Florian Schieder

Programmieren mal einfacher
Windows-Anwendungen mit AutoIt 3

1. Einführung

Schon immer mal Lust gehabt, zu zeigen, was man technisch drauf hat, und ein eigenes Programm zu entwickeln? Aber trotzdem schreckt man doch jedes mal davor zurück, weil das Lernen von Sprachen wie C, C++ oder Java nicht unbedingt ein Spaß ist. Meistens wird schon bei komplexeren Konsolenanwendungen der Code ziemlich unübersichtlich – von Benutzeroberflächen gar nicht zu schweigen. Microsoft mit Visual Basic, Visual C++ und Konsorten hingegen macht es nicht nur mit den Lizenzbedingungen, sondern auch mit .NET-Framework-Abhängigkeiten schwerer. Bei Qt-Bibliotheken und Frameworks ist das nicht anders. Anders hingegen mit AutoIt. Die Geschichte von AutoIt begann 1999, als der Brite Jonathan Bennett mit einem Team von professionellen C++ Entwicklern (dem "AutoIt Team") die Skriptsprache AutoIt entwickelte. Ohne Probleme kann man einfach Skripte mit Benutzeroberflächen erstellen. Einfache kompilierte Programme sind selten größer als 1 Megabyte und auf jedem System – Windows XP, Windows Servern bis zu Windows 10 lauffähig. (Anmerkung am Rand: AutoIt steht für "Automatize It", weil man über simulierte Tastatur- und Mausbefehle unter anderem Softwareinstallationen automatisieren kann.)

Warum das denn nicht für Linux oder Mac OS, so cool und einfach wie das doch wäre?

Dafür gibt es – für mich zumindest – zwei triftige Erklärungen: die vielen DLLs, die dynamisch aufgerufen werden und Werte zurückliefern – oder die Windows-API. Zumindest wird es so bei Google und anderen Quellen erklärt. AutoIt-Programme lassen sich aber größtenteils (nicht problemlos!) über Emulatoren wie WINE starten.

Und was wäre, wenn ich an die Grenzen von AutoIt stoße?

Wo man an die Grenzen von AutoIt stößt, kann man z.B. in C# eine DLL erstellen und die Funktion dynamisch aufrufen. Oder interne Windows-DLLs aufrufen. Alles erlaubt.

Darf man AutoIt-Skripte oder sogar die kompilierten Programme verkaufen?

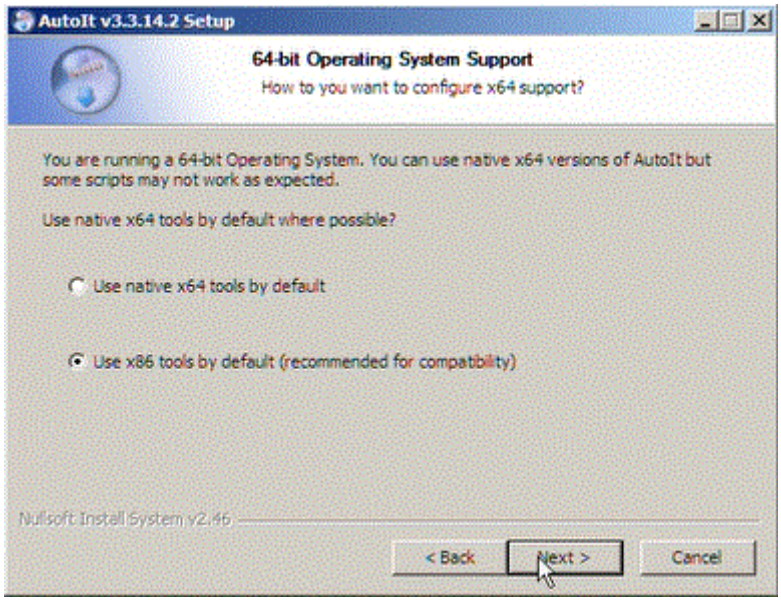
Ja! Als Gewissensstütze ein Auszug aus den Lizenzbedingungen:
"*You may use the SOFTWARE PRODUCT for commercial purposes. You may sell for profit and freely distribute scripts and/or compiled scripts that were created with the SOFTWARE PRODUCT.*"
Abgesehen davon, dass es selbstverständlich bei "Fremdhilfe" wie anderen Funktionen, zum Beispiel von Foren-Usern das Einverständnis zur Veröffentlichung und Vermarktung braucht – und demnach ohne Einverständnis der Code zu 100% selbst entwickelt sein muss, oder dass das Risiko auf eine gecrackte Version des Programms vorhanden ist, kann man mit AutoIt-Skripts und kompilierten AutoIt-Programmen machen, wass man will.

2. Installation

Im folgenden Kapitel wird ausführlich auf die Installation von AutoIt eingegangen.

2.1 AutoIt-Kern installieren

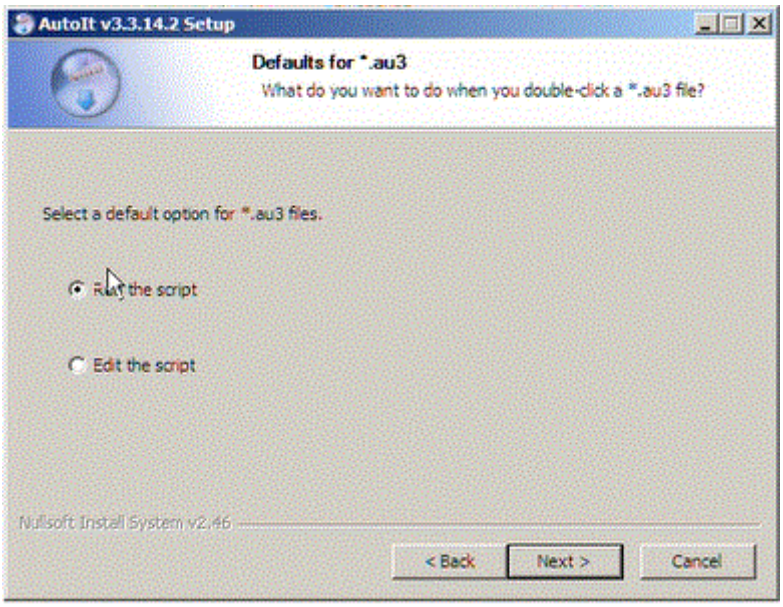
Zunächst muss AutoIt v3 mitsamt Compiler und allem Drumherum von http://www.autoitscript.com/ oder http://www.autoit.de heruntergeladen werden. Wurde der Installer (normalerweise *autoit-v3-setup.exe*) heruntergeladen, muss er ausgeführt werden.
Wurde den Lizenzbedingungen zugestimmt, wird fortgefahren:



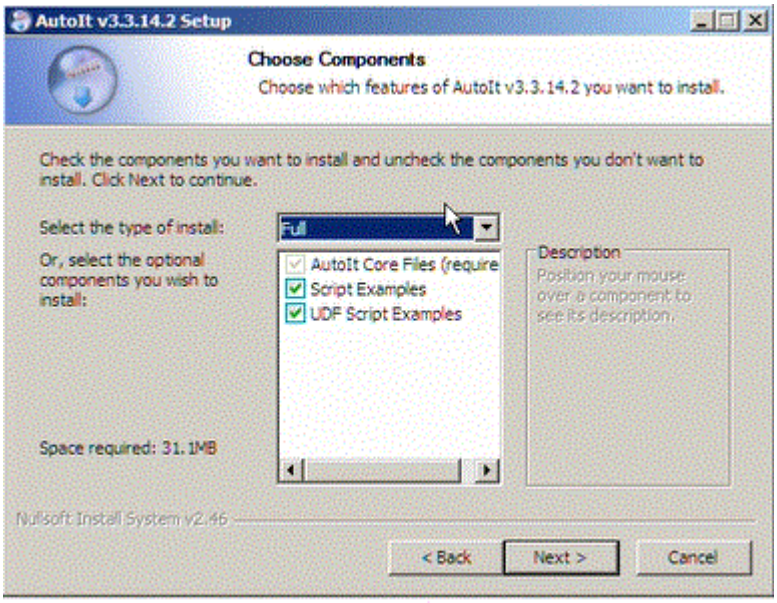
Sollte diese Meldung erscheinen, sollte – wie hingewiesen – kompatiblitätstechnisch lieber die x86-Version verwendet werden.

Moment! Aber wenn ich 64-bit-Anwendungen erstellen will, muss ich ja doppelt installieren, oder?

Nein. Wenn die x64-Option gewählt wird, werden nur die x64-Tools installiert. Wenn nicht, werden alle installiert.



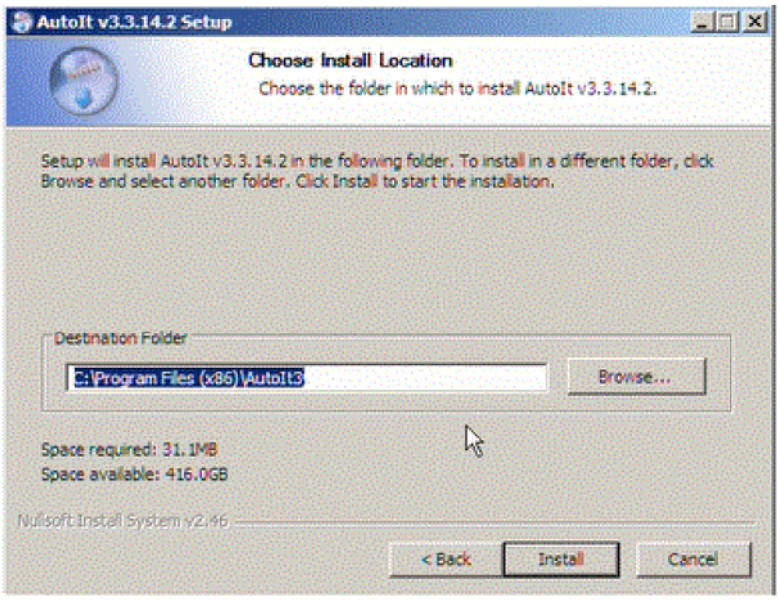
"Run the script" – Diese Option ist denkbar sinnvoller, als das Skript zu editieren.



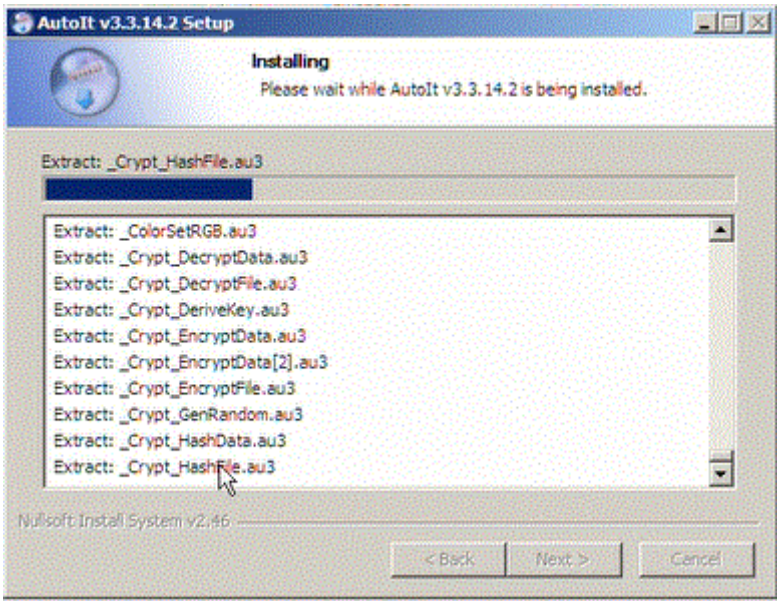
Alle Features aktiviert lassen.

Moment, nur mal so eine Zwischenfrage. "AutoIt Core Files" ist ja klar. Script Examples auch noch. Aber was ist eine UDF? Ist das nicht ein Dateisystem für CDs oder DVDs? Was haben die denn jetzt mit AutoIt zu tun?

An sich gar nichts. UDF steht – um es mal vorwegzunehmen – für User Defined Functions. Also Funktionen, die von anderen Benutzern entwickelt wurden, und hier in AutoIt integriert wurden. Aber dazu werden wir später noch kommen.



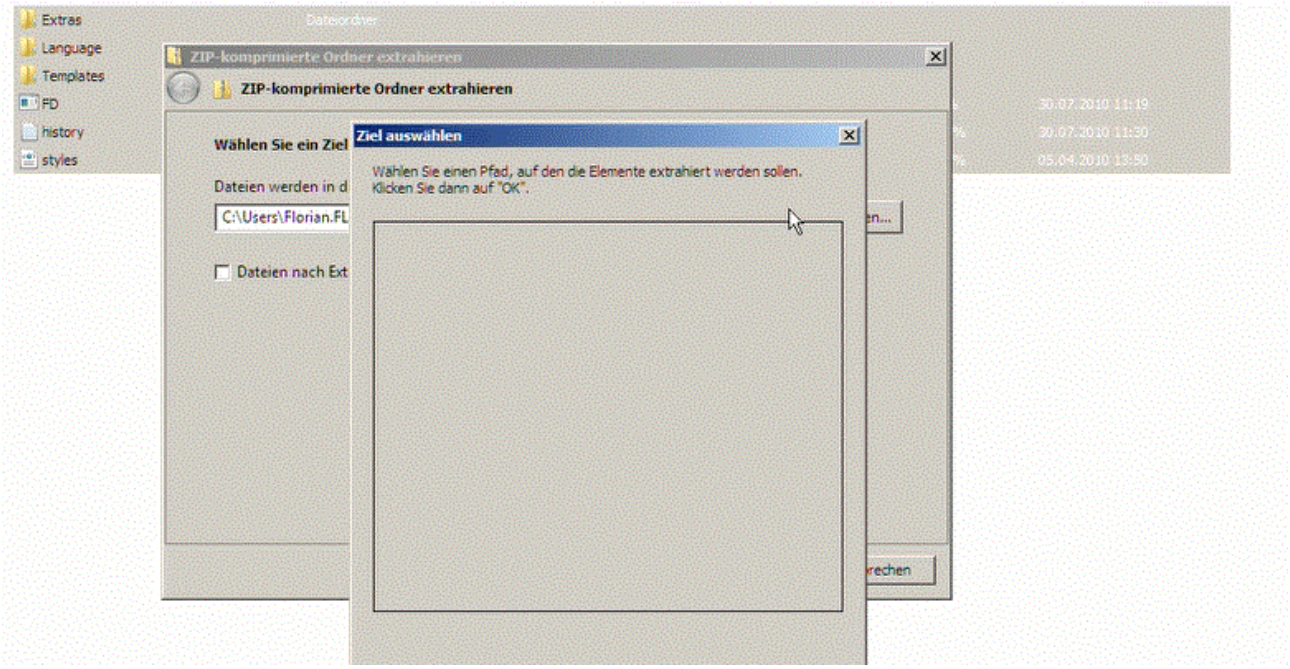
Es wäre sinnvoller, z.B. im Benutzerverzeichnis einen Ordner "AutoIt Programme" zu erstellen, und im Unterordner "Tools" AutoIt zu installieren. Vom nicht zwingend beschreibbaren Programmverzeichnis würde ich persönlich abraten.



