Control-Events in AutoIt

Einleitung

GUI

Labels

Buttons

InputBox

ListBox

ComboBox

Edit

Graphic

CheckBox

Radio-Control

Slider

UpDown

Nutzung des Windows-Nachrichtendienstes

- `Label`
- `Button_OK`
- `ListBox`
- `ComboBox`

ListBox – Datenein- und Ausgabe

Einleitung

In der AutoIt-Hilfe wird ausgeführt, dass mit

```
Opt("GUIOnEventMode", 1)
```

die grundsätzliche Aktivierung des GUI-Event-Modus möglich ist, d.h. dass damit sowohl vom GUI selbst, wie auch von ihren Controls so genannte „Events“ (Ereignisse) generiert werden können, die dann auswertbar sind.
Da in der AutoIt-Hilfe nur sehr wenige Informationen über Details dieser der Windows-Philosophie so grundlegenden Arbeitsweise vorhanden sind, erhebt sich die Frage:
„Was bedeutet das im Einzelnen?“

Die bloße Erwähnung, bzw. die recht dürftigen Hinweise in der AutoIt-Hilfe bringen oft nicht genügend Klarheit, oder man kommt mit der berühmt-berüchtigten „*Try & Error*“-Methode nur mühsam ans Ziel.

Von den 29 in AutoIt verfügbaren Controls sollen deshalb hier die wichtigsten im Zusammenhang mit dem GUI einer eingehenden Untersuchung unterzogen werden.

GUI

Für das GUI sind drei Ereignisse von besonderer Bedeutung, weil sie von den GUI-eigenen Control-Boxen erzeugt werden:

- Schließen - (X)
- GUI-Fenster „herunterklappen“ (auf die Statusleiste) (_)
- GUI-Fenster wiederherstellen (□)

Da bei der globalen Aktivierung des GUI-Event-Modus `GUIGetMsg` nicht verwendet werden kann, müssen in einer speziellen Funktion die dazu notwendigen Aktionen generiert werden:

(nach der Erzeugung des GUI)

```
GUISetOnEvent($GUI_EVENT_CLOSE, "SpecialEvents")
GUISetOnEvent($GUI_EVENT_MINIMIZE, "SpecialEvents")
GUISetOnEvent($GUI_EVENT_RESTORE, "SpecialEvents")
```

Die dazugehörende Funktion sieht dann wie folgt aus:

```
Func SpecialEvents()                        ; hier sind die Special-Events der GUI:
                                            ; Fenster Schliessen, Minimieren, Maximieren,

      ;MsgBox (0, "ID", @GUI_CtrlId)        ; nur zum Test
      Select
        Case @GUI_CtrlId = $GUI_EVENT_CLOSE
            ;MsgBox(0, "Schliessen gedrückt", "ID=" & @GUI_CtrlId & _
                                " WinHandle=" & @GUI_WinHandle)
            Exit

        Case @GUI_CtrlId = $GUI_EVENT_MINIMIZE
            ;MsgBox(0, "Fenster minimiert", "ID=" & @GUI_CtrlId & _
                                " WinHandle=" & @GUI_WinHandle)

        Case @GUI_CtrlId = $GUI_EVENT_RESTORE
            ;MsgBox(0, "Fenster wiederhergestellt", "ID=" & @GUI_CtrlId & _
                                " WinHandle=" & @GUI_WinHandle)

    EndSelect
;
EndFunc
```

Die auskommentierten `MsgBoxen` dienen lediglich Testzwecken, d.h. sie haben sonst keinerlei Funktion.
In der AutoIt-Hilfe gibt es eine Liste mit weiteren Funktionen (`Special-ID`), die hier nicht weiter behandelt werden. Die Event-Funktion auf ein normales `OnClick`-Ereignis fehlt jedoch.
Will man ein normales `OnClick`-Ereignis auf das GUI auswerten, kommt man mit

```
GUISetOnEvent($Form1, "GUI_OnClick")
```

nicht zum Erfolg. Offensichtlich wird dieses „`GUI_OnClick`“-Event nicht unterstützt!
Mit einem Trick ist das aber durchaus möglich.
Dazu wird das Control-Ereignis für `Label` „missbraucht“ (siehe dort).

Von den oben erwähnten Funktionen (`Special-ID`) ist vielleicht im Zusammenhang mit dem GUI doch noch eines interessant: `$GUI_EVENT_RESIZED`.

Dieses Ereignis tritt immer dann ein, wenn das GUI mit der Mouse auf eine andere Größe gezogen wurde. Vorraussetzung dazu ist allerdings, dass das GUI mit dem Style `$WS_SIZEBOX BitOR-`verknüpft mit `$WS_MAXIMIZEBOX, $WS_MINIMIZEBOX` als Parameter erzeugt wird. Das weitere Handling, welches zugegebenermaßen nicht ganz unkompliziert ist, wurde bereits im Hauptteil unter 4. beschrieben.

Labels

Normalerweise werden `Label` nur für die Beschriftung der GUI verwendet. Die dafür vorhandenen Click-Ereignisse sind von untergeordneter Bedeutung, weil sie im Prinzip nicht gebraucht werden und somit kaum vorkommen, wenn sie nicht zu speziellem Zweck programmiert werden.
Wie bereits unter „GUI“ erwähnt wurde, gibt es kein normales „OnClick“-Ereignis auf dem GUI. Man kann jedoch das für `Label` vorhandene „OnClick“-Ereignis dafür „missbrauchen“. Das geht so:

```
$Form1 = GUICreate("Test", 800, 600, 200, 100)
	; erzeugt GUI, mit Namen und Größen-Parameter: Breite, Höhe, von links, von oben
$Label1 = GUICtrlCreateLabel("", 0, 0, 800, 600)
	; erzeugt Schriftzug, mit Namen und Größen-Parameter: von links, von oben, Breite, Höhe
```

Es wird ein `Label`-Control mit der gleichen Größe des GUI erzeugt, jedoch ohne Schriftinhalt. Somit ist auf dem GUI nichts zu sehen.
Wird jetzt mit

```
GUICtrlSetOnEvent ($Label1, "Label1_Click")
```

der Event-Handler

```
Func Label1_Click()
;
	MsgBox(0, "Test", "Click auf Label1")
;
EndFunc
```

aktiviert, kann jedem Click auf irgendeine Stelle des GUI das Ereignis ausgelöst werden.

Werden nachfolgend weitere Controls programmiert, so sind diese aber wieder sichtbar, weil sie in der Layerschichtung über `$Label1` liegen. Das Click-Ereignis von `$Label1` ist davon unberührt.
Was passiert aber mit den Events der darüber liegenden Controls? Welches hat dann Vorrang, bzw. in welcher Reihenfolge werden die Events erzeugt?
Bei den nachfolgend behandelten Buttons soll das mit untersucht werden.

