

JUNG Facility-Pilot Skript, Version 2.0

Inhalt:

1 EINFÜHRUNG.....	3
2 BETRIEBSARTEN.....	3
3 AUTOMATISCHE ABLÄUFE MIT AUTORUN.VISSCRIPT.....	4
4 DOKUMENTMODELL.....	4
4.1 SYSTEMVARIABLEN.....	4
4.2 DVISAPPLICATION.....	5
4.3 DVISPROJECT.....	6
4.4 DVISPROCESS.....	7
4.5 DVISPROCESSITEM.....	8
4.6 DVISHEET.....	10
4.7 DVISHEETPLANE.....	11
4.8 DVISHEETITEM.....	12
4.9 DVISHEETOLEITEM.....	14
4.10 DVISOLEVERB.....	14
4.11 DVISRECT.....	15
5 TIPPS.....	16
5.1 LOGIKFUNKTIONEN.....	16
5.2 WIE KANN ICH ZEITFUNKTIONEN REALISIEREN.....	17
5.3 DEBUGGING.....	17
5.4 VERZEICHNISSE.....	17
5.5 SONSTIGES.....	17

Ansprechpartner

ALBRECHT JUNG GMBH & CO. KG

Volmestraße 1
58579 Schalksmühle

Telefon +49 (0) 23 55/80 60
Telefax +49 (0) 23 55/80 61 89



mail.info@jung.de

Copyright

Copyright ©2006 ESF Software GmbH
Alle Rechte vorbehalten

Warenzeichen

EIB® ist ein eingetragenes Warenzeichen der EIB association (EIBA).

LON® ist ein eingetragenes Warenzeichen der Echelon Corporation registriert in Amerika und anderen Ländern.

OPC® ist ein eingetragenes Warenzeichen der OPC Foundation.

Sax Basic Engine ist ein eingetragenes Warenzeichen der Sax Software Corporation.

Adobe Acrobat® ist ein eingetragenes Warenzeichen der Adobe Systems Incorporated.

Microsoft®, ActiveX®, DirectX®, Windows®, Windows NT®, Excel®, Visual Basic® sind eingetragene Warenzeichen der Microsoft Corporation.

Alle Handelsnamen, Firmennamen und Produktnamen sind Warenzeichen oder eingetragene Warenzeichen ihrer jeweiligen Besitzer.

1 Einführung

VisScript ist eine in die Visualisierung integrierte BASIC Programmierumgebung. Sie beruht auf einer Anpassung von [SaxBasic](#), das ist ein zu [Visual Basic for Applications](#) (VBA) fast vollständig kompatibler BASIC- Dialekt. Damit ist die Programmierung mit VisScript ähnlich zu der Programmierung mit den Microsoft Office- Paketen.

Durch das Dokumentmodell der Visualisierung können mit VisScript kundenspezifische Lösungen programmiert werden, mit Zugriff auf den technischen Prozess und die Anzeige der Prozessbilder. Eine mit VisScript geschriebene Anwendung kann durch den Player der Visualisierung automatisch gestartet werden und läuft dann im Hintergrund mit. Zusätzlich kann VisScript auch für die Definition von Makros im Visualisierungseditor eingesetzt werden.

Mit VisScript können beispielsweise zeitabhängiges Verhalten oder Prozesslogik programmiert, Daten an andere Systeme übergeben, oder die Prozessbilder dynamisiert werden.

Wichtiger Hinweis:

Durch die Möglichkeiten der Skriptprogrammierung kann natürlich auch die Funktionsfähigkeit der Visualisierung und auch des angeschlossenen technischen Prozesses beeinträchtigt werden. Daher beachten Sie bitte die Hinweise in dieser Beschreibung. Weder für die Richtigkeit der Beschreibung, noch für die Tauglichkeit und mögliche Schäden aus dem Gebrauch der Skripte können wir Haftung übernehmen.

Die Skriptprogrammierung ist ein leistungsfähiges Werkzeug, das mit Bedacht eingesetzt werden muss. Das gilt insbesondere für Skripte, die automatisch aus dem Player gestartet werden und auf den technischen Prozess wirken. Wer mit Skripten arbeitet, die im Player aktiv sind, sollte Programmiererfahrung haben und wissen, was er tut: er tut es auf eigene Gefahr.

