

Tutorial

Aus AutoItWiki

AUS TECHNISCHEN GRÜNDEN MUSS DAS TUTORIAL GETEILT WERDEN! DEN ZWEITEN ZEILT FINDET IHR HIER: Tutorial2 DANKE FÜR EUER VERSTÄNDNISS! =)

<<<ACHTUNG!! Um dieses Tutorial bearbeiten zu können, müsst ihr euch bitte anmelden! (rechts oben). Als angemeldeter User könnt ihr auch zB Bilder hochladen und einiges mehr!>>>>

Herzlich willkommen zu diesem Tutorial für die Skriptsprache AutoIt. Deine Entscheidung, dich intensiver mit dieser Sprache zu beschäftigen, wird sich aller Voraussicht nach lohnen. Die meisten, die sich an AutoIt gewagt haben, sind aus den verschiedensten Gründen begeistert. Zu den Vorteilen unten mehr. So, jetzt geht's aber los!

Inhaltsverzeichnis

- 1 Geschichte
- 2 Features
- 3 Installation
 - 3.1 Versionen
 - 3.2 Download
 - 3.3 Reihenfolge
- 4 Erste Schritte
 - 4.1 Vorlage ändern
- 5 Hilfe
- 6 Kommentare
 - 6.1 Kommentare in SciTE
 - 6.2 Beispiel zu Kommentaren
- 7 Funktionen aufrufen
 - 7.1 Beispiel zu Funktionsaufrufen
- 8 Variablen
 - 8.1 Deklaration
 - 8.1.1 Beispiel zu Variablendeklarationen
- 9 Konstanten
 - 9.1 Veränderung
 - 9.1.1 Beispiel
 - 9.2 Ausgabe
 - 9.2.1 Ausnahmen
 - 9.2.2 Beispiel Variablen manipulieren und ausgeben
- 10 Strings
 - 10.1 Strings umwandeln
 - 10.2 Strings Verschlüsseln
 - 10.3 Sonstiges
- 11 Arrays
 - 11.1 Arrays anzeigen
 - 11.2 Array nachträglich hinzufügen
 - 11.3 Etwas aus einem Array löschen
 - 11.4 Arrays sortieren
 - 11.5 Ein paar weitere Funktionen mit Arrays

- 11.6 Arrays als Elemente eines Arrays
- 12 Arrays Mehrdimensional
 - 12.1 Deklaration
 - 12.2 Größenbestimmung
 - 12.3 Anwendungsbeispiele
 - 12.4 Sortierung von zweidimensionalen Arrays
- 13 TCP/IP
- 14 If-Abfragen
 - 14.1 Syntax von If-Abfragen
 - 14.2 Beispiel
 - 14.3 Alternativen
 - 14.3.1 Select
 - 14.3.2 Switch
- 15 Rückgabewerte von Funktionen
 - 15.1 Beispiel Rückgabewerte
- 16 Fehlerbehandlung
- 17 Schleifen
 - 17.1 For-Beispiel
 - 17.2 While-Schleife
 - 17.2.1 While-Endlosschleife
- 18 Eigene Funktionen
- 19 Maus- und Tastatursimulation
- 20 Ini-Dateien
 - 20.1 Werte schreiben
 - 20.2 Werte lesen
 - 20.3 Beispiele
- 21 Registry
- 22 Reguläre Ausdrücke
 - 22.1 Bsp.: Prüfung auf Pfadkonformität
- 23 Dateien
- 24 Werte aus dem Speicher lesen / bearbeiten
- 25 GUI
 - 25.1 GUIControl
 - 25.1.1 Colored Button
 - 25.1.1.1 Button Style verwenden
 - 25.1.1.2 echte farbige Button
 - 25.1.1.3 Label als Button
 - 25.1.2 List Control
- 26 Koda(FormDesigner)
 - 26.1 Der Koda(FormDesigner)
 - 26.2 Koda(FormDesigner) auf deutsch
 - 26.3 GUI erstellen
 - 26.4 GUI bearbeiten

Geschichte

AutoIt begann seinen Siegeszug 1999. Damals begann der Engländer Jonathan Bennett für seine Arbeit eine Skriptsprache zu entwickeln, die dafür gedacht war, hunderte Computer mit der gleichen Software und Konfiguration auszustatten. Dieses Programm wurde immer besser und erreichte schon in Version 2 weite Verbreitung. Nach und nach fanden sich immer mehr fähige C++-Programmierer, die Jon bei der Entwicklung unterstützten. Anfangs war das Programm OpenSource-Software. Doch nachdem sich andere (kostenpflichtige) Automationstools offenbar daran bedient hatten, wurden die außergewöhnlichen Teile des Quelltextes wieder zurückgezogen. Deshalb ist das Programm heute nur noch Freeware. Dennoch gibt es einen Gutteil des Quelltextes hier zum Download. Für angehende C++-Entwickler durchaus zu empfehlen, da Jons Programmierstil sehr angenehm zu lesen und geschickt ist. Auf der offiziellen Seite <http://www.autoitscript.com> gibt es alle Downloads und ein Forum mit Hilfestellungen und tausenden AutoIt-Skripten zum freien Download. Während die offizielle Seite leider im Wesentlichen auf

Englisch gehalten ist, gibt es unter <http://www.autoit.de> ein deutsches Forum, dass sich um deutschen Support bemüht und damit recht erfolgreich ist.

Features

- * Windows- und DOS-Dateien ausführen
- * Tastendrücke simulieren (die meisten Tastaturschemata werden unterstützt)
- * Mausbewegungen und -klicks simulieren
- * Fenster verschieben, verändern (z.B. Größe) und anderweitig manipulieren
- * Direkt mit Steuerelementen in Fenstern interagieren (Text setzen/auslesen, verschieben, (de)aktivieren,
- * Mit der Zwischenablage arbeiten um das Ausschneiden und Einfügen von Texten zu ermöglichen
- * Mit der Registry arbeiten
- * Kompilierte Exe-Dateien erstellen, die vollkommen selbständig lauffähig sind und maximal 250 KB groß und
- * Unterstützung aller Kernfunktionen in Windows 95 - 2003 gleichermaßen (sofern vom Betriebssystem unterstütz
- * AutoIt ist kostenlos - die meisten "Konkurrenz-Produkte" kosten dutzende Euro und bieten i.d.R. weniger

Mit den so genannten UDFs (das sind Sammlungen von Funktionen, die selbst in AutoIt geschrieben sind und sich in eigene Skripte einbinden und wie normale AutoIt-Funktionen nutzen lassen), können zahlreiche weitere Funktionen erreichen, wie z.B.:

- * Hashing-Funktionen (MD5, SHA1, ...)
- * Komplette Fernsteuerung des Internetexplorers (siehe IE.au3)
- * Zip-Funktionalität
- * ...

Installation

Versionen

Es gibt von AutoIt immer zwei Entwicklungsstränge. Eine stabile Version erscheint nur alle paar Monate. Diese Version ist im Wesentlichen als bug-frei anzusehen, auch wenn das natürlich nie zu 100% zu garantieren ist. Allerdings sind die Hauptfunktionen schon so lange im Einsatz, dass man sich auf deren Funktionieren verlassen kann. Bei UDFs ist das selbst in der stabilen Version nicht immer so, aber in den allermeisten Fällen sind diese auch ausreichend getestet. Daneben gibt es im Abstand von einigen Tagen Beta-Versionen, die neue Funktionen enthalten, die aber noch nicht komplett ausgetestet sind, und gelegentlich Bugs enthalten. Meldungen über bereits gefundene Bugs gibt es im BugReport-Forum. Dort müssten Sie sich auch melden, wenn du einen Bug gefunden zu haben glaubst. Es wird empfohlen, zum Programmieren die aktuell stabile sowie eine möglichst aktuelle Beta installiert zu haben. So kann man bei Fehlern in der Beta noch mit der stabilen Version kompilieren, aber auch alle Beta-Funktionen nutzen.

Download

Die aktuell stabile Version von AutoIt gibt es unter Autoit-download (<http://www.autoitscript.com/autoit3/downloads.php>) - dort lädst du aus der obersten Zeile die Datei hinter "Download AutoIt v3" Die Beta-Version liegt hier: Autoit-Beta-download (<http://www.autoitscript.com/autoit3/files/beta/autoit/>) - dort ist die Version mit der höchsten Nummer und der Dateiendung .exe relevant. Direkte Downloadlinks findest du unter <http://www.autoit.de> rechts oben auf der Startseite. Editor

Man kann problemlos seine Skripte mit jedem Texteditor verfassen, denn die au3-Dateien sind ganz normale Textdateien. Allerdings haben die AutoIt-Entwickler den freien Editor SciTE speziell an AutoIt angepasst, sodass man mit Syntaxhighlighting, Funktions- und Variablenvervollständigung, Kompilieren (F7 für stabile Version, Alt+F7 für Beta) und Ausführen (F5 stabil, Alt+F5 Beta) direkt aus dem Editor heraus angenehme Features bekommt. Zudem liefert SciTE zahlreiche Tools wie Tidy (Codeeintrückung), Koda (GUI-Aufbau) und Makrorekorder mit. Daher ist die Installation dieses Editors unbedingt angeraten. Herunterladen kann man ihn hier: SciTE-download (<http://www.autoitscript.com/autoit3/scite/downloads.php>)

Reihenfolge

Die besten Erfahrungen haben wir mit folgender Installationsreihenfolge gemacht:

1. Installation der stabilen AutoIt-Version
(wenn möglich am besten mit Standardpfad, aber auch ein anderes Verzeichnis sollte er verkraften)
2. Installation der SciTE-Version für die stabile Version
3. Installation der AutoIt-Beta-Version
4. Installation der SciTE-Updates für die AutoIt-Beta-Version

Bei den SciTE-Installationen sollte die Standardaktion für einen Doppelklick auf die Datei auf "Edit Script" eingestellt werden, damit dann SciTE geöffnet wird.

Damit sollte AutoIt fertig eingerichtet sein und ein neuen Kapitel in deiner Programmierkarriere beginnen ;-). Aber keine Angst, es wird ein schönes werden!

Erste Schritte

Die Entwicklung jedes Skriptes beginnt damit eine au3-Datei anzulegen. Das geht am einfachsten im Explorer. Dort kann man auch gleich einen Unterordner für das Skript anlegen, damit die Ordnung gewahrt bleibt und mehrere Skripte sich nicht vermischen. Die Datei legst du dann im Kontextmenü in dem neu angelegten Ordner über "Neu" -> "AutoIt v3 Script" an. Vergib noch einen sprechenden Namen und öffne die Datei per Doppelklick in SciTE.

Vorlage ändern

Wenn du die Informationen, die die Standard-Vorlage für au3-Dateien enthält, ändern möchtest, dann passe dazu einfach die Datei "C:\Windows\ShellNew\Template.au3" (oder entsprechend bei anderem Windows-Verzeichnis) wie gewünscht an und speichere sie an ihrem ursprünglichen Ort ab.