Buttons

Der Eventmodus bei Buttons ist von fundamentaler Bedeutung.
Nach der Erzeugung eines Buttons, z.B.

```
$Button1 = GUICtrlCreateButton("OK", 200, 500, 60, 25)
        ; erzeugt eine Schaltfläche, Parameter: von links, von oben, Breite, Höhe
```

kann dann mit

```
GUICtrlSetOnEvent($Button1, "Button1_OK")
```

das `Button1`-Event zu aktivieren.
Wird jetzt auf die Schaltfläche (`Button`) geklickt, so wird die Funktion „`Button1_OK`“ aufgerufen, worin dann die der Schaltfläche zugedachten Aktionen programmiert werden können, im einfachsten Fall:

```
Func Button1_OK()
;
      MsgBox (0, "Button1", $Button1)      ; nur zu Testzwecken,
                                           ; es wird der Rückgabewert der Erstellungsfunktion
                                           ; von Button1 ausgegeben
;
EndFunc
```

Wird die Schaltfläche auf das bereits GUI-weit bestehende `$Label1` gesetzt, so ist diese zwar sichtbar, aber ohne jede Funktion, d.h. ein `Button1_OK`-Event kommt nicht mehr zustande.
Wird das GUI-weite `$Label1` versuchsweise auskommentiert (das Auskommentieren von

`GUICtrlSetOnEvent ($Label1, "Label1_Click")` reicht nicht), funktioniert das `Button1_OK` -Ereignis.
Wie kann dieser Effekt umgangen werden?
Zunächst muss die Erstellungs-Reihenfolge incl. der Event-Aktivierung geändert werden:

1. `Button1_OK`
2. `$Label1`

Damit ist aber zunächst `Button1_OK` nicht sichtbar, jedoch kann mit jedem darüber liegenden Child-Fenster, z.B. einer `MsgBox` die Sichtbarkeit hergestellt werden.
Programm-technisch wird das einfacher gelöst, indem nach der `$Label1`-Erstellung noch einmal die gleiche Schaltfläche `Button1_OK` generiert wird.
Nun ist sowohl die Schaltfläche `Button1_OK` sofort sichtbar und auch beide Events - `Button1_OK` und `Label1_Click` - sind aktiviert.

InputBox

Die Event-Aktivierung einer `InputBox` läuft nach dem gleichen Schema ab, wie bei Button.
In der AutoIt-Hilfe steht unter `GUICtrlSetOnEvent` (Zitat):
„Definiert eine nutzerdefinierte Funktion, die aufgerufen wird, wenn ein Control angeklickt wird."
Nach den von mir gemachten Erfahrungen ist das im Fall der `InputBox` nicht korrekt.
Beim Klicken auf eine `InputBox` wird kein (!) Event erzeugt - wohl aber, wenn irgendwelche Veränderungen (`Changes`) am Inhalt der `InputBox` durch Eingaben vorgenommen wurden und danach die `InputBox` den Focus verliert, weil z.B. auf ein anderes Control geklickt wurde.
Werden Inhalts-Veränderungen programmtechnisch erzeugt (z.B. mit `GUICtrlSetData`), wird kein `InputBox`-Ereignis erzeugt, auch nicht bei Focus-Verlust.
Der Code sieht beispielsweise so aus:

```
$input = GUICtrlCreateInput ("text", left, top, width, height)
                            ; erstellt eine Listbox, Parameter: von links, von oben, Breite, Höhe
                            ; weitere Parameter zu „Styles" – siehe AutoIt-Hilfe

GUICtrlSetOnEvent ($Input, "Input_Change")
```

der dazugehörende Event-Handler im einfachsten Fall so:

```
Func InputChange()
```

```
;
      MsgBox (0, "Input1", $Input)
;
EndFunc
```

Wohl gemerkt – das `InputBox`-Ereignis wird nur ausgelöst, wenn sich Änderungen am Inhalt der `Inputbox` ergeben haben und die `InputBox` anschließend den Focus verloren hat!
Wird dazu auf eine Schaltfläche geklickt, wird zwar das `InputBox`-Ereignis ausgelöst, aber das `Button`-Ereignis wird ignoriert. Ein abermaliges Clicken darauf löst allerdings dann das `Button`-Ereignis aus.

ListBox

Eine `ListBox` dient in der Regel zur Darstellung von mehrzeiligen Datenblöcken. Eine direkte Eingabemöglichkeit besteht nicht. Wenn solche benötigt wird, muss man die `ComboBox` verwenden.
Die Erstellung der Listbox kann wie folgt vorgenommen werden:

```
$list = GUICtrlCreateList ("text1", left, top, width, hight)
              ; erstellt eine Listbox, Parameter: von links, von oben, Breite, Höhe
              ; es kann gleich ein Anfangstext übergeben werden ( -> 1.Zeile)
              ; mehrzeilige "text"-Einträge können nicht mit "@CRLF" erzwungen werden!
              ; weitere Parameter zu „Styles“ – siehe AutoIt-Hilfe
```

Ein nachträglicher Daten-Eintrag mit

```
GUICtrlSetData ($list, "text2")
              ; schreibt eine neue Zeile mit "text2" in die ListBox
```

Soll nun die Event-Funktionalität aktiviert werden, so ergibt sich mit

```
GUICtrlSetOnEvent ($list, "ListBox_Click")
```

dass die angeclickte Zeile markiert (selektiert) wird und ein `OnClick`-Ereignis eintritt. Im einfachsten Fall ist das wieder mit

```
Func ListBox_Click()
;
      MsgBox (0, "ListBox_Click", $list)  ; nur zu Testzwecken, es wird der
                                          ; Rückgabewert der Erstellungsfunktion
                                          ; von „ListBox“ ausgegeben
;
EndFunc
```

zu überprüfen. Da in einer `ListBox` keine Eingaben vorgenommen werden können, gibt es im Gegensatz zur `InputBox` in diesem Sinne auch kein `Change`-Ereignis - auch nicht, wenn der Eintrag programmtechnisch wie oben beschrieben erfolgt.
Wenn allerdings das Verändern der Zeilen-Selektion als Change-Ereignis angesehen werden soll, so stellt sich die Frage, wie daraus eine Information über die selektierte Zeile gewonnen werden kann. Dazu kommt dann folgender Befehl zum Einsatz:

```
$sel = ControlCommand ("Test", "", $List, "GetCurrentSelection", "")
```

zu überprüfen wie üblich mit

```
MsgBox (0, "ListBox_Click", $sel)                ; nur zu Testzwecken
```

Vorsicht ist geboten bei Anwendung des o.g. GUI-Click-Ereignisses mit dem GUI-weiten `Label`. Dann wird zunächst wieder nichts zu sehen sein, wenn die `Listbox` vor dem GUI-weiten `Label` erzeugt wurde. Anderenfalls ist die Funktionalität eingeschränkt. Abhilfe schafft auch hier wieder das Verfahren, wie unter „Buttons“ beschrieben.

ComboBox

Im Gegensatz zu einer `Listbox` sind mit einer `ComboBox` auch Eingaben möglich.
Zur Erstellung wird im Allgemeinen

```
$Combo = GUICtrlCreateCombo ("", 250, 50, 100, 100)
                    ; erstellt eine Combobox, Parameter: von links, von oben, Breite, Höhe
```

verwendet (zu Alternativen später). Zur Verwendung der weiteren, optional zu verwendenden Parameter sei auf die AutoIt-Hilfe verwiesen.