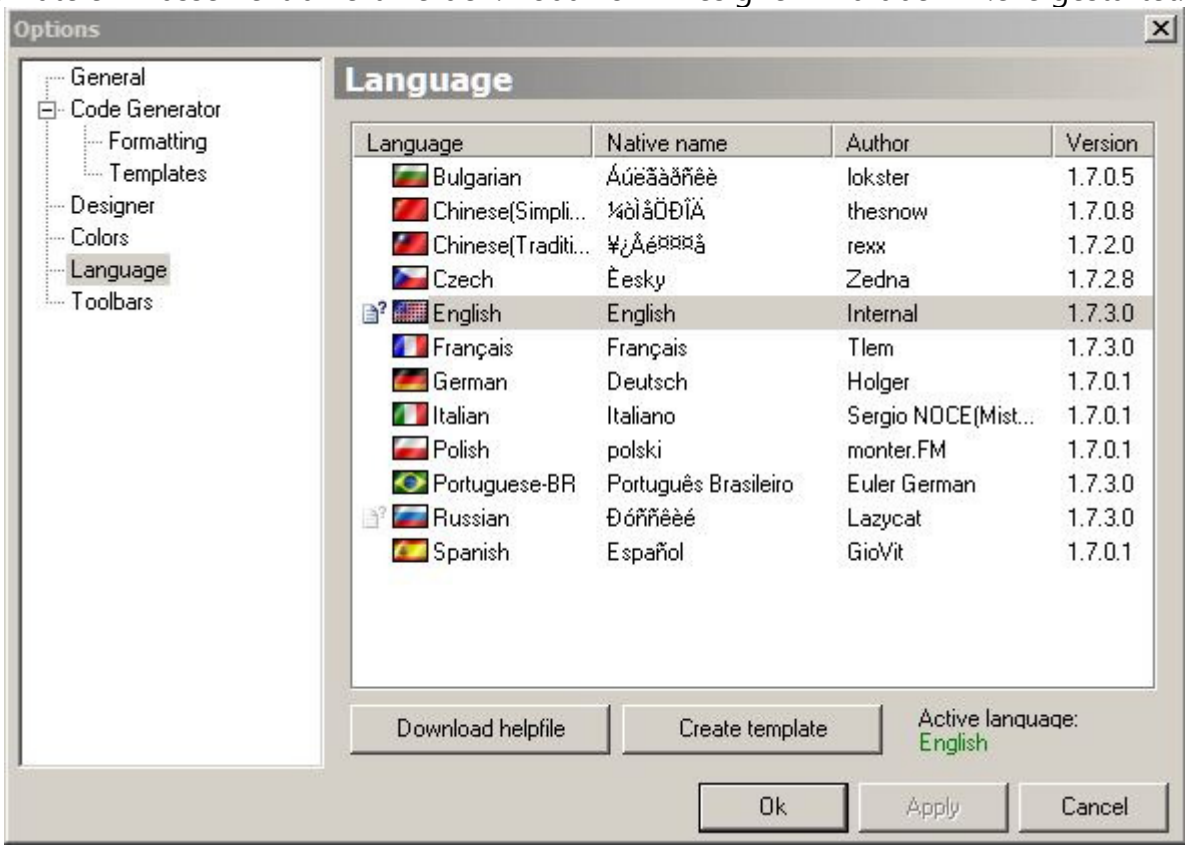
Und brav entpackt der AutoIt-Installer alle Dateien – in diesem Fall die Beispieldateien für die "_Crypt"-UDF-Sektion.

2.2 Koda Form Designer installieren

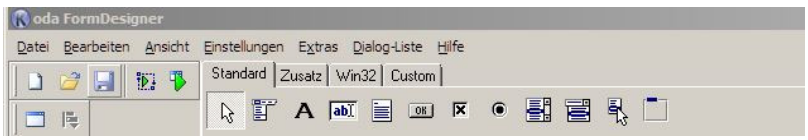
Zunächst müssen aus dem ZIP-Archiv, herunterzuladen von http://koda.darkhost.ru/dl.php?file=koda_1.7.3.0.zip (alternative Quelle: HEISE Download), alle Dateien extrahiert werden (sinnvollerweise in den Ordner, wo auch AutoIt installiert wurde.



Alle Dateien müssen extrahiert werden. Koda Form Designer wird über FD.exe gestartet.



Dort kann schließlich auf die Sprache "German / Deutsch" umgestellt werden...



... was schließlich bewirkt, dass die Benutzeroberfläche nun auf Deutsch ist.

Irgendwas scheint hier nicht funktioniert zu haben. Im "Objekt Inspektor" tauchen überall noch unzählige englische Namen auf. Fehlt etwa eine Sprachbibliothek?

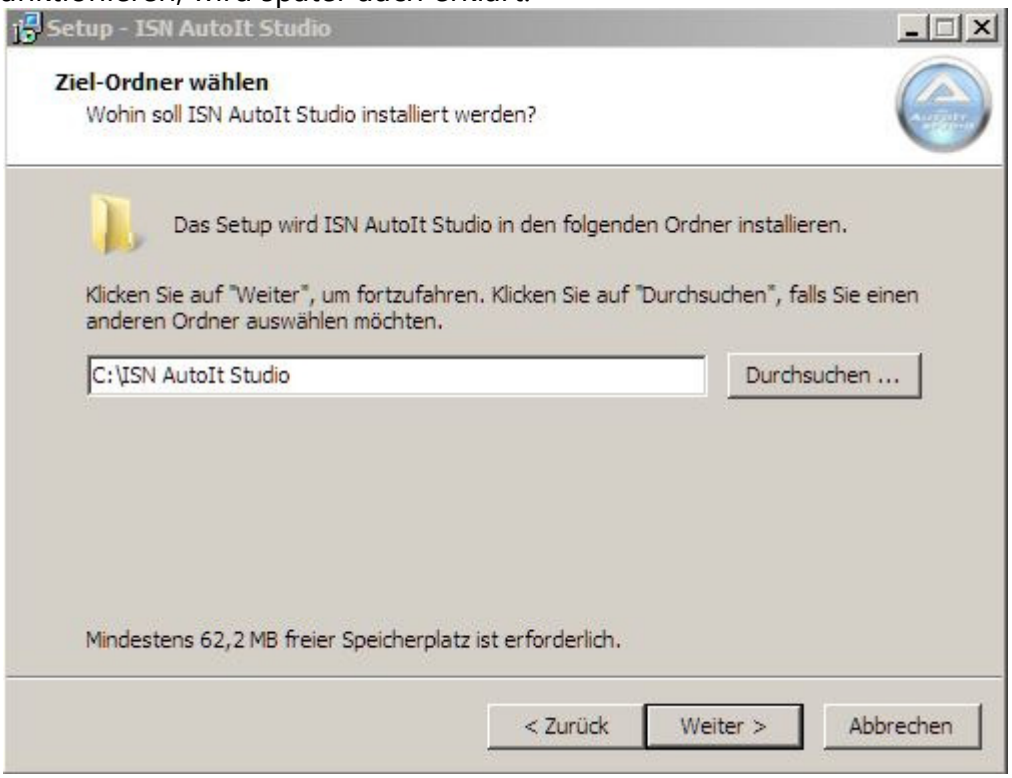
Nein. Das sind AutoIt-interne Bezeichnungen, die nicht übersetzt werden.

Somit ist auch der FormDesigner – der teilweise etwas besser ist als das im ISN AutoIt Studio integrierte Form Studio – uneingeschränkt lauffähig.

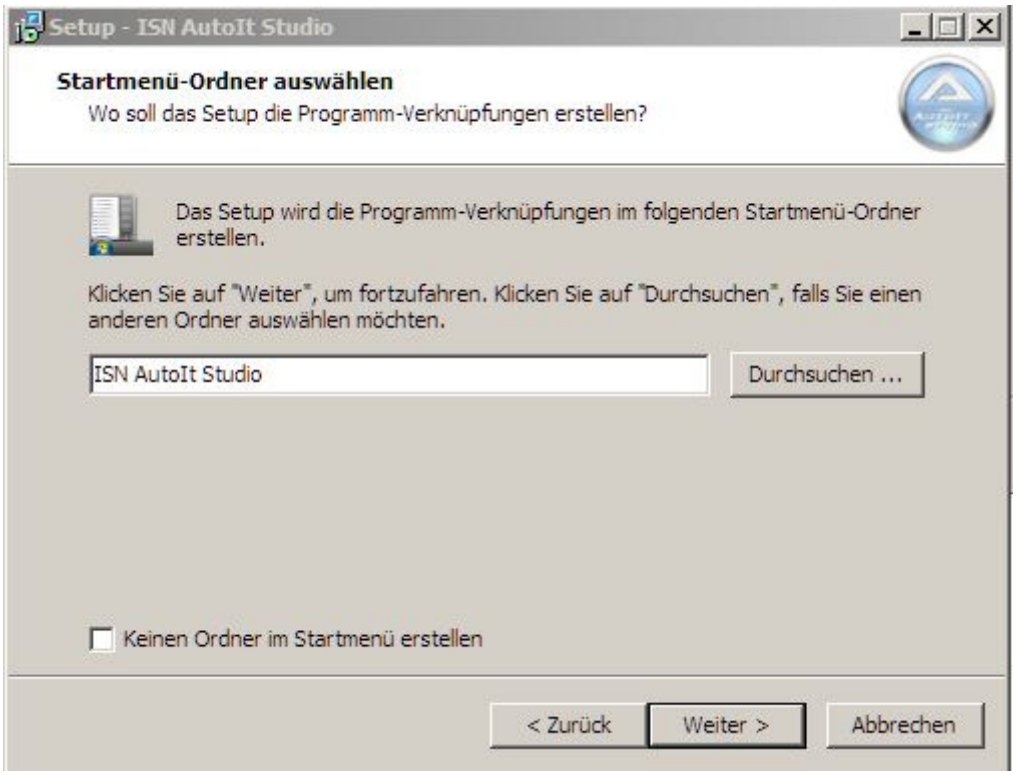
2.3 ISN AutoIt Studio installieren und konfigurieren

Der offizielle Download für das ISN AutoIt Studio ist auf der Seite https://www.isnetwork.at/ zu finden (Programme > ISN AutoIt Studio > Downloads). Zu empfehlen ist der Installer, auf den sich das Tutorial auch bezieht.

In diesem Buch wird mit der zurzeit aktuellen Version 1.05 Stable gearbeitet. Für eine uneingeschränkte Lauffähigkeit – wenn auch etwas "out-of-date" – wäre es naheliegend, wenn (falls zu diesem Zeitpunkt verfügbar) diese Version heruntergeladen wird. Wie Updates funktionieren, wird später auch erklärt.



Installation in den vorgeschlagenen Pfad ist sicher nicht von Nachteil, auch wenn es in den AutoIt-Entwicklungsordner installiert werden könnte.

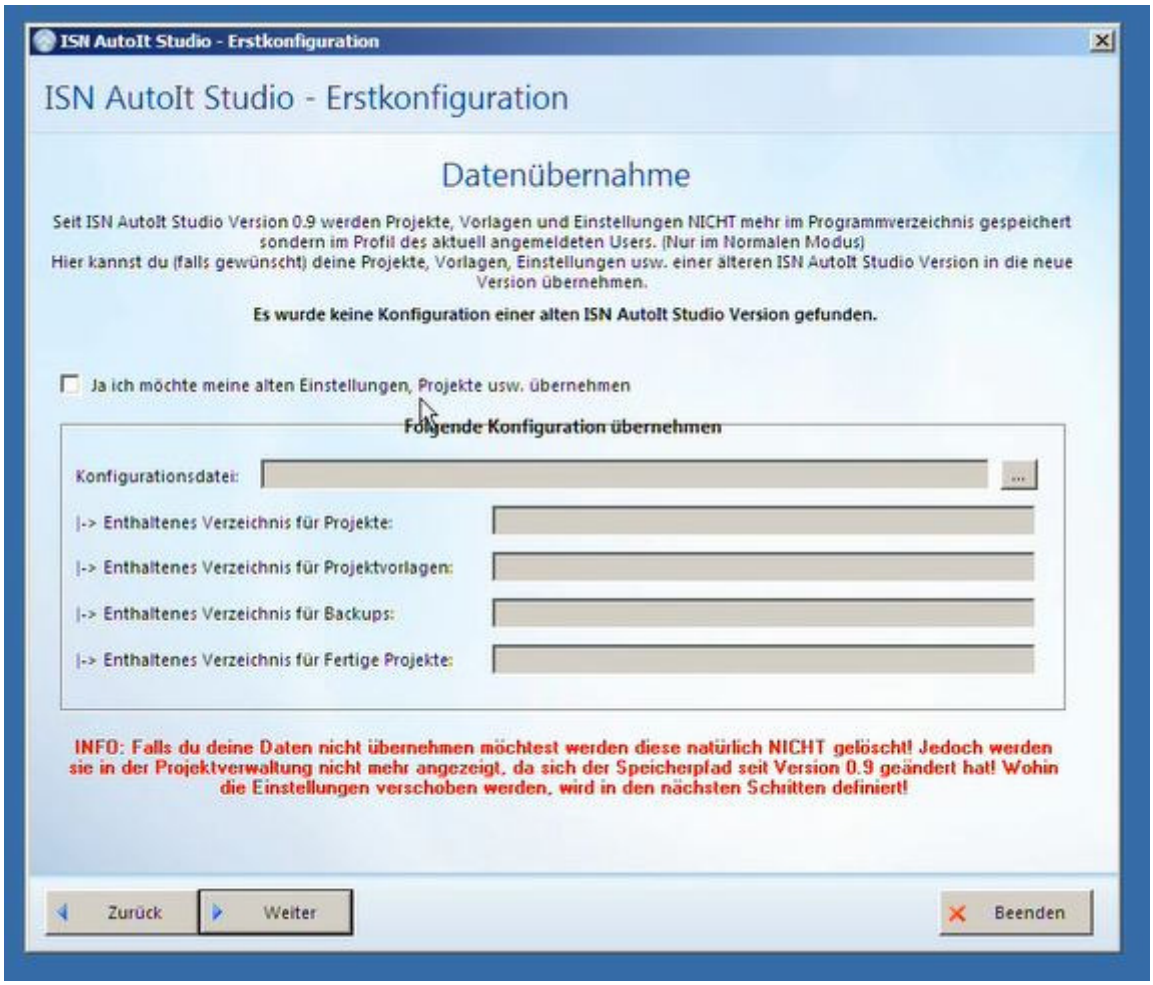


Ob Startmenü oder nicht, das spielt keine Rolle. Empfehlung an dieser Stelle: es bietet sich an, zum FormDesigner und zum ISN AutoIt Studio eine Taskleistenverknüpfung zu erstellen. (Wer nicht weiß, wie das geht, sollte sich lieber erstmal ein Buch zur Windows-Bedienung gönnen. ☺)

