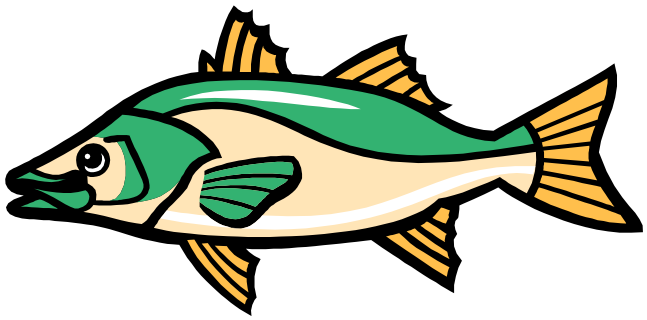
Einführung in die Programmierung mit

vbaFish40

Ulrich Müller



Inhaltsverzeichnis

Anhang 48

Ulrich Müller, D-33100 Paderborn, Lange Wenne 18. Fon 05251/56873, Fax 05251/55709
eMail : UM@ftcomputing.de
HomePage : www.ftcomputing.de

Das Programm wurde mit dem Entwicklungssystem Visual Studio der Firma Microsoft in Visual Basic Version 6 SP5 erstellt.
Die hier genutzte umFish40.DLL wurde mit VC++ 6.0 des Visual Studio erstellt.
Das Copyright für das verwendete SB6Ent.OCX liegt bei der Firma
Polar Egineering and Consulting.
Dieses Dokument wurde mit MS Office 97 WinWord erstellt und mit dem Acobe Acrobat 5.0 in das PDF-Format konvertiert.
Die Help-Datei FishFa40AX.HLP wurde ebenfalls mit WinWord erstellt und mit DocToHelp 2000 in das HLP-Format gewandelt.
Das Setup Programm wurde mit Inno Setup v2.0 von Jordan Russell erstellt.
Die verwendeten Bilder wurden – sofern selbst erstellt – entweder mit dem
HP Scanjet 3400C oder der Olympus Kamera Camedia C-1400 L erstellt und mit dem Microsoft Image Composer v1.5 nachbearbeitet.
Die oben angeführte HomePage wurde mit MS FrontPage 2000 erstellt.

Die umFish40.DLL nutzt die FtLib_Static_LIBCMT_Release.lib der Firma fischertechnik.

Dokumentname : vbaFish40.doc. Druckdatum : 10.06.2005
Titelbild : Einfügen | Grafik | AusDatei | Office | Fish12.WMF

Übersicht

Allgemeines

vbaFish ist eine IDE (Integrated Development Environment) für ftComputing auf Basis der Sprache VBA (Visual Basic for Applications und des ActiveX FishFa40AX.DLL. Dabei wurden die Sprachelemente von FishFa40AX.DLL nahtlos in die Sprache VBA integriert.). VBA wird in der Realisierung als Sax Basic der Firma Polar Engineering and Consulting in Form des SB6Ent.OCX eingesetzt.

vbaFish wurde bewußt einfach gestaltet um Programmier-Anfängern ein einfaches und, in allen seinen Komponenten, *kostenloses* Werkzeug zur Verfügung zu stellen, mit dem im Selbststudium der Einstieg in die Welt der Programmierens gefunden werden kann.

Deshalb ist der Abschnitt "Einführung in die Programmierung" der wesentliche Teil dieses Buchs. Man arbeitet ihn am besten in der vorgegebenen Reihenfolge durch.

vbaFish ist geeignet für Programme im Umfang von ein paar Zeilen bis zu ein paar Seiten. Die Möglichkeiten der Kommunikation mit dem Bediener sind bei VBA beschränkt, aber in diesem Rahmen in Verbindung mit der IDE voll ausreichend. Ist man aus diesem Rahmen herausgewachsen, sollte man auf "größere" Sprachen der Visual Basic Sprachfamilie umsteigen. Das ist problemlos möglich. Siehe Anhang.

Neuerungen

Der Hauptunterschied zu vbaFish30 ist die Unterstützung der ROBO Interfaces (an USB, COM und über Datalink). Die Unterstützung des Unversal Interfaces (an LPT) ist dafür entfallen.

Der Befehlsumfang ist weitgehend gleich geblieben. Beim ROBO Interface sind Befehle für die Spannungseingänge (GetVoltage) und den Betrieb mit dem IR-Sender (GetIRKey, FinishIR, WaitForInputIR) hinzugekommen.

Das Menü Interface Daten für den neuen Gegebenheiten angepaßt.

Installation

vbaFish wird in Form eines Setup-Programmes : vbaFish30Setup.EXE ausgeliefert. Die Installation erfolgt weitgehend automatisch nach den üblichen Regeln. Zusätzlich ist der Acrobat Reader ab Version 4 erforderlich. Bei Bedarf kann er von www.adobe.de/products/acrobat/readstep.html (Stand Mai 2003) heruntergeladen werden.

vbaFish läuft unter 32bit Windows ab Win 98.
Getestet wurde mit Windows NT, 2000 und XP.
Die mitgelieferten Systemfiles haben den Stand von Windows 2000. Bei Installationen auf Windows XP kann deswegen gefragt werden, ob auf Win XP bereits vorhandene Systemfiles gleichen Namens erhalten werden sollen, das sollte mit JA quittiert werden.

Aktuelle Firmwarestände für das ROBO Interface und USB-Treiber bekommt man bei http://www.fischertechnik.de/computing/software.html. Das Überspielen der Firmware auf das ROBO Interface geschieht am besten mit ROBO Pro (Demoversion ebenda), die USB-Treiber werden schlicht in ein passendes Verzeichnis (z.B. das von vbaFish40) kopiert. Das System erkennt dann beim Anstecken eines ROBO Interface die Lage und installiert dann selbst.

Das Setup-Programm vbaFish40Setup.EXE enthält alles was man zum Arbeiten mit vbaFish40 benötigt. Ein MS VBA-System (MS-Office) muß nicht installiert sein.
vbaFish40.EXE wird in einen wählbaren Pfad {app} installiert (default : C:\Programme\ftComputing).

{app}	ftComputing.vbaFish40.TXT vbaFish40.TXT (ReadMe-File) ftComputing.ico FishFa40AX.DLL (ActiveX mit Klasse FishFace, wird registriert) FishFa40AXRef.HLP/CNT (Helpfile dazu)
{sys}	umFish40.DLL Sax-VBA-Dateien und VB6-Dateien
{app}\vbaFish40	vbaFish40.EXE (Das Programm) vbaFish40PDF (Tutorial und Referenz)

{app}\vbaFish40\Samples : Beispielprogramme

Im Startmenü wird eine Gruppe ftComputing angelegt, die Einträge für :
Die vbaFish40 IDE
Das Liesmich
Das Handbuch

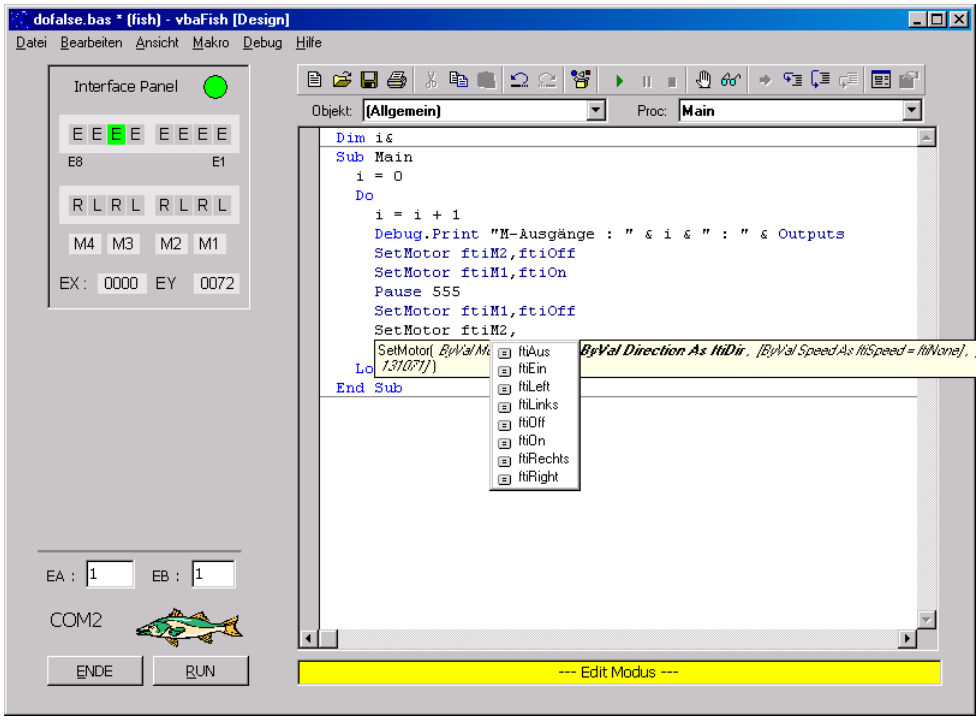
enthält.

Auf dem Desktop wird ein Icon für die vbaFish40 IDE angelegt.

Die Deinstallation kann über START | Einstellungen | Systemsteuerung | Software (Win2000) erfolgen.

IDE – Integrated Development Environment

Übersicht



IDE im Edit Modus

vbaFish ist eine Integrierte Programmierumgebung (IDE), die zwei wesentliche Betriebsarten - Editier-Modus und Run-Modus – kennt.

Die IDE gliedert sich in folgende Teile :

1. **Bedienung der Gesamt IDE** : Buttons links unter dem Strich. Die Funktion der Buttons wechselt in Abhängigkeit von den Betriebszuständen.
2. **Anzeige des aktuell eingestellten Interfaces** (hier COM2). Die Einstellung selber erfolgt über das Menü Extras | Interface-Daten.
3. **Kommunikation mit dem Modell über das Interface** : Interface Panel
   Im Edit-Modus kann das Interface manuell mit der Maus bedient werden. (Maus-Klick + Strg-Taste schalten den entsprechenden M-Ausgang dauerhaft).
4. **Parameter für die Anwendung** : Die Felder EA und EB. Der Zugriff aus der Anwendung erfolgt über die gleichnamigen Funktionen EA und EB.
5. **Anzeige und Editieren der Anwendung** : Editier-Feld mit dem Source-Programm. Die Eingabe von Programmtext wird durch die Funktionen "Intellisense" (Anzeige der möglichen Parameter einer Funktione) und Auswahllisten (Anzeige der möglichen Werteeingabe) sowie durch Syntax-Highlighting unterstützt.
   Auf dem linken, grauen, Balken können Stop-Adressen gesetzt werden.
   Über Rechts-Klicks kann ein Pop-Menü erreicht werden.
   Über die F1-Taste kann – wie üblich – zu den einzelnen Textelementen Hilfe abgerufen werden.
6. **Protokoll-Fenster** : Im Run-Modus erscheint über dem Editier-Feld ein zusätzliches Fenster mit vier über Tabs schaltbaren Felder zur Protokollierung des Programmablaufs (Schnell - Direktfenster) und weitere Funktionen. Dort können auch direkte Eingaben

gemacht werden. Das Schnell-Feld kann auch über das Menü Ansicht dauerhaft geschaltet werden.

7. **Toobar** : über dem Editier-Feld befindet sich ein Toolbar mit den wichtigsten Funktionen für Editier- und Run-Modus. z.B. Open / Save File, Run – Pause – Stop Programm, setzen von Stop-Adressen und Einzelschritt.
8. **Statuszeile** : zur Anzeige des Betriebszustandes (Farbcode und Text) sowie der Ausgabe von Fehlermeldungen und Ausgaben aus der Anwendung (Befehl : PrintStatus).
9. **Menü-Zeile** : Die Menü-Zeile enthält alle Funktionen des Toolbars und eine größere Anzahl weiterer wie z.B. die überlichen Bearbeiten- und Ansicht-Funktionen. Die Menü-Zeile wechselt mit dem Fokus auf das Editierfeld und dem Fokus auf andere Felder.
10. **Dokumentation** : Neben der Hilfe über die F1-Taste stehen über das Hilfe-Menü weitere Dokumentationen zur Verfügung. Insbesondere wird zusammen mit vbaFish dieses Handbuch ausgeliefert.

Bedienung der IDE im Edit-Modus

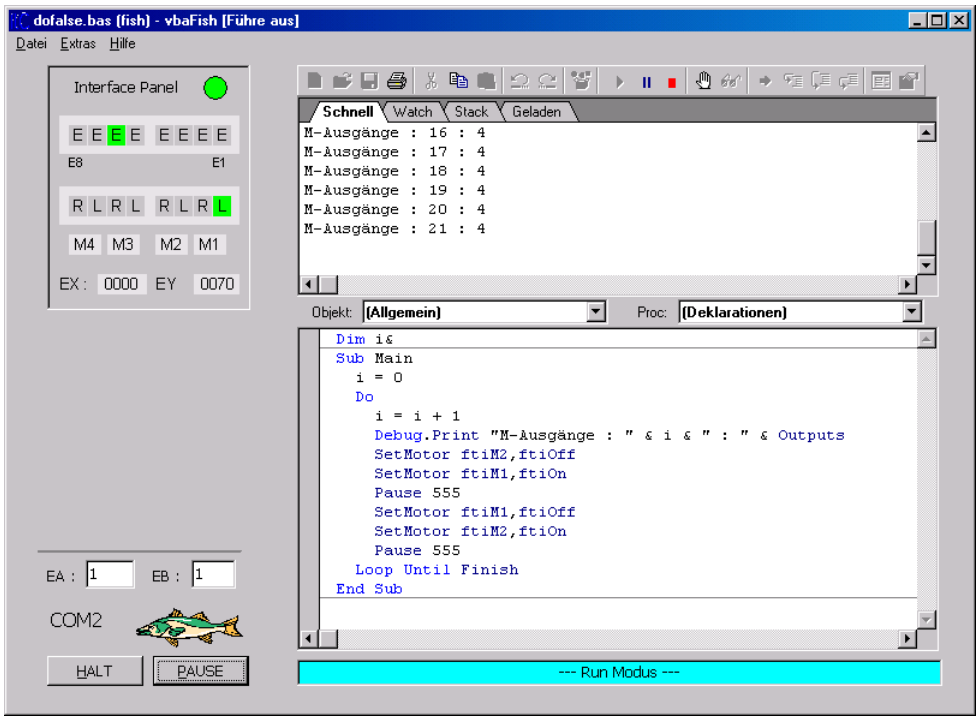
Der Edit-Modus wird aus dem **Start-Modus** durch Klicken des START-Buttons erreicht. Vorher ist ggf. noch über Menü Extras | InterfaceDaten das aktuelle Interface einzustellen. Die Auswahl NULL erlaubt ein Arbeiten im Edit-Modus ohne ein Interface.

Eine erfolgreiche Interface-Verbindung wird durch den grünen Button rechts neben dem Wort Interface Panel angezeigt. Ab jetzt kann das Interface durch Maus-Klick auf die L / R Felder bedient werden. Hinweis : Maus-Klick bei gedrückter Strg-Taste schaltet die M-Ausgänge dauerhaft, Ausschalten durch Klick auf Mn.

Der Edit-Modus wird durch "--- Edit Modus ---" auf gelben Hintergrund in der Statuszeile angezeigt.

Das Programm kann über den ENDE-Button verlassen werden.

Bedienung der IDE im Run-Modus



IDE im Run-Modus

Der Run-Modus wird aus dem Edit-Modus durch Klicken des RUN-Buttons erreicht. Das Programm startet dann. Dabei wechselt die Beschriftung des RUN-Buttons in Pause. Alternativ kann auch über das grüne Dreieck-Symbol des Toolbars gestartet werden.

Die Anwendung kann durch Klick auf den Pause-Button unterbrochen werden, die M-Ausgänge werden dabei vorrübergehend abgeschaltet. Die Beschriftung des Pause-Buttons wechselt in RUN, durch einen Klick kann das Programm wieder fortgesetzt werden.

Durch Klicken auf den HALT-Button wird die Anwendung abgebrochen. Die M-Ausgänge werden abgeschaltet. Es wird dann in den Edit-Modus gewechselt.

Alternativen : Toobar mit dem blauen Pause-Symbol und dem roten Ende-Symbol oder die Menü-Zeile- Ebenfalls möglich ist der Abbruch über die Esc-Taste.

Über die F9-Taste (oder Toolbar, Menü, Maus-Klick auf den "grauen Balken") können Haltepunkte markiert werden. Bei Erreichen : Wechsel in Pause-Modus.

Über die F8-Taste (oder Toolbar, Menü) und weitere kann die Anwendung im Einzelschritt ausgeführt werden. Nach jedem Schritt : Pause-Modus.

Bei Anwendung-Halt im Run-Modus wird in der Statuszeile "--- Pause Modus ---" auf Magenta Hintergrund angezeigt. Die M-Ausgänge sind vorrübergehend abgeschaltet.

Wenn die Anwendung auf "natürlichem" Weg (Erreichen End Sub, Exit Sub) zum Ende kommt, wird automatisch in den Edit-Modus gewechselt. Das Protokoll-Fenster wird dabei abgeschaltet. Will man es weiter offenhalten, so kann man das über Menü Ansicht | Immer teilen tun. Alternativ Stop Adresse auf End Sub.

Im Run-Modus wird zusätzlich das Protokoll-Fenster angezeigt. Im Schnell-Feld können über Debug.Print von der Anwendung Ausgaben vorgenommen werden. Wenn sich die Anwendung im Pause-Modus befindet, können aber auch Eingaben gemacht werden. Mit ? Variable wird z.B. der Inhalt einer Variablen angezeigt. Es können aber auch genauso Befehle ausgeführt werden.

Variablen-Inhalte werden aber auch anzeigt, wenn die Maus über der entsprechenden Variablen steht.

Die weiteren Felder des Protokollfensters sind z.Zt. noch nicht so interessant.

Unterstütze Interfaces

Unterstützt werden die Interfaces der ROBO Reihe mit Anschluß über USB / COM / RF Datalink ggf. mit biszu drei Extensions und das Intelligent Interface ggf. mit Extension.

Die Anschlußbezeichnungen der Interfaces unterscheiden sich etwas, es werden die der ROBO Interfaces verwendet :
- I-Eingänge, beim Intelligent Interface E-Eingänge
– AX / AY, beim Intelligent Interface EX / EY
– A1, A2 und AV sind beim Intelligent Interface nicht vorhanden, ebenso fehlt der IR-Eingang
– D1 / D2 werden z.Zt. nicht unterstützt.

Ein Betrieb des ROBO Interfaces über das ROBO RF Datalink ist bei eingebauter RF-Platine möglich.

Anschluß des Interfaces

Anschluß je nach Gerät an USB bzw. COM, Stromversorgung 9V mit einem Netzteil, das min. 500mA, besser 1000 mA liefert, das Netzteil sollte stabilisiert sein. Die fischertechnik Netzteile erfüllen meist diese Bedingungen. Hinweis : alte (graue) Motoren können auch mit 9V am Intelligent Interface laufen.

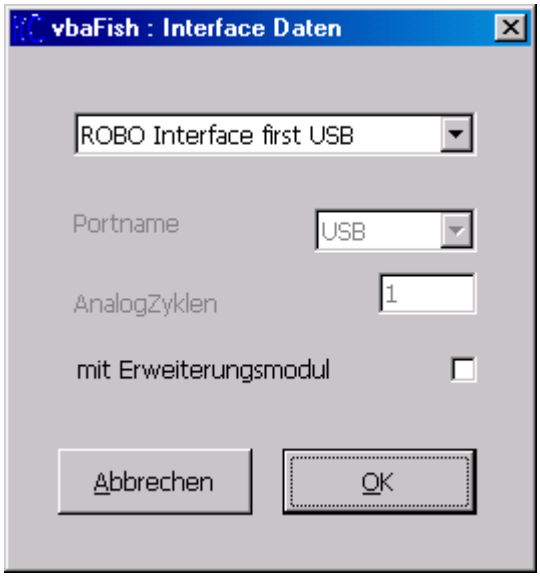
Der Download Mode des ROBO bzw. Intelligent Interface wird nicht unterstützt.

Die Änderung des aktuellen Interfaces (Default COM1) erfolgt über das Menü | Extras | InterfaceDaten gleich nach Start der Anwendung. Hier können auch weitere Interface Parameter eingestellt werden. Anschließend wird über den START-Button in den Edit-Modus

gewechselt, das ausgewählte Interface muß dann angeschlossen und betriebsbereit sein. Wird ein Interface NULL gewählt, kann ohne ein angeschlossenes Interface in den Edit-Modus gewechselt werden.

Einstellen der Interfacewerte

Die Werte für den Interfacebetrieb werden über das Menü Extras | Optionen der IDE eingestellt.



Man sollte die Interfacewerte in der Reihenfolge der Form einstellen, also zuerst den Interfacetyp wählen. Damit verbunden werden für die weiteren Werte Standard-Vorgaben gemacht.

Zu einer Zeit kann nur ein Interface – ggf. mit bis zu vier Extensions – betrieben werden. Im Interface Panel wird allerdings – nach Markieren von "mit Erweiterungsmodul" – nur das erste Erweiterungsmodul angezeigt.

- ROBO Interface first USB
  das erste an USB gefundene ROBO Interface (Main Interface, Extension, RF Datalink)
- Intelligent Interface
- Intelligent Interface mit Extension
- ROBO Interface IIM (im Intelligent Interface Mode)
- ROBO Interface COM (Betrieb über COM-Anschluß)

Bei Anschluß über COM muß der COM-Name angegeben werden. Sollen beim Intelligent Interface auch die Analog-Eingänge abgefragt werden, so ist das extra anzugeben (ROBO nicht). Angegeben werden die Anzahl der Abfragezyklen nach denen die Analog-Eingänge ausgelesen werden sollen (typisch 5 – 10, 0 : keine Abfrage).

"mit Erweiterungsmodul" regelt einmal die InterfacePanel Anzeige (mit oder ohne Extension), sagt beim ROBO Interface aber nichts über die Anzahl der tatsächlich angeschlossenen Extension aus. Beim Intelligent Interface dagegen schon.

Hilfe – Dokumentation

VBA

Die mitgelieferte englische Hilfe-Datei bildet eine Referenz aller VBA-Befehle. Aufruf über F1 Ebenso über das Hilfe-Menü und das Windows Start-Menü.

Im Anhang dieses Handbuchs gibt es eine deutsche Referenz der im Kapitel "Einführung in die Programmierung" verwendeten VBA-Befehle. Die Referenzen im genannten Kapitel beziehen sich darauf.

Eine weitere deutsche Hilfe zu VBA ist auf Rechner mit installiertem MS Office verfügbar : die Dateien VEDEUI3.HLP und VEDEUI3.CNT. Sie sind meist im Verzeichnis C:\Programme\Gemeinsame Dateien\Microsoft Shared\VBA zu finden. Sie werden aber nicht immer mit MS Office installiert, ggf. ist hier nachzuinstallieren (einfach MS Office CD einlegen, Ändern der Installation wählen und entsprechen ankreuzen). Die Angaben beziehen sich auf MS Office 97, bei anderen Versionen könnte etwas Suchen angesagt sein.

Wers lieber gedruckt hätte, kann sich nach einem VBA-Buch umsehen, die gibt es allerdings meist nur in Verbindung mit der WinWord oder Excel-Programmierung. Wenn man in dieser Richtung (z.B. beruflich) ohnehin etwas vor hatte, ist das eine gute Wahl. Die allgemeinen Kapitel zum Sprachkern lassen sich auch hier gut gebrauchen.

FishFace

Zu FishFace gibt es dieses Handbuch im PDF-Format. Es besteht aus einem **Übersichtsteil**, einer **Einführung in die Programmierung** und einem **Referenzteil** für FishFa40AX.DLL und die wichtigsten VBA-Befehle. Aufruf über das Hilfe Menü und das Windows-Start Menü.

Die intern verwendete FishFa40AX.DLL für die Interfacebefehle (GetInput, SetMotor ....) nutzt eine eigene FishFa40AXRef.HLP-Datei die über F1 aufgerufen wird.

Einführung in die Programmierung

Allgemeines

Im diesem Abschnitt wird eine einfache Einführung in die Programmierung gegeben. Sie ist für Programmieranfänger mit einiger Windows-Erfahrung, aber ohne Programmierkenntnisse gedacht. Die Einführung erfolgt anhand von praktischen Beispielen, auf eine theoretische Unterlegung wird verzichtet. Ebenso auf eine komplette Darstellung der Sprache VBA. Die Einführung erfolgt bewußt auf der Basis von "prozeduralen" Elementen um die Einstiegsschwelle niedrig zu halten. Eine Nutzung der OO-Elemente von VBA ist aber problemlos möglich.

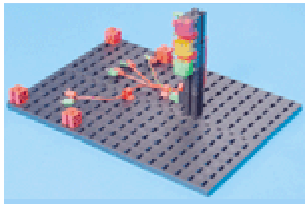
Die Kapitel bauen aufeinander auf und nehmen im Schwierigkeitsgrad zu, sie sollten deswegen nacheinander durchgearbeitet werden.

Bei weitergehendem Interesse kann nahtlos zu der in ausreichendem Maße vorhandenen Computerliteratur gegegriffen werden.

Die Programmbeispiele beziehen sich alle auf Modelle des fischertechnik Kastens "Computing Starter" No. 16 553. Wenn man bereits einige fischertechnik Teile (auch "graue") einschl. fischertechnik Interface besitzt, kann man die Modelle auch leicht ohne den Kasten nachbauen, dann wird empfohlen die "Computing Starter – Bauanleitung" No. 30 434, Preis 7,70 € bei www.knobloch-gmbh.de zu kaufen. Dort sind auch evtl. fehlende Teile erhältlich.

Erste Befehle

Lampen



Lampen an M1 – M3
grün – gelb – rot

```
SetMotor ftiM1, ftiEin
Pause    1000
SetMotor ftiM2, ftiEin
Pause    1000
SetMotor ftiM3, ftiEin
Pause    1000
```

1. Aufbau des Modells mit den Lampen an M1 – M3 des Interfaces.
2. Anschluß des Interfaces an den Rechner, Netzteil anschließen
3. Aufruf von vbaFish, den Interface-Anschluß (COM1 ...) kontrollieren, START Klicken. Der Punkt neben dem Text Interface Panel muß grün werden.
4. Am Interface Panel der Reihe nach mit der Maus auf die Ls klicken. Die entsprechende Lampe muß leuchten.

5. Die oben angegebenen Befehle in das Editier-Feld zwischen Sub Main und End Main eingeben.Nochmal kontrollieren.
6. RUN klicken.
   Wenn denn doch etwas falsch eingegeben wurde, kommt eine entsprechende Fehlermeldung und die "schuldige" Zeile wird markiert – korrigieren – RUN

Was passiert : die Lampen gehen - eine nach der anderen – im Abstand von einer Sekunde an.

Verwendet wurden dazu zwei **Befehle** `SetMotor` zum Einschalten der Lampen und `Pause` zum Anhalten des Programms.
SetMotor hat zwei **Parameter** die **symbolischen Konstanten** `ftiM1` und `ftiEin`. Der erste steht für den Anschluß (M1 ...), der zweite für die Aktion (Ein, Aus, Links/Rechts drehen). Bei Lampen reicht ein schlichtes `ftiEin`.
Pause hat als Parameter eine **numerische Konstante** (eine Zahl), die angibt, wie lange das Programm anzuhalten ist und zwar in Millisekunden. Die `1000` hier steht also für 1 Sekunde halten.
Das wars denn auch schon.

FishFace : Eine genaue Beschreibung der Befehle ist im Referenzteil zu finden.

Schleife

Das mit den Lampen war ja ganz schön, aber auch schön schnell zu Ende. Man sollte das in einer Schleife (Schweizer und Österreicher : Schlaufe) wiederholen. Dafür gibt es den Befehl Do .... Loop Finish
mit dem man eine Folge von anderen Befehlen "einrahmen" kann um sie zu wiederholen :

```
Sub Main
  Do
    SetMotor ftiM1, ftiEin
    Pause    1000
    SetMotor ftiM2, ftiEin
    Pause    1000
    SetMotor ftiM3, ftiEin
    Pause    1000
    ClearMotors
    Pause    1000
  Loop Until Finish
End Sub
```

Die Schleife läuft solange (**Until**), bis die Finish-**Bedingung** True (wahr) wird, d.h. bis in der IDE auf den HALT-Button oder auf der Tastatur auf die Esc-Taste gedrückt wird, man könnte auch bei Finish noch einen Taster angeben : `Finish(ftiI8)`, dann würde die Schleife auch wenn der Eingang I8 am Interface True ist (weil z.B. ein Taster gedrückt wurde) abgebrochen.
Anstelle von Finish können auch andere Bedingungen angegeben werden ... das kommt später.

Wenn man sich den Code (die Befehle) genau ansieht, wird man zwei zusätzliche entdecken. Pause ist bekannt, `ClearMotors` löscht alle M-Ausgänge (M1 – M4). Wenn man sie wegläßt, rührt sich ab dem zweiten Durchlauf gar nichts mehr in der Schleife, alle Lampen werden ständig wieder eingeschaltet, obwohl sie ja schon an sind. Mit ClearMotors ändert sich das und damit mans auch merkt die Pause.

VBA : Do..Loop-Anweisung
FishFace Referenz : Finish

Schönheit - Lesbarkeit

```
' --- Ampel1.ftC : Eine einfache Ampel ----------------
Option Explicit
Const mGruen = 1, mGelb = 2, mRot = 3
Const cLangePause = 1000, cKurzePause = 300

Sub Main
  Do
    SetMotor mGruen, ftiEin
    Pause    cLangePause
    SetMotor mGruen, ftiAus
    SetMotor mGelb,  ftiEin
    Pause    cKurzePause
    SetMotor mGelb,  ftiAus
    SetMotor mRot,   ftiEin
    Pause    cLangePause
    SetMotor mGelb,  ftiEin
    Pause    cKurzePause
    SetMotor mRot,   ftiAus
    SetMotor mGelb,  ftiAus
  Loop Until Finish
End Sub
```

Man kann anstelle der allgemeinen symbolischen Konstanten, die die Anschlüsse am Interface bezeichnen, eigene Konstanten einführen, die die am Interface angeschlossenen Geräte bezeichnen. Hier sind das die verschiedenfarbigen Lampen mGruen, mGelb, mRot. Und wenn man schon dabei ist auch noch cLangePause und cKurzePause für den Pause Parameter. So kann man leicht die Einschaltdauer zentral verändern. Die kleinen Buchstaben vor den Konstanten stehen für die Bedeutung der Konstanten (m = M-Ausgang des Interfaces, c = allgemeine Konstante, e stände dann für die E-Eingänge). Das ist eine verbreitete Sitte, deren jeweilige Form mit großem Nachdruck vom Anwender vertreten wird, diese Form ist eine Marotte des Autors.

Groß- und Kleinschreibung ist ebenfalls eine viel "diskutierte" Möglichkeit, den Code lesbarer zu machen. Hier wird konsequent die Kamel-Schreibweise (mit Höckern durch weitere Großbuchstaben im Wort) angewendet. VBA schluckt jede Schreibweise (ist nicht case-sensitiv). Man sollte aber bei einer einheitlichen Schreibweise bleiben.

Üblicherweise wird der Code in der Schleife – ebenfalls aus Gründen der besseren Lesbarkeit – eingerückt.

Zusätzlich sind noch **Kommentare** möglich. Sie beginnen mit einem Hochkomma ('), der Rest der Zeile wird dann von VBA nicht mehr beachtet. Hier wurde ein Kommentar als Programmüberschrift verwendet. Man kann auch Kommentar auf den Rest einer Befehlszeile schreiben.

VBA ist eine zeilenorientierte Programmiersprache. D.h. Befehle mitsamt ihren Parametern müssen auf einer Zeile stehen. Ggf. kann eine Fortsetzungszeile eingeführt werden : am Zeilenende ein Leerzeichen (blank) und einen Underscore (_) und dann weiter auf der nächsten Zeile. Andere Sprachen verwenden ein Semikolon (;) zur Begrenzung eines Befehls. Beides hat seine Vor- und Nachteile.

VBA : Const-Anweisung

Variable

```
' --- Ampel1.ftC : Eine einfache Ampel ----------------
Option Explicit
Const mGruen = 1, mGelb = 2, mRot = 3
Dim LangePause&, KurzePause&, Runde&

Sub Main
  LangePause = 1000*EA
  KurzePause = LangePause/4
  Do
    SetMotor mGruen, ftiEin
    Pause    LangePause
    SetMotor mGruen, ftiAus
    SetMotor mGelb,  ftiEin
    Pause    KurzePause
    SetMotor mGelb,  ftiAus
    SetMotor mRot,   ftiEin
    Pause    LangePause
    SetMotor mGelb,  ftiEin
    Pause    KurzePause
    SetMotor mRot,   ftiAus
    SetMotor mGelb,  ftiAus
  Loop Until Finish
End Sub
```

Konstanten enthalten einen festen Wert, Variable können einen veränderlichen (variablen) Wert enthalten. Durch Änderung von `Const cLangePause = 1000` des vorhergehenden Beispiels in `Dim LangePause&` erhält man eine Variable der man noch einen Anfangswert zuweisen muß : `LangePause = 1000`, (standard ist 0) wie gehabt als Konstante. Der Kringel (&) hinter dem Namen steht für Dim LangePause As Long, die Angabe um welchen Typ Variable (hier Ganzzahl 32bit) es sich handelt. Der Vorteil von Variablen ist, dass man diesen Wert wieder ändern kann, das auch mehr oder weniger kompliziert durch **Ausdrücke** : `KurzePause = LangePause / 4`. Der Ausdruck (LangePause / 4) wird hier einer weiteren Variablen (KurzePause) **zugewiesen** d.h. der Wert der KurzePause beträgt ¼ der LangePause. Man kann natürlich noch viel komplexere Ausdrücke bilden. U.a. kann man alle **Operatoren** (+, -, *, /, ()) der Arithmetik einsetzen.
In Ausdrücken können Variablen, Konstante und Funktionen (kommt später) eingesetzt werden. Im Beispiel oben waren es die Variable LangePause und die Konstante 4.
(50 + 100) * 2 –50 ist ebenfalls ein gültiger Ausdruck und ergibt wieder 250.