2 Betriebsarten

Für die Skripte gibt es zwei Betriebsarten:

- a) Im Editor können die Skripte als Makros verwendet werden. Bei der Ausführung des Skripts endet das Skript mit dem Ende der Skriptprozedur ([Public Sub Main](#)). Im Editor ist das Schreiben und Lesen von Prozessvariablen nicht möglich.
- b) Der Player startet automatisch das Skript [autorun.visscript](#) und darin die Skriptprozedur [Private Sub Main](#). Das Skript endet nicht mit dem Ende dieser Skriptprozedur, sondern bleibt aktiv, bis der Player beendet wird. Bei der Änderung von Prozessvariablen und zum Beispiel dem Ablauf von Timern werden in [autorun.visscript](#) vorhandene Prozeduren zur Behandlung der Ereignisse aufgerufen.

3 Automatische Abläufe mit **autorun.visscript**

Die Skriptdatei **autorun.visscript** wird vom Editor automatisch erzeugt und vom Player automatisch geladen. Sie hat folgenden Grundaufbau:

```
*** autorun.visscript ***  
Private Sub Main  
    'Diese Befehle werden beim Laden ausgeführt.  
End Sub  
  
Sub OnVisSheetLoaded(SheetName As String)  
    'Diese Befehle werden ausgeführt, wenn ein Arbeitsblatt geladen wird.  
End Sub  
  
Sub OnVisSheetClosed(SheetName As String)  
    'Diese Befehle werden ausgeführt, wenn ein Arbeitsblatt geschlossen wurde.  
End Sub  
  
Sub OnVisProcessEvent(ProcessModel As DVisProcessModel, _  
    ProcessItem As DVisProcessItem)  
    'Diese Befehle werden ausgeführt, wenn eine Prozessvariable geändert wurde  
End Sub  
  
Sub OnVisTimerEvent(TimerID As Long)  
    'Diese Befehle werden ausgeführt, wenn ein vom Skript gesetzter Timer  
    'abgelaufen ist.  
End Sub
```

Die bei Ereignissen aufgerufenen Prozeduren werden als Handler bezeichnet. Die Visualisierung stellt sicher, dass ein Handler nur aufgerufen wird, wenn kein Handler zum gleichen Zeitpunkt aktiv ist. Im Konfliktfall hat *OnVisProcessEvent* Vorrang vor der Behandlung anderer Ereignisse.

Um **autorun.visscript** zu verwenden, werden eigene Befehle eingesetzt und bei Bedarf weitere Prozeduren definiert, die als Befehle aufgerufen werden.

4 Dokumentmodell

Die Verbindung von Skripten mit der Visualisierung erfolgt durch den Aufruf der Methoden in **autorun.visscript** für Ereignisse und den Zugriff auf Objekte der Visualisierung durch das Dokumentmodell der Visualisierung, das im Folgenden beschrieben wird. Die Skripte beziehen sich immer auf das gesamte Visualisierungsprojekt.

4.1 Systemvariablen

In jedem Skript sind automatisch die Variablen *VisApplication* und *VisProject* definiert.

- 🟡 **VisApplication** ist der Zugriff auf die Visualisierungsapplikation. Die Variable hat die unter **DVisApplication** beschriebenen Eigenschaften.
- 🟡 **VisProject** ist der Zugriff auf das Visualisierungsprojekt und hat die Eigenschaften, die unter **DVisProject** beschrieben sind. Ausgehend von *VisProject* kann man auf Arbeitsblätter und deren Elemente, sowie auf Prozessvariablen zugreifen.

4.2 DVisApplication

DVisApplication enthält folgende Funktionen:

Boolean *PlaySound(String strWavFile)*

PlaySound spielt eine WAV- Datei oder einen Systemklang ab. Der Erfolg wird als Resultat zurückgegeben. Namen von Systemklänge findet man in den Audio-Einstellungen der Systemsteuerung.