Hilfe

Die jeweils aktuelle Hilfe findest du in deinem AutoIt-Verzeichnis auf der Festplatte. Aus SciTE kannst du die Hilfe zur installierten stabilen Version auf Englisch mit F1 aufrufen, die Beta-Hilfe bringt Alt+F1 auf den Schirm. Ist bei dem Tastendruck der Cursor in einem Funktionsnamen, so wird dieser direkt angezeigt.

Die deutsche Hilfe gibt es unter <http://www.autoitscript.com/autoit3/docs.php>. Allerdings deckt sie derzeit nur Version 3.1.0 ab. Es wird momentan aber an einer aktualisierten Fassung gearbeitet. Die Hilfe der Version 3.1.0 ist auch komplett online verfügbar und zwar unter <http://www.autoit.de/dokumentation/>.

Kommentare

Kommentare können in AutoIt auf unterschiedliche Arten gekennzeichnet werden. Alles, was in einer einzelnen Zeile nach einem Semikolon ("Strichpunkt", ;) steht, wird als Kommentar gewertet, ergo nicht ausgeführt/interpretiert.

Ganze Blöcke können zwischen #cs und #ce bzw. #comments-start und #comments-end eingeschlossen werden, um sie als Kommentar zu markieren.

Kommentare in SciTE

In SciTE kann man mit der Minus-Taste auf dem Ziffernblock einzelne oder mehrere (dann markierte) Zeilen schnell ein- und auskommentieren. Ebenso kann man dafür den Hotkey STRG+Q verwenden. Das ist besonders vorteilhaft, wenn man am Notebook sitzt.

Beispiel zu Kommentaren

```
; Diese Zeile ist auskommentiert wegen des Semikolons  
Sleep(2000) ; Hier wird das Sleep ausgeführt. Dieser ToolTip("Test") aber nicht  
#cs  
Sleep(3000)  
Sleep(4000)  
#ce
```

Dieses Skript wird für 2 Sekunden (2000 Millisekunden) im Systray erscheinen, da nur der erste Sleep-Befehl nicht auskommentiert ist.

Funktionen aufrufen

AutoIt bringt ab "Werk" bereits einige hundert Funktionen mit. Alle diese Funktionen sind in der Hilfe detailliert beschrieben. Diese Hilfe wird ihr ständiger Begleiter werden, um nachzusehen, was eine Funktion tut und wie man mit ihr umzugehen hat. Um Funktionen aufzurufen, gibt man ihren Namen, gefolgt von einer öffnenden runden Klammer ein. Danach übergibt man die Parameter; abgeschlossen wird der Aufruf von einer schließenden runden Klammer. Parameter, die in der Syntaxdefinition in der Hilfe von eckigen Klammern umschlossen sind, sind optional und können weggelassen werden. Dabei gilt analog zu anderen Sprachen (Pascal, C++, ...), dass normalerweise nur von hinter her weggelassen werden darf, damit der Interpreter weiß, welcher Wert zu welchem Parameter gehört. Man kann jedoch bei optionalen Parametern mit dem Schlüsselwort *Default* auf den vorgegebenen Standardwert ausweichen. So dann können auch Parameter in der Mitte der Definition, die nicht angegeben werden sollen, aufgefüllt werden. Sollte *Default* Probleme bereiten, kann man auch die alte Variante mit leeren Strings ("") für String-Parameter und -1 für Integer-Parameter versuchen.

Hier mal ein längeres Beispiel für Funktionsaufrufe. Details zu dem, was die Funktionen tun und der Bedeutung der Parameter entnehmen Sie bitte den entsprechenden Einträgen in der Hilfe.

Beispiel zu Funktionsaufrufen

```
; Notepad starten  
Run("notepad")  
  
; Warten, bis Notepad gestartet und aktiv ist  
WinWaitActive("Unbenannt - Edi")  
  
; Einige Tastendrücke senden  
Send("{TAB}Dies ist Text, den AutoIt so gesendet hat, als käme er vom Benutzer...")  
  
; eine kleine Pause einlegen  
Sleep(1500)  
  
; Fenster schließen  
WinClose("Unbenannt", "")  
  
; nach rechts und Enter (drückt "Nein" im Speichern-Dialog)  
Send("{RIGHT}{ENTER}")  
  
; Skript beenden  
Exit
```

Bitte betätigen Sie während das Skript läuft weder Maus noch Tastatur, da das die die Skriptfunktion beeinträchtigen könnte. Selbstverständlich kann man wichtigere Skripte dagegen absichern (Stichwort BlockInput).

Variablen

Deklaration

In AutoIt gibt es so genannte Variants, das heißt, es wird von Fall zu Fall entschieden, ob ein Variablenwert als String oder Zahl interpretiert und behandelt werden soll. Das ist einerseits angenehm, da man nicht ständig zwischen den verschiedenen Typen konvertieren muss, erfordert allerdings genaue Programmierung, damit Werte nicht falsch interpretiert werden und unerklärliche Fehler produzieren. Man kann Variablen deklarieren, muss man aber nicht.

Wenn man es macht, verwendet man dazu entweder DIM oder, wenn man möchte, dass die Variable überall gilt, GLOBAL. Man kann Zeichenketten oder Zahlen gleich als Startwert zuweisen. Bei Arrays gibt man die Größe in eckigen Klammern an. Variablenamen beginnen immer mit einem Dollarzeichen "\$" und bestehen aus Ziffern, Buchstaben und Unterstrich "_". Umlaute dürfen nicht verwendet werden.

Beispiel zu Variablendeklarationen

```
Dim $variable ; Variant
Dim $array[100]
Dim $var = "Wert"

Global $__var_1, $var__2

$ergebnis = 1 + 7*7

Dim $meine_variable
```

Beim Ausführen dieses Skriptes werden Sie nichts bemerken - es werden ja keine Ausgaben getätigt. Erreicht das Skript die letzte Zeile, wird es einfach beendet.

Konstanten

Eine Variable kann auch als Const deklariert werden, was auch mit Global/Local verbunden werden kann. Dann kann sie nicht mehr im Nachhinein innerhalb des Scripts verändert werden.

```
Const $PI = 3.141592653589793238

Func Beispiel_1()
    Local Const $PI = 3.141592653589793238
EndFunc

Func Beispiel_2()
    Global Const $PI = 3.141592653589793238
EndFunc

$PI = "Test"; <- Error, weil Variable bereits als Konstante deklariert.
```

Veränderung

In AutoIt verbindet man zwei Zeichenketten ("Strings") durch das Zeichen &.

Für die Rechnung mit Zahlen verwendest du die Zeichen +, -, /, *. Dazu sind einige mathematische Funktionen verfügbar (siehe Hilfe, Abschnitt Funktionsreferenz -> Mathematische Funktionen).

Wichtig ist, dass man auf der linken Seite des = eine Variable angibt, die das Ergebnis der Berechnungen oder Aktionen auf rechten Seite aufnimmt. Diese Variable darf auch rechts auftauchen, dann wird ihr Ursprungswert allerdings mit dem Ergebnis überschrieben.

Beispiel

Siehe hier .

Ausgabe

Du zeigst den Wert einer Variablen am einfachsten mit der Funktion MsgBox an. Diese Funktion gibt eine Meldung an den Nutzer aus. Außerdem kann man zur puren Anzeige TrayTip oder ToolTip verwenden.

Man kann Variablen aber auch an jeder anderen Stelle verwenden, an der Strings oder Zahlen stehen dürften.

Ausnahmen

Ausnahmen sind Funktionsnamen (wobei das mit Call wieder geht) und FileInstall

Beispiel Variablen manipulieren und ausgeben

```
Dim $test_var_string = 'Dies ist eine lustige "Egal"-Zeichenkette'
Dim $test_var_int = 4925

MsgBox(524320, "Testprogramm", "Der Wert der Variablen $test_var_string:" & @CRLF & $test_var_string)
; @CRLF ist eine Variable, die für einen Zeilenumbruch steht.
MsgBox(0, "Testprogramm", "Der Wert der Variablen $test_var_int:" & @CRLF & $test_var_int)

; ein bisschen Manipulation
$test_var_string = $test_var_string & ", die ein wenig gewachsen ist ;-)"
$test_var_int = $test_var_int + 5 * (8 + 7)

MsgBox(0, "Testprogramm", "Der neue Wert der Variablen $test_var_string:" & @CRLF & $test_var_string)
MsgBox(0, "Testprogramm", "Der neue Wert der Variablen $test_var_int:" & @CRLF & $test_var_int)
```

Strings

Bevor wir uns den Arrays hingeben, sehen wir uns die Funktionen der Strings näher an. Und damit meine ich nicht die Unterwäsche, davon sollte die Funktion ja klar sein *gg*.

Also, wir haben einen String, im meinem Beispiel den Source einer Internetseite, den ihr über AutoIt in einen String gelesen habt.

Der wert ist zb das:

```
$source = "<html><body> Hello world! </body> </html>"
```

Uns interessiert aber nur das "Hello World!", also weg mit den befehlen! Entweder wir arbeiten jetzt mit Stringleft, aber das ist uns zu...unflexibel ;)

Also mit _StringBetween.

Syntax: _StringBetween(\$string, \$abda, \$bisda)

In unserem Fall also:

```
#include <string.au3> ; <-- Wichtig!

$string = "<html><body> Hello world! </body> </html>"
$abda = "<html><body>"
$bisda = " </body> </html>"

$between = _StringBetween($string, $abda, $bisda)
```

Soweit so gut, lassen wir uns das Ergebniss ausgeben!

```
MsgBox(0, "String", $between)
```

AHH!!

das ist ja nix?! Aber wieso??

Ganze einfach, _StringBetween gibt ein Array zurück!

Also entweder wir arbeiten mit dem Array weiter und lassen es zb mit `_ArrayDisplay` ausgeben (`#include <array.au3>` nicht vergessen!) oder wir wollen es als String. Damit das einwandfrei funktioniert, darf `_StringBetween` aber nur EINE lösung haben!

zb das wäre falsch:

```
#include <string.au3> ; <-- Wichtig!
$string = "<html><body> Hello world! </body> </html> <html><body> Hello world! </body> </html>"
$abda = "<html><body>"
$bisda = " </body> </html>"
$between = _StringBetween($string, $abda, $bisda)
```

Wieso?

Weil es 2 werte zwischen `$abda` und `$bisda` gibt. Gibt es nur einen, können wir ihn zum String machen mit `$between = $between[0]`.

Um zu überprüfen, ob es nur 1 Ergebnis gibt, kann zb. `UBound` verwendet werden.

Strings umwandeln

Zb. Strings ins HEX Format umwandeln, über

```
_StringToHex("String")
```

also zb:

```
#include <String.au3> ; wieder: gaaaaaaanz wichtig^^
$kek = _StringToHex("22")
```

oder ins binärsystem:

```
StringToBinary ("String")
```

oder umgekehrt:

Binär zu String:

```
BinaryToString ("String")
```

um die Länge der Binärdaten zu wissen, einfach:

```
BinaryLen ("binärstring")
```

nehmen!