Obwohl Breite und Höhe wie bei der ListBox angegeben ist, wird zunächst nur die Eingabezeile der `ComboBox` angezeigt. Das „Runterklappen“ mit dem rechtsangeordnetem Pfeil funktioniert solange nicht, wie die `ComboBox` leer ist. Eine Eingabe überträgt so einfach nicht die Eingabedaten in das Listenfeld der `ComboBox`. Dies kann aber z.B. bereits bei der Erstellung mit

```
$Combo = GUICtrlCreateCombo ("test", 250, 50, 100, 100)
```

für einen String „test“ vorgenommen werden, oder wie bereits bei der `ListBox` mit

```
GUICtrlSetData ($Combo, "text1")     ; schreibt eine neue Zeile mit "text" in die ComboBox
```

Erst wenn Daten-Zeilen vorhanden sind (eine reicht schon), funktioniert auch das „Runterklappen“.
Nach dem programmtechnischen Eintrag ist zunächst wieder nichts zu sehen, weil nur die leere Eingabe-Zeile abgebildet wird. Nach dem „Runterklappen“ sind aber die Zeilen-Einträge sichtbar und mit einem Mouse-Click auf die betreffende Zeile wird diese selektiert, wobei diese dann gleichzeitig in die Eingabezeile kopiert wird. Veränderungen in der Eingabezeile haben allerdings keine Auswirkungen auf die eigentlichen Daten-Zeilen der `ComboBox`.
Im Gegensatz zur Erstellungs-Sequenz können hier mit dem Datenseparator "|" Mehrzeileneinträge erzwungen werden:

```
GUICtrlSetData ($Combo, "text2|text3")
```

Für eine permanente Darstellung auch des Listenfeldes der `ComboBox` muss bereits bei der Erstellung ein weiterer Parameter angegeben werden:

```
$Combo = GUICtrlCreateCombo ("", 250, 50, 100, 100, $CBS_SIMPLE)
```

Auch hier ist mit

```
GUICtrlSetOnEvent ($Combo, "ComboBox_Click")
```

ein Aktivieren der Event-Funktionalität möglich. Jede durch Mouse-Click selektierte Daten-Zeile hat ein Ereignis `ComboBox_Click` zur Folge - einfach zu überprüfen mit:

```
Func ComboBox_Click()
;
      MsgBox (0, "ListBox_Click", $Combo)       ; nur zu Testzwecken,
                                                ; es wird der Rückgabewert der
                                                ; Erstellungsfunktion von ListBox
                                                ; ausgegeben
      $sel = ControlCommand ("Test", "", $Combo, "GetCurrentSelection", "")
      MsgBox (0, "ListBox_Click_Sel", $sel)     ; nur zu Testzwecken
;
EndFunc
```

Auch hier kann wieder mit `ControlCommand - "GetCurrentSelection"` die zu selektierende Zeile bestimmt werden.

Offen ist jedoch - wie kann man den Eingabewert programmtechnisch auslesen?
Dazu ist es dann notwendig eine UDF (*user defined function*) zu bemühen und so zu sagen von „außen“, d.h. mittels des expliziten Auslese-Befehles:

```
$selEd = _GUICtrlComboBox_GetEditText($Combo)
      MsgBox (0, "ListBox_Click_SelEd", $selEd)       ; nur zu Testzwecken
```

ist dann in `$selEd` der aktuelle Eingabestring enthalten.
Da das `OnClick`-Ereignis nicht aus dem Edit-Fenster heraus erzeugt wird, kann somit die Event-Funktionalität hier leider auch nicht verwendet werden. Offensichtlich wird zuerst die selektierte Daten-Zeile in das Edit-Fenster kopiert, bevor das `OnClick`-Ereignis entsteht. Mit anderen Worten - der Inhalt des Edit-Fensters der `ComboBox` wird sofort durch die selektierte Daten-Zeile überschrieben.
Sinn sollte es aber machen, einen Eingabewert, z.B. als neue Daten-Zeile in der `ComboBox` erscheinen zu lassen.

Edit

Beim `Edit`-Control ist eigentlich nur die mehrzeilige Variante der bereits besprochenen `InputBox` gemeint. Die Erstellung wird wie üblich vorgenommen mit:

```
$edit = GUICtrlCreateEdit ("text", left, top, width, height)
                    ; “text” = Vorgabetext, left = Position von links, top = Position von oben,
                    ; width / height = optionale Angaben für Breite und Höhe
                    ; weitere optionale Angaben siehe AutoIt-Hilfe
```

Die Event-Funktionalität wird dann wieder mit

```
GUICtrlSetOnEvent ($edit, "EditBox_Click")
```

aktiviert. Das daraus sich ergebende Event-Verhalten ist ähnlich dem, wie bereits für die `ListBox` beschrieben, d.h. nur beim Fokus-Verlust und einer Änderung des Inhalts der `EditBox` wird das Ereignis erzeugt und der betreffende Event-Handler aufgerufen, hier wieder im einfachsten Fall mit:

```
Func EditBox_Click()
;
      MsgBox (0, "EditBox_Click", $edit)
;
EndFunc
```

Ausgegeben wird die ID der `EditBox`. Ein Zugriff auf den Inhalt der `EditBox` mit:

```
$sel = GUICtrlRead ($edit)
```

ist nur möglich, wenn sich daran etwas verändert hat. Es wird immer nur der gesamte Text ausgelesen.
Zu beachten ist, dass bei der `ListBox` immer dann ein Zugriff auf eine Daten-Zeile möglich ist, wenn diese markiert (selektiert) ist, während bei der `EditBox` unbedingt eine Inhalts-Änderung vorliegen muss, sonst wird kein Ereignis ausgelöst. Selektierungen bei der `EditBox` sind nur temporär stabil - im Gegensatz zur `ListBox`, wo diese auch dann bestehen bleiben, auch wenn die `ListBox` den Fokus verliert.

Alternativ kann natürlich wieder unter zu Hilfenahme einer UDF, in diesem Fall die „GUI Edit Management“ eine Vielzahl von Spezialfunktionen zur Anwendung gebracht werden.

Dazu muss `#include <GuiEdit.au3>` eingefügt werden. Auf weitere Einzelheiten soll an dieser Stelle zunächst verzichtet werden. Näheres siehe dazu in der AutoIt-Hilfe.

Graphic

Der Umgang mit der `GraphicBox` wurde in seinen Grundzügen ausführlich schon im 3. und 4. Abschnitt im Hauptteil beschrieben.
Wenn eine `GraphicBox` mit

```
$graph = GUICtrlCreateGraphic(left, top, width, height, $SS_NOTIFY)
                    ; erstellt eine GraphicBox, Parameter: von links, von oben, Breite, Höhe
```

erstellt wird, so ist zunächst noch nichts zu sehen. Erst wenn mit

```
GUICtrlSetBkColor($graph, 0xffffff)        ; Hintergrundfarbe: weiss
```

die Hintergrundfarbe entsprechend im Gegensatz zur Farbe des GUI verändert wird, ist die `GraphicBox` sichtbar.
Zwar ist mit dem Parameter `$SS_NOTIFY` bereits die Event-Funktionalität gegeben (siehe AutoIt-Hilfe), aber die läßt sich so ohne weiteres nicht auswerten (es sei denn, man bemüht wiederum den Windows-Nachrichtendienst).
Wird allerdings wieder die einfache Version mit

```
GUICtrlSetOnEvent ($graph, "GraphicBox_Click")
```

verwendet, so ist dann mittels des entsprechenden Event-Handlers:

```
Func GraphicBox_Click()
;
      MsgBox (0, "GraphicBox_Click", $graph)     ; nur zu Testzwecken,
                                                 ; es wird der Rückgabewert der
                                                 ; Erstellungsfunktion von GraphicBox
                                                 ; ausgegeben
;
EndFunc
```

die Event-Funktion nachweisbar.
Welche Aktionen anstelle der nur zu Testzwecken verwendeten `MsgBox` programmiert werden sollen, ist unmittelbar von der(den) zu lösenden Aufgabe(n) abhängig.
Ein Zugriff auf den Inhalt der `GraphicBox` ist nicht möglich. Da in der `GraphicBox` keine Texte abgelegt werden können hat eine Abfrage mit:

```
$sel = GUICtrlRead ($graph)
```

auch immer einen Leerstring in `$sel` zur Folge. Es kann lediglich mit

```
GUICtrlSetGraphic ($graph, type, pm1, ... pm6)  ; type: Zeichenfunktion
                                                ; pm1…pm6: Parameter der Zeichenfunktion
```