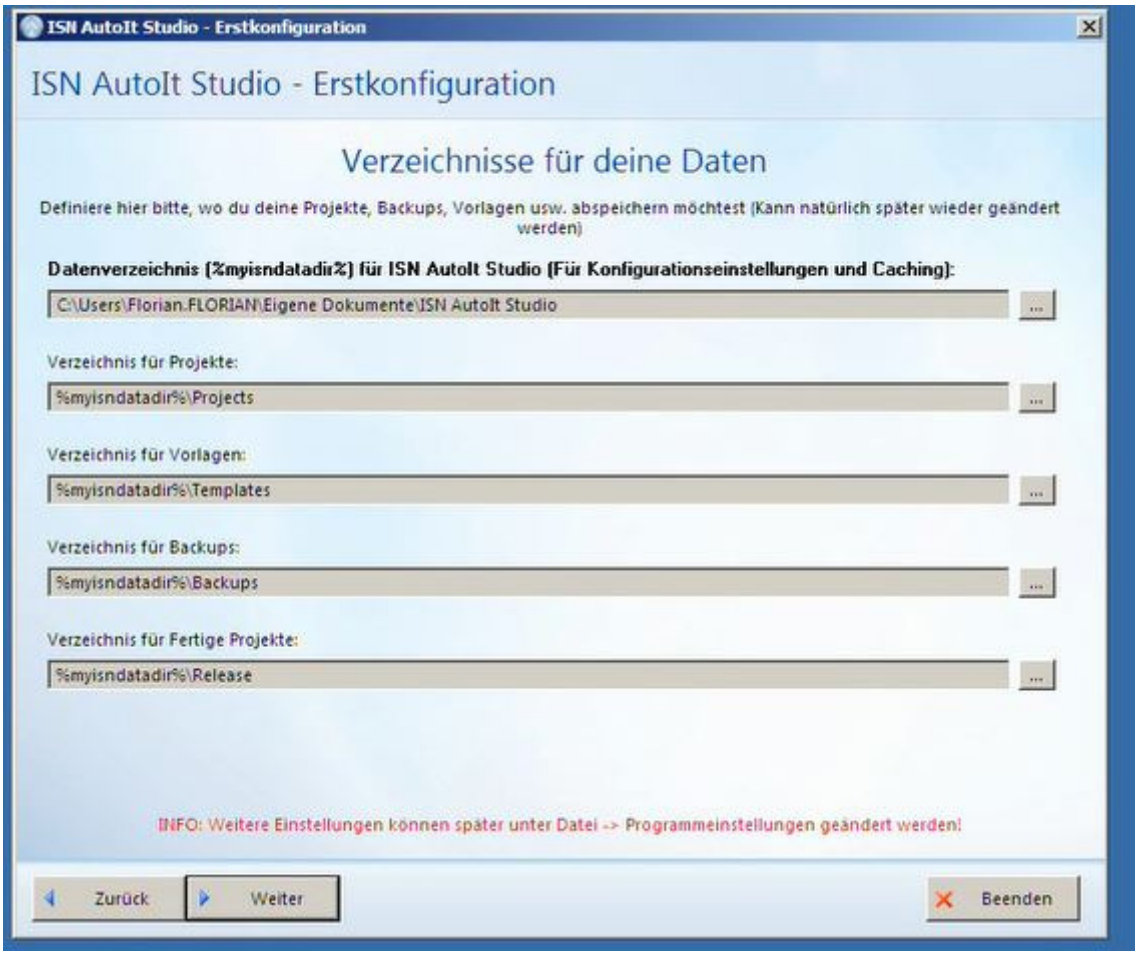
Nach der Installation sollte das Häkchen zum Programmstart gesetzt werden. Nach dem Programmstart sollte sich folgendes Fenster zeigen:

Auf jeden Fall unsere Sprache auswählen!

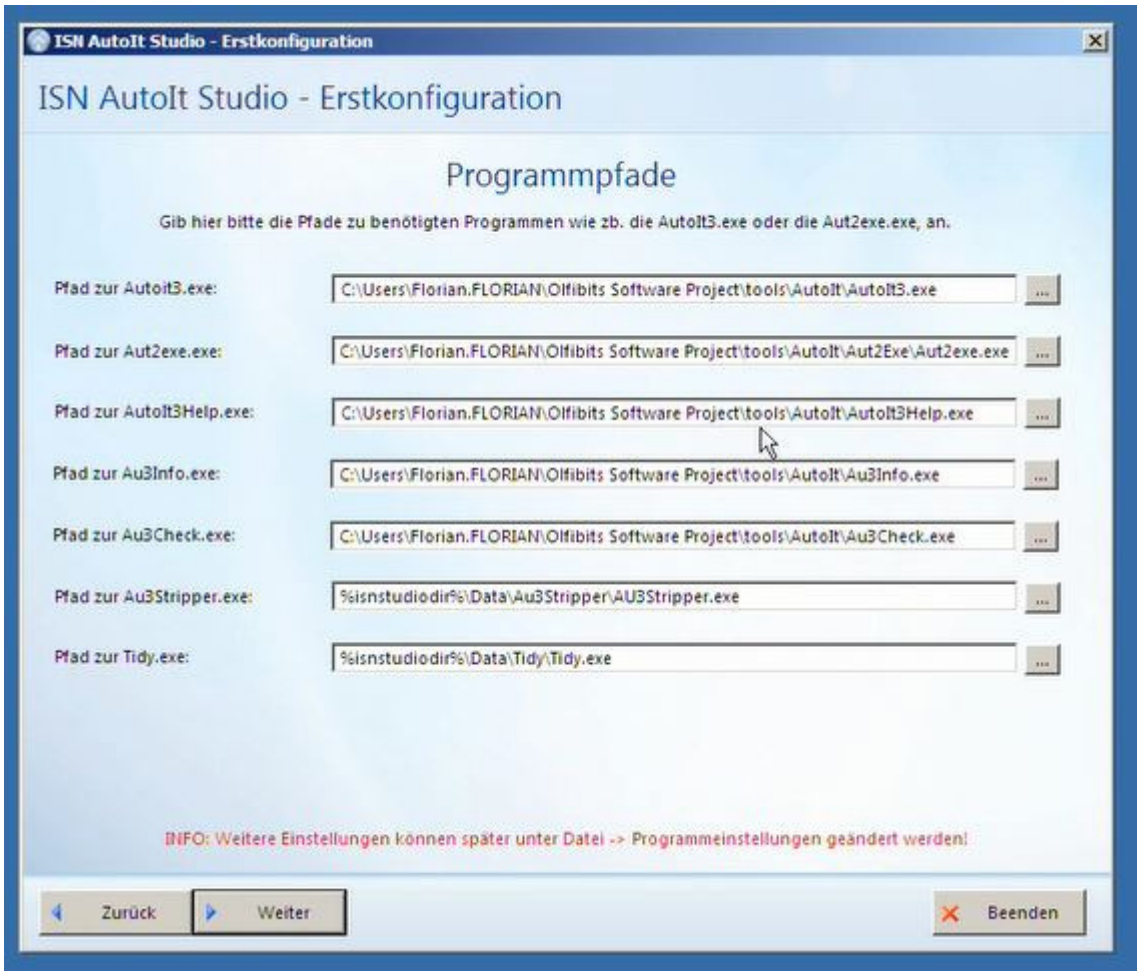
Normaler Modus macht für eine dauerhafte Verwendung nur auf dem PC mehr Sinn.



Datenübernahme wird nicht gebraucht, da (normalerweise) keine ältere Version vorhanden ist.



Es macht Sinn, diese Pfade beizubehalten.



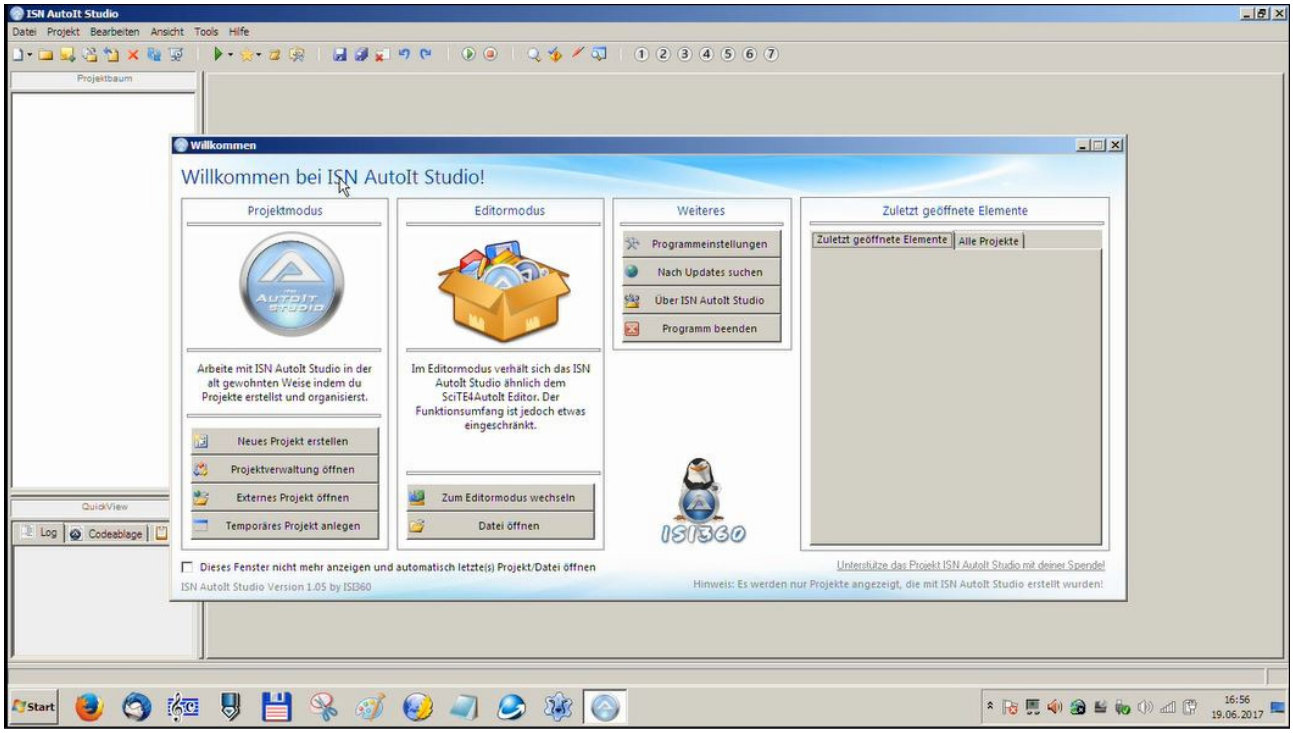
Abgesehen von "AU3Stripper" oder "Tidy" muss man in der Regel alle betreffenden AutoIt-Programme eingestellt werden – daher macht es auch sinn, dass der AutoIt-Kern schon vorhin installiert wurde.

Warum verknüpfen wir die *.au3-Dateien nicht einfach mit dem ISN AutoIt Studio? Das wäre doch für einzelne Dateien viel einfacher!

Eine berechtigte Frage. Aber auf längere Sicht macht es mehr Sinn, mit Doppelklick die Datei direkt zu öffnen. Das Kontextmenü-Item kann man getrost setzen.

Dieses Fenster zeigt, dass die Initialisierung läuft. Das dauert normalerweise nie über eine Minute...

Toll, hat funktioniert.



Und das ist das Startfenster des ISN AutoIt Studio v1.05. (Anmerkung: sie wurde bis auf ein paar Bibliotheken, z.B. Syntax-Highlighting, komplett in AutoIt geschrieben.)

3. Erste Codestrukturen in AutoIt

Bevor wir in die Entwicklung einsteigen, geht es jetzt zunächst nur mal an trockene Theorie. So langweilig, wie es aufs Erste scheint, ist es vielleicht doch nicht – außerdem hilft es zweifellos, um zu verstehen, wie ein AutoIt-Programm aufgebaut ist und welche Funktionen es gibt. Die Auflistung beschränkt sich auf die **gebräuchlichsten** Direktiven, Schlüsselwörter und Makros.

Grundlegende Syntax-Struktur in AutoIt:

(1) Anweisungen werden nicht mit einem Semikolon abgeschlossen.
(2) Semikola leiten einen Kommentar ein – daher wäre es auch kein Fehler bei (1)
(3) Variablennamen beginnen immer mit einem Dollarzeichen.
(4) Strings stehen in Anführungszeichen, Variablen können nicht direkt in Strings eingebunden werden.
(5) Zuweisung erfolgt über den Ist-Gleich-Operator **=**
(6) Stringverkettung erfolgt über den Und-Operator **&**
(7) Datentypen können umgewandelt werden, müssen aber nicht explizit deklariert werden.
(8) Mathematische Operatoren:
    1. + (Plus)
    2. - (Minus)
    3. / (Teilen)
    4. * (Malnehmen)
    5. % (Modulo)
(9) Vergleichs-Operatoren:
    1. = (gleich)
    2. < (kleiner)
    3. > (größer)
    4. <> (ungleich)

Halt. Ein Ist-Gleich-Operator als Vergleichsoperator und einmal als Zuweisungsoperator? Ernsthaft?

Ja, das ist in AutoIt (leider) so. Aber es besteht im Prinzip keine Verwechslungsgefahr, weil der Vergleichsoperator nur in If-Anweisungen gebraucht wird.

Schlüsselwörter

And	Steht in einer Bedingung, bei der mehrere definierte Bedingungen erfüllt sein müssen.	`If $var1 = "Ja" And $var2 = "Ja" Then`
		`; …`
		`EndIf`

ByRef	Stellt eine Referenz auf ein Objekt/Array. Verwendung häufig in Funktionen als Funktionsparameter.	`Func _Example1(ByRef $array)`
		`; …`
		`EndFunc`

Case	Verwendet in Select/Switch-Anweisungen – leitet einen konkreten Fall für einen Variablenwert ein.	`Switch GUIGetMsg()`
		`  Case $GUI_EVENT_CLOSE`
		`  Exit`
		`EndSwitch`

Const	Konstante: feststehender Wert, wird versucht, sie später zu überschreiben, wird ein Fehler erzeugt.	`Const $pi = 3.141592653589793`

ContinueCase	Verwendet in Select/Switch-Anweisungen – normalerweise wird auf die anderen Fälle, wenn einer erfüllt ist, keine Rücksicht genommen. Das wird hier verhindert.	`Switch GUIGetMsg()`
		`  Case $GUI_EVENT_CLOSE`
		`  MsgBox(0, "Info", "Schließen")`
		`  ContinueCase`
		`  Case -3 ; = $GUI_EVENT_CLOSE`
		`  MsgBox(0, "Info", "Und tschüss!")`
		`  Exit`
		`EndSwitch`

ContinueLoop	Definitiver Aufruf, eine Schleife (egal welcher Art) fortzuführen.	`For $i = 0 To 25 Step +1`
		`  If $i = 20 Then`
		`  ContinueLoop`
		`  EndIf`
		`Next`

Default	Default-Wert bei Funktionsaufruf.	`MsgBox(0, "Titel", "Text", Default, WinGetHandle("AutoIt.pdf", ""))`
Dim	Definiert ein Array	`Dim $aArray[10]`

Do	Einleitung zu einer Do...While-Schleife	`$i = 0`
		`Do`
		`  MsgBox(0, "Zahl", $i)`
		`  $i = $i + 1`
		`Until $i = 10`

Else	Einleitung zu anderem unbestimmten Fall in If-Anweisung	`If $i < 10 Then`
		`  ; …`
		`Else`
		`  ; …`
		`EndIf`

ElseIf	Einleitung zu anderem bestimmten Fall in If-	`If $i < 10 Then` `  ; …`

	Anweisung	`ElseIf $i > 10 Then`
		`  ; …`
		`Else`
		`  ; …`
		`EndIf`

EndFunc	Ende einer Funktion	`Func _Example2()`
		`  ; …`
		`EndFunc`

EndIf	Ende einer If-Anweisung	`If $i = 10 Then`
		`  ; …`
		`EndIf`

EndSelect	Ende einer Select-Anweisung	`Select`
		`  Case $i = 0`
		`  ; …`
		`  Case $c = 0`
		`  ; …`
		`  Case Else`
		`  ; … nichts davon`
		`EndSelect`

EndSwitch	Ende einer Switch-Anweisung	`Switch GUIGetMsg()`
		`  Case $GUI_EVENT_CLOSE`
		`  Exit`
		`  Case $aExitButton`
		`  Exit`
		`EndSwitch`

Enum	Enumeration für Variablennamen	`Enum $spalte1, $spalte2, $spalte3`
Exit	Beendet das Programm	`Exit`