Strings Verschlüsseln

Mit `_StringEncrypt` geht das!

syntax:

```
_StringEncrypt ( $i_Encrypt, $s_EncryptText, $s_EncryptPassword [, $i_EncryptLevel ] )
```

Sieht kompliziert aus, ist es nicht ;)

In der Praxis so:

```
_StringEncrypt(1, "Autoit rox", "Autoit.de")
```

Das VERSCHLÜSSELT (durch die 1, 0 wäre ENTSCHLÜSSELN) den text Autoit Rox mit dem passwort Autoit.de
Optional ginge noch das Sicherheitslevel =D

Noch sicherer gehts mit `_StringReverse` dazu, das dreht den String einfach um *gg*

```
#Include <String.au3>  
_StringReverse ("Autoit") ; ergbit tiotuA
```

Sonstiges

String wiederholen:

Mit

```
_StringRepeat($string, $anzahl)
```

Kann man einen String beliebig oft wiederholen lassen,

um Platz zu sparen oder einfach aus anderen gründen =)

Praktische Anwendung zb:

```
#include <String.au3>  
MsgBox(0, '', _StringRepeat("+-", 40))  
Exit
```

Wissenswertes:

```

String zu Uppercase formatieren geht mit StringUpper ( "string" )
String zu lowercase formatieren geht mit StringLower ( "string" )
Strings teilen geht mit StringSplit ( "string", "delimiters" [, flag ] ), zb $string = StringSplit("12:03", ":")
Achtung: StringSplit gibt ein Array (siehe unten) zurück!!)

String(teile) ersetzen geht mit StringReplace($ganzerstring, $sollweg, $sollhin) zb $string = StringReplace("12:03", ":", "00")
Um die Länge eines Strings zu bekommen, genügt StringLen("string")

Um zu überprüfen ob der String eine Eigenschaft hat (zb eine Zahl ist, Buchstaben usw) geht:

StringIsAlNum
StringIsAlpha
StringIsASCII
StringIsDigit
StringIsFloat
StringIsLower
StringIsSpace
StringIsUpper
StringIsXDigit

```

Informationen finden sich in der Hilfe! =D Für Rechtschreibfehler haftet der Weihnachtsmann, Nordpolstraße 20, 19277 Nordpol.

Arrays

Hier wollen wir noch einmal kurz auf die Arrays eingehen!

Ein weiteres Tutorial zum Thema Array findet ihr hier: Array-Tut (http://home.arcor.de/bugfix/array_tut.htm)

Wie Ihr ein Array definieren könnt, wurde oben ja schon beschrieben. Hier nochmal:

```
Dim $array[100]
```

Doch wann sind Arrays nützlich?

Zum Ersten sind sie sehr nützlich, wenn man eine große Menge an Daten verarbeiten will. Wir müssen also nicht 100 neue Variablen erstellen, sondern können es mit einer einzigen Variablen lösen!

Arrays anzeigen

Mit `_ArrayDisplay` können wir den Wert eines Arrays anzeigen. Wir werden im Weiteren verlaufe immer mit `$array` arbeiten!

```
_ArrayDisplay($array, $titel)
```

Der Titel ist, was im blauen Rand oben angezeigt wird! (und im Taskmanager ;)) Natürlich müssen wir für den Titel keine Variable benutzen, sondern einen fixen String. (um auf Fehler zu prüfen: Wenn alles ok ist, wird eine 1 zurück gegeben, Fehler ergibt 0, beim `@error` tag: wenn alles ok ist, wird kein Error gesetzt, wenn die angegebene Variable kein Array war, wird `@error` auf 1 gesetzt)

Array nachträglich hinzufügen

Ihr habt jetzt also ein Array erstellt, müsst aber nachträglich noch jemanden in euer Array hinzufügen!

das geht, und zwar mittels `_ArrayAdd`

```
_ArrayAdd ($array, $Wert)
```

wenn alles OK ist, wird:

`@error` nicht gesetzt, Gibt 1 zurück (=return)

ansonsten: (wenn `$array` kein Array ist)

@error gesetzt, Gibt -1 zurück

Hier ein Beispiel, was bis jetzt erklärt wurde: (aus der Hilfe)

```
#include <Array.au3>
Dim $avArray[10] ; Array wird definiert : eigentlich 10 Personen
$avArray[0] = "JPM" ; bei 0 wird begonnen
$avArray[1] = "Holger"
$avArray[2] = "Jon"
$avArray[3] = "Larry"
$avArray[4] = "Jeremy"
$avArray[5] = "Valik"
$avArray[6] = "Cyberslug"
$avArray[7] = "Nutster"
$avArray[8] = "JdeB"
$avArray[9] = "Tylo"
_ArrayDisplay( $avArray, "Whole array" )
_ArrayAdd( $avArray,"Brian" ) ; --> Hier wird etwas in das Array hinzugefügt
_ArrayDisplay( $avArray, "Updated Array" )
```

Etwas aus einem Array löschen

Ebenso wie wir etwas zu einem Array hinzufügen können, können wir Werte aus einem Array löschen.

Das geht via `_ArrayDelete`

Syntax: `_ArrayDelete ( $avArray, $iElement )`

also: `_ArrayDelete($array, $gekuendigt)` ; Würde den Wert der Variable `$gekuendigt` aus dem Array löschen.

wenn alles OK ist, wird:

@error nicht gesetzt, Gibt 1 zurück (=return)

ansonsten: (wenn \$array kein Array ist)

@error 1 gesetzt, @error 2 bedeutet dass das Array nur 1 Wert hat, Gibt 0 zurück

Arrays sortieren

Unsere Arrays sind zwar schön definiert, aber sehr unübersichtlich! Deswegen müssen wir sie sortieren - und zwar via `_ArraySort`!

```
_ArraySort ( ByRef $a_Array [, $i_Descending [, $i_Base=0 [, $i_Ubound=0 [, $i_Dim=1 [, $i_SortIndex=0 ]]]]] )
```

Sieht kompliziert aus, ist es aber nicht! Eigentlich ganz einfach!

```
_ArraySort($array)
```

das würde alphabetisch Ordnen!

```
_ArraySort($array, 1)
```

das genaue Gegenteil: Es wird "gegen" das Alphabet geordnet!

```
_ArraySort($array, 0, 1,5)
```

Sortiert nur die ersten 5 Werte des Arrays!

Es gibt noch ein paar andere Optionen, für die schaut bitte in die Hilfe!

Ein paar weitere Funktionen mit Arrays

Wir können ein Array in das Clipboard geben , manchmal recht nützlich!

```
$array = StringSplit("a,b,c,d,e,f,g,h,i","")
$RetCode = _ArrayToClip( $array, 1 ) ; dank der 1 wird das Array nochmal sortiert! _ArrayToClip() gibt das A
MsgBox( 0, "Clipboard Test", ClipGet() ) ; Der Wert des ClipBoards wird abgerufen!
```

Arrays als Elemente eines Arrays

Es ist möglich, innerhalb eines Arrays andere Arrays zu speichern. Das ist z.B. sinnvoll, wenn ich aus einer Datei Daten einlese, die innerhalb einer Zeile mehrere Werte getrennt durch Trennzeichen enthält. (z.B. CSV-Datei)

Unsere "muster.csv" könnte so aussehen:

```
1. Spalte;2. Spalte;3. Spalte;4. Spalte;5. Spalte
2. Zeile;1. Inhalt;2. Inhalt;3. Inhalt;4. Inhalt
3. Zeile;5. Inhalt;6. Inhalt;7. Inhalt;8. Inhalt
4. Zeile;9. Inhalt;10. Inhalt;11. Inhalt;12. Inhalt
```

So kann ich die Daten zur Weiterverarbeitung in ein Array einlesen:

```
#include <File.au3>
Dim $arCSV
$csvFile = "muster.csv"
FileReadToArray($csvFile, $arCSV)
```

Im Element \$arCSV[0] ist die Anzahl der Zeilen (= Array-Elemente) hinterlegt. Jedes Element in \$arCSV enthält eine Zeile der Datei "muster.csv". Die einzelnen Werte sind durch ";" getrennt. Mithilfe von StringSplit() kann jetzt jeder Wert vereinzelt werden. Dabei wird der enthaltene String eines Elements durch das mit StringSplit() entstehende Array ersetzt.

```
For $i = 1 To $arCSV[0]
    $arCSV[$i] = StringSplit($arCSV[$i], ";")
Next
```

StringSplit() liefert ein 1D-Array mit Angabe der enthaltenen Elemente an Pos[0]. Wenn ich nun auf das Element[3] der \$arCSV zugreifen möchte und aus dem enthaltenen Array die Anzahl ausgeben will wäre dies also: \$arCSV[3] ==> [0] Wie soll hier das Element abgefragt werden ? \$arCSV[3][0] ==> Das wäre ein 2D-Array, das aber nicht vorliegt, also Falsch! (\$arCSV[3])[0] ==> Das trennt zwar logisch, ist aber syntaktisch nicht möglich, auch Falsch! Hier gibt es nur den Weg über einen Zwischenschritt:

```
$x = $arCSV[3]
MsgBox(0, '', 'Anzahl der Elemente von $arCSV[3]: ' & $x[0])
```

Arrays Mehrdimensional

Hier der Umgang mit Mehrdimensionalen Arrays am Bsp. eines 2-Dimensionalen Arrays.

Wenn ihr euch eine mehrspaltige Tabelle anschaut, so ist diese ein 2-Dimensionales Array. Eine Dimension sind die Zeilen, die zweite Dimension sind die Spalten.

Deklaration

```
Dim $array[10][4]
```

Ein Array mit 10 Zeilen und 4 Spalten.

Wie bei 1-Dimensionalen Arrays besteht auch hier ein Null-basiertes Indexsystem, d.h. Element in Zeile 1 / Spalte 1 ist \$array[0][0], Element in Zeile 1 / Spalte 2 ist \$array[0][1].

Größenbestimmung

Hier wird ebenfalls der Befehl

```
UBound()
```

verwendet. Allerdings unter zusätzlicher Angabe der gewünschten Dimension.

Für unser Beispiel:

```
MsgBox(0, 'erste Dimension', UBound($array, 1))      ; gibt den Wert: 10 zurück
MsgBox(0, 'zweite Dimension', UBound($array, 2))     ; gibt den Wert: 4 zurück
```

Anwendungsbeispiele

Nehmen wir an, wir haben in einer GUI ein 4-spaltiges ListView-Control (\$listview), das wir mit Daten aus dem Array bestücken wollen.

```
#Include <GuiListView.au3>
#include <Array.au3>

For $i = 0 To UBound($array,1) -1      ; $i = Element 1 (Index=0) bis Zeilen (Index= Zeilen -1)
    For $k = 0 To 3                    ; $k Spalten-Index (im Array und im ListView-Control)
        _GUICtrlListViewInsertItem ($listview, $k, $array[$i][$k])
    Next
Next
```

Oder hier das Gegenstück, wir lesen alle Daten aus einem 4-spaltigen ListView-Control (\$listview) aus und speichern diese in unserem Array. Da _ArrayAdd() nur für 1-Dimensionale Arrays verwendet werden kann, muß darauf geachtet werden, dass entsprechend die Größe des Arrays mit ReDim neu festgelegt wird.

```
#Include <GuiListView.au3>
#include <Array.au3>
Dim $array[1][4]

For $i = 0 To _GUICtrlListViewGetItemCount($listview) -1      ; $i = Index Listenelemente
    If $i > 0 Then ReDim $array[UBound($array, 1) +1][4]      ; ab 2. mal: Zeilen +1
    For $k = 0 To 3                                            ; $k = Index für die Spalten
        $array[$i][$k] = _GUICtrlListViewGetItemText($listview, $i, $k)
    Next
Next
```

Sortierung von zweidimensionalen Arrays

Es gilt der gleiche Befehl, wie zur Sortierung eindimensionaler Arrays. Allerdings werden mehr Parameter angegeben.

```
_ArraySort($Array [,Descending [,Base [,UBound [,Dim [,SortIndex]]]])
```

\$Array	zu sortierendes Array
Descending	0 = aufsteigend, 1 = absteigend sortieren
Base	Index des Elements, bei dem die Sortierung starten soll, i. A. 0 od. 1
Ubound	Anzahl der zu sortierenden Einträge, mit 0 wird das gesamte Array sortiert
Dim	Anzahl Elemente in 2-ter Dimension (Spalten)
SortIndex	Index der Spalte für die Sortierung, 0 = erste/ 1 = zweite Spalte usw. ...

