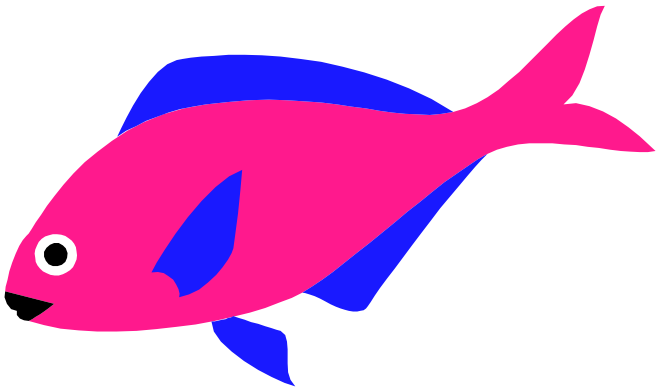
Einführung in die Programmierung mit

mscFish

JavaScript (JScript) Version

Ulrich Müller



Inhaltsverzeichnis

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

Dokumentname : mscFishJS.doc. Druckdatum : 13.08.2003
Titelbild : Einfügen | Grafik | AusDatei | Office | Fish8.WMF

Übersicht

Allgemeines

mscFish ist eine IDE (Integrated Development Environment) für ftComputing auf Basis der Microsoft Sprachen VBScript und JScript (Microsoft Version von JavaScript) sowie des ActiveX FishFa30.DLL. Dabei wurden die Sprachelemente von FishFa30.DLL nahtlos in die Sprachen VBScript/JScript integriert. In diesem Handbuch wird die Nutzung der Sprache **JScript** beschrieben.

mscFish 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. Parallel dazu sollte man sich mit der Microsoft Dokumentation zu JScript/VBScript (siehe "Hilfe – Dokumentation") vertraut machen.

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

Warum JavaScript - JScript

mscFish bietet derzeit die Möglichkeit Programme in VBScript und JScript zu erstellen. Diese Einführung beschreibt den Umgang mit JScript.

JavaScript wird man wählen, wenn durch die Erstellung von Webseiten bereits Kenntnisse im Umgang mit JavaScript vorliegen oder man plant Webseiten durch JavaScript-Elemente zu erweitern.

Die Wahl einer Programmiersprache hat oft recht emotionale Gründe. Man mag Basic einfach nicht oder findet geschweifte{}Klammern viel schöner. Dann sollte man zu JavaScript greifen. In Funktionalität und OO-Eigenschaften unterscheiden sich VBScript und JScript kaum.

Installation

mscFish wird in Form eines Setup-Programmes : mscFish30Setup.EXE ausgeliefert. Die Installation erfolgt weitgehend automatisch nach den üblichen Regeln.

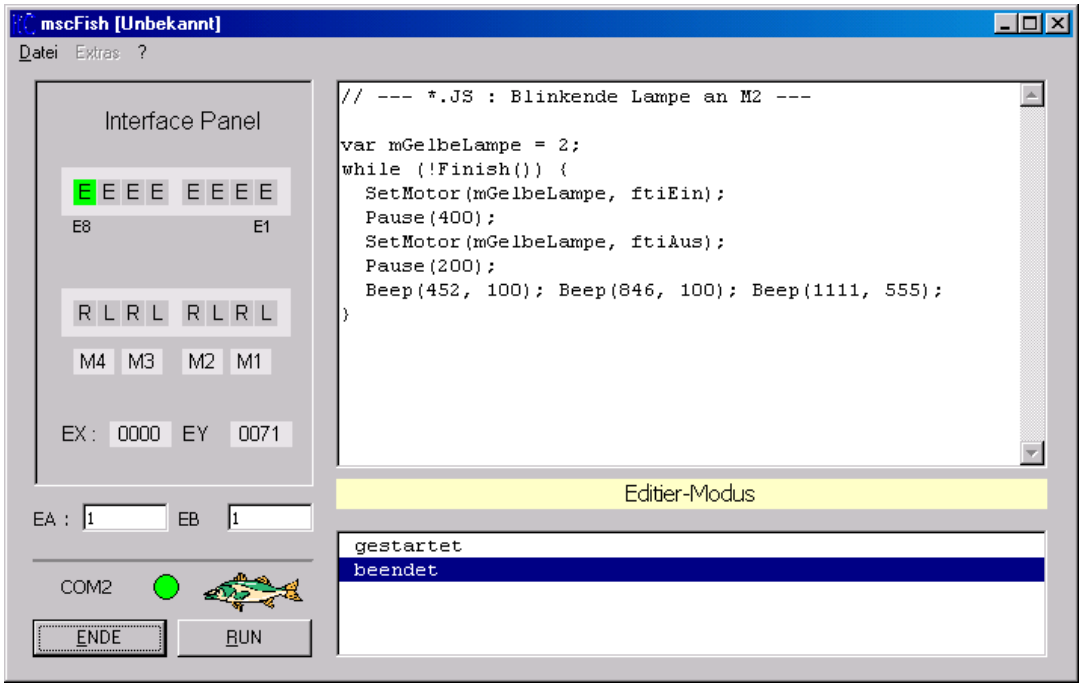
mscFish läuft ab Windows 98 und setzt ein installiertes Visual Basic 6 Laufzeitsystem (MSVBVM60.DLL) und einen installierten und aktivierten WSH (Windows Scripting Host einschließlich JScript-Runtime : installierten Internet Explorer) voraus. Siehe auch Anhang "Übersicht mscFish".

Wenn ein altes Universal-Interface am LPT-Port betrieben werden soll, ist bei der Installation EINE der LPT-Optionen auszuwählen.

Wenn auf dem Rechner FrontPage 2000 (oder höher) installiert ist, sollte man darauf achten, das der Microsoft Script-Editor installiert ist. Das erfolgt automatisch bei Anwahl Menü : Extras | Makro | Microsoft-Editor. So kann man bei Bedarf aus mscFish heraus auf die Testmöglichkeiten des Script-Editors zugreifen.

IDE – Integrated Development Environment

Übersicht



Die IDE gliedert sich in folgende Teile :

1. Bedienung der Gesamt IDE : Buttons links unter dem Strich
2. Kommunikation mit dem Modell über das Interface : Interface Panel
3. Parameter für die Anwendung : Die Felder EA und EB
4. Anzeige und Editieren der Anwendung : Editier-Feld, rechts oben
5. Protokollierung des Programmablaufs : Log-Feld, rechts unten
6. Anzeige des Programmstatus : Die Statuszeile, dazwischen
7. Laden und Speichern von Anwendungen : Menü Datei
8. Dokumentation : Script56.CHM für J/Script/VBScript und mscFish.PDF für FishFace z.Zt. als separate Dateien.
9. Fehleranzeigen : über Message Boxen

Bedienung der IDE

1. Einstellen der Interface-Verbindung über das Menü Extras | Interface Optionen. Der eingestellte Wert wird bei Programmende gespeichert und unten links über dem ENDE-Button angezeigt.
2. Einstellen der gewünschten Programmiersprache über das Menü Extras | Sprachen.
3. START-Button klicken.
4. Eine erfolgreiche Interface-Verbindung wird über einen grünen Button links neben der Interface-Anzeige angezeigt, die Beschriftung des START-Buttons wechselt in RUN, in der Statuszeile steht Editier-Modus. Anderfalls eine MessageBox : "Fehler beim Öffnen des Interfaces". Oft ist dann das Netzteil nicht angeschlossen.

