sGUI v0.8.4

Einleitung

Vielen Dank allen aus dem Forum die fleißig genervt,getestet und Tipps gegeben haben :)

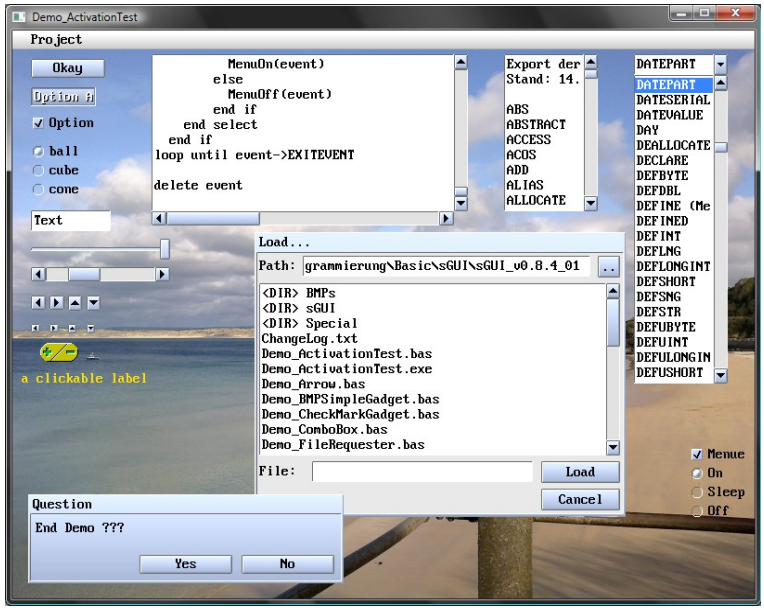
sGUI ist eine GUI Simulation geschrieben in FreeBASIC. Mit einer Menge LINE und PSET Befehlen und einem „Eventloop“ wird hier versucht, dass Verhalten echter GUIs rudimentär nachzubilden. Wie gesagt, nur ein Versuch, wenn man mal auf die Schnelle ein paar Buttons für sein Programm braucht. Für „professionelle“ Sachen sollte man sich denn doch mit entsprechenden Bibliotheken auseinandersetzen.
Etwas Wissen über Referenzierung und Pointer (Zeiger) wäre nicht schlecht, ist aber nicht zwingend erforderlich. Ich bin in dieser Thematik auch nicht allzu sattelfest. :-D

Ich werfe hier oft die Begriffe Controls und Gadgets durcheinander, ich bitte dies zu entschuldigen. In beiden Fällen ist das gleiche gemeint.

Mit dieser neuen Version ist endlich mal auch wieder ein neues Control hinzugekommen, das Mausrad hat Einzug gehalten.
Einige Bugs die sich so im Laufe der Zeit gezeigt haben, sind entfernt worden. Die neuen Bugs sind natürlich Features ;).
Viele interne Bezeichner haben sich geändert, der gesamte Code ist neu aufgeteilt worden, ein NAMESPACE wird jetzt benutzt.

Welche Controls werden in sGUI simuliert:

- *PullDownMenü
- *SimpleGadget
- *ToggleGadget
- *CheckMarkGadget
- *RadioButton
- *StringGadget
- *TrackBar
- *Scrollbar
- *Arrows
- *ListBox
- *MultiLineEdit/TextViewBox
- *BMPSimpleGadget
- *MessageBox
- *ComboBox



Lizenz

WTFPL... macht damit was ihr wollt

***
---
bekannte Probleme und Tipps
---

*INKEY liefert keine Ergebnisse:

Es nicht möglich, innerhalb des EventLoops die INKEY-Funktion zu nutzen. Dies liegt schlichtweg einfach daran daß diese Funktion schon von sGUI aufgerufen wurde und nun der Tastatur-Puffer leer ist.
Informationen über gedrückte Tasten sind dann „nur“ noch über den EventHandle möglich!
(siehe weiter unten im Abschnitt EventHandle ->KEY)

*sGUI ist zu langsam / reagiert nicht:

sGUI läuft nicht in einem extra Thread. Sämtlicher Code bezüglich Ereignisse,Control-Handling und GrafikRefresh erfolgt letztlich über die Methode ->xSLEEP im EventLoop. Wer sich in einer selbst geschriebenen Prozedur befindet, die ellenlange Berechnungen durchführt und nicht ab und zu mal im EventLoop „vorbeischaut“, wird zumindest dann an der Benutzbarkeit von sGUI zweifeln. Dies ist ja schließlich kein Profiwerkzeug :-P.

*Hintergrundbild:

Grundsätzlich ist es möglich einen Hintergrundbild zu benutzen. Dabei ist zu bemerken, daß alle Controls den von ihnen bedeckten Hintergrund zwischenspeichern. Dies geschieht allerdings nur einmal zum Zeitpunkt ihr Erzeugung. Es sollte also der Hintergrund schon festgelegt sein, bevor die Controls mittels ihrer Add...()-Prozedur erzeugt werden!

***

---
ein kleines Beispiel

Wie funktioniert das Ganze?
Nun, zuerst benötigt man einen Screen mit einer Tiefe von mindestens 16 bpp. Drunter geht nichts.
Hier ein kleiner Beispiel-Code, anschließend dazu ein paar Erklärungen.