ExitLoop	Steigt aus einer Schleife aus.	`$guiMsg = GUIGetMsg()`
		`If $guiMsg = $GUI_EVENT_CLOSE Then`
		`ExitLoop`
		`EndIf`

False	Konstante False (0)	`If False <> 0 Then ; <> ist UNGLEICH`
		`  MsgBox(0, "Ups", "AutoIt ist dumm!")`
		`EndIf`

For	Bedingte Zähleranweisung	`For $i = 25 To 50`
		`  MsgBox(0, "Zahl", $i)`
		`Next`

Func	Benutzerdefinierte Funktion	`Func Pi($param1, $param2 = "Pi")`
		`  Return $param2 & "=" &`
		`3.141592653589793`
		`EndFunc`

Global	Globale Variable (überall Zugriff)	`Global Const $pi = 3.141592653589793`

If	Leitet Bedingungsanweisung ein.	`Global Const $pi = 3.141592653589793`
		`If $pi <> 3.141592653589793 Then`
		`  ; ..`
		`EndIf`

In	Verwendung in For-Schleife mit enumerierten Arrays	`$i = 0`
		`For $element In $array`
		`  MsgBox(0, "$array[" & $i & "]",`
		`$element)`
		`$i = $i + 1`

		`Next`

Local	Lokale Variable (z.B. nur Zugriff in einer Funktion, nicht extern abrufbar)	`Func _Funktion()`
		`  $pi = 3.141592653589793`
		`EndFunc`
		`MsgBox(0, "Pi", $pi) ; Leerer Text`

Next	Ende einer For-Anweisung	`For $i = 0 To 20`
		`  $i = $i + 1 ; Doppelschritte`
		`Next`

Not	Kehrt True oder False um.	`If Not True Then`
		`  ; False`
		`EndIf`

Or	Steht in einer Bedingung, bei der einer von verschiedenen definierten Bedingungen erfüllt sein muss.	`If $i = 0 Or $c = 0 Then`
		`  MsgBox(0, "Einer wars!", "i oder c ist 0!")`
		`EndIf`

ReDim	Überschreibt Arrays.	`ReDim $aArray[100]; vorhin waren's 10`

Return	Definiert Rückgabewert einer Funktion → führt zur Beendung des Funktionscodes.	`Func _Example($param)`
		`  Return "Parameter: " & $param`
		`EndFunc`

Select	Analog zu Switch, nur erweiterte Auswahl (nicht nur auf eine Variable beschränkt)	`Select`
		`  Case $i = 0`
		`  ; …`
		`  Case $c = 0`
		`  ; …`
		`  Case Else`
		`  ; … nichts davon`
		`EndSelect`

Static	Definiert lokale Variable in Funktion, wird danach vernichtet.	`Func _Static()`
		`  Static $blabla = "jaja"`
		`  MsgBox(0, "Variable", $blabla)`
		`EndFunc`
		`MsgBox(0, "Variable", $blabla)`
		`; Text wird leer sein! :-)`

Step	Definiert den Schritt in einer Zähleranweisung.	`For $i = 25 To 0 Step -1`
		`  MsgBox(0, "Zahl", $i)`
		`Next`

Switch	Fallunterscheidung einer Variablen.	`Switch GUIGetMsg()`
		`  Case $GUI_EVENT_CLOSE`
		`  Exit`
		`  Case $aExitButton`
		`  Exit`
		`EndSwitch`

Then	Verknüpfung If...Then	`If True Then`
		`  ; bereits bekannt`
		`EndIf`

To	Schlüsselwort in bedingter Zähleranweisung	`For $i = 25 To 0 Step -1`
		`  MsgBox(0, "Zahl", $i)`
		`Next`

True	Konstante True (1)	`If True <> 1 Then ; <> ist UNGLEICH`
		`  MsgBox(0, "Ups", "AutoIt ist dumm!")`
		`EndIf`

Until	Schlüsselwort Until (Do...Until)	`$i = 0`
		`Do`
		`  MsgBox(0, "Zahl", $i)`
		`  $i = $i + 1`
		`Until $i = 10`
		`; nicht neues`

WEnd	Ende einer While-Schleife	`$i = 0`
		`While $i < 10`
		`  MsgBox(0, "Status", "i ist noch unter 10")`
		`$i = $i+1`
		`WEnd`

While	Eine bedingte Schleife.	

With	In dieser Struktur kann man ein Objekt wählen, das verkürzt aufgerufen wird.	`; aus der AutoIt-Hilfe`
		`Local $oExcel = ObjCreate("Excel.Application")`
		`$oExcel.visible = 1`
		`$oExcel.workbooks.add`
		
		`With $oExcel.activesheet`
		`  .cells(2, 2).value = 1`
		`  .range("A1:B2").clear`
		`EndWith`
		
		`$oExcel.quit`

EndWith	Ende einer With-Struktur	

Präprozessor-Direktiven

#ce / #comments-end	Kommentar Ende	`---`
#cs / #comments-start	Kommentar Start	`---`

#include	Bindet .au3-Datei ein.	`;Globale Bibliothek`
		`#include <GDIPlus.au3>`
		
		`;im aktuellen Verzeichnis`
		`#include "Funktionen.au3"`

#include-once	Schlüsselwort, dass .au3-Datei nur einmal eingebunden werden darf (Definition in externer Funktionsbibliothek)	`---`
#NoTrayIcon	Kein Tray-Icon in der Taskleiste.	`#NoTrayIcon`
#OnAutoItStartRegister	Dahinter Funktionsname zu nennen, dessen Funktion beim AutoIt-Programmstart aufgerufen wird.	`#OnAutoItStartRegister "TimerStart"`
#RequireAdmin	Fordert Administratorrechte zum Starten des Programms.	`#RequireAdmin`
#EndRegion	Ende einer Region	`; aus Koda`

#Region	Zur Gliederung des Codes – auslesbar von anderen Programmen (z.B. KODA zur Rekonstruierung der GUI aus AutoIt3 Skript)	`#Region ### START Koda GUI section ### Form= $Form1 = GUICreate("Form1", 623, 449, 192, 114) GUISetState(@SW_SHOW) #EndRegion ### END Koda GUI section ###`

Vordefinierte Makros

@AppDataCommonDir	Gemeinsames Anwendungsdatenverzeichnis
@AppDataDir	Anwendungsdatenverzeichnis des aktuellen Benutzers
@AutoItExe	Kompiliert: Pfad der kompilierten Anwendung Skript: Pfad des AutoIt3-Interpreters
@AutoItPID	PID der laufenden AutoIt-Anwendung (s. @AutoItExe)
@AutoItVersion	AutoIt-Version
@AutoItX64	Liefert 1, wenn die Anwendung als 64-bit-Anwendung ausgeführt wird.
@CommonFilesDir	Gemeinsame Dokumente
@Compiled	1 bei *.exe / *.a3x 0 bei *.au3
@ComputerName	Name des PCs
@ComSpec	Pfad der Kommandozeile
@CPUArch	Prozessorarchitektur: X32 (=X86) oder X64
@CR	Wagenrücklauf
@CRLF	Zeilenumbruch
@DesktopCommonDir	Gemeinsames Desktop-Verzeichnis
@DesktopDepth	Desktop-Tiefe
@DesktopDir	Desktop-Verzeichnis
@DesktopHeight	Desktop-Höhe
@DesktopRefresh	Desktop-Aktualisierungsrate
@DesktopWidth	Desktop-Breite
@DocumentsCommonDir	Gemeinsames Dokumentenverzeichnis
@error	Letzter Fehlercode
@exitCode	Exit-Code des Programms

@exitMethod	Exit-Methode des Programms
@extended	Letzter (erweiterter) Fehlercode
@FavoritesCommonDir	Gemeinsames Favoritenverzeichnis
@FavoritesDir	Favoritenverzeichnis
@HOUR	Stunden als Zahl
@IPAddress1	IP-Adresse
@KBLayout	Keyboard-Darstellung
@LF	Zeilenvorschub
@LogonDNSDomain	Anmelde-DNS-Domain
@LogonDomain	Anmelde-Domain
@LogonServer	Anmelde-Server
@MDAY	Tag im Monat als Zahl
@MIN	Minuten als Zahl
@MON	Der aktuelle Monat als Zahl
@MSEC	Millisekunden als Zahl
@MyDocumentsDir	Mein Dokumentenverzeichnis
@NumParams	Nummer der Parameter, mit der das Programm aufgerufen wurde
@OSArch	Systemarchitektur
@OSBuild	Systemerstellungsnummer
@OSLang	Systemsprache
@OSServicePack	Servicepack des Systems
@OSType	Typ des Systems
@OSVersion	Version des Systems
@ProgramFilesDir	Programmverzeichnis
@ProgramsCommonDir	Gemeinsames Programmverzeichnis
@ProgramsDir	Programmverzeichnis
@ScriptDir	Verzeichnis des Skripts / der Anwendung
@ScriptFullPath	Voller Pfad des Skripts / der Anwendung
@ScriptLineNumber	Aktuelle Zeile (**nur im Skript)**
@SEC	Sekunden als Zahl
@StartMenuCommonDir	Gemeinsames Startmenüverzeichnis

@StartMenuDir	Startmenüverzeichnis
@StartupCommonDir	Gemeinsames AutoStart-Verzeichnis
@StartupDir	AutoStart-Verzeichnis
@SW_DISABLE	Fenster "einfrieren"
@SW_ENABLE	Fenster "entfrieren"
@SW_HIDE	Fenster verstecken
@SW_MAXIMIZE	Fenster maximieren
@SW_MINIMIZE	Fenster minimieren
@SW_RESTORE	Fenster wiederherstellen
@SW_SHOW	Fenster anzeigen
@SW_SHOWDEFAULT	Fenster standardmäßig anzeigen
@SW_SHOWMAXIMIZED	Fenster maximiert anzeigen
@SW_SHOWMINIMIZED	Fenster minimiert anzeigen
@SystemDir	Systemverzeichnis
@TAB	Tab
@TempDir	Temporäres Verzeichnis des Benutzers
@UserName	Benutzername des aktuellen Benutzers
@UserProfileDir	Verzeichnispfad des aktuellen Benutzers
@WDAY	Tag in der Woche als Zahl
@WindowsDir	Windows-Verzeichnis
@WorkingDir	Aktuelles Arbeitsverzeichnis
@YDAY	Tag im Jahr als Zahl
@YEAR	Jahr als Zahl

Bei @Compiled ist die Rede von *.a3x. Was ist dass jetzt schon wieder?

Nichts anderes als ein verschlüsseltes AutoIt-Skript, das nur von AutoIt selbst geöffnet werden kann. So kann man Skripte verteilen, ohne den Quellcode zu zeigen, aber auch ohne *.exe-Anwendungen zu verteilen.

Somit wurden schon einmal die wichtigsten Direktiven und Kontrollstrukturen erläutert. Zu Funktionen werde ich wie oben so explizit nicht eingehen können, da das den Rahmen des Buches eindeutig sprengen würde. Hierbei kann man sich an der AutoIt-Hilfedatei orientieren.

Trotzdem kurz zu den Funktionen und ihren Unterschieden:

- Standard-Funktionen: Sie haben "ganz normale Namen", sind bei AutoIt immer dabei und man kann sich auf sie verlassen.
- UDFs: UDFs – User Defined Functions – haben einen Unterstrich als Kennzeichnung.
- Standard-UDFs: UDFs, die standardmäßig in AutoIt eingebunden sind. Freie Benutzung für alle Zwecke erlaubt.
- Nicht-Standard-UDFs: UDFs, die jeder normale Mensch in einem Forum entwickeln kann. Sie sind **je nach Erlaubnis / Genehmigung** des betreffenden Autors zur Benutzung freigegeben.

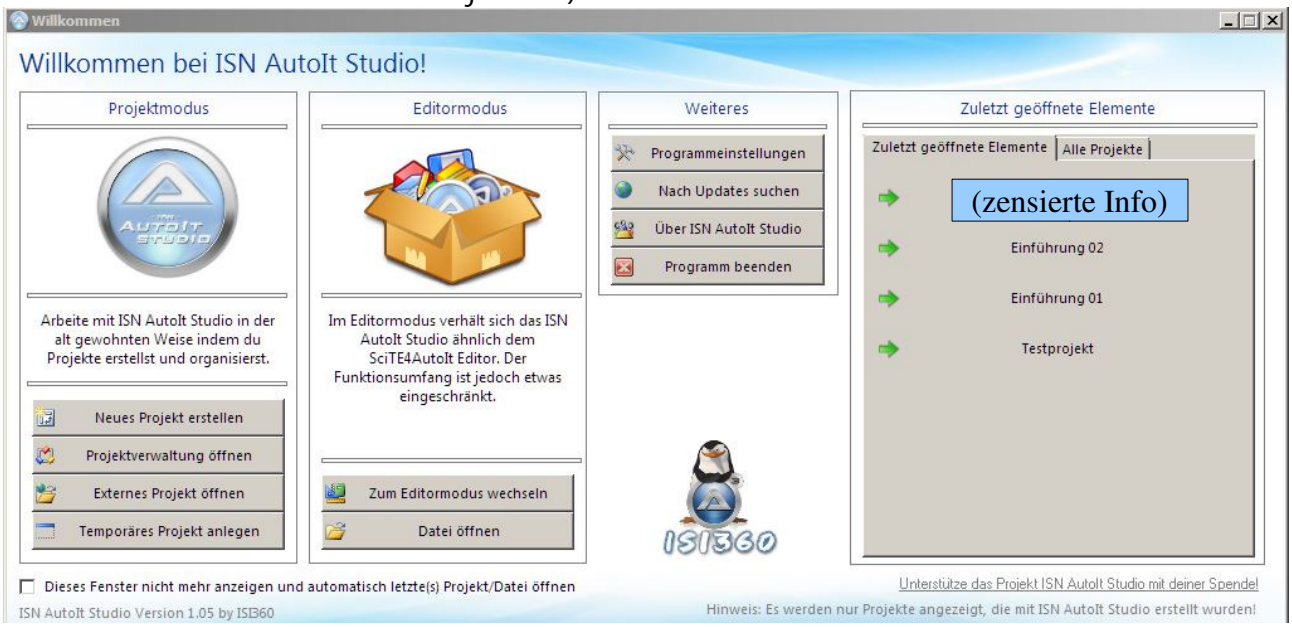
Wenn ich also im Forum frage, wie man eine Funktion erstellt, die einen benutzerdefiniert formatierten Zeitstempel ausgibt – ist das schon eine Nicht-Standard-UDF?

Streng genommen ja. Ein anderer User hat normalerweise das Urheberrecht an dem Skript. D.h. Wenn du es veröffentlichen willst, brauchst du die Zustimmung dafür.