Boolean *SetTimer(TimerID As Long, long TimeInterval As Long)*

Erzeugt bei Erfolg einen Timer mit der Nummer *TimerID*. Bis der Timer mit *KillTimer* gelöscht oder das Projekt im Player geschlossen wird, wird alle *TimeInterval* Millisekunden der Handler *OnVisTimerEvent(TimerID As Long)* aufgerufen. *TimerID* muss einen Wert größer als 0 haben. Die Funktion ist auf der Basis von WINDOWS Standard-Timerfunktionen realisiert. Es gelten daher die für WINDOWS-Timer geltenden Einschränkungen, insbesondere kann WINDOWS nur eine beschränkte Anzahl unterschiedlicher Timer zur Verfügung stellen und können diese Timer auch ungenau sein. Intervalle kleiner als 50 ms werden nicht empfohlen. Die Funktion ist nur im Zusammenhang mit automatischen Skripts (*autorun.visscript*) sinnvoll.

Boolean *KillTimer(TimerID As Long)*

Ein zuvor gesetzter Timer mit der Nummer *TimerID* wird gelöscht.

SysMsg(String strMsg)

Die Nachricht ***strMsg*** wird in das Systemfenster von Editor bzw. Player geschrieben.

4.3 DVisProject

DVisProject enthält folgende Funktionen und Eigenschaften:

Eigenschaften:

Rückgabe	Name	Zugriff	Beschreibung
String	Name	Lesen	Der Name des Projekts
String	ID	Lesen	Die ID des Projekts
Array	Sheets	Lesen	Liefert das Array der Arbeitsblätter
Array	ProcessModels	Lesen	Liefert das Array der Prozessmodelle, die mit dem Visualisierungsprojekt verwendet werden (in der aktuellen Version ist das höchstens ein Prozessmodell).

Beispiel: Durchlaufen aller Arbeitsblätter eines Visualisierungsprojektes.

```
Sub Main
    Dim PrjSheets
    PrjSheets = VisProject.Sheets

    Dim nSheets
    nSheets = UBound(PrjSheets) + 1

    For nSheet = 0 To nSheets - 1
        MsgBox "Arbeitsblatt " & PrjSheets(nSheet).Name
        Set Sheet = PrjSheets(nSheet)
    Next nSheet
End Sub
```

Funktionen:

DVisSheet GetSheetByName(String ArbeitsblattName)

Die Funktion liefert bei Erfolg das Arbeitsblatt mit dem Namen *ArbeitsblattName*.

DVisSheet GetSheetByID(String ArbeitsblattID)

Die Funktion liefert bei Erfolg das Arbeitsblatt mit der ID *ArbeitsblattID*.

DVisProcess GetProcessModelByScriptName (String ProcessName)

Die Funktion liefert bei Erfolg das Prozessmodell mit dem Namen *ProcessName*, falls das Prozessmodell mit dem Visualisierungsprojekt verwendet wird.

DVisProcess GetProcessModelByID (String ProcessID)

Die Funktion liefert bei Erfolg das Prozessmodell mit der ID *ProcessID*, falls das Prozessmodell mit dem Visualisierungsprojekt verwendet wird.

4.4 DVisProcess

Eigenschaften:

Rückgabe	Name	Zugriff	Beschreibung
String	Name	Lesen	Der Name des Prozessmodells
String	ID	Lesen	Die ID des Prozessmodells
Array	RootItems	Lesen	Liefert das Array der Prozessvariablen der obersten Ebene

Funktionen:

DVisProcessItem *GetItemByScriptName (String itemName)*

Die Funktion liefert bei Erfolg die Prozessvariable mit dem Namen *itemName*. Dazu muss zuvor ein Skriptname für die Prozessvariable im Eigenschaftsfenster der Prozessvariablen eingegeben worden sein.

DVisProcessItem *GetItemByID (String itemID)*

Die Funktion liefert bei Erfolg die Prozessvariable mit der ID *itemID*.