Beispiel:

```
$array[6][2]
```

Zeile	Spalte 1	Spalte 2
1	7	5
2	3	2
3	8	6
4	2	1
5	5	4
6	4	3

Wir wollen aufsteigend nach Spalte 2 sortieren. Das gesamte Array bei Index 0 beginnend.

```
_ArraySort($Array, 0, 0, 0, 2, 1)
```

Jetzt sieht das Array so aus:

Zeile	Spalte 1	Spalte 2
1	2	1
2	3	2
3	4	3
4	5	4
5	7	5
6	8	6

TCP/IP

Jaja, das gute alte TCP/IP.. TCP/IP ist wenn man es erst einmal verstanden hat (grade bei der AutoIt Version 3.2.4.5 und aufwärts) ziemlich einfach... Die wichtigsten befehle Lauten TCPStartup, TCPSend, TCPRecv, TCPConnect, TCPListen, TCPAccept und TCPShutdown... TCPStartup startet das TCP engine von AutoIt. Das muss immer vor der ersten TCP Funktion stehen. Ein geeigneter Platz dafür ist ganz am anfang des Skriptes (Zeile 1) Aufruf: TCPStartup()

TCPShutdown beendet die TCP-Engine wieder. Ist nicht zwingend notwendig, ist aber einfach sauberer. Aufruf: TCPStartup()

TCPSend sendet Strings zu einem Client oder zu einem Server. Ab Version 3.2.4.0 kann man mit AutoIt auch binäre (binary) Daten mit autoit über TCP senden, dank befehlen wie Binary und BinaryToString und alsowas... Das ist gut um z.B. exe dateien über TCP/IP zu senden. Aufruf: TCPSend(\$mainsocket,\$daten) ; \$mainsocket wird durch TCPConnect oder TCPAccept erstellt, \$daten ist der Text der gesendet werden soll

TCPRecv received (empfängt) die Daten von einem Client oder einem Server die mit TCPSend gesendet wurden. Aufruf: TCPRecv (\$mainsocket, \$MaximaleLänge [, flag]) ;\$mainsocket siehe TCPSend, \$MaximaleLänge ist die Länge des Strings der empfangen werden soll. Wenn z.B. über TCPSend 10 Zahlen gesendet werden, sagen wir mal 1234567890, aber bei TCPRecv bei \$MaximaleLänge 5 steht kommen nur die ersten 5 Zahlen an, also 12345. , flag[Optional] wenn flag=1 dann wird binäres empfangen erzwungen. Standard ist flag auf 0 also es wird nicht erzwungen.

TCPConnect wird meistens als Client benutzt. Dieser befehl verbindet sich mit einem Server. Aufruf: TCPConnect(\$IPAdresse,\$Port) ;\$IPAdresse ist die IPAdresse auf die sich der Client verbinden soll, also z.B. 192.168.0.1 , \$Port ist der Port auf dem er sich verbinden soll. Der Port muss bei TCPListen genau gleich lauten, damit die beiden sich gegenseitig finden.

TCPListen wird meistens auf dem Server verwendet und erstellt sozusagen ein abhörer auf dem Computer. Dieser befehl muss mit TCPAccept in verbindung treten, ohne TCPAccept bringt der Befehl nichts! Aufruf: TCPListen(\$IPAdresse,\$Port,\$MaximaleVerbindungen) ;\$IPAdresse ist die IPAdresse auf dem der Server abhören soll (meistens @IPAdress1 also die eigene IP Adresse, in unserem fall aber z.B. 192.168.0.1) , \$Port ist der Port auf dem er abhören soll. Der Port muss bei TCPConnect genau gleich lauten, damit die beiden sich gegenseitig finden. , \$MaximaleVerbindungen [Optional] sind die Maximalen verbindungen die auf den Server einkommen können, wenn du den wert z.B. auf 100 setzt können sich bis zu 100 Clients auf den Server Verbinden.

TCPAccept muss immer mit TCPListen zusammen auftreten. TCPAccept läuft meistens in einer schleife während

TCPListen nur einmal aufgerufen wird. Aufruf: TCPAccept(\$mainsocket) ; \$mainsocket ist die variable die TCPListen zurückgibt (\$mainsocket=TCPListen(\$Ipadresse,\$port,[\$maxpend]))

So... Das sind jetzt so die wichtigsten befehle. Sonst kannst du auch gerne in der AutoIt Hilfe nachgucken, da gibt es noch ein paar mehr befehle. Die AutoIt Hilfe findest du normalerweise hier: C:\Programme\AutoIt3\AutoIt.chm Hier noch ein kleines Beispielskript: Server: (Den musst du in diesem Beispiel als erstes Starten!!)

```

;SERVER!! Start Me First !!!!!!!!!!!!!!!
;Autor: GtaSpider
#include <GUIConstants.au3>

; Set Some reusable info
; Set your Public IP address (@IPAddress1) here.
Dim $szIPADDRESS = @IPAddress1
Dim $nPORT = 33891

; Start The TCP Services
;=====
TCPStartUp()

; Create a Listening "SOCKET".
; Using your IP Address and Port 33891.
;=====
$MainSocket = TCPListen($szIPADDRESS, $nPORT)

; If the Socket creation fails, exit.
If $MainSocket = -1 Then Exit

; Create a GUI for messages
;=====
Dim $G00EY = GUICreate("My Server (IP: " & $szIPADDRESS & ")", 300, 200)
Dim $edit = GUICtrlCreateEdit("", 10, 10, 280, 180)
GUISetState()

; Initialize a variable to represent a connection
;=====
Dim $ConnectedSocket = -1

; Wait for and Accept a connection
;=====
Do
    $ConnectedSocket = TCPAccept($MainSocket)
Until $ConnectedSocket <> -1

; Get IP of client connecting
Dim $szIP_Accepted = SocketToIP($ConnectedSocket)

Dim $msg, $recv
; GUI Message Loop
;=====
While 1
    $msg = GUIGetMsg()

; GUI Closed
;-----
    If $msg = $GUI_EVENT_CLOSE Then ExitLoop

; Try to receive (up to) 2048 bytes
;-----
    $recv = TCPRecv( $ConnectedSocket, 2048 )

; If the receive failed with @error then the socket has disconnected
;-----
    If @error Then ExitLoop

; Update the edit control with what we have received
;-----
    If $recv <> "" Then GUICtrlSetData($edit, _
        $szIP_Accepted & " > " & $recv & @CRLF & GUICtrlRead($edit))
WEnd

If $ConnectedSocket <> -1 Then TCPCloseSocket( $ConnectedSocket )

TCPShutDown()

; Function to return IP Address from a connected socket.
;-----
Func SocketToIP($SHOCKET)
    Local $sockaddr = DLLStructCreate("short;ushort;uint;char[8]")

    Local $aRet = DLLCall("ws2_32.dll", "int", "getpeername", "int", $SHOCKET, _
        "ptr", DLLStructGetPtr($sockaddr), "int_ptr", DLLStructGetSize($sockaddr))
    If Not @error And $aRet[0] = 0 Then
        $aRet = DLLCall("ws2_32.dll", "str", "inet_ntoa", "int", DLLStructGetData($sockaddr, 3))
        If Not @error Then $aRet = $aRet[0]
    Else
        $aRet = 0
    EndIf

    $sockaddr = 0

    Return $aRet
EndFunc

```

Client: (Den musst du in diesen Beispiel als 2tes starten, also nach dem Server!!)

```

;CLIENT! Start server first please!!
;Autor: GtaSpider
;see TCPRecv example
#include <GUIConstants.au3>

; Start The TCP Services
;=====
TCPStartUp()

; Set Some reusable info
;-----
Dim $szServerPC = @ComputerName;Der Servername (@Computername: der Computername des pcs wo das skript läuft)
; Set $szIPADDRESS to wherever the SERVER is. We will change a PC name into an IP Address
Dim $szIPADDRESS = TCPNameToIP($szServerPC);Hier wird der Name zur IP umgewandelt
Dim $nPORT = 33891

; Initialize a variable to represent a connection
;=====
Dim $ConnectedSocket = -1

;Attempt to connect to SERVER at its IP and PORT 33891
;=====
$ConnectedSocket = TCPConnect($szIPADDRESS,$nPORT)

Dim $szData

; If there is an error... show it
If @error Then
    MsgBox(4112,"Error","TCPConnect failed with WSA error: " & @error)
; If there is no error loop an inputbox for data
; to send to the SERVER.
Else
;Loop forever asking for data to send to the SERVER
    While 1
        ; InputBox for data to transmit
        $szData = InputBox("Data for Server",@LF & @LF & "Enter data to transmit to the SERVER:")

        ; If they cancel the InputBox or leave it blank we exit our forever loop
        If @error Or $szData = "" Then ExitLoop

        ; We should have data in $szData... lets attempt to send it through our connected socket.
        TCPSend($ConnectedSocket,$szData)

        ; If the send failed with @error then the socket has disconnected
        ;-----
        If @error Then ExitLoop
    WEnd
EndIf

```

If-Abfragen

Eine der wichtigsten Möglichkeiten einer Programmiersprache ist es, den Programmablauf in Abhängigkeit vom Wert von Variablen steuern zu können. Dazu verwendet man in AutoIt wie in allen modernen Sprachen "If"-Anweisungen.

Syntax von If-Abfragen

```

If <expression> Then
    statements
    ...
[ElseIf expression n Then
    [elseif statements ... ]]
    ...
[Else
    [else statements]
    ...]
EndIf

```

Man gibt also einen auswertbaren Ausdruck nach If und vor Then (die in einer Zeile stehen müssen!) an. In der Regel enthält dieser Ausdruck ein Gleichheitszeichen (=), um zwei Werte (typischerweise eine Variable und den erwarteten oder zu testenden Wert) zu vergleichen. Ist dieser Ausdruck wahr (true) - also die beiden Seiten gleich, dann werden die Befehle ausgeführt, die bis Else, ElseIf oder EndIf folgen. Ansonsten nicht. Man kann statt = auch die

mathematischen Vergleichsoperatoren <, <=, >, >= und <> (ungleich) verwenden.

Ein Else-Zweig wird genau dann ausgeführt, wenn der vorhergehende If-Zweig nicht ausgeführt wurde. Mit EndIf verlässt man die If-Definition und mehrt wieder in den normalen Quelltext zurück, der immer ausgeführt wird.

Hängt nur ein Befehl von der If-Abfrage ab, so kann man die If-Abfrage verkürzt aufschreiben:

```
-----
If <expression> Then statement
-----
```

Beispiel

```
-----
$zahl = InputBox("Bitte Zahl eingeben!", "Deine Zahl bitte:", "4")
; Kurzform:
If $zahl = 7 Then MsgBox(0, $zahl, "Du hast 7 eingegeben!")
If $zahl > 5 Then
    MsgBox(0, $zahl, "Du hast eine Zahl eingegeben, die größer als 5 ist!")
ElseIf $zahl = 5 Then
    MsgBox(0, $zahl, "Du hast 5 eingegeben!")
Else
    MsgBox(0, $zahl, "Ich kann keine Aussage über die eingegebene Zahl machen. Sorry")
EndIf
-----
```

Alternativen

Sobald man mehrere ElseIf-Zweige in einer Abfrage hat, wird es unübersichtlich. Hier gibt AutoIt die beiden weiteren Abfragemöglichkeiten "Switch" und "Select" vor.

Select

Select gibt die Möglichkeit verschiedene If-Abfragen in einem Block zusammenzufassen. Die einzelnen Abfragen werden als einzelne Blöcke innerhalb der Select-Abfrage zusammengefasst, die mit Case anfangen und bis zum nächsten Case gehen.

```
-----
$One = 1
$Two = 2
Select
    Case $One < 0
        MsgBox( 0, "", "$One ist kleiner als 0" )
    Case $One > 0
        MsgBox( 0, "", "$One ist größer als 0" )
    Case $Two < 0
        MsgBox( 0, "", "$Two ist kleiner als 0" )
    Case $Two > 0
        MsgBox( 0, "", "$Two ist größer als 0" )
EndSelect
-----
```

Switch

Switch fasst alle Abfragen, die auf eine Variable zurückgehen zusammen, ist dafür aber übersichtlicher und weniger Tipparbeit als Select :P

```
-----
$Var = True
Switch $Var ; <- hier wird die Variable angegeben, die abgefragt werden soll.
    Case True; <- wäre das selbe wie "If $Var = True Then"
        MsgBox( 0, "", "$Var ist True" )
    Case False
        MsgBox( 0, "", "$Var ist False" )
EndSwitch
-----
```

Rückgabewerte von Funktionen

Fast alle Funktionen geben Werte zurück. Das kann z.B. das Ergebnis einer Aktion mit den Parametern (Mathematikfunktionen oder Stringfunktionen). In diesem Fall wird das Flag @error (das sich wie eine Variable auslesen lässt) mit Informationen gefüllt, ob die Ausführung geklappt hat oder nicht.

Funktionen, die nichts zurückgeben müssen, melden Schwierigkeiten in der Regel über den normalen Rückgabewert.

Für alle eingebauten Funktionen ist in der Hilfe aufgelistet, wo und mit welchen Werten sie was signalisieren.

Den Rückgabewert fängt man quasi auf, indem man eine Variable und ein Gleichheitszeichen vor den Funktionsaufruf stellt. Der Interpreter führt zunächst die Funktion aus und ersetzt ihren Aufruf durch den Rückgabewert. Dann findet dessen Zuweisung zur Variablen statt.

Das Flag @error müssen Sie direkt nach dem Ausführen der Funktion abrufen, wenn Sie das möchten, denn die nächste Version kann wieder in dieses Flag schreiben, um ihrerseits bestimmte Ergebnisse anzuzeigen.

Beispiel Rückgabewerte

```
Dim $notepad
; Notepad starten und "die PID des Prozesses, der gestartet wurde", in $notepad speichern
$notepad = Run("notepad", "", @SW_MAXIMIZE)
; Warten, bis Notepad aktiv ist und nur, wenn es innerhalb von 5 Sek. aktiv wird, Text schreiben
If WinWaitActive("Unbenannt - Editor", "", 5) Then
    ; Einige Tastendrücke senden
    Send("Dies ist Text, den AutoIt so gesendet hat, als käme er vom Benutzer...")
    Sleep(1000)
EndIf
; Den Prozess, den Run vorhin gestartet hat, schließen (erspart auch die Speichern-Abfrage)
ProcessClose($notepad)
```

Wenn im Ausdruck bei If kein Vergleich durchgeführt, so wird quasi gegen ... <> 0 verglichen, denn alle Werte außer 0 ergeben True und erfüllen damit die Abfrage.

Fehlerbehandlung

Für professionelle Programme ist es wichtig, möglichst viele Fehler abzufangen - sei es vom User oder vom Programm selbst.

Leider bietet AutoIt (noch) keine Struktur a la try...catch...finally o.ä. Das bedeutet, dass die Fehlerbehandlung auf Basis der Rückgabewerte und des @error-Makros stattfinden muss. Das ist insbesondere dann lästig, wenn man mehrere Fälle abdecken muss.

Bei Usereingaben gibt es eine recht komfortable Möglichkeit mit Regulären Ausdrücken Eingaben auf Korrektheit zu prüfen.

Für den Programmablauf bleiben aber tatsächlich nur die beiden oben angesprochenen Werte, die gesetzt werden.

Schleifen

AutoIt unterstützt eine Vielzahl von Schleifentypen, um es Ihnen zu ersparen, Code vielfach untereinander zu hängen und es erst zu ermöglichen, die Anzahl der Durchläufe von Gegebenheiten abhängig zu machen, die erst zur Laufzeit klar werden.

Eine Schleife ist der Weg, einen Skriptteil mehrfach auszuführen. Wahrscheinlich willst du einen Befehlsblock entweder x-mal wiederholen oder solange, bis eine bestimmte Bedingung wahr oder falsch ist. In AutoIt gibt es folgende Schleifen-Typen:

```
* For...Next
* While...WEnd
* Do...Until
```

Obwohl alle ähnliche Dinge ausführen, sind sie doch leicht verschieden und es wird je nach Situation zu entscheiden sein, welcher Typ zu benutzen ist. Hier sei beispielshalber die For-Schleife kurz angesprochen.

For-Beispiel

```
For $i = 5 to 1 Step -1
    MsgBox(0, "Countdown!", $i)
Next
MsgBox(0, "", "Fertig!")

For $j = 1 to 5
    MsgBox(0, "Countup!", $j)
Next
MsgBox(0, "", "Fertig!")
```

Man gibt also eine Zählvariable an, sowie Start- und Endwert. Zusätzlich kann man nach dem Schlüsselwort step noch die Schrittweite, also den Wert, der bei jedem Durchlauf zur Zählvariablen dazugezählt wird, angeben.

Alle Befehle bis zum Schlüsselwort Next werden dann wiederholt ausgeführt. Danach wird das Skript unterhalb der Schleife fortgesetzt.

While-Schleife

Mit der While-Schleife werden in der Regel Endlosschleifen realisiert. Vor allem bei GUI-Skripten muss es immer eine solche Endlosschleife geben, damit das Skript nicht unerwartet sein Ende erreicht. Diese Schleife sieht dann so aus:

While-Endlosschleife

```
While 1
    Sleep(10)
WEnd
```

Eigene Funktionen

Wie erstelle ich eine UDF, was ist zu beachten?

Eine **User Defined Function** unterscheidet sich von einer 'gewöhnlichen' programmbezogenen Funktion dadurch, dass sie allgemeingültig anzuwenden ist. Es ist zu unterscheiden ob übergebene Parameter von der Funktion verändert werden oder ob damit ein Rückgabewert erstellt wird. Wird z.B. ein Array mit der Funktion verändert, übergebe ich das Array in der Form: "ByRef \$Array". Wenn Parameter mit Standardwerten vorbelegt werden, werden sie somit optional. D.h. "\$ind=0" ruft die Funktion mit "0" für die Variable "\$ind" auf OHNE dass die Variable im Funktionsaufruf enthalten sein muss. z.B.

```
MyFunc($var1, $ind=0)
```

Der Aufruf:

```
_MyFunc(5, 0)
```

ist also identisch mit:

```
_MyFunc(5)
```

Der Funktionsname sollte eindeutig auf den Funktionsinhalt hinweisen. Ein wichtiger Bestandteil ist auch die Fehlerbehandlung. Es müssen alle falschen Werte abgefangen werden.

Anhand einer Beispielfunktion gehen wir das jetzt Schritt für Schritt durch.

Funktionsname:	_Array2DDelete	In einem 2D-Array einen Eintrag am gegebenen Index löschen.
Parameter:	ByRef \$ARRAY	Array mit zu löschendem Eintrag
	\$iDEL	Index des zu löschenden Eintrags

```
Func _Array2DDelete(ByRef $ARRAY, $iDEL)
```

Prüfen ob übergebenes Array auch ein Array ist, wenn nicht wird @error auf '1' gesetzt und die Funktion mit Rückgabe von '0' beendet.

```

If ( Not IsArray($ARRAY) ) Then
    SetError(1)
    Return 0
EndIf

```

Prüfen ob Parameter für Index gültig ist, also nicht kleiner '0' und nicht größer als UBound(Array)-1, wenn nicht wird @error auf '2' gesetzt und die Funktion mit Rückgabe von '0' beendet.

```

If ( $iDEL < 0 ) Or ( $iDEL > UBound($ARRAY)-1 ) Then
    SetError(2)
    Return 0
EndIf

```

Hier folgt die eigentliche Funktion

```

Local $UBound2nd = UBound($ARRAY,2)
If @error = 2 Then
    Local $arTmp[UBound($ARRAY)-1]
    $k = 0
    For $i = 0 To UBound($ARRAY)-1
        If $i <> $iDEL Then
            $arTmp[$k] = $ARRAY[$i]
            $k += 1
        EndIf
    Next
Else
    Local $arTmp[UBound($ARRAY)-1][$UBound2nd]
    $k = 0
    For $i = 0 To UBound($ARRAY)-1
        If $i <> $iDEL Then
            For $l = 0 To $UBound2nd-1
                $arTmp[$k][$l] = $ARRAY[$i][$l]
            Next
            $k += 1
        EndIf
    Next
EndIf
$ARRAY = $arTmp

```

Der erfolgreiche Abschluß der Funktion wird üblicherweise durch Rückgabe von '-1' bestätigt, es sei denn das Funktionsergebnis wird durch Return zurückgegeben.

```
Return -1
EndFunc ;==>_Array2DDelete
```

Es sollte versucht werden, weitestgehend auf Includes in der Funktion zu verzichten. So lässt sich z.B.

```
#Include <Array.au3>
ArrayAdd($Array, $Value)
```

folgendermaßen ersetzen:

```
ReDim $Array[Ubound($Array)+1]
$Array[Ubound($Array)-1] = $Value
```

Maus- und Tastatursimulation

[Weitere Erklärungen folgen]

Ini-Dateien

Werte in einer .ini-Datei zu speichern, hat viele Vorteile!

1. Es werden keine Daten in die Registry geschrieben, es sind dort also keine Schreibrechte notwendig
2. Man kann die Ini ganz leicht kopieren, was Backups stark vereinfacht
3. Der Zugriff ist über eine standardisierte Struktur mit frei wählbaren Namen möglich

Werte schreiben

Mit dem Befehl IniWrite kannst du Werte in eine Ini-Datei schreiben. Dazu musst du folgende Informationen angeben: Den Pfad der Ini-Datei, den Namen der "Section", den Namen des Schlüssels und natürlich den Wert, der geschrieben werden soll.

Der volle Syntax von IniWrite lautet also:

```
IniWrite(Datei, Section, Schlüssel, Wert)
```

Werte lesen

Mit dem Befehl IniRead kannst du Daten aus einer Ini-Datei auslesen. Dazu gibst du folgendes an: Den Pfad der Ini-Datei, den Namen der "Section", den Namen des Schlüssels und einen Wert, der zurückgegeben werden soll, wenn der angefragte Schlüssel nicht existiert. Allgemein:

```
$name = IniRead(Datei, Section, Schlüssel, Standardwert)
```

Mit dem Standardwert kann man überprüfen, ob ein Schlüssel existiert.

Beispiele

Hier ein Beispiel für das Arbeiten mit Ini-Dateien:

```
; Skript zu Ini-Dateien von Huggy
$name = InputBox("Name", "Hallo, wie ist dein Name?") ; Variable $name wird definiert
IniWrite(@ScriptDir & "\Klasse.ini", "Name", "user", $name) ; $name wird in die Ini Datei geschrieben
$name2 = IniRead(@ScriptDir & "\Klasse.ini", "Name", "user", "0")
MsgBox(0, "Dein Name", $name2)
```

Wir haben \$name durch eine InputBox definiert, den Wert in "Klasse.ini" im Skriptverzeichnis gespeichert. Dann haben wir \$name2 definiert, und zwar durch den gerade eingetragenen Wert. Natürlich hat dies in der Praxis wenig Sinn, aber für Lernzwecke sollte es dienen können.

Ein etwas komplexeres Script:

```
; von Huggy
$wert1 = IniRead(ScriptDir & "\Klasse.ini", "1A", "name", "0")
if $wert1 = "0" Then
    $name = InputBox("Dein Name", "Sagst du mir deinen Namen?")
else
    $name = $wert1
endif
MsgBox(0, "Dein Name", $name)
```

Wir haben überprüft, ob der Eintrag "name" , in der Section "1A" existiert. Wenn er existiert, ist \$name gleich dem Wert dieses Eintrags. Wenn nicht, wird \$name durch eine InputBox definiert und schließlich angezeigt.

Registry

Zum Arbeiten mit der Registry bietet AutoIt ähnliche Funktionen an wie für Ini-Dateien. Es gibt RegRead sowie RegWrite. Die Registry hat viele Vorteile, zum Beispiel:

Es muss keine extra .ini Datei erstellt werden, das Auslesen der Werte geht sehr schnell, die Befehle sind simpel und man kann damit Programme in den Autostart geben.

Nachteile:

Will man Einstellungen zb auf den USB-Stick oder einfach nur auf der Festplatte sichern, muss man sich entweder die Werte rausschreiben, oder das ganze durch ein anderes Script machen lassen, was wiederum ziemlich umständlich ist. Nach und nach "vermüllt" man die Registry, wenn man nicht regelmäßig die nichtmehr gebrauchten Werte löscht. (Achtung, an der Registry was zu verändern ist für Neulinge nicht immer ungefährlich!)

F: Wie schreibe ich einen Wert in die Registry A:

Mit RegWrite

```
(Registry Write) --> RegWrite ( "keyname" [, "valuename", "type", value] )
```

also zum Beispiel:

```
RegWrite("HKEY_LOCAL_MACHINE\SOFTWARE\Test", "TestKey", "REG_SZ", "Hello this is a test")
```

Gehen wir nach und nach die einzelnen Teile durch.

HKEY_LOCAL_MACHINE\SOFTWARE\Test: Der Pfad wo der Wert geschrieben werden soll. TestKey: Der Name des Werts. "REG_SZ", "REG_MULTI_SZ", "REG_EXPAND_SZ", "REG_DWORD", or "REG_BINARY": Der Typ des Wertes Hello this is a test: Der Wert unseres Eintrags.

Wir haben also folgendes erstellt:

Einen Registry Eintrag in HKEY_LOCAL_MACHINE\SOFTWARE\Test , namens TestKey. Es ist ein Wert des Types REG_SZ mit dem Wert: Hello this is a test.

Und genauso einfach ist es auch, Werte zu lesen!

Das geht mit:

```
RegRead ( "keyname", "valuenam" )
```

Keyname: der pfad zum Key und Valuenam ist der Name des Werts (im oberen Beispiel also "HKEY_LOCAL_MACHINE\SOFTWARE\Test" und "TestKey"

Am besten ist, wir definieren hierfür eine Variable \$path, ich zeige es euch am Beispiel meines Firefox 2

```
$path = RegRead("HKEY_LOCAL_MACHINE\SOFTWARE\Mozilla\Mozilla Firefox\2.0.0.1 (de)\Main", "PathToExe")  
MsgBox(0, "", $path) ; getestet - es funktzt natürlich :-)
```

Bei mir wird die Variable \$path also richtig definiert, und zwar mit C:\PROGRA~1\MOZILL~1\FIREFOX.EXE

So liest man also Werte aus!

Werte löschen geht auch ganz einfach:

```
RegDelete ( "keyname" [, "valuenam"] )
```

Also zB

```
RegDelete("HKEY_LOCAL_MACHINE\SOFTWARE\Mozilla\Mozilla Firefox\2.0.0.1 (de)\Main", "PathToExe")
```

Ihr könnt ganz leicht überprüfen ob es fehlgeschlagen ist oder ob alles gut gegangen ist, denn im 2en Fall ist der Return-Wert 1, wenn der Wert nicht existiert hat 0 und wenn es einfach nicht geklappt hat, wird eine 2 zurück gegeben!

ich teste das jetzt nicht, da ich weiß das es funktioniert und ich mir meinen FF nicht zerschießen will :-)

Reguläre Ausdrücke

Die Beta-Versionen von AutoIt unterstützen Reguläre Ausdrücke als Kriterium für Suchen, Ersetzen und Erkennen (z.B. von Fenstertiteln). Letzteres erst seit Version 3.2.1.12. StringRegExp sucht nach Zeichenkettenteilen, die der RegEx-Vorschrift entsprechen, StringRegExpReplace ersetzt diese durch andere Werte. Die englische Beta-Hilfe erklärt kurz die Bedeutung der einzelnen Steuerzeichen.

RegEx ist ein sehr mächtiges Instrument - aber nur in den richtigen Händen. Oft ist es wesentlich weniger Code als andere Lösungen, die dafür aber oft viel schneller umgesetzt sind. Alternativen können z.B. _StringBetween (seit 3.2.1.13) oder StringInStr sein.

Bsp.: Prüfung auf Pfadkonformität

Mit der folgenden Funktion wird ein gegebener String daraufhin untersucht, ob er Konformität zu den Erstellungsregeln eines üblichen Pfades besitzt. Die Funktion prüft den String mit Hilfe eines Regulären Ausdrucks.

Die Funktion wird mit folgenden Parametern aufgerufen:

```

$PATHSTRING:      der zu prüfende Pfad
$TYPE (optional): -1 (default) Pfad mit Laufwerksbuchstabe, beendender Backslash optional
                  0 Pfad ohne Lw.-buchstabe, beendender Backslash erforderlich
                  1 Pfad ohne Lw.-buchstabe, beendender Backslash optional
$SZ (optional):   Sonderzeichen die mit im Pfad enthalten sein dürfen (Umlaute, ! etc.)
Rückgabewert:     1 wenn korrekter Pfad, sonst 0

```

```

If _PathCheck("C:\Picture\Löffel", -1, "äöü") = 1 Then
    MsgBox(0, "", "Pfad korrekt")
Else
    MsgBox(0, "", "Pfad nicht korrekt")
EndIf

Func _PathCheck($PATHSTRING, $TYPE=-1, $SZ=-1)
    Local $SONDERZEI = ""
    If $SZ <> -1 Then
        $SONDERZEI = $SZ
    EndIf
    Local $PATT_FULL = "((?<!.)([a-zA-Z]\:\\)(?!\\s)(([\\w-+." & $SONDERZEI & _
        "]*((?<!\s)(\\s{1})(?!\\|)(?<!\|)(\\s{1})(?!\\s))*)(\\|\\|)(?!\\.))"
    Local $PATT_O_LW_M_BS = "((?<!.)(\\|)(?!\\s)(([\\w-+." & $SONDERZEI & _
        "]*((?<!\s)(\\s{1})(?!\\|)(?<!\|)(\\s{1})(?!\\s))*)(?<!\|)(\\|)(?!\\.))"
    Local $PATT_O_LW_O_BS = "((?<!.)(\\|)(?!\\s)(([\\w-+." & $SONDERZEI & _
        "]*((?<!\s)(\\s{1})(?!\\|)(?<!\|)(\\s{1})(?!\\s))*)(?<!\|)(\\|)(?!\\.))"
    Local $RET
    Switch $TYPE
        Case -1 ; prüft auf "C:\abd\vlb" bzw. "C:\abd\vlb\"
            $RET = StringRegExp($PATHSTRING, $PATT_FULL)
        Case 0 ; prüft auf "\agv\ad\"
            $RET = StringRegExp($PATHSTRING, $PATT_O_LW_M_BS)
        Case 1 ; prüft auf "\agv\ad\"
            $RET = StringRegExp($PATHSTRING, $PATT_O_LW_O_BS)
    EndSwitch
    Return $RET
EndFunc ;==>PathCheck

```

Eine ausführliche Erläuterung, wie der Reguläre Ausdruck sich zusammensetzt, findet ihr hier (<http://home.arcor.de/bugfix/PathCheck.htm>) .

Dateien

AutoIt kann direkt mit Textdateien zusammenarbeiten. Es ist also nicht nötig, die Dateien etwa in Notepad zu öffnen und dann mit Send o.ä. zu bearbeiten. AutoIt kann Dateien lesend oder schreibend mit FileOpen öffnen. Der Modus, der als Parameter übergeben wird ist dabei entscheidend für die möglichen Manipulationen. FileWriteLine, FileWrite, FileRead, FileReadLine dienen im Wesentlichen zum Lesen und Schreiben der Daten. FileClose schließt die geöffnete Datei wieder und gibt sie damit auch wieder für andere Programme zum Zugriff frei. Sollte eine Datei nicht per FileClose geschlossen werden, wird sie am Ende des Skriptes automatisch geschlossen. AutoIt kann maximal 64 Dateien parallel geöffnet halten.

Man kann eine Datei einfach in ein Array einlesen, indem man _FileReadToArray aus der File.au3 (diese includen!) nutzt. In Arrays lassen sich strukturierte Daten oft leichter bearbeiten (sortieren, einzelne Zeilen entfernen usw.). Das Ergebnis schreibt man dann mit _FileWriteFromArray zurück in die Datei.