4. Die Entwicklungsumgebung ISN AutoIt Studio kennenlernen

4.1 Erste Schritte

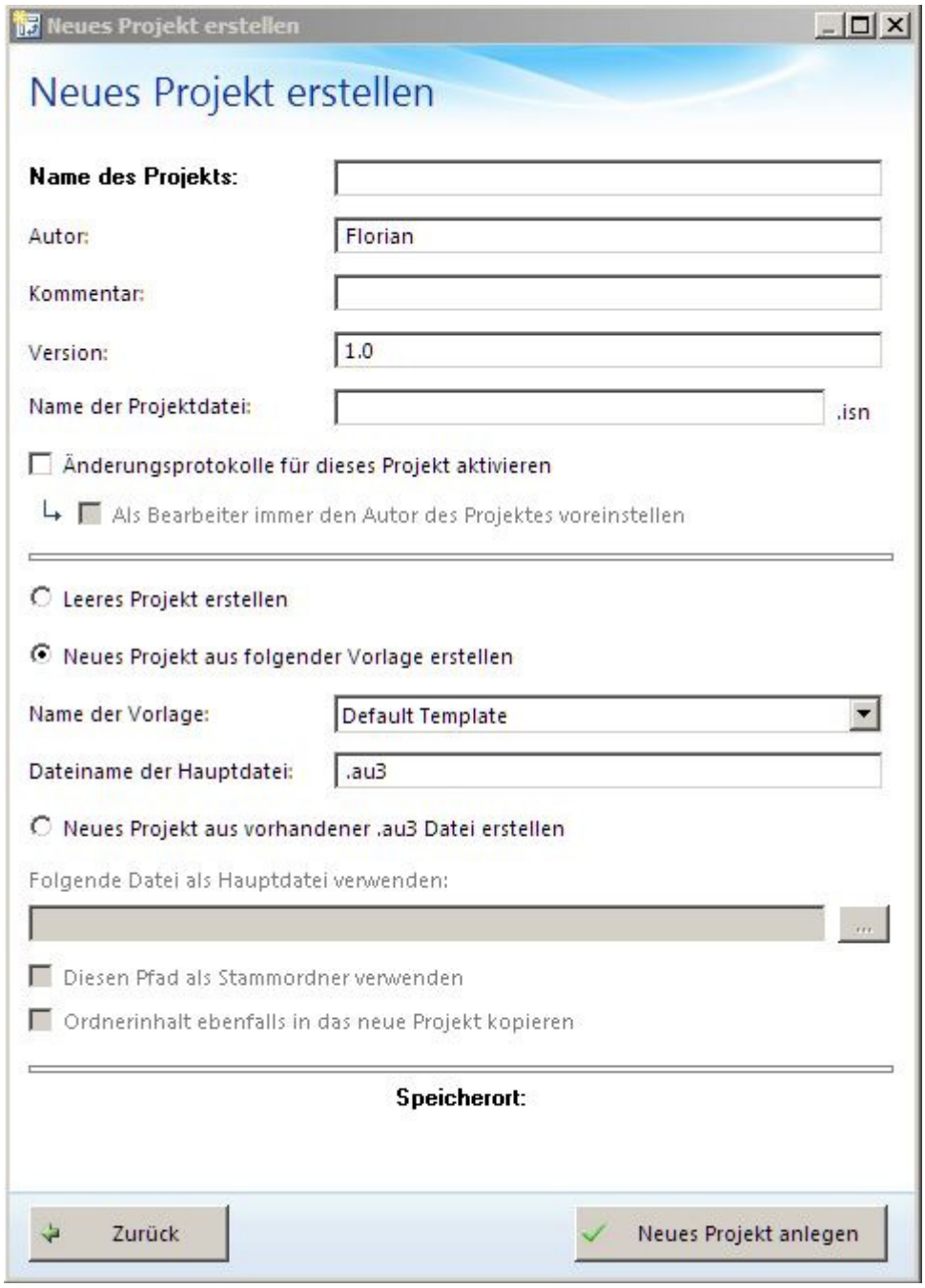
ISN AutoIt Studio sollte schon fertig installiert und betriebsbereit sein.
Nach dem zweiten Start von AutoIt Studio sollte folgendes Fenster auftauchen (abgesehen von meinen bereits erstellten Projekten..).



Der Startbildschirm gibt schon Aufschluss, was man alles im AutoIt Studio machen kann – Projekte erstellen, den Editormodus für einfache Dateien, Programmeinstellungen, Updates, Projektverwaltung... alles, was man braucht, um anständig zu entwickeln. Genau das, was wir brauchen!

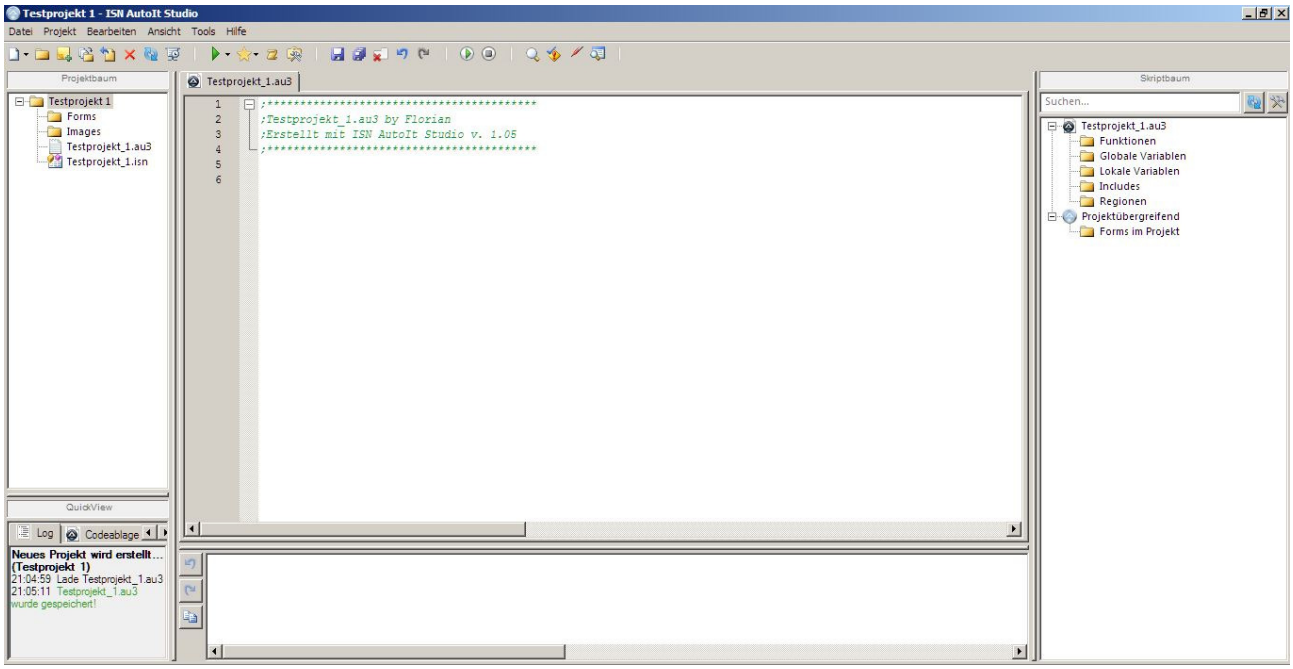
4.2 Das erste Projekt

Um ein neues Projekt zu erstellen, muss man auf "Neues Projekt erstellen" klicken. Wer hätt's gedacht? :)



Um Himmels willen! Was sind das denn alles für Optionen?

Es sieht schlimmer aus, als man denkt. Wir geben einfach den Namen "Testprojekt 1" ein, fügen vielleicht noch einen Kommentar dazu und klicken auf "Neues Projekt anlegen". Fertig. Alles andere macht das ISN AutoIt Studio automatisch.



Und so ist's auch. Zehn Sekunden später ist das Projekt bereit zum Start.

Eine wahnsinnig komplexe, aber tolle Entwicklungsumgebung.
Zunächst mal die grobe Einteilung:
Der Projektbaum enthält alle Dateien im Projektordner. In der QuickView gibt es eine Codeablage, einen automatischen Reporter und eine To-Do-Liste (Pfeil nach rechts in der Tabansicht). In der Mitte sieht man oben das Skript (mit Inhalt aus Vorlage), darunter die Ausgabe vom Compiler/Debugger (z.B. Fehlermeldungen). Der Skriptbaum enthält die eingestellten Inhalte aus dem aktuellen Skript.

Zu den Symbolen von links nach rechts:

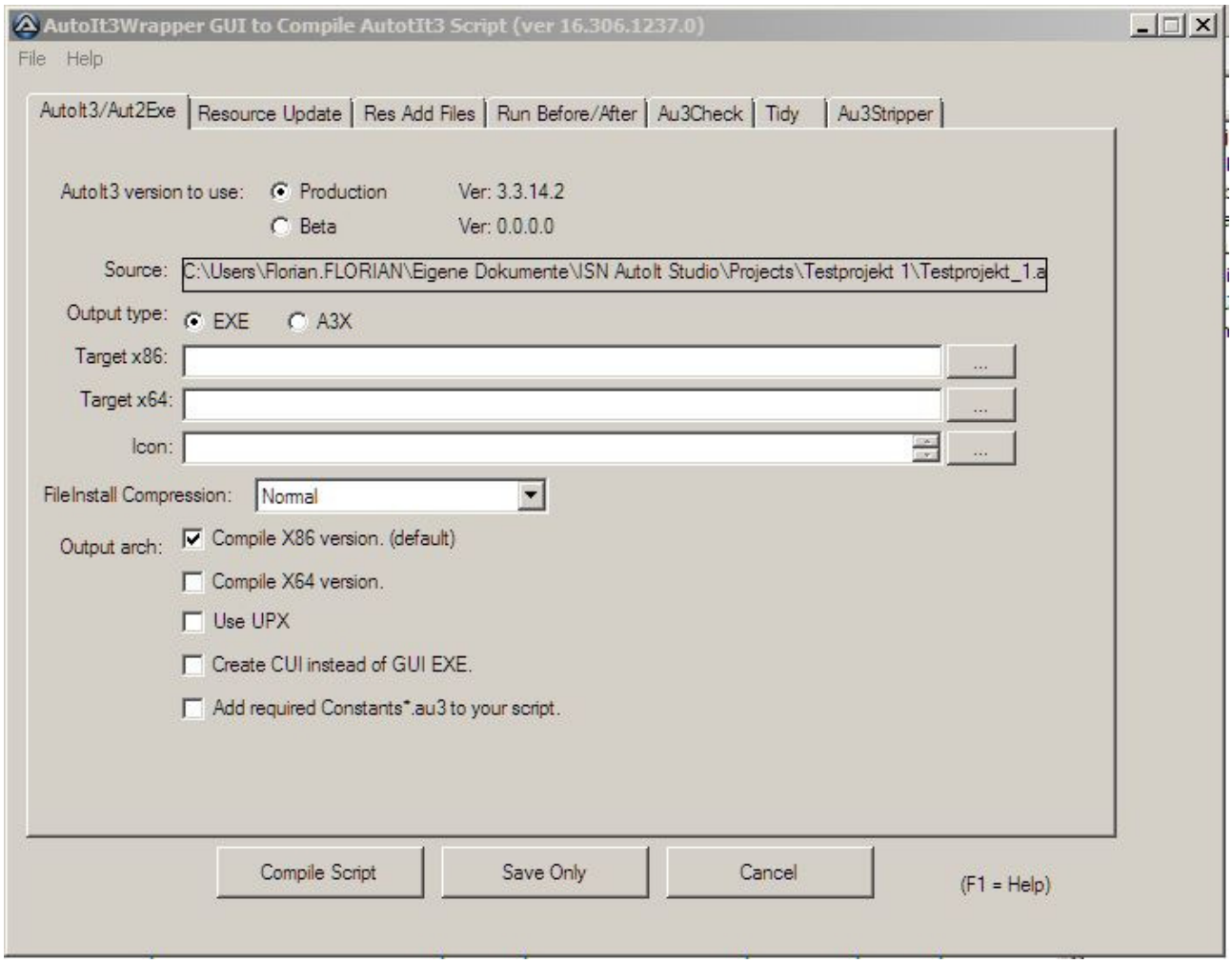
- Datei erstellen
- Ordner erstellen
- Datei(en) importieren
- Ordnerinhalt importieren
- gewähltes Element aus dem Projekt exportieren
- Datei löschen
- Projektbaum aktualisieren
- Vollbild
- Starten
- Kompilieren
- Makros
- Programmeinstellungen
- Speichern
- Alle Tabs speichern
- Aktuellen Tab schließen
- Rückgängig
- Wiederherstellen

- Aktuellen Tab starten
- Aktuelles Skript schließen
- Suchen
- Syntax-Check
- Zeile auskommentieren
- Fenster Info Tool

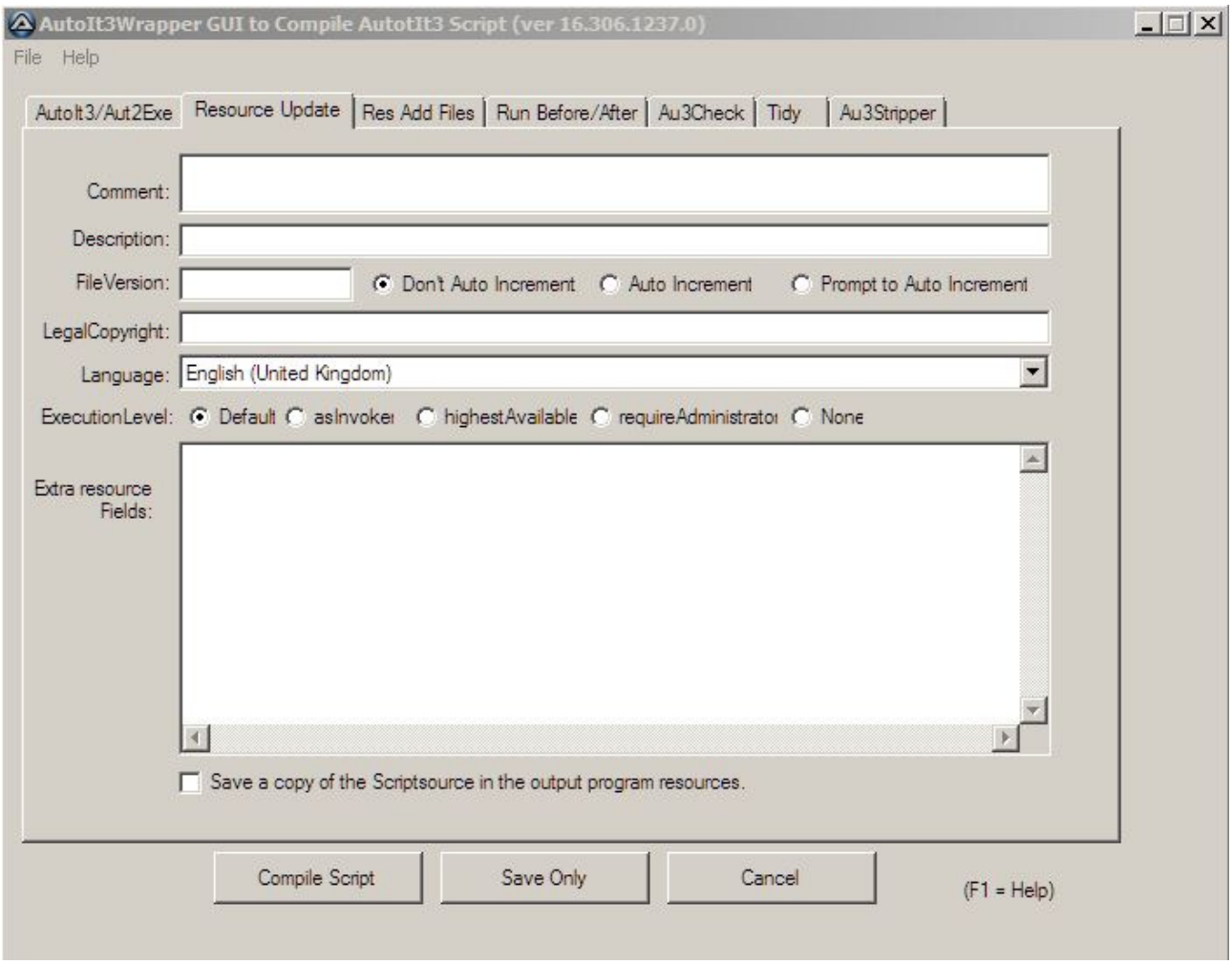
Von der bereits erwähnten *Kompilieren*-Option halte ich persönlich nichts. Ich bevorzuge den AutoIt3Wrapper.