5. Bei erfolgreicher Interface-Verbindung kann das Interface Panel genutzt werden. Getätigte E-Eingänge werden in der E-Zeile angezeigt.
   Die Werte der Analog-Eingänge in den Feldern EX und EY
   Die M-Ausgänge können durch Maus-Klick auf L bzw. R bedient werden. Wird gleichzeitig noch die Strg-Taste gedrückt, bleibt der M-Ausgang dauerhaft eingeschaltet. Ausschalten durch Klick auf das zugehörende Mx-Feld.
6. Ebenso kann bei Anzeige Editier-Modus das im Editier-Feld angezeigt Anwendungsprogramm geändert und ergänzt werden. Nach Start der IDE enthält das Editier-Feld ein kleines Beispiel-Programm, das man nach eigenem Bedarf ändern oder auch löschen kann (Neu.JS im Pfad von mscFish). Eigene Anwendungen kann man über das Menü Datei laden und speichern.
7. Ein Klick auf den jetzt mit RUN beschrifteten Button startet die Anwendung. Die Beschriftung des Buttons wechselt in HALT, in der Status-Zeile wird läuft angezeigt (solange es nicht durch die Anwendung überschrieben wird).
8. Vor dem Start der Anwendung können die EA / EB – Felder mit numerischen Werten belegt werden. Die Anwendung kann dann über gleichnamige Funktionen darauf zugreifen.
9. Das Programm kann durch Klick auf den HALT-Button beendet werden. Es gibt Situationen wo das nicht mehr möglich ist. Dann hilft nur noch der Weg über den Task-Manager (Strg+Alt+Entf), vorher kann die Anwendung aber noch gespeichert werden.
10. Während des Programmablaufs können Fehlermeldungen angezeigt werden. Das geschieht in einer MessageBox. Der Fehlertext beginnt mit Fehler in Zeile ... Man sollte ggf. die MessageBox so verschieben, daß man die markierte Zeile im Editierfeld lesen kann. Im Fehlertext folgt eine Nummer und dann die Beschreibung des Fehlers. Die Beschreibung des Fehlers ist meist recht aussagekräftig. Allerdings kann der Grund für die Fehlermeldung durchaus auch an anderer Stelle liegen. Man sollte im Zweifelsfall die Dokumentation zurate ziehen.
11. Die IDE kann normalerweise über den ENDE-Button beendet werden, während des Programmlaufs ist er allerdings deaktiviert, das Programm muß erst über den HALT-Button angehalten werden.

Die IDE greift beim Start einer Anwendung auf die Dateien Global.VBS und Neu.VBS (im Progammverzeichnis) zu. Beide Dateien können – mit Vorsicht - nach eigenen Bedürfnissen geändert werden. Global.VBS enthält Konstanten, die für jede Anwendung gelten (z.B. ftiEin / ftiAus) und Neu.VBS enthält das kleine Beispiel, das bei Start der IDE angezeigt wird, den Inhalt kann man schlicht löschen oder aber auch einen eigenen Standard-Programmrahmen erfinden.

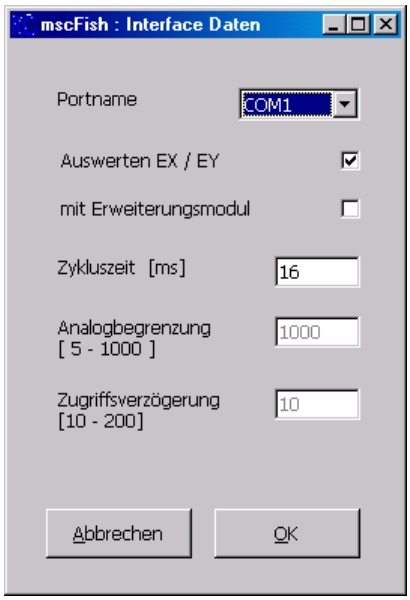
Anschluß des Interfaces

Anschluß je nach Gerät an COM bzw. LPT, Stromversorgung 9V bzw. 6V 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 Active Mode des Intelligent Interface wird nicht unterstützt. Slaves/Extension Modules unterstützt werden unterstützt, z.Zt. aber nicht im Interface Panel angezeigt.

Einstellen der Interfacewerte

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



Man sollte die Interfacewerte in der Reihenfolge der Form einstellen, also zuerst den Portnamen wählen. Damit verbunden werden für die weiteren Werte Standard-Vorgaben gemacht, die man nur mit Vorsicht ändern sollte.

EX / EY	Slave	Zykluszeit COM	Zykluszeit LPT*
x	x	16	100
x	-	16	100
-	x	12	8
-	-	10	8

* Die LPT-Werte sind "vorsichtig" gewählt, sie können unterschritten werden. Mit EX/EY = 20, ohne = 1.

Wenn man nicht vorhat, Analogwerte zu messen, kann man den Haken bei "Auswerten EX / EY deaktivieren. Wird diese Option beim parallelen (Universal) Interface angewählt, so steigt die vorgegebene "Zykluszeit" d.h. das Zeitintervall in dem der Interface Status upgedatet wird, drastisch. Man sollte die Zykluszeit nur mit Vorsicht kleiner wählen, da der Rechner dabei "einfrieren" kann.

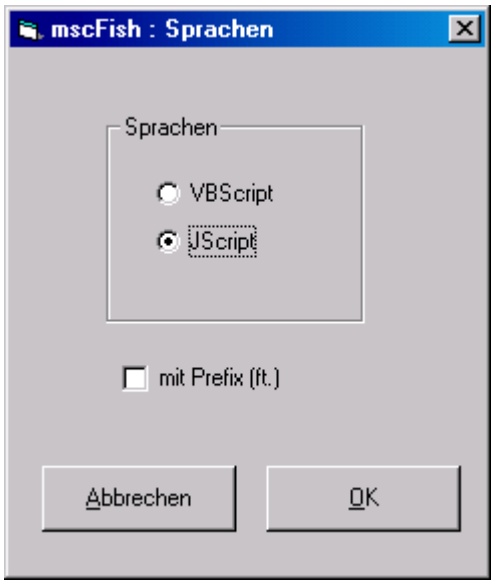
Der Erweiterungsmodul (Slave / Extension Module) wird unterstützt, allerdings z.Zt. nicht im Interface Panel angezeigt.

Die Analogbegrenzung beeinflußt beim parallen Interface die angezeigten Analogwerte. Beim LPT1/3 werden sie gestreckt d.h. bei größeren Werten über einen größeren Anzeigebereich gedehnt, Bei LPT ist die Analogbegrenzung eine Obergrenze bei der darüber hinausgehende Werte abgeschnitten werden.

Die Zugriffsverzögerung ist eine Korrektur beim Zugriff auf das Interface bei LPT1/3. Größere Werte für schnellere Rechner.

Einstellen der Programmiersprache

mscFish kann wahlweise mit JavaScript (in der Microsoft-Version JScript) bzw. VBScript betrieben werden. Die Auswahl der Programmiersprache erfolgt über das Menü Extras | Sprachen.



Das ist nur gleich nach dem Programmstart möglich. Außerdem kann ausgewählt werden, ob den FishFace-Befehlen ein ft. vorangestellt werden muß. Sinn voll ist das dann, wenn das mit mscFish erstellte Programm auch außerhalb der mscFish-Umgebung (z.B. innerhalb einer HTML-Seite) laufen soll – siehe Anhang.

Hilfe – Dokumentation

JScript

Die deutsche Hilfe-Datei **Script56.CHM** von Microsoft enthält im Kapitel JScript | JScript-Benutzerhandbuch eine ausführliche Übersicht über die kompletten Möglichkeiten von JScript und im Kapitel JScript | JScript-Sprachverzeichnis eine Referenz aller JScript-Befehle. Zusätzlich im Kapitel Script-Laufzeit | FileSystemObject eine Beschreibung der umfangreichen Möglichkeiten des Dateizugriffs.

Im nachfolgenden Abschnitt "Einführung in die Programmierung" wird am Ende der Kapitel auf passende Seiten der JScript-Dokumentation hingewiesen.

Script56.CHM ist Bestandteil von mscFish30Setup. Es kann außerdem kostenlos von www.microsoft.com/germany/scripting geladen werden (Stand März 2003)

FishFace

Zu FishFace gibt es zwei im Aufbau und Inhalt recht ähnliche Handbücher für JScript/VBScript im PDF-Format. Sie bestehen aus einem **Übersichtsteil**, einer **Einführung in die Programmierung** und einem **Referenzteil** für FishFa30.DLL.

Einführung in die Programmierung

Allgemeines

Im diesem Abschnitt wird eine einfache Einführung in die Programmierung mit JavaScript gegeben. Genutzt wird hier JScript, die JavaScript-Implementiung von Microsoft, die mit dem Internet-Explorer verfügbar ist.

Die Einführung 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 JScript. Die Einführung erfolgt bewußt auf der Basis von "prozeduralen" Elementen um die Einstiegsschwelle niedrig zu halten. Eine Nutzung der OO-Elemente von JScript ist aber problemlos möglich.