Werte aus dem Speicher lesen / bearbeiten

AutoIt kann Werte einer Speicheradresse lesen und sogar verändern. Dazu braucht man jedoch die aktuelle UDF Memory.au3 (<http://www.autoitscript.com/forum/index.php?showtopic=28351&view=findpost&p=201269>) . Damit erschließen sich folgende Grundfunktionen:

```

$pid = WinGetProcess("Fenstertitel")
$info = _MemoryOpen($pid)
$read = _MemoryRead(speicheradresse, $info)
_MemoryWrite(adresse, $info, wert)

```

Einige Beispiele dazu: Im folgenden sehen wir eine Art "Cheat", und zwar wird bei "3D Pinball" von Microsoft der

Punktstand erhöht. Zuerst habe ich mir davor mit einem Programm (Link steht am Ende dieses Beitrages) die Speicheradresse des Punktstandes herausgesucht.

```
HotKeySet("{^}", "_Incrase")

$nummer = InputBox("Nummer", "UM was soll erhöht werden?")
$pid = WinGetProcess("3D-Pinball für Windows - Space Cadet") ; hier besorge ich die PID
$info = _MemoryOpen($pid) ; ich öffne den Wert mit dem oben herausbekommenen PID
$read = _MemoryRead(0xAB3C14, $info) ; ich lese den gerade geöffneten Wert in der Speicheradresse 0xAB3C14.
iTrayTip("Trainer", "Got Points: " & $read, 2) ; Zeigt den Aktuellen Punktstand als TrayTip an.
sleep(2000)

Func _Incrase()
    $read = _MemoryRead(0xAB3C14, $info) ; ich lese wieder aus dem Speicher mit der adresse 0xAB3C14 .
    $new = $read + $nummer ; mein neuer Punktstand: Aktueller Punktstand + vorher definierte Zahl.
    _MemoryWrite(0xAB3C14, $info, $new) ; es gibt 2 Werte. In beiden ist der Punktstand gespeichert,
    _MemoryWrite(0xC2AEBA, $info, $new) ; und es funktioniert nur, wenn man beide Werte ändert!
EndFunc

While 1
    sleep(100000)
Wend
```

Hier noch ein kleines Beispielscript für MineSweeper:

```
$pidm = WinGetProcess("MineSweeper")
$info = _MemoryOpen($pidm)
$readm = _MemoryRead(0x0100579C, $info)
MsgBox(0, "", $readm) ; aktueller Punktstand wird angezeigt

Func _Reset()
    $newm = 0
    _MemoryWrite(0x0100579C, $info, $newm) ; Gebrauchte Zeit wird auf 0 resettet
EndFunc

Func _weniger()
    $neweniger = $readm - 1
    _MemoryWrite(0x0100579C, $info, $neweniger) ; gebrauchte Zeit wird um 1 Sekunde verringert.
EndFunc
```

Das ist nur ein Teil der vielen Möglichkeiten! Schaut es euch einfach an!

ein sehr effektives Programm zum auslesen / bekommen von Speicheradressen heißt Cheat-Engine
(<http://www.cheatengine.org/>)

GUI

GUIControl

Colored Button

Bisher besteht standardmäßig nicht die Möglichkeit, gefärbte Button zu verwenden. Es gibt nun mehrere Varianten, trotzdem zu farbigen Button zu kommen.

Button Style verwenden

Die wahrscheinlich einfachste Variante, man erstellt seinen Button mit dem Style \$BS_ICON oder \$BS_BITMAP und weist ein erstelltes Icon/Bitmap dem Button zu.

```
GUICtrlCreateButton("", left,top,width,height, $BS_ICON)
GUICtrlSetImage(-1, $IconPfad & "Icon.ico") ; analoges Vorgehen mit Bitmap
```

Nachteil: Wenn ich Veränderungen am Schriftbild oder der Farbe vornehmen möchte, muß ich jedes Mal ein neues Icon/Bitmap erstellen.

echte farbige Button

Es gibt eine Lösung von **gafrost**, die wirklich gut ist. Allerdings ist der Code auch recht umfangreich. Da stellt sich die Frage, ob ich diesen Aufwand betreiben will, um z.B. 3 Button farbig darzustellen. Doch macht euch selbst ein Bild. colored button
(<http://www.autoitscript.com/forum/index.php?showtopic=25077&hl=colored%20%20button&st=0>)

Label als Button

Diese Lösung von **rakudave** stellt eine gute Alternative dar und läßt sich, da vom Umfang recht gering, problemlos in jedes Script einbinden. Ich habe ein kleines Bsp. erstellt, um die Wirkungsweise zu verdeutlichen. Ihr findet es hier: Label als Button (<http://autoit.aufwaerts.de/thread.php?postid=13584#post13584>)

List Control

Eine einfache Form Daten aufzulisten bietet das List-Control. Einige Möglichkeiten, Daten in eine Liste einzutragen, zu löschen oder auszulesen etc. habe ich in einer Beispielanwendung zusammengefaßt: List Control
(<http://autoit.aufwaerts.de/thread.php?postid=5214#post5214>)

Koda(FormDesigner)

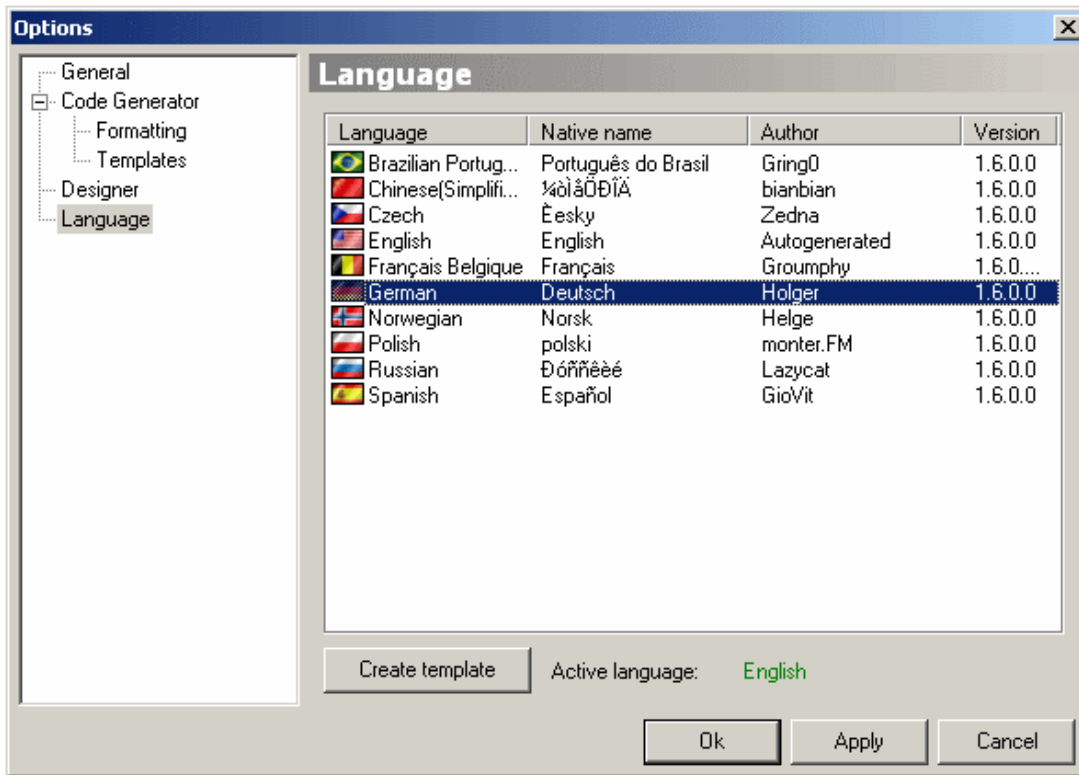
Der Koda(FormDesigner)

Der Koda (FormDesigner) ist ein Programm um GUI's (Graphical User Interface) für AutoIt Programme zu schreiben und zu gestalten. Die Aktuelle Bezugsquelle ist die Entwicklerseite des Koda(FormDesigner)
(<http://www.autoitscript.com/fileman/users/lookfar/formdesign.html>) Teams (ein Lob an dieser Stelle für die Entwickler). In dieser Doku werde ich erstmal darauf eingehen wie man ein GUI erstellt in ein Programm einbindet und wieder bearbeitet.

Koda(FormDesigner) auf deutsch

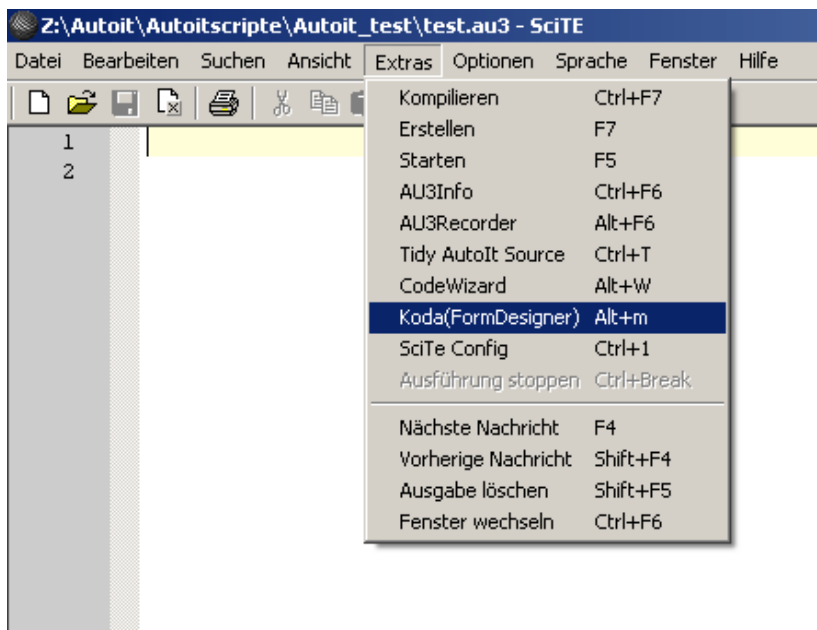
Dies ist ziemlich einfach.

Man klickt auf dem Menuepunkt "Options" und wählt danach "Settings". Hier dann "German" anwählen nun OK klicken und Koda präsentiert sich in deutscher Sprache.