4.3 Die AutoIt3WrapperGUI – der erweiterte AutoIt-Compiler

Der AutoIt3Wrapper wird über *Tools* > *AutoIt3Wrapper GUI* gestartet. So sieht die Oberfläche nach dem Start aus:



Hier kann man im Prinzip alle Optionen wählen – den 32-bit-Anwendungsnamen oder den 64-bit-Anwendungsnamen, ob 64-bit auch kompiliert wird, das Icon, ob komprimiert wird, sogar die Versionsinformationen (unter "Resource Update").



Hier kann man auch alle notwendigen Versionsinformationen einbinden – Kommentar, Beschreibung, Dateiversion und wie bei Rekompilierung mit ihr verfahren wird, Copyright, Sprache, erforderliche Rechte, extra Ressourcen... Im Prinzip werden für unsere Zwecke nur die ersten zwei Tabs und die drei unteren Knöpfe benötigt.

VORSICHT!!!!!!!!!!!!

Wer nicht unbedingt will, dass alle den Quellcode des Programms kennen: Pfoten weg von dem Kontrollhaken "Save a copy of the Scriptsource..."

4.4 Eine Projektvorlage erstellen

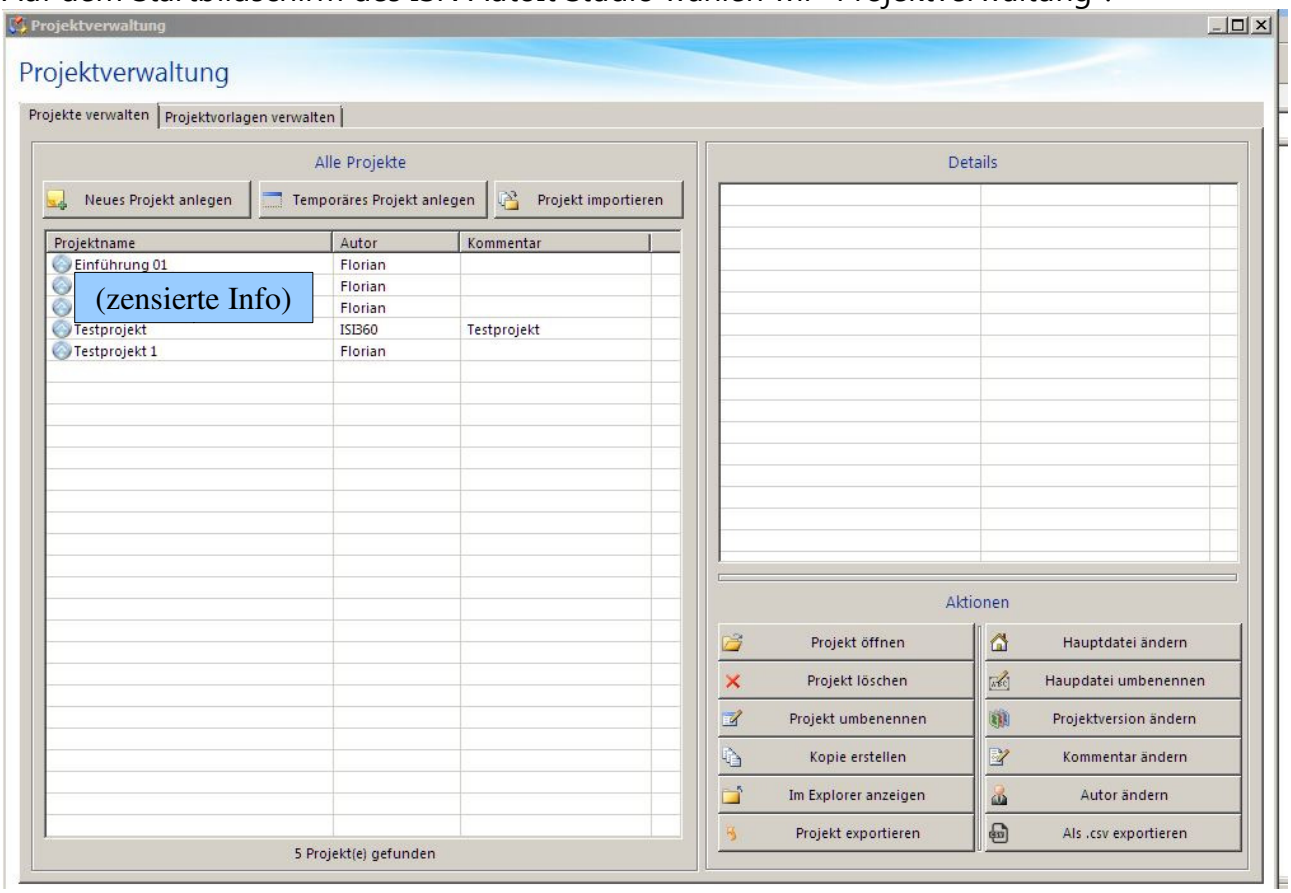
Der AutoIt3Wrapper legt seine Daten im aktuellen Skript-Tab ab (das man ja auch kompilieren will). Wir werden diese Eigenschaft für uns folgendermaßen nutzen, weil wir eine Projektvorlage erstellen werden.

1. Wir werden alle nötigen Felder irgendwie mit Platzhaltern versehen und "Save only" klicken, dass nicht für jedes Projekt der Wrapper neu gefüttert werden muss.
2. Wir werden alle Includes der Standard-UDFs einfügen.
3. Die Eigenschaft, dass standardmäßig kein Tray-Icon angezeigt wird, soll auch mit in der

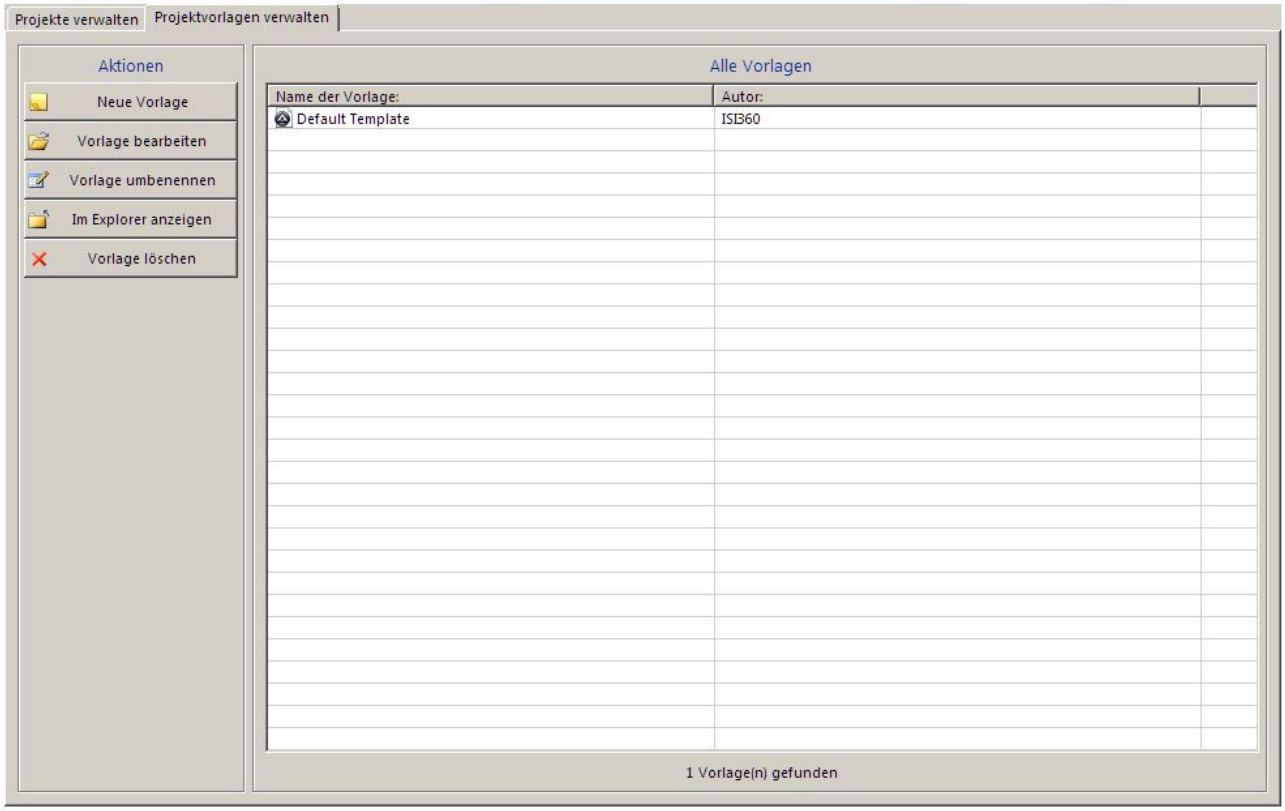
Vorlage landen. Die kann man ja später in neuen Projekten auch noch rauslöschen.

So, wie war nochmal die Direktive, kein Tray-Icon anzuzeigen?

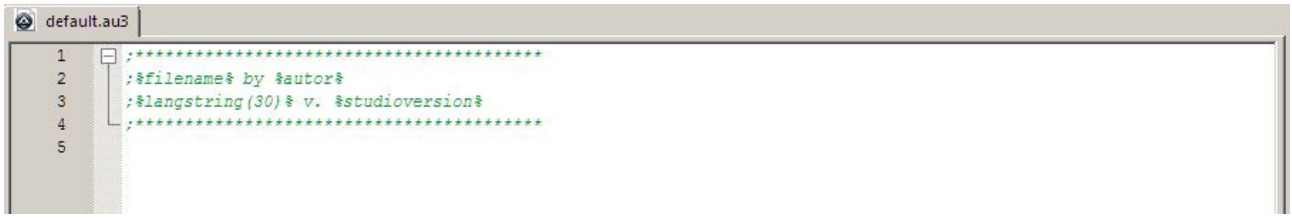
Auf dem Startbildschirm des ISN AutoIt Studio wählen wir "Projektverwaltung".



Wir wechseln zu "Projektvorlagen verwalten".



Bitte einmal ein Klick auf "Default Template" und unter "Aktionen" die Option "Vorlage bearbeiten" auswählen.
Folgender Code sollte (in etwa) dort eingegeben sein:



Codeerklärung:

- %filename% ist der Name der Hauptdatei
- %autor% der aktuelle Benutzername
- %langstring(XXX) ist ein String aus der Sprachdatei – hier "Erstellt mit ISN AutoIt Studio"
- %studioversion% ist die Version (hier normalerweise 1.05)

Wir wollen im folgenden diese Vorlage ändern bzw. eigentlich nur erweitern:

```
;****************************************
;%filename% by %autor%
;%langstring(30)% v. %studioversion%
;****************************************
```

Diese Zeilen sollen gleich bleiben. Jetzt wird erweitert – mit Zuhilfenahme der AutoIt3Wrapper-Direktiven:

```
#Region ;**** Directives created by AutoIt3Wrapper_GUI ****
#AutoIt3Wrapper_Icon=_ICON_.ico
#AutoIt3Wrapper_Outfile=%projectname%.exe
#AutoIt3Wrapper_Res_Comment=_KOMMENTAR_
#AutoIt3Wrapper_Res_Description=_BESCHREIBUNG_
#AutoIt3Wrapper_Res_Fileversion=1.0.0.0
#AutoIt3Wrapper_Res_LegalCopyright=(C) %autor%, %year%
#AutoIt3Wrapper_Res_Language=1031 ; Deutscher Code. Englisch: 1033
#EndRegion ;**** Directives created by AutoIt3Wrapper_GUI ****
```

Kurzer Hinweis: Leerzeichen können nach dem Ist-Gleich-Operator (=) ohne Probleme auftauchen (hier z.B. im Copyright)

Codeerklärung:

- Zeile 1: Die Region muss definiert werden, dass der AutoIt3Wrapper sie später evtl. ändern kann.
- Zeile 2: Platzhalter für ein Icon – der Pfad kann geändert werden.
- Zeile 3: Standardmäßiger Name der Output-EXE-Datei soll der Projektname sein – kann geändert werden.
- Zeile 4: Kommentar des Programms in den Versionsinformationen – änderbar
- Zeile 5: Beschreibung des Programms in den Versionsinformationen – änderbar
- Zeile 6: Dateiversion des Programms – änderbar (Produktversion ist immer die aktuelle AutoIt-Version – normalerweise 3.3.14.2)
- Zeile 7: Copyright des Programms in den Versionsinformationen – änderbar
- Zeile 8: Sprache des Programms in den Versionsinformationen – änderbar

```
#NoTrayIcon
Opt("GUICloseOnESC", 0)
```

Codeerklärung:

- Zeile 1: Die Direktive, dass das Tray-Icon ausgeschaltet wird. Das kann auch entfernt werden oder (empfohlen) das Standard-Tray-Menü geändert werden, da die Funktion "Paused script" im Menü das Skript pausiert und die Wiederherstellung äußerst unzuverlässig arbeitet.
- Zeile 2: `Opt("OptionName", Code)` setzt eine Option – in diesem Fall dass die Benutzeroberfläche nicht (Code 0 / FALSE) mit der Escape-Taste (Optionname GUICloseOnESC) geschlossen werden kann.

Die folgenden Zeilen sollen alle Include-Dateien (u.a. der Standard-UDFs) einbinden. Aufgrund der Tatsache, dass es sich dabei um 138 Include-Dateien handelt, werden diese hier außen vor gelassen.