Beispiel: Zugriff auf eine Prozessvariable.

```
Sub Main
    Dim LogPrj As DVisProcessModel
    Set LogPrj = VisProject.GetProcessModelByScriptName("Schreibtisch.prj")
    Dim Schalter As DVisProcessItem
    Set Schalter = LogPrj.GetItemByScriptName("Schalter")
    MsgBox Schalter.AddrInfo 'gibt die Gruppenadresse des EIB-Schalters aus
End Sub
```

4.5 DVisProcessItem

DVisProcessItem beschreibt Eigenschaften von Prozessvariablen und hat folgende Eigenschaften und Funktionen:

Eigenschaften:

Rückgabe	Name	Zugriff	Beschreibung
String	Name	Lesen	Der Name der Prozessvariablen, zum Beispiel Lampe .
String	FullName	Lesen	Der volle Name der Prozessvariable, zum Beispiel der Form Erdgeschoss.Flur.Lampe .
String	ID	Lesen	Die ID der Prozessvariable
String	AddrInfo	Lesen	Eine Information über die Herkunft der Prozessvariable, z. B. eine EIB Gruppenadresse innerhalb eines EIB-Projekts.
String	ScriptName	Lesen	Der für die Prozessvariable vergebene Skriptname, falls vorhanden. Die Skriptnamen sind immer eindeutig und dienen als Abkürzung für den Zugriff.
Long	VarType	Lesen	Der Typ der Prozessvariablen.
	Für den Typ von Prozessvariablen ist die Aufzählung VisProcessVarType mit folgenden Werten definiert: VisProcessVarTypeNone = 0 'Nicht angegeben VisProcessVarTypeBinary = 1 'Binär VisProcessVarTypeAnalog = 2 'Analog VisProcessVarTypeCounter = 3 'Zähler		
Boolean	IsReadable	Lesen	Wahr, wenn die Prozessvariable gelesen werden kann
Boolean	IsWriteable	Lesen	Wahr, wenn die Prozessvariable geschrieben werden kann
Variant	Value	Lesen	Der Wert im Cache, falls vorhanden
Boolean	HasSubItems	Lesen	Gibt an, ob die Prozessvariable selbst noch weitere Prozessvariablen enthält
Array	SubItems	Lesen	Die in der Prozessvariablen enthaltenen Unterelemente .

Funktionen:

ReadValue (Options As Long)

Die Funktion fordert eine Leseoperation an. Da das Lesen asynchron erfolgt, steht das Ergebnis der Leseoperation nicht sofort zur Verfügung. Wenn ein neuer Wert für die Prozessvariable gelesen wurde, wird der Handler *OnVisProcessEvent* aufgerufen.

Options ist für zukünftige Zwecke reserviert und muss bis dahin immer den Wert 0 haben.

Boolean WriteInt (value As Integer)

Boolean WriteBool (value As Boolean)

Boolean WriteDouble (value as Double)

Diese Funktionen schreiben einen Wert in eine Prozessvariable, falls möglich (Schreibrechte,..) und geben bei Erfolg **True** zurück. Die dabei übergebenen Werte werden für die Prozessvariable intern in die passende Form gewandelt.

Beispiel: Durchlaufen aller Prozessvariablen in einem Prozessmodell.

```
Dim ProcessModel As DVisProcessModel
```

```
Sub Main
```

```
    Set ProcessModel = VisProject.GetProcessModelByScriptName("Demo.prj")
```

```
    Dim RootItems As Variant
```

```
    RootItems = ProcessModel.RootItems 'Elemente der obersten Ebene
```

```
    ShowSubItems RootItems
```

```
End Sub
```

```
Sub ShowItem(PVItem As DVisProcessItem)
```

```
    MsgBox PVItem.FullName 'Info zum Item ausgeben
```

```
End Sub
```

```
Sub ShowSubItems (Items As Variant)
```

```
    'Alle Subitems durchlaufen
```

```
    Dim nItems
```

```
    nItems = UBound(Items) + 1
```

```
    Dim nItem
```

```
    Dim Item As DVisProcessItem
```

```
    For nItem = 0 To nItems - 1
```

```
        Set Item = Items(nItem)
```

```
        ShowItem(Item)
```

```
        If Item.HasSubItems Then
```

```
            ShowSubItems(Item.SubItems)
```

```
        End If
```

```
    Next nItem
```

```
End Sub
```

4.6 DVisSheet

DVisSheet beschreibt Eigenschaften und Funktionen eines Arbeitsblatts.