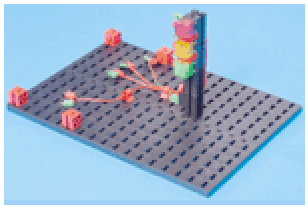
Die Kapitel bauen aufeinander auf und nehmen im Schwierigkeitsgrad zu, sie sollten deswegen nacheinander durchgearbeitet werden.

Bei weitergehendem Interesse kann nahtlos zu der in ausreichendem Maße vorhandenen Computerliteratur gegegriffen werden (z.B. Stefan Koch : JavaScript – Einführung, Programmierung und Referenz. ISBN 3-89864-111-2, dritte Auflage, www.dpunkt.de). Auf die bei Microsoft kostenlos erhältliche JScript-Dokumentation (s.o.) wird immer wieder am Ende eines Kapitels hingewiesen, sie sollte parallel zum Durcharbeiten dieses Abschnitts zu Rate gezogen 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 mscFish, den Interface-Anschluß (COM1 ...) kontrollieren, START Klicken. Der Punkt neben der Anschluß-Auswahl muß grün werden.
4. Am Interface Panel der Reihe nach mit der Maus auf die Ls klicken. Die entsprechende Lampe muß leuchten.
5. Den Text im Editier-Feld löschen und die oben angegebenen Befehle in das Editier-Feld 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 `while` (!Finish()) { ..... }
mit dem man eine Folge von anderen Befehlen "einrahmen" kann um sie zu wiederholen :

```
while (!Finish()) {
  SetMotor(ftiM1, ftiEin);
  Pause(1000);
  SetMotor(ftiM2, ftiEin);
  Pause(1000);
  SetMotor(ftiM3, ftiEin);
  Pause(1000);
  ClearMotors();
  Pause(1000);
}
```

Die Schleife läuft solange (**while**), wie die Finish-**Bedingung** nicht erfüllt wird ("!" ist eine Verneinung im Sinne Ausführung solange die Finish-Bedingung nicht erfüllt ist. Die Finish-Bedingung ist erfüllt, wenn in der IDE auf den HALT-Button oder auf der Tastatur auf die Esc-Taste gedrückt wird. Man könnte bei Finish auch noch einen Taster angeben z.B.: `Finish(ftiE8)`, dann würde die Schleife auch wenn der Eingang E8 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.

JScript Index Suchbegriff : do...while, while-Anweisung
FishFace Referenz : Finish

Schönheit – Lesbarkeit

```
// --- Ampel1.JS : Eine einfache Ampel ---------------

var mGruen = 1, mGelb = 2, mRot = 3;
var cLangePause = 1000, cKurzePause = 300;

while (!Finish()) {
  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);
}
```

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. Man sollte aber bei einer einheitlichen Schreibweise bleiben. Wichtig in diesem Zusammenhang ist, daß JavaScript eine "casesensitive" Sprache ist. D.h. es wird zwischen Groß- und Kleinbuchstaben unterschieden. cLangePause ist also nicht gleich clangepause. Die Befehlsworte von Javascript müssen einheitlich klein geschrieben werden. Die Befehlworte (Methoden) von FishFace dagegen mit Großbuchstaben(wenn einen das stört, kann man sie allerdings auch klein schreiben). Die Angelegenheit ist etwas verwirrend, aber man gewöhnt sicht dran, das sind so die kleinen Unterschiede bei den verschiedenen Programmiersprachen.

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

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

JavaScript (JS) ist eine Programmiersprache deren einzelne Befehle durch ein Semikolon (;) beendet werden. Befehle können über mehrere Zeilen verteilt werden, umgekehrt können auch mehrere Befehle in einer Zeile stehen. Man sollte nach Lesbarkeit entscheiden, meist ist es sinnvoll, nur einen Befehl pro Zeile zu schrieben.

---

* genaugenommen kennt JavaScript gar keine Konstanten, sondern nur Variable (s.u.), die im Sinne von Konstanten verwendet werden. Also Variable denen gleich bei der Deklaration ein Wert zugewiesen wird, der dann später nicht mehr verändert wird.

Variable

```
// --- Ampel1.JS : Eine einfache Ampel ----------------

var mGruen = 1, mGelb = 2, mRot = 3;
var LangePause = 1000, KurzePause;

KurzePause = LangePause/4;
while (!Finish()) {
  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);
}
```

Variable können eine veränderlichen (variablen) Wert enthalten

Konstanten enthalten einen festen Wert, Variable können einen veränderlichen (variablen) Wert enthalten. Durch Änderung von `var cLangePause = 1000` des vorhergehenden Beispiels in `var LangePause = 1000;` bezeichnet man eine Variable der gleich Anfangswert zugewiesen wurde (var LangePause; geht genauso, dann ist aber der Inhalt der Variablen undefinier "NaN"). Man sollte beachten, daß JavaScript keinen Unterschied zwischen Variablen und Konstanten macht. Die Unterscheidung wurde hier getroffen um die Verwendung von Variablen im Sinne Konstanten (wie sie in anderen Programmiersprachen anzutreffen sind) klarer zu machen.

Der Vorteil von Variablen ist, dass man diesen Wert wieder ändern kann, das auch mehr oder weniger kompliziert durch **Ausdrücke** : `KurzePause = LangePause / 4`. gebildet werden können 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.

Eine Deklaration der Variablen durch `var` vor der ersten Nutzung ist sehr sinnvoll, aber nicht erforderlich. Variable können nach Belieben numerische Werte (1, 2.34, 10000) oder Texte ("String", 'Text') sowie Wahrheitswerte (true, false) enthalten. Das muß nicht deklariert werden.

JScript Inhalt : JScript-Benutzerhandbuch | Grundlegende Informationen | JScript-Variablen

Ein- und Ausgaben

```
// --- Ampel1.VBS : Eine einfache Ampel ----------------

var mGruen = 1, mGelb = 2, mRot = 3;
var LangePause, KurzePause, Runde = 0;

LangePause = 1000*EA;
KurzePause = LangePause/4;
PrintStatus('--- Die Ampel bei der Arbeit ---');
while(!Finish()) {
  Runde = Runde + 1;
  PrintLog('Runde : ' + Runde);
  SetMotor(mGruen, ftiEin);
  Pause(LangePause);
  // .........
  SetMotor(mGelb, ftiAus);
}
```

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

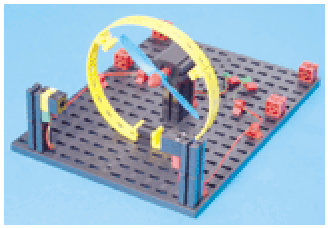
Mit dem Befehl **PrintStatus** kann in die hellblaue Statuszeile geschrieben werden. Hier wird die **Textkonstante** '--- Die Ampel bei der Arbeit ---' ausgegeben. Das ist eine neue Art von Konstante – bisher waren nur numerische Konstanten im Spiel. Textkonstanten beginnen und enden mit einem Hochkomma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

Die neue Variable **Runde** soll die durchlaufenen Schleifen zählen. Deklaration oben mit var wie gewohnt, mit dem Anfangswert 0, sonst ist der Inhalt der Variable undefiniert (NaN). 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. Die Kurzschreibweise `Runde += 1;` ist ebenfalls möglich.

PrintLog schreibt in den bisher noch nicht genutzten Log-Bereich. Geschrieben wird eine Text-Konstante und eine Zahl (der aktuelle Wert von Runde) die durch den Operator Plus (+) mit einander in geeigneter Weise zu Text verbunden werden. Im Log-Bereich werden die Ausgaben nicht überschrieben, sondern gescrollt (nach oben geschoben), die jeweils letzte Zeile wird markiert und bleibt im Fenster sichtbar. Über den Scrollbar rechts können auch die anderen Zeilen angezeigt werden.

FishFace Referenz : PrintStatus, PrintLog

Motoren, Taster und Lichtschranken



Motor an M1,
Lichtschranke M2 Lampe,
E1 Phototransistor

```
var mMotor = 1, mLampe = 2;
var eTaster = 1, ePhoto = 1;
while (!Finish()) {
  SetMotor(mMotor, ftiEin);
  Pause(5000);
  SetMotor(mMotor, ftiAus);
}
```

"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 E1 probiert.

```
while (!Finish()) {
  if (GetInput(eTaster)) {
    SetMotor(mMotor, ftiEin);
    Pause(5000);
    SetMotor(mMotor, ftiAus);
  }
}
```

Als erstes fällt die neue Konstruktion `if (...) { ... }` auf. Sie ist, wie das schon bekannte while (..) {...}, eine Programmklammer. Hier werden die Befehle zum Schalten des Trocknermotors eingeklammert. Dem if folgt eine Bedingung (ganz wie beim while), hier ist das GetInput(eTaster). Dieser Befehl fragt den Eingang E1 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 E1 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);
while (!Finish()) {
  if (GetInput(ePhoto)) {
    SetMotor(mMotor, ftiEin);
    Pause(5000);
    SetMotor(mMotor, ftiAus);
  }
}
```

Vor dem while muß erstmal die Lichtschranke aktiviert werden (jetzt ist der Phototransistor an E1 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 (!GetInput(ePhoto)) {
```

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 **!**. Jetzt geht's!

JScript Inhalt : JScript-Sprachverzeichnis | JScript-Anweisungen |

Jetzt noch einige Verzierungen :

```
// --- Trockner.VBS : Händetrockner ---

var mMotor = 1, mLampe = 2;
var eTaster = 1, ePhoto = 1;
var Kunden = 0;

SetMotor(mLampe, ftiEin);
Pause(1000);
while (!Finish()) {
  PrintStatus("Trockner bereit");
  if (!GetInput(ePhoto)) {
    PrintStatus("Trockner läuft");
    Kunden += 1;
    PrintLog("Kunde Nr : " + Kunden + " am " + Date());
    SetMotor(mMotor, ftiEin);
    Pause(5000);
    SetMotor(mMotor, ftiAus);
  }
}
```

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 auf 0 initialisiert. Jedesmal, wenn der Motor gestartet werden soll wird **Kunden + 1** hochgezählt und mit PrintLog 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 **Date()**, 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 für aus über 7 für ftiHalf und 15 für ftiFull. Man kann also noch mehr spielen.

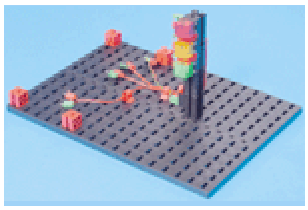
Und noch ein Nachtrag

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.

JScript Inhalt : JScript | JScript-Sprachverzeichnis | JScript-Operatoren

Intermezzo : Nocheinmal die Ampel



Lampen an M1 – M4
grün – gelb – rot –
FußGrün (an der Seite)
Taster (1/3) an E1 und E2

```
var mGruen = 1, mGelb = 2, mRot = 3;
var mFuss = 4;
var eFussWunsch = 1;

while (!Finish()) {
  SetMotor(mGruen, ftiEin);
  if (GetInput(eFussWunsch)) {
    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);
  }
}
```

Das Programm ist weitgehend bekannt, aber eine 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 :

```
while (!Finish()) {
  SetMotor(mGruen, ftiEin);
  if (GetInput(eFussWunsch)) {
    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);
  }
}
```

Geschachtelte if's und functions

Nachts werden Ampeln oft auf Gelb-Blinken umgestellt :

```
var eFussWunsch = 1, eGelbBlinken = 2;

while (!Finish()) {
  if (GetInput(eGelbBlinken)) {
    Blinken();
    }
  else { if (GetInput(eFussWunsch)) {
    Pause(1000*EA);
    SetMotor(mGruen, ftiAus);
    // .....
    SetMotor(mGelb, ftiAus);
    SetMotor(mGruen, ftiEin);
    }
  else {
    SetMotor(mGruen, ftiEin);
    }
  }
}

function Blinken() {
  ClearMotors();
  SetMotor(mGelb, ftiEin);
  Pause(500*EA);
  SetMotor(mGelb, ftiAus);
  Pause(400*EA);
}
```

Das GelbBlinken wird über die Abfrage eines weiteren Tasters eGelbBlinken geschaltet. Die vorhandene if (...) {...} Klammer wird um ein weiteres if erweitert. Beim ersten 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 – eine **Funktion** (function) – gesteuert. In einer Function werden logisch zusammengehörenden Befehle zusammengefaßt und außerhalb des normalen Programmablaufs abgestellt. Eine Funktion wird durch seinen Namen aufgerufen (beim if eGelbBlinken : **Blinken();**)*. Vorteil einer Funktion : das eigentliche (Haupt)Programm wird übersichtlicher, bei mehrmaligem Aufruf (wenn man auch noch woanders Blinken will) wird vorhander Code wiederverwendet.

Einer Funktion können außerdem noch Parameter übergeben werden. Ebenso kann eine Funktion auch noch einen Wert zurückgeben (Das kommt gleich).

JScript Index : JScript | JScript-Sprachverzeichnis | JScript-Anweisungen

* Vorsicht : Auch wenn eine Funktion keine Parameter besitzt müssen beim Aufruf Klammern angegeben werden. Also Blinken(); statt Blinken; . Sonst "versandet" der Funktionsaufruf kommentarlos und man sucht endlos.

SetMotorS

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

```
// --- FussAmpel3.JS : Eine einfache Ampel --------

var mGruen = 0x1, mGelb = 0x4, mRot = 0x10, mFuss = 0x40;
var eFussWunsch = 1, eGelbBlinken = 2;

while (!Finish()) {
  if (GetInput(eGelbBlinken)) Blinken();
  else { if (GetInput(eFussWunsch)) {
    Pause(1000*EA);
    SetMotors(mGelb);
    Pause(250*EA);
    SetMotors(mRot + mFuss);
    Pause(1000*EA);
    SetMotors(mGelb);
    Pause(250*EA);
    SetMotors(mGruen);
    }
  else {
    SetMotors(mGruen);
    }
  }
}

function Blinken() {
  SetMotors(mGelb);
  Pause(500*EA);
  ClearMotors();
  Pause(400);
}
```

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.

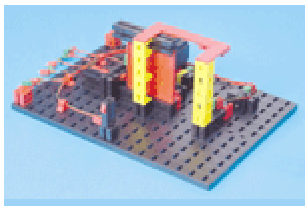
```
00 01 00 00       (0x10, mRot in binär Darstellung)
01 00 00 00 +     (0x40, mFuss)

-------------
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 PrintLog-Ausgaben des Händetrockners entsprechend modifiziert übernehmen.

FishFace Referenz : SetMotors, ClearMotors

Über das Türenschieben



Motor an M1
Lichtschranke M2, E4
Taster TürAuf E2
Taster TürZu E1
Taster Öffnen E3

"Wenn Taster E3 gedrückt wird, soll sich die Tür öffnen und nach fünf Sekunden wieder schließen."

```
// --- Tuer.JS : Schiebetür ---

var mTuer = 1, mLampe = 2;
var eTuerZu = 1, eTuerAuf = 2, eOeffnen = 3;
var ePhoto = 4;

while (!Finish()) {
  SetMotor(mTuer, ftiLinks);
  WaitForInput(eTuerZu, false);
  SetMotor(mTuer, ftiAus);

  WaitForInput(eOeffnen);

  SetMotor(mTuer, ftiRechts);
  WaitForInput(eTuerAuf);
  SetMotor(mTuer, ftiAus);
  Pause(5000);
}
```

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

```
SetMotor(mLampe, ftiEin);
Pause(1000);
WaitForInput(ePhoto, true);

while (!Finish()) {
  while (GetInput(eTuerZu)) {
    SetMotor(mTuer, ftiLinks);
    if (!GetInput(ePhoto)) TuerOeffnen();
  }
  SetMotor(mTuer, ftiAus);

  if (GetInput(eOeffnen) || !GetInput(ePhoto)) TuerOeffnen();
}

function TuerOeffnen() {
  SetMotor(mTuer, ftiRechts);
  WaitForInput(eTuerAuf);
  SetMotor(mTuer, ftiAus);
  Pause(5000);
}
```

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 while los.

Es folgt dann gleich noch einer `while (GetInput(eTuerZu))`. Zum Türschließen. Dies ist, wie auch die bekannte while(!Finish()) eine "abweisende" Schleife. D.h. die Schleife wird übersprungen, wenn die Bedingung nicht erfüllt ist, also die Tür bereits geschlossen ist. Die Schleife do {...} while (...); dagegen würde mindestens einmal durchlaufen.

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 die Funktion 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.

```
function TuerOeffnen(Normal) {
  PrintStatus('--- Tür öffnet ---');
  if (Normal) PrintLog('Tür wurde geöffnet : ' + Date());
  else PrintLog('Tür-Zwischenfall   : ' + Date());
  SetMotor(mTuer, ftiRechts);
  WaitForInput(eTuerAuf);
  SetMotor(mTuer, ftiAus);
  PrintStatus('--- Tür geöffnet ---');
  Pause(5000);
}
```

Man kann eine Funktion auch einen oder mehrere Parameter mitgeben um den Ablauf des Unterprogramms zu steuern. Dazu müssen sie (die Parameter) in der function-Definitionszeile angeführt werden (formale Parameter, hier **Normal**), sie können dann in der Funktion 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.

JScript Inhalt : JScript | JScript-Sprachverzeichnis |
JScript-Operatoren | Logischer OR Operator
JScript-Anweisungen | do ... while, function

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."

```
var mMotor = 1, mLampe = 2;
var aNTC = 0;
var Temperatur;

while (!Finish()) {
  Temperatur = GetAnalog(aNTC);
  if (Temperatur < EA) {
    SetMotor(mLampe, ftiAus);
    SetMotor(mMotor, ftiEin);
    }
  else { if (Temperatur > EB) {
    SetMotor(mLampe, ftiEin);
    SetMotor(mMotor, ftiAus);
    }
  }
}
```

Als erstes wird in der while 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) besetzt. 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 ein geschachteltes if. 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

HINWEIS : Beim parallelen (Universal) Interface ist hier von der Standard-Einstellung des Interfaces (Menü Extras | Optionen) abzuweichen. "Auswerten EX / EY " muß angekreuzt werden. Die vorgegebene Zykluszeit von 100 ms sollte erstmal beibehalten werden. Man sollte aber mit dem Wert experimentieren. Diese drastisch höhere Zykluszeit (Standard ohne = 8) verändert das Anwortverhalten des Interfaces. Alternativ kann mit **`GetAnalogDirect`** gearbeitet werden, dann kann die Einstellung "ohne" mit Zykluszeit 8 beibehalten werden. Der Analogwert wird dann direkt vom Interface gelesen, das Programm hält dann an dieser Stelle ca. 100 ms.

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.JS : Temperaturregelung ---

var mMotor = 1, mLampe = 2;
var aNTC = 0;
var UT, OT;

UT = EA * (1 - EB / 100);
OT = EA * (1 + EB / 100);
PrintLog('Zieltemperatur : ' + EA);
PrintLog('Untergrenze    : ' + UT);
PrintLog('Obergrenze     : ' + OT);

while (!Finish()) {
  if (Temperatur() < UT) {
    PrintStatus('--- Heizt : ' + Temperatur() + ' ---');
    SetMotor(mLampe, ftiEin);
    SetMotor(mMotor, ftiAus);
    UT = EA;
    }
  else { if (Temperatur() < OT) {
    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;
    }
  }
}

function Temperatur() {
  return (1000 - GetAnalog(aNTC)) / 10 - 12;
}
```

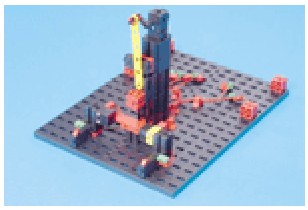
Hier wurden mehrere Maßnahmen in einen Schritt in ein (beinahe) neues Programm eingebaut :

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. 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 abgzogen, das sollen dann Grad Celsius sein. Der NTC wurde nicht wirklich geeicht, aber die Werte sind realitätsnah im Bereich 25 – 35. Achtung : Die Funktion Temperatur wird mit Klammern - `Temperatur();` aufgerufen!

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 Log-Feld ausgegeben, weils doch mit dem Kopfrechnen so seine Probleme hat.
4. Das geschachtelte if wurde um einen else Zweig erweitert, die zugehörenden Temperatur-Abfragen wurden geändert und den neuen Temperaturwerten angepaßt :
   if : **Zu niedrige Temperatur**
   else if : **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.

Stanzmaschine



Motor an M1
Lampe an M2
Endtaster an E1
Photo-Widerstand an E2
Bedientaster links an E3
Bedientaster rechts an E4

"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.JS ---

var mStanze = 1, mLampe = 2;
var eEnde = 1,   ePhoto = 2, eLinks = 3,
eRechts = 4;
var Produktion = 0, TeilOK, i;

SetMotor(mLampe, ftiEin);
SetMotor(mStanze, ftiEin);
WaitForInput(eEnde);
SetMotor(mStanze, ftiAus);

while (!Finish()) {
  if (GetInput(ePhoto) &&
      GetInput(eLinks) &&
      GetInput(eRechts)) {
    TeilOK = true;
    for (i = 1; i <= EA; i++) {
      if (GetInput(ePhoto)) {
        SetMotor(mStanze, ftiEin);
        WaitForHigh(eEnde);
        SetMotor(mStanze, ftiAus);
        }
      else {
        SetMotor(mStanze, ftiAus);
        PrintLog('PhotoMist : ' + Date());
        TeilOK = false;
        Beep(1111, 100); Beep(555, 100);
        break;
      }
    }
    if (TeilOK) Produktion += 1;
  }
  PrintStatus('Anzahl produzierte Teile : '
                              + Produktion);
}
```

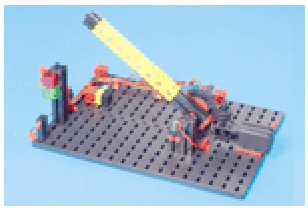
Die interessantesten Punkte des Programms sind :

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 && (Logisches AND) verknüpft sind : alle Bedingungen müssen aufeinmal wahr sein.
3. Neu ist auch die **for (...) {...} Schleife**, die die Anzahl der in EA vorgegebenen Hübe überwacht.
4. Vor jedem Hub wird die Lichtschranke nocheinmal kontrolliert und ggf. kräftig gemeckert und mit break die for 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 ... 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 Log-Fenster angezeigt.

FishFace Referenz : WaitForHigh, Beep
JScript Inhalt : JScript | JScript-Sprachverzeichnis | JScript-Anweisungen | for-Anweisung
JScript-Operatoren | Logisch AND

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 E1
AufTaster an E2
BedienTaster an E3
Photowiderstand E4

"Durch Betätigen des Tasters E3 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.JS ---

var mSchranke = 1, mRot = 2,
    mGruen = 3,    mLicht = 4;
var eZu = 1, eAuf = 2,
    eOeffnen = 3, ePhoto = 4;
var sZu = 1, sAuf = 2;

SetMotor(mLicht, ftiEin);
SetMotor(mRot, ftiEin);

while (!Finish()) {
  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('--- Wait for Durchfahrt ---');
  WaitForLow(ePhoto);
  WaitForHigh(ePhoto);
  SetMotor(mGruen, ftiAus);
  SetMotor(mRot, ftiEin);
  WaitForTime(500);
}
```

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()) {
    PrintStatus('--- Schranke öffnet ---');
    // .....
    WaitForTime(500);
  }
}

function WaitForCode() {
  if (Secret(PromptBox('Bitte Zugangscode eingeben!')))
                                      return true;
  if (AlertBox('Sorry - das wars nicht')) {
    NotHalt = true;
    PrintLog('Das Programm wurde gewaltsam beendet ' + Date());
  }
  return false;
}
```

```
function Secret(PINcode) {
  heute = new Date();
  if (PINcode == heute.getDate()*100 + heute.getMonth()+1)
     return true; else return false;
}
```

Anstelle des bisherigen WaitForHigh tritt ein if (WaitForCode()) 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 mscFish-Funktion **PromptBox**, zurückgegeben wird der eingegebene PIN-Code. Das Ergebnis wird einer hier erstellten Function **Secret** übergeben, diese wiederum vergleicht es mit dem intern gebildeten aktuellen PIN.

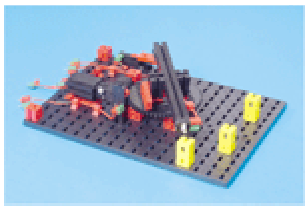
War der eingegebene PIN-Code falsch wird mit **AlertBox** gemeckert.

Bei der Anwort true 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 PromptBox vergleicht, ja die wird aus Tag und Monat zusammengebastelt (heute war es 208 und das war im August). Man kann die Sache auch noch viel spannender machen.

FishFace Referenz : WaitForLow, PromptBox, AlertBox

Der Schweißroboter



Motor an M1
Lampe an M2
EndeTaster an E1
ImpulsTaster an E2

"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.JS ---

var mRobot = 1, mSchweiss = 2;
var eEnde = 1, eImpuls = 2;
var IstPosition;

while (!Finish()) {
  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();
}

function Schweissen() {
var i;
  for (i = 1; i <= EA; i++) {
    SetMotor(mSchweiss, ftiEin);
    Pause(100 * EB);
    SetMotor(mSchweiss, ftiAus);
    Pause(100 * EB);
  }
}
```

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.JS ---

var mRobot = 1, mSchweiss = 2;
var eEnde = 1, eImpuls = 2;

while (!Finish()) {
  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();
}
```

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 nicht erforderlich.

SetMotors setzt mit dieser Parameterliste einen RobotMotor voraus. D.h. einen Motor mit festzugeordneten Tastern. Bei M1 sind das E1 als Endtaster und E2 als Impulstaster. Der Endtaster muß außerdem mit ftiLinks angefahren werden können. Für die weiteren Motoren gilt entsprechend M2 : E3/E4, M3 : E5/6 ... 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 Impuls 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.JS ---

var mRobot = 1, mSchweiss = 2;
var eEnde = 1, eImpuls = 2;
var RobotPos;

while (!Finish()) {
  Home();
  MoveTo(56);
  Schweissen(EA, EB);
  MoveTo(144);
  Schweissen(EA/2, EB*2);
  MoveTo(94);
  Schweissen(EA, EB);
}

function MoveTo(ZielPos) {
  if (RobotPos < ZielPos) {
    SetMotor(mRobot, ftiRechts, ftiFull, ZielPos - RobotPos);
    do {
      PrintStatus('RobotPos : ' + (ZielPos - GetCounter(eImpuls)));
      } while (WaitForMotors(100, mRobot) == ftiTime)
    RobotPos = ZielPos + GetCounter(eImpuls);
    }
  else {
    SetMotor(mRobot, ftiLinks, ftiFull, RobotPos - ZielPos);
    do PrintStatus('RobotPos : ' + (ZielPos + GetCounter(eImpuls)));
    while (WaitForMotors(100, mRobot) == ftiTime);
    RobotPos = ZielPos + GetCounter(eImpuls);
  }
  PrintStatus('RobotPos : ' + RobotPos);
}

function Schweissen(Mal, Dauer) {
  for (var i = 1; i <= Mal; i++) {
    SetMotor(mSchweiss, ftiEin);
    Pause(100 * Dauer);
    SetMotor(mSchweiss, ftiAus);
    Pause(100 * Dauer);
  }
}

function Home() {
  PrintStatus('--- Es geht nach Hause ---');
  SetMotor(mRobot, ftiLinks);
  WaitForInput(eEnde);
  SetMotor(mRobot, ftiAus);
  RobotPos = 0;
  PrintStatus('--- Zu Hause ---');
}
```

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 FishFa30 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 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 .....

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 (0-1, ftiNr)
AnalogWert	Rückgabewert beim Auslesen von EX/EY (0-1024)
Counter	Wert eines ImpulsCounters
Direction	Schaltzustand eines M-Ausganges (0-2, ftiDir)
InputNr	Nummer eines E-Einganges (0-8(16), ftiNr)
InputStatus	Rückgabewert beim Auslesen aller E-Eingänge (0 – 0xFFFF)
LampNr	Nummer eines "halben"-M-Ausganges (0-8(16), ftiNr)
ModeStatus	Status der Betriebsmodi aller M-Ausgänge. Jeweils 4 bit pro Ausgang. Begonnen bei 0-3 für M1 (0000 normal, 0001 RobMode).
MotorNr	Nummer eines M-Ausganges (0-4(8), ftiNr)
MotorStatus	SollStatus aller M-Ausgänge. Jeweils 2 bit pro Ausgang Begonnen bei 0-1 für M1 (00 = Aus, 01 = Links, 10 = Rechts)
mSek	Zeitangabe in MilliSekunden
NrOfchanges	Anzahl Impulse
OnOff	Ein/Ausschalten eines M-Ausganges (Boolean, ftiDir)
PortName	Name des wählbaren Anschlußports (String) ("LPT", "COM1" – "COM8", "LPT1" – "LPT3")
Position	Position in Impulsen ab Endtaster
Speed	Geschwindigkeitsstufe mit der ein M-Ausgang (Motor) betrieben werden soll (0-15, ftiSpeed)
SpeedStatus	Status der Geschwindigkeiten aller M-Ausgänge Jeweils 4 bit pro Ausgang. Begonnen bei 0-3 für M1 Werte 0000 (stop) – 1111 (full)
TermInputNr	Nummer eines E-Einganges mit der die (Wait)Methode beendet werden soll (ftiNr)
Value	Allgemeiner numerischer Wert
WaitWert	Rückgabewert von WaitForMotors (ftiWait)

Die Aufrufparameter sind Werteparameter, d.h. sie übergeben Werte, geben aber keine zurück, Ausname Parameter IstPosition bei WaitForPosDown / WaitForPosUp

Eine Reihe von Parametern sind optional, d.h. sie brauchen nicht angegeben werden. Intern werden sie dann mit sinnvollen Werten belegt. Das wird beim einzelnen Befehl beschrieben.

Symbolische Konstanten

Um ein Programm lesbarer zu machen, werden von mscFish eine Reihe von symbolischen Konstanten angeboten, die in der Datei Global.JS zusammengefaßt sind, sie können bei Bedarf geändert werden. Die Konstanten werden unter den folgenden Oberbegriffen (die nicht selber verwendet werden können) zusammengefaßt :

ftiDir	Angabe des Schaltzustandes (Drehrichtung, Ein/Aus) ftiLeft, ftiRight, ftiOff, ftiOn, ftiLinks, ftiRechts, ftiAus, ftiEin
ftiNr	Angabe der Nummer eines Ein- bzw. Ausganges ftiEX, ftiEY, ftiE1 .. ftiE8, ftiM1 .. ftiM4
ftiSpeed	Angabe einer Geschwindigkeitsstufe ftiNone = 0, ftiHalf = 7, ftiFull = 15, Zwischenwerte können numerisch eingegeben werden.
ftiWait	Returnwerte der Methode WaitForMotors ftiEnde, ftiEnd, ftiTime, ftiNotHalt, ftiEsc

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. `while (!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. Dazu wird intern für jede Geschwindigkeitsstufe eine entsprechende Schaltliste vorgehalten. Die Geschwindigkeitsstufe wird durch die Parameter Speed bzw. SpeedStatus für einen bzw. alle Motoren bei der Methode SetMotor(s) bestimmt.

Counter

Zu jedem E-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

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

```
SetMotor(ftiM1, ftiLeft, ftiHalf, 50);
SetMotor(ftiM2, ftiRight, ftiFull, 60);
WaitForMotors(0, ftiM1, ftiM2);
```

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

Lampen am Interface

Die vier M-Ausgänge des Interfaces sind primär zum Schalten von Motoren in zwei Laufrichtungen vorgesehen. Doch bietet sich zusätzlch die Möglichkeit, Geräte, die nur ein- und ausgeschaltet werden müssen, an nur einem Pol eines M-Ausganges und Masse anzuschließen. Auf diese Weise ist es möglich z.B. acht Lampen mit einem Interface zu schalten. Hiefür gibt es die Methode SetLamp.

Sollen Lampen gemeinsam (z.B. bei einer Verkehrampel) geschaltet werden, können sie auch über die Methode SetMotors geschaltet werden. Dazu sind die einzelnen Lampenbits im MotorStatus zu setzen.

Es gibt Unterschiede zwischen den Interfaces :

Universal (paralleles) Interface : im nicht geschalteten Zustand sind die Lampen aus. M1 vorderer (gelber) Kontakt : Lampe 1, hinterer (oranger) Kontakt Lampe 2 ...

Intelligent (serielles) Interface : im nicht geschalteten Zustand sind die Lampen **an**. M1 vorderer Kontakt : Lampe 1, hinterer Kontakt Lampe 2 ...

Liste der Befehle

AlertBox

Anzeige einer Alarmbox mit variablem Text. Die Alarmbox kann mit OK bzw. Abbrechen quiitiert werden.

Bool = **AlertBox**(Text)

Beispiel

```
if (AlertBox('mscFish ist gestartet')) {
  // hier geht's bei OK weiter
}
```

Der Text mscFish ist gestartet wird angezeigt, der Code in der if-Klammer wird bei Quittierung mit OK ausgeführt.

Beep

Ausgabe eines Tones auf dem Systemlautsprecher

Beep(Frequenz, Dauer)

Frequenz : Tonhähe in Hz
Dauer : Dauer der Tones in Millisekunden

Beispiel

```
Beep(432, 1000);
```

Der Kammerton a quäkt 1 Sekunde lang auf dem Systemlautsprecher.

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

CloseInterface

Schließen der Verbindung zum Interface
-> Im Betrieb mit mscFish nicht erforderlich, aber beim Einsatz von JScript-Programmen außerhalb der Umgebung von mscFish.

ft.**CloseInterface**()

Siehe auch OpenInterface

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, E-Eingang)

Boolean = **Finish**(InputNr)

Der Parameter InputNr ist optional.

Siehe auch GetInput, GetInputs, Inputs

Beispiel
```
while (!Finish(ftiE1)) {
   ....
}
```

Die while-Schleife wird solange durchlaufen, bis entweder NotHalt = True, die Esc-Taste gedrückt oder E1 = True wurde.

GetAnalog

Feststellen eines Analogwertes (EX / EY).
Es wird der intern vorliegende Wert ausgegeben. AnalogScan beim OpenInterface ist erforderlich.

AnalogWert = **GetAnalog**(AnalogNr)

Siehe auch GetAnalogs, AnalogsEX, AnalogsEY, AnalogScan, OpenInterface

Beispiel
```
PrintStatus(ft.Analog(ftiEX));
```

Der aktuelle EX-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(ftiE2));
```

Die aktuelle Turm Position wird in der Statuszeile angezeigt.

GetInput

Feststellen des Wertes des angegebenen E-Einganges

Boolean = **GetInput**(InputNr)

Siehe auch GetInputs, Inputs, Finish

Beispiel

```
if (GetInput(ftiE1)) {
   ...
}
else {
   ...
}
```

Wenn der E-Eingang E1 (Taster, PhotoTransistor, Reedkontakt ...) = true ist, wird der Zweig hinter dem if durchlaufen sonst der hinter dem else.

GetInputs

Feststellen der Werte aller E-Eingänge

InputStatus = **GetInputs**()

Siehe auch GetInput, Inputs, Finish

Beispiel

```
var e;
   e = GetInputs();
   if ((e & 0x1) || (e & 0x4)) { ...
```

Wenn die E-Eingänge E1 oder E3 true sind, ist die gesamte Bedingung true.

NotHalt

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

Get | Set, Boolean, Default = false

Beispiel

```
while(i < 100) {
   ....
   if (NotHalt) break;
}
```

Die while-Schleife wird beendet, wenn NotHalt den Wert true hat (oder i >= 100 ist).

Siehe auch Finish

OpenInterface

Herstellen der Verbindung zum Interface und Setzen/Bestimmen von dazu erforderlichen Parametern. OpenInterface muß deswegen als erste Methode aufgerufen werden.
-> Im Betrieb mit mscFish nicht erforderlich, aber beim Einsatz von JScript-Programmen außerhalb der Umgebung von mscFish.

ft.**OpenInterface**(PortName, AnalogScan, Slave, PollInterval, LPTAnalog, LPTDelay)

Die Parameter ab AnalogScan sind optional :

- PortName (String) : LPT | COM1 – COM8 | LPT1 – LPT3
  Zuordnung des zuverwendeden Interfaces durch Angabe des Ports an dem es angeschlossen ist.
- AnalogScan (Boolean) : Angabe, ob auch die Analogeingänge (EX / EY) gepollt werden sollen. Es ist dann ein höheres PollInterval erforderlich. Default : False
- Slave (Boolean) : Angabe ob an das primäre Interface ein weiteres angeschlossen ist. Default = False
- PollInterval (Long) : Angabe in mSek in welchen Intervallen auf das Interface zugegriffen werden soll. Default = 0 : Bestimmung durch die Methode OpenInterface in Abhängigkeit vom Kontext.
- LPTAnalog (Long) : AnalogSkalierung. Begrenzung des Analogwertes nach oben und damit der für die Messung erforderlichen Zeit (nur LPT-Interface). Default = 5
- LPTDelay (Long) : Ausgabeverzögerung. Bei LPT-Interface auf schnellen Rechnern erforderlich. Default = 10

Siehe auch CloseInterface

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.

PrintLog

Ausgabe eines Textes in das Log-Feld der IDE

PrintLog(stringausdruck)

Beispiel

```
PrintStatus('Schranke geöffnet : ' + Date());
```

Im Log-Fenster wird ausgegeben :

```
Schranke geöffnet : Sat Aug 02 22:58:18 2003
```

Siehe auch PrintStatus

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 PrintLog

PromptBox

Anzeige einer EingabeBox mit der Möglichkeit Daten einzugeben.

Wert = **PrintStatus**(Text)

Beispiel

```
var a = PromptBox('Bitte Anzahl Kisten eingeben');
```

Aufforderung einen Wert einzugeben.

SetCounter

Setzen eines Counters

SetCounter(InputNr, Value)

Siehe auch GetCounter, ClearCounter, ClearCounters

SetLamp

Setzen eines "halben" M-Ausganges. Anschluß einer Lampe oder eines Magneten ... an einen Kontakt eines M-Ausganges und Masse. Siehe auch "Lampen am Interface"

SetLamp(LampNr, OnOff)

Siehe auch SetMotor, SetMotors, ClearMotors

Beispiel

```
var lGruen = 1, lGelb = 2, lRot = 3;

  SetLamp(lGruen, ftiEin);
  Pause(2000);
  SetLamp(lGruen, ftiAus);
 SetLamp(lGelb, ftiEin);
```

Die grüne Lampe an M1-vorn und Masse wird für 2 Sekunden eingeschaltet und anschließend die gelbe an M1-hinten ...

SetMotor

Setzen eines M-Ausganges (Motor)

SetMotor(MotorNr, Direction, Speed, Counter)

Die Parameter ab Speed sind optional

MotorNr (ftiNr) : 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 linksdrehend, volle Geschwindigkeit eingeschaltet und anschließen für 1000 mSek rechtsdrehend, halbe Geschwindigkeit.

Beispiel 2

```
SetMotor(ftiM1, ftiLeft, 12, 123);
```

Der Motor am M-Ausgang M1 wird für 123 Impulse am E-Eingang E2 oder E1 = True mit Geschwindigkeitstufe 12 eingeschaltet. Das Abschalten erfolgt selbsttätig.

SetMotors

Setzen des Status aller M-Ausgänge

SetMotors(MotorStatus, SpeedStatus, ModeStatus)

Die Parameter ab SpeedStatus sind optional

MotorStatus (Long) : Schaltzustand der M-Ausgänge

SpeedStatus (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 ClearMotors, SetMotor, SetLamp, Outputs

Beispiel

```
SetMotors(0x1 + 0x80);
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 (ftiNr) : E-Eingang an dem die Impulse gezählt werden.

NrOfChanges (Long) : Anzahl Impulse

TermInputNr (ftiNr) : Alternatives Ende. E-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(ftiE2, 123, ftiE1);
SetMotor(ftiM1, ftiOff);
```

Der M-Ausgang (Motor) M1 wird linksdrehend geschaltet, es wird auf 123 Impulse an E-Eingang E2 oder E1 = 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(ftiE1);
SetMotor(ftiM2, ftiOff);
```

Eine Lichtschranke mit Lampe an M-Ausgang M1 und Phototransistor an E-Eingang E1 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 E-Eingang den vorgegebenen Wert annimmt.