Die Steueranweisung **`Option Explicit`** sollte stets als erste Anweisung (ggf. nach Kommentaren) im Programm stehen. Sie weist VBA an nur Variablen (bzw. Konstanten) zu akzeptieren, die vorher durch einen Dim / Const Befehl angemeldet (deklariert) wurden. Es ginge auch ohne, in diesem Fall würden Schreibfehler bei einer Variablen, z.B. LangePausen (statt LangePause) zum Anlegen einer weiteren Variablen mit dem Anfangswert 0 führen. D.h. hieße, daß die oben so schön auf 1000 initialisierte (mit dem Wert 1000 besetzte) Variable LangePause im Befehl `Pause LangePause`**`n`** zu einer Pause der Länge 0 führen würde, d.h. Lampe ist gleich wieder aus. Macht man den Schreibfehler schon bei KurzPause = LangePausen/4, so sind alle KuzePause = 0 und die gelbe Lampe leuchtet nicht mehr wahrnehmbar. Da wird man recht heftig ins Grübeln kommen (Schon weil man solche krummen Sachen eher der IDE als sich selber zutraut. Ist ja schließlich "alles" richtig!).

VBA : Datentypen, Option Explicit

Ein- und Ausgaben

```
' --- Ampel1.ftC : Eine einfache Ampel ----------------
Option Explicit
Const mGruen = 1, mGelb = 2, mRot = 3
Dim LangePause&, KurzePause&, Runde&

Sub Main
  LangePause = 1000*EA
  KurzePause = LangePause/4
  PrintStatus "--- Die Ampel bei der Arbeit ---"
  Do
    Runde = Runde + 1
    Debug.Print "Runde : " & Runde
    SetMotor mGruen, ftiEin
    Pause    LangePause
    .........
    SetMotor mGelb,  ftiAus
  Loop Until Finish
End Sub
```

In der neuen Fassung wird über den Befehl **EA** auf das korrespondierende Feld der IDE zugegriffen und ein Faktor zur Modifikation der Ablaufzeiten ausgelesen (Vorgabe ist 1). So kann man recht elegant die bisherige "auf Tempo" getrimmte Fassung besser den realen Bedingungen an einer Ampel anpassen (bei einem Faktor 30 muß man dann genauso endlos warten wie an einer realen Ampel).

Mit dem Befehl **PrintStatus** kann in die hellblaue Statuszeile geschrieben werden. Hier wird die **Textkonstante** "--- Die Ampel bei der Arbeit ---" ausgegeben. Das ist eine neue Art von Konstante – bisher waren nur numerische Konstanten im Spiel. Textkonstanten beginnen und enden mit einem doppelten Hochkomma (") und können alle anzeigbaren Zeichen enthalten, sie müssen auf eine Zeile passen.

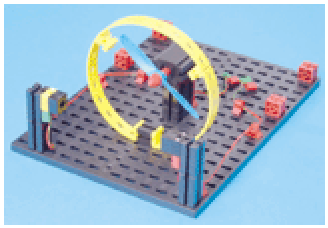
Die neue Variable **Runde** soll die durchlaufenen Schleifen zählen. Deklaration oben mit Dim wie gewohnt, keine Besetzung mit einem Anfangswert, hier wird der Standardwert 0 genutzt. Zu Beginn jeder Schleife wird dann der Wert von Runde um 1 erhöht
Der Wert von Runde + 1 ist ein Ausdruck, der wieder Runde zugewiesen wird, dadurch wird der alte Wert von Runde (beim ersten Mal 0), der noch im Ausdruck galt, überschrieben.

Debug.Print schreibt dann in den bisher noch nicht genutzten Protokoll-Bereich. Geschrieben wird eine Text-Konstante und eine Zahl (der aktuelle Wert von Runde) die durch den Operator Ampersand (&) mit einander in geeigneter Weise zu Text verbunden werden. Im Protokoll-Bereich werden die Ausgaben nicht überschrieben, sondern gescrollt (nach oben geschoben). Über den Scrollbar rechts können auch die anderen Zeilen angezeigt werden. Der Protokoll-Bereich ist normalerweise nur im Run-Modus sichtbar, er kann aber auch durch Menü Ansicht | Immer teilen dauerhaft sichtbar gemacht werden.

Der Protokoll-Bereich erlaubt auch Eingaben. z.B. `? KurzePause` zur Anzeige des Inhalts der Variablen KurzePause oder KurzePause = 123 um den Wert von KurzePause zu ändern. Möglich ist auch ein SetMotor mGruen, ftiEin, es lohnt aber kaum, da man es schneller über das Interface Panel erledigen kann. Ebenso kann man den Inhalt von Variablen einfacher durch Positionieren des Cursors (des Mauszeigers) über der Variablen anzeigen.

VBA : Debug.Print, Debug.Clear FishFace Referenz : PrintStatus

Motoren, Taster und Lichtschranken



Motor an M1,
Lichtschranke M2 Lampe,
I1 Phototransistor

```
Const mMotor = 1, mLampe = 2
Const eTaster = 1, ePhoto = 1
Sub Main
  Do
    SetMotor    mMotor, ftiEin
    Pause 5000
    SetMotor    mMotor, ftiAus
  Loop Until Finish
End Sub
```

"Der Händetrockner soll nun so programmiert werden, daß, sobald die Lichtschranke unterbrochen wird, der Lüfter ein- und nach 5 Sekunden wieder ausgeschaltet wird."

Mit dem oben angegebenen Programm läuft der Trockner, aber leider trotz Pause ewig. Was fehlt ist ein Auslöser : die Lichtschranke oder ein einfacher Taster, erstmal wird ein Taster an I1 probiert.

```
  Do
    If GetInput(eTaster) Then
      SetMotor    mMotor, ftiEin
      Pause 5000
      SetMotor    mMotor, ftiAus
    End If
  Loop Until Finish
```

Als erstes fällt die neue Konstruktion `If ... Then ... End IF` auf. Sie ist, wie das schon bekannte Do .. Loop, eine Programmklammer. Hier werden die Befehle zum Schalten des Trocknermotors eingeklammert. Dem If folgt eine Bedingung (ganz wie beim Loop Until), hier ist das GetInput(eTaster). Dieser Befehl fragt den Eingang I1 ab, wenn der geschlossen ist, liefert GetInput das Ergebnis True (Wahr). D.h. der Inhalt der Klammer wird unter der Bedingung ausgeführt, daß der Eingang I1 True ist.
Wenn man nun kurz auf den **Taster** (Schließer : angeschlossen sind die Kontakte 1 und 3) drückt, läuft der Motor an und tut das weitere 5 Sekunden und wird dann wieder abgeschaltet. D.h. das gesamte Programm wartet hier, ein erneutes Drücken des Taster hat in dieser Zeit keine Wirkung.

Der Taster ist recht unbequem, deswegen nun zur Lichtschranke.

```
SetMotor mLampe, ftiEin
Pause 1000
Do
  If GetInput(ePhoto) Then
    SetMotor    mMotor, ftiEin
    Pause 5000
    SetMotor    mMotor, ftiAus
  End If
Loop Until Finish
```

Vor dem Do-Loop muß erstmal die Lichtschranke aktiviert werden (jetzt ist der Phototransistor an I1 angeschlossen). Also SetMotor mLampe, ftiEin und ein wenig warten, damit sich der Phototransistor an das Licht gewöhnt, man kann das auf dem Interface Panel kontrollieren. Und dann die If Bedingung in GetInput(ePhoto) ändern. Der Motor läuft, nur leider auch schon, wenn die Lichtschranke noch gar nicht unterbrochen ist. Wenn man dann die Hand reinhält, bleibt der Motor nach spätestens 5 Sekunden stehen, es läuft also genau umgekehrt. Daran muß gedreht werden :

```
If Not GetInput(ePhoto) Then
```

stellt die Bedingung auf den Kopf. Das If wird jetzt ausgeführt, wenn GetInput(ePhoto) False liefert, die Lichtschranke also unterbrochen ist. Verursacht wird das durch das vorangestellte **NOT**. Jetzt geht's!

Jetzt noch einige Verzierungen :

```
' --- Trockner.ftC : Händetrockner ---
Option Explicit
Const mMotor = 1, mLampe = 2
Const eTaster = 1, ePhoto = 1
Dim Kunden

Sub Main
  Kunden = 0
  Debug.Clear
  SetMotor mLampe, ftiEin
  Pause 1000
  Do
    PrintStatus "Trockner bereit"
    If Not GetInput(ePhoto) Then
      PrintStatus "Trockner läuft"
      Kunden = Kunden + 1
      Debug.Print "Kunde Nr : " & Kunden & " am " & Now
      SetMotor    mMotor, ftiEin
      Pause 5000
      SetMotor    mMotor, ftiAus
    End If
  Loop Until Finish
End Sub
```

Mit PrintStatus wird in der blauen Statuszeile angezeigt, ob der "**Trockner bereit**" ist oder ob der "**Trockner läuft**".

Außerdem wird in im Log-Bereich die Nutzung des Trockner protokolliert. Dazu wird eine Variable **Kunden** angelegt und vor dem Do-Loop auf 0 initialisiert, außerdem wird der Protokoll-Bereich durch Debug.Clear gelöscht. Bei jedem Motorstart wird **Kunden + 1** hochgezählt und mit Debug.Print ausgegeben.
Die Ausgabe wird dabei ganz raffiniert "zusammengebastelt" :
Zuerst die **Textkonstante** "Kunde Nr : ", dann, durch & verknüpft, die aktuelle Kundenzahl, dann wieder eine Textkonstante " am " und zum Schluß **Now**, das Tagesdatum mit Uhrzeit, das vom System bereitgestellt wird.

Das wars.

Und noch eine Spielerei

```
    SetMotor mMotor, ftiEin
    Pause 3000
    SetMotor mMotor, ftiEin, ftiHalf
    Pause 2000
    SetMotor mMotor, ftiAus
```

Nach dem gewohnten SetMotor kommt noch eins mit einem zusätzlichen Parameter **ftiHalf**. Bedeutet : der Motor läuft jetzt nur noch mit halber Drehzahl (so zum Nachtrocknen). Der Befehl SetMotor hat also einen Parameter mehr als man dachte. Das funktioniert so : eigentlich hat er immer drei, wenn man den dritten wegläßt, wird das intern erkannt und er wird durch einen default (Ersatz) Wert (hier ftiFull) ersetzt. Erlaubt sind hier auch die Werte 0 – 15, 0 für aus über 7 für ftiHalf und 15 für ftiFull. Man kann also noch mehr spielen.

Und noch ein Nachtrag

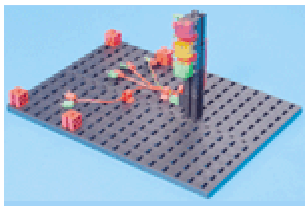
Wenn an einem M-Ausgang ein Motor angeschlossen ist, kann der Ausgang auch mit **ftiLinks** (= ftiEin) und **ftiRechts** betrieben werden, es kann also die Drehrichtung des Motors vorgegeben werden (also der Langsamgang vielleicht ftiRechts – rum?).

Zur Drehrichtung : mit ftiLinks wird die Drehrichtung bezeichnet, die der Motor einnimmt, wenn man beim Interface Panel auf "L" drückt. Wenn das falschrum ist, sollte man am Motor

das Kabel umstecken. Am Interface selber sollt der rote Stecker immer in "der ersten Reihe" (also vorn) stecken, das schafft Übersicht.

VBA : Operatoren

Intermezzo : Nocheinmal die Ampel



Lampen an M1 – M4
grün – gelb – rot –
FußGrün (an der Seite)
Taster (1/3) an I1 und I2

```
Option Explicit
Const mGruen = 1, mGelb = 2, mRot = 3,
Const mFuss = 4
Const eFussWunsch = 1

Sub Main
Do
  SetMotor mGruen, ftiEin
  If GetInput(eFussWunsch) Then
    Pause    1000*EA
    SetMotor mGruen, ftiAus
    SetMotor mGelb,  ftiEin
    Pause    250*EA
    SetMotor mGelb,  ftiAus
    SetMotor mRot,   ftiEin
    Pause    1000*EA
    SetMotor mGelb,  ftiEin
    Pause    250*EA
    SetMotor mRot,   ftiAus
    SetMotor mGelb,  ftiAus
  End If
Loop Until Finish
End Sub
```

Das Programm ist weitgehend bekannt, aber eine If ... End If "Klammer" ist hinzu gekommen und schon ist daraus eine Fußgängerampel geworden. GetInput(eFussWunsch) wartet auf das Drücken der Tasters zur Anforderung einer Fußgängerphase, erst dann läuft es mit mGelb weiter. eFussWunsch wurde vor der vorhanden Pause platziert um zu verhindern, daß sofort nach Ablauf einer Fußphase gleich wieder eine neue gestartet wird.

Was noch fehlt ist eine Fußgängerampel, da es am Interface ein wenig eng wird, nur eine grüne an M4. Das sieht dann so aus :

```
Do
  SetMotor mGruen, ftiEin
  If GetInput(eFussWunsch) Then
    Pause    1000*EA
    SetMotor mGruen, ftiAus
    SetMotor mGelb,  ftiEin
    Pause    250*EA
    SetMotor mGelb,  ftiAus
    SetMotor mRot,   ftiEin
    SetMotor mFuss,  ftiEin
    Pause    1000*EA
    SetMotor mGelb,  ftiEin
    SetMotor mFuss,  ftiAus
    Pause    250*EA
    SetMotor mRot,   ftiAus
    SetMotor mGelb,  ftiAus
  End If
Loop Until Finish
```

Elself und Subs

Nachts werden Ampeln oft auf Gelb-Blinken umgestellt :

```
Const eFussWunsch = 1, eGelbBlinken = 2
Sub Main
  Do
    If GetInput(eGelbBlinken) Then
```

```
      Blinken
    ElseIf GetInput(eFussWunsch) Then
      Pause    1000*EA
      SetMotor mGruen, ftiAus
      .....
      SetMotor mGelb,  ftiAus
      SetMotor mGruen, ftiEin
    Else
      SetMotor mGruen, ftiEin
    End If
  Loop Until Finish
End Sub

Sub Blinken
  SetMotor mGelb, ftiEin
  Pause 500*EA
  SetMotor mGelb, ftiAus
  Pause 400*EA
End Sub
```

Das GelbBlinken wird über die Abfrage eines weiteren Tasters eGelbBlinken geschaltet. Die vorhandene If ... End If Klammer wird um eine neue Klausel – ein **ElseIf** – erweitert. Beim If wird wie bisher auf eGelbBlinken getestet, wenn das nicht zutrifft (eGelbBlinken hat Vorrang) auf eFussWunsch und wenn das auch nichts war wird ganz normal das (Auto)mGruen eingeschaltet (wird jetzt also nicht mehr bei jedem Schleifendurchlauf geschaltet).

Das Blinken selber wird durch ein ebenfalls neues Sprachelement – ein **Unterprogramm** (Sub) – gesteuert. In einem Unterprogramm werden logisch zusammengehörenden Befehle zusammengefaßt und außerhalb des normalen Programmablaufs abgestellt. Ein Unterprogramm wird durch seinen Namen aufgerufen (beim If eGelbBlinken : **Blinken**). Vorteil eines Unterprogramms : das eigentliche (Haupt)Programm wird übersichtlicher, bei mehrmaligem Aufruf (wenn man auch noch woanders Blinken will) wird vorhander Code wiederverwendet.