[code]
```
#include “sGUI\sGUI.bas“               'diese  Include ist natürlich ein Muss
#include once “sGUI\SimpleGadget.bas“  'für ein SimpleGadget wird diese .bas benötigt
using sGUI

screenres 200,200,32,,&H04
InitGFX                               'immer (direkt) nach Screen (und Width wenn benutzt) aufrufen

dim as EventHandle ptr event          'Dimensionierung einer EventHandle Variable
event=new EventHandle                 'Erzeugen eines EventHandles

dim as Gadget ptr button              'Dimensionierung einer Gadget Variable
button=AddSimpleGadget(event,20,20,140,20,“Klick mich“)'Erzeugen eines SimpleGadgets
GadgetOn (button)                     '“Einschalten“ des Gadgets (sichtbar- und benutzbar
                                      'machen)

do                                    'Beginn des Eventloops
  event->xSleep(1)                    'Methode des EventHandles, wartet auf ein Ereignis

 if event->GADGETMESSAGE then         'Abfrage ob eine GADGETMESSAGE anliegt wenn ja dann...
   select case event->GADGETMESSAGE   'Select-Case-Block für die Auswertung der GADGETMESSAGE
     case Button                      'wurde unser Button gedrückt....
       beep                           'dann piepsen wir mal. Man könnte hier auch wichtigere Sachen machen... ;).
   end select
 end if
loop until event->EXITEVENT           'EXITEVENT von xSLEEP beim Klicken auf den CloseButton gesetzt

delete event                          'Löscht EventHandle und alle mit ihm verbundenen Gadgets
```
[/code]

Eigentlich schon kommentiert, ein paar Sätze mehr sind vielleicht hilfreich.
Nachdem die Grafik initiiert wurde, muß zunächst eine „Kontrolleinheit“ erzeugt werden, die ich mal großzügig als Eventhandle bezeichne(event=new EventHandle). Dieser „Eventhandle“ verwaltet und überprüft alle ihm zugewiesenen Controls auf Eingabe des Users.
Mit AddSimpleGadget() wird ein einfacher Button erzeugt und (siehe 1. Parameter) dem Eventhandle bekannt gemacht. Dann erfolgt das Einschalten des Controls.
Um permanent Abzufragen ob der User etwas bedient hat, gibt es die Ereignis-Schleife (Eventloop).
Mit der Methode ->xSleep() wird festgestellt, ob der User eine Eingabe getätigt hat. Das kann das Bedienen eines Controls, Menüeintrages als auch „nur“ eine Mausbewegung oder ein Tastendruck sein. Sollte ein Ereignis eingetreten sein, können wir nun darauf entsprechend reagieren.
In diesem Fall fragen wir ab, ob ein Control bedient wurde (if event->GADGETMESSAGE then...).
Sollte ein Solches eingetreten sein, haken wir etwas näher nach:
„->GADGETMESSAGE“enthält nämlich d i e Information, w e l c h e s Control bedient wurde.
Was für eine Aktion nun beim jeweiligen Control geschehen soll, wird dann im SELECT CASE Block realisiert. Ein Klick auf den Close-Button beendet den Loop, bzw. nach Aufräumarbeiten auch das Programm.

Das war es eigentlich schon.

***

Namespace

Die komplette GUI ist jetzt in einem Namespace namens „sGUI“ eingebettet. Somit ist allen Variablen und Routinen entweder ein „sGUI.“ voranzustellen oder aber ein „using sGUI“ direkt nach allen sGUI-Includes einzufügen.(Mit Dank an nemored für die Idee und den Kollisionshinweis).
Alles über die Funktionsweise von Namespaces ist in der FreeBasic-Referenz zu finden (siehe unter NAMESPACE und USING).

***

SystemEvents

Im Gegensatz zu allen Vorgängerversionen sind alle erfassten Systemereignisse nun in globalen Variablen hinterlegt:

KEY	enthält das (aufbereitete) Ergebnis der INKEY Funktion (Länge 1 Zeichen)
ASCCODE	enthält den ASCII-Code von KEY
EXTENDED	ist 255 wenn INKEY einen 2-Zeichen-langen String zurück gab
MOUSEX	x-Position der Maus (via GETMOUSE)
MOUSEY	y-Position der Maus (via GETMOUSE)
LMB	Status der linken Maustaste (via GETMOUSE)
MMB	Status der mittleren Maustaste (via GETMOUSE)
RMB	Status der rechten Maustaste (via GETMOUSE)
WHEEL	ist -1 wenn das Mausrad einen Tick vom Benutzer weg bewegt wurde, 1 wenn zu ihm hin
CLOSEBUTTON	wenn 1 wurde der CloseButton des Screen gedrückt

zu den Maustasten:
Normalerweise unterscheidet Freebasic „nur“ zwischen gedrückt und nicht gedrückt, also zwei Zustände.
In sGUI mußte ich dies etwas erweitern. Es werden (bei allen drei Tasten) nun vier Zustände erfaßt:

HIT	die Taste ist soeben gedrückt worden, wenn man so will der Vorgang des Drückens... die Taste senkt sich
HOLD	die Taste ist gedrückt und wird gehalten
RELEASE	die Taste ist soeben losgelassen worden, