
5  ' area of delete icon
6    XLeft = HmiRuntime.Screens("P&ID").ScreenItems("MenuObject_Delete").Left + HmiRu
7    YDown = HmiRuntime.Screens("P&ID").ScreenItems("MenuObject_Delete").Top + HmiRunt
8    YTop = HmiRuntime.Screens("P&ID").ScreenItems("MenuObject_Delete").Top
9
10
11 'select actual element
12   Select Case SmartTags("TypeOfObject")
13     Case 1020 'Valve hor
14       Select Case SmartTags("indexOfObject")
15         Case 0
16           If(SmartTags("DB_Valve_Hor_Valve_Hor{0}_HMI_HMI_XPos") < XLeft And SmartT
17             SmartTags("DB_Valve_Hor_Valve_Hor{0}_HMI_HMI_Visible") = False
18           End If
```

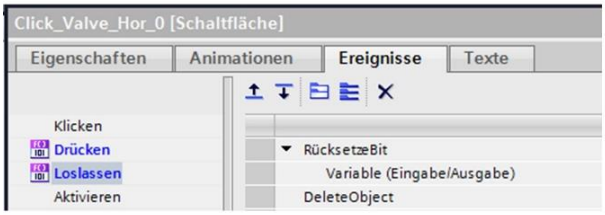


Bild 6.12 Der Bildausschnitt vom Skript *DeleteObject*

7 Objekte in der Größe verändern

In **Kapitel 6** wurde das Bild-Objekt Linie angewendet. Sie muss nicht nur bewegt, sondern auch in ihrer Größe verändern werden. Betrachtet man in **Kapitel 2** die **Tabelle 2.2**, ist zu erkennen, dass das Basis-Objekt *Linie* in der Breite und Höhe (*Width, Height*) nicht änderbar ist (nur *Read*-Funktionen). Für Linien muss deswegen das Basis-Objekt *Rechteck* verwendet werden. Tatsächlich ist das auch bereits in **Kapitel 6** als Rechteck angewendet worden und wurde mit einer Rahmenbreite von 2 gezeichnet.

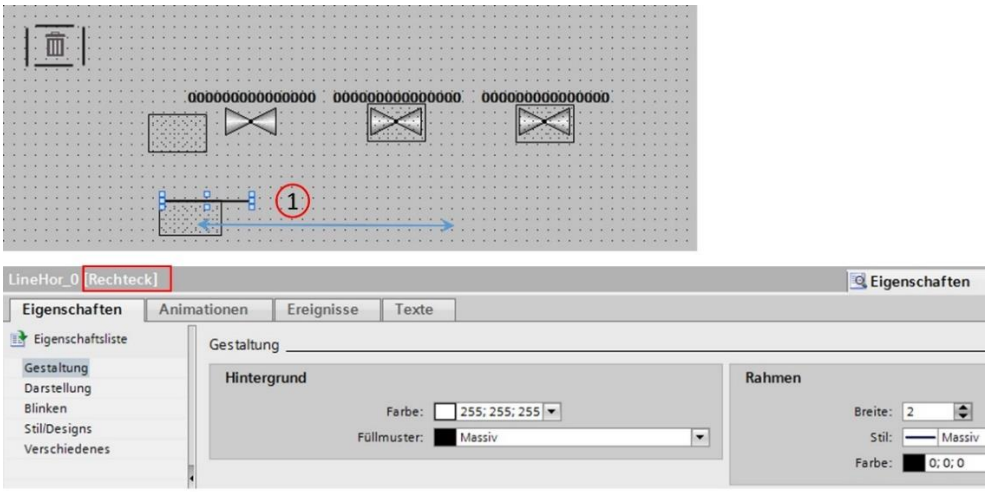


Bild 7.1 Das Basis-Element Rechteck soll als Linie vergrößert werden

Betrachtet man das **Bild 7.1**, so kann man feststellen, dass die Klick-Fläche einmal für die Bewegung und einmal für die Größenanpassung benutzt wird (**Punkt 1**). Es wird eine Kennung benötigt, die im Skript *SelectObject* zwischen beiden Funktionen unterscheiden kann. Damit diese Anwendung einen praktischen Bezug hat, soll ein bewegter Menü-Button die Lösung demonstrieren.

7.1 Der Main-Cursor

In **Bild 7.2** ist der Main-Cursor zu sehen. Insgesamt besteht er aus 14 Elementen. Die im Bild sichtbaren Elemente, einschließlich der vier weißen Flächen. Diese müssen alle gleichzeitig mit der Maus bewegt werden, wenn auf die mittlere Schaltfläche *MoveMenu* gedrückt wird. Das erledigt der Skript *MoveObjectMainMenu.* Im Ereignis *Drücken* (**Bild 7.3**) der Schaltfläche *MoveMenu* wird diesmal nur der *TypeOfObject und requestUserAction* eingetragen. *Der sonst übliche IndexOfObject* entfällt, da sich üblicherweise nur ein Main-Cursor im Projekt befindet. Der *TypeOfObject* mit 9999 kann dann direkt in *SelectObject* abgefangen werden. Alle Bildobjekte in diesem TIA-Projekt zum Kapitel, werden in der

Skriptleiste beim Loslassen der Maustaste mit dem Skript *DeleteObject* anstatt *Move2LeftGrid* aufgerufen, da hier kein Positionieren auf dem Raster erfolgen soll.

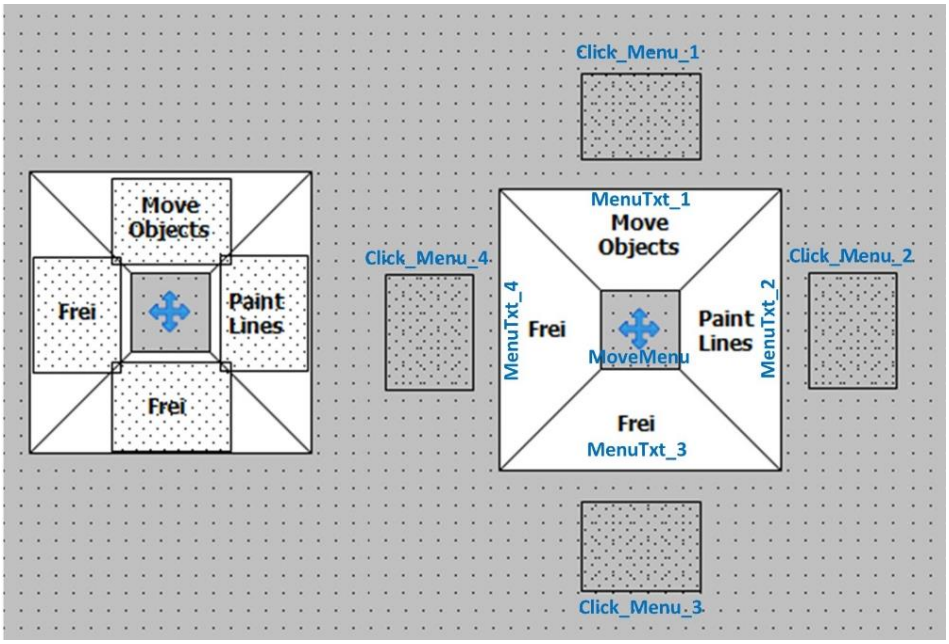


Bild 7.2 Der Menü-Button besteht aus insgesamt 14 Elementen

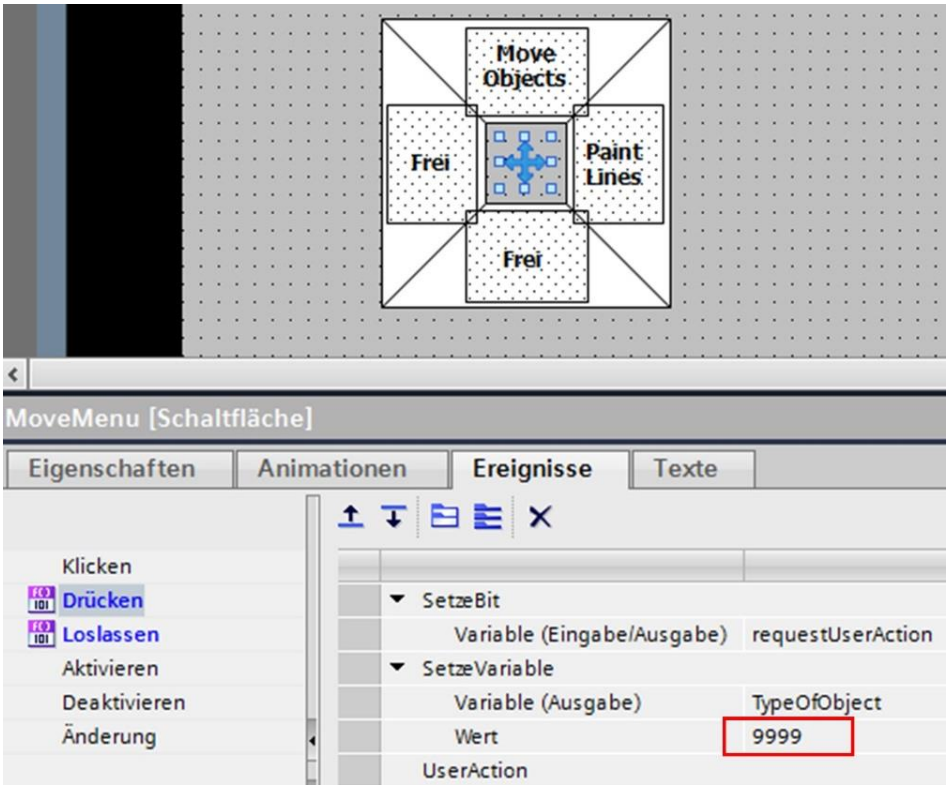


Bild 7.3 Das Ereignis *Drücken* für *MoveMenu* wird in *TypeOfObject* mit 9999 angegeben

7.1.1 Der Skript *MoveObjectMainMenu*

Wie aus **Bild** 7.3 ersichtlich, wird nach *SetzeVariable* der Skript *UserAction* aufgerufen, welcher dann den Skript *SelectObjects* aufruft. Der *TypeOfObject* ist in *SelectObject* bekannt, muss aber dort noch auf den Wert 9999 programmiert werden. In **Bild 7.4** ist der Aufruf des Skriptes *MoveObjectMainMenu* in Zeile 17 zu sehen (siehe zum Vergleich dazu **Bild 3.14**). Danach wird *SelectObject* durch Zeile 15 verlassen, da eine weitere Bearbeitung anderer Objekte nicht mehr notwendig ist. Dieser Zyklus wird solange wiederholt, bis die Maustaste losgelassen wird. Im Ereignis *Loslassen* kann auf den Aufruf *DeleteObject* verzichtet werden, da der Main-Cursor in diesem Skript nicht bearbeitet wird.

```
1  Sub SelectObject()
2
3  Const BorderX = 2 ' 2% from the screen width
4  Const BorderY = 2 ' 2% from the screen height
5  Const VERTICAL = False
6  Const HORIZONTAL = True
7
8  Dim ObjectScreen_Width, ObjectScreen_Height
9  Dim ObjectScreen_Left, ObjectScreen_Top
10 Dim Object, ClickObject, ObjectText, ObjectUnit, ObjectValue
11
12 ' call main-cursor
13   If(SmartTags("TypeOfObject")=9999)Then
14     MoveObjectMainMenu
15     Exit Sub
16   End If
17 'end main-cursor
```

Bild 7.4 In *SelectObject* wird der Menu-Cursor mit dem Index 9999 abgefangen

In **Bild 7.5** wird der erste Zyklus zur Berechnung der *Offsets* gezeigt. Im Vergleich dazu (**Bild 3.16**) werden hier alle 13 Elemente des Main-Cursors im *Offset-Array* gespeichert. Das scheint sehr aufwendig zu sein; trotzdem ist das *Array* noch lange nicht ausgelastet. In den HMI-Variablen *HMI_UserAction* ist das *Array* für 32 Elemente deklariert (siehe **Bild 3.13**). Dieses Beispiel zeigt, dass trotz der vielen Elemente sich der Main-Cursor fließend über die Oberfläche bewegen lässt.

Die VBS-Skripte in WinCC haben demnach eine sehr gute Performance, sodass der zyklische Ablauf zu diesem Beispiel noch keine Grenzen erkennen lässt und noch mehr leisten kann als vermutet wird.

Im nächsten Abschnitt (**Bild 7.6**) wird dann der *Offset* entsprechend den Elementen verrechnet (Zeilen 67+68). Im Gegensatz zum Skript *Move2Objects* (**Bild 3.16**), werden hier die Bedingungen der Bildgrenzen für die X- und Y-Achsen einzeln ausgewertet. Z. B. zeigt Zeile, dass der Skript mit *Exit Sub* verlassen wird, wenn die X-Position außerhalb der Grenzen liegt. So kann der Menü-Cursor nur bewegt werden, wenn er sich innerhalb der Bildfläche befindet. Bei *Move2Objects* können im Gegensatz dazu die Bild-Objekte z.B. auf der Y-Achse weiter verschoben werden, auch wenn die X-Achse sich bereits außerhalb der Grenzen befindet. Da der Main-Cursor ein Einzelobjekt ist, sollen nur die Klicks im Vordergrund stehen. Deswegen wird auf die Positionierung nicht so viel Wert gelegt.

```
Sub MoveObjectMainMenu()
' const and dim
Const RandX = 2 ' %
Const RandY = 2 ' %
Dim Max_X, Max_Y, Min_X, Min_Y
Dim tempX_Obj_0, tempY_Obj_0, tempX_Obj_1, tempY_Obj_1
Dim xPos,yPos

'Screen limits
Max_X = HmiRuntime.Screens("P&ID").Width - ((HmiRuntime.Screens("P&ID").Width / 100) * RandX)
Max_Y = HmiRuntime.Screens("P&ID").Height - ((HmiRuntime.Screens("P&ID").Height / 100) * RandY)
Min_X = (HmiRuntime.Screens("P&ID").Width / 100) * RandX
Min_Y = (HmiRuntime.Screens("P&ID").Height / 100) * RandY

' first call init
If( Not SmartTags("userActionActiv") ) Then
  xPos = SmartTags("mouseXpos")
  yPos = SmartTags("mouseYpos")

  SmartTags("XOffset")(0) = (xPos - HmiRuntime.Screens("P&ID").ScreenItems("MoveMenu").Left)
  SmartTags("YOffset")(0) = (yPos - HmiRuntime.Screens("P&ID").ScreenItems("MoveMenu").Top)
  SmartTags("XOffset")(1) = (xPos - HmiRuntime.Screens("P&ID").ScreenItems("Menu_Trapez_1").Left)
  SmartTags("YOffset")(1) = (yPos - HmiRuntime.Screens("P&ID").ScreenItems("Menu_Trapez_1").Top)
  SmartTags("XOffset")(2) = (xPos - HmiRuntime.Screens("P&ID").ScreenItems("Menu_Trapez_2").Left)
  SmartTags("YOffset")(2) = (yPos - HmiRuntime.Screens("P&ID").ScreenItems("Menu_Trapez_2").Top)
  SmartTags("XOffset")(3) = (xPos - HmiRuntime.Screens("P&ID").ScreenItems("Menu_Trapez_3").Left)
  SmartTags("YOffset")(3) = (yPos - HmiRuntime.Screens("P&ID").ScreenItems("Menu_Trapez_3").Top)
  SmartTags("XOffset")(4) = (xPos - HmiRuntime.Screens("P&ID").ScreenItems("Menu_Trapez_4").Left)
  SmartTags("YOffset")(4) = (yPos - HmiRuntime.Screens("P&ID").ScreenItems("Menu_Trapez_4").Top)
  SmartTags("XOffset")(5) = (xPos - HmiRuntime.Screens("P&ID").ScreenItems("MenuTxt_1").Left)
  SmartTags("YOffset")(5) = (yPos - HmiRuntime.Screens("P&ID").ScreenItems("MenuTxt_1").Top)
  SmartTags("XOffset")(6) = (xPos - HmiRuntime.Screens("P&ID").ScreenItems("MenuTxt_2").Left)
  SmartTags("YOffset")(6) = (yPos - HmiRuntime.Screens("P&ID").ScreenItems("MenuTxt_2").Top)
  SmartTags("XOffset")(7) = (xPos - HmiRuntime.Screens("P&ID").ScreenItems("MenuTxt_3").Left)
  SmartTags("YOffset")(7) = (yPos - HmiRuntime.Screens("P&ID").ScreenItems("MenuTxt_3").Top)
  SmartTags("XOffset")(8) = (xPos - HmiRuntime.Screens("P&ID").ScreenItems("MenuTxt_4").Left)
  SmartTags("YOffset")(8) = (yPos - HmiRuntime.Screens("P&ID").ScreenItems("MenuTxt_4").Top)
  SmartTags("XOffset")(9) = (xPos - HmiRuntime.Screens("P&ID").ScreenItems("Click_Menu_1").Left)
  SmartTags("YOffset")(9) = (yPos - HmiRuntime.Screens("P&ID").ScreenItems("Click_Menu_1").Top)
  SmartTags("XOffset")(10) = (xPos - HmiRuntime.Screens("P&ID").ScreenItems("Click_Menu_2").Left)
  SmartTags("YOffset")(10) = (yPos - HmiRuntime.Screens("P&ID").ScreenItems("Click_Menu_2").Top)
  SmartTags("XOffset")(11) = (xPos - HmiRuntime.Screens("P&ID").ScreenItems("Click_Menu_3").Left)
  SmartTags("YOffset")(11) = (yPos - HmiRuntime.Screens("P&ID").ScreenItems("Click_Menu_3").Top)
  SmartTags("XOffset")(12) = (xPos - HmiRuntime.Screens("P&ID").ScreenItems("Click_Menu_4").Left)
  SmartTags("YOffset")(12) = (yPos - HmiRuntime.Screens("P&ID").ScreenItems("Click_Menu_4").Top)
  SmartTags("userActionActiv") = True
  Exit Sub
End If
```

Bild 7.5 Der erste Zyklus im Skript *MoveObjectMainMenu*

```

' control move and move only Click button
  xPos = SmartTags("mouseXpos")
  yPos = SmartTags("mouseYpos")
  tempX_Obj_0 = xPos - SmartTags("XOffset")(1)
  If( (tempX_Obj_0 > (Max_X - HmiRuntime.Screens("P&ID").ScreenItems("Menu_Trapez_1").Width
    Exit Sub
  End If
  tempY_Obj_0 = yPos - SmartTags("YOffset")(1)
  If( tempY_Obj_0 < Min_Y )Then
    Exit Sub
  End If
  tempY_Obj_0 = yPos - SmartTags("YOffset")(7)
  If( (tempY_Obj_0 + HmiRuntime.Screens("P&ID").ScreenItems("Menu_Trapez_3").Height) > Max_
    Exit Sub
  End If

  ' set object coordinates
  HmiRuntime.Screens("P&ID").ScreenItems("MoveMenu").Left = xPos - SmartTags("XOffset")(0)
  HmiRuntime.Screens("P&ID").ScreenItems("MoveMenu").Top = yPos - SmartTags("YOffset")(0)
```

Bild 7.6 Skript-Ausschnitt *MoveObjectMainMenu*

7.1.2 Programmierung und Test des Main-Cursors

In der HMI hat der Main-Cursor in den *Eigenschaft/Animation* einige Bedingungen (**Bild 7.7**) bezüglich der Gestaltung und Sichtbarkeit.

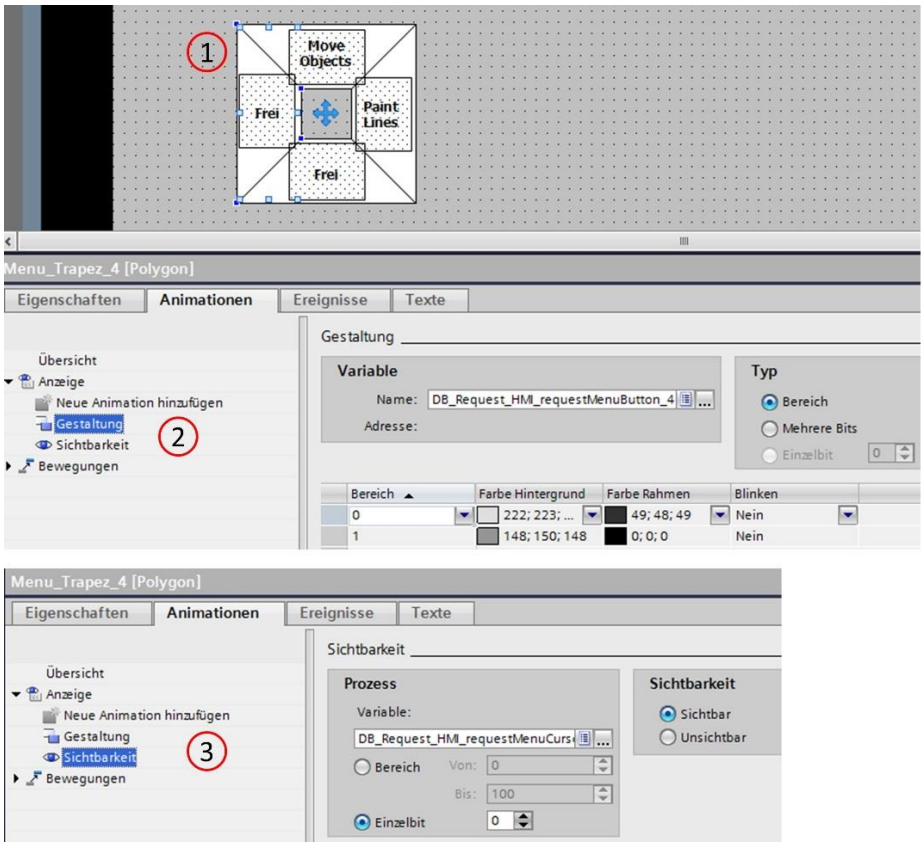


Bild 7.7 Die Eigenschaften des Main-Cursors für die PLC

Das **Bild 7.7** zeigt am Beispiel der weißen Fläche (**Punkt 1**), dass in der Gestaltung die PLC-Variable *_requestMenuButton_4* die Fläche dunkelgrau gestaltet wird (**Punkt 2**), wenn diese Variable den Status 1=*True* besitzt. In der PLC ist diese Variable in *HMI_UserAction* vom Typ *Bool* deklariert. Das gilt für die übrigen 3 weißen Flächen entsprechend mit den PLC-Variablen *_requestMenuButton_3 bis _requestMenuButton_1*. In der Sichtbarkeit (**Punkt 3**) wird die Sichtbarkeit von Main-Cursor mit der PLC-Variablen *_requestMenuCursorVisible* geregelt.

Hat *_requestMenuCursorVisible* den Status *True,* dann ist der gesamte Main-Cursor sichtbar, da alle Elemente diese Verknüpfung zur Sichtbarkeit besitzen. Die PLC-Variable *_requestMenuCursorVisible* wird in der Deklaration in der SPS mit dem Startwert *True* versehen, sodass beim Starten der SPS der Main-Cursor vorerst sichtbar ist. Dieser kann

dann im Anwenderprogramm beliebig umgeschaltet werden. Wie im **Bild 7.8** ersichtlich ist, schiebt sich der Main-Cursor über die Bild-Objekte, da seine Elemente auf den Layer 28 gezeichnet sind. In der Mitte des Cursors kann man das Bild-Element Ventil sehen. Eine pfiffige Idee, finde ich ☺.

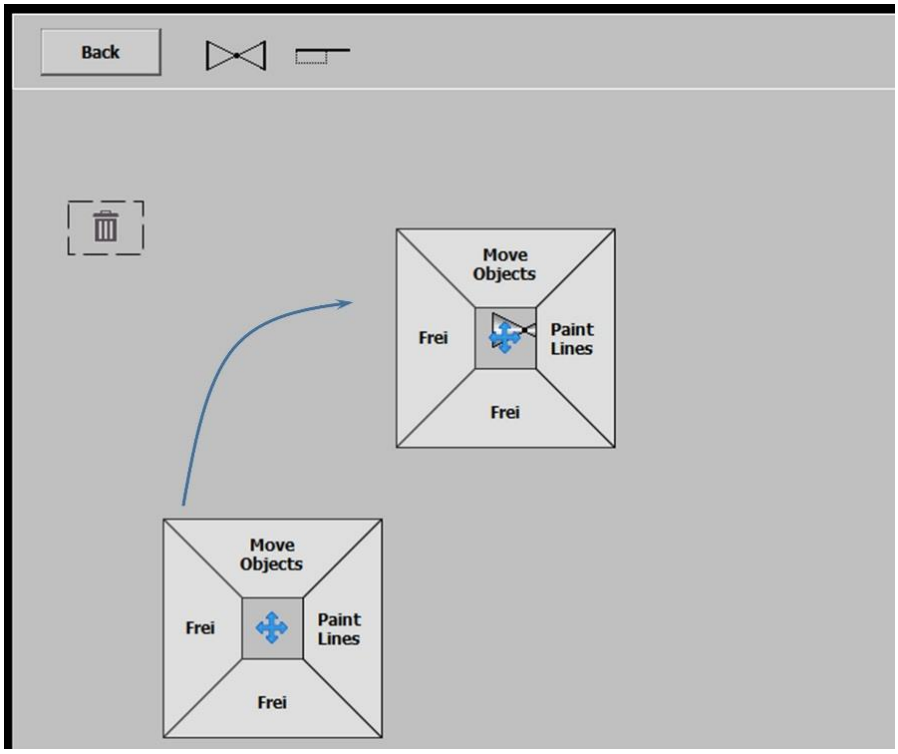


Bild 7.8 Der Main-Cursor ist in Layer 28 gezeichnet und schiebt sich über die Bild-Objekte

Drückt man nun auf eine der Schaltflächen, so wird diese dunkelgrau. Es kann immer nur eine Schaltfläche dunkelgrau angezeigt werden. Sind alle Schaltflächen hellgrau, ist keine Funktion angewählt. Damit jede Schaltfläche zwischen hell- und dunkelgrau wechseln kann, muss im Ereignis das entsprechende *Bit* invertiert werden. Im **Bild** 7.9 ist die Anwendung für die Schaltfläche *Click_Menu_1* (*Move Objects*) dargestellt. Das gilt entsprechend für die restlichen Schaltflächen. In der SPS werden die Variablen dann so ausgewertet (**Bild** 7.10), dass die beschriebene Funktion gewährleistet wird.

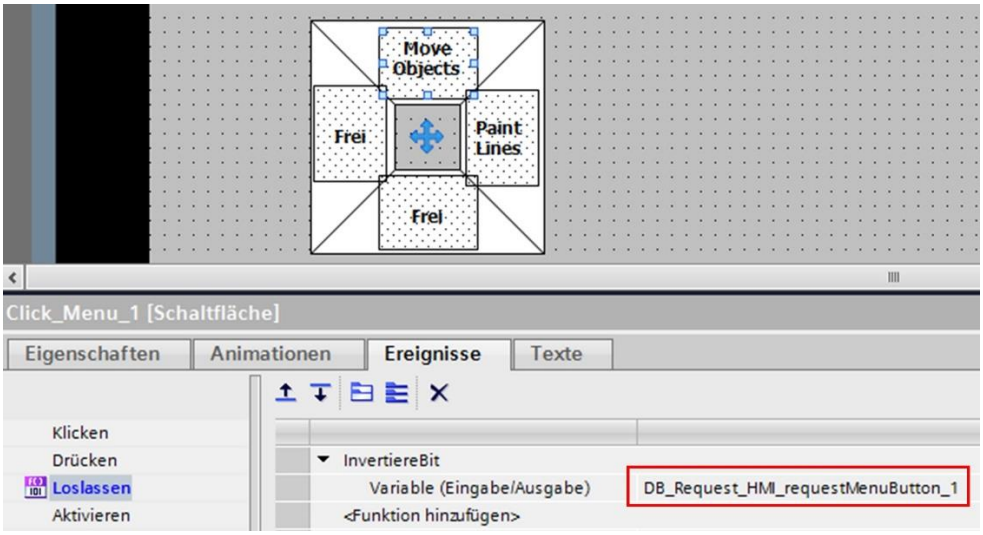


Bild 7.9 Das Ereignis *Loslassen* invertiert das Bit

7.1.2.1 Der *FB_MenuControl*

Wird die Schaltfläche für *Move Object* gedrückt, so wird die Variable *requestMenuButton_1* durch die HMI auf *True* gesetzt. Im *FB_MenuControl* werden alle vier Schaltflächen über eine XOR-Verknüpfung ausgewertet **(Bild 7.10, Punkt 1**). Es kann also nur eine Variable den Status *True* besitzen um in das *IF-Statement* zu verzweigen. Dort wird die entsprechend gedrückte Schaltfläche in die Variable *#static_MemberRequestButton_1* gespeichert (**Bild** 7.10, **Punkt 2**). Dieser Zustand bleibt solange erhalten, bis die gleiche oder eine andere Schaltfläche gedrückt wird.

Ist das der Fall, verzweigt das *IF-Statement* in den *Else*-Teil. Dort wird ausgewertet, welche Schaltfläche gespeichert und (*AND*) welche Schaltfläche zusätzlich gedrückt wurde. In (**Bild** 7.11) wurde die *IF*-Anweisung für die Schaltfläche *MoveObject* zur besseren Lesbarkeit auf die Zeilen 17-23 aufgeteilt. Wurde z.B. die Schaltfläche *PaintLines* gedrückt, so werden die Variablen *requestMenuButton_1* und der Speicher *#static_MemberRequestButton_1* wieder gelöscht und es bleibt nur die Variable *requestMenuButton_2* mit *True* erhalten. Nach dem *Else*-Teil ist also im nächsten SPS-Zyklus der *IF*-Teil wieder aktiv und der Speicher *#static_MemberRequestButton_2* wird gesetzt, denn nur *requestMenuButton_2* ist *True,* sodass die *XOR*-Anweisung wieder zuschlägt. Diese Erklärung gilt dann analog für jede Schaltfläche, welche zusätzlich gedrückt wurde.

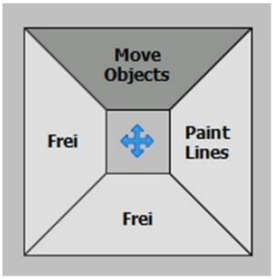
Wird z. B. die Schaltfläche *Move Object* gedrückt und danach dieselbe Schaltfläche noch einmal, dann wird im *Else*-Teil keine Übereinstimmung gefunden, da keine Variable der Buttons den Zustand *True* hat.

Es bleibt somit der Speicher *#static_MemberRequestButton_1* solange gespeichert, bis eine beliebige andere Schaltfläche gedrückt wird. In diesem Fall ist die *XOR*-Anweisung wieder gültig und es wird wieder der aktuelle Speicher gesetzt und alle anderen Speicher gelöscht.

FB_MenuControl

	Name	Datentyp
1	Input	
2	<Hinzufügen>	
3	Output	
4	<Hinzufügen>	
5	InOut	
6	<Hinzufügen>	
7	Static	
8	static_MemberRequestButton_1	Bool
9	static_MemberRequestButton_2	Bool
10	static_MemberRequestButton_3	Bool
11	static_MemberRequestButton_4	Bool
12	Temp	
13	tempBool	Bool

3




```
1  // FB_MenuControl
2  // control the buttons of the main-cursor
3  // only one button is valid
4
5  IF ("DB_Request_HMI".requestMenuButton_1
6    XOR
7    "DB_Request_HMI".requestMenuButton_2
8    XOR
9    "DB_Request_HMI".requestMenuButton_3
10   XOR
11   "DB_Request_HMI".requestMenuButton_4) THEN
12   #static_MemberRequestButton_1 := "DB_Request_HMI".requestMenuButton_1;
13   #static_MemberRequestButton_2 := "DB_Request_HMI".requestMenuButton_2;
14   #static_MemberRequestButton_3 := "DB_Request_HMI".requestMenuButton_3;
15   #static_MemberRequestButton_4 := "DB_Request_HMI".requestMenuButton_4;
16 ELSE
```

1

2

Bild 7.10 Der *FB_MenuControl*

Mit dieser Lösung kann nur eine Variable der Schaltflächen den Status *True* besitzen. Die Variablen der Schaltflächen *requestMenuButton_1* bis *requestMenuButton_4* können nun entsprechend im Skript *SelectObject* ausgewertet werden. Das Verschieben der Objekte kann damit mit *requestMenuButton_1* verknüpft werden. Hat *requestMenuButton_2* den Status *True,* so wird mit dem Klick auf die Schaltfläche der Linie, diese nicht verschoben, sondern vergrößert (**Kapitel 7.2**).

Sollten in der HMI in RT einmal mehrere Schaltflächen zum Main-Cursor dunkelgrau sein, dann sollte die HMI noch einmal **komplett** übersetzt werden. Achten Sie auch darauf, dass die Variablen in der HMI für den Main-Cursor auf den Erfassungszyklus von 100ms stehen.

```
17  IF ((#static_MemberRequestButton_1
18    AND
19    ("DB_Request_HMI".requestMenuButton_2
20    OR
21    "DB_Request_HMI".requestMenuButton_3
22    OR
23    "DB_Request_HMI".requestMenuButton_4))) THEN
24     "DB_Request_HMI".requestMenuButton_1 := False;
25     #static_MemberRequestButton_1 := False;
26      RETURN;
27  END_IF;
28  IF ((#static_MemberRequestButton_2 AND ("DB_Request_HMI"
29    "DB_Request_HMI".requestMenuButton_2 := False;
30    #static_MemberRequestButton_2 := False;
31    RETURN;
32  END_IF;
33  IF ((#static_MemberRequestButton_3 AND ("DB_Request_HMI"
34    "DB_Request_HMI".requestMenuButton_3 := False;
35    #static_MemberRequestButton_3 := False;
36    RETURN;
37  END_IF;
38  IF ((#static_MemberRequestButton_4 AND ("DB_Request_HMI"
39    "DB_Request_HMI".requestMenuButton_4 := False;
40    #static_MemberRequestButton_4 := False;
41    RETURN;
42  END_IF;
43
44 END_IF;
```

Bild 7.11 Schaltfläche wieder löschen

7.2 Der Skript *ZoomLines*

In **Kapitel 7.1** wird der Main-Cursor vorgestellt. Dieser setzt die Variablen der einzelnen Schaltflächen des Main-Cursors. So können wir nun im Skript *SelectObject* eine Zuordnung treffen, ob wir das Bild-Objekt bewegen oder in der Größe verändern möchten. Damit auch die Vergrößerung einer vertikalen Linie gezeigt wird, wurde das TIA-Projekt zu diesem Kapitel aus **Kapitel 6** kopiert und entsprechend um eine vertikale Linie erweitert (**Bild** 7.12, **Punkt 1**). Dazu wurde ein neuer DB in der PLC eingerichtet (**Punkt 2**) und die entsprechenden Verbindungen in den HMI-Variablen eingetragen (**Punkt 3**).

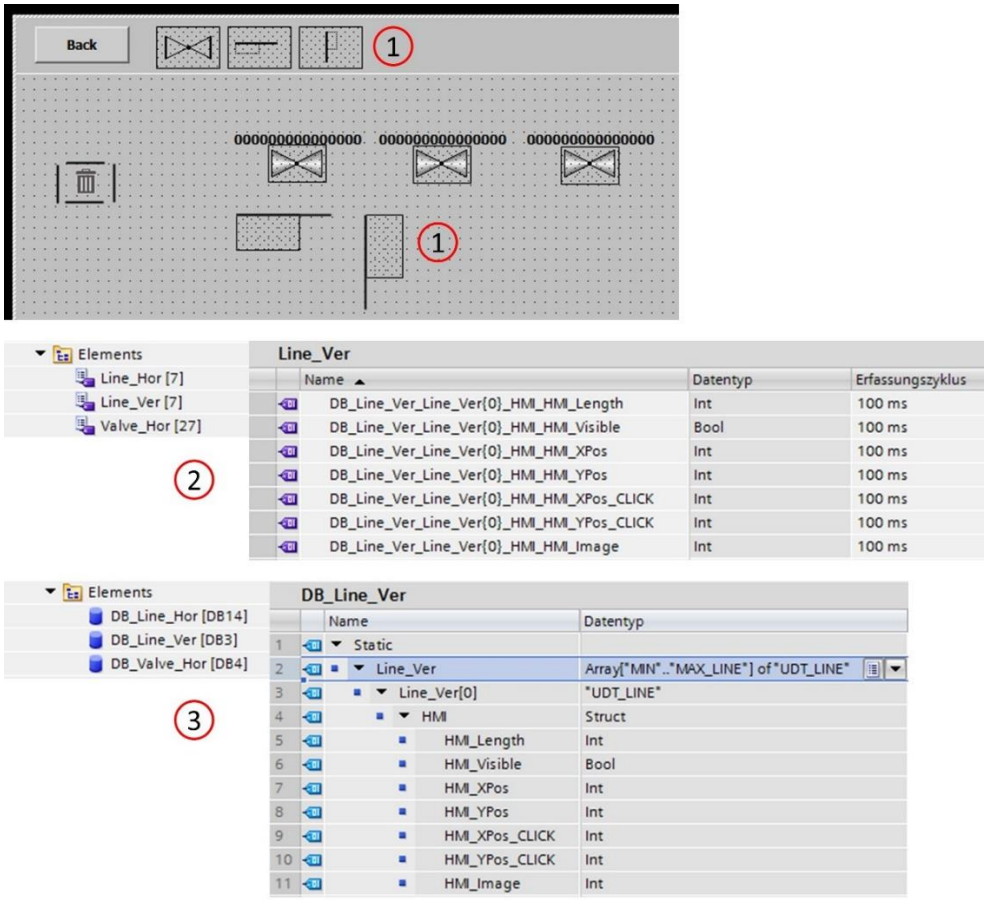


Bild 7.12 Erweiterung mit einer vertikalen Linie

Im Skript *SelectObject* (**Bild** 7.13) wird nun die Schaltfläche für *Paint Lines* eingefügt (**Punkt 1**). Mit *TypeOfObject* kann nun die horizontale oder die vertikale Linie selektiert werden (**Punkt 2+3**).

Der Aufruf für den Skript *ZoomLines* (**Punkt 4**) unterscheidet sich im letzten Parameter, damit der Skript weiß, ob er die Höhe oder die Breite *zoomen* soll. Denn bei dem Linien-Bild-Objekt handelt es sich um ein Rechteck.

```
24 ' main-cursor-button PaintLines
25   If( SmartTags("DB_Request_HMI_requestMenuButton_2") )Then   (1)
26     'line script
27       Select Case SmartTags("TypeOfObject")
28     (2) Case 1060 'Line hor
29           Select Case SmartTags("indexOfObject")
30             Case 0
31               Set Object =  HmiRuntime.Screens("P&ID").ScreenItems("LineHor_0")
32               Set ClickObject = HmiRuntime.Screens("P&ID").ScreenItems("Click_LineHor_0")
33               SmartTags("DB_Line_Hor_Line_Hor{0}_HMI_HMI_Length") = Object.Width
34             Case 1
35               ' and so on
36           End Select 'indexOfObject
37       (4) ZoomLines Object,ClickObject,ObjectScreen_Width,ObjectScreen_Height, HORIZONTAL
38           Exit Sub
39
40     (3) Case 1061 'Line ver
41           Select Case SmartTags("indexOfObject")
42             Case 0
43               Set Object =  HmiRuntime.Screens("P&ID").ScreenItems("LineVer_0")
44               Set ClickObject = HmiRuntime.Screens("P&ID").ScreenItems("Click_LineVer_0")
45               SmartTags("DB_Line_Ver_Line_Ver{0}_HMI_HMI_Length") = Object.Height
46             Case 1
47             ' and so on
48           End Select 'indexOfObject
49       (4) ZoomLines object,ClickObject,ObjectScreen_Width,ObjectScreen_Height,VERTICAL
50           Exit Sub
51           End Select 'indexOfObject
52       Exit Sub
53     'end line script
54   End If
55   'end cursor-button PaintLines
```

Bild 7.13 Der Skript *SelectObject* wird erweitert für die Zoom-Technik

Bei *VERTICAL und HORIZONTAL* handelt es sich um Konstanten, wobei *VERTICAL* auf *False* und *HORIZONTAL* auf *True* festgelegt ist. Zusätzlich wurde der Skript *NewObject* um die vertikale Linie erweitert, damit das neue Objekt auch von der Menüleiste eingefügt werden kann. Zu guter Letzt auch der Skript *DeleteObject* und *InitObjects*, sonst könnte man die vertikale Linie nicht mehr löschen und beim Bildwechsel würde die vertikale Linie nicht ordnungsgemäß gezeichnet. Die drei Erweiterungen sind Kopien aus der Linie horizontal mit Änderung der Bezeichnung, deswegen wird auf ein Bild im Buch verzichtet. Das gleiche gilt auch für den *FB_ObjectControl* in der SPS, welcher um die vertikale Linie erweitert wurde.

Es ist darauf zu achten, dass bei vertikalen Linien immer die Höhe (*Height*) und bei horizontalen Linien immer die Breite (*Width*) verwendet wird.

Betrachten wir nun das zoomen im Skript *ZoomLines* (**Bild 7.14**), so erkennen wir eine ähnliche Struktur wie beim Verschieben der Objekte (**Bild 3.16**). Im ersten Schritt wird im Offset-Array die aktuelle Mausposition gespeichert (**Punkt 1**). In den Zeilen 15 bis 29 wird der Zoom-Bereich geprüft, damit die Linie nicht außerhalb des Bildbereiches gezoomt wird. Zusätzlich wird hier die Größe mit der Konstanten MIN_ZOOM überwacht (Zeilen 20+27).

Diese Konstante ist auf 2 Pixel eingestellt. So wird verhindert, dass die Linie nicht zu klein wird und auf dem Monitor verloren geht. Das eigentliche Zoomen findet im letzten Abschnitt statt (**Punkt 2**).

Der Parameter *Direction* bestimmt ob die Höhe oder die Breite gezoomt wird. Ist die Linie horizontal (*True*), dann wird die Breite (*Width*), sonst die Höhe (*Height*) verändert.

```
1  Sub ZoomLines(ByRef Object, ByRef ObjectClick,
2
3  ' const and dim                ByRef ScreenMax_X, ByRef ScreenMax_Y, ByRef Direction)
4  Const MIN_ZOOM = 2
5  Dim tempX_Object, tempY_Object
6
7  ' first call init
8  If( Not SmartTags("userActionActiv") ) Then
9    SmartTags("XOffset")(0) = SmartTags("mouseXpos")
10   SmartTags("YOffset")(0) = SmartTags("mouseYpos")
11   SmartTags("userActionActiv") = True
12   Exit Sub                                          1
13 End If
14
15 ' control zoom
16   tempX_Object = SmartTags("mouseXpos") - SmartTags("XOffset")(0)
17   If( Object.Left + (Object.Width + tempX_Object ) > ScreenMax_X) Then
18     Exit Sub
19   End If
20   If( (Object.Width + tempX_Object ) <  MIN_ZOOM) Then
21     Exit Sub
22   End If
23   tempY_Object = SmartTags("mouseYpos") - SmartTags("YOffset")(0)
24   If( Object.Top + (Object.Height + tempY_Object ) > ScreenMax_Y) Then
25     Exit Sub
26   End If
27   If( (Object.Height + tempY_Object ) <  MIN_ZOOM) Then
28     Exit Sub
29   End If
30
31   ' set object width and height                      2
32   If(Direction)Then
33     Object.Width = Object.Width + tempX_Object
34     SmartTags("XOffset")(0) = SmartTags("mouseXpos")
35   Else
36     Object.Height = Object.Height + tempY_Object
37     SmartTags("YOffset")(0) = SmartTags("mouseYpos")
38   End If
39
40 End Sub
```

Bild 7.14 Der Skript ZoomLines

In **Bild** 7.15 ist das Ergebnis sichtbar. Die vertikale Linie wird gerade vergrößert. Die Schaltfläche ist gut zu erkennen. Sie dient auch als Orientierung zum anliegenden Objekt (hier die horizontale Linie). In dieser Übung ist nur die Schaltfläche *PaintLines* berücksichtigt. Wird diese wieder ausgeschaltet, können die Objekte wieder verschoben werden, obwohl die Schaltfläche *MoveObjects* nicht gedrückt ist.

Außerdem ist es möglich Objekte in der Menüleiste anzuklicken. Diese werden zuerst mit dem Skript *NewObject* auf die Oberfläche in der Nähe der Schaltfläche positioniert. Können bei eingeschalteter Schaltfläche *PaintLines* allerdings nicht mehr bewegt werden. Dafür sorgt der *Exit Sub* in Zeile 52 (**Bild 7.13**).

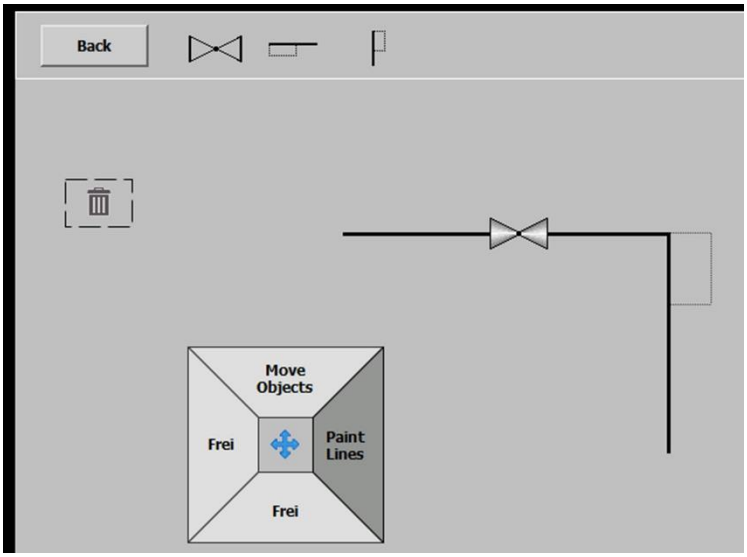


Bild 7.15 Bildelemente in der Größe verändern

8 Der Farb-Cursor

Werden Objekte mit der Maus bewegt oder dessen Größe verändert, dann besteht auch der Wunsch, dessen Farbe zu ändern. Beispielhaft ein Lager bestehend aus verschiedenen Schüttgütern, wie Gips oder Klinker, welche in der Zementverarbeitung benötigt werden, haben viele rechteckige Felder mit der farblichen Materialunterscheidung. Die Farbzuordnung wird normalerweise permanent in der HMI über die *Animation/Gestaltung* programmiert und dann über entsprechenden Variablen die Farbwerte zugewiesen. Bei einer Änderung der Farben muss dann allerdings der Programmierer wieder auf die Baustelle. Mit dem Farb-Cursor kann der Anwender in der RT seine Farben ändern. Das TIA-Projekt zu diesem Kapitel ist aufbauend zum **Kapitel 7.1** und zeigt den Main-Cursor, welcher nun benutzt werden soll den Farb-Cursor zu aktivieren. Dieser soll dann ebenfalls mit der Maus bewegt werden.

8.1 Die Bestandteile des Farb-Cursor

Der Farb-Cursor besteht aus 33 Objekten (**Bild 8.1, Punkt 1**) bestehend aus einer Hauptschaltfläche (**Punkt 2**), welche den gesamten Farb-Cursor mit der Maus bewegen soll und 16 Farben bestehend aus einem Rechteck (**Punkt 3**) und einer Schaltfläche (**Punkt 4**) für die Farbanwahl. So ergeben sich 33 Objekte.

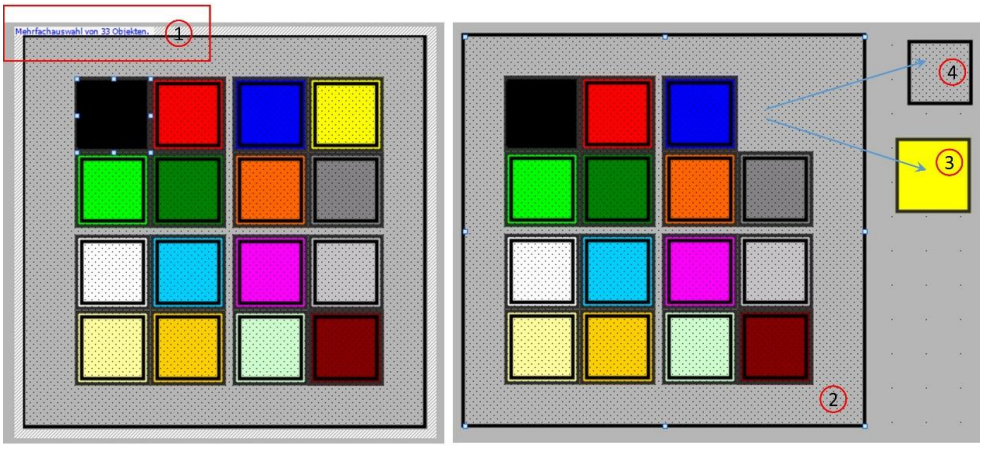


Bild 8.1 Der Farb-Cursor besteht aus 33 Objekten

Der gesamte Farb-Cursor wird mit der schon bekannten Variablen *_requestMenuButton_2* sichtbar, wenn diese *True* ist (siehe das Beispiel für den MenuButton_1 in **Bild 7.9**). In **Bild** 8.2 ist die Zuweisung der Sichtbarkeit in *Animation/Sichtbarkeit* mit *_requestMenuButton_2* zu sehen. Der Farb-Cursor kann mit der Maus an eine beliebige Stelle verschoben werden, auch wenn die Schaltfläche *MoveObjects* im Main-Cursor nicht betätigt ist. Sie verhält sich also bezüglich der Bewegung mit der Maus wie der Main-Cursor. Dafür ist der Skript

MoveObjectSetColor zuständig. In **Kapitel 7.1.1** ist der Main-Cursor bereits beschrieben, sodass auf die Beschreibung des Skriptes *MoveObjectSetColor* verzichtet werden kann, da diese funktional identisch sind.

Wird nun ein Objekt auf der Bildoberfläche angeklickt, dann soll dieses, wenn möglich, die angewählte Farbe im Farb-Cursor annehmen. Dazu betrachten wir im folgenden Kapitel die Programmierung (**Kapitel 8.2**) des Skriptes *SetColor*, welche die angewählte Farbe dem Objekt zuordnet.

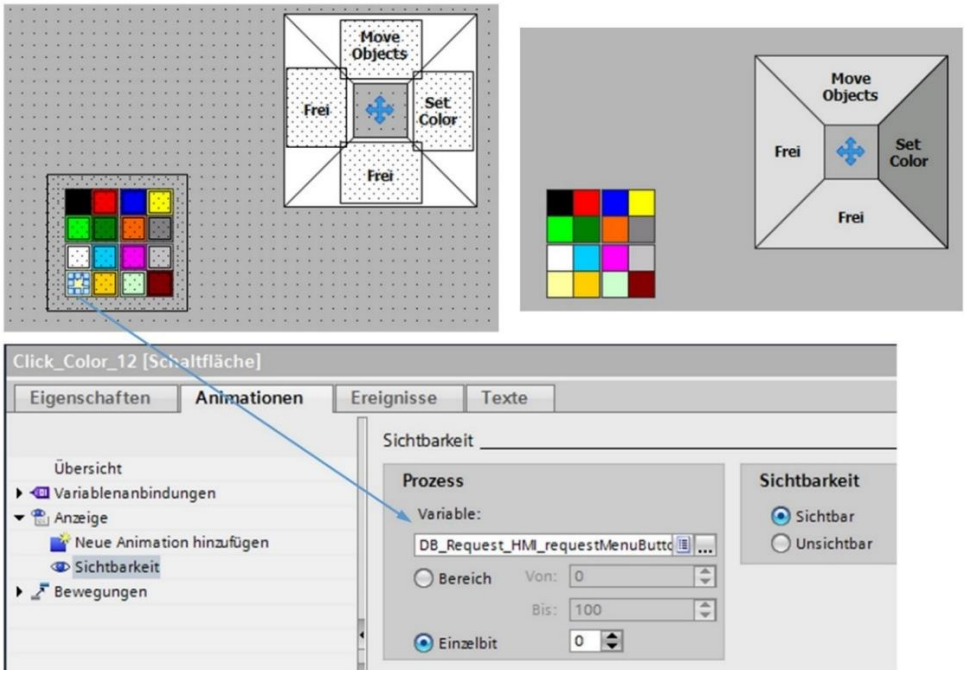


Bild 8.2 Mit *SetColor* wird der Farb-Cursor sichtbar

8.2 Der Skript *SetColor*

SetColor (**Bild** 8.3) besitzt einen Parameter (**Punkt 1**). Es handelt sich hier um ein Objekt, welches *ByRef*, also dessen Objekt-Adresse, übergeben wird (Zeiger). Dieses Objekt ist z. B. eine Kreisfläche oder ein Rechteck und kann so im Skript *SetColor* über den Zeiger *Object* direkt angesprochen werden. Dann wird die angewählte Farbe über die interne HMI-Variable *ColorNumber* selektiert. Das Setzen von *ColorNumber* erfolgt über das Ereignis und *Setze Variable* in der Funktionsliste (**Punkt 3).** Die Farben im Farb-Cursor sind entsprechend von 0-15 nummeriert und dessen RBG-Code im Listing eingetragen. Z. B. für die erste Farbe Schwarz mit RGB(0,0,0) in Zeile 3. Dieser Farb-Code wird dann dem Objekt mit *Object.BackColor* zugeordnet. Das Objekt wird über *Select Case* (Zeile 2) entsprechend der *ColorNumber* selektiert.

```
1  Sub SetColor(ByRef Object)
2    Select Case  SmartTags("ColorNumber")
3      Case 0: Object.BackColor = RGB(0, 0, 0)
4      Case 1: Object.BackColor = RGB(255, 0, 0)
5      Case 2: Object.BackColor = RGB(0, 0, 255)
6      Case 3: Object.BackColor = RGB(255, 255, 0)
7      Case 4: Object.BackColor = RGB(0,255,0)
8      Case 5: Object.BackColor = RGB(0,130,0)
9      Case 6: Object.BackColor = RGB(255,101,0)
10     Case 7: Object.BackColor = RGB(132,130,132)
11     Case 8: Object.BackColor = RGB(255,255,255)
12     Case 9: Object.BackColor = RGB(0,207,255)
13     Case 10: Object.BackColor = RGB(255,0,255)
14     Case 11: Object.BackColor = RGB(198,195,198)
15     Case 12: Object.BackColor = RGB(255,255,156)
16     Case 13: Object.BackColor = RGB(255,207,0)
17     Case 14: Object.BackColor = RGB(206,255,206)
18     Case 15: Object.BackColor = RGB(132,0,0)
19   End Select
20
21 End Sub
```

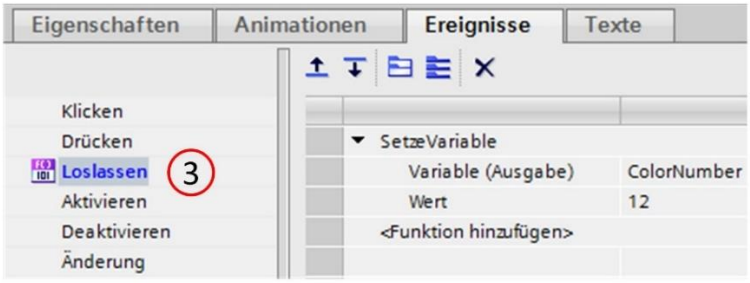


Bild 8.3 Der Skript *SetColor*

Wie wird nun *SetColor* aufgerufen? Da wir den Farb-Cursor nur sehen, wenn im Main-Cursor *SetColor* angewählt ist, wissen wir, dass ein Klick auf das Objekt den Skript auslösen soll und dieser auch bearbeitet wird. In **Bild** 8.4**, Punkt 1** ist der Aufruf an einem Beispiel in der Funktionsliste zu sehen.

Da wir das Objekt *Kreis_1* als Parameter übergeben müssen, haben wir ein Problem. Die Funktionsliste bietet uns so keine Möglichkeit den Parameter zu übergeben (deswegen ist dies rot gekennzeichnet). Da sind die Grenzen in der Funktionsliste erreicht. Deswegen müssen wir in der Funktionsliste einen Skript aufrufen, welcher dann *SetColor* aufruft. Da bietet sich der uns bekannte Skript *UserAction* an (**Bild 3.4**), welcher wieder den Skript *SelectObject* aufruft (**Bild 8.4** unten). IM Skript *SelectObject* können wir dann das Objekt wie gewohnt selektieren und den Skript *SetColor* mit seinem Parameter aufrufen.

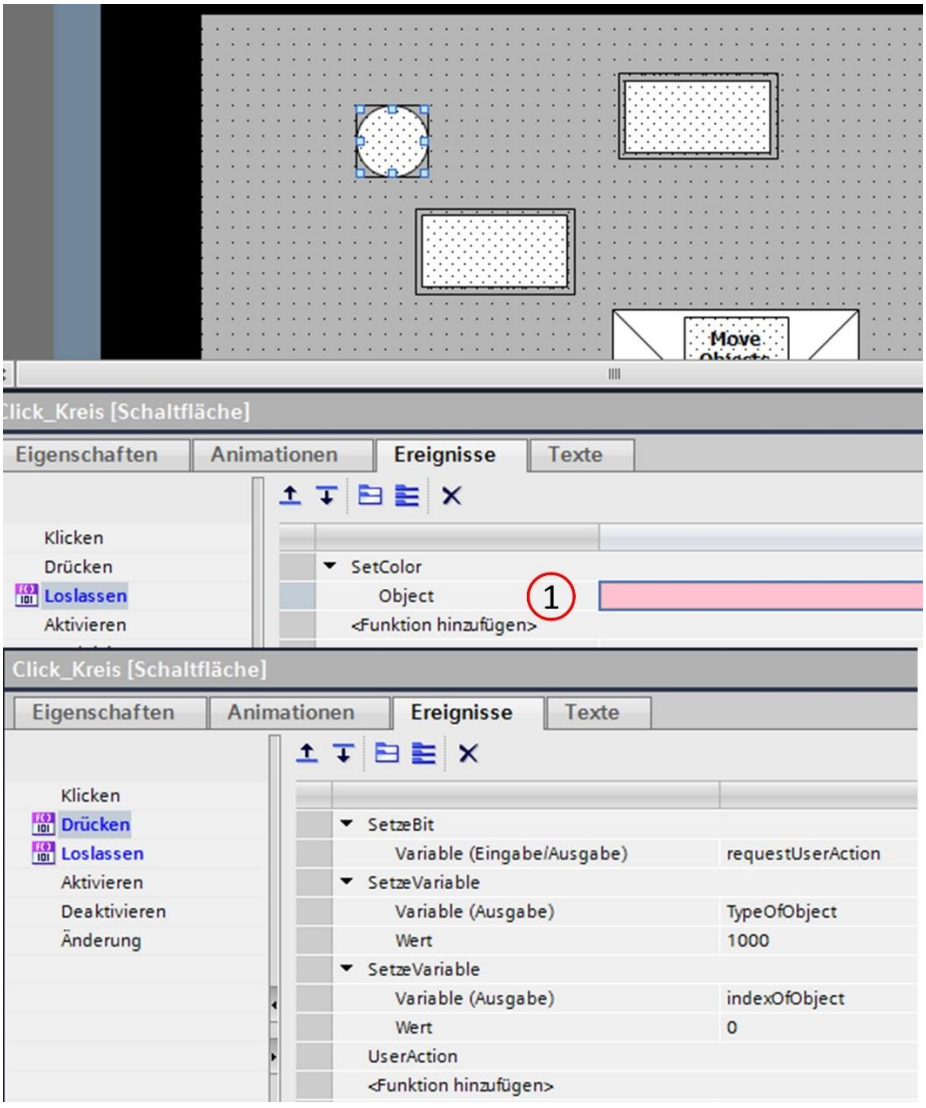


Bild 8.4 Der Aufruf erfolgt auch hier über den Skript *UserAction*

In **Bild 8.5** wird das entsprechende Objekt selektiert (**Punkt 1**). Unter **Punkt 2** sind die Aufrufe der jeweiligen Objekte für den Skript *SetColor* und im unteren Teil von **Bild** 8.5 ist das Ergebnis zu sehen.

```
'call SetColor
  If(SmartTags("DB_Request_HMI_requestMenuButton_2"))Then
    Select Case SmartTags("TypeOfObject")
      Case 1000 'Kreis
        Select Case SmartTags("indexOfObject")
          Case 0
            SetColor HmiRuntime.Screens("Main").ScreenItems("Kreis_1")'object
        End Select 'end indexOfObject
        Exit Sub
      Case 1010 'Viereck
        Select Case SmartTags("indexOfObject")
          Case 0
            SetColor HmiRuntime.Screens("Main").ScreenItems("Rechteck_1")'object
          Case 1
            SetColor HmiRuntime.Screens("Main").ScreenItems("Rechteck_2")'object
        End Select 'end indexOfObject
        Exit Sub
    End Select 'end typeOfObject

  End If
'end SetColor
```

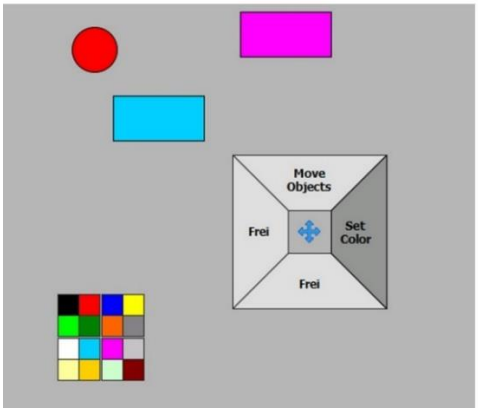


Bild 8.5 Der Skript *SelectObject* und das Ergebnis von *SetColor*

9 Objektdaten auf die Festplatte speichern und lesen

In **Kapitel 6** wurde die Menüleiste vorgestellt. Objekte werden wie bei einem Smartphone auf die Bildoberfläche gezogen und beliebig positioniert. Das Problem war der Seitenwechsel, da die Objekte wieder so gezeichnet werden, wie diese ursprünglich in der Entwicklungsumgebung gezeichnet wurden. Aus diesem Grund wurde der Skript *InitObjects* erstellt.

Nun gibt es trotzdem ein weiteres Problem, denn die Objekt-Daten für die Position werden in der SPS gespeichert. Wird diese neu initialisiert oder sogar ausgewechselt, dann ist das komplette Bild auch verloren.

Zudem besteht der Wunsch mehrere Bildseiten auf eine Ebene darzustellen. Es muss also eine Lösung gefunden werden die Objekte zu speichern, um sie dann wieder neu in die SPS zu laden, damit der Skript *InitObjects* unser gewünschtes Bild wieder komplett und richtig anzeigt.

Das TIA-Projekt zu diesem Kapitel wurde aus **Kapitel 6** übernommen und mit der Möglichkeit die Objektdaten zu speichern erweitert.

9.1 Der Skript *WriteObjects*

Das erste Problem taucht sofort auf, wenn der File zum Schreiben geöffnet werden soll, da ein Ziel (Dateiname) komplett angegeben werden muss. Dazu gehören auch die Laufwerksbezeichnung und der Ort, wo das File abgelegt werden soll. Zur Erinnerung an **Kapitel 3.2** wurde auf den Maus-Add-on verwiesen. Der Exe-File **MausAddOn.EXE** befindet sich im Verzeichnis:

C:/TIA-Expert/MoveObjects/MoveObjects_Basic/Request

In diesem Verzeichnis können auch unter ***/Objects*** die Daten der Objekte gespeichert werden (**Bild 9.1**, **Punkt 1**).

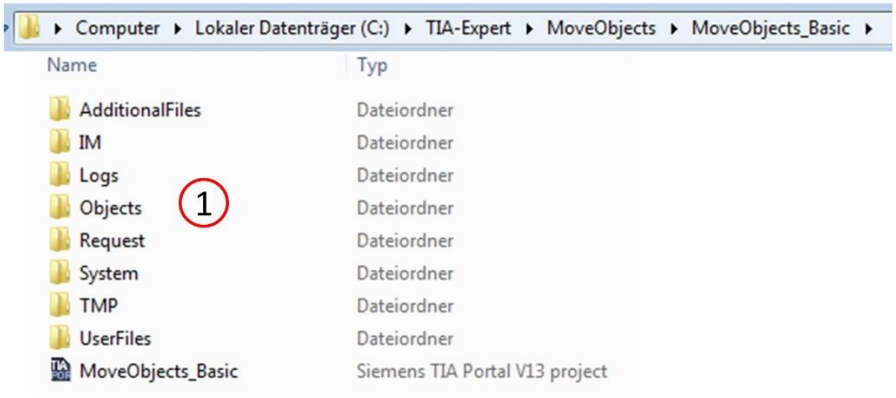


Bild 9.1 Schreiben und Lesen der Objekt-Daten in das Verzeichnis /Objects

Sie können auch ein anderes Verzeichnis wählen. Dazu müssen die hier gezeigten Skripte entsprechend nach Ihren Wünschen angepasst werden.

Das Verzeichnis muss immer komplett angegeben werden! Das Schreiben in das Hauptverzeichnis *C:/name der Datei* ist nicht so einfach möglich. Wählen Sie immer ein Unterverzeichnis, wie z.B. *C:/MeinFile/Objekte*.

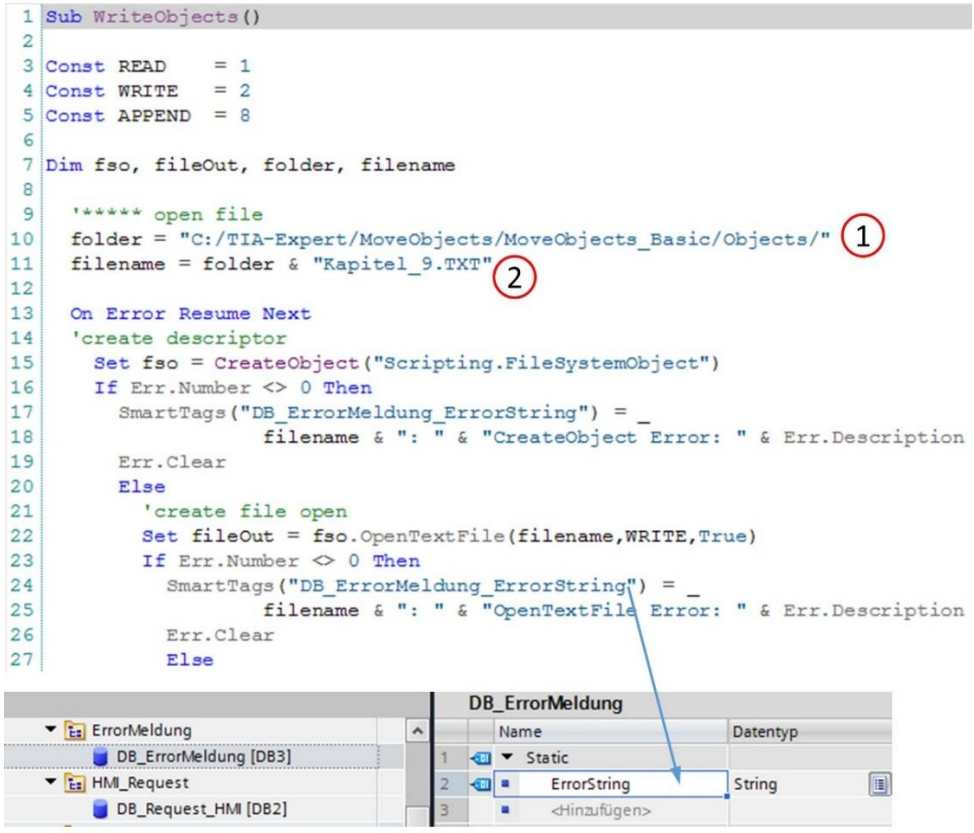


Bild 9.2 Der Skript WriteObjects Teil-1

Das Verzeichnis ist im Skript *WriteObjects* in Zeile 10 (**Punkt 1**) zu erkennen. Der Dateiname *Kapitel_9.TXT* in Zeile 11 (**Punkt 2**). Sollte es Probleme geben den File zu kreieren, wird eine Fehlermeldung (Zeilen 17+23) erzeugt, welche über den neu angelegten *DB_ErrorMeldung* (Bild unten) auf dem Monitor angezeigt werden kann.

In **Bild 9.3** ist das Speichern der Objekte zu sehen. Aus dem *DB_Line_Hor* und *DB_Valve_Hor* (Bild unten) werden die Positionen und *Visible* einzeln in das File geschrieben. So kann man sich gut vorstellen, dass bei vielen Objekten ein entsprechend großes Listing entsteht. Die Performance ist sehr hoch, sodass man sich darüber normal keine Gedanken machen muss. Allerdings muss man die Reihenfolge der Objekte mit dem Skript *ReadObjects* exakt einhalten sonst bekommen Objekte Koordinaten, welche nicht zum eigentlichen Objekt passen.

```
26            'Valve Hor
27              'index 0
28                fileOut.WriteLine(SmartTags("DB_Valve_Hor_Valve_Hor{0}_HMI_HMI_Visible"))
29                fileOut.WriteLine(SmartTags("DB_Valve_Hor_Valve_Hor{0}_HMI_HMI_XPos"))
30                fileOut.WriteLine(SmartTags("DB_Valve_Hor_Valve_Hor{0}_HMI_HMI_YPos"))
31                fileOut.WriteLine(SmartTags("DB_Valve_Hor_Valve_Hor{0}_HMI_HMI_XPos_CLICK"))
32                fileOut.WriteLine(SmartTags("DB_Valve_Hor_Valve_Hor{0}_HMI_HMI_YPos_CLICK"))
33              'index 1
34                fileOut.WriteLine(SmartTags("DB_Valve_Hor_Valve_Hor{1}_HMI_HMI_Visible"))
35                fileOut.WriteLine(SmartTags("DB_Valve_Hor_Valve_Hor{1}_HMI_HMI_XPos"))
36                fileOut.WriteLine(SmartTags("DB_Valve_Hor_Valve_Hor{1}_HMI_HMI_YPos"))
37                fileOut.WriteLine(SmartTags("DB_Valve_Hor_Valve_Hor{1}_HMI_HMI_XPos_CLICK"))
38                fileOut.WriteLine(SmartTags("DB_Valve_Hor_Valve_Hor{1}_HMI_HMI_YPos_CLICK"))
39              'index 2
40                fileOut.WriteLine(SmartTags("DB_Valve_Hor_Valve_Hor{2}_HMI_HMI_Visible"))
41                fileOut.WriteLine(SmartTags("DB_Valve_Hor_Valve_Hor{2}_HMI_HMI_XPos"))
42                fileOut.WriteLine(SmartTags("DB_Valve_Hor_Valve_Hor{2}_HMI_HMI_YPos"))
43                fileOut.WriteLine(SmartTags("DB_Valve_Hor_Valve_Hor{2}_HMI_HMI_XPos_CLICK"))
44                fileOut.WriteLine(SmartTags("DB_Valve_Hor_Valve_Hor{2}_HMI_HMI_YPos_CLICK"))
45            'Line_Hor
46              'index 0
47                fileOut.WriteLine(SmartTags("DB_Line_Hor_Line_Hor{0}_HMI_HMI_Visible"))
48                fileOut.WriteLine(SmartTags("DB_Line_Hor_Line_Hor{0}_HMI_HMI_XPos"))
49                fileOut.WriteLine(SmartTags("DB_Line_Hor_Line_Hor{0}_HMI_HMI_YPos"))
50                fileOut.WriteLine(SmartTags("DB_Line_Hor_Line_Hor{0}_HMI_HMI_XPos_CLICK"))
51                fileOut.WriteLine(SmartTags("DB_Line_Hor_Line_Hor{0}_HMI_HMI_YPos_CLICK"))
52
53              fileOut.Close()
54              SmartTags("DB_ErrorMeldung_ErrorString") = filename & ": File write ok"
55              ' close file
56              fso = Nothing
57            End If
58          End If
59
60  End Sub
```

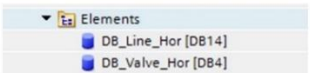


Bild 9.3 Der Skript *WriteObjects* Teil-2

9.2 Der Skript *ReadObjects*

Die Objekte werden wieder mit dem Skript *ReadObjects* gelesen(**Bild 9.4**). Im Listing ist ein Teil davon abgebildet. Der obere Teil entspricht im Wesentlichen dem Skript *WriteObjects*. In Zeile 21 wird der File geöffnet für das Lesen. Die Objekte werden in der gleichen Reihenfolge gelesen (Zeilen 29-52), wie diese im Skript *WriteObjects* geschrieben wurden.

```
20      'create file open
21      Set fileIn = fso.OpenTextFile(fileName, READ, True)
22      If Err.Number <> 0 Then
23        SmartTags("DB_ErrorMeldung_ErrorString") = fileName & ": " & "OpenTextFile Error: "
24        Err.Clear
25        Set fso = Nothing
26        Else
27          'Valve Hor
28              'index 0
29                SmartTags("DB_Valve_Hor_Valve_Hor{0}_HMI_HMI_Visible") = fileIn.ReadLine
30                SmartTags("DB_Valve_Hor_Valve_Hor{0}_HMI_HMI_XPos") = fileIn.ReadLine
31                SmartTags("DB_Valve_Hor_Valve_Hor{0}_HMI_HMI_YPos") = fileIn.ReadLine
32                SmartTags("DB_Valve_Hor_Valve_Hor{0}_HMI_HMI_XPos_CLICK") = fileIn.ReadLine
33                SmartTags("DB_Valve_Hor_Valve_Hor{0}_HMI_HMI_YPos_CLICK") = fileIn.ReadLine
34              'index 1
35                SmartTags("DB_Valve_Hor_Valve_Hor{1}_HMI_HMI_Visible") = fileIn.ReadLine
36                SmartTags("DB_Valve_Hor_Valve_Hor{1}_HMI_HMI_XPos") = fileIn.ReadLine
37                SmartTags("DB_Valve_Hor_Valve_Hor{1}_HMI_HMI_YPos") = fileIn.ReadLine
38                SmartTags("DB_Valve_Hor_Valve_Hor{1}_HMI_HMI_XPos_CLICK") = fileIn.ReadLine
39                SmartTags("DB_Valve_Hor_Valve_Hor{1}_HMI_HMI_YPos_CLICK") = fileIn.ReadLine
40              'index 2
41                SmartTags("DB_Valve_Hor_Valve_Hor{2}_HMI_HMI_Visible") = fileIn.ReadLine
42                SmartTags("DB_Valve_Hor_Valve_Hor{2}_HMI_HMI_XPos") = fileIn.ReadLine
43                SmartTags("DB_Valve_Hor_Valve_Hor{2}_HMI_HMI_YPos") = fileIn.ReadLine
44                SmartTags("DB_Valve_Hor_Valve_Hor{2}_HMI_HMI_XPos_CLICK") = fileIn.ReadLine
45                SmartTags("DB_Valve_Hor_Valve_Hor{2}_HMI_HMI_YPos_CLICK") = fileIn.ReadLine
46            'Line_Hor
47              'index 0
48                SmartTags("DB_Line_Hor_Line_Hor{0}_HMI_HMI_Visible") = fileIn.ReadLine
49                SmartTags("DB_Line_Hor_Line_Hor{0}_HMI_HMI_XPos") = fileIn.ReadLine
50                SmartTags("DB_Line_Hor_Line_Hor{0}_HMI_HMI_YPos") = fileIn.ReadLine
51                SmartTags("DB_Line_Hor_Line_Hor{0}_HMI_HMI_XPos_CLICK") = fileIn.ReadLine
52                SmartTags("DB_Line_Hor_Line_Hor{0}_HMI_HMI_YPos_CLICK") = fileIn.ReadLine
53
54              fileIn.Close()
55              SmartTags("DB_ErrorMeldung_ErrorString") = fileName & ": File read ok"
56              ' close file
57              fso = Nothing
58
59            End If
60        End If
61
62  End Sub
```

Bild 9.4 Der Skript *ReadObjects*

9.2.1 Der Test zu *Write-* und *ReadObjects*

In **Bild 9.5** ist die zu speichernde Situation der Objekte dargestellt (Phase 1). Im oberen Teil ist ein E/A-Feld platziert, damit das Ergebnis der Speicherung beobachtet werden kann. Im unteren Teil des Bildes werden die Objekte verschoben (Phase 2), aber nicht gespeichert.

Bild 9.5 Die Objekte werden gezeichnet und dann gespeichert

Die Textmeldung bleibt solange erhalten, bis der Skript *Write-* oder *ReadObjects* aufgerufen wird (Zeile 54 in *WriteObjects* und Zeile 55 in *ReadObjects*). In **Bild 9.6** wurde die Schaltfläche *Bild laden* gedrückt (Phase 3). Das Lesen erfolgte ohne Probleme, wie im Anzeigetext zu sehen ist. Die Objekte sind wieder an der Stelle, wie diese im **Bild 9.5** oben, zu sehen waren. Im unteren Teil in **Bild 9.6** sind die Ereignisse der beiden Schaltflächen zu sehen.

Die Objekte werden mit *ReadObjects* zwar gelesen, aber nicht neu positioniert. Deswegen muss mit der Schaltfläche *Bild laden* auch der Skript *InitObjects* (**Kapitel 4.1**) **nach dem Lesen** aufgerufen werden.

Anmerkung: Sollen mehrere Daten aus dem *DB* gespeichert werden, z.B. der Name zum Ventil, dann sind diese einfach hinzuzufügen. So kann auch, das mal nebenbei bemerkt, der Inhalt eines DBs auf die Festplatte verlagert werden.

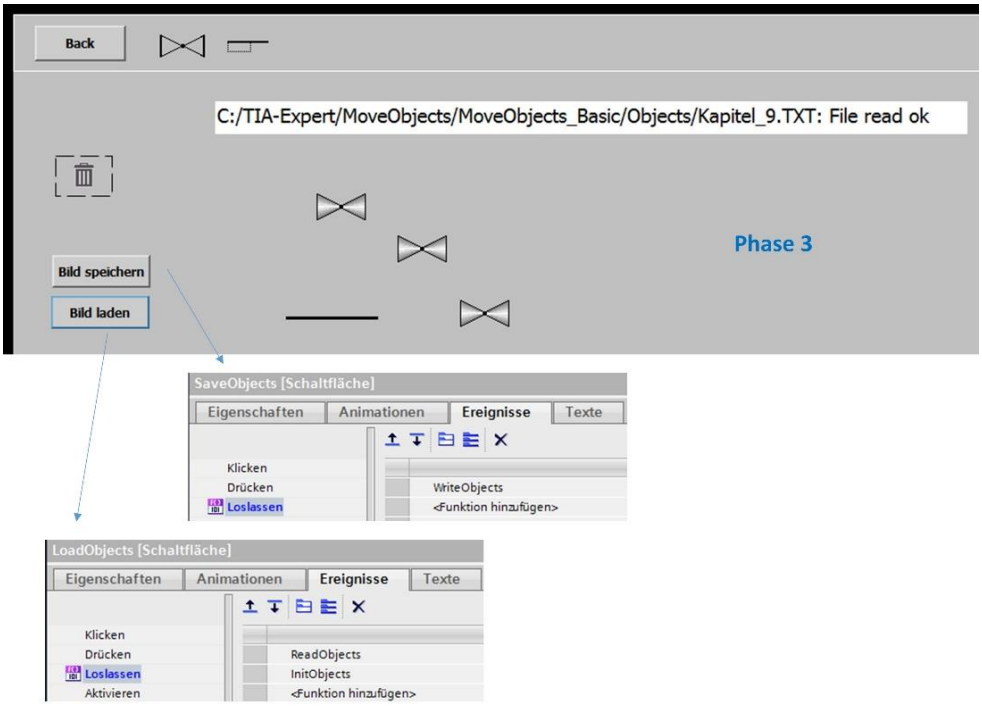


Bild 9.6 Die Objekte werden wieder geladen und durch *InitObjects* wieder richtig angezeigt

10 Einführung in Skripte schreiben mit WinCC Advanced (VBS)

VBS-Programmierer welche auf dem Betriebssystem Windows Programme schreiben und anwenden, müssen sich beim VBS in WinCC auf einige Unterschiede und fehlende Funktionen einstellen. Zum Beispiel beim Zugriff auf das Dateisystem. Der Grund liegt darin, dass das Runtime-System unterschiedlich ist. Es gibt Panels mit den Runtime-System Windows CE und PC-Systeme mit dem PC Runtime System WinCC. Beide entsprechen nicht dem eventuell gewohnten VBS-Programmierer auf der PC-Welt und den üblichen Betriebssystemen, wie z.B. Windows 7 oder Windows 10.

Grundsätzlich muss der Anwender darüber nachdenken, warum er Skripte einsetzt und wie er das tut. Die Anwendung von Skripten kann über die Funktionsliste erfolgen und das Tippen eigener Quellen entfällt. Aufgerufen werden alle von WinCC gelieferten Funktionen. Die Funktionsliste (**Bild** 10.1) bezieht sich immer auf ein Objekt. Hier z. B. erfolgte ein Klick auf die Bildoberfläche und es werden unter *Ereignisse* nun die Systemfunktionen zum Bildaufbau angeboten.

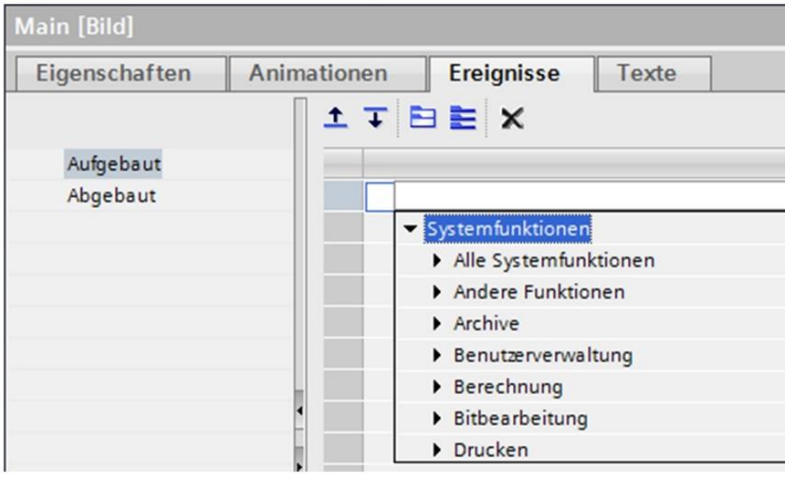


Bild 10.1 Funktionsliste bietet Systemfunktionen an

Die Systemfunktionen können auch in eigenen, benutzerdefinierten Skripten aufgerufen werden. Der Unterschied besteht darin, dass z. B. bei einem Bildwechsel automatisch der Skript aufgerufen wird, der in der Funktionsliste ausgewählt wurde. Das können auch benutzerdefinierte Skripte sein. Befindet sich der benutzerdefinierter Skript nicht in der Funktionsliste, dann wird er durch einen Event aus WinCC aufgerufen. Den Event aber, muss der Benutzer anregen.

Auch sollte überlegt werden, ob der Skript durch eine Anwendung in der SPS ersetzt werden sollte, z. B. durch eine SCL-Funktion. Hat der Skript eine Menge Variablen aus der SPS zu sammeln und werden diese nicht so direkt für die Manipulation des Bildes oder einer Systemschnittstelle benutzt, dann ist eventuell eine SCL-Funktion in der SPS und kein Skript auf der Bedienerebene vorzuziehen.

10.1 Benutzerdefinierte Funktionen schreiben

Die Anwendung der Funktionsliste ist dem TIA-Programmierer normalerweise bekannt. Sei es z. B. durch den Bildwechsel oder die Anwendung einer Schaltfläche mit dem Event *Drücken* oder *Loslassen*. Im Hilfesystem können Information über die Systemfunktionen nachgelesen werden. Die Funktionsliste ist aus diesem Grund **nicht** Thema dieser Einführung, sondern die benutzerdefinierten Funktionen, sprich proprietäre Skript-Programmierung.

Eigene Skripte können nur geschrieben werden, wenn dies das verwendete Bediengerät auch zulässt.

Für den Einstieg eigener Skripte werden wir für den Aufruf unserer Skripte vorerst die Funktionsliste benutzen. Anschließend beschäftigen wir uns mit dem Skript-Aufruf über den Aufgabenplaner und zu guter Letzt den indirekten Aufruf über die SPS. Die Syntax für Skripte soll sich überwiegend auf die gezeigten, praktischen Anwendungen begrenzen.

Ein Grundkurs über VBS sollte in jedem Fall für das Verständnis der folgenden Zeilen vorhanden sein.

10.2 Der erste Skript

Lernschritte:

- Erstellen eines Skriptes ohne Parameter
- Aufruf des Skriptes über die Funktionsliste
- Bild-Objekte im Skript deklarieren
- Bild-Objekte ändern

In (**Bild** 10.2) wird der Skript neu erstellt (**Punkt 1**). Über *Skripte/VB-Skripte/Neue VB-Funktion hinzufügen* wird der neu eingefügte Skript *VBFunktion_1* in *ErsterSkript* umbenannt (**Punkt 2**). Das Ergebnis ist in **Punkt 3** ersichtlich. Ein leerer Skript bietet sich an. An dieser Stelle möchte ich auf **Kapitel 2.1** verweisen. Dort werden die Hilfsmöglichkeiten für die ersten Eingaben in einen Skript vorgestellt. Auch wie die Help-Anzeige als Kommentar ausgeschaltet wird (**Bild 2.3**), ist dort zu finden.

Der Skript-Name ***Sub ErsterSkript()*** (Zeile 1) zeigt uns, dass es sich hier um eine Subroutine handelt. Diese haben keine Rückgabewerte. Der Typ des Skriptes kann auch als Funktion

angewählt werden (**Punkt 4**). Funktionen können an den Aufrufer einen Rückgabewert zurückgeben. Durch die Klammern **()** ohne Inhalt, ist im **Bild** 10.2 (Zeile 1) auch zu erkennen, dass die Subroutine keine Parameter besitzt. Das Thema Rückgabewert und Parameterübergabe wird in **Kapitel 10.5** vorgestellt.

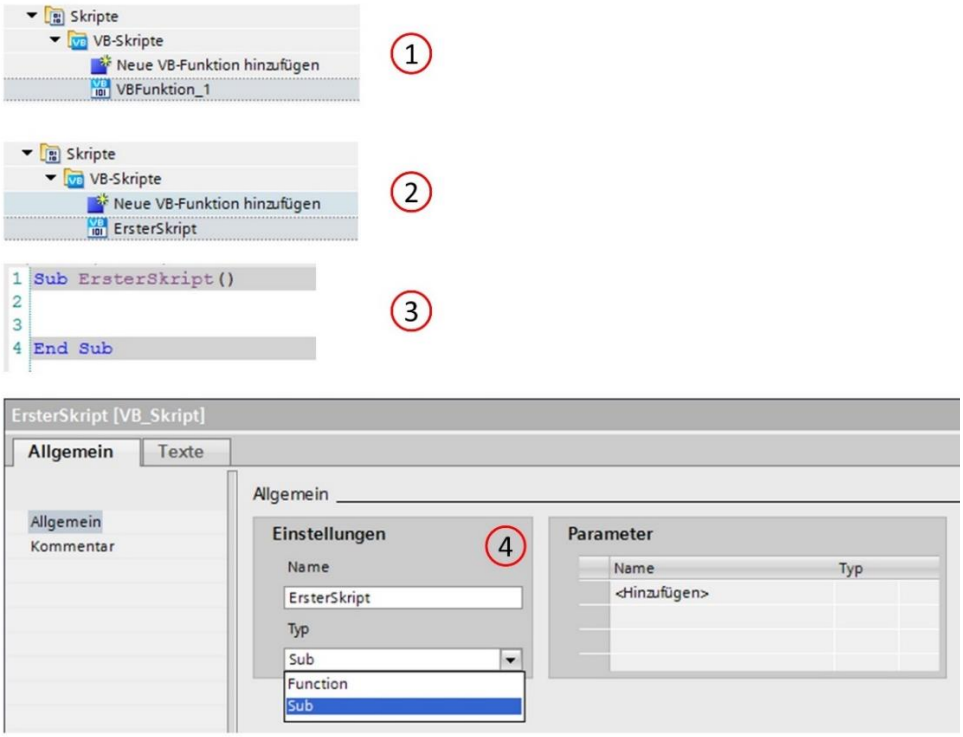


Bild 10.2 Der erste Skript wird erstellt

10.2.1 Skript aufrufen

Zur ersten Anwendung des Skriptes in der HMI nutzen wir die Funktionsleiste für den Aufruf des Skriptes. Damit mit dem Ereignis *Loslassen* der *Skript* aufgerufen wird, muss der Skriptname in die Liste eingetragen werden.

Im Beispiel zur Erklärung werden wir die Hintergrundfarbe der beiden Objekte Kreis und Viereck ändern (**Bild 10.3**). Eine triviale Angelegenheit welche für die erste Anwendung aber ausreichend ist. Wie im **Bild 10.3** zu sehen (Zeile 4), wie der *BackColor* zum Rechteck_1 mit dem Punkt-Operator durch das Auswahlfenster selektiert (**Kapitel 2.1**). Das fertige Listing für beide Bild-Objekte ist im **Bild** 10.4 zu sehen. Der Button (**Punkt 1**) dient der Syntax-Überprüfung der Skriptzeilen. Das Ergebnis ist in **Punkt 2** zu sehen**.**

Es handelt sich hier nur um eine Syntax-Überprüfung. Der Skript kann auch ohne Syntax-Fehler **nicht** funktionieren, wie z. B. bei logischen Fehlern oder durch einen Skriptabbruch nach einem unerlaubten Zugriff.

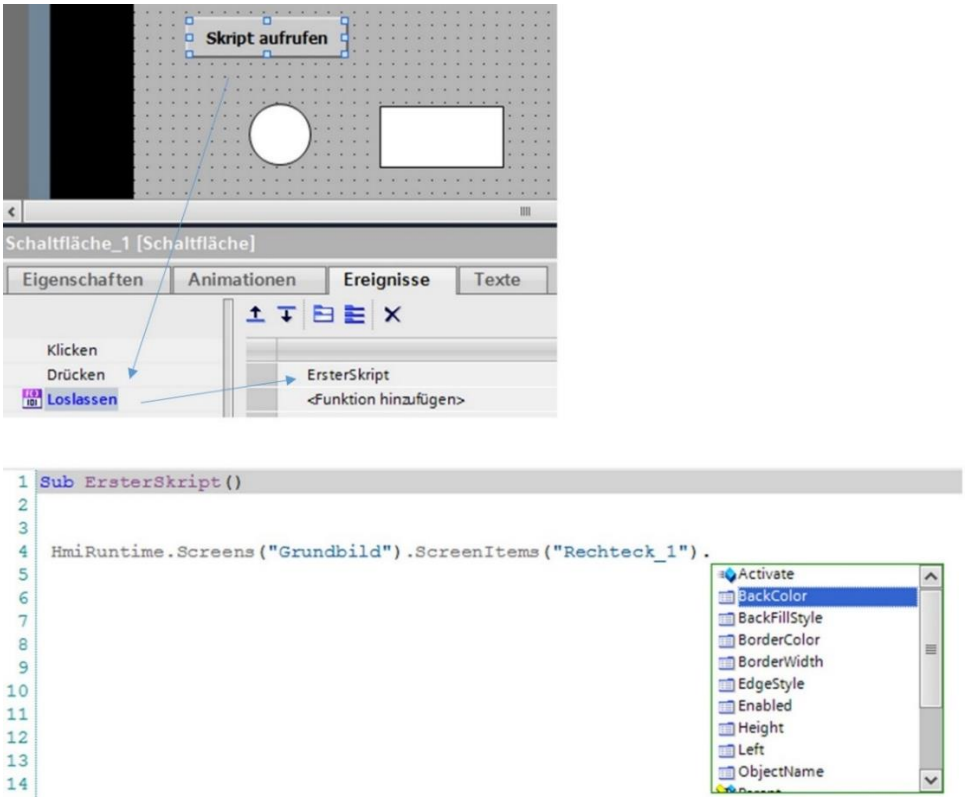


Bild 10.3 Der Skript

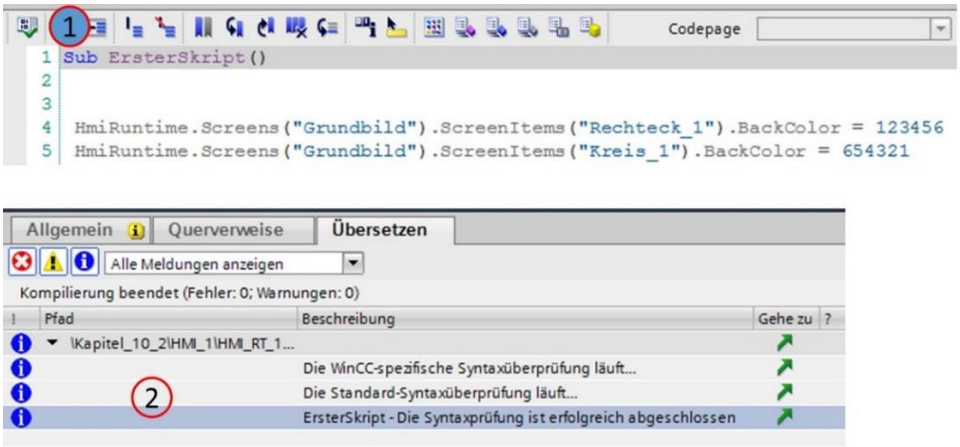


Bild 10.4 Der Skript wird auf Syntax-Fehler überprüft

Die Zuweisungen der Zahlen zur Hintergrundfarben in den Zeilen 4+5 sind willkürlich. Im nachfolgendem (**Bild 10.5**) ist das Ergebnis unseres kleinen Einstieges zu sehen.

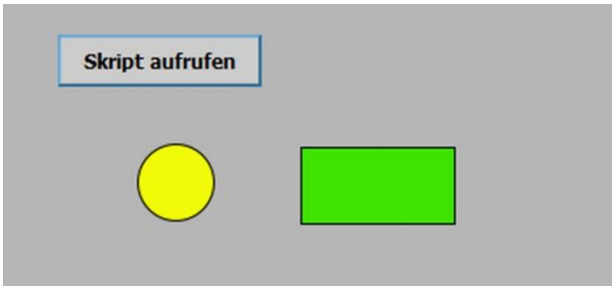


Bild 10.5 Das erste Ergebnis

10.3 Skripte mit internen HMI-Variablen

Lernschritte:

- HMI-interne Variablen erstellen
- Kommentare
- Einfache Operatoren, wie der Zuweisungsoperator und arithmetische Operatoren

Das TIA-Projekt zu diesem Kapitel ist aufbauend zum **Kapitel 10.2**. Wir möchten nun nicht nur die Hintergrundfarbe der Bild-Objekte verändern, sondern auch nach jedem Klick auf die Schaltfläche dessen Positionen vertauschen. Betrachten wir den direkten Zugriff auf Bild-Objekte wie z.B.in **Bild** 10.4 bereits gezeigt, dann benötigen wir für das Speichern der Positionen eines Bildobjektes zwei Merker. Dazu betrachten wir das erweiterte Listing in (**Bild** 10.6).

Die im Listing gekennzeichneten grünen Texte (Zeilen 3, 6, 9, 12) beginnen mit einem Hochkomma und kennzeichnen eine Kommentar-Zeile.

Kommentare werden mit dem Hochkomma ‘ eingeleitet. Die Zeile ist dann für den Übersetzter (Compiler) unsichtbar.

In den HMI-Variablen wurden zwei Merker deklariert, welche mit *Strg+j* editiert werden können (**Punkt 3**). Auf diesem Weg lassen sich am einfachsten *SmartTags* einfügen; z.B. in Zeile 7 mit *SmartTags(“Merker_1“)*.

SmartTags können für HMI-Variablen benutzt werden, müssen aber nicht. Es kann auch die Bezeichnung der HMI-Variablen verwendet werden. Dies unterliegt aber der VBS-Namenskonventionen und bedeutet, dass bei einem nicht akzeptierten Namen der Skript abbricht. Es wird empfohlen immer die Referenzierung (*SmartTags*) anzuwenden.

Zeilen 7+8:

SmartTags(“Merker_1“) = HmiRuntime.Screens("Grundbild").ScreenItems("Kreis_1").Left
SmartTags(“Merker_2“) = HmiRuntime.Screens("Grundbild").ScreenItems("Kreis_1").Top

Mit dem Zuweisungsoperator = wird der rechte Teil der Anweisung in den linken Teil einer Variablen kopiert.

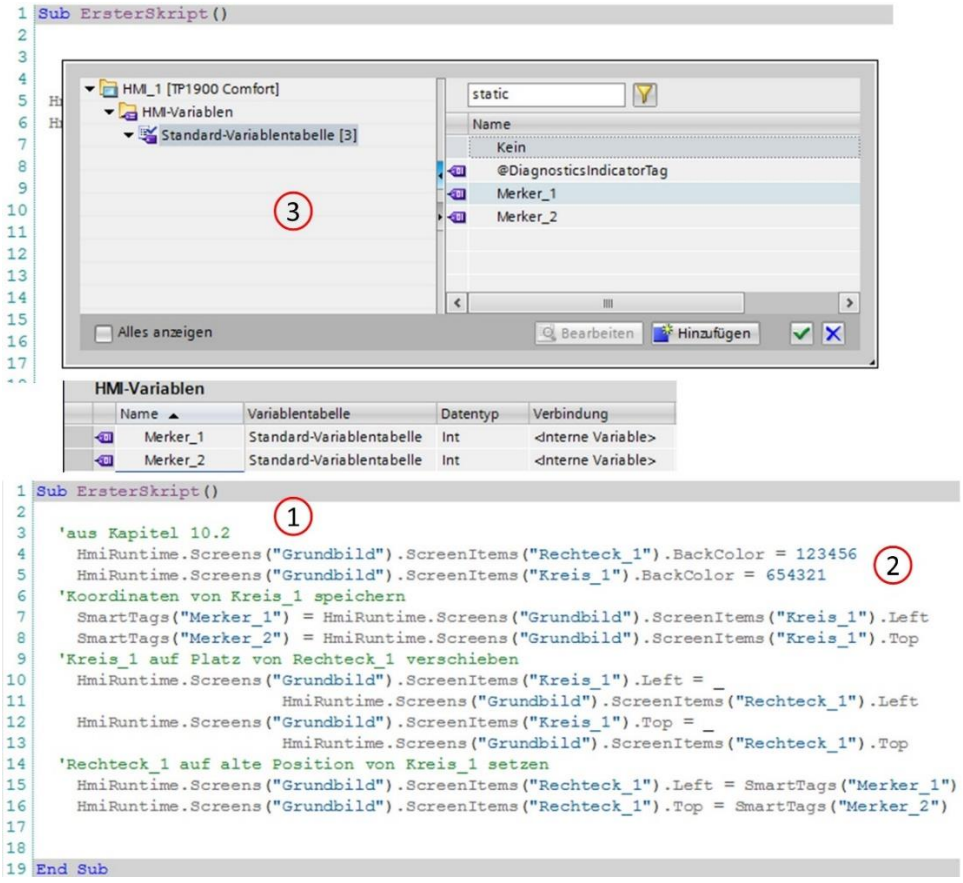


Bild 10.6 Das erweiterte Listing

Die Position des Kreises ist nun gespeichert, sodass dieser auf die Position des Rechteckes verschoben werden kann.

Zeilen 10+11:

HmiRuntime.Screens("Grundbild").ScreenItems("Kreis_1").Left =
*HmiRuntime.Screens("Grundbild").ScreenItems("Rechteck_1").**Left***
HmiRuntime.Screens("Grundbild").ScreenItems("Kreis_1").Top =
*HmiRuntime.Screens("Grundbild").ScreenItems("Rechteck_1").**Top***

Die Position *Left* entspricht der X-Koordinate und die Position *Top* der Y-Koordinate, welche mit dem Punktoperator vom Bild-Objekt selektiert werden.

Zeilen 13+14:

*HmiRuntime.Screens("Grundbild").ScreenItems("Rechteck_1").**Left** =*
SmartTags("Merker_1")
*HmiRuntime.Screens("Grundbild").ScreenItems("Rechteck_1").**Top** =*
SmartTags("Merker_2")

Dann wird das Rechteck auf die gespeicherten Koordinaten des Kreises verschoben. Das Ergebnis ist in (**Bild 10.7**) zu sehen.

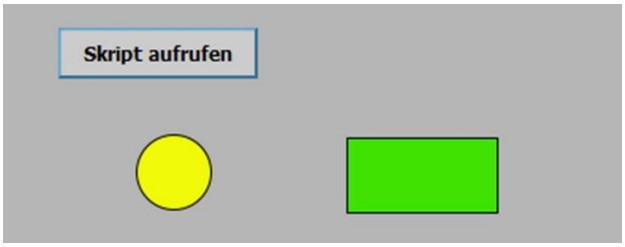


Bild 10.7 Die Bildobjekte sind vertauscht

10.3.1 Operatoren anwenden

Weitere Operatoren sind z.B. arithmetische Operatoren wie **+,-**. Diese Operatoren sind normalerweise einfach anzuwenden (Schulmathematik). So könnten mit den SmartTags z.B. folgende Zeilen geschrieben werden:

*SmartTags("Merker_1") = SmartTags("Merker_1") * 2*
SmartTags("Merker_2") = SmartTags("Merker_1") + SmartTags("Merker_2")

Solche Operatoren sind in VBS problemlos anzuwenden, da man sich über die Datentypen keine besonderen Gedanken machen muss und die Operatoren den üblichen Regeln wie Punktrechnen geht vor Strichrechnen funktionieren. Mit dem Klammer-Operator können Prioritäten gesetzt werden, wie z.B.:

*SmartTags("Merker_1") = (4+5) * 10*

Im Allgemeinen gelten hier in VBS die Regeln, welche auch in anderen Programmiersprachen gelten. Aus diesem Grunde sollen an dieser Stelle auch keine weiteren Beispiele folgen. Die Anwendung logischer Operatoren dagegen ist schon etwas komplizierter. Allerdings steht die Anwendung solcher Operatoren im Skript oft in Konkurrenz zu den logischen Operatoren in der SPS. Aus diesem Grunde sind in den Skripten zum Thema Objekte bewegen selten logischen Operatoren zu finden. Ein Beispiel für die typische Anwendung ist der logische Operator *NOT*, welcher den logischen Zustand *True* oder *False* negiert.

SmartTags("Merker_1") = True
SmartTags("Merker_2") = NOT SmartTags("Merker_1")

Diese Eingaben sind problemlos möglich, obwohl im Skript *SmartTags("Merker_1")* zuvor eine Koordinate gespeichert hatte. Die Wandlung vom Datentyp Integer nach Boolean wird hier vom Compiler automatisch zugeordnet. Ebenfalls sind Vergleichsoperatoren *(>, <, >=. <=, =, <>*) öfters auch in Skripten zu finden, damit z.B. Abgrenzungen bezüglich der Bildkoordinaten überprüft werden können. Auch hier ist zu empfehlen die Informationen im Hilfesystem zu Nutze zu ziehen.

Die im Beispiel verwendeten internen HMI-Variablen sind flüchtige Variablen. Fällt die Betriebs-Spannung an der HMI weg, gehen alle Werte der internen HMI-Variablen verloren. Variablen aus der SPS dagegen können remanent sein und über eine Verbindung kann die HMI auf diese Variablen lesen oder schreiben. Im folgenden Kapitel fordert eine SPS eine Zufallszahl in einem definierten Zahlenbereich.

10.4 Grundkonzepte für Kontrollanweisungen

Lernschritte:

- PLC-Variablen im Skript anwenden
- Einfache Kontroll-Strukturen
- Select Case Konstruktion
- Subroutinen vorzeitig verlassen
-

Im TIA-Projekt zu diesem Kapitel soll untersucht werden, wie die Variablen aus der SPS in der HMI grundsätzlich verwendet werden können. Dazu benötigen wir eine SPS mit einer Verbindung zur HMI und einen Speicherbereich in der SPS.

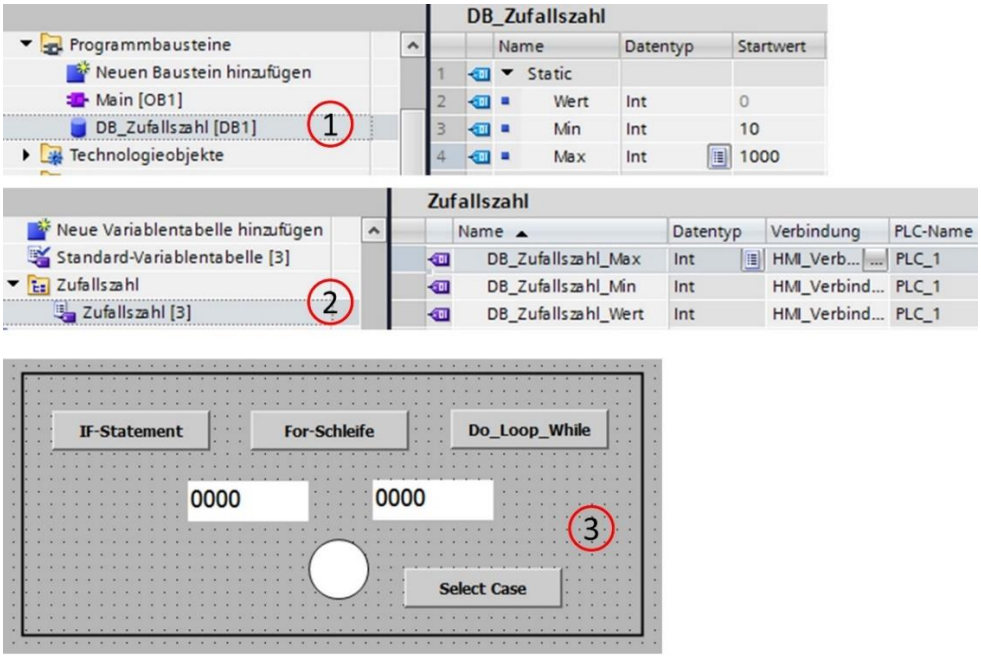


Bild 10.8 DB_Zufallszahl

In **Bild 10.8** ist der *DB_Zufallszahl* angelegt (**Punkt 1**) und in den HMI-Variablen eingetragen (**Punkt 2**). Eine Zufallszahl zu erzeugen, ist in VBS einfach und schnell programmiert. Bei der Gelegenheit sollen drei Möglichkeiten (**Punkt 3**) der Anwendung eines Zufallsgenerators untersucht werden.

10.4.1 *IF*-Statement

Mit der Schaltfläche *IF-Statement* wird die Subroutine *IF_Statement()* aufgerufen (**Bild** 10.9).

```
1  Sub IF_Statement()
2
3  'Zufallszahl ermitteln und in SPS schreiben
4    SmartTags("DB_Zufallszahl_Wert") = Rnd * (SmartTags("DB_Zufallszahl_Max") _
5                  - SmartTags("DB_Zufallszahl_Min"))+SmartTags("DB_Zufallszahl_Min")
6  'Zufallszahl in interne HMI-Variable
7    SmartTags("Merker_1") = Rnd * (SmartTags("DB_Zufallszahl_Max") _
8                  - SmartTags("DB_Zufallszahl_Min"))+SmartTags("DB_Zufallszahl_Min")
9
10 'Farbe zuordnen
11   'Zuordnung PLC>HMI (Blau)
12     If(SmartTags("DB_Zufallszahl_Wert") > SmartTags("Merker_1"))Then
13       HmiRuntime.Screens("Grundbild").ScreenItems("PLC_HMI_Anzeige_1")._
14         BackColor = RGB(0,0,255)
15     End If
16
17   'Zuordnung PLC<HMI (Grün)
18     If(SmartTags("DB_Zufallszahl_Wert") < SmartTags("Merker_1"))Then
19       HmiRuntime.Screens("Grundbild").ScreenItems("PLC_HMI_Anzeige_1")._
20         BackColor = RGB(0,255,0)
21     End If
22
23   'Zuordnung PLC=HMI (Rot)
24     If(SmartTags("DB_Zufallszahl_Wert") = SmartTags("Merker_1"))Then
25       HmiRuntime.Screens("Grundbild").ScreenItems("PLC_HMI_Anzeige_1")._
26         BackColor = RGB(255,0,0)
27     End If
28 'end Farbe zuordnen
29
30 End Sub
```

Bild 10.9 Der Skript *IF_Statement*

Die Grundsyntax heißt *IF-Then-Else*. In unserem Beispiel fällt der Else-Teile weg, da er nicht benötigt wird. Die Zufallszahl wird in Zeilen 4+6 mit der Funktion ***Rnd*** aufgerufen. Die Zahl wird durch die Multiplikation auf die Werte in der SPS begrenzt (**Bild** 10.8, Startwert Min/Max) und liefert als Rückgabewert eine Zufallszahl von 10-1000.

Rnd ist eine System-Funktion mit Rückgabewert und liefert eine Real-Zahl zwischen 0.0 – 1,0, welche durch die Zuweisung in den *DB* nach Integer gerundet wird.

Einmal wird eine Zufallszahl in die SPS-Variable und einmal in die interne HMI-Variable gespeichert (Zeilen 4+6). Danach folgt die Auswertung mit dem *IF-Statement*, ob die Zufallszahl der SPS größer, kleiner oder gleich der Zufallszahl der HMI-Variable ist. Entsprechend wird dann die Hintergrundfarbe des Kreises gesetzt. Dieser wird nach jedem Tastendruck Grün, Blau oder Rot sein. Dass dabei der Vergleich auf Gleichheit fällt (**=**) ist wohl selten der Fall. Ob das überhaupt möglich ist, kann mit einer Schleife getestet werden.

10.4.2 Schleifen

Mit der Schaltfläche *For-Schleife* wird der Skript *For_Schleife()* gestartet. Hier fällt auf, dass die sogenannte Laufvariable kein SmartTags ist, sondern eine interne, lokale Variable (***Dim i***). Diese wird durch die *For-Anweisung* von 0-200 gezählt (Zeile 6 + 19). Innerhalb der For-Anweisung werden die Zufallszahlen ermittelt. Das *IF-Statement* in Zeile 13 prüft auf

Gleichheit der beiden Variablen und wird dann beim Ergebnis *True* die Hintergrundfarbe auf Rot setzen und die Subroutine mit ***Exit Sub*** (Zeile 17) verlassen.

In Schleifenkönnen nur lokale Variablen als Laufvariable verwendet werden

```
1  Sub For_Schleife()
2
3  'Variable für die Schleife deklarieren
4  Dim i
5
6  For i = 0 To 200
7    'Zufallszahl ermitteln und in SPS schreiben
8      SmartTags("DB_Zufallszahl_Wert") = Rnd * (SmartTags("DB_Zufallszahl_Max")_
9          -SmartTags("DB_Zufallszahl_Min"))+SmartTags("DB_Zufallszahl_Min")
10   'Zufallszahl in interne HMI-Variable
11     SmartTags("Merker_1") = Rnd * (SmartTags("DB_Zufallszahl_Max")_
12         -SmartTags("DB_Zufallszahl_Min"))+SmartTags("DB_Zufallszahl_Min")
13   If(SmartTags("DB_Zufallszahl_Wert") = SmartTags("Merker_1"))Then
14     'Zuordnung PLC=HMI (Rot)
15     HmiRuntime.Screens("Grundbild").ScreenItems("PLC_HMI_Anzeige_1")._
16       BackColor = RGB(255,0,0)
17     Exit Sub
18   End If
19 Next
20
21  HmiRuntime.Screens("Grundbild").ScreenItems("PLC_HMI_Anzeige_1")._
22   BackColor = RGB(255,255,255)
23
24 End Sub
```

Bild 10.10 Der Skript *For_Schleife*

Die *For-Anweisung* wird im schlechtesten Falle 201 Schleifen durchlaufen (0-200). Schleifen in einem Skript sind dann kritisch, wenn diese eine lange Laufzeit benötigen. Normalerweise haben diese in einem WinCC-Programm nichts zu suchen.

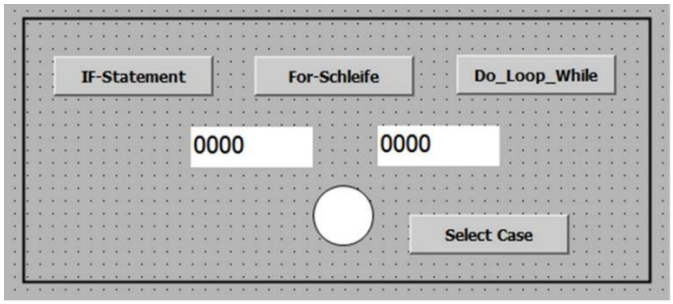
```
1  Sub Do_Loop_While_Statement()
2
3  'Koennte lange dauern
4  Do
5    'Zufallszahl ermitteln und in SPS schreiben
6      SmartTags("DB_Zufallszahl_Wert") = Rnd * (SmartTags("DB_Zufallszahl_Max")_
7        -SmartTags("DB_Zufallszahl_Min"))+SmartTags("DB_Zufallszahl_Min")
8    'Zufallszahl in interne HMI-Variable
9      SmartTags("Merker_1") = Rnd*(SmartTags("DB_Zufallszahl_Max")_
10       -SmartTags("DB_Zufallszahl_Min"))+SmartTags("DB_Zufallszahl_Min")
11   If(SmartTags("DB_Zufallszahl_Wert") = SmartTags("Merker_1"))Then
12     'Zuordnung PLC=HMI (Rot)
13       HmiRuntime.Screens("Grundbild").ScreenItems("PLC_HMI_Anzeige_1")._
14         BackColor = RGB(255,0,0)
15       Exit Sub
16   End If
17 Loop While(True)
18
19 End Sub
```

Bild 10.11 Der Skript Do_Loop_While_Statement()

In **Bild 10.11** ist die *Do-Loop-While*-Schleife zu sehen. Die Schleife wird solange laufen, bis die *While*-Bedingung ein *True* liefert. Hier (Zeile 17) handelt es sich um eine Endlos-Schleife, falls diese nicht durch ein Exit Sub (Zeile 15) verlassen wird. Wie lange das nun dauert hängt vom Zufallsgenerator ab, bis beide Variablen die gleiche Farbe haben. Mit der zuvor dargestellten *For-Schleife* ist bei einer Schleife mit 201 Wiederholungen schon ab und zu eine Gleichheit entstanden. Bei einer Schleife von 0-1000 kommt das schon wesentlich öfters vor. Also ist die hier gezeigte Endlos-Schleife nicht ganz so kritisch. Aber auch diese sollte in einem WinCC-Programm ohne Laufzeitabsicherung nicht verwendet werden.

10.4.3 Select Case

Die Auswahl-Entscheidung ist in den Skripten meistens anzutreffen. Immer wenn mehrere Entscheidungen zu fällen sind, sollten keine verschachtelten *IF-Anweisungen* verwendet werden. Das führt nur zu sogenannten *IF*-Gräbern und macht die Programme unübersichtlich. Bei der *Select Case*-Anweisung wird immer nur der *Case-Fall* oder der *Else-Teil* abgearbeitet. Das erhöht die Arbeitsgeschwindigkeit enorm und ergibt eine klare Übersicht, welches Zustand dieser *Select-Case* gerade hat. In unserem Beispiel werden drei Zufallszahlen selektiert, welche den Kreis farbig kennzeichnen, wenn eine davon zugeschlagen hat. Sonst bleibt der Kreis weiß. In **Bild 10.12** ist zu sehen, dass im TIA Portal die Anweisungen als Code-Vorlage in den Editor gezogen werden können. Das erleichtert gegebenenfalls die Syntax-Suche, besonders dann, wenn man schon lange nicht mehr in VBS gearbeitet hat.




```
'Please replace all sequences which are enclosed
Select Case _testexpression_
  Case _expressionlist_
    'statementblock_1
  Case _expressionlist_
    'statementblock_2
  Case Else
    'statementblock_n
End Select
```

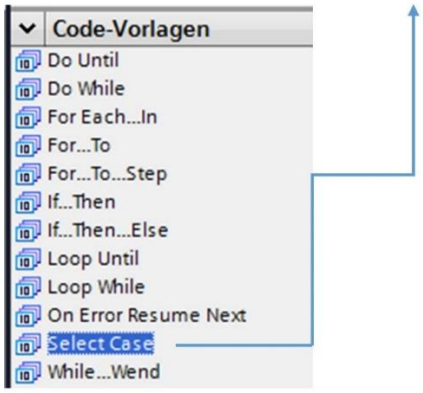


Bild 10.12 Code-Vorlagen helfen bei der Eingabe

In **Bild** 10.13 ist das *Select Case*–Statement zu sehen. Hier wurde die Zufallszahl auf 10-20 gelegt, damit beim Testen auch in absehbarer Zeit ein Ergebnis zur Verfügung steht. Auf die Zuweisung der internen HMI-Variable wurde ebenfalls verzichtet.

```
1  Sub Select_Case()
2
3  'Zufallszahl ermitteln und in SPS schreiben
4    'random von 0-20
5    SmartTags("DB_Zufallszahl_Wert") = (Rnd * 10) + SmartTags("DB_Zufallszahl_Min")
6  Select Case SmartTags("DB_Zufallszahl_Wert")
7    Case  17
8      HmiRuntime.Screens("Grundbild").ScreenItems("PLC_HMI_Anzeige_1")._
9      BackColor = RGB(0,255,0)
10   Case  14
11     HmiRuntime.Screens("Grundbild").ScreenItems("PLC_HMI_Anzeige_1")._
12     BackColor = RGB(255,0,0)
13   Case  20
14     HmiRuntime.Screens("Grundbild").ScreenItems("PLC_HMI_Anzeige_1")._
15     BackColor = RGB(0,0,255)
16   Case Else
17     HmiRuntime.Screens("Grundbild").ScreenItems("PLC_HMI_Anzeige_1")._
18     BackColor = RGB(255,255,255)
19 End Select
20
21 End Sub
```

Bild 10.13 Der Skript *Select_Case()*

10.5 *Function* oder *Subroutine*

Lernschritte:

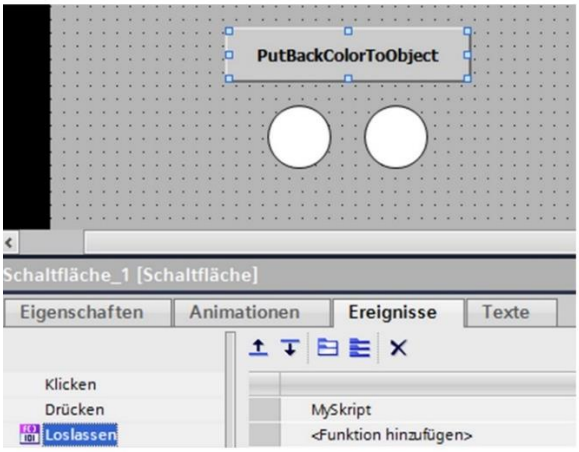
- Parameter-Übergabe
- Objekte mit Set verbinden
- Funktionsaufruf mit Rückgabewert

Die bis zu diesem Kapitel angewendeten Skripte waren Subroutinen ohne Parameter. Subroutinen und Funktionen können mit bis zu maximal 8 Parametern aufgerufen werden. Die Parameter werden unter *Eigenschaften/Parameter* eingetragen. Diese können als Kopie (Call by Value) oder als Referenz (Call by Reference) übergeben werden. Das TIA-Projekt zu diesem Kapitel zeigt das Beispiel der Hintergrundfarben für den Kreis über einen Zufallsgenerator, welcher einmal als Subroutine und einmal als Funktion angewendet wird.

Der Skript *MySkript()* wird über eine Schaltfläche aus der Funktionsliste aufgerufen. Dieser ruft die Skript-Funktion *PutBackColorToObject_Func* und die Skript-Subroutine *PutBackColorToObject_Sub* auf. Beide erzeugen eine Farbe innerhalb eines Farbbereiches, welcher durch die SPS vorgeben wird.

In **Bild 10.14** ist der Skript *MySkript()* zu sehen. Die Dim-Anweisung deklariert eine interne Variable, hier *objec*t (Zeile 3). Diese wird durch den *Set*-Befehl in Zeile 5 mit dem Bild-Objekt des ersten Kreises verbunden. Nun kann die Funktion *PutBackColorToObject_Func* aufgerufen werden (Zeile 6). Diese liefert als Funktion einen Rückgabewert, welcher o*bject.BackColor* zugewiesen wird. Die Parameter zur Funktion werden über die SmartTags aus der SPS übergeben.

Der Aufruf der Funktion hat für die Parameter eine Klammerung **()**. Das hat eine Subfunktion **nicht**. Der Rückgabewert der Funktion **muss** zugewiesen werden. Sonst betrachtet der Übersetzer die Funktion als Subroutine und meckert über die gesetzten Klammern.




```
1  Sub MySkript()
2
3    Dim object
4
5    Set object =  HmiRuntime.Screens("Grundbild").ScreenItems("PLC_HMI_Anzeige_1")
6    object.BackColor = PutBackColorToObject_Func(SmartTags("DB_Zufallszahl_Min"),_
7      SmartTags("DB_Zufallszahl_Max"))
8    Set object =  HmiRuntime.Screens("Grundbild").ScreenItems("PLC_HMI_Anzeige_2")
9    PutBackColorToObject_Sub SmartTags("DB_Zufallszahl_Min"),_
10     SmartTags("DB_Zufallszahl_Max"),object
11
12 End Sub
```

Bild 10.14 Der Skript MySkript

In Zeile 8 wird die Subroutine *PutBackColorToObject_Sub* aufgerufen. Die Parameter werden nicht in Klammer () gesetzt. Damit die Subroutine auch ein Objekt zur Hintergrundfarbe besitzt, wird diese als dritter Parameter übergeben (*object)*.

Die Parameter einer Subroutine werden nicht in Klammer () eingebunden. Eine Subroutine hat keinen Rückgabewert.

```
Function PutBackColorToObject_Func(ByRef Min, ByRef Max)

  PutBackColorToObject_Func = (Rnd * (Max-Min)) + Min
  Exit Function

End Function
```

```
Sub PutBackColorToObject_Sub(ByRef Min, ByRef Max, ByRef BildObject)

  BildObject.BackColor = (Rnd * (Max-Min)) + Min
  Exit Sub

End Sub
```

Bild 10.15 Die Skripte zur Funktion und zur Subroutine

Die Skripte unterscheiden sich in der Namensgebung *Funktion* und *Sub* (**Bild** 10.15). Die Parameter sind nur funktionell zu betrachten und unterscheiden sich in der Syntax nicht. Das Ende einer *Function* wird beispielhaft mit *Exit Function* und das der Subroutine mit *Exit Sub* eingeleitet.

10.6 Skripte automatisch aufrufen

Lernschritte:

- Anwendung des zyklischen Events

Mit dem Zufallsgenerator wurden in **Kapitel 10.5** der Skript *MySkript()* über die Funktionsliste durch ein Ereignis der Schaltfläche aufgerufen. Dieser Skript ruft dann zwei weitere Skripte auf.

Skripte in der Funktionsliste werden der Reihe nach von oben nach unten aufgerufen. In den benutzerspezifischen Skripten werden die Skripte entsprechend der Reihenfolge des Aufrufes abgearbeitet. Insgesamt kann immer nur ein benutzerspezifischer Skript aktiv sein (FiFo).

Wird innerhalb eines Skriptes eine Systemfunktion aufgerufen, so kann diese u. U. im Hintergrund abgearbeitet werden (File-Zugriff). Auch kann ein Skript, automatisch durch eine Wertänderung in die Warteschleife (FiFo) eingereiht werden. Die Synchronisation der benutzerspezifischen Skripte könnte dann in beiden Fällen nicht mehr gewährleistet sein und es kommt bei der Übersetzung des Projektes zu einem Warning.

In der Logik des Skriptes muss berücksichtigt werden, wenn ein Skriptaufruf das Ergebnis eines anderen Skriptes benötigt. Die Ergebnisse in den Objekten sollten so abgesichert sein,

dass Null-Werte oder veraltete Werte nicht verwendet werden. Betrachten wir im folgenden Kapitel den Skriptaufruf durch eine Wertänderung.

10.6.1 Zyklischer Skriptaufruf durch eine Wertänderung

Das TIA-Projekt zu diesem Kapitel soll wieder unsere beiden, bekannten Kreise färben (**Kapitel 10.5**). Allerdings soll das solange automatisch geschehen, bis beide Kreise die gleiche Farbe haben, ohne dass die Gefahr besteht eine **interne** Endlos-Schleife zu programmieren. In **Bild** 10.16 sind die Bedingungen für den Aufruf des Skriptes MySkript zu sehen. Der *DB-Zufallszahl* (**Punkt 3**) und die HMI-Variablen (**Punkt 4**) wurden erweitert.

Es ist darauf zu achten, dass die Variablen *_ZyklusAktiv* und *_Dauerlauf* auf den Erfassungszyklus von 100 ms eingestellt sind (**Punkt 4**).

Diese beiden Variablen werden für den automatischen Zyklus benötigt und im *DB_Zufallszahl* auf die Startwerte 0 und *false* eingestellt. Die Farben sollen zwischen *Min* = 255 und *Max* = 16384 ausgesucht werden. Die Werte sind einfach so festgelegt und haben keine besondere Bedeutung. Bevor der Skript *MySkript* (**Punkt 2**) zum ersten Mal gestartet wird, muss die Variable *_ZyklusAktiv* auf *True* gesetzt sein (**Punkt 1**). Die Erklärungen dazu im folgenden Kapitel.

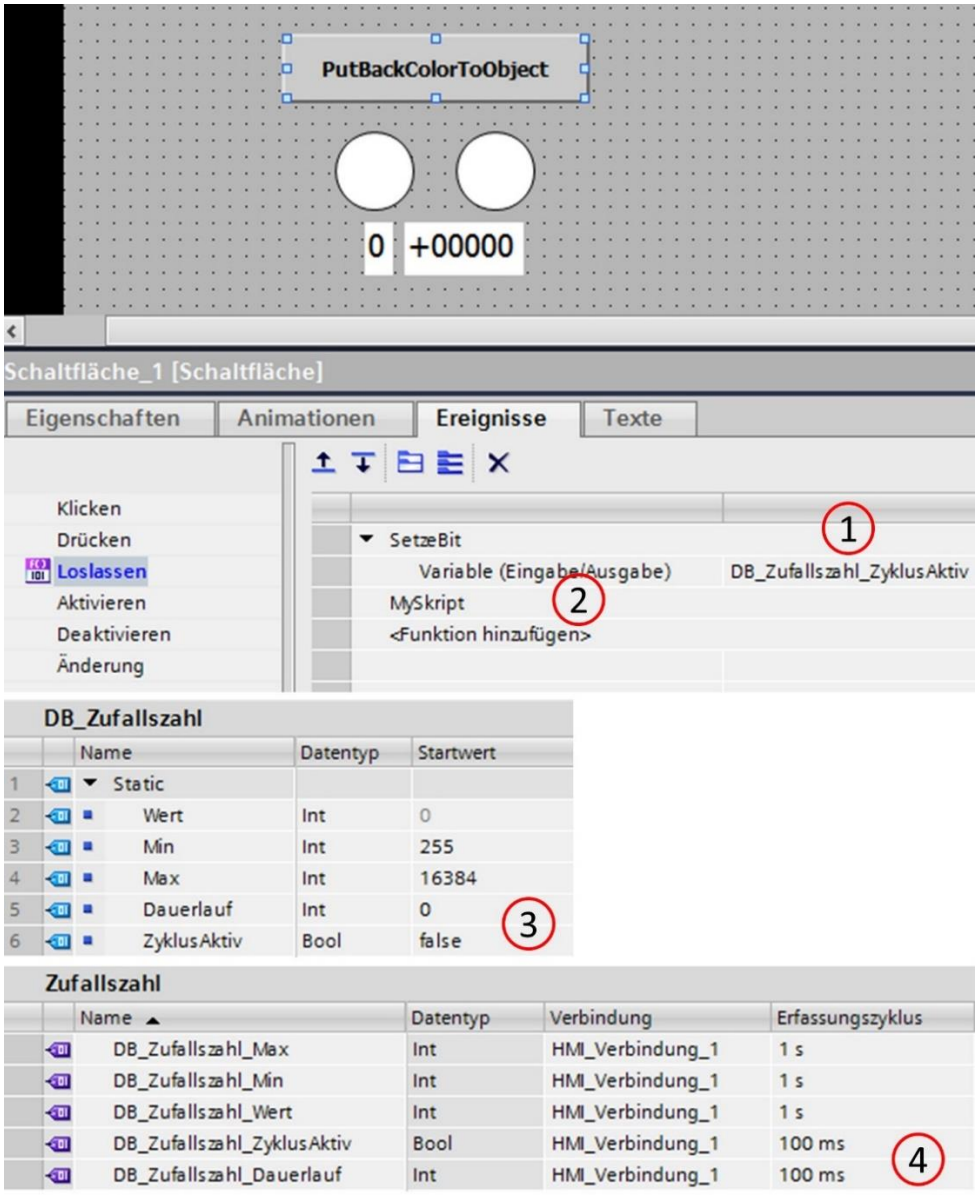


Bild 10.16 Aufrufbedingungen für den Skript *MySkript*

10.6.1.1 Der Skript *MySkript*

Der zyklische Aufruf des Skriptes *MySkript()* wird zuerst durch die Funktionsliste eingeleitet. In (**Bild** 10.17). ist das Listing zu sehen.

```
1  Sub MySkript()
2
3    Dim object_1, object_2
4    Dim min,max
5
6    'nur aufrufen, wenn Zyklus gesetzt ist
7    If SmartTags("DB_Zufallszahl_ZyklusAktiv")Then
8      min = SmartTags("DB_Zufallszahl_Min")
9      max = SmartTags("DB_Zufallszahl_Max")
10     'objekte einbinden
11       Set object_1 =  HmiRuntime.Screens("Grundbild")._
12                              ScreenItems("PLC_HMI_Anzeige_1")
13       Set object_2 =  HmiRuntime.Screens("Grundbild")._
14                              ScreenItems("PLC_HMI_Anzeige_2")
15     'Kreise faerben
16       object_1.BackColor = PutBackColorToObject_Func(min,max)
17       PutBackColorToObject_Sub min, max, object_2
18     'pruefen ob beide Kreise gleich sind
19       If(object_1.BackColor = object_2.BackColor)Then
20         SmartTags("DB_Zufallszahl_ZyklusAktiv") = False
21       End If
22       'wiederhole skriptaufruf
23         SmartTags("DB_Zufallszahl_Dauerlauf") = _
24                   SmartTags("DB_Zufallszahl_Dauerlauf") + 1
25         'Ueberlauf absichern
26         If(SmartTags("DB_Zufallszahl_Dauerlauf") > 10000)Then
27           SmartTags("DB_Zufallszahl_Dauerlauf") = 0
28         End If
29   Else
30     SmartTags("DB_Zufallszahl_Dauerlauf") = 0
31   End If
32
33 End Sub
```

Bild 10.17 Der Skript *MySkript()*

Zur besseren Darstellung wurden einige Objekte dimensioniert, sodass die Darstellung, speziell für dieses Kapitel im Buch besser lesbar wird. Auf die Performance des Skriptes hat das keinen wesentlichen Einfluss. Für die Aufrufe zur Bestimmung der Zufallszahl, werden die Objekte aus dem Bild eingebunden (Zeilen 11+13). In Zeile 16 erfolgt der Aufruf der Funktion *PutBachColorToObject_Func*. Der Rückgabewert der Funktion wird direkt in das *object_1.BackColor* gespeichert. In Zeile 17 erfolgt der Aufruf der Subroutine *PutBachColorToObject_Sub*, welche intern direkt in den Parameter *object_2* die Hintergrundfarbe verändert. Fällt der Vergleich in Zeile 19 mit *True* aus, dann wird die Variable *DB_Zufallszahl_ZyklusAktiv* auf *False* gesetzt, so dass beim nächsten Aufruf des Skriptes dieser in die Zeile 30 in den *Else*-Teil verzweigt (Zeile 7) und somit die Wertänderung in Zeile 23 nicht mehr stattfindet. Das hat zu guter Letzt zur Folge, dass der automatische Zyklus zu Ende ist.

In **Bild 10.18** ist ein zufälliges Ergebnis sichtbar. Bis beide Farben gleich sind, vergingen einige Sekunden. Dieses Prinzip, den Skript automatisch zyklisch aufzurufen, wird auch bei der Mausbewegung genutzt (**Kapitel 3**).

Der automatische Aufruf eines Skriptes über eine Wertänderung, welche der Skript selbst ausübt, funktioniert nur mit einer PLC-Variablen.

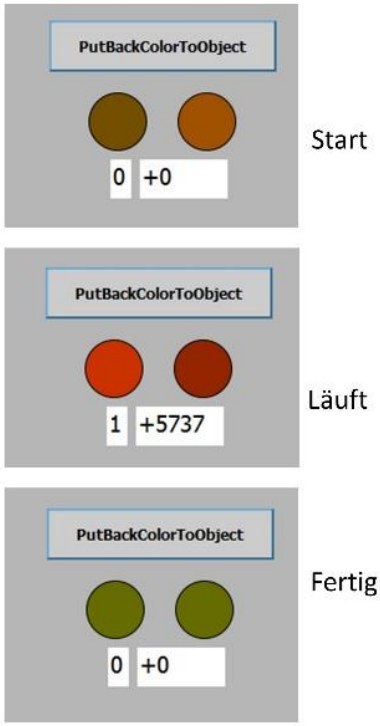


Bild 10.18 Das Ergebnis durch den Zufallsgenerator im automatischen Zyklus

Betrachtet man die Zahlenfolge während der Skript läuft, dann kann man sehr gut erkennen, dass der Zyklus mit 1000 Wertänderungen (PC-abhängig) kleiner als eine Sekunde beträgt. Also 1kHz und das ist für viele Anwendungen in der HMI völlig ausreichend.

10.7 Der Dateizugriff

Lernschritte:

- Umgang mit Deskriptoren
- Daten aus der SPS schreiben, anhängen und lesen

Dateien auf ein Zielsystem zu schreiben ist eine häufig gestellte Aufgabe. Hier unterscheidet sich „WinCC-VBS“ gegenüber VBS auf anderen Betriebssystemen, wie z.B. Windows-

Betriebssystemen. Besonders in WinCC Advanced müssen einige Dinge bezüglich der Anwendung beachtet werden. Das folgende Beispiel in **Kapitel 10.7.3** zeigt einen zyklischen *Write* auf die Festplatte. Zuvor einige Worte über den Filedeskriptor, welcher für einen Dateizugriff wie z.B. Lesen, Schreiben oder Anhängen eine notwendige Voraussetzung ist.

10.7.1 Der Filedeskriptor

Der File-Zugriff erfolgt über einen sogenannten Filedeskriptor. Einfach erklärt holt sich WinCC die Erlaubnis vom Betriebssystem mit dem Dateisystem zu kommunizieren, wie z.B. das Schreiben, Lesen oder Löschen von Dateien. Der Filedeskriptor wird über die Einbindung (*Set*) im aktuellen Skript erzeugt:

```
Set fso = CreateObject("Scripting.FileSystemObject")
```

Ist kein Fehler aufgetreten, dann liefert *fso* die Verbindung zu einer Struktur. In (**Bild** 10.19) ist ein Auszug der Struktur mit dessen Möglichkeiten zu sehen.

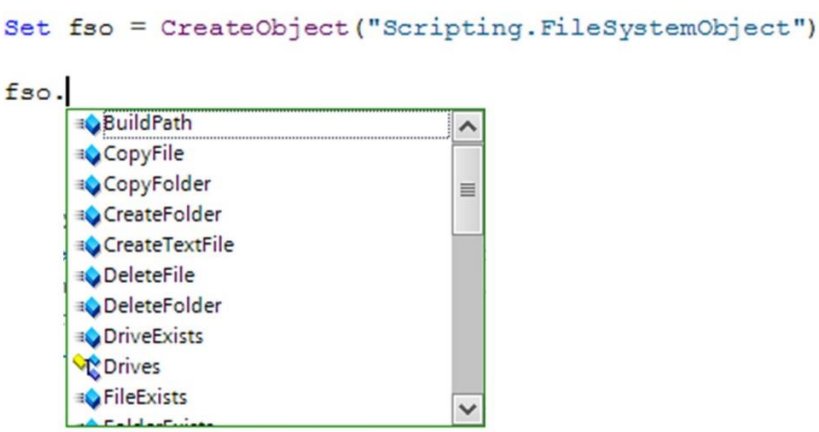


Bild 10.19 Der Filedeskriptor

Der Filedeskriptor, im Beispiel in *fso* gespeichert, sollte nur innerhalb des Skriptes verwendet werden und **nicht** über einen Parameter außerhalb des Skriptes verwendet werden.

Tritt bei der Zuweisung mit *CreateObject* ein Fehler auf, so sollte der Skript verlassen werden.

Der Fehler kann über die globale Struktur *Err* ermittelt werden.

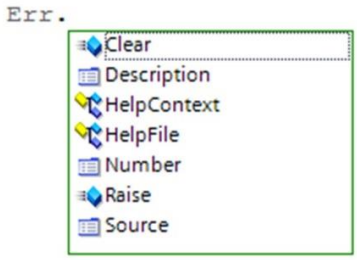


Bild 10.20 Die globale Struktur *Err*

In **Bild 10.20** sind die Möglichkeiten aufgezeigt, was die globale Variable *Err* so alles anbietet. Ist *Err.Number=0*, dann kann mit dem Deskriptor gearbeitet werden. Das werden wir uns im folgenden Kapitel ansehen.

10.7.2 In eine Datei schreiben, lesen oder anhängen

Nachdem der Filedeskriptor erfolgreich zugewiesen wurde, möchten wir eine Text-Datei zum Schreiben auf die Festplatte öffnen.

```
Set fileOut = fso.OpenTextFile(FileName, WRITE, True)
```

Das erfolgt mit der Methode *fso.OpenTextFile* und den entsprechenden Parametern. Auch hier muss die globale Variable *Err* überprüft werden, ob das Öffnen der Datei ohne Fehler funktioniert hat. Ist ein Fehler aufgetreten, sollte der Skript verlassen werden. In diesem Fall muss der schon zugewiesene Filedeskriptor *fso* nicht mehr berücksichtigt werden (kein *nothing* verwenden), da dieser wie alle anderen Variablen automatisch nach dem Verlassen des Skriptes gelöscht werden (Destruktor-Funktion).

Parameter:
Der Parameter *FileName* muss komplett angegeben werden. Im Buch wird immer das Verzeichnis zum Basisprojekt verwendet:

```
C:/TIA-Expert/MoveObjects/MoveObjects_Basic/Objects
```

Der Parameter *WRITE* ist eine Konstante und teilt dem *OpenTextFile* mit, dass Daten geschrieben werden sollen.

Die Datei wird **neu** angelegt, auch wenn diese schon existiert. In diesem Fall geht der Inhalt der schon vorhandenen Datei verloren.

Möchte man Daten in eine schon bestehende Datei schreiben (anhängen), dann wird die Konstante *APPEND* anstatt *WRITE* verwendet. Soll eine Datei zum Lesen geöffnet werden, so ist die Konstante *READ* anzuwenden.

Der Parameter *True* ist optional und kann auch weggelassen werden. Er ermöglicht gewisse Dinge zu erlauben, wie z.B. das Locken eines Files, damit andere Programme in diesem Moment nicht auf die Datei zugreifen dürfen. Auch können Dateien erzeugt werden, falls diese noch nicht existieren (*APPEND*).

Im folgenden Kapitel betrachten wir ein praktisches Beispiel für das Schreiben von Daten in einer extremen Situation betrachten.

10.7.3 Beispiel zum Dateizugriff unter Extrem-Bedingung

Betrachten wir das Beispiel aus **Kapitel 10.6** und stellen uns folgende Aufgabe:

Nachdem die beiden Farben durch den Zufallsgenerator ermittelt wurden, sollen diese durch ein Komma getrennt in eine Textdatei gespeichert werden.

Eine schon gehobene Aufgabenstellung, da dies aus einem automatischen Zyklus geschehen soll. Es darf keine Störung innerhalb dieser Schleife auftreten, da sonst der Skript durch *WinCC* abgebrochen wird (Schutzmechanismus). Betrachten wir das TIA-Projekt zu diesem Kapitel, dann sehen wir in **Bild 10.21** die Änderungen gegenüber dem **Kapitel 10.6** bezüglich der Bedienung der Schaltfläche *PutBackColorToObject*. Diese darf nach dem Start nicht mehr bedienbar sein. Dafür sorgt die Variable *_ZyklusAktiv* da diese dann *True* sein wird.

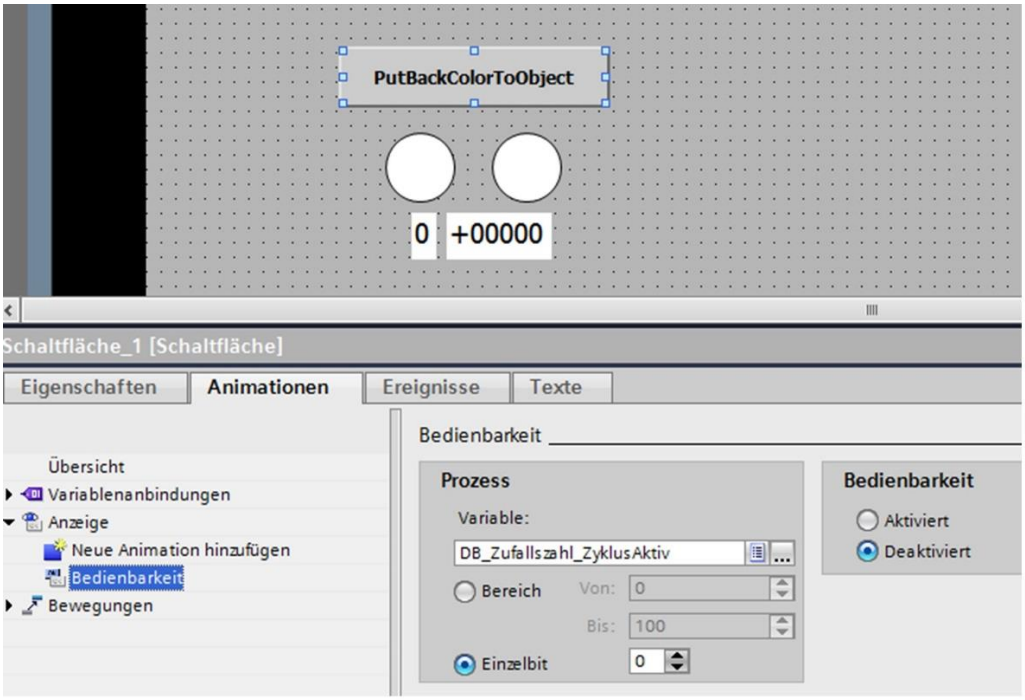


Bild 10.21 Die Schaltfläche wird in der Bedienung begrenzt

Da die Datei mit den Farb-Nummern spätestens ab dem zweiten Durchlauf schon existiert, muss diese zuerst gelöscht werden.

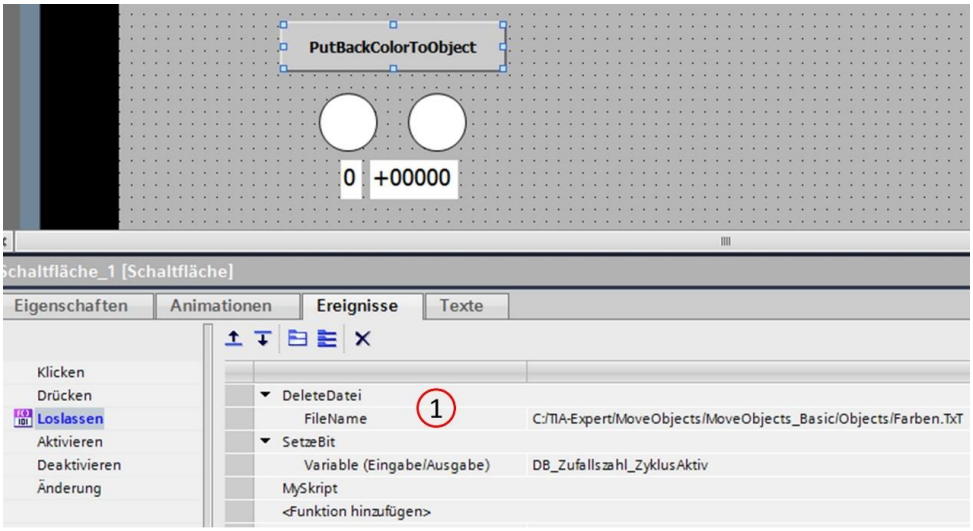


Bild 10.22 Die Skriptliste zur Schaltfläche

Dazu dient der neue Skript *DeleteDatei* welcher in der Skriptliste **zuerst** aufgerufen wird (**Punkt 1**). Als Parameter wird der komplette Dateinamen übergeben (Pfad + Name + Dateibezeichnung). Bei der Eingabe *FileName* muss auf String umgeschaltet werden, sonst erhalten Sie eine Fehlermeldung, dass der Dateiname ungültig ist. Zum Schluss wird der Skript *MySkript* aufgerufen. In diesem Skript müssen wir den Aufruf des Skriptes *WriteDatas* noch einfügen.

10.7.3.1 Der Skript *DeleteDatei*

Im oberen Teile des Skriptes *DeleteDatei()* (**Bild 10.23**) wird der Filedeskriptor zugewiesen (Zeilen 5-9). Die Fehlerbehandlung kann in **Kapitel 9** nachgelesen werden. In den Zeilen 13+14 (**Punkt 1**) wird der File mit *DeleteFile* gelöscht, falls dieser vorhanden ist (Funktion *FileExists)*.

```
1  Sub DeletDatei(ByRef FileName)
2
3  Dim fso
4    SmartTags("DB_Error_Texte_FileDescriptor_txt") = ""
5    Set fso = CreateObject("Scripting.FileSystemObject")
6    If Err.Number <> 0 Then
7      Set fso = CreateObject("Scripting.FileSystemObject")
8      SmartTags("DB_Error_Texte_FileDescriptor_txt") = _
9                    Err.Description & ": Create Filedescriptor"
10     Err.Clear
11   Else
12     'jetzt kann die Datei gelöscht werden
13     If(fso.FileExists(FileName)) Then
14       fso.DeleteFile FileName                    (1)
15     End If
16   End If
17
18 End Sub
```

Bild 10.23 Der Skript *DeleteDatei*

10.7.3.2 Der Skript *MySkript*

Im folgenden **Bild** 10.24 werden nur die Änderungen des Skriptes *MySkript* dargestellt, da in **Kapitel 10.6.1.1** der Skript schon vorgestellt wurde.

```
5   Dim Folder, DateiName
6                                          (1)
7   Folder = "C:/TIA-Expert/MoveObjects/MoveObjects_Basic/Objects/"
8   DateiName = "Farben.TXT"

19      ' Farben speichern                 (2)
20      WriteDatas Folder&DateiName,object_1.BackColor,object_2.BackColor
```

Bild 10.24 Die Änderungen in *MySkript*

In Zeile 5 wird der *Folder* und der *DateiName* deklariert, die dann in den Zeilen 7+8 zugewiesen werden (**Punkt 1**). In **Punkt 2** wird der Skript *WriteDatas* aufgerufen. Hier werden die Parameter übergeben, damit der Skript die Farben in die Datei schreiben kann.

Eine etwas andere Form, anstatt die Parameter in der Funktionsliste zu übergeben, da der Parameter *FileName* vom Skript aus einem *&*-Operator zusammengesetzt wird.

Nun wird durch den automatischen Zyklus die Datei *Farben.TXT* ständig erweitert. Sie werden sehen, dass dieses Prinzip sehr gut funktioniert und sicherlich anregt, andere Dinge auf dieser Basis zu konstruieren.

Betrachten Sie dieses Beispiel als Vorlage für weitere tolle Ideen, mit Skripten in einer automatischen Schleife zu arbeiten.

10.7.3.3 Der Skript *WriteDatas*

Der Skript *WriteDatas* in **Bild 10.25** muss die Daten aus den Parametern in den Textfile anhängen. Betrachten wir zunächst den Kommentar in Zeile 7.

Das Statement ***On Error Resume Next*** kann bei einem Laufzeitfehler den Abbruch des Skriptes verhindern.

Der Skript würde also weiter laufen und benötigt dazu eine Stelle, wo dieser wieder den Skript aufnehmen soll. Das wird durch das Statement ***On Error goto 0*** bekannt gegeben. In unserem Beispiel in Zeile 28. Da wir den Fehler auswerten und den Skript verlassen, ist das nicht notwendig. Könnte aber für eine erweiterte Fehlerbehandlung an einer anderen Stelle des Skriptes interessant sein.

```
1  Sub WriteDatas(ByRef FileName, ByRef Farbe_1, ByRef Farbe_2)
2
3  Const APPEND = 8
4  Dim fso, fileOut
5
6    SmartTags("DB_Error_Texte_FileDescriptor_txt") = ""
7    'On Error Resume Next
8    Set fso = CreateObject("Scripting.FileSystemObject")
9    If Err.Number <> 0 Then
10     Set fso = CreateObject("Scripting.FileSystemObject")
11     SmartTags("DB_Error_Texte_FileDescriptor_txt") = _
12                   Err.Description & ": Create Filedescriptor"
13     Err.Clear
14   Else
15     'create file open
16     Set fileOut = fso.OpenTextFile(FileName, APPEND, True)
17     If Err.Number <> 0 Then
18       Set fso = CreateObject("Scripting.FileSystemObject")
19       SmartTags("DB_Error_Texte_FileDescriptor_txt") = _
20                     Err.Description & ": Open for Write"
21       Err.Clear
22     Else
23       'jetzt kann geschrieben werden
24       fileOut.WriteLine(Farbe_1 & ", " & Farbe_2)
25       fileOut.Close
26     End If
27   End If
28   'On Error GoTo 0
29
30 End Sub
```

Bild 10.25 Der Skript *WriteDatas*

Die Zeilen 8-21 sind bereits aus **Kapitel 9** bekannt, nur dass in Zeile 16 der *OpenTextFile* mit *APPEND* geöffnet wird.

Auch hier ist der letzte Parameter mit *True* wertvoll, denn falls die Datei nicht existiert und das ist bei unserem Beispiel der Fall, wird die Datei neu angelegt.

In Zeile 24 werden die Farben mit dem *&*-Operator und einem Komma zusammengesetzt und als Parameter dem Skript *WriteLine* übergeben. Zum Schluss wird der File mit *fileOut.Close* ordnungsgemäß geschlossen.

Ein geöffneter File muss wieder geschlossen werden.

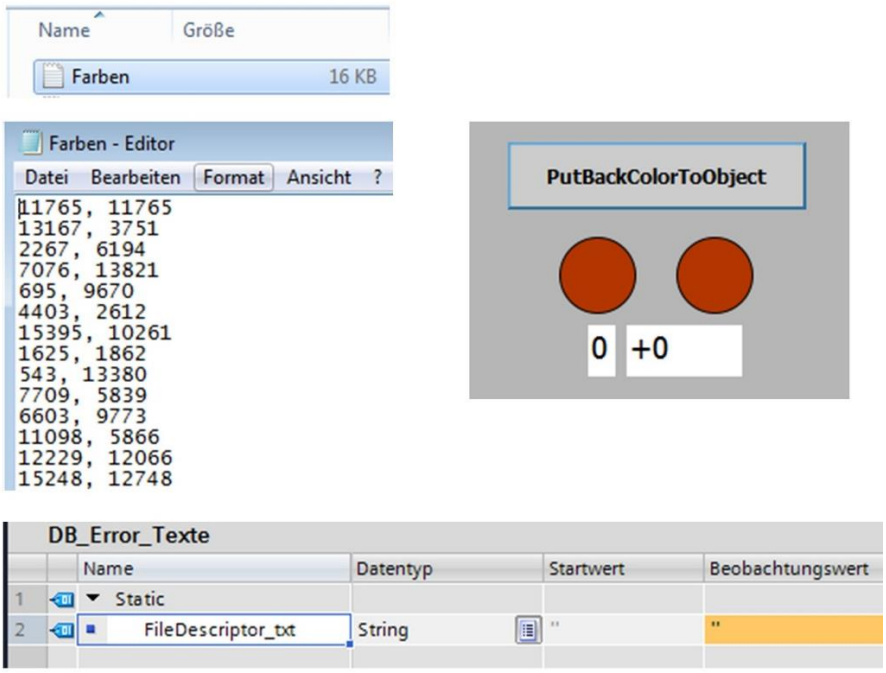


Bild 10.26 Das Ergebnis nach einem Start im zyklischen Rhythmus

Im oberen Teil des Bildes **Bild** 10.26 ist die Datei Farben.TXT auf 16KB angewachsen, bis beide Farben durch den Zufallsgenerator gleich waren. Im Bild ist auch ein Ausschnitt der Datei zu sehen. Die beiden letzten Werte sind identisch und entsprechen den roten Kreisen. Im unteren Teil ist der *DB_Error_Texte.FileDescriptor_txt* im Beobachtungsmodus zu sehen. Es ist während dieser Laufzeit kein Fehler aufgetreten. Das Ermitteln der Farben kann eine ganze Weile dauern.

11 Die C/C++ - Schnittstelle

Die Mauskoordinaten werden über **MausAddOn.EXE** im Skript *UserAction* über den Skript *GetMousePosition* geladen (siehe **Bild 3.7**). Der Austausch der Koordinaten erfolgt über das File-System. Ein ausgeklügeltes Chema nach dem Kollussionsprinzip. Der Algorithmus soll hier nicht vorgestellt werden, da das Medium Buch dazu ungeeignet ist. Damit Sie als Leser die Möglichkeit haben einen eigenen Maustreiber zu erarbeiten, wird im folgenden Kapitel eine C/C++-Schnittstelle vorgestellt, mit der grundsätzlich Daten über das File-System zwischen einem C/C++-Programm und der HMI (VBS) kommuniziert werden können. Dieses Prinzip funktioniert sehr gut und kann beliebig erweitert und verbessert werden.

11.1 Synchronisation einer Schnittstelle

Ein *Request* verursacht beim Kommunikationspartner, dass dieser darauf reagieren soll und entsprechend eine Gegenaktion auslöst (Client-Server-Prinzip). Bei der Kommunikation über ein File-System ist das nicht so einfach. Es fehlt eine zeitliche Abstimmung, wann wird geschrieben und wann gelesen. Wie schon in der Einleitung erwähnt, könnte das auf dem Prinzip der Kollision erfolgen. Die Performance der Hardware eines PCs (Festplatte) nach dem heutigen Stand zur Zeit der Bucherstellung (2016) unterstützt den File-Austausch mehr denn je. D. h. beide Programme versuchen einen File-Zugriff in der Hoffnung, dass zu diesem Zeitpunkt der Zugriff auf das File-System erlaubt wird (gleichzeitiger Zugriff für Read/Write ist nicht möglich). Ist machbar, lässt sich aber sehr schlecht dokumentieren.

Welche Basis kann gefunden werden, damit Daten über das File-System ohne Kollision ausgetauscht werden können?

Das C/C++-Programm befindet sich auf dem PC und das HMI-Programm ebenfalls. Gemeinsam können beide Programme Daten ohne eine Kollision austauschen, wenn sich beide an eine Zeitscheibe halten.

Die gemeinsame Zeitscheibe ist die Systemzeit des PCs aufgelöst in Millisekunden.

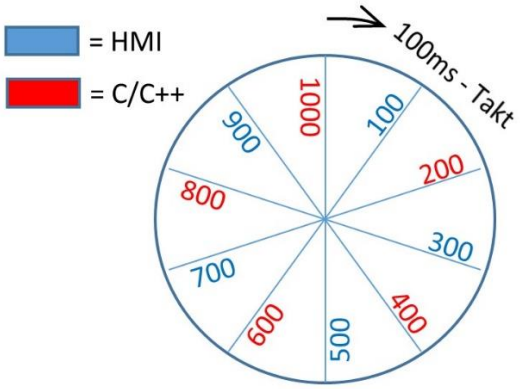


Bild 11.1 Die Zeitscheibe mit 100ms je Teilnehmer

Da beide Programme die gleiche Zeitbasis verwenden, könnte man hier jedem Programm z. B. 100 ms zuordnen. Zuerst liest die HMI die Koordinaten (**Bild** 11.1), dann schreibt die SPS die neuen Koordinaten, dann liest die HMI wieder usw.

Dieses Prinzip ist auch mit kleineren Zeitabständen möglich. Die Zeitabstände müssen auf dem System ermittelt werden. Diese ist auch abhängig, wieviel Daten werden innerhalb des Zeitabschnittes gelesen bzw. geschrieben. Auch sollte zwischen den Übergängen (Blau-Rot) eine kleine Reserve vorhanden sein, sonst gibt es womöglich doch noch Kollisionen.

Die HMI liest die Koordinaten mit dem Ereignis der Schaltfläche und die SPS schreibt die Koordinaten **kontinuierlich** für die aktuelle X- und Y-Position.

Diese Aufgabenstellung soll im folgenden Kapitel umgesetzt werden.

11.2 Datenaustausch C/C++ und HMI

Bei dieser Aufgabenstellung geht es um das Prinzip einer C/C++-Schnittstelle über eine gemeinsame Zeitscheibe von 100 ms. Das TIA-Projekt zu diesem Kapitel zeigt die Umsetzung dazu. Basis ist das im **Kapitel 3** gezeigte zyklische Skript-Intervall mit dem Kreis-Objekt. Betrachten wir dazu noch einmal den Skript *UserAction*.

11.2.1 Der Skript *UserAction*

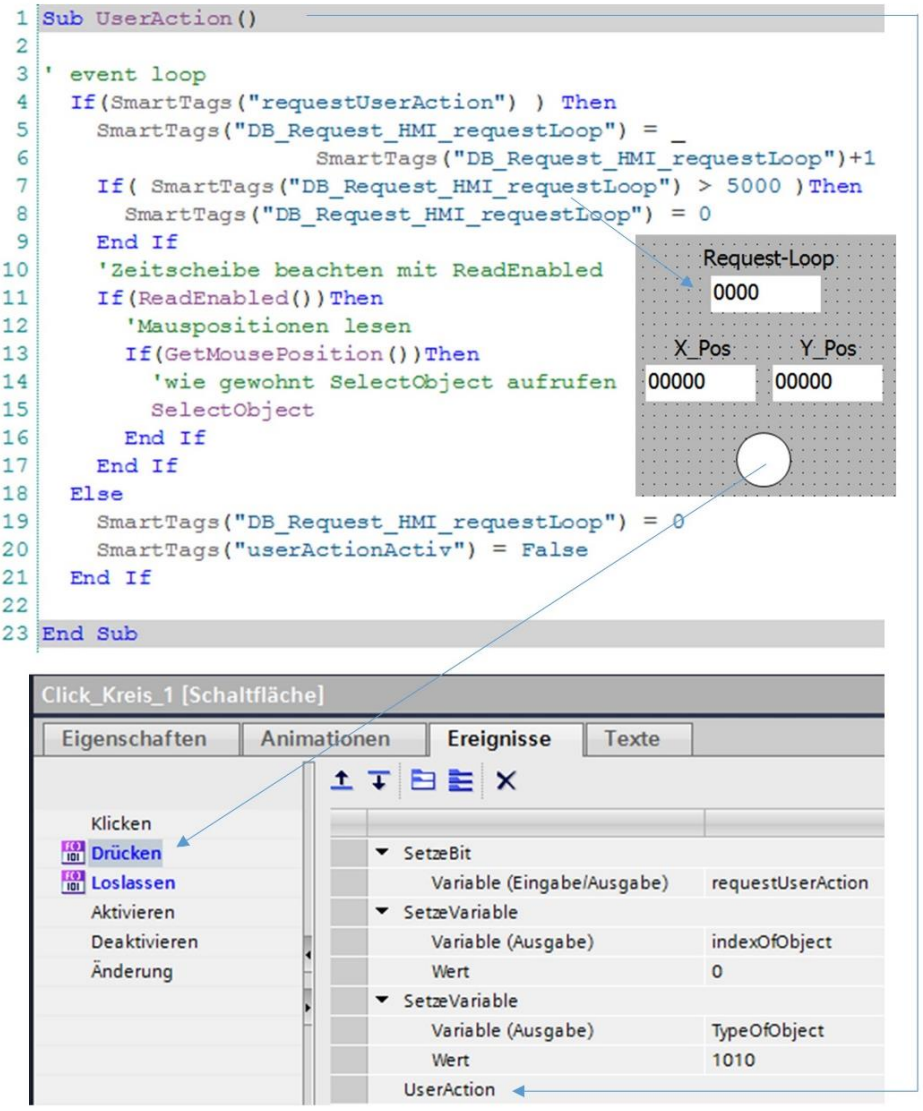


Bild 11.2 Der Skript *UserAction*

Durch Drücken auf die Schaltfläche des Kreises, wird das Ereignis wie in **Bild 11.2** dargestellt ausgelöst. Im Skript *UserAction* wird in Zeile 11 die Funktion *ReadEnabled* aufgerufen und bei *True* die Mauspositionen mit dem Skript *GetMousePosition* eingelesen. Danach wird der Skript *Select* aufgerufen, welcher die Position des Kreises den Maus-Koordinaten anpassen soll. *ReadEnabled* liefert alle 100 ms ein True.

11.2.2 Der Skript *ReadEnabled*