VBA : If ..Then..Else-Anweisung

SetMotors

Die vielen SetMotor Ein und Aus sind langsam unübersichtlich, mit SetMotor**s** kann man sie zusammenfassen und alle Lampen auf einmal schalten :

```
' --- FussAmpel3.ftC : FußgängerAmpel mit SetMotors --------
Option Explicit
Const mGruen = &H1, mGelb = &H4, mRot = &H10, mFuss = &H40
Const eFussWunsch = 1, eGelbBlinken = 2

Sub Main
  Do
    If GetInput(eGelbBlinken) Then
      Blinken
    ElseIf GetInput(eFussWunsch) Then
      Pause     1000*EA
      SetMotors mGelb
      Pause     250*EA
      SetMotors mRot + mFuss
      Pause     1000*EA
      SetMotors mGelb
      Pause     250*EA
      SetMotors mGruen
    Else
      SetMotors mGruen
    End If
```

```
  Loop Until Finish
End Sub

Sub Blinken
  SetMotors mGelb
  Pause     500*EA
  ClearMotors
  Pause 400
End Sub
```

Dazu muß man wissen, das der Status aller M-Ausgänge in einem OutputStatusword abgebildet werden kann, jeweils zwei bit für einen M-Ausgang : `00 00 00 00 die` M-Ausgänge M4 – M1 sind abgeschaltet. Es wird mit SetMotor**s** an das Interface weitergegeben.

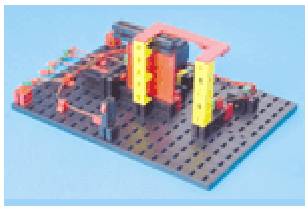
```
00 01 00 00       (&H10, mRot  in binär Darstellung)
01 00 00 00 +     (&H40, mFuss)

-------------
01 01 00 00 =     mFuss ein, mRot ein, mGelb aus, mGruen aus
```

Dazu wurden die Lampen-Konstanten entsprechend geändert. Sie enthalten jetzt nicht mehr die Nummer des M-Ausganges sondern die bit-Position des M-Ausganges im OutputStatusword. Geschrieben wurde das in der kürzeren Hexa-Darstellung, möglich wäre auch die dezimale Darstellung 1, 4, 16 und 64, die interne Darstellung ist immer binär.

Wenn man das Programm noch verschönern will, kann man die PrintStatus- und Debug.Print-Ausgaben des Händetrockners entsprechend modifiziert übernehmen.

FishFace Referenz : SetMotors, ClearMotors
VBA Call, Sub, Function

Über das Türenschieben



Motor an M1
Lichtschranke M2, I4
Taster TürAuf I2
Taster TürZu I1
Taster Öffnen I3

"Wenn Taster I3 gedrückt wird, soll sich die Tür öffnen und nach fünf Sekunden wieder schließen."

```
' --- Tuer.ftC : Schiebetür ---
Option Explicit
Const mTuer = 1, mLampe = 2
Const eTuerZu = 1, eTuerAuf = 2, eOeffnen = 3
Const ePhoto = 4

Sub Main
  Do
    SetMotor mTuer, ftiLinks
    WaitForInput eTuerZu, False
    SetMotor mTuer, ftiAus

    WaitForInput eOeffnen

    SetMotor mTuer, ftiRechts
    WaitForInput eTuerAuf
    SetMotor mTuer, ftiAus
    Pause 5000
  Loop Until Finish
End Sub
```

Zunächstmal wird die Tür geschlossen (SetMotor). Das wird durch den neuen Befehl **WaitForInput** überwacht. WaitForInput überwacht den eTuerZu-Eingang am Interface und zwar darauf, daß der zugehörende Taster öffnet (deswegen der Parameter False). Das ist zunächst etwas überraschend, wenn man sich aber die baulichen Verhältnisse und die Vorliebe für eine Schaltung als Schließer (Kontakte 1 und 3) ansieht, verständlich.

Und dann wird schon wieder gewartet : auf eine Anforderung zum Türöffnen über eOeffnen. Der Taster ist wieder als Schließer geschaltet und wird gedrückt (geschlossen) deswegen wird hier auf `WaitForInput eOeffnen, True` gewartet, da True der default Parameter ist, kann man ihn auch weglassen, also `WaitForInput eOeffnen`.

Und dann geht endlich die Tür für 5 Sekunden auf.

FishFace Referenz : WaitForInput

Überwachung durch Lichtschranke

```
Sub Main
  SetMotor mLampe, ftiEin
  Pause 1000
  WaitForInput ePhoto, True
  Do
    Do While GetInput(eTuerZu)
      SetMotor mTuer, ftiLinks
      If Not GetInput(ePhoto) Then TuerOeffnen
    Loop
    SetMotor mTuer, ftiAus

    If GetInput(eOeffnen) Or GetInput(ePhoto) = False Then _
       TuerOeffnen
  Loop Until Finish
End Sub

Sub TuerOeffnen
  SetMotor mTuer, ftiRechts
  WaitForInput eTuerAuf
  SetMotor mTuer, ftiAus
```

```
  Pause 5000
End Sub
```

Das Modell sieht sie ja vor, die Lichtschranke ePhoto – mLampe, nun wird sie auch genutzt. Das fängt, wie vom Händetrockner gewohnt, mit dem "Anwärmen" der Lichtschranke an. Dann wird noch gewartet, dass sie auch wirklich OK ist, dann erst geht's mit dem gewohnten Do – Loop los.

Es folgt dann aber gleich noch einer, aber ein ungewohnter.
`Do While GetInput(eTuerZu)`. Neu ist das While, die Schleife wird solange durchlaufen, wie die Bedingung eTuerZu zutrifft (d.h. ja eigentlich Tuer ist noch nicht ganz zu, s.o.), außerdem ist dies eine "abweisende Schleife", wenn die Bedingung nicht zutrifft, d.h. die Tür bereits geschlossen ist, wird sie nicht durchlaufen (die bisherigen – nicht abweisenden – Schleifen wurden mindestens einmal durchlaufen). Das hat hier den Vorteil, daß nicht ständig an der Tür "gerüttelt" wird.

In der Schleife wird dann der Türmotor eingeschaltet (Richtung Schließen) das wird aber gleich wieder revidiert, wenn irgendetwas in die Lichtschranke (GetInput(ePhoto) geraten ist, dann wird das Unterprogramm TuerOeffnen ausgeführt. Nach der Schleife wird der Türmotor wieder ausgeschaltet.

Das nachfolgende If ersetzt das bisherige WaitForInput. Jetzt werden der eOeffnen-Taster und die Lichtschranke auf eine Anforderung zum TürÖffnen abgefragt. Im positiven Fall : TuerOeffnen. Hier also ein Beispiel für die mehrmalige Verwendung eines Unterprogramms.

```
Sub TuerOeffnen(Normal)
  PrintStatus "--- Tür öffnet ---"
  If Normal Then Debug.Print "Tür wurde geöffnet : " & Now _
  Else Debug.Print "Tür-Zwischenfall   : " & Now
  SetMotor mTuer, ftiRechts
  WaitForInput eTuerAuf
  SetMotor mTuer, ftiAus
  PrintStatus "--- Tür geöffnet ---"
  Pause 5000
End Sub
```

Mann kann einem Unterprogramm auch einen, oder mehrere Parameter mitgeben um den Ablauf des Unterprogramms zu steuern. Dazu müssen sie (die Parameter) in der Sub Definitionszeile angeführt werden (formale Parameter, hier **Normal**), sie können dann im Unterprogramm wie normale Variable genutzt werden, hier wird unterschieden, ob die Tür per Anforderung oder durch "Kiste in der Lichtschranke" geöffnet wurde. Beim Aufruf des Unterprogramms muß dann ein entsprechender aktueller Parameter mitgegeben werden : `TuerOeffnen False` bei der "NotÖffnung" und `TuerOeffnen True` nach Anforderung.

VBA: Or , Sub Anweisung

Temperatur-Regelung

Motor an M1
Lampe an M2
NTC-Widerstand an EX

"Oberhalb einer bestimmten Temperatur schaltet die Heizung aus- und die Kühlung ein, bei Erreichen eines unteren Grenzwertes soll die Heizung ein- und die Kühlung ausgeschaltet werden."

```
Option Explicit
Const mMotor = 1, mLampe = 2
Const aNTC = 0
Dim Temperatur&

Sub Main
  Do
    Temperatur = GetAnalog(aNTC)
    If Temperatur < EA Then
      SetMotor mLampe, ftiAus
      SetMotor mMotor, ftiEin
    ElseIf Temperatur > EB Then
      SetMotor mLampe, ftiEin
      SetMotor mMotor, ftiAus
    End If
  Loop Until Finish
End Sub
```

Als erstes wird in der Do – Loop Schleife die Variable Temperatur mit dem aktuellen Wert des NTC (Negative Temperature Coefficient, Widerstand, der bei steigender Temperatur kleinere Werte annimmt, also nicht parallel zur Temperatur steigt – deswegen Negative). Die Werte können unterschiedlich ausfallen, man sollte anfangs ein wenig experimentieren (der NTC – zwischen Daumen und Zeigefinger genommen – erwärmt sich rapide) und die Werte bei EX auf dem Interface Panel ablesen.

Danach treffen wir wieder auf eine If ... ElseIf ... End If Konstruktion. Wenn der Temperaturwert kleiner als der untere Wert (EA) ist, dann ist es zu heiß (-> NTC), es wird gekühlt. Wenn dann der Temperaturwert größer als EB ist, wird wieder geheizt. Dies Programm wurde mit EA = 575 und EB = 600 getestet.

FishFace Referenz : GetAnalog

Dreipunkt-Regelung

Bei dem vorherigen Beispiel war immer etwas los (Kühlen / Heizen im Wechsel) und trotzdem wurde die Temperatur im Versuch nur im Bereich von 575 bis 600 gehalten, könnte man das nicht auch einfach durch "Abwarten" erreichen. Also abschalten und warten bis die Temperatur – je nach Lage der Dinge – von alleine wieder steigt bzw. fällt. Das würde dann auch noch Energie sparen :

```
' --- TemperaDreiTT.ftC : Temperaturregelung ---
Option Explicit
Const mMotor = 1, mLampe = 2
Const aNTC = 0
Dim UT&, OT&

Sub Main
  UT = EA * (1 - EB / 100)
  OT = EA * (1 + EB / 100)
  Debug.Print "Zieltemperatur : " & EA
  Debug.Print "Untergrenze    : " & UT
  Debug.Print "Obergrenze     : " & OT
  Do
    If Temperatur < UT Then
      PrintStatus "--- Heizt : " & Temperatur & " ---"
      SetMotor mLampe, ftiEin
      SetMotor mMotor, ftiAus
      UT = EA
    ElseIf Temperatur < OT Then
      PrintStatus "--- Temperatur : " & Temperatur & " ---"
      ClearMotors
      UT = EA * (1 - EB / 100)
      OT = EA * (1 + EB / 100)
    Else
      PrintStatus "--- Kühlt : " & Temperatur & " ---"
      SetMotor mLampe, ftiAus
      SetMotor mMotor, ftiEin
      OT = EA
    End If
  Loop Until Finish
End Sub

Function Temperatur
  Temperatur = (1000 - GetAnalog(aNTC)) / 10 - 12
End Function
```

Hier wurden mehrere Maßnahmen in einen Schritt in ein (beinahe) neues Programm eingebaut :

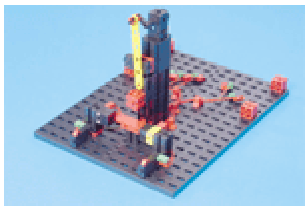
1. **Zieltemperatur** : In EA wird jetzt eine echte Temperatur angegeben, die als Mittelwert zu halten ist. Die vom Mittelwert erlaubten Abweichungen werden in EB in Prozent angegeben. Im Beispiel sind das EA = 29 (geschätzte Grad) und EB = 2%
2. **Function** Temperatur : die mit GetAnalog gemessenen Werte im Bereich von 0 – 1000 werden in Grad umgerechnet. Dazu wird ein neues Konstrukt, eine Function, genutzt. Eine Function ist ein Unterprogramm, das einen Wert als Ergebnis seines Aufrufs zurückgibt. Hier ist das die aktuelle Temperatur. Der gemessene Wert (GetAnalog) wird erstmal umgedreht (1000 – eNTC, jetzt entspricht ein größerer Wert auch einer höheren Temperatur. Der ermittelte Wert wird durch 10 geteilt und dann werden nochmal 12 abgezogen, das sollen dann Grad Celsius sein. Der NTC wurde nicht wirklich geeicht, aber die Werte sind realitätsnah im Bereich 25 – 35.

3. Die Variablen **UT** und **OT** stehen für untere/obere zulässige Temperatur. Sie werden aus Zieltemperatur EA und prozentualer Abweichung EB errechnet und gleich auch im Protokoll-Feld ausgegeben, weils doch mit dem Kopfrechnen so seine Probleme hat.
4. Die If ... ElseIf ... End If Konstruktion wurde um einen Else Zweig erweitert, die zugehörenden Temperatur-Abfragen wurden geändert und den neuen Temperaturwerten angepaßt :
   If : **Zu niedrige Temperatur**
   ElseIf : **Temperatur zwischen UT und OT** also im erlaubten Zielbereich
   Else : **Zu hohe Temperatur**
5. Geschaltet wird wie bisher, aber UT und OT werden laufend geändert. Motto : **Wenn schon Kühlen/Heizen, dann aber richtig und dann Pause**. Wird eine niedere Temperatur erkannt wird UT = EA gesetzt um ein höheres Aufheizen bis zur Zieltemperatur zu erreichen. Beim Kühlen wird dann entsprechend OT auf EA abgesenkt. Bei Temperaturen im Zielbereich werden UT / OT wieder auf die Ausgangswerte gesetzt.
6. In der **StatusZeile** wird die aktuelle Temperatur angezeigt verbunden mit dem Hinweis : Heizt, Temperatur, Kühlt.

Ganz schön viel Stoff für so ein doch noch eher kleines Programm. Tip : Wenn einseitig nur geheizt und pausiert wird. Daumen und Zeigefinger um den NTC führen ihn zu beachtlichen Temperaturen und der Wind kommt.

VBA : Function-Anweisung

Stanzmaschine



Motor an M1
Lampe an M2
Endtaster an I1
Photo-Widerstand an I2
Bedientaster links an I3
Bedientaster rechts an I4

"Die Maschine soll ein Teil in einem Arbeitsgang mit 4 Hüben stanzen. Sie darf nur starten, wenn der Bediener beide Taster betätigt und gleichzeitig die Lichtschranke geschlossen ist. Eine Unterbrechung der Lichtschranke während eines Arbeitsgangs stoppt die Maschine mit Warnsignal."

```
' --- StanzMaschine.ftC ---
Option Explicit
Const mStanze = 1, mLampe = 2
Const eEnde  = 1, ePhoto = 2
Const eLinks = 3, eRechts = 4
Dim Produktion&, TeilOK As Boolean, i&

Sub Main
  Produktion = 0
  SetMotor mLampe,  ftiEin
  SetMotor mStanze, ftiEin
  WaitForInput      eEnde
  SetMotor mStanze, ftiAus

  Do
    If GetInput(ePhoto) And _
       GetInput(eLinks) And _
       GetInput(eRechts) Then
       TeilOK = True
       For i = 1 To EA
         If GetInput(ePhoto) Then
            SetMotor mStanze, ftiEin
            WaitForHigh eEnde
            SetMotor mStanze, ftiAus
         Else
            SetMotor mStanze, ftiAus
            Debug.Print "PhotoMist : " & Now
            TeilOK = False
            Beep 1111, 100
            Beep 555,  100
            Beep 1111, 100
            Exit For
         End If
        Next i
        If TeilOK Then _
           Produktion = Produktion + 1
    End If
    PrintStatus "Anzahl Teile : " _
                               & Produktion
  Loop Until Finish
End Sub
```

Die interessantesten Punkte des Programms sind :

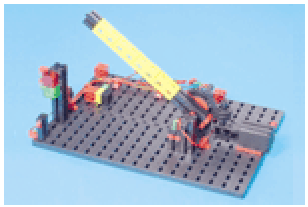
1. Beim Start des Programms wird die Maschine auf **Ausgangslage** gefahren (Lichtschranke an, Stanze oben, Produktion = 0)
2. Ein **Stanzvorgang** kann nur ausgelöst werden, wenn die Lichtschranke geschlossen ist. Die Auslösung erfolgt durch "**Zweihandeinrückung**" : eLinks und eRechts gleichzeitig. Das ergibt dann ein gewaltiges If bei dem die einzelnen Bedingungen durch And verknüpft sind : alle Bedingungen müssen aufeinmal wahr sein. Achtung es werden hier Fortsetzungszeilen verwendet (_) am Zeilenende. Vor dem (_) muß eine Leerstelle stehen.
3. Neu ist auch die **For ... Next Schleife**, die die Anzahl der in EA vorgegebenen Hübe überwacht.
4. Vor jedem Hub wird die Lichtschranke nocheinmal kontrolliert und ggf. kräftig gemeckert und gehupt und mit Exit For die For ... Next Schleife und damit der **Stanzvorgang abgebrochen.**

5. Der einzelne Hub wird mit **WaitForHigh** kontrolliert. WaitForHigh wartet dass eEnde erst auf False und dann auf True wechselt. Das ist erforderlich, da ein einfaches Warten auf True sofort zum Erfolg führen kann, da der Taster vom letzten Hub noch auf True steht.
6. Nach jedem Stanzvorgang wird das **Produktionsergebnis** aufaddiert. Es werden aber nur die erfolgreichen Stanzvorgänge gezählt. Dazu die Variable TeilOK, die vor der For ... Next Schleife verdachtsweise auf True gesetzt wird und ggf. bei Abbruch im Zuge des Meckerns auf False geändert wird.
7. Die aktuelle **Produktion** wird in der Statuszeile und die **Zwischenfälle** werden im Protokoll-Fenster angezeigt.