Einfluß auf den Inhalt genommen werden, d.h. hier sind die verschiedensten Zeichenfunktionen implementiert (näheres dazu in der AutoIt-Hilfe).
Die Problematik des Umgangs mit den Zeichenfunktionen wurde bereits im Hauptteil ausführlich erörtert.
Erwähnenswert sei aber im Zusammenhang mit der Verwendung der Event-Funktion der `GraphicBox` noch die Ermittlung der aktuellen Mouse-Zeigerposition beim Click auf diese.
So können bestimmte Parameter visueller Punkte auf dem Bild der `GraphicBox` ermittelt werden:

```
$Mpos = MouseGetPos()
```

In `$Mpos` sind die aktuellen Mouse-Positionsdaten in einem Array mit 2 Elementen enthalten.
Wichtig dabei ist die Modus-Einstellung dieser Funktion mit:

```
AutoItSetOption ( "MouseCoordMode" , pm )       ; pm    = 0: relativ,
                                                        = 1: absolut,
                                                        = 2: rel. zur Fensterfläche(Client)
```

Alternativ läßt sich zu `MouseGetPos()` auch die Funktion

```
GUIGetCursorInfo()
```

einsetzen, welche noch den Vorteil hat, dass damit auch die verschiedenen Mouse-Tasten unterschieden werden können (weitere Infos siehe AutoIt-Hilfe).
Welche weiteren Aktionen mit den Informationen zum aktuellen Cursor-Punkt vorgenommen werden sollen, hängt natürlich von der Aufgabenstellung ab. Im Hauptteil wurde damit die Realisierung von „Selection-Points“, bzw. „Selection-Bereichen“ für die in der `GraphicBox` abgebildeten Grafik-Elemente vorgenommen. Denkbar ist aber z.B. auch die Ermittlung der Farbinformation zu dem Pixel des in der `GraphicBox` geladenen Bildes, was sich gerade beim Click unter dem Cursor befindet.

CheckBox

Auch bei der `CheckBox` gibt es das von der Windows-Philosophie herrührende Ereignis.
Die Erstellung einer `CheckBox` erfolgt mit:

```
$Check = GUICtrlCreateCheckbox ( "text", left, top, width, height)
                                    ; “text” schreibt diesen rechts neben der CheckBox
                                    ; left = von links, top = von oben
                                    ; width = Breite der CheckBox incl. „text“
                                    ; height = Höhe
```

Breite und Höhe haben keinen Einfluß auf die eigentliche Größe der `CheckBox`. Lediglich die „text“-Darstellung wird davon beeinflusst. Zu weiteren optionalen Parametern siehe AutoIt-Hilfe.
Die Event-Funktionalität wird wieder mit:

```
GUICtrlSetOnEvent ($Check, "CheckBox_Click")
```

aktiviert und mit dem betreffenden Event-Handler:

```
Func CheckBox_Click()
;
      MsgBox (0, "GraphicBox_Click", $Check)    ; $Check = ID der CheckBox
;
$sel = GUICtrlRead ($Check)
MsgBox (0, "CheckBox_Click_Sel", $sel)
;
EndFunc
```

läßt sich dann in der zweiten MsgBox für den Rückgabewert ermitteln:

1 = aktiviert ($GUI_CHECKED)

4 = deaktiviert ($GUI_UNCHECKED)

Radio-Control

Ein `Radio`-Control (“Optionsknopf”) wird mit:

```
$radio = GUICtrlCreateRadio ("text", left, top, width, height)
```

erstellt. Auch hier haben wieder Breite und Höhe keinen Einfluß auf die eigentliche Größe der `RadioBox`. Lediglich die „text“-Darstellung wird davon beeinflusst.
Im Gegensatz zur `CheckBox` kann die `RadioBox` zwar durch Mouse-Click gesetzt werden, aber nicht wieder zurück den Ausgangszustand.
Zu weiteren optionalen Parametern siehe AutoIt-Hilfe.
Die Event-Funktionalität wird wieder mit:

```
GUICtrlSetOnEvent ($radio, "RadioBox_Click")
```

hergestellt und mit dem betreffenden Event-Handler:

```
Func RadioBox_Click()
;
      MsgBox (0, "RadioBox_Click", $radio)
;
      $sel = GUICtrlRead ($Radio)
      MsgBox (0, "RadioBox_Click_Sel", $sel)
;
      GUICtrlSetState ($radio, $GUI_UNCHECKED)
      MsgBox (0, "RadioBox_Click_Sel", GUICtrlRead ($Radio))
;
EndFunc
```

wird sowohl die Funktionalität nachgewiesen, als auch mit `GUICtrlSetState ($radio, 4)` die Rückstellung der `RadioBox` vorgenommen. Das Ereignis wird bei jedem Mouse-Click auf die `RadioBox` erzeugt, unabhängig davon, ob diese aktiviert ist oder nicht.

Oft werden `RadioBox`-Controls in Gruppen verwendet, wobei sich bei der Aktivierung von einer `RadioBox` die vorher aktivierte deaktiviert. Diese Funktionalität kann entweder mit dem oben beschriebenen Event-Handler (ggf. anpassen bzgl. der Aktivierung/Deaktivierung) erfolgen, oder es wird mit dem Gruppen-Control gearbeitet:

```
$group = GUICtrlCreateGroup ("text", left, top, width, height)
```

Auf weitere Eigenarten und spezielle Programmierung soll an dieser Stelle verzichtet werden, siehe dazu die AutoIt-Hilfe.

Slider

Das `Slider`-Control, oder auch Schieberegler genannt, wird wie folgt erzeugt:

```
$slider = GUICtrlCreateSlider (left, top, width, height)
```

Die dazugehörende Event-Funktionalität wird wiederum mit:

```
GUICtrlSetOnEvent ($slider, "Slider_Click")
```

aktiviert, somit kann mit der Event-Handler:

```
Func Slider_Click()
;
      MsgBox (0, "Slider_Click", $slider)
      ;
      $sel = GUICtrlRead ($slider)
      ;
      MsgBox (0, "Slider_Click_Sel", $sel)
      ;
EndFunc
```

dann die momentane Stellung des Schiebereglers in `$sel` ausgelesen werden
(Wertebereich von 0 … 100, oder in %).
Das Ereignis tritt allerdings erst ein, wenn die Mouse-Taste losgelassen wird.

UpDown

Als letztes Control soll hier noch UpDown etwas näher betrachtet werden.

Das UpDown-Control – auch Pfeiltasten-Control genannt - wird ganz einfach erzeugt mit:

```
$upDown = GUICtrlCreateUpdown (inputcontrolID)
                                    ; inputcontrolID = ID der InputBox,
                                    ; auf den sich das Control bezieht
```

UpDown-Controls werden immer im Zusammenhang mit einer InputBox benutzt zum bequemen Auf-/Abwärtszählen bei Eingabe-Aktionen von Zahlen.

Das UpDown-Control wird immer direkt rechtsbündig in die InputBox gesetzt. Wichtig dabei ist, dass die InputBox mit genügend großer Höhe definiert wird (>20), sonst ist die „Bedienung“ des UpDown-Controls sehr problematisch.

Jedes Betätigen des UpDown-Controls bewirkt bereits die Auslösung des InputBox-Ereignisses, ohne dass die Eingabe in der InputBox von Hand verändert werden muss (siehe dazu den Abschnitt InputBox). Obwohl hier ebenfalls intern die Veränderung herbeigeführt wird, kommt es zum Auslösen des Events, im Gegensatz zu programmtechnisch veranlassten Änderungen am Inhalt der InputBox.

Wird wieder mit:

```
GUICtrlSetOnEvent ($upDown, "UpDown_Click")
```

zusätzlich noch das UpDown-Ereignis aktiviert, so ist festzustellen, dass in jedem Fall zuerst das InputBox-Ereignis erzeugt wird und danach erst dasjenige von UpDown.

Mit dem betreffenden Event-Handler kann das leicht überprüft werden:

```
Func UpDown_Click()
;
      MsgBox(0, "Test", "UpDown-Event")
;
EndFunc
```

Nutzung des Windows-Nachrichtendienstes

Für die o.g. Auslese-Funktion des Edit-Fensters der `ComboBox` und noch eine ganze Reihe weiterer interessanter Funktionen für dieses Control muß folgende UDF „includiert“ (eingebunden) werden.

```
#Include <GuiComboBox.au3>
```

Damit ist aber leider noch nicht die Aufgabe gelöst, eine Eingabe im Edit-Fenster irgendwie nutzen zu können, ohne jedes Mal ein Fremd-Ereignis (z.B. OK-Bottum) benutzen zu müssen. In den Hilfe-Texten zu

`_GUICtrlComboBox_Create` bzw.
`_GUICtrlComboBoxEx_Create` (Unterstützung von Image-Items)

sind in den Beispielen andere Varianten des Event-Managements aufgeführt. Letzteres ist durch Einbeziehung von DLL-Strukturen derart kompliziert, dass zunächst darauf verzichtet werden soll. In den Beispielen wird mit

```
GUIRegisterMsg($WM_Command, "WM_Command_Handler")
        ; WM_Command_Handler = Ereignishandler für die verschiedenen Windows-Nachrichten
```

bzw.

```
GUIRegisterMsg($WM_NOTIFY, "WM_Notify_Handler")
```

die Funktionalität des Windows-Nachrichtendienstes ausgenutzt, um verschiedene Aktionen durchzuführen. Das Verfahren ist allerdings komplizierter, als man das sonst von AutoIt-Funktionen gewöhnt ist und außerdem ist zusätzlich noch

```
#include <WindowsConstants.au3>
```

zu „includieren“.
Mit der oben gezeigten einfachen Variante des Ereignis-Handlers kommt man dann der Funktionalität allerdings auch noch nicht näher, da laufend Windows-Nachrichten erzeugt werden und somit die `MsgBox` nicht mehr geschlossen werden kann. Man muss also die Windows-Nachrichten in geeigneter Weise selektieren, um zu den relevanten Informationen zu kommen.

Zunächst ist aus der o.g. UDF die modifizierte Version der Funktion zur Erzeugung einer `ComboBox` zu verwenden (mit der alten Variante ist das Nachfolgende sonst wirkungslos):

```
$Combo = _GUICtrlComboBox_Create($Form1, "", 250, 50, 100, 100, $CBS_SIMPLE)
```

Auch die Einfügung von Daten in die Daten-Zeilen der `ComboBox` funktioniert dann nur noch mit den neuen Funktionen

```
_GUICtrlComboBox_AddString($Combo, "Text1") oder
_GUICtrlComboBox_InsertString($Combo, "Text2", $iIndex)
        ; Einfügung an die mit $iIndex bezeichneten Daten-Zeile
```

In dem `...Create...`-betreffenden Hilfe-Beispiel ist ein Auswerte-Mechanismus aufgeführt, der für eine Event-Auswertung des Edit-Fensters der `ComboBox` von Interesse ist. Dazu wird ein vereinfachter `WM_Command_Handler` programmiert:

```
Func WM_Command_Handler($hWnd, $iMsg, $iwParam, $ilParam)
```

Die Variablen `($hWnd, $iMsg, $iwParam, $ilParam)` sind bereits in den Include-Dateien deklariert und geben z.T. kryptische Zahlenwerte zurück:

```
MsgBox (0, "ComboBox_Click", $Combo & "   " & $hWnd & "   " & $iMsg & _
                                                    "   " & $ilParam)
```

Damit kann man allerdings noch nichts anfangen. Deshalb wird diese `MsgBox` deaktiviert und die empfangenen Nachrichten etwas „zerlegt“:

```
$hWndFrom = $ilParam
$iIDFrom = BitAND($iwParam, 0xFFFF)        ; separieren niederwertiges Wort
                                           ; ($iwParam ist ein Doppelwort)
$iCode = BitShift($iwParam, 16)            ; separieren höherwertiges Wort
```

Die Überprüfung der Ergebnisse mit

```
MsgBox(0, "$hWndFrom & iIDFrom & iCode", $hWndFrom & "   " & $iIDFrom & _
                                                    "   " & $iCode)
```

bringt zwar noch nicht wirklich die gewünschten Informationen, aber damit kann schon untersucht werden, in welcher Reihenfolge von welchen Controls Ereignisse auftreten. Daraus lassen sich dann Schlussfolgerungen ziehen, wie die einzelnen Ereignisse ausgewertet werden können. Zu beachten ist allerdings, dass die Aktivierung von `GUIRegisterMsg` grundsätzlich alle Controls der betreffenden GUI beeinflusst. In dem bisher behandelten Beispiel sind das:

- `Label1`
- `ListBox`
- `Button1`
- `ComboBox`

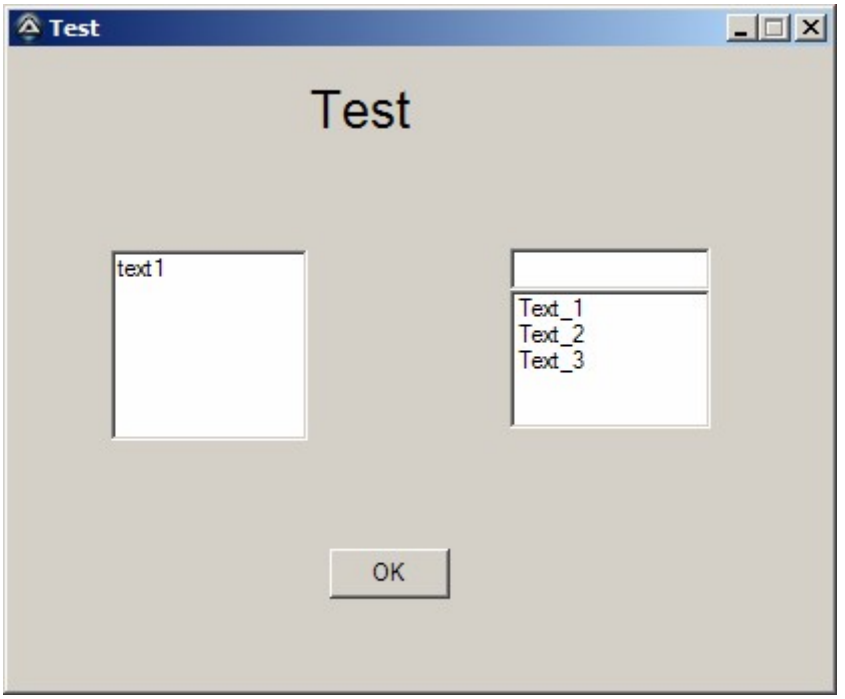
Für die einzelnen Controls gibt es unterschiedliche Anzahl von Event-Meldungen (`NOTIFICATIONS`), so sind z.B. für die `ComboBox` u.a. diese interessant (`ComboConstants`):

- `SELCHANGE = 1`
- `DBLCLK = 2`
- `SETFOCUS = 3`
- `KILLFOCUS = 4`
- `EDITCHANGE = 5`
- `EDITUPDATE = 6`
- `DROPDOWN = 7`
- `CLOSEUP = 8`
- `SELENDOK = 9`
- `SELENDCANCEL = 10`

Für die ListBox gilt folgendes (`ListBoxConstants`):

- `SELCHANGE = 1`
- `DBLCLK = 2`
- `SELCANCEL = 3`
- `SETFOCUS = 4`
- `KILLFOCUS = 5`

Welche dabei wirklich relevant sind und in welcher Reihenfolge sie auftreten, soll nun untersucht werden. Dazu wird das GUI gestartet (vereinfachte Version, nur auf die Controls `Label, Bottum, ListBox` und `ComboBox` beschränkt) und das dabei entstehende Erscheinungsbild beobachtet:



Der Fenster-Ausdruck muss mit „AltGr + Druck“ erfolgen. Mit der normalen „Alt“-Taste wird der Focus bereits auf `Button1_OK` gelegt, was allerdings keine Event-Meldungen zu Folge hat.
Wenn man mit dem Cursor das GUI und die darauf befindlichen Controls „abfährt“, sind keine Veränderungen, sowohl beim Cursor selbst, als auch bei den Controls zu beobachten, allerdings mit einer Ausnahme:
Im Bereich des Edit-Fensters der `ComboBox` verändert sich der Cursor bereits zu dem bekannten Erscheinungsbild für eine Eingabe.
Nun zu den Meldungen im Einzelnen und deren Reihenfolge ihres Auftretens.

`Label`

Wird das `Label1` angeclickt, so wird eine Meldung mit

```
$hWndFrom    =      0x00E80306  (kann nach jedem Start ein anderer Wert sein!)
$iIDFrom     =      3 (Label)
$iCode       =      0
```

ausgelöst, der nicht in o.g. Liste vorkommt. Anschließend hat dann automatisch `Button1_OK` den Focus.

`Button_OK`

Wird auf `Button_OK` geclickt, ohne das dieses vorher den Fokus hatte (sichtbar an den verstärkten Kanten), so wird eine Meldung mit

```
$hWndFrom   =     0x010402C4
$iIDFrom    =     4 (Button)
$iCode      =     0
```
ausgelöst. Allerdings wird anschließend sofort der betreffende Event-Handler ausgeführt, was zur Folge hat, dass die Datenzeile „text2“ in die `ListBox` eingefügt wird. Der Eintrag ist dabei so schnell, dass noch Zeit bleibt, die im Event-Handler von `Button_OK` abschließend vorhandene Abfrage

```
$selEd = _GUICtrlListBox_GetText($List, _GUICtrlListBox_GetCount($List) - 1)
```

erfolgreich auszuführen. Sie gibt den letzten Eintrag, d.h. „text2“ zurück. Der Fokus verbleibt auf `Button1_OK` und es erfolgt zunächst keine Markierung irgendeiner Daten-Zeile der `ListBox`, bzw. der `ComboBox`.
Ein Fokus-Wechsel zwischen dem GUI (Click auf das GUI, nicht auf ein Control!) und einem anderen Windows-Fenster erzeugt jetzt keine weiteren Event-Meldungen!

`ListBox`

Der Click auf den leeren (!) Bereich der `ListBox` bringt eine Ereignis-Meldung mit folgendem Inhalt:

```
$hWndFrom   =     0x009A0314
$iIDFrom    =     5 (ListBox)
$iCode      =     4 (SetFokus)
```

`Button_OK` hat den Fokus verloren, d.h. die Kantenverstärkung ist verschwunden. An der `ListBox` ist dabei noch keine Veränderung zu erkennen. Wird anschließend in ein anderes Windows-Fenster gewechselt, so wird

(1) `$iCode = 1` (ChangeFokus)

(2) `$iCode = 5` (LostFokus)

erzeugt und anschließend der `ListBox_Click_Handler` aufgerufen. Da aber keine Daten-Zeile in der `ListBox` selektiert ist, wird `ListBox_Click_Sel = 0` zurückgegeben.
Zurück auf das GUI geclickt wird dann auch wieder `$iCode = 4` (SetFokus) erzeugt.

Wird jetzt wieder auf einen leeren Bereich der `ListBox` geclickt, so ergibt sich folgende Ereignis-Reihenfolge:

(1) `$iCode = 1` (ChangeFokus)

(2) `$iCode = 5` (LostFokus) 3x (!)

dann wird der `ListBox_Click_Handler` aufgerufen (angezeigt mit der betreffenden `MsgBox`, die allerdings sofort wieder den Fokus verliert!) und wenn man diese schließen will, wird sofort wieder ein Ereignis

`$iCode = 5` (LostFokus)

erzeugt, was dann gefolgt wird von der Ausgabe des `ListBox_Click_Handlers` mit `ListBox_Click_Sel = 0`.

Ein Click auf eine Daten-Zeile der `ListBox` bewirkt folgendes Ereignis:

(1) `$iCode = 4` (SetFokus)

gefolgt von der Markierung der angeclickten Daten-Zeile. Eigenartigerweise wird zunächst kein `OnClick`-Ereignis des normalen Event-Mechanismus (`GUICtrlSetOnEvent`) ausgelöst. Erkennbar ist dieser Zwischenzustand daran, dass die Markierung einfach durch Mouse-Bewegung hin- und her verschoben werden kann, ohne jedoch darauf clicken zu müssen. Wird jetzt aber auf eine ausgewählte Markierung nochmals geclickt, ergibt sich folgende Ereignis-Reihenfolge:

(1) `$iCode = 1` (ChangeFokus)

(2) `$iCode = 5` (LostFokus) 3x (!)

dann wird der `ListBox_Click_Handler` aufgerufen (angezeigt durch die betreffende `MsgBox`, die allerdings sofort wieder den Fokus verliert!) und wenn man diese schließen will, wird sofort wieder ein Ereignis

(3) `$iCode = 5` (LostFokus)

erzeugt. Nach Schließung der `MsgBox` wird wiederum

(5) `$iCode = 4` (SetFokus)

erzeugt, was dann gefolgt wird von der Ausgabe des `ListBox_Click_Handlers` mit `ListBox_Click_Sel = “text1“ (bzw. „“text2“)` und abschließend wird dann noch mal:

(6) `$iCode = 4` (SetFokus)

erzeugt.

Erkennbar ist der Status SetFokus – LostFokus auch an der Farbveränderung der blauen Kopfleiste des Windows-Fensters.

Bemerkenswert ist auch noch, dass bei jeder Fokus-Veränderung zu einem anderen Windows-Fenster (z.B. eines anderen Programms) auch ein

`$iCode = 5` (LostFokus)

und bei der Rückkehr zum GUI (durch Click auf das GUI) dann

`$iCode = 5` (LostFokus) 2x (!)

erzeugt wird.

`ComboBox`

Da es sich bei der `ComboBox` im Windows-Prinzip um zwei Fenster handelt, sind auch zwei verschiedene Fälle zu behandeln:

(1) Click in das Eingabe-Fenster

(2) Click in das Daten-Fenster

Hatte die `ComboBox` bisher noch nicht den Fokus - was leicht zu erkennen ist, weil weder im Eingabe-Fenster der Eingabe-Cursor blinkt, noch eine Daten-Zeile (wenn vorhanden) mit dem blauen Markierungsbalken versehen ist, so wandelt sich der Mouse-Zeiger über dem Eingabe-Fenster in die bekannte Eingabe-Form um. Über dem Daten-Fenster sind keine Veränderungen des Mouse-Zeigers zu beobachten.

Wird jetzt in das Eingabe-Fenster geclickt, so ergibt sich folgende Ereignis-Reihenfolge (bezogen auf die `ComboBox`):

`$iCode = 5` (LostFokus)	ListBox-Ereignis, nur wenn diese vorher den Focus hatte!
`$iCode = 3` (SetFokus)	ComboBox-Ereignis, nur wenn vorher die ListBox nicht den Focus hatte!

Kennzeichnend ist jetzt das Blinken des Eingabe-Cursors im Eingabe-Fenster der `ComboBox`.

Wird zurück auf die `ListBox` geclickt, so ergibt sich Folgendes:

`$iCode = 4` (KillFokus)	ComboBox-Ereignis
`$iCode = 4` (SetFokus)	ListBox-Ereignis

Die `ListBox` befindet sich nun in dem o.g. Zwischenzustand (frei beweglicher Markierungsbalken ohne Click).

Wieder zurück in das Eingabe-Fenster der `ComboBox` wird aber nur das `ListBox`-Ereignis:

`$iCode = 5` (LostFokus)	ListBox-Ereignis

erzeugt (s.o.).

Anders ist das, wenn in das Daten-Fenster geclickt wird. Auch hier sind wieder zwei Fälle zu unterscheiden:

(1) Click auf eine freie Fläche

(2) Click auf eine Daten-Zeile

Wird eine freie Fläche im Daten-Fenster der ComboBox angeclickt und diese hatte vorher nicht den Focus (Eingabe-Cursor blinkt nicht), so wird lediglich der blinkende Cursor im Eingabe-Fenster gesetzt, weitere Ereignisse ergeben sich nicht.
Wird hingegen auf eine Daten-Zeile geclickt, so wird folgende Ereignis-Reihenfolge erzeugt:

```
$iCode = 9 (SelEndOK)      ComboBox-Ereignis
$iCode = 1 (SelChange)     ComboBox-Ereignis
```

Anschließend wird sofort der Inhalt der betreffenden Daten-Zeile in das Eingabe-Fenster kopiert.
Bemerkenswert ist auch hier wieder der Zwischenzustand der ComboBox (wie schon bei der ListBox beschrieben), welcher sich z.B. ergibt, wenn der Fokus zu dem Windows-Fenster einer anderen Anwendung gewechselt ist und anschließend wieder zurück direkt auf das Daten-Fenster der ComboBox geclickt wurde.

Um die soeben beschriebenen Ereignis-Vorgänge in Bezug auf die ComboBox weiter zu verarbeiten, muss der oben aufgeführte Programm-Code noch etwas erweitert werden:

```
If $hWndFrom = $Combo Then                          ; wenn Sende-Fenster = ComboBox
      If $iCode = $CBN_EDITCHANGE   Then            ; relevanter Code-Wert = 6
                  ; ist möglicherweise der Text im Input-Control der ComboBox geändert worden.
      MsgBox(0, "$CBN_EDITCHANGE", "--> hWndFrom:" & @TAB & $hWndFrom & _
                        @LF & "-->IDFrom:" & @TAB & $iIDFrom & @LF & _
                                          "-->Code:" & @TAB & $iCode)
EndIf
```

Mit

```
EndIf
Return $GUI_RUNDEFMSG           ; ist für die Weitergabe von Windows-Nachrichten notwendig
;
EndFunc
```

wird die Funktion abgeschlossen.
Immer, wenn die Windows-Nachricht $CBN_EDITCHANGE auftritt, könnte somit anstelle der MsgBox eine wirkliche Aktion durchgeführt werden, z.B. mit

```
$selEd = _GUICtrlComboBox_GetEditText($Combo)
```

der aktuelle String im Eingabe-Fenster der `ComboBox` ausgelesen werden, welcher dann mit

```
_GUICtrlComboBox_AddString($Combo, $selEd)
```

in das Daten-Fenster der `ComboBox` eingefügt werden kann.
Die Basisfunktion

```
GUICtrlSetData ($Combo, $selEd)
```

ist hier nicht anwendbar, da die `ComboBox` mit der UDF erstellt wurde.

Diese einfache Codesequenz hat jedoch zur Folge, dass mit jeder Änderung im Edit-Fenster eine neue Daten-Zeile in die `ComboBox` eingefügt würde, was jedoch eigentlich nicht der Aufgabe entspricht.
Die Strategie zur Verwendung der Windows-Nachricht `$CBN_EDITCHANGE` muss deshalb erweitert werden. Dazu ist dann allerdings noch etwas mehr Funktionalität zu programmieren.
Beim erstmaligen Auftreten der Nachricht ist zunächst die sofortige Einfügung richtig.
Beim zweiten Mal und den folgenden muss jedoch vorher die bereits bestehende Daten-Zeile (die letzte) gelöscht werden, bevor die neue Einfügung vorgenommen werden kann. Um zu unterscheiden zwischen der ersten und weiteren Nachrichten wird der „Merker“ `$mk` eingeführt, welcher vor den ersten Mal noch den Wert `False` hat und danach auf `True` gesetzt wird. Die Rücksetzung erfolgt dann erst, wenn die Code-Sequenz mit dem Click auf die neue Daten-Zeile abgeschlossen wird.

```
$selEd = _GUICtrlComboBox_GetEditText ($Combo)
If $mk = False Then
      _GUICtrlComboBox_AddString ($Combo, $selEd)     ; fügt eine neue Datenzeile
                                                      ; mit dem Inhalt $selEd ein
      $mk = True
      ;
Else
      $count = _GUICtrlComboBox_GetCount($Combo)
      _GUICtrlComboBox_DeleteString($Combo, $count - 1)     ; löscht die letzte
                                                            ; Daten-Zeile
      _GUICtrlComboBox_AddString ($Combo, $selEd)     ; fügt eine neue Datenzeile
                                                      ; mit dem Inhalt $selEd ein
      ; ….
```

```
EndIf
```

Da die `DeleteString`–Funktion von „0“ an zählt, aber `GetCount` mit „1“ beginnt, musste das entsprechend berücksichtigt werden.
Als sehr unvorteilhaft ergibt sich nun allerdings, dass nach der Rückkehr aus dem WM-Handler die Eingabe im Edit-Fenster der `ComboBox` vollständig markiert ist, so dass jedes Mal der Cursor von Hand ganz nach rechts gesetzt werden muss.
Leider lässt sich das nicht beeinflussen, da über den Selektion-Status des Edit-Fensters der `ComboBox` keinerlei Windows-Nachrichten beobachtet werden konnten, d.h. auch bei einer händischen Manipulation des Cursors im Edit-Fenster werden leider keine (auswertbaren) Windows-Nachrichten erzeugt.