Eigenschaften:

Rückgabe	Name	Zugriff	Beschreibung
String	Name	Lesen	Der Name des Arbeitsblatts.
String	ID	Lesen	Die ID des Arbeitsblatts.
Array	Planes	Lesen	Die Ebenen des Arbeitsblatts.

Funktionen:

Boolean IsOpen ()

Die Funktion gibt zurück, ob das Arbeitsblatt zum aktuellen Zeitpunkt offen ist.

Open(Options As Long)

Die Funktion öffnet das Arbeitsblatt, falls es noch nicht geöffnet ist. Ist es schon geöffnet, dann wird sein Fenster in den Vordergrund gebracht. Es ist nicht schädlich, diese Funktion aufzurufen, wenn das Arbeitsblatt schon geöffnet wurde und im Vordergrund ist.

Options ist für zukünftige Zwecke reserviert und muss den Wert 0 haben.

Close()

Schließt das Arbeitsblatt, falls es offen ist. Es ist nicht schädlich, diese Funktion aufzurufen, wenn das Arbeitsblatt schon geschlossen ist.

DVisSheetPlane GetPlaneByName(PlaneName As String)

Bei Erfolg liefert die Funktion die Ebene mit dem Namen *PlaneName*.

DVisSheetPlane GetPlaneByID(PlaneID As String)

Bei Erfolg liefert die Funktion die Ebene mit der ID *PlaneID*.

DVisSheetItem GetItemByName(ItemName As String)

Bei Erfolg liefert die Funktion das Element im Arbeitsblatt mit dem Namen *ItemName*. Der Name muss für das Element im Eigenschaftsfenster des Editors eingetragen worden sein.

DVisSheetItem GetItemByID(ItemID As String)

Bei Erfolg liefert die Funktion das Element im Arbeitsblatt mit der ID *ItemID*.

4.7 DVisSheetPlane

DVisSheetPlane beschreibt Eigenschaften und Funktionen der Ebene eines Arbeitsblatts.

Eigenschaften:

Rückgabe	Name	Zugriff	Beschreibung
String	Name	Lesen	Der Name der Ebene.
String	ID	Lesen	Die ID der Ebene.
Array	Items	Lesen	Die Elemente der Ebene.

Funktionen:

DVisSheetItem GetItemByName(Name As String)

Bei Erfolg liefert die Funktion das Element der Ebene mit dem Namen *ItemName*.
Der Name muss für das Element im Eigenschaftsfenster des Editors eingetragen worden sein.

Hinweise: *Namen von Arbeitsblattelementen sind eindeutig bezogen auf das Arbeitsblatt, nicht nur der Ebene.*

DVisSheetItem GetItemByID(ItemID As String)

Bei Erfolg liefert die Funktion das Element im Arbeitsblatt mit der ID *ItemID*.

4.8 DVisSheetItem

DVisSheetItem beschreibt Eigenschaften und Funktionen eines Elements in einem Arbeitsblatt.

Eigenschaften:

Rückgabe	Name	Zugriff	Beschreibung
String	Name	Lesen	Der Name des Elements, falls ein Name für das Element festgelegt wurde.
String	ID	Lesen	Die ID des Elements.
DVisSheetPlane	Plane	Lesen	Die Ebene, in der das Element vorkommt.
Long	Type	Lesen	Der Typ des Elements
	Für die Typen von Arbeitsblattelementen gibt es die Aufzählung VisSheetItemType :		
	VisSheetItemTypeUnknown = 0 'Kein Typ vorhanden		
	VisSheetItemTypeOLE = 1 'OLE-Objekt		
	'Beachte: Auch die Parameteranzeigen sind OLE-Objekte		
	VisSheetItemTypeRect = 2 'Rechtecktyp des Editors		
	VisSheetItemTypeText = 3 'Texttyp des Editors		
	VisSheetItemTypeLink = 4 'Link zu einem Arbeitsblatt		
Boolean	IsVisible	Lesen	Liefert die Information, ob das Element sichtbar ist.
DVisSheetOleItem	AsOleItem	Lesen	Für ein Arbeitsblattelement vom Typ VisSheetItemTypeOLE wird ein Objekt mit weiteren Funktionen geliefert.