FishFace Referenz : WaitForHigh
VBA : For..Next-Anweisung, Beep

Findige Bediener werden, der Sicherheitsmaßnahmen zum Trotz, doch noch ihre Finger in die laufende Maschine stecken können (die Abschaltung erfolgt erst nach einem Hub, es wird nicht geprüft, ob die Zweihandeinrückung vor jedem Stanzvorgang betätigt wird – ein "Dauerlauf" durch "festgeklemmte" Taster ist möglich). Hier soll es aber erstmal genügen.

Parkhausschranke



Motor an M1
Rote Lampe an M2
Grüne Lampe an M3
Lichtschranke an M4
ZuTaster an I1
AufTaster an I2
BedienTaster an I3
Photowiderstand I4

"Durch Betätigen des Tasters I3 soll die Schranke geöffnet werden. Ist die Schranke offen, leuchtet die Ampel grün. Erst wenn die Lichtschranke passiert wurde, springt die Ampel auf Rot und die Schranke schließt wieder"

```
' --- Parkhaus1.ftC ---
Option Explicit
Const mSchranke = 1, mRot = 2, mGruen = 3
Const mLicht = 4
Const eZu = 1, eAuf = 2, eOeffnen = 3
Const ePhoto = 4
Const sZu = 1, sAuf = 2

Sub Main
  SetMotor mLicht,    ftiEin
  SetMotor mRot,      ftiEin
  Do
    PrintStatus  "--- Schranke schließt ---"
    SetMotor     mSchranke, sZu
    WaitForInput            eZu
    SetMotor     mSchranke, ftiAus
    PrintStatus  "--- Wartet auf Kunden ---"
    WaitForHigh  eOeffnen
    PrintStatus  "--- Schranke öffnet ---"
    SetMotor     mSchranke, sAuf
    WaitForInput eAuf
    SetMotor     mSchranke, ftiAus
    SetMotor     mRot,      ftiAus
    SetMotor     mGruen,    ftiEin
    PrintStatus  "- Warten auf Durchfahrt -"
    WaitForLow   ePhoto
    WaitForHigh  ePhoto
    SetMotor     mGruen,    ftiAus
    SetMotor     mRot,      ftiEin
    WaitForTime  500
  Loop Until Finish
End Sub
```

Das zugehörende Programm ist eine schöne Sammlung bekannter Elemente, lediglich ein **WaitForLow** als Pendant zu WaitForHigh (diesmal : True/False Durchgang) hat sich eingefunden. Mit der Kombination WaitForLow/WaitForHigh wird kontrolliert ob ein Auto in die Lichtschranke ein- und wieder ausgefahren ist. Sicherheitshalber wird danach noch ein wenig gewartet. Die eingestreuten PrintStatus erklären den weiteren Ablauf.

Spannend wird das, wenn anstelle eines schlichten Taster-Drucks ein PIN eingegeben werden muß :

```
  PrintStatus  "--- Wartet auf Kunden ---"
  If WaitForCode Then
    PrintStatus  "--- Schranke öffnet ---"
    .....
    WaitForTime  500
  End If
Loop Until Finish

Function WaitForCode
  WaitForCode = True
  If Secret(InputBox("Bitte Zugangscode eingeben!", _
                     "ftParkhaus GBR")) Then Exit Function
  If MsgBox("Sorry - das wars nicht", _
            vbOKCancel+vbCritical, _
            "ftParkhaus GBR") = vbCancel Then
    NotHalt = True
    PrintLog "Das Programm wurde gewaltsam beendet " & Now
  End If
  WaitForCode = False
End Function

Function Secret(PINcode)
  Secret = False
  If PINcode = CStr(Day(Now)*100 + Month(Now)) Then Secret = True
End Function
```

Anstelle des bisherigen WaitForHigh ein `If WaitForCode Then` das als Ergebnis ein True oder False zurückgibt. Im True-Falle geht's weiter wie bisher, sonst wird der Rest übersprungen und man landet wieder beim WaitForCode. Die Funktion WaitForCode sieht zwar so aus wie die anderen WaitFor-Funktionen (damit man sie auch ernst nimmt), ist aber eine Funktion, die gleich unten im Programm zu finden ist.

In WaitForCode gibt es als erstes eine Abfrage nach der PIN. Das geschieht über die (System)Funktion **InputBox**, zurückgegeben wird der eingegebene PIN-Code. Die Parameter von InputBox bedeuten der Reihe nach : Eingabeaufforderung, Text in der Titelzeile. Das Ergebnis wird einer hier erstellten Function **Secret** übergeben, diese wiederum vergleicht es mit dem intern gebildeten aktuellen PIN.

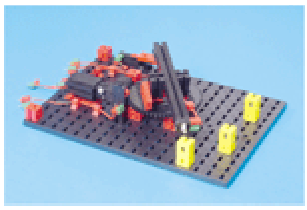
War der eingegebene PIN-Code falsch wird mit **MsgBox** gemeckert. Die Parameter : der Mecker-Spruch, die Button mit denen geantwortet werden kann + ein Icon mit dem die Meldung verziert wird und dann noch der Text in der Titelzeile.

Bei der Anwort Abbrechen (Cancel) wird eine "**Notbremse**" aktiviert die FishFace-Eigenschaft NotHalt wird auf True gesetzt, das wird dann auch noch protokolliert. Die Funktion wird wie bei Antwort OK mit False verlassen. In beiden Fällen wird dann der nachfolgende Code übersprungen. Aber im Fall NotHalt wird am Loop-Ende Finish hellhörig und bricht das Programm ab.

Und dann wäre da noch die Function **Secret**, die die aktuelle PIN liefert und mit dem als Parameter übergebenen PIN-Code aus der InputBox vergleicht, ja die wird aus Tag und Monat zusammengebastelt (heute war es 1303 und das war im März). Man kann die Sache auch noch viel spannender machen.

FishFace Referenz : WaitForLow
VBA : InputBox-Funktion, MsgBox-Funktion

Der Schweißroboter



Motor an M1
Lampe an M2
EndeTaster an I1
ImpulsTaster an I2

"Der Roboter soll drei Positionen anfahren und an jeder Position eine Schweißung durchführen. Danach soll er in seine Ausgangsposition zurückkehren und von vorne beginnen."

```
' --- SchweissRobot1.ftC ---
Option Explicit
Const mRobot = 1, mSchweiss = 2
Const eEnde = 1, eImpuls = 2
Dim IstPosition&

Sub Main
  Do
    SetMotor       mRobot, ftiLinks
    WaitForInput   eEnde
    SetMotor       mRobot, ftiAus
    IstPosition    = 0

    SetMotor       mRobot,  ftiRechts
    WaitForPosUp   eImpuls, IstPosition, 56
    SetMotor       mRobot,  ftiAus
    Schweissen

    SetMotor       mRobot,  ftiRechts
    WaitForPosUp   eImpuls, IstPosition, 144
    SetMotor       mRobot,  ftiAus
    Schweissen

    SetMotor       mRobot,  ftiLinks
    WaitForPosDown eImpuls, IstPosition, 94
    SetMotor       mRobot,  ftiAus
    Schweissen
  Loop Until Finish
End Sub

Sub Schweissen
Dim i&
  For i = 1 To EA
    SetMotor mSchweiss, ftiEin
    Pause 100 * EB
    SetMotor mSchweiss, ftiAus
    Pause 100 * EB
  Next i
End Sub
```

Es geht alles schön der Reihe nach. Begonnen wird mit dem Anfahren der Ausgangsposition (Home Position) um einen Bezugspunkt zu gewinnen. Das ist IstPosition = 0.

Anschließend folgen drei sehr ähnliche Anweisungsblöcke mit denen die gewünschten Positionen angefahren werden. Motor einschalten (Richtung beachten), Mit dem neuen Befehl **WaitForPosUp** / **WaitForPosDown** darauf warten, daß die als Konstante vorgegebene Zielposition erreicht wird (d.h. Zielposition = IstPosition wird), Motor wieder abschalten und in Ruhe "Schweissen" (Unterprogramm Schweissen).

WaitForPosUp zählt die IstPosition solange hoch, bis die Zielposition erreicht ist, WaitForPosDown zählt sie herunter. Gezählt werden Impulse am Taster eImpuls. Um den Motor kümmert sich der Befehl nicht.

FishFace Referenz : WaitForPosUp, WaitForPosDown

Das geht auch anders

Relatives Positionieren – Asynchrones Fahren

```
' --- SchweissRobot2.ftC ---
Option Explicit
Const mRobot = 1, mSchweiss = 2
Const eEnde = 1, eImpuls = 2

Sub Main
  Do
    SetMotor      mRobot, ftiLinks
    WaitForInput  eEnde
    SetMotor      mRobot, ftiAus

    SetMotor      mRobot, ftiRechts, ftiFull, 56
    WaitForMotors 0,      mRobot
    Schweissen

    SetMotor      mRobot, ftiRechts, ftiFull, 88
    WaitForMotors 0,      mRobot
    Schweissen

    SetMotor      mRobot, ftiLinks, ftiFull, 50
    WaitForMotors 0,      mRobot
    Schweissen
  Loop Until Finish
End Sub
```

Der Programmaufbau ist der gleiche, aber der ach so bekannte SetMotor hat noch mehr Parameter als man so denkt. `SetMotor mRobot, ftiRechts, ftiFull, 56`. Die ersten Parameter sind bekannt, ftiFull als Geschwindigkeitskeitsangabe ist auch schon mal vorgekommen. Neu ist der letzte Parameter, der gibt die Anzahl Impulse an, die der Motor in die vorgegebene Richtung fahren soll. Er tut das asynchron, d.h. ohne, daß das Programm anhält. Deswegen das nachfolgende **WaitForMotors** mit dem auf das Erreichen der vorgegebenen Position gewartet wird, ein Abschalten ist aber nicht erforderlich.

SetMotors setzt mit dieser Parameterliste einen RobotMotor voraus. D.h. einen Motor mit festzugeordneten Tastern. Bei M1 sind das I1 als Endtaster und I2 als Impulstaster. Der Endtaster muß außerdem mit ftiLinks angefahren werden können. Für die weiteren Motoren gilt entsprechend M2 : I3/I4, M3 : I5/6 ... WaitForMotors kann auch auf mehrere Motoren gleichzeitig warten :

```
SetMotor ftiM1, ftiRechts, ftiFull, 123
SetMotor ftiM4, ftiLinks,  ftiHalf, 34
WaitForMotors 0, ftiM1, ftiM4
```

Hier wird darauf gewartet, daß M1 123 Impulse mit ftiFull nach rechts und M4 34 Impulse mit ftiHalf nach links macht.

Die Positionsangaben sind **relativ** d.h. sie beziehen sich auf die aktuelle Postion. Zunächst werden, von IstPosition = 0 ausgehend, 56 Impulse nach rechts gefahren, ZielPosition ist 56, dann 88 Impulse, ZielPosition ist dann 56+88 = 144. Die nächsten 50 Impulse gehen nach links, das ergibt eine ZielPosition von 144-50 = 94.

FishFace Referenz : RobMotoren, SetMotor, WaitForMotors

Es geht aber noch einfacher, auch wenns erstmal komplizierter ist :

Absolute Positionierung

Die hatten wir ja eigentlich schon in der ersten Version hier wird es viel komplizierter, aber technisch anspruchsvoller gelöst. Verbunden mit einer Verlagerung in ein Unterprogramm wird die "Nutzanwendung" erstaunlich kurz und übersichtlich :

```
' --- SchweissRobot3.ftC ---
Option Explicit
Const mRobot = 1, mSchweiss = 2
Dim   RobotPos&

Sub Main
  Do
    Home
    MoveTo     56
    Schweissen EA, EB
    MoveTo     144
    Schweissen EA/2, EB*2
    MoveTo     94
    Schweissen EA, EB
  Loop Until Finish
End Sub

Sub MoveTo (ZielPos)
  If RobotPos < ZielPos Then
    SetMotor mRobot, ftiRechts, ftiFull, ZielPos - RobotPos
    Do
      PrintStatus "RobotPos : " & ZielPos - GetCounter(eImpuls)
    Loop While WaitForMotors(100, mRobot) = ftiTime
    RobotPos = ZielPos + GetCounter(eImpuls)
  Else
    SetMotor mRobot, ftiLinks, ftiFull, RobotPos - ZielPos
    Do
      PrintStatus "RobotPos : " & ZielPos + GetCounter(eImpuls)
    Loop While WaitForMotors(100, mRobot) = ftiTime
    RobotPos = ZielPos + GetCounter(eImpuls)
  End If
  PrintStatus "RobotPos : " & RobotPos
End Sub

Sub Schweissen (Mal, Dauer)
Dim i&
  For i = 1 To Mal
    SetMotor mSchweiss, ftiEin
    Pause 100 * Dauer
    SetMotor mSchweiss, ftiAus
    Pause 100 * Dauer
  Next i
End Sub

Sub Home
  PrintStatus "--- Es geht nach Hause ---"
  SetMotor mRobot, ftiLinks
  WaitForInput eEnde
  SetMotor mRobot, ftiAus
  RobotPos = 0
  PrintStatus "--- Zu Hause ---"
End Sub
```

Das Unterprogramm **MoveTo** unterscheidet zunächst, ob es nach links oder rechts geht. Anhand dessen wird die Anzahl Impulse bestimmt die mit SetMotor zu fahren sind und dann, wie gewohnt, mit WaitForMotors auf die Fertigmeldung gewartet. Hier wird WaitForMotors

aber gesagt nicht gleich auf fertig zu warten sondern nur 100 MilliSekunden (die 0 von vorher bedeutet endlos warten) und dann mit einem Returncode zurückzukehren. Abgefragt wird ftiTime : Ablauf der vorgegebenen Zeit. Es gibt auch noch ftiEnde, ftiNotHalt und ftiEsc. Das geschieht in einer Schleife, bis die Bedingung ftiTime nicht mehr zutrifft. Hier nimmt man schlicht an, daß alles gut gegangen ist (ftiEnde). Hier nocheinmal der Hinweis, es kann auch auf bis zu 8 Motoren gleichzeitig gewartet werden. Das Programm wird dann allerdings ein wenig komplizierter (siehe FishFa40AX Handbuch für Visual Basic 6).

Da hier in einer Schleife gewartet wird, kann da auch etwas getan werden. Hier z.B. die Anzeige der aktuellen Position in der Statuszeile. Der Ordnung halber wird sie zum Schluß noch einmal auf "Vordermann gebracht", es wird ja nur alle 100 Millisekunden abgefragt. Helfen tut dabei die Funktion **GetCounter,** die den aktuellen Stand der Impulszählung zurückgibt (immer positiv, gerechnet vom Startwert heruntergezählt auf 0).

Das Unterprogramm Schweißen wurde etwas modernisiert, man kann die einzelnen Punkte jetzt mit unterschiedlichen Werten schweißen. Die Befehle zum Anfahren der Ausgangsposition wurden in ein Unterprogramm verlagert.

FishFace Referenz : GetCounter

Jetzt könnte man natürlich auf die Idee kommen, den Schweißroboter durch Austausch der Lampe durch einen Photowiderstand in einen Lichtsuchroboter umzuwandeln und dann die gemessenen Lichtwerte samt Position in der Warteschleife auszugeben .....

FishFace-Referenz

Allgemeines

Verwendete Parameterbezeichnungen

In der Referenz werden für Parameter und Returnwerte besondere Bezeichnungen verwendet um deren Bedeutung zu charakterisieren. Sie stehen gleichzeitig für einen Variablentyp bzw. alternativ eine Enum.

AnalogNr Nummer eines Analog-Einganges (ftiInp 1-5)

AnalogWert Rückgabewert beim Auslesen von AX/AY... (0-1024)

AnalogZyklen Anzahl der Zyklen nach dem die Analogwerte ausgelesen werden (nur Intelligent Interface, Long, typisch 5)

Code Angabe mit welcher Code-Taste die Eingaben des IR-Senders ausgewertet werden sollen (ftiIRCode).