Resümierend kann festgestellt werden, dass der Aufwand und die daraus resultierende Funktionalität eigentlich keine besonderen Vorteile im Vergleich mit der Button-Lösung bringt.

ListBox – Datenein- und Ausgabe

`ListBox`en sollten sich eigentlich gut zur Darstellung von Tabellen eignen – so jedenfalls der erste Ideenansatz. Die Alternative zur Verwendung einer mehrzeiligen `EditBox` mit direkter Eingabe hat den Nachteil, daß der Anwender mit jeder Eingabe die Struktur der Tabelle verändern kann.

Eine `ListBox` wird mit

```
$List1 = GUICtrlCreateList("text", [Left], [Top], [width], [height],
[style], [exStyle])
```

erstellt, wobei lediglich `[Left], [Top], [width], [height]` von existenzieller Bedeutung sind und `[style], [exStyle]` als optionale Parameter gelten.
Der Angabe `"text"` dient einer Vorbelegung des `ListBox`-Inhaltes bei deren Erstellung.
Sollen Tabellen-ähnliche Einträge entstehen, so ist man geneigt den Tabulator mit dem "|"-Zeichen zu gestalten. Da dieses Zeichen jedoch von der `ListBox` intern als `CRLF` interpretiert wird, kommt keine Tabellenform zustande.
Zur Gestaltung von Tabellen in einer `ListBox` sind deshalb zwei wichtige Voraussetzungen zu beachten:

- Verwendung einer breitenkonstanten Schriftform, z.B. `Courier`,
- Ergänzung der Spalten-Einträge mit einer berechnete Anzahl von Leerzeichen, damit die nächste Spalte immer an der gleichen Stelle beginnt.

```
GUICtrlSetFont(-1, 10, 400, 0, "Courier")        ; setzt die Schriftart,
                                                 ; (weiters in der AutoIt-Hilfe)

Const $Inp1Len = 15                              ; max. Länge + Abstand des Eingabestrings
$inp1 = GUICtrlRead ($Input1)                    ; aktuelle Werte aus z.B. InputBoxen lesen
$lenInp1 = StringLen($inp1)                      ; Ermittlung der Stringlänge
      For $n = 0 To $Inp1Len - $lenInp1          ; Vorgabe - aktuelle Stringlänge
            $fuellStr = $fuellStr & " "          ; Auffüllen mit Leerzeichen
      Next
```

In `$fuellStr` ist dann ein sog. „Füllstring“ aus Leerzeichen enthalten, der an den eigentlichen Eingabe-String angehängt wird, um eine einheitliche „Startposition“ für die nächste Spalte der

Tabelle zu erhalten. Dieses Verfahren ist für alle, bis auf den letzten Spalteneintrag durchzuführen.
Sind alle Spalteneinträge incl. ihrer Füllstrings zu einem String in `$str` zusammengesetzt, so kann mit:

```
GUICtrlSetData ($List1, $str)          ; $str ergibt jeweils eine neue Zeile
```

eine neue Zeile in die `ListBox` eingefügt werden. Leider erfolgt die Einfügung nicht in chronologischer Reihenfolge, sondern es findet immer eine alphanumerische Sortierung im Zusammenhang mit den bereits bestehenden Zeilen statt.
Inwiefern das durch spezielle `Style`-Einstellungen vermieden werden kann, ist noch offen. Auch ist der Einfluß einer vorhergehenden Zeilen-Selektierung mit `ControlCommand` auf den Ort der Einfügung (noch) nicht untersucht worden.

Sollen hingegen Daten aus der `ListBox` ausgelesen werden, so bedarf es dazu einiger Vorbereitungen.
Zwar könnte mit

```
$temp = GUICtrlRead ($List1)
```

eine Zeile ausgelesen werden, aber dazu muss diese vorher selektiert (ausgewählt) werden, sonst ergibt sich ein sog. „Leerstring“.
Die Auswahl einer Zeile in der `ListBox` erfolgt programmtechnisch mit:

```
ControlCommand ("GUI-Name", "", $List1, "SetCurrentSelection", $n )
```

wobei `$List1` die ID-Variable der `ListBox` ist und `SetCurrentSelection` der Befehl lt. Liste in der AutoIt-Hilfe (dort sind auch weitere Angaben zu `ControlCommand`). Mit der Variablen `$n` wird schließlich die gewünschte Zeilennummer, beginnend bei „0“ eingetragen.
Damit ergibt sich aber eine weitere Schwierigkeit bei der Handhabung der `ListBox`. Es ist leider nicht bekannt, wie die aktuell vorhandene Zeilenzahl in einer ListBox ermittelt werden kann. Mit

```
$n = ControlCommand ( "Form1", "", $List1, "GetLineCount", "" )
```

funktioniert das leider nicht, da das offensichtlich nur für ein mehrzeiliges `EditControl` zutrifft.
Um aber `ControlCommand` und `GUICtrlRead` in einer Do … Until-Schleife einsetzen zu können, müsste ein Abbruch-Kriterium vorhanden sein und das wäre eben die Zeilenanzahl. Wenn man versuchsweise für `$n` in `ControlCommand` immer größere Werte einsetzt, bis die aktuelle Zeilenzahl überschritten ist, wird als `GUICtrlRead` -Ergebnis ein Leerstring erzeugt. Damit wäre das leicht auswertbar.
Wird dieses zur Laufzeit programmtechnisch vorgenommen, wird jedoch nach dem Überschreiten der aktuellen Zeilenzahl immer wieder nur die letzte Zeile ausgegeben. Da es sehr unwahrscheinlich ist, dass zwei Zeilen den gleichen Inhalt haben, könnte man auch das als Abbruchkriterium auswerten. Ganz korrekt ist das aber nicht!

Für eine Anwendung zum Speichern des `ListBox`-Inhalts wird die `Do … Until`-Schleife wie folgt verwendet:

```
Local $n = 0, $temp, $tempG, $var
$NP = Run("notepad.exe")                    ; Notepad starten
      WinWait("[CLASS:Notepad]")            ; warten bis Notepad gestartet
      ControlCommand("GUI-Name", "", $List1, "SetCurrentSelection", $n)
                                            ; Cursor auf 1.Zeile setzen ("0"), List1
Do
      $temp = GUICtrlRead ($List1)                ;
      $tempG = $var & @CRLF & $temp         ; neuen String zusammensetzen
      ControlSetText("[CLASS:Notepad]", "", "Edit1", $tempG)
      $n = $n + 1                           ; Cursor-Variable erhöhen
      ControlCommand("GUI-Name", "", $List1, "SetCurrentSelection", $n)
                                            ; Cursor auf nächste Zeile setzen ($n+1)
Until $temp = GUICtrlRead ($List1)          ; wenn $temp(n) = $temp(n+1) dann beenden
```

(Ende)