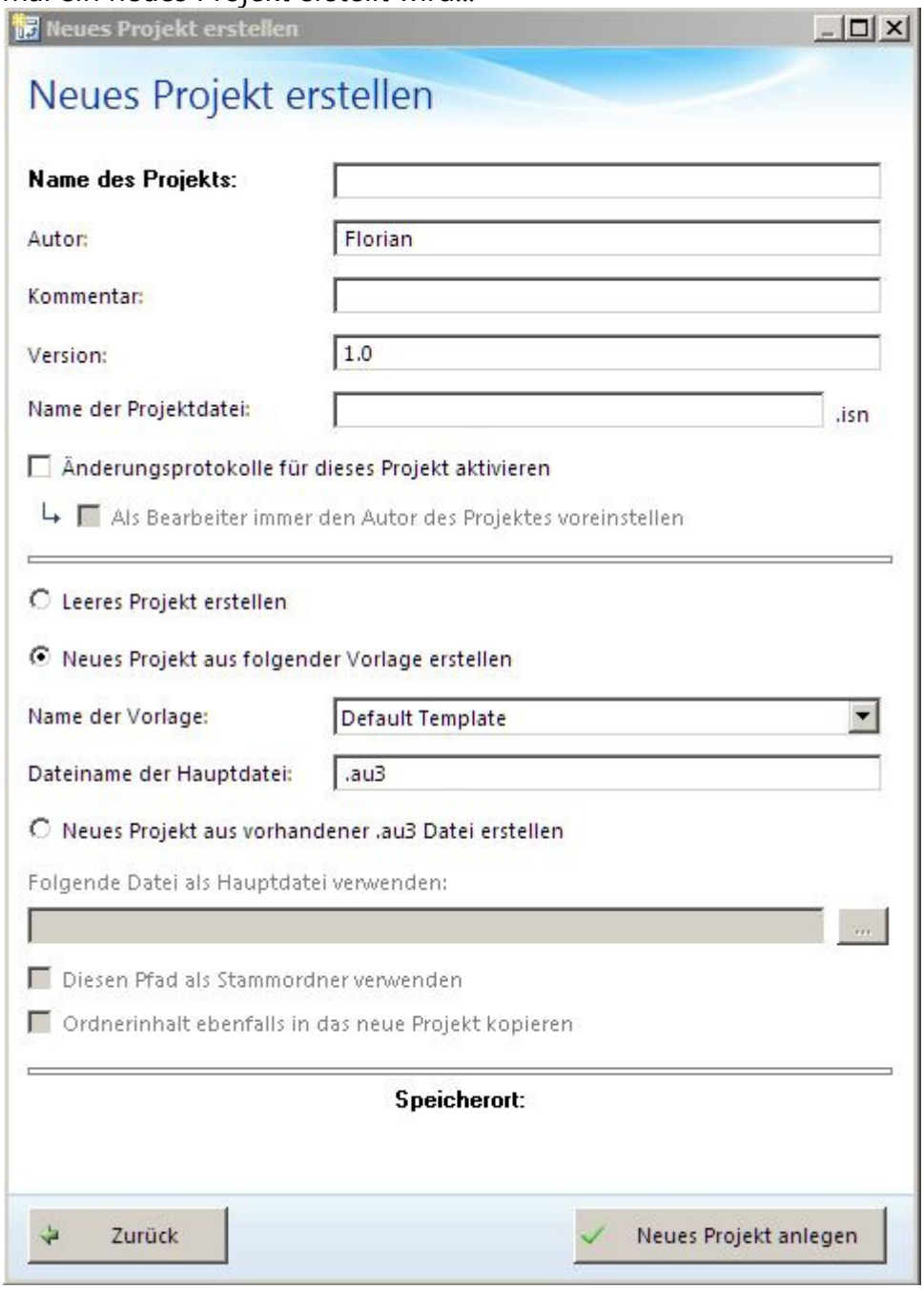
138 Includes?! Als ob so viele nötig wären.

Im Prinzip bräuchte man viele davon nicht. Das Programm ist so auch beim Start etwas langsamer. Aber so ist man auf der sicheren Seite und braucht nur

die Vorlage zu verwenden, ohne sich Sorgen zu machen, dass ein Include fehlt. (Forts.)

Danach wird der Tab gespeichert und die Vorlage kann geschlossen werden (Projekt > Aktuelles Projekt schließen).
Falls eine Bestätigung auftaucht, ob die Vorlage geschlossen werden soll, "Ja" zum Fortfahren klicken.

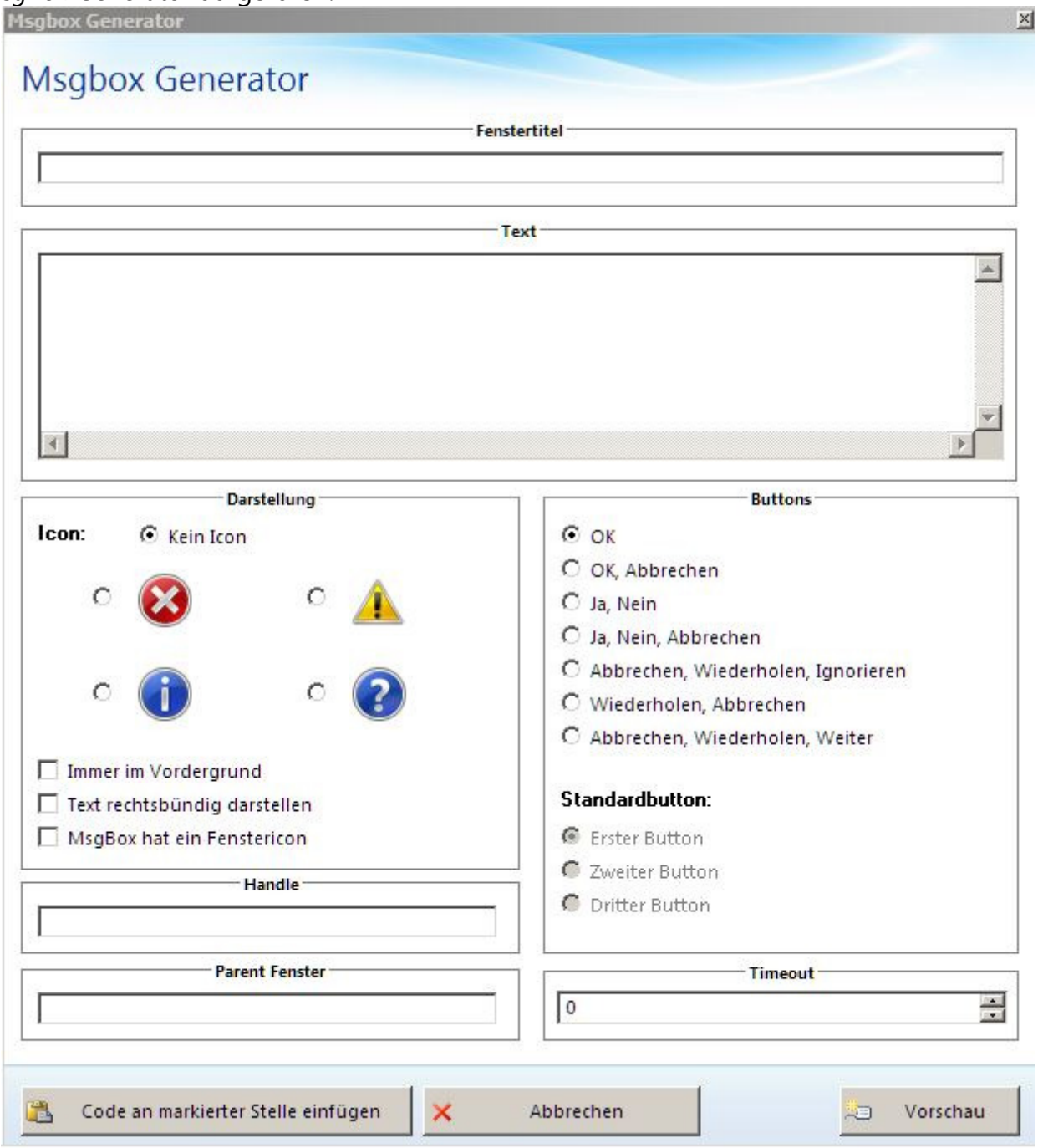
Wenn nochmal ein neues Projekt erstellt wird...



...kann einfach die Default-Template gewählt werden und das Projekt erstellt werden.

4.5 MsgBox Generator

Der MsgBox Generator ist ein Hilfsmittel für interaktive Infoboxen und wird über *Tools* > *MsgBox Generator* aufgerufen.



Optionen, die man in der Regel braucht:

- Titel: Titel der Infobox
- Text: Text der Infobox – Zeilenumbrüche möglich
- Darstellung: Darstellung der Infobox und dasIcon, das in der Infobox angezeigt werden soll
- Buttons: Knöpfe, die in der Infobox angezeigt werden
- Handle: Variablenname der Infobox, definitiv nötig bei mehr als einem Knopf
- Timeout: wann die Infobox sich schließen soll

Wir wählen folgende Optionen im Wizard:

- Titel: *imageres.dll – Dateifehler*
- Text: *Der Vorgang konnte nicht ausgeführt werden, weil die Datei imageres.dll auf Ihrem Computer fehlt.*
- Darstellung: *Fehlericon* (kein zusätzlicher Haken)
- Buttons: *Abbrechen, Wiederholen, Ignorieren*
- Handle: *FileError*
- Timeout: keines

Bei "Code an markierter Stelle einfügen" ergibt sich folgender Code:

```
$FileError = MsgBox(18,"imageres.dll - Dateifehler","Der Vorgang konnte nicht
ausgeführt werden, weil die Datei imageres.dll auf Ihrem Computer fehlt.",0)
switch $FileError

      case 2 ;CANCEL
      ;Your code here...

      case 4 ;RETRY
      ;Your code here...

      case 5 ;IGNORE
      ;Your code here...

endswitch
```

Dass die erste Zeile die Infobox erzeugt, ist ja noch klar. Aber warum soll *$FileError* jetzt eine Zahl enthalten? Müsste die Variable nicht ein Objekt mit allen Attributen der Infobox enthalten?

Grundsätzlich kennt AutoIt nur COM(=Component Object Model)-Objekte. Und nein, die Codeausführung läuft folgendermaßen ab:

1. Das Skript wird angehalten, bis ein Button in der Infobox bzw. der Schließen-Knopf der Infobox gedrückt wurde.
2. Die Variable *$FileError* enthält einen Rückgabewert, der eindeutig einem Knopf zugewiesen ist.

Wurde also der Knopf "Abbrechen" gedrückt, hat *$FileError* den Wert 2. Das lässt sich aus dem Code erschließen, in dem hinter den einzelnen Fällen die englischen Bezeichnungen der Buttons stehen. Anstatt `;Your code here...` kann eigener AutoIt-Code eingefügt werden.

Weiten wir also die Switch-Case-Anweisung aus:

```
switch $FileError

      case 2 ;CANCEL
            MsgBox(64, "Knopf gedrückt", "Es wurde auf [Abbrechen] geklickt!")

      case 4 ;RETRY
            MsgBox(64, "Knopf gedrückt", "Es wurde auf [Wiederholen] geklickt!")

      case 5 ;IGNORE
            MsgBox(64, "Knopf gedrückt", "Es wurde auf [Ignorieren] geklickt!")

endswitch
```

(Information: die Zahl 64 steht für eine Infobox mit dem Info-Icon und nur OK-Knopf)
So lassen sich also mit dem MsgBox Generator einfach interaktive Abfragen erstellen. Im Code-Listing wurden beide Optionen – mit leerer Vorlage und mit Aktion separat auskommentiert geschrieben. Entfernen Sie also #cs und #ce an den Stellen, wo die Infobox abgefragt werden soll.

4.6 UDF-Header

Erstellen wir eine Funktion, die Zahlen hochrechnet (nur natürliche Zahlen als Exponent!)

```
Func _Mathe_Hochrechnung($basis, $exponent)
	$ergebnis = 1

	For $i = 1 To $exponent
		$ergebnis = $ergebnis * $basis
	Next

	Return $ergebnis

EndFunc
```

Der Codeablauf sollte meiner Meinung nach jedem einleuchten.

Nein, mir nicht! Warum ist das Ergebnis von vornherein 1 und nicht 0?

Also doch den Codeablauf erklären, OK.
Potenzierung ist doch so definiert, dass die Basis sooft mit sich selbst multipliziert wird, wie der Exponent groß ist. Klar? Wenn ich also das Ergebnis mit 0 multipliziere, kommt immer null raus. Und da 1 mal Basis gleich Basis ist, kann ich ruhig auf diese Weise rechnen.

Wenn das Schlüsselwort der Funktion (Func) markiert wird und unter *Tools* > *Erstelle UDF-Header* ausgewählt wird, wird vorndran ein auskommentierter Kopf zur Funktionsbeschreibung gestellt. Diesen füge ich hier ein und ändere ihn ab – inklusiver der hundert Ist-Gleichs...

```
; #FUNCTION# ====================================================================
; Name ..........: _Mathe_Hochrechnung
; Description ...: Multipliziert $basis $exponent-mal mit sich selbst.
; Syntax ........: _Mathe_Hochrechnung($basis, $exponent)
; Parameters ....: $basis               - Reelle Zahl
;                  $exponent            - Natürliche Zahl
; Return values .: None
; Author ........: $basis $exponent-mal mit sich selbst multipliziert (Zahl)
; Modified ......: 24.06.2017
; Remarks .......: --
; Related .......: --
; Link ..........: --
; Example .......: No
;===============================================================================
```

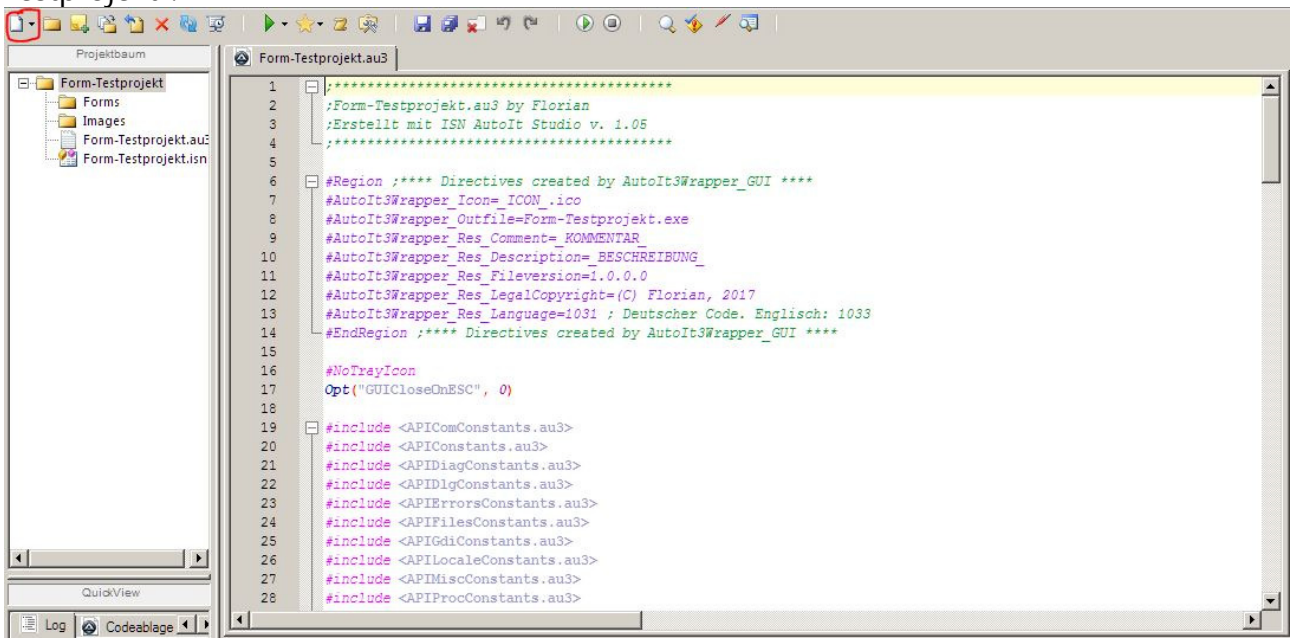
4.7 Erweiterte Projektoptionen

Eigentlich handelt es sich bei diesen Optionen nicht wirklich um spektakuläre Dinge, die meisten wurden bereits bei der Übersicht über die Toolbar-Items erwähnt.