ComNr Nummer des COM-Ports für eine Interface-Verbindung (ftiPorts).

Counter Wert eines ImpulsCounters (Long)

Direction Schaltzustand eines M-Ausganges (Long 0-2, ftiDir)

ifTyp Typ des angeschlossenen Interfaces (ftiInterfaces)

InputNr Nummer eines I-Einganges (Long 0-8(16), ftiOut)

InputStatus Rückgabewert beim Auslesen aller I-Eingänge (0 - &HFFFFFFFF)

KeyNr Nummer der vom IR-Sender erwarteten Taste (ftiIRKeys)

LampNr Nummer eines O-Ausganges ("halben"-M-Ausganges) (Long 1-8(32), ftiOut)

ModeStatus Status der Betriebsmodi aller M-Ausgänge(Long). Jeweils 2 bit pro Ausgang. Begonnen bei 0-1 für M1 (00 normal, 01 RobMode).

MotorNr Nummer eines M-Ausganges (Long 0-4(16), ftiOut)

MotorStatus SollStatus aller M-Ausgänge (Long). Jeweils 2 bit pro Ausgang Begonnen bei 0-1 für M1 (00 = Aus, 01 = Links, 10 = Rechts)

mSek Zeitangabe in MilliSekunden (Long)

NrOfchanges Anzahl Impulse (Long)

OnOff Ein/Ausschalten eines M-Ausganges oder O-Ausganges (ftiDir)

Position Position in Impulsen ab Endtaster (Long)

SerialNr Standardseriennummer eines ROBO-Interfaces

Speed Geschwindigkeitsstufe mit der ein M-Ausgang (Motor) betrieben werden soll (Long 1-7, ftiSpeed)

SpeedStatus Status der Geschwindigkeiten aller M-Ausgänge (Long)
Jeweils 4 bit pro Ausgang. Begonnen bei 0-3 für M1
Werte 0000 (stop) – 1111 (full)

TermInputNr Nummer eines I-Einganges mit der die (Wait)Methode beendet werden soll (Long, ftiNr)

Value Allgemeiner Long Wert

VoltNr Nummer eine analogen Spannungseinganges (A1, A2, AV, ftiInp)

WaitWert Rückgabewert von WaitForMotors (Long, ftiWait)

Die Aufrufparameter werden ByVal (Ausnahme WaitForPosition, Parameter Counter) übergeben.

Eine Reihe von Parametern sind optional, sie werden dann meist im Sinne einer Funktions-Überladung (Overload). Das wird dann bei der betreffenden Methode besondersbeschrieben.

Symbolische Konstanten (Enums)

Um ein Programm lesbarer zu machen, werden eine Reihe von symbolischen Konstanten angeboten, die in Enums zusammengefaßt sind. Die Konstanten werden unter den folgenden Oberbegriffen zusammengefaßt :

ftiInterfaces Bezeichnung der anschließbaren Interfaces

ftiPorts Bezeichnung der unterstützten Ports

ftiDir Angabe des Schaltzustandes (Drehrichtung, Ein/Aus)

ftiInp Angabe der Nummer eines Einganges

ftiOut Angabe der Nummer eines Ausganges

ftiIRCode Auswertungsart beim IR Receiver

ftiIKeys Getätigte Taste am IR Receiver

ftiSpeed Angabe einer Geschwindigkeitsstufe

ftiWait Returnwerte der Methode WaitForMotors

Die Enums können als Parameter parallel zu den numerischen Parameterwerten (Long) angegeben werden.

Eigenschaften

ActDeviceName

Name des angeschlossenen Interfaces

ActDeviceType

Typ des angeschlossenen Interfaces

ActDeviceSerialNr

Standardseriennummer des angeschlossenen Interfaces

ActDeviceFirmware

Firmwarestand des angeschlossenen Interfaces

NotHalt

Anmelden eines Abbruchwunsches, Auswertung durch die Wait-Methoden und Finish

Get | Set, Boolean, Default = False

Outputs

Lesen Werte aller M-Ausgänge.

Get, Long | MotorStatus

Version

Lesen der Version von FishFa40AX.CLS/DLL

Get, String

Befehle

Allgemeines

Abbrechbar

Länger laufende Methoden (Wait...) können von außen durch Setzen der Eigenschaft NotHalt = True oder durch Drücken der Esc-Taste abgebrochen werden.

Wird bei den betroffenen Methoden besonders angegeben.

NotHalt

Die Eigenschaft NotHalt (siehe auch "Abbrechbar") wird intern genutzt um langlaufende Funktionen ggf. Abzubrechen. NotHalt wird von OpenInterface auf False gesetzt. Es kann im Programm (z.B. über einen HALT-Button) genutzt werden um den Programmlauf abzubrechen oder auch eine Endlosschleife (z.B. Do ... Loop Until ft.Finish) zu beenden. Soll das Programm anschließend weiterlaufen, so ist NotHalt wieder auf False zu setzen.

Speed

Die Geschwindigkeitssteuerung beruht auf einem zyklischen Ein- und Ausschalten der betroffenen M-Ausgänge (Motoren) im Takt des PollIntervals. Die Geschwindigkeitsstufe wird durch die Parameter Speed bzw. SpeedStatus für einen bzw. alle Motoren bei der Methode SetMotor(s) bestimmt.

Counter

Zu jedem I-Eingang wird ein Counter geführt, in dem die Impulse (Umschalten von True auf False und umgekehrt) gezählt werden. Die Counter werden bei OpenInterface auf 0 gesetzt. Sie werden außerdem von einigen Methoden interngenutzt (SetMotor, WaitForxxx). Sie können mit GetCounter abgefragt und mit SetCounter / ClearCounter(s) gesetzt werden. In der Regel werden sie zur Positionsbestimmung eingesetzt.

RobMotoren

Unter RobMotoren wird eine Kombination von einem M-Ausgang und zwei E-Eingängen mit den Funktionen Endtaster / Impulstaster verstanden, die im Betrieb eine Einheit bilden. Dabei muß am M-Ausgang ein Motor angeschlossen sein und an den E-Eingängen Taster mit Schließfunktion (Kontakte 1-3). Auf der Motorwelle muß ein "Impulsrädchen" montiert sein, das den Impulstaster betätigt. Der Motor muß linksdrehend angeschlossen werden. D.h. er läuft bei ftiLinks auf den Endtaster zu. Folgende Kombinationen sind zulässig

Motor	Endtaster	Impulstaster
1	1	2
2	3	4
3	5	6
4	7	8

Und so weiter bis Motor 16. Die RobMotoren können über die Methode
`SetMotor MotorNr, Direction, Speed, Counter`
betrieben werden. Die Methode erlaubt das simultane Anfahren vorgegebener Positionen mit

bis zu 8 Motoren. Bei Erreichen einer Position wird der zugehörende Motor abgeschaltet. Die Methode ist asynchron. D.h.Das Erreichen der vorgegebenen Positionen wird von der Methode nicht abgewartet (der Ausführungsteil der Methode läuft in einem separatem Thread). Die Synchronität zum Programm kann durch die Methode WaitForMotors wieder hergestellt werden.

Beispiel

```
ft.SetMotor ftiM1, ftiLeft, ftiHalf, 50
ft.SetMotor ftiM2, ftiRight, ftiFull, 60
ft.WaitForMotors 0, ftiM1, ftiM2
```

Motoren M1 und M2 werden gestartet, anschließend wird auf das Erreichen der Positionen gewartet.

O-Ausgänge am Interface

Die ROBO-Interfaces können die beiden Pole eines M-Ausganges einzeln schalten. Sie werden deswegen auch zusätzlich mit On bezeichnet. Geräte für die keinen Umpolung im laufenden Betrieb erforderlich ist (Lampen, Magnet aber auch teilweise Motoren) können an einen O-Ausgang und Masse angeschlossen werden. Dadurch wird die Schaltkapazität eines Interface deutlich erhöht. Geschaltet werden sie mit der Methode SetLamp.

Liste der Befehle

ClearCounter

Löschen des angegebenen Counters (0)

ClearCounter InputNr

Siehe auch ClearCounters, GetCounter, SetCounter

ClearCounters

Löschen aller Counter (0)

ClearCounters

Siehe auch ClearCounter, GetCounter, SetCounter

ClearLog

Das Log-Fenster der IDE wird gelöscht

ClearLog

Siehe PrintLog, PrintStatus

ClearMotors

Abschalten aller M-Ausgänge (ftiAus)

ClearMotors

Siehe auch SetMotor, SetMotors, SetLamp, Outputs

EA

Auslesen des Wertes im Feld EA der IDE

wert = **EA**

Beispiel

```
AnzahlRunden = EA + 1
```

Die Variable Anzahlrunden wird mit dem numerischen Inhalt des IDE Feldes EA + 1 besetzt.

Siehe auch EB

EB

Auslesen des Wertes im Feld EB der IDE

wert = **EB**

Siehe auch EA

Finish

Feststellen eines Endewunsches (NotHalt, Esc-Taste, I-Eingang)

Boolean = **Finish**(InputNr)

Siehe auch GetInput, GetInputs, Inputs

Beispiel

```
Do
   ....
Loop Until Finish(ftiI1)
```

Die Do Loop-Schleife wird solange durchlaufen, bis entweder NotHalt = True, die Esc-Taste gedrückt oder I1 = True wurde.

FinishIR

Feststellen eines Endewunsches (NotHalt, Esc-Taste, IRKey)

Boolean = **FinishIR**(Code, KeyNr)

Siehe auch GetInput, GetInputs, Inputs

Beispiel

```
Do
   ....
Loop Until FinishIR(ftiCode1, ftiM3L)
```

Die Do Loop-Schleife wird solange durchlaufen, bis entweder NotHalt = True, die Esc-Taste gedrückt oder M3L am IR-Sender – bei Code1 - gedrückt wurde.

GetAnalog

Feststellen eines Analogwertes (AX / AY / AXS1 / AXS2 / AXS3).
Es wird der intern vorliegende Wert ausgegeben. Beim Intelligent Interface ist die AnalogZyklen-Angabe beim OpenInterfaceCOM ist erforderlich.

AnalogWert = **GetAnalog**(AnalogNr)

Siehe auch GetAnalogs, AnalogsEX, AnalogsEY, AnalogScan, OpenInterfaceCOM

Beispiel

```
PrintStatus ft.Analog(ftiAX)
```

Der aktuelle AX-Wert wird in der Statuszeile angezeigt.

GetCounter

Feststellen des Wertes des angegebenen Counters

Counter = **GetCounter**(InputNr)

Siehe auch SetCounter, ClearCounter, ClearCounters

Beispiel

Beipiel

```
PrintStatus "Turm Position : " & GetCounter(ftiI2)
```

Die aktuelle Turm Position wird in der Statuszeile angezeigt.

GetInput

Feststellen des Wertes des angegebenen I-Einganges

Boolean = **GetInput**(InputNr)

Siehe auch GetInputs, GetIRKey, Finish, FInishIR

Beispiel

```
If GetInput(ftiI1) Then
   ...
Else
   ...
EndIf
```

Wenn der I-Eingang I1 (Taster, PhotoTransistor, Reedkontakt ...) = True ist, wird der Then-Zweig durchlaufen.

GetInputs

Feststellen der Werte aller E-Eingänge

InputStatus = **GetInputs**()

Siehe auch GetInput, GetIRKey, Finish, FinishIR

Beispiel

```
Dim e
   e = GetInputs
   If (e And &H1) Or (e And &H4) Then ...
```

Wenn die I-Eingänge I1 oder I3 True sind, wird der Then-Zweig ausgeführt.

GetIRKey

Feststellen des Wertes des angegebenen IR-Einganges. Die Code-Tasten der IR-Bedienung werden wahlweise ausgewertet.

Boolean = **GetIRKey**(Code, KeyNr)

Siehe auch GetInputs, GetInput, Finish, FinishIR

Beispiel

```
If GetIRKey(ftiCode1, ftiM2L) Then
   ...
Else
   ...
EndIf
```

Wenn die IR-Taste M2L = True ist und Code1 aktiv ist, wird der Then-Zweig durchlaufen.

GetVoltage

Feststellen des Wertes des angegebenen A-Einganges (A1, A2, AV). Beim Intelligent Interface ist die AnalogZyklenangabe beim OpenInterfaceCOM erforderlich.

Boolean = **GetVoltage**(VoltNr)

Siehe auch GetAnalog, OpenInterfaceCOM

Beispiel

```
PrintStatus GetVoltage(ftiA1)
```

In der Statuszeile wird der aktuelle Wert von A1 angezeigt.

NotHalt

Anmelden eines Abbruchwunsches, Auswertung durch die Wait-Methoden und Finish

Get | Set, Boolean, Default = False

Beispiel

```
If NotHalt Then Exit Sub
```

Das Unterprgramm wird beendet, wenn NotHalt den Wert True hat

Siehe auch Finish

Pause

Anhalten des Programmablaufs.

Pause mSek

Siehe auch WaitForTime

Beispiel

```
SetMotor ftiM1, ftiLinks
```

```
Pause 1000
SetMotor ftiM1, ftiAus
```

Der Motor am M-Ausgang M1 wird für eine Sekunde (1000 MilliSekunden) eingeschaltet.

PrintStatus

Ausgabe eines Textes in die Statuszeile der IDE

PrintStatus stringausdruck

Beispiel

```
PrintStatus "Temperatur : " & (1000 - GetAnalog(aNTC)) / 10 - 12
```

In die Statuszeile der IDE wird Temperatur : 37,4 ausgegeben. GetAnalog(aNTC) ergab den Wert 506.

Siehe auch Debug.Print (VBA Kurzreferenz)

SetCounter

Setzen eines Counters

SetCounter InputNr, Value

Siehe auch GetCounter, ClearCounter, ClearCounters

SetLamp

Setzen eines O-Ausganges ("halben" M-Ausganges, nur ROBO-Interface). Anschluß einer Lampe oder eines Magneten ... an einen Kontakt eines M-Ausganges und Masse. Siehe auch "O-Ausgänge am Interface"

SetLamp LampNr, OnOff, Power

Der Parameter Power ist optional (default = 7)

Siehe auch SetMotor, SetMotors, ClearMotors

Beispiel

```
Const lGruen = ftiO1, lGelb = ftiO2, lRot = ftiO3

  SetLamp lGruen, ftiEin
  Pause 2000
  SetLamp lGruen, ftiAus
  SetLamp lGelb, ftiEin
```

Die grüne Lampe an O1 und Masse wird für 2 Sekunden eingeschaltet und anschließend die gelbe an O2 ...

SetMotor

Setzen eines M-Ausganges (Motor)

SetMotor MotorNr, Direction, Speed, Counter

Die Parameter ab Speed sind optional

MotorNr (ftiOut) : Nummer des zu schaltenden M-Ausganges

Direction (ftiDir) : Schaltzustand (ftiLinks, ftiRechts, ftiEin, ftiAus)

Speed (ftiSpeed) : Geschwindigkeitsstufe, Default : ftiFull

Counter (ftiNr) : Begrenzung der Einschaltzeit. Default = 0, unbegrenzt. Werte > 0 geben die Anzahl Impulse an, die der M-Ausgang eingeschaltet sein soll (Fahren eines Motors um n Impulse). Siehe auch "RobMotoren"

Siehe auch SetMotors, ClearMotors, SetLamp, Outputs

Beispiel 1

```
SetMotor ftiM1, ftiRechts, ftiFull
Pause 1000
SetMotor ftiM1, ftiLinks, ftiHalf
Pause 1000
SetMotor ftiM1, ftiAus
```

Der Motor am M-Ausgang M1 wird für 1000 MilliSekunden rechtsdrehend, volle Geschwindigkeit eingeschaltet und anschließend für 1000 mSek linksdrehend, halbe Geschwindigkeit.

Beispiel 2

```
SetMotor ftiM1, ftiLeft, 4, 123
```

Der Motor am M-Ausgang M1 wird für 123 Impulse am I-Eingang I2 oder I1 = True mit Geschwindigkeitstufe 4 eingeschaltet. Das Abschalten erfolgt selbsttätig.

SetMotors

Setzen des Status aller M-Ausgänge

SetMotors MotorStatus, SpeedStatus, SpeedStatus16, ModeStatus

Die Parameter ab SpeedStatus sind optional

MotorStatus (Long) : Schaltzustand der M-Ausgänge

SpeedStatus, SpeedStatus16 (Long) : Geschwindigkeitsstufen der M-Ausgänge.
Default : ftiFull

ModeStatus (Long) : Betriebsmodus der M-Ausgänge. Default = 0, normal
Bei ModeStatus RobMode sind vorher mit SetCounter die zugehörenden Counterstände zu setzen.