Funktionen:

SetVisible(bVisible As Boolean)

Das Arbeitsblattelement wird sichtbar / unsichtbar geschaltet. Wenn das Element bereits sichtbar bzw. unsichtbar ist, dann hat die Funktion keine weitere Wirkung.

DVisRect GetRect()

Liefert das Anzeigerechteck des Arbeitsblattelements als separates Objekt. Das Rechteck ist eine Kopie. Soll das Rechteck des Arbeitsblattelements geändert werden, dann ist ein Aufruf von **SetRect** erforderlich.

SetRect(rect As DVisRect)

Setzt das Anzeigerechteck des Arbeitsblattelements.

Beispiel: Durchlaufen aller Arbeitsblattelemente in einem Visualisierungsprojekt.

Sub Main

```

    Dim PrjSheets
    PrjSheets = VisProject.Sheets
    Dim nSheets
    nSheets = UBound(PrjSheets) + 1
    Dim Sheet As DVisSheet
    Dim nSheet
    For nSheet = 0 To nSheets - 1
        Set Sheet = PrjSheets(nSheet)
        ShowSheet(Sheet)
    Next nSheet
End Sub
Sub ShowSheet(Sheet As DVisSheet)
    MsgBox "Arbeitsblatt " & Sheet.Name
    Sheet.Open 0
    Dim SheetPlanes
    SheetPlanes = Sheet.Planes
    Dim PlanesCount
    PlanesCount = UBound(SheetPlanes) + 1
    Dim nPlane
    Dim Plane As DVisSheetPlane
    For nPlane = 0 To PlanesCount - 1
        Set Plane = SheetPlanes(nPlane)
        ShowPlane(Plane)
    Next nPlane
End Sub
Sub ShowPlane(Plane As DVisSheetPlane)
    MsgBox "Ebene " & Plane.Name
    Dim PlaneItems
    PlaneItems = Plane.Items
    Dim ItemsCount
    ItemsCount = UBound(PlaneItems) + 1
    Dim nItem
    Dim SheetItem As DVisSheetItem
    For nItem = 0 To ItemsCount - 1
        Set SheetItem = PlaneItems(nItem)
        ShowItem(SheetItem)
    Next nItem
End Sub
Sub ShowItem(Item As DVisSheetItem)
    MsgBox "Element " & Item.ID & " Typ: " & Item.Type
End Sub

```

4.9 DVisSheetOleItem

DVisSheetOleItem liefert erweiterte Eigenschaften und Funktionen für Arbeitsblattelemente, die vom Typ *VisSheetItemTypeOLE* sind. Das trifft zum Beispiel für die Parameteranzeigen zu. Die Verwendung dieser Eigenschaften setzt Kenntnisse des speziellen OLE- Objekts voraus und Grundkenntnisse der WINDOWS OLE- Mechanismen.

Eigenschaften:

Rückgabe	Name	Zugriff	Beschreibung
Object	Object	Lesen	Das IDispatch-Interface des Objekts, falls vorhanden.
Boolean	IsRunning	Lesen	Das OLE- Objekt ist im ‚Running‘-Zustand
Array	Verbs	Lesen	Liefert ein Array von Beschreibungen (<i>DVisOleVerb</i>) zu den OLE- Verben

Funktionen:

Boolean DoVerb(nVerb As Long)

Führt, falls möglich, das OLE- Verb mit der Nummer *nVerb* aus.

Boolean Run()

Versetzt das OLE- Objekt in den ‚Running‘- Zustand, falls möglich.

Close()

Beenden den ‚Running‘- Zustand des OLE- Objekts.

4.10 DVisOleVerb

DVisOleVerb dient der Beschreibung von OLE- Verben.