- Dateien importieren: aus dem Dateisystem können Dateien ausgewählt werden, die in das Projektverzeichnis kopiert werden
- Ordner erstellen: Ordner im Projektverzeichnis erstellen
- Ordnerinhalt in Projekt importieren: aus dem Dateisystem kann ein Ordner ausgewählt werden, dessen Dateien in das Projektverzeichnis kopiert werden

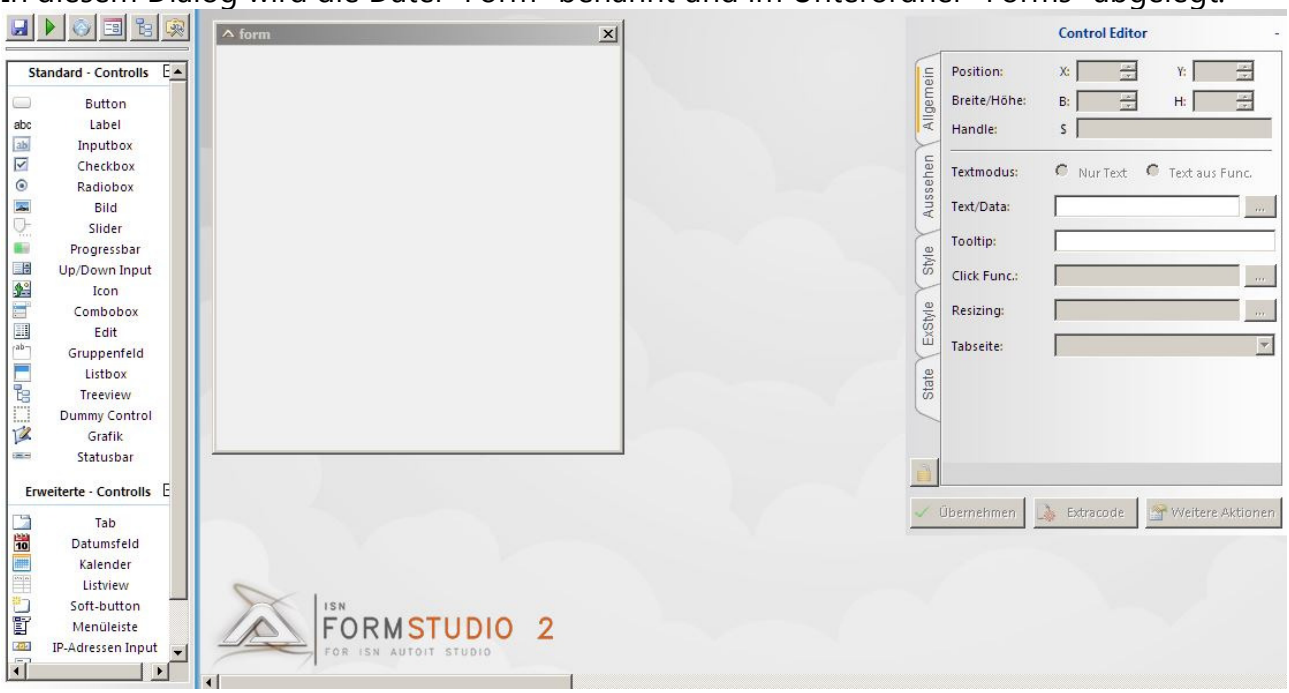
4.8 ISN Form Studio 2

Zunächst erstellen wir mit der Default-Template ein neues Projekt namens "Form-Testprojekt".

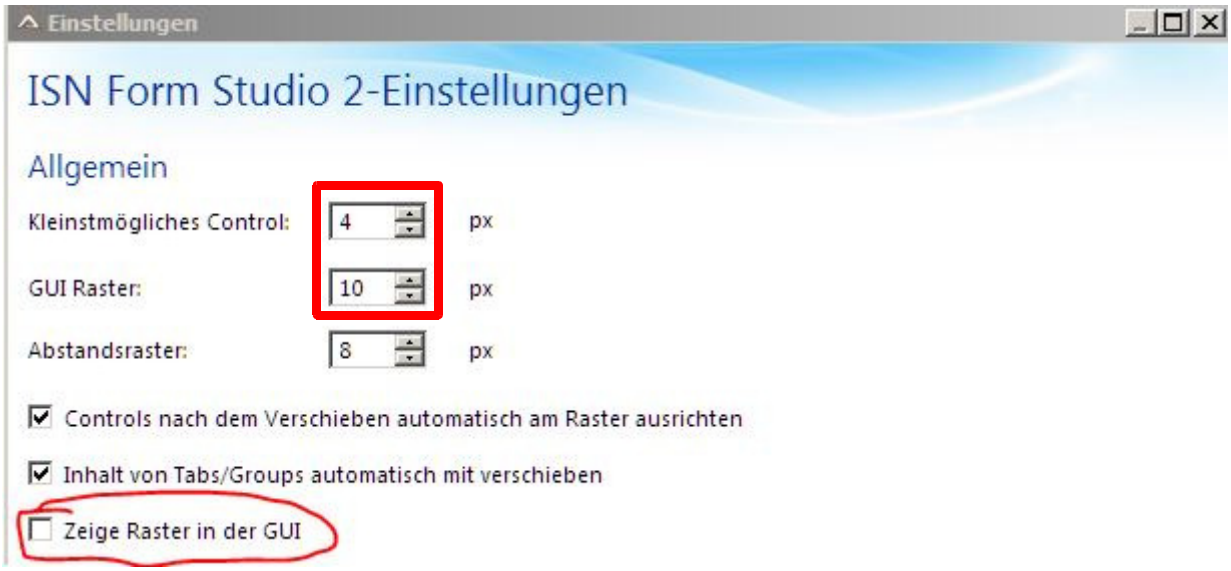


Wir brauchen das rot eingemarkerte Icon und wählen den Eintrag "AutoIt GUI (.isf) (ISN Form Studio 2)".

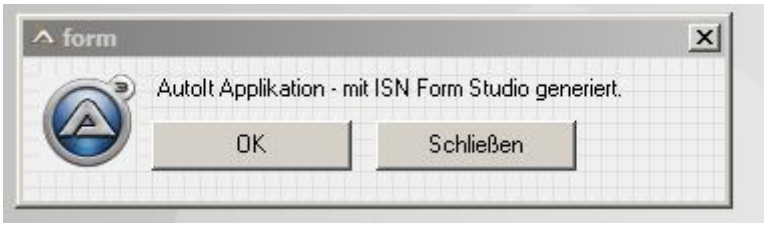
In diesem Dialog wird die Datei "Form" benannt und im Unterordner "Forms" abgelegt.



Bevor wir fortfahren, müssen wir in den Einstellungen zur besseren Übersicht (linke Spalte, Icon ganz rechts) das Raster aktivieren und alle Optionen auf 8px setzen.



Alle oberen Optionen sollten auf 8px eingestellt werden.
Wir designen uns eine kleine GUI – im Prinzip ist es egal, wie, Hauptsache, sie sieht ähnlich aus wie unten. Zusätzlich wird in den Projektordner das im Listing befindende "icon1.ico" importiert werden.



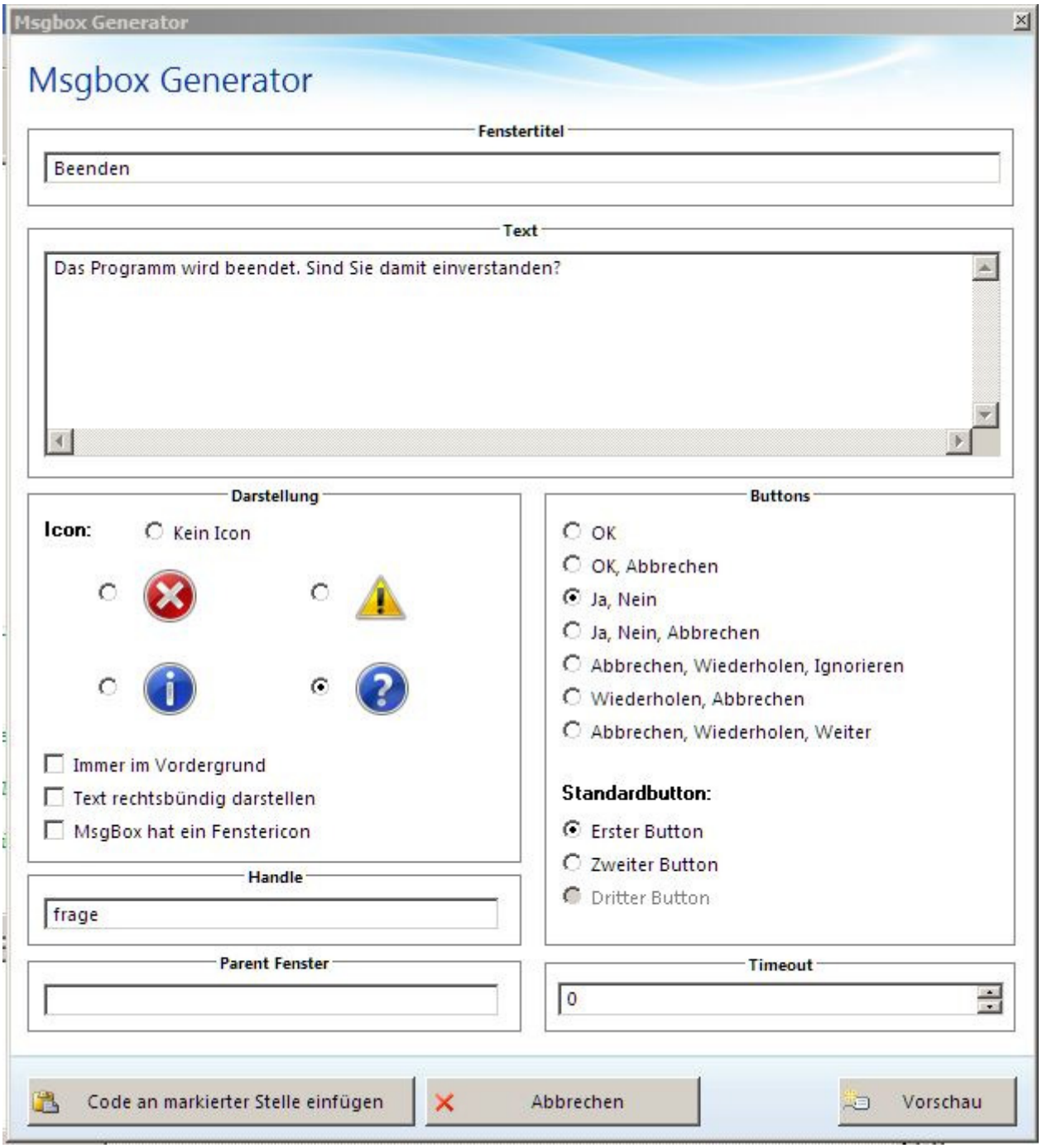
Durchgeführte Aktionen:

1. Icon importiert (das Bild) und als "Icon"-Element aus der Auswahlliste gewählt.
2. "Label"-Element erstellt ("AutoIt Applikation...") und entsprechend vergrößert, dass der gesamte Text sichtbar ist.
3. Zwei Buttons mit Höhe 25px nebeneinander eingefügt (Text "OK" und "Schließen")
4. Die Buttons erhielten die Handles *OK* für "OK" und *Exit* für "Schließen" (Dazu das jeweilige Element anklicken und im Control Editor den Namen im Handle eingeben – ähnlich zur Identifikation des Rückgabewerts wie bei der Infobox).

Dieser Tab kann nun geschlossen werden. Wurden alle Haken vorhin bei der Form-Erstellung richtig gesetzt, sollte am Ende der Hauptdatei (normalerweise "Form-Testprojekt.au3") folgende Zeile stehen:

```
#include "Forms\form.isf"
```

Falls diese Zeile nicht im Skript steht, bitte manuell eingeben.



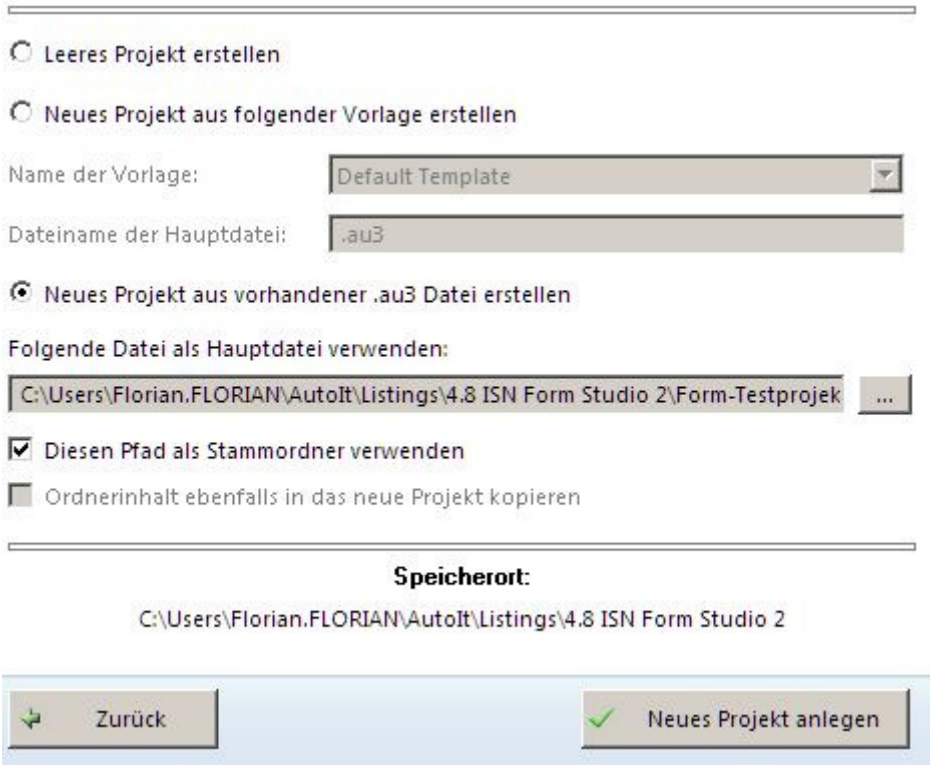
Zuerst wird im MsgBox-Generator noch ein kleines Codestück mit diesen Einstellungen oben erzeugt und Aktionscode eingefügt:

```
                  $frage = MsgBox(36,"Beenden","Das Programm wird beendet. Sind
Sie damit einverstanden?",0)
                  switch $frage

                        case 6 ;YES
                              Exit

                        case 7 ;NO
                              MsgBox(16, "Fehler", "Sie sind nicht
einverstanden. Also wird auch nicht geschlossen.")

                  endswitch
```

Danach werden folgende Zeilen eingegeben:

```
GUISetState(@SW_SHOW) ; zeigt die Benutzeroberfläche an
While 1
      Switch GUIGetMsg()
            Case $GUI_EVENT_CLOSE ; Schließen-Knopf oben rechts
                  Exit
            Case $OK ; OK-Knopf in der GUI
                  ; Code aus dem MsgBox-Generator
                  $frage = MsgBox(36,"Beenden","Das Programm wird beendet. Sind
Sie damit einverstanden?",0)
                  switch $frage

                        case 6 ;YES
                              Exit

                        case 7 ;NO
                              MsgBox(16, "Fehler", "Sie sind nicht
einverstanden. Also wird auch nicht geschlossen.")

                  endswitch

            Case $Exit ; Schließen-Knopf in der GUI
                  Exit
      EndSwitch
Wend
```

In einer Endlosschleife (While 1) wird ermittelt, welches Element in der GUI, also der Wert aus GUIGetMsg(), geklickt wurde. $GUI_EVENT_CLOSE ist eine Konstante für "Schließen" rechts oben im Fenster, die anderen Items ($OK und $Exit) sind die Handles aus der im Form Studio generierten Oberfläche.
Der grüne Pfeil in der Toolbar startet die Benutzeroberfläche (Ergebnis s. Listing / Video1).



(Neues Projekt – aus Listing Projekt erstellen – Voraussetzung: beschreibbarer Ordner)

4.9 Updates

Nach Updates für das ISN AutoIt Studio kann manuell unter *Hilfe* > *Nach Updates suchen* gesucht werden.
Es öffnet sich ein Fenster, das den Updatestatus abruft.

In diesem Fall wurde nach Fertigstellung des Abrufens am 24.06.2017 kein Update gefunden.