Siehe auch SetCounter, ClearMotors, SetMotor, SetLamp, Outputs

Beispiel

```
SetMotors &H1 + &H80
Pause 1000
ClearMotors
```

Der M-Ausgang (Motor) M1 wird auf links geschaltet und gleichzeitig M4 auf rechts. Alle anderen Ausgänge werden ausgeschaltet. Nach 1 Sekunde werden alle M-Ausgänge abgeschaltet.

WaitForChange

Warten auf eine vorgegebene Anzahl von Impulsen

WaitForChange InputNr, NrOfChanges, TermInputNr

Der Parameter TermInputNr ist optional

InputNr (ftiInp) : I-Eingang an dem die Impulse gezählt werden.

NrOfChanges (Long) : Anzahl Impulse

TermInputNr (ftiInp) : Alternatives Ende. I-Eingang = True

Intern wird Counter (InputNr) verwendet, der zu Beginn der Methode zurückgesetzt wird

Siehe auch WaitForPositionDown, WaitForPositionUp, WaitForInput, WaitForLow, WaitForHigh

Beispiel

```
SetMotor ftiM1, ftiLeft
WaitForChange ftiI2, 123, ftiI1
SetMotor ftiM1, ftiOff
```

Der M-Ausgang (Motor) M1 wird linksdrehend geschaltet, es wird auf 123 Impulse an I-Eingang I2 oder I1 = True gewartet, der Motor wird abgeschaltet. Vergleiche mit dem Beispiel bei SetMotor. Hier wird das Programm angehalten solange der Motor läuft.

WaitForHigh

Warten auf einen False/True-Durchgang an einem E-Eingäng

WaitForHigh InputNr

Siehe auch WaitForLow, WaitForChange, WaitForInput

Beispiel

```
SetMotor ftiM1, ftiOn
SetMotor ftiM2, ftiLeft
WaitForHigh ftiI1
SetMotor ftiM2, ftiOff
```

Eine Lichtschranke mit Lampe an M-Ausgang M1 und Phototransistor an I-Eingang I1 wird eingeschaltet. Ein Förderband mit Motor an M2 wird gestartet, es wird gewartet bis ein Teil auf dem Förderband aus der Lichtschranke ausgefahren ist (die Lichtschranke wird geschlossen), dann wird abgeschaltet. Die Lichtschranke muß vorher False sein (unterbrochen).

WaitForInput

Warten daß der angegebene I-Eingang den vorgegebenen Wert annimmt.

WaitForInput InputNr, OnOff

OnOff (Boolean) : Endebedingung für I-Eingang InputNr, Default = True

Siehe auch WaitForChange, WaitForLow, WaitForHigh

Beispiel

```
SetMotor ftiM1, ftiLeft
WaitForInput ftiI1
SetMotor ftiM1, ftiOff
```

Der Motor an M-Ausgang M1 wird gestartet, es wird auf I-Eingang = True gewartet, dann wird der Motor wieder abgeschaltet : Anfahren einer Endposition.

WaitForInputIR

Warten daß der angegebene IR-Eingang den vorgegebenen Wert annimmt.

WaitForInput Code, KeyNr, OnOff

OnOff (Boolean) : Endebedingung für IR-Eingang KeyNr, Default = ftiEin

Siehe auch WaitForChange, WaitForLow, WaitForHigh

Beispiel

```
SetMotor ftiM1, ftiLeft
WaitForInput ftiI1
SetMotor ftiM1, ftiOff
```

Der Motor an M-Ausgang M1 wird gestartet, es wird auf I-Eingang = True gewartet, dann wird der Motor wieder abgeschaltet : Anfahren einer Endposition.

WaitForLow

Warten auf einen True/False-Durchgang an einem I-Eingang

WaitForLow InputNr

Siehe auch WaitForChange, WaitForInput, WaitForHigh

Beispiel

```
SetMotor ftiM1, ftiOn
SetMotor ftiM2, ftiLeft
WaitForLow ftiI1
SetMotor ftiM2, ftiOff
```

Eine Lichtschranke mit Lampe an M-Ausgang M1 und Phototransistor an I-Eingang I1 wird eingeschaltet. Ein Förderband mit Motor an M2 wird gestartet, es wird gewartet bis ein Teil auf dem Förderband in die Lichtschranke einfährt (sie unterbricht), dann wird abgeschaltet. Die Lichtschranke muß vorher True sein (nicht unterbrochen).

WaitForMotors

Warten auf ein MotorReadyEreignis oder den Ablauf von Time

WaitWert = **WaitForMotors**(Time, MotorNr .....)

Time (Long) : Zeit in MilliSekunden. Bei Time = 0 wird endlosgewartet.

MotorNr (ftiNr) : Nummern der M-Ausgänge auf die gewartet werden soll. Es wird auf MotorStatus = ftiAus für die angegebenen M-Ausgänge gewartet. MotorNr ftiM1 – ftiM16 in beliebiger Reihenfolge (max 8). Die nicht betroffenen Motoren müssen nicht angegeben werden.
Bei den Return-Werten ftiWait.ftiNotHalt und .ftiESC werden alle betroffen Motoren angehalten.

Beispiel

```
SetMotor ftiM4, ftiLeft,  ftiHalf, 50
SetMotor ftiM3, ftiRight, ftiFull, 40
Do
  PrintStatus GetCounter(ftiI6) & " - " & GetCounter(ftiI8)
Loop While WaitForMotors(100, ftiM4, ftiM3) = ftiTime
```

Der Motor am M-Ausgang M4 wird linksdrehend mit halber Geschwindigkeit für 50 Impulse gestartet, der an M3 rechtsdrehend mit voller Geschwindigkeit für 40 Impulse. Die Do Loop-Schleife wartet auf das Ende der Motoren (WaitForMotors). Alle 100 MilliSekunden wird in der Schleife die aktuelle Position angezeigt (100 .... = ftiTime). Wenn die Position erreicht ist <> ftiTime, ist der Auftrag abgeschlossen, die Motoren haben sich selber beendet. Achtung hier wurde nicht auf NotHalt (ftiNotHalt) oder Esc-Taste (ftiEsc) abgefragt, es könnte also auch vor Erreichen der Zielposition abgebrochen worden sein.

WaitForPosDown

Warten auf Erreichen einer vorgegebenen Position.

WaitForPosDown InputNr, IstPosition, ZielPosition, TermInputNr

Ausgegangen wird von der aktuellen Position, die in IstPosition gespeichert ist, es werden solange Impulse von IstPosition abgezogen, bis der in ZielPosition angegebene Stand erreicht ist. IstPosition enthält dann tatsächlich erreichte Position (kann um einen Wert höher liegen, wenn der Motor nochmal "geruckt" hat). Alternativ wird die Methode durch I-Eingang TermInputNr = True beendet. IstPosition und ZielPosition müssen immer positive Werte (einschl. 0) enthalten.

Siehe auch WaitForPosUp, WaitForChange

Beispiel

```
Dim IstPosition
  IstPosition     = 12
  SetMotor       ftiM1, ftiLinks
  WaitForPosDown ftiI2, IstPosition, 0
  SetMotor       ftiM1, ftiAus
```

Die aktuelle Position ist 12 (IstPosition), der Motor an M-Ausgang M1 wird linksdrehend gestartet. WaitForPosDown wartet dann auf Erreichen der Position 0, der Motor wird dann ausgeschaltet

WaitForPosUp

Warten auf Erreichen einer vorgegebenen Position.

WaitForPosUp InputNr, IstPosition, ZielPosition, TermInputNr

Ausgegangen wird von der aktuellen Position in IstPosition, es werden solange Impulse auf IstPosition addiert, bis der in ZielPosition angegebene Stand erreicht ist. IstPosition enthält dann tatsächlich erreichte Position (kann um einen Wert höher liegen, wenn der Motor nochmal "geruckt" hat). Alternativ wird die Methode durch I-Eingang TermInputNr = True beendet. IstPosition und ZielPosition müssen immer positive Werte (einschl. 0) enthalten.

Siehe auch WaitForPosDown, WaitForChange

Beispiel

```
Dim IstPosition
  IstPostion  = 0
  SetMotor     ftiM1, ftiRechts
  WaitForPosUp ftiI2, IstPosition, 24
   SetMotor     ftiM1, ftiAus
```

Die aktuelle Position ist 0 (IstPosition), der Motor an M-Ausgang M1 wird rechtsdrehend gestartet. WaitForPosUp wartet dann auf Erreichen der Position 24, der Motor wird dann ausgeschaltet. Siehe auch Beispiel zu WaitForPosDown, hier wird in Gegenrichtung gefahren.

WaitForTime

Anhalten des Programmablaufs.

WaitForTime mSek

Synonym für Pause

Siehe auch Pause

Beispiel

```
Do
  SetMotors &H1
  WaitForTime 555
  SetMotors &H4
  WaitForTime 555
Loop Until Finish
```

In der Schleife Do Loop Until Finish wird erst M-Ausgang (Lampe) M1 eingeschaltet und alle anderen abgeschaltet (binär : 0001), dann gewartet, M2 (Lampe) eingeschaltet (Rest aus, binär : 0100) und gewartet. Ergebnis ein Wechselblinker.

Anhang

VBA-Kurzreferenz

Referenz der im Kapitel "Einführung in die Programmierung" verwendeten Befehle. Weitere Befehle sind der (englischen) Hilfe-Datei oder der deutschen Literatur über VBA zu entnehmen. Unter "Siehe auch" werden auch Hinweise auf Anweisungen und Funktionen aufgeführt, die nur in der englischen Dokumentation aufgeführt sind. Ebenso wurden in einigen Fällen Details weggelassen, die im Abschnitt "Einführung in die Programmierung" nicht besprochen wurden.

Datentypen

VBA kennt eine Reihe von Datentypen, die Inhalt und Aufbau von Variablen beschreiben, die wichtigsten sind hier aufgelistet :

Bezeichnung	Kurzzeichen	Erklärung
Integer	%	Ganzzahl von 2 Byte Länge
Long	&	Ganzzahl von 4 Byte Länge
Single	!	Fließkomma, einfache Genauigkeit, 4 Bytes
Double	#	Fliesßkomma, doppelte Genauigkeit, 8 Bytes
Date		Datum, 8 Bytes
String	$	Zeichenfolge, 10 Byte + Textlänge
Boolean		Boolscher Wert, 2 Bytes
Variant		Defaultwert

Call Anweisung

Syntax

```
Call name[(parameterliste)]
-oder-
name [parameterliste]
```

Beschreibung Übergibt die Steuerung an eine Sub- oder Function-Prozedur. Das Schlüsselwort Call kann entfallen, in diesem Fall müssen auch die Klammern entfallen.

Siehe auch **Declare**, **Sub**.

Beispiel

```
Sub Anzeigen(Titel$, Wert)
    Debug.Print Titel; "="; Wert
End Sub

Sub Main
    Call Anzeigen("2000/9",2000/9)  ' --- 222.2222222222
    Anzeigen "1<2",1<2                  ' --- True
End Sub
```

Const Definition

Syntax

```
Const name[typ] [As Typ] = ausdruck[, ...]
```

Beschreibung Deklariert name und wert einer Konstante. Im Ausdruck können einfache konstante Werte und VBA-Funktionen verwendet werden.

Beispiel

```
Sub Main
    Const mLampe = 4, Pi = 4*Atn(1), e = Exp(1)
    Debug.Print Pi                ' --- 3.14159265358979
    Debug.Print e                 ' --- 2.71828182845905
    Debug.Print mLampe            ' --- 4
End Sub
```

Debug Objekt

Syntax

```
Debug.Clear
-oder-
Debug.Print [ausdruck[; ...][;]]
```

Beschreibung Form 1: Löschen der "Schnell"-Fensters im Protokoll-Bereich.

Form 2: Ausgeben (Anzeigen) der Ausdrucksliste im Schnell-Fenster. Die einzelnen Ausdrücke werden durch ";" getrennt. Ein ";" am Ende der Anweisung verhindert einen Zeilenvorschub, d.h. das nächste Debug.Print wird in die gleiche Zeile ausgegeben. Sonst wird jedes Debub.Print in eine eigene Zeile ausgegeben. Alternativ können auch einzelne Ausdrücke durch "&" zu einem einzigen Ausdruck verbunden werden. Das wurde in der "Einführung" gemacht.

Beispiel

```
Sub Main
    X = 4
    Debug.Print "X/2="; X/2    ' --- 2
    Debug.Print "Start...";    ' --- Unterdrücken eines Zeilenvorschubs
    Debug.Print "Finish"       ' ---
    Debug.Print "Wert von X : " & X
End Sub
```

Dim Definition

Syntax

```
Dim name[typ][([dim[, ...]])][As typ][, ...]
```

Beschreibung Deklariert Variablen und reserviert Speicherplatz dafür. Es können einfache (skalare) Variable, aber auch Variablen-Bereiche (Arrays) deklariert werden. Bei Bereichen muß die Obergrenze angegeben werden, die Zählung beginnt bei 0 (Ausnahme siehe Option Base). Die Wert werden mit 0 vorbesetzt. Den Variablen und Bereichen sollten Datentypen (& | Long, ! | Single, $ | String, Boolean) zugewiesen werden. Ist das nicht der Fall, wird der allgemeine Datentyp Variant angenommen.

Siehe auch **Option** Base.

Beispiel

```
Sub MachWas()
    Dim C0,C1(5),C2(2,3)
    C0 = 1
    C1(0) = 2
    C2(0,0) = 3
    Debug.Print C0; C1(0); C2(0,0)     ' --- 1 2 3
End Sub

Sub Main
    MachWas
End Sub
```

Do Anweisung

Syntax

```
Do
    anweisungen
Loop
-oder-
Do {Until|While} bedingung
    anweisungen
Loop
-oder-
Do
    anweisungen
Loop {Until|While} bedingung
```

Beschreibung Form 1: Endlosschleife. Die Schleife kann durch Exit Do verlassen werden.

Form 2: Schleife bei der die Abbruch-Bedingung vor der Ausführung eines Schleifendurchlaufs gepüft wird. Trift die Bedingung schon vor dem ersten Durchlauf nicht zu, wird die Schleifen nicht durchlaufen.

Form 3: Schleife bei der die Abbruch-Bedingun nach der Ausführung eines Schleifendurchlaufs geprüft wird. Die Schleife wird also mindestens einmal durchlaufen.

Beenden einer Schleife :

- Until *bedingung*: Beenden der Schleife, wenn die Until Bedingung **True** wird.
- While *bedingung*: Durchführen der Schleife solange die While Bedingung **True** ist.
- Exit Do. Wenn die Anweisung Exit Do erreicht wird.

Siehe auch **For**, **Exit** Do, **While**.

Beispiel

```
Sub Main
    I = 2
    Do
        I = I*2
    Loop Until I > 10
    Debug.Print I        ' --- 16
End Sub
```

End Anweisung

Syntax

```
End
```

Beschreibung Die End Anweisung beendet das Programm sofort.

Siehe auch **Stop**.

Beispiel

```
Sub ActionSub
    L = InputBox$("Bitte ENDE eingeben")
    If L = "ENDE" Then End
    Debug.Print "Es wurde kein ENDE eingegeben"
End Sub

Sub Main
    Debug.Print "Vor ActionSub"
    ActionSub
    Debug.Print "Nach ActionSub"
End Sub
```

Exit Anweisung

Syntax

```
Exit {Do|For|Function|Sub}
```

Beschreibung Die Exit Anweisung veranlaßt das Programm, einige oder alle nachfolgenden Anweisungen zu überspringen.

Exit	Beschreibung
Do	Sofortiges Verlassen eines Do..Loops
For	Sofortiges Verlassen einer For..Next Schleife
Function	Sofortiges Verlassen einer Function.
Sub	Sofortiges Verlassen einer Sub

Beispiel

```
Sub Main
   Dim L$
   L = InputBox$("Bitte Do, For, Sub eingeben")
   Debug.Print "Vor ActionSub"
   ActionSub L
   Debug.Print "Nach Action Sub"
End Sub

Sub ActionSub(L$)
   Do
      If L = "Do" Then Exit Do
      I = I+1
   Loop While I < 10
   If I = 0 Then Debug.Print "Do wurde eingegeben"

   For I = 1 To 10
      If L = "For" Then Exit For
   Next I
   If I = 1 Then Debug.Print "For wurde eingegeben"

   If L = "Sub" Then Exit Sub
   Debug.Print "Sub wurde NICHT eingegeben"
End Sub
```

For Anweisung

Syntax

```
For num = erster To letzter [Step schrittweite]
    anweisungen
Next [num]
```

Beschreibung Ausführen der anweisungen solange num im Bereich erster .. letzter (einschließlich) ist..

Parameter	Beschreibung
Num	Iterationsvariable
Erster	Anfangswert von Num
Letzter	Endwert von Num.
Schrittweite	Schrittweite in der Num nach jedem Schleifendurchlauf erhöht wird, ohne Angabe = 1. Die Schleife wird beendet wenn Num größer oder gleich Letzter ist.

Siehe auch **Do**, **Exit** For

Beispiel

```
Sub Main
    For I = 1 To 2000 Step 100
        Debug.Print I; I + I; I * I
    Next I
End Sub
```

Function Definition

Syntax

```
Function name[typ][([parameter[, ...]])] [As typ[()]]
    anweisungen
End Function
```

Beschreibung Deklariert den Namen, die Parameter und den Code für eine Funktions-Prozedur. Eine Funktion liefert als Ergebnis einen Funktionswert, der dem Namen der Funktion zugewiesen wurde.

Siehe auch **Declare**, **Sub**.

Beispiel

```
Function Hoch(X,Y)
    P = 1
    For I = 1 To Y
        P = P*X
    Next I
    Hoch = P
End Function

Sub Main
    Debug.Print Hoch(2,8)      ' --- 256
End Sub
```

If Anweisung

Syntax

```
If bedingung Then [anweisung] [Else anweisung]
-oder-
If bdedingung Then
    anweisungen
[ElseIf bedingung Then
    anweisungen]...
[Else
    anweidungen]
End If
```

Beschreibung Form 1: Einzeilige Anweisung. Wenn die Bedingung zutrifft, wird die Anweisung nach Then, wenn nicht die Anweisung nach Else ausgeführt. Der Else-Zweig ist optional.

Form 2: Mehrzeilige Anweisung, Ablauf wie Form 1. Zusätzlich können über ElseIf weitere Bedingungen angegeben werden. Trifft keine der angegebenen Bedingungen zu, wird der Else-Zweig ausgeführt.

Siehe auch **Select Case**, **Choose( )**, **Iif( )**.

Beispiel

```
Sub Main
    S = InputBox("Bitte Hallo, Tschüs, Mahlzeit, Salve, Grüzi
eingeben")
    If S = "Hallo" Then Debug.Print "Herein spaziert"
    If S = "Tschüs" Then Debug.Print "see you later"
    If S = "Mahlzeit" Then
        Debug.Print "Bitte einzutreten"
        Debug.Print "Sind gleich soweit"
    ElseIf S = "Salve" Then
        Debug.Print "dto."
        Debug.Print "Aber heute is nix"
    Else
        Debug.Print "Versteh ich nich"
    End If
End Sub
```

InputBox Funktion

Syntax

```
InputBox[$](nachricht$[, Titel$][, default$][, XPos, YPos])
```

Beschreibung

Anzeige einer Inputbox, die eine Eingabe durch den Bediener erlaubt. Bei Abschluß durch den OK-Button wird die Eingabe an das Programm übergeben, bei Abschluß durch Abbrechen wird ein leerer String übergeben.

Parameter	Beschreibung
nachricht	Anzeigetext in der Inputbox
Titel	Anzeigetext in der Titelzeile der Inputbox
default	Vorgabewert für die Eingabe, wenn nicht vorhanden ist er leer.
XPos / YPos	Position der linken oberen Ecke auf dem Bildschirm in Pixeln, wenn ausgelassen wird die Inputbox zentiert angezeigt.

Beispiel

```
Sub Main
    L = InputBox$("Bitte Name und Titel eingeben", _
           "Input Box Beispiel","Prof. Dr. Giselher von Sorge")
    Debug.Print L
End Sub
```

MsgBox Anweisung / Funktion

Syntax

```
MsgBox nachricht$[, typ][, titel$]
-oder-
MsgBox(nachricht$[, typ][, titel$])
```

Beschreibung

Anzeige einer NachrichtenBox mit dem Titel *titel.* Dabei wird über *typ* das Erscheinungsbild und die dazugehörenden Quittungs-Buttons beschrieben.
In der Variante als Funktion wird als Ergebnis der getätigte Button zurückgegeben.

Ergebnis	Wert	Button
vbOK	1	OK Button
vbCancel	2	Abbrechen Button
vbAbort	3	Abort button
vbRetry	4	Retry button
vbIgnore	5	Ignore button
vbYes	6	Ja Button
vbNo	7	Nein Button

Parameter	Beschreibung
nachricht	Anzeigetext in der NachrichtenBox
typ	Typ der Nachrichtenbox, siehe Tabelle. Die Werte der Einzeltabellen können addiert werden.
Titel	Anzeigetext in der Titelzeile der Box.

Button	Wert	Ergebnis
vbOkOnly	0	OK button
vbOkCancel	1	OK and Cancel buttons
vbAbortRetryIgnore	2	Abort, Retry, Ignore buttons
vbYesNoCancel	3	Yes, No, Cancel buttons
vbYesNo	4	Yes and No buttons
vbRetryCancel	5	Retry and Cancel buttons

Icon	Wert	Ergebnis
	0	No icon
vbCritical	16	Stop icon
vbQuestion	32	Question icon

vbExclamation	48	Attention icon
vbInformation	64	Information icon

Default	Wert	Ergebnis
vbDefaultButton1		
	0	First button
vbDefaultButton2		
	256	Second button
vbDefaultButton3		
	512	Third button

Mode	Value	Effect
vbApplicationModal		
	0	Application modal
vbSystemModal		
	4096	System modal
vbMsgBoxSetForeground		
	&h10000	System modal

Beispiel

```
Sub Main
    MsgBox "Bitte OK Button drücken"
    If MsgBox("Bitte einen Knopf drücken", _
                vbOkCancel+vbExclamation+vbDefaultButton1, _
                "Knopf-Test") = vbOK Then
        Debug.Print "OK wurde gedrückt"
    Else
    Debug.Print "Abbrechen wurde gedrückt"
    End If
End Sub
```

Now Funktion

Syntax `Now`

Beschreibung Gibt das aktuelle Datum und die Uhrzeit im Date Format zurück

Siehe auch **Date**, **Time**, **Timer**.

Beispiel

```
Sub Main
    Debug.Print "Heute ist : "; Now
End Sub
```

Operatoren

Syntax

```
^ Not * / \ Mod + - & < <= > >= = <> Is And Or Xor Eqv Imp
```

Beschreibung

Die angeführten Operatoren sind auf die Zahlen n1 und n2 bzw. die Strings s1 und s2 anwendbar. Die Operatoren haben eine Vorrangregelung in der Reihenfolge der Liste. Die Vorrangregelung kann durch Klammern geändert werden.

Operator	Beschreibung
- *n1*	Kehrwert von *n1*.
n1 ^ *n2*	Esponation *n1* hoch *n2*.
n1 * *n2*	Multiplikation *n1* mal *n2*.
n1 / *n2*	Division *n1* durch *n2*.
n1 \ *n2*	Division n1 durch n2. Zurückgegeben wird der Festkommaanteil der Division. n1 und n2 werden vor der Division aufgerundet.
n1 Mod *n2*	Teilungsrest der FestkommaDivision *n1 durch n2*.
n1 + *n2*	Addition *n1* plus *n2*.
s1 + *s2*	Verbinde String *s1* mit *s2*.
n1 - *n2*	Subtrahiere *n1* von *n2*.
s1 & *s2*	Verbinde *s1* with *s2*. (Auch s1 & n1 ist möglich).
n1 < *n2*	Return **True** wenn *n1* kleiner als *n2*.
n1 <= *n2*	Return **True** wenn *n1* kleiner oder gleich *n2*.
n1 > *n2*	Return **True** wenn *n1* größer als *n2*.
n1 >= *n2*	Return **True** wenn *n1* größer oder gleich *n2*.
n1 = *n2*	Return **True** wenn *n1* gleich *n2*.
n1 <> *n2*	Return **True** wenn *n1* ungleich *n2*.
s1 < *s2*	Return **True** wenn *s1* kleiner *s2*.
s1 <= *s2*	Return **True** wen *s1* kleiner oder gleich *s2*.
s1 > *s2*	Return **True** wenn *s1* größer *s2*.
s1 >= *s2*	Return **True** wenn *s1* größer oder gleich *s2*.
s1 = *s2*	Return **True** wenn *s1* gleich *s2*.
s1 <> *s2*	Return **True** wenn *s1* ungleich *s2*.
Not *n1*	Bitweise Ukehrung des Intergerwertes von *n1*.
n1 And *n2*	Bitweises And
n1 Or *n2*	Bitweises Or
n1 Xor *n2*	Bitweises Exclusiv Or
n1 Eqv *n2*	Bitweises Equivalence
n1 Imp *n2*	Bitweise Implication

Beispiel

```
Sub Main
    Dim N1 as Long, N2 as Long, S1 as String, S2 as String
    N1 = 10
    N2 = 3
    S1 = "asdfg"
    S2 = "hjkl"
    Debug.Print -N1            ' ---  -10
    Debug.Print N1 ^ N2        ' --- 1000
    Debug.Print Not N1         ' ---  -11
    Debug.Print N1 * N2        ' ---   30
    Debug.Print N1 / N2        ' ---    3.3333333333333
    Debug.Print N1 \ N2        ' ---    3
    Debug.Print N1 Mod N2      ' ---    1
    Debug.Print N1 + N2        ' ---   13
    Debug.Print S1 + S2        ' --- "asdfghjkl"
    Debug.Print N1 - N2        ' ---    7
    Debug.Print N1 & N2        ' ---  "103"
    Debug.Print N1 < N2        ' --- False
    Debug.Print N1 <= N2       ' --- False
    Debug.Print N1 > N2        ' --- True
    Debug.Print N1 >= N2       ' --- True
    Debug.Print N1 = N2        ' --- False
    Debug.Print N1 <> N2       ' --- True
    Debug.Print S1 < S2        ' --- True
    Debug.Print S1 <= S2       ' --- True
    Debug.Print S1 > S2        ' --- False
    Debug.Print S1 >= S2       ' --- False
    Debug.Print S1 = S2        ' --- False
    Debug.Print S1 <> S2       ' --- True
    Debug.Print N1 And N2      ' ---  2
    Debug.Print N1 Or N2       ' ---  11
    Debug.Print N1 Xor N2      ' ---  9
    Debug.Print N1 Eqv N2      ' ---  -10
    Debug.Print N1 Imp N2      ' ---  -9
End Sub
```

Option Definition

Syntax

```
Option Base [0|1]
-oder-
Option Compare [Binary | Text]
-oder-
Option Explicit
```

Beschreibung

Form 1: Setzen des Defaultwertes für die Array-Untergrenze. Defaultwert ist 0.

Form 2: Setzen des Defaultwertes für einen Stringvergleich.

- Option Compare Binary – Verwendung der internen Darstellung (default)
- Option Compare Text – Verwendung der aktuellen Sortierfolge.

Die Stringvergleiche mit <, <=, =, >, >=, <>, **Like** und **StrComp** sind von dieser Einstellung betroffen.

Form 3: Alle Variablen sind vor ihrem ersten Gebrauch mit Dim zu deklarieren.

Beispiel

```
Option Base 1
Option Explicit

Sub Main
    Dim A
        Dim C(2)       '  --- entspricht Dim C(1 To 2)
        Dim D(0 To 2)
    A = 1
    B = 2               ' --- B wurde nicht deklariert
End Sub
```

Select Case Anweisung

Syntax

```
Select Case ausdruck
[Case caseausdruck[, ...]
    anweisungen]...
[Case Else
    anweisungen]
End Select
```

Beschreibung Wählt den den zutreffenden Fall durch Vergleich des ausdrucks mit jedem der caseausrücke. Die Anweisungen des ersten zutreffenden werden ausgeführt. Trift keiner zu, wird Case Else ausgeführt. Fehlt Case Else wird bei Fehlen eines zutreffenden Falls die gesamte Case Anweisung übersprungen.

caseausdruck	Beschreibung
ausdruck	Ausführen bei Gleichheit.
Is < *ausdruck*	Ausführen ,wenn kleiner
Is <= *ausdruck*	Ausführen, wenn kleiner oder gleich
Is > *ausdruck*	Ausführen, wenn größer
Is >= *ausdruck*	Ausführen, wenn größer oder gleich.
Is <> *ausdruck*	Ausführen, wenn ungleich.
ausdr1 To ausdr*2*	Ausführen, wenn größer gleich ausdruck1 und kleiner gleich ausdruck2.

Siehe auch **If**, **Choose( )**, **Iif( )**.

Beispiel

```
Sub Main
    S = InputBox("Bitte  Hallo, Tschüs, Mahlzeit oder Nacht eingeben")
    Select Case S
    Case "Hallo"
        Debug.Print "Bitte einzutreten"
    Case "Tschüs"
        Debug.Print "see you later"
    Case "Mahlzeit"
        Debug.Print "come in"
        Debug.Print "Dinner will be ready soon."
    Case "Nacht"
        Debug.Print "Sorry."
        Debug.Print "Wir sind ausgebucht"
    Case Else
        Debug.Print "Was is?"
    End Select
End Sub
```

Stop Anweisung

Syntax

```
Stop
```

Beschreibung Anhalten der Programmausführung. Wenn die Ausführung wieder aufgenommen wird, wird mit der nächsten Anweisung fortgefahren.

Siehe auch **End**

Beispiel

```
Sub Main
    For I = 1 To 10
        Debug.Print I
        If I = 3 Then Stop
    Next I
End Sub
```

Sub Definition

Syntax

```
Sub name[([parameter[, ...]])]
    statements
End Sub
```

Beschreibung Deklariert den Namen, die Parameter und den Code für eine Sub-Prozedur. Eine Sub liefert im Gegensatz zur Function kein Ergebnis zurück, kann aber die Aufrufparameter verändern.

Siehe auch **Declare**, **Function**.

Beispiel

```
Sub Ausgabe (W1, W2, W3)
   Debug.Print W1; " - "; W2; " - "; W3
End Sub

Sub Plus(W1, W2)
  W1 = W1 + W2
  Debug.Print W1; W2
End Sub

Sub Main
  Dim W1&, W2&
  Ausgabe 1, 2, 3
  W1 = 45
  W2 = 82
  Debug.Print "Vor Plus : "; W1
  Plus W1, W2
  Debug.Print"Nach Plus : "; W1
End Sub
```

Übergang zu anderen Sprachen der VB-Sprachfamilie

Ein Übergang zu anderen Sprachen lohnt :

- wenn man seine Programme mit einer eigenen Oberfläche ausstatten will,
- wenn man komplexere Programme (also über die oben angegebenen "paar" Seiten hinauskommt) erstellen will,
- Programme wünscht, die ohne die IDE von vbaFish laufen
- oder man Wert auf eine komfortable Entwicklungsumgebung legt.

Dabei ist generell zu beachten, daß in diesem Fall eine Instanz von FishFa40AX.DLL selber zu erstellen ist z.B mit Set ft = CreateObject(FishFa40AX.FishFace). Die Instanzvariable – hier ft – ist dann allen FishFace-Befehlen voranzustellen.

Die Befehle, die direkt auf die IDE zugreifen Debug.Print, Debug.Clear, PrintStatus, EA / EB)sind durch geeignete der jeweiligen Sprache zu ersetzen. Die Befehle WaitForPosUP/Down sind in WaitForPositionUP/Down zu ändern.

Zu beachten ist, das die Einarbeitung in die neue Entwicklungsumgebung – Ausnahme VBScript pur – einigen Aufwand verlangen wird. Die Sprache Visual Basic und damit die in ihr geschriebenen Programme, bleiben erhalten.

Beispiel für Programme in anderen Sprachen finden sich auf www.ftcomputing.de

VBScript pur

Ist ohne Installation neuer Software möglich, lohnt aber nur, wenn man Programme ohne eigene Oberfläche erstellen will, sie laufen dann "unsichtbar" im Hintergrund. Zu beachten ist, daß sich die Befehle PrintStatus und Debug.Print schlecht ersetzen lassen.

VBScript und HTML

Wenn man Erfahrung mit der Erstellung von HTML-Seiten (und vielleicht auch Tools dazu hat), ist es interessant Programme mit eigener HTML-Oberfläche zuerstellen, die die vorhandenen VBScript-Routinen nutzen.

VBA – Visual Basic for Applications

Wenn auf dem Rechner bereits MS WinWord oder Excel installiert ist, hat man mit VBA bereits ein vollwertiges Entwicklungssystem, das auch die Entwicklung eigener Oberflächen gestattet. Lediglich der Zugang über WinWord und das Menü Extras | Makros ist etwas umständlich.

Visual Basic 6

VB6 bietet in der preisgünstigen Einsteiger Edition ein vollwertiges, professionelles Entwicklungssystem und läßt sich durch die deutlich teurere Professionelle Edition noch durch weitere Funktionalität ausstatten.

Visual Basic .NET

VB.NET ist eine Sprache der .NET Sprachfamilie, die auf Microsofts .NET Framework basiert. Ein Umstieg ist besonders dann reizvoll, wenn man die neue Technologie kennenlernen will.