Eigenschaften:

Rückgabe	Name	Zugriff	Beschreibung
String	Name	Lesen	Der Name des Verbs
Long	Number	Lesen	Die Nummer des Verbs

4.11 DVisRect

DVisRect beschreibt ein Rechteck und wird verwendet, um zu Arbeitsblattelementen das Anzeigerechteck zu Lesen oder zu Ändern. Die Angaben sind jeweils in Pixel.

Eigenschaften:

Rückgabe	Name	Zugriff	Beschreibung
Long	Left	Schreiben/Lesen	Die linke Grenze
Long	Right	Schreiben/Lesen	Die rechte Grenze
Long	Top	Schreiben/Lesen	Die obere Grenze
Long	Bottom	Schreiben/Lesen	Die untere Grenze

Funktionen:

Long GetHeight()

Liefert die Höhe des Rechtecks.

Long GetWidth()

Liefert die Breite des Rechtecks.

SetRect(Left As Long, Top As Long, Right As Long, Bottom As Long)

Setzt alle Grenzen des Rechtecks.

5 Tipps

5.1 Logikfunktionen

🟡 Verwenden Sie Timer oder verwenden Sie den Handler *OnVisProcessEvent*.

Beispiel: Schalten eines Ausgangs nach Vergleich zweier Temperaturen

Vorbereitung:

Vergeben Sie Skriptnamen an die Prozessvariablen (im Eigenschaftsfenster zur Prozessvariablen), zum Beispiel *Schalter*, *Temp1*, *Temp2*. Sei *Projekt.prj* der Name des Prozessmodells.

```
Dim Process As DVisProcessModel
Dim Schalter As DVisProcessItem
Dim Temp1 As DVisProcessItem
Dim Temp2 As DVisProcessItem
```

Private Sub Main

```
Set Process = VisProject.GetProcessModelByScriptName("Projekt.prj")
```

```
Set Schalter = Process.GetItemByScriptName("Schalter")
```

```
Set Temp1 = Process.GetItemByScriptName("Temp1")
```

```
Set Temp2 = Process.GetItemByScriptName("Temp2")
```

End Sub

```
Sub OnVisProcessEvent(ProcessModel As DVisProcessModel, _
    ProcessItem As DVisProcessItem)
```

```
    If (0 = StrComp(ProcessItem.ScriptName, "Temp1")) _
```

```
        Or (0 = StrComp(ProcessItem.ScriptName, "Temp2")) Then
```

```
        'Entweder Temp1 oder Temp2 haben sich geändert, also vergleiche jetzt
```

```
        If Temp1.Value > Temp2.Value Then
```

```
            Schalter.WriteBool(True)
```

```
        End If
```

```
    End If
```

End Sub

5.2 Wie kann ich Zeitfunktionen realisieren

- 🟡 Verwenden Sie Timer.

5.3 Debugging

- 🟡 Im Player ist normalerweise kein Fenster für das Skripting sichtbar. Wenn Sie den Player aber mit der Option `-VISDEBUG` starten, wird ein Fenster für das Skripting sichtbar gemacht.
- 🟡 Wenn Sie einen Handler debuggen wollen, dann setzen Sie den Haltepunkt in den Handler selbst, nicht in die vom Handler aufgerufen Funktionen.

5.4 Verzeichnisse

- 🟡 Die Skripts eines Visualisierungsprojekts liegen in dem `scripts`- Unterverzeichnis des Projekts. Sie können Skripts frei kopieren.
- 🟡 Wenn `autorun.visscript` gelöscht wurde, wird die Datei von Player oder Editor wieder automatisch erzeugt.

5.5 Sonstiges

- 🟡 Senden Sie Telegramme nicht schneller als der technische Prozess verkraften kann. Denken Sie daran, dass auch andere Geräte senden.
- 🟡 Seien Sie vorsichtig mit der Verwendung von `Debug.Print`, wenn das Skript automatisch läuft. Das Schnellfenster des Skripts löscht keine Meldungen, so dass Sie auf die Dauer Speicher verlieren, auch wenn das Fenster nicht sichtbar ist. Der Player erzeugt das Fenster auch unsichtbar!
- 🟡 Verwenden Sie nicht zu viele Timer (ihre Anzahl in WINDOWS ist beschränkt) und erwarten Sie keine allzu große Genauigkeit.