WaitForInput(InputNr, OnOff)

OnOff (Boolean) : Endebedingung für E-Eingang InputNr, Default = true

Siehe auch WaitForChange, WaitForLow, WaitForHigh

Beispiel

```
SetMotor(ftiM1, ftiLeft);
WaitForInput(ftiE1);
SetMotor(ftiM1, ftiOff);
```

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

WaitForLow

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

WaitForLow(InputNr)

Siehe auch WaitForChange, WaitForInput, WaitForHigh

Beispiel

```
SetMotor(ftiM1, ftiOn);
SetMotor(ftiM2, ftiLeft);
WaitForLow(ftiE1);
SetMotor(ftiM2, ftiOff);
```

Eine Lichtschranke mit Lampe an M-Ausgang M1 und Phototransistor an E-Eingang E1 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 – ftiM8 in beliebiger Reihenfolge. 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(ftiE6) + ' – ' + GetCounter(ftiE8));
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 while-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.

IstPosition = **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. WaitForPosDown gibt dann die tatsächlich erreichte Position zurück(kann um einen Wert höher liegen, wenn der Motor nochmal "geruckt" hat). Alternativ wird die Methode durch E-Eingang TermInputNr = true beendet. IstPosition und ZielPosition müssen immer positive Werte (einschl. 0) enthalten.

Siehe auch WaitForPosUp, WaitForChange

Beispiel

```
var IstPosition = 12;
  SetMotor(ftiM1, ftiLinks);
  IstPosition = WaitForPosDown(ftiE2, IstPosition, 0);
  SetMotor(ftiM1, ftiAus);
```

Die aktuelle Position ist 12 (IstPosition, vor dem WaitForPosDown), der Motor an M-Ausgang M1 wird linksdrehend gestartet. WaitForPosDown wartet dann auf Erreichen der Position 0, der Motor wird dann ausgeschaltet. Als Return-Wert wird die tatsächlich erreichte Position zurückgegeben. Der Return-Wert muß nicht ausgewertet werden.
`WaitForPosDown(ftiE2, IstPosition, 0);` reicht auch, wenn man auf die Angabe der tatsächlichen Zielpostion verzichten kann.

WaitForPosUp

Warten auf Erreichen einer vorgegebenen Position.

IstPostion = **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. WaitForPosUp gibt dann tatsächlich erreichte Position zurück(kann um einen Wert höher liegen, wenn der Motor nochmal "geruckt" hat). Alternativ wird die Methode durch E-Eingang TermInputNr = true beendet. IstPosition und ZielPosition müssen immer positive Werte (einschl. 0) enthalten.

Siehe auch WaitForPosDown, WaitForChange

Beispiel

```
var IstPosition = 0;
   SetMotor(ftiM1, ftiRechts);
   WaitForPosUp(ftiE2, 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

```
while (!Finish()) {
   SetMotors(0x1);
   WaitForTime(555);
   SetMotors(0x4);
   WaitForTime(555);
}
```

In der while-Schleife 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

Übersicht mscFish

mscFish ist eine Entwicklungumgebung für JScript/VBScript Programme, die FishFa30.DLL nutzen. Sie unterstützt die Erstellung von Anwendungen zur Steuerung von fischertechnik Modellen, die über ein fischertechnik Interface an einen Windows PC angeschlosen sind.

Einsetzbar auf Windows 98/Me bzw. Windows NT/2000/XP

mscFish besteht aus folgenden Komponenten :

1. Der Entwicklungsumgebung : mscFish30.EXE
2. Dem Handbuch dazu : mscFish30VBS.PDF (VBScript) und mscFish30JS.PDF (JScript)
3. Der JScript/VBScript-Dokumentation : Script56.CHM

Von diesen Komponenten werden folgende weitere Komponenten genutzt :

1. Das Visual Basic 6 Laufzeitsystem : MSVBVM60.DLL und davon abhängige Komponenten
2. Die ActiveX-Komponente FishFa30.DLL mit umFish30.DLL und weiteren Komponenten.
3. Der Acrobat Reader.
4. Das WSH-System. Der Windows Scripting Host mit seinen Komponenten
5. Ein Hilfe-System für CHM-Dateien

Diese Komponenten können wie folgt bereitgestellt werden :

Zu 1 : Ist auf den moderneren Windows-System standardmäßig installiert

Zu 2 : in mscFish30Setup.EXE enthalten

Zu 3 : ist meistens installiert (Version 4 reicht), kann ggf. aus dem Internet heruntergeladen werden : www.adobe.de/products/acrobat/readstep.html

Zu 4 : ist meistens installiert, kann ggf. aus dem Internet heruntergeladen werden. Benötigt werden die Pakete scr56de.exe für Win 98/ME/NT bzw. scriptde.exe für Win 2000/XP www.microsoft.com/germany/scripting

und dort rechts oben "Windows Script 5.6 freigegeben" und dann das Paket scd56de.exe (Stand März 2003).

Zu 5 : ist auf neueren Windows-Systemen installiert.

Folgende Files werden als SharedFiles nach ...\System32 installiert :

CMDLGDE.DLL	MSScript.OCX
COMDLG32.DLL	SCRrnde.DLL
	SCRrun.DLL

Betrieb außerhalb der mscFish-Umgebung

JScript pur

Wenn man für den "Nutzeinsatz" das DrumRum von mscFish als unnötig empfindet kann man auch mit JScript-Pur arbeiten. JS.Files, sich durch einfachen Click auf das File im Windows Explorer starten lassen. Dazu müssen die vorhandenen mscFish-JScript-Files ein wenig modifiziert werden :

1. Den FishFace-Befehlen muß ein `ft.` vorangestellt werden. Man kann das generell durch den Menü-Punkt : Extras | Sprachen vorschreiben.
2. Erstellen einer Instanz von FishFace durch ActiveXObject
3. Öffnen / Schließen der Verbindung zum Interface durch OpenInterface/CloseInterface
4. Verzicht auf die Befehle der mscFish-Umgebung :
   AlertBox, Beep, ClearLog, EA, EB, PrintLog, PrintStatus, PromptBox, WaitForPosDown, WaitForPosUp.
5. AlertBox kann durch WScript.Echo ersetzt werden.
6. WaitForPosDown/Up können durch WaitForPositionDown/Up ersetzt werden.

```
var cLampeRot = 3, cLampeGelb = 2, cLampeGruen = 1;
var cLampeAus = 0, cLampeEin = 1;
var ft, i;

WScript.Echo("Ampel wird gestartet");
var ft = new ActiveXObject("FishFa30.FishFace");
ft.OpenInterface("COM2");                    // alternativ : "COM1"
WScript.Echo("Modellbetrieb gestartet (Ende ESC)");

for(i=1; i<=3; i++){
  ft.SetMotor(cLampeGruen, cLampeAus);
  ft.SetMotor(cLampeGelb, cLampeEin);
  ft.WaitForTime(300);
  ft.SetMotor(cLampeGelb, cLampeAus);
  ft.SetMotor(cLampeRot, cLampeEin);
  ft.WaitForTime(1000);
  ft.SetMotor(cLampeGelb, cLampeEin);
  ft.WaitForTime(300);
  ft.SetMotor(cLampeRot, cLampeAus);
  ft.SetMotor(cLampeGelb, cLampeAus);
  ft.SetMotor(cLampeGruen, cLampeEin);
  ft.WaitForTime(800);
}
ft.ClearMotors();
ft.CloseInterface();
WScript.Echo("Modellbetrieb beendet");
```

JScript Luxus

Wenn man über einen HTML-Editor verfügt, kann man die JScript-Programme auch in eine HTML-Seite einbetten und dem Programm so eine komfortable Oberfläche verschaffen. Manche HTML-Editoren verfügen zudem noch über schöne JScript-Editoren.

Beipiel : FrontPage + zugehörenden JS-Script-Editor (Menü | Extras | Makro | Microsoft) einschl. JScript-Testsystem.

Solche Editoren bieten einigen Luxus, benötigen aber auch einigen Einarbeitungsaufwand und sind bei der Erstellung einfacher 10-Zeilen-Programme schon eher hinderlich.