```
1  Function ReadEnabled()
2
3  Dim ms
4
5  'aktuelle Zeit ermitteln
6    ms = (Timer()-((Hour(Now)*3600)+(Minute(Now)*60)+Second(Now)))
7    ms = Int(ms*1000)
8
9  'gueltiger zeitrahmen ermitteln
10   If( (ms > 0 And ms < 100) _
11     Or _
12     (ms > 200 And ms < 300) _
13     Or _
14     (ms > 400 And ms < 500) _
15     Or _
16     (ms > 600 And ms < 700) _
17     Or _
18     (ms > 800 And ms < 900) ) Then
19
20       If( Not SmartTags("readFile") )Then
21         SmartTags("readFile") = True
22         ReadEnabled = True
23         Exit Function
24       End If
25
26   Else
27     SmartTags("readFile") = False
28   End If
29
30 ReadEnabled = False
```

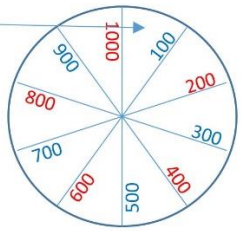


Bild 11.3 Der Skript *ReadEnabled*

Die HMI darf nur lesen, wenn die Bedingung des aktuellen Zeitwertes sich in den blauen Feldern befindet. In den Zeilen 6+7 wird die Zeit ermittelt. Da *Timer* die aktuelle Zeit bis hundertstel liefert und *Now* die Zeit ohne hundertstel, bleiben durch die Subtraktion die Hundertstel übrig (Zeile 6). Diese müssen dann noch mit *Int* aufbereitet werden (Zeile 7). Innerhalb der 100 ms soll nur einmal gelesen werden, deswegen in Zeile 22 die Variable *ReadEnabled*, welche auf *True* gesetzt wird. Diese wird dann in Zeile 30 wieder auf *False* gesetzt, während die SPS die neuen Koordinaten schreibt. In Zeile 23 liefert der Skript durch *Exit Function* alle 100 ms ein *True*, sodass die Mauskoordinaten gelesen werden können. Das schauen wir uns im folgenden Kapitel an.

11.2.3 Der Skript *GetMousePosition*

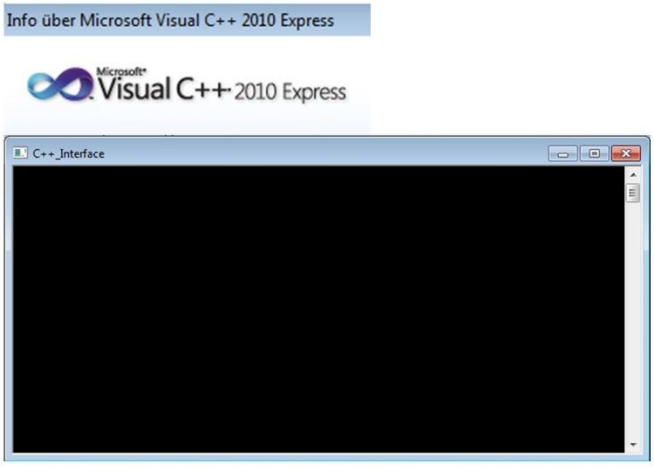
Das Lesen der Mauskoordinaten erfolgt immer dann, wenn das SPS-Programm **nicht** schreibt, so wird eine Kollusion innerhalb der beiden Programme vermieden. In **Bild** 11.4 ist das Listing zu *GetMousePosition* zu sehen.

```
Function GetMousePosition()

Const READ     = 1

Dim fileName, folder, fso, fileOutX, fileOutY
Dim object
  'Verzeichnis festlegen
  folder = "C:/TIA-Expert/MoveObjects/MoveObjects_Basic/Request/"
  fileName = folder & "X_Pos" & ".txt"
  GetMousePosition = False
  'create descriptor
    Set fso = CreateObject("Scripting.FileSystemObject")
    If Err.Number <> 0 Then
      SmartTags("DB_Error_Texte_FileDescriptor_txt") _
                          = "CreateObject: " & Err.Description
        Err.Clear
    Else
      'create file open
      Set fileOutX = fso.OpenTextFile(fileName,READ,True)
      If Err.Number <> 0 Then
        SmartTags("DB_Error_Texte_FileDescriptor_txt") _
                          = "OpenTextFile_x: " & Err.Description
        Err.Clear
      Else
        fileName = folder & "Y_Pos" & ".txt"
        'create file open
          Set fileOutY  = fso.OpenTextFile(fileName,READ,True)
          If Err.Number <> 0 Then
            SmartTags("DB_Error_Texte_FileDescriptor_txt") _
                          = "OpenTextFile_y: " & Err.Description
            Err.Clear
          Else
            SmartTags("mouseXpos") = fileOutX.ReadAll
            SmartTags("mouseYpos") = fileOutY.ReadAll
            fileOutX.Close
            fileOutY.Close
            GetMousePosition = True
          End If
      End If
    End If
    'end descriptor

End Function
```

Bild 11.4 Der Skript *GetMousePosition*

Wird in den Zeilen 20 oder 28 ein Fehler entstehen, kann das in der SPS mit *_Texte_FileDescriptor_txt* im Beobachtungsmodus beobachtet werden. Das sollte fehlerfrei laufen. Die Positionen werden vom C/C++-Programm für X und Y getrennt in zwei Dateien geschrieben (Zeilen 33+34). Das kann natürlich vom SPS-Programm innerhalb einer Datei geschrieben werden. Es gibt für die hier verwendeten zwei Dateien keinen nennenswerten Grund, außer dass in VBS die Positionen X und Y dann wieder selektiert werden müssten. Im folgenden Kapitel ist das C/C++-Programm ersichtlich, welches die beiden Dateien *X_Pos.TXT* und *X_Pos.TXT* erzeugt.

11.2.4 Das C/C++-Programm




```
1  // C/C++-Interface
2  // Kapitel 11.2
3
4  #include "stdafx.h"
5  #define DAT_NAME_XPOS   L"C://TIA-Expert//MoveObjects//MoveObjects_Basic//Request//X_Pos.TXT"
6  #define DAT_NAME_YPOS   L"C://TIA-Expert//MoveObjects//MoveObjects_Basic//Request//Y_Pos.TXT"
7
8  int _tmain(int argc, _TCHAR* argv[])
9  {
10     SetConsoleTitle(TEXT("C++_Interface"));
11
12     HANDLE      write2File;
13     SYSTEMTIME  localTime;
14     CFile       file;
15     POINT       p;
16     char        buffer[128];
17     bool        send = false;
```

Bild 11.5 Teil-1 *C++_Interfase*

In **Bild** 11.5 ist der erste Teile des C/C++-Programmes ersichtlich. Das Programm wurde mit Visual C++ 2010 Express als Konsolenanwendung kompiliert. Die Dateiverzeichnisse (Zeilen 5+6) sind wie im HMI-Skript (**Bild 11.4**, Zeile 8) gleich, plus die Dateibezeichnung jeweils für *X_Pos.TXT* und *Y_Pos.TXT*.

Danach folgen die Deklarationen für das Schreiben der Dateien ab Zeile 19 (**Bild 11.6**). Hier wird zum Gegensatz des HMI-Skriptes mit der Zeitscheibe ab 100 ms begonnen. Damit die Dateien innerhalb der 100 ms nur einmal geschrieben werden, wird die Variable *send* verwendet, welche in Zeile 35 auf *true* gesetzt wird. In Zeile 53 wird *send* wieder auf *false* gesetzt, während die HMI die Dateien liest.

```
19  do{
20
21      GetLocalTime(&localTime);
22      if( (localTime.wMilliseconds > 100 && localTime.wMilliseconds < 200)
23          ||
24          (localTime.wMilliseconds > 300 && localTime.wMilliseconds < 400)
25          ||
26          (localTime.wMilliseconds > 500 && localTime.wMilliseconds < 600)
27          ||
28          (localTime.wMilliseconds > 700 && localTime.wMilliseconds < 800)
29          ||
30          (localTime.wMilliseconds > 900 && localTime.wMilliseconds < 1000) )
31      {
32          if(!send)
33          {
34              GetCursorPos(&p);
35              send = true;
36              write2File = file.OpenFile(DAT_NAME_XPOS);
37              if( write2File != NULL)
38              {
39                  sprintf_s(buffer,128,"%5d",p.x);
40                  file.WriteData(write2File,buffer,strlen(buffer));
41                  file.CloseFile(write2File);
42              }
43              write2File = file.OpenFile(DAT_NAME_YPOS);
44              if( write2File != NULL)
45              {
46                  sprintf_s(buffer,128,"%5d",p.y);
47                  file.WriteData(write2File,buffer,strlen(buffer));
48                  file.CloseFile(write2File);
49              }
50          }
51      }
52      else
53          send = false;
54  }while(true);
```

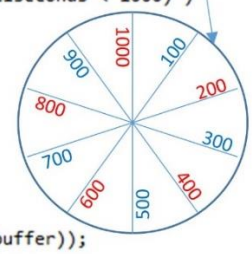


Bild 11.6 Die Zeitscheibe und das Schreiben der Dateien

Zur Vervollständigung sind im folgenden Kapitel die C++-Sourcen der Klasse *CFile* abgebildet, so dass Sie auch einen anderen C++-Kompiler verwendet können.

Denken Sie daran, das Programm MausAddOn.EXE für diesen Test auszuschalten und das Programm C++Schnittstelle.exe einzuschalten. Nachfolgend noch einmal das Verzeichnis: **C:\TIA-Expert\MoveObjects\MoveObjects_Basic\Request**

11.2.4.1 Die Klasse *CFile*

In den Bilder **Bild 11.7** und **Bild 11.8** sind die Programme der Klasse *CFile* ersichtlich. Die Deklaration ist in **Bild 11.5** in Zeile 14 zu sehen.

```
1  #pragma once
2
3  #include "stdafx.h"
4  #include <Windows.h>
5  #include <string>
6  using namespace std;
7
8  class CFile
9  {
10     public:
11
12         CFile(void);
13         virtual ~CFile(void);
14
15         HANDLE WINAPI OpenFile( const wstring& );
16         DWORD ReadData(HANDLE, LPVOID, DWORD);
17         DWORD WriteData(HANDLE, LPVOID, DWORD);
18         void CloseFile(HANDLE);
19
20     private:
21         DWORD dwBytesRead;
22         DWORD dwBytesWrite;
23 };
```

Bild 11.7 Die Header-Datei *File.h*

```
#include "StdAfx.h"

#include <sys/types.h>
#include <sys/stat.h>

#include "File.h"

CFile::CFile(void){}

CFile::~CFile(void){}

void CFile::CloseFile(HANDLE hwnd)
{
    CloseHandle(hwnd);
}

HANDLE WINAPI CFile::OpenFile( const wstring& sFileName)
{
    return  CreateFile( sFileName.c_str(), GENERIC_WRITE, 0,\
        NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
}

DWORD CFile::ReadData(HANDLE FileHandle, LPVOID buffer, DWORD anzByte)
{
    OVERLAPPED stOverlapped = {0};
    ReadFile(FileHandle, buffer, anzByte, &dwBytesRead, &stOverlapped);
    return dwBytesRead;
}

DWORD CFile::WriteData(HANDLE FileHandle, LPVOID buffer, DWORD anzByte)
{
    OVERLAPPED stOverlapped = {0};
    WriteFile(FileHandle, buffer, anzByte, &dwBytesWrite, &stOverlapped);
    return dwBytesWrite;
}
```

Bild 11.8 Die Sourcen der Klasse *File.cpp*

Das Zusammenspiel der beiden Programme (HMI und C/C++) zeigt das **Bild** 11.9. Die Kreisfläche wurde hier z. B. auf die Position 394, 273 verschoben. Im Beobachtungsmodus des Datenbausteines *DB_Error_Texte* ist während dieser Bewegung des Kreises keine Fehlermeldung registriert worden und zeigt, dass es keine Kollisionen gegeben hat. Die Zeitscheibe von 100 ms ist für eine Mausbewegung nicht optimal bezüglich der Dynamik. Hier kann die Zeitscheibe z.B. auf 50 ms verkürzt werden, damit ein besseres Zeitverhalten entsteht. Grundsätzlich können mit dieser Methode auch Daten von der SPS in eine Datei geschrieben werden, welche dann von einem Anwenderprogramm kontinuierlich gelesen und ausgewertet werden.

Je nach Aufgabenstellung muss die Zeitscheibe entsprechend angepasst werden. Der Auslöser könnte über eine Variablenänderung in der SPS erfolgen oder über eine Schaltfläche welche einen Report auslöst.

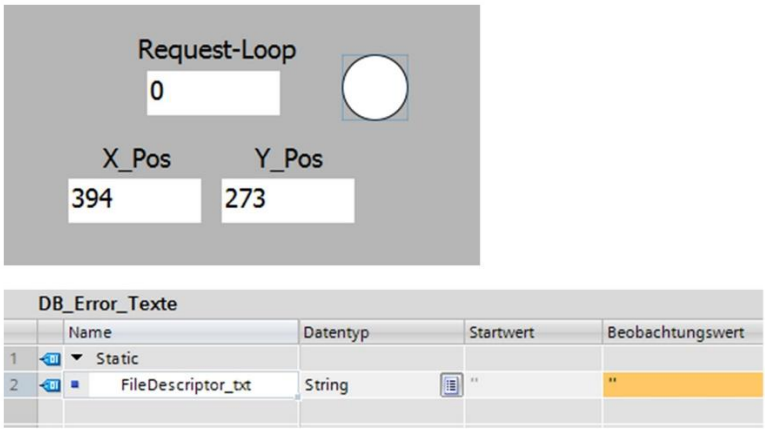


Bild 11.9 Test der Kommunikation zwischen HMI und C/C++-Programm

Mit dieser Anregung über eine gemeinsame Zeitscheibe zu kommunizieren denke ich, dass Sie daraus sicherlich eigene, neue Ideen entwickeln. Ich wünsche Ihnen viel Spaß und gutes Gelingen ☺.

K

L

M

N

O

P

R

S

T

U

V

W

Z

www.TIA-Expert.com

Das Fachbuch zeigt wie Bildelemente der HMI in Runtime mit der Maus wie auf einem Smartphone bewegt werden können. Stufenweise werden in den dargestellten TIA-Projekten die Programmierung der HMI und der SPS zur Bewegung eines Bildobjektes mit der Maus vorgestellt. Am Beispiel einer SPS (S7-1500®) und dem Touch-Monitor TP1900 Comfort beginnt die Programmierung mit einer einfachen Objektbewegung über einen Skript in VBS. Die daraus entstehenden Probleme werden einzeln analysiert und schrittweise zur Lösung geführt. Daraus entsteht schließlich eine Menüleiste mit Bildobjekten, welche es erlaubt Objekte mit der Maus, wie auf einem Smartphone, in die Bildoberfläche zu ziehen und zu positionieren. Auch die Änderung der Objektgröße (Höhe und Breite) ist ein Thema des Buches, damit sogenannte Quasi-Linien mit der Maus gezeichnet werden können.

Damit der Leser auch komplette Projekte im Zusammenspiel mit der SPS erzeugen kann, zeigt der Autor wie die Daten aus der SPS für die Bildobjekte auf die Festplatte geschrieben und von dieser wieder gelesen werden. So besteht die Möglichkeit, dass der Leser sich eigene Routinen schreibt um bewegte Bilder in Runtime von der Festplatte zu laden oder zu dokumentieren. Ein Kapitel zeigt wie extrem gut ein zyklusgesteuerter Skript für das das Schreiben von Daten aus der SPS auf die Festplatte funktionieren kann.

Zur Erklärung der Skripte in den TIA-Projekten wird ein Kapitel zur Einführung in die Sprachgrundelemente innerhalb VBS vorgestellt. Dieser Abschnitt ist kein Programmierkurs, sondern dient dem besseren Verständnis zu den gezeigten Skripten in den verschiedenen TIA-Projekten. Alle TIA-Projekte werden über die Simulation des TIA Portals (PLCSIM®) im Buch vorgestellt. Der Leser benötigt eine TIA Portal Version ab V13 SP1.

Mit den Kenntnissen der im Buch dargestellten Techniken, kann der Leser sein Fachwissen deutlich erweitern und neue Ideen zur Visualisierung umsetzen.

Aus dem Themenbereich:

- ⇒ **Bildobjekte in Runtime bewegen**
- ⇒ **Zyklisches Skript-Intervall**
- ⇒ **Die Menüleiste**
- ⇒ **Der Main-Cursor**
- ⇒ **Der Farb-Cursor**
- ⇒ **Objekte in der Größe verändern**
- ⇒ **Daten aus der SPS mit VBS zyklisch auf die Festplatte speichern**
- ⇒ **Einführung in Skripte schreiben**
- ⇒ **Die C++-Schnittstelle**

ISBN 978-84-617-8556-8