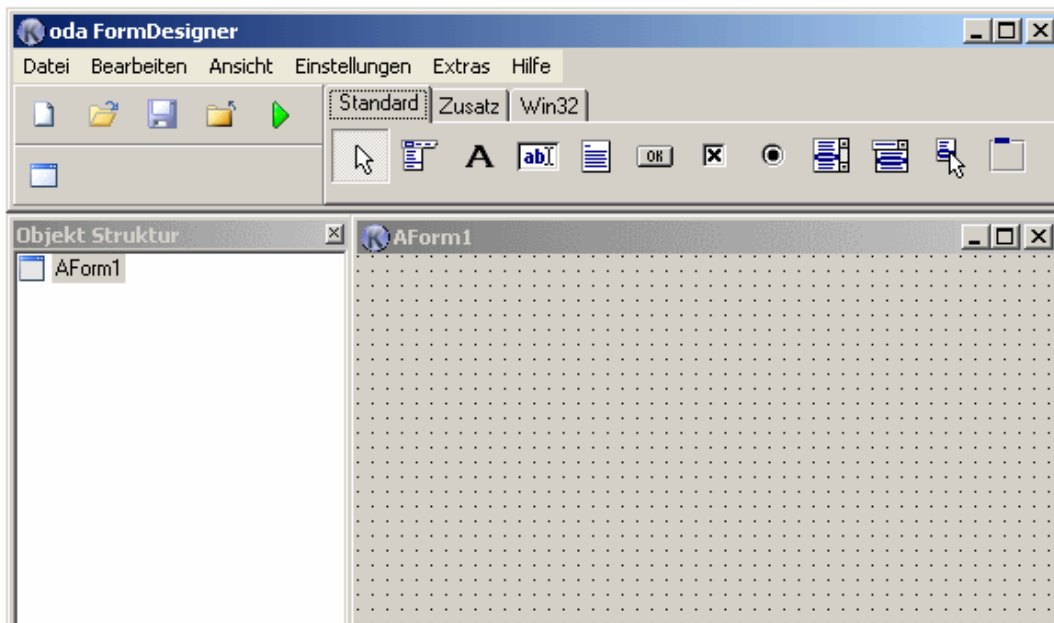


GUI erstellen

Zuerst eine neue *.au3 Datei in dem Programm SciTE anlegen. Danach unter dem Punkt "Extras" Koda (FormDesigner) auswählen und starten. Wahlweise kann man den FormDesigner auch mit "Alt+m" starten.



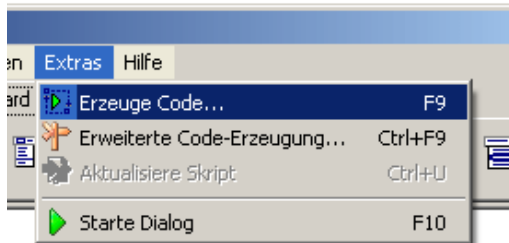
Nach dem klicken öffnet sich der Koda (FormDesigner) in einem neuen Fenster.



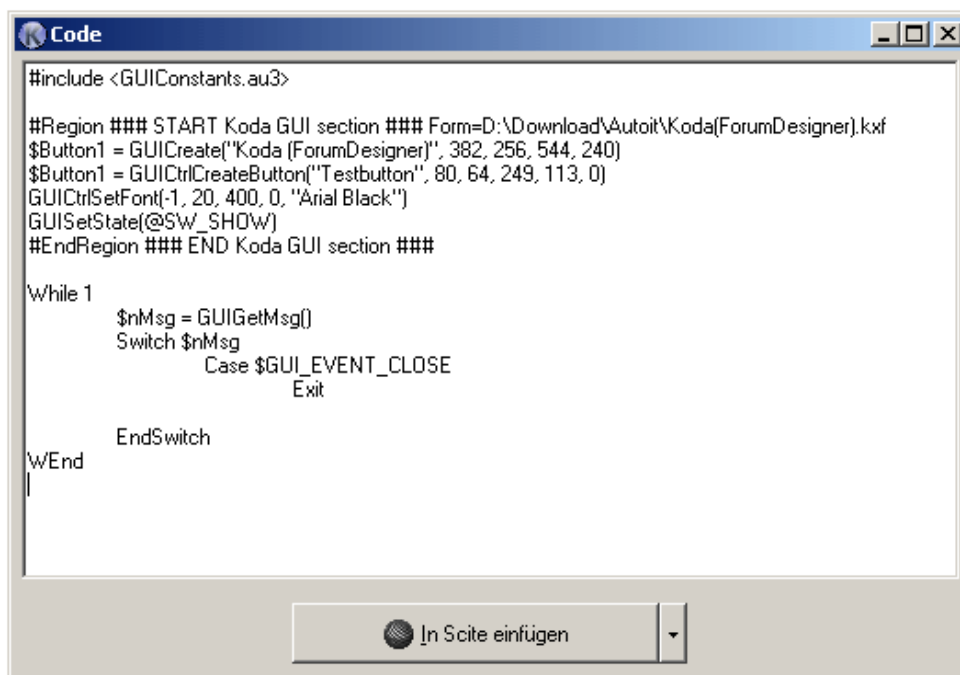
Hier wird nun das GUI erstellt. (Auf die Erstellung des GUI gehe ich hier noch nicht ein da es sich fast selber erklärt. Nach dem erstellen wird das GUI als XML Datei gespeichert. Dazu einfach auf Datei - Speichern unter klicken und einen Namen für das GUI vergeben.

Dieser ist später wichtig wenn man das GUI bearbeiten möchte.

Die Dateierweiterung nennt sich ".kxf" (Man sollte die Datei genauso nennen wie das Programm was man schreibt, so findet man sie leichter.) Nachdem die Datei gespeichert wurde klickt man auf "Extras - Erzeuge Code.."



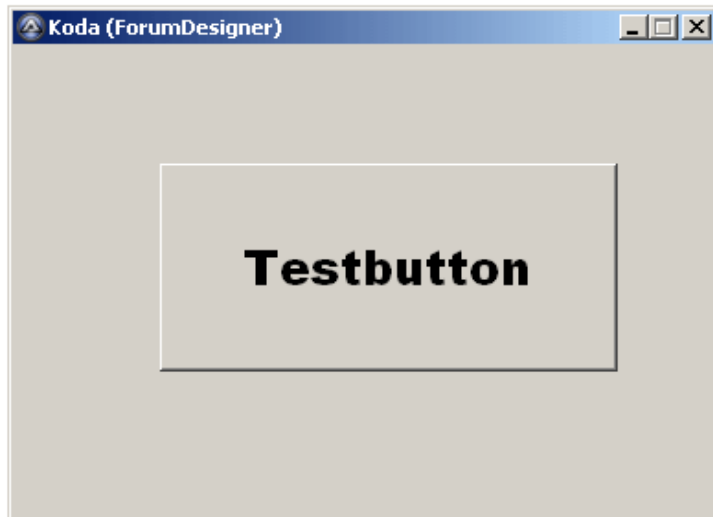
Folgendes Fenster erscheint indem der Quellcode angezeigt wird.



Nun auf "In Scite einfügen" klicken und das GUI ist erstellt und man kann in der While-Schleife Funktionen usw. einfügen.

!!!Achtung alles was zwischen !!!

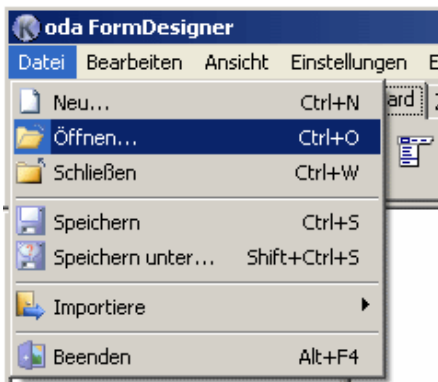
"#Region und #EndRegion" steht ist später wichtig um das GUI mit dem Koda (FormDesigner) zu bearbeiten. Dieser Bereich darf nicht von Hand editiert werden, da er beim nächsten Aufruf und ändern von Koda eh überschrieben wird. So nun ist das GUI erstellt und wartet auf seine Bearbeitung.



GUI bearbeiten

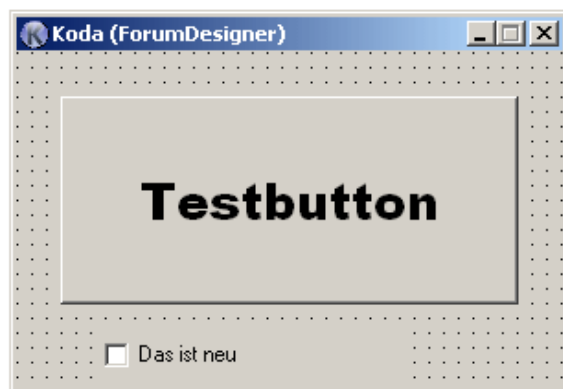
So nun kann man das GUI ganz einfach wieder öffnen und bearbeiten.

Dazu die ".au3" Datei und den Koda (FormDesigner) öffnen (wie oben beschrieben).



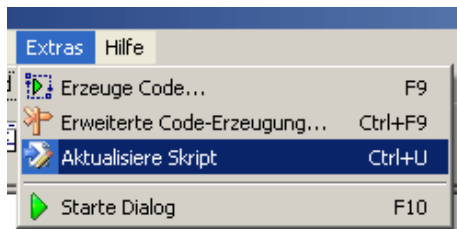
Dann die passende ".kxf" Datei auswählen und in das Programm einladen.

Man bekommt dann das GUI wieder angezeigt und kann es wunderbar bearbeiten und verändern.



Nachdem alle Änderungen und Anpassungen durchgeführt wurden, einfach wieder im Menu auf "Datei" und danach auf "speichern" klicken. Nun ist das GUI wieder gespeichert und man kann den Quellcode wieder zurück schreiben lassen.

Hierzu wählt man aus dem Menu "Extras" nicht den Punkt "Erzeuge Code..." sondern "Aktualisiere Skript".



Nachdem klicken auf "Aktualisiere Skript" kommt keine Rückmeldung, (Schade vielleicht könnte man hier was verbessern) wartet man 2 bis 3 Sekunden und dann schliesst man den Koda (FormDesigner) mit "Datei" und dann "Beenden".

So nun kann man ganz einfach seine GUI's erweitern , speichern und wieder öffnen. Hier noch der Beweis:

```

D:\Download\Autoit\Koda(ForumDesigner).au3 * SciTE
Datei Bearbeiten Suchen Ansicht Extras Optionen Sprache Fenster Hilfe
[Icons]
1  #include <GUIConstants.au3>
2
3  #Region ### START Koda GUI section ### Form=d:\download\autoit\koda(forumdesigner).kxf
4  $Button1 = GUICreate("Koda (ForumDesigner)", 298, 186, 451, 276)
5  $Button1 = GUICtrlCreateButton("Testbutton", 24, 24, 249, 113, 0)
6  GUICtrlSetFont(-1, 20, 400, 0, "Arial Black")
7  $Checkbox1 = GUICtrlCreateCheckbox("Das ist neu", 48, 152, 161, 25) ; <---- Das kam dazu :->
8  GUISetState(@SW_SHOW)
9  #EndRegion ### END Koda GUI section ###
10
11 While 1
12     $nMsg = GUIGetMsg()
13     Switch $nMsg
14         Case $GUI_EVENT_CLOSE
15             Exit
16     EndSwitch
17 EndWhile
18
19

```

AUS TECHNISCHEN GRÜNDEN MUSS DAS TUTORIAL GETEILT WERDEN! DEN ZWEITEN ZEILT FINDET IHR HIER: Tutorial2 DANKE FÜR EUER VERSTÄNDNISS! =)

Von „<http://wiki.autoit.de/wiki/index.php/Tutorial>“

WIE ALT WIRST DU?

- Diese Seite wurde zuletzt am 27. Oktober 2007 um 15:55 Uhr geändert.
- Diese Seite wurde bisher 22.094 mal abgerufen.
- Inhalt ist verfügbar unter der GNU Free Documentation License 1.2.
- Datenschutz
- Über AutoItWiki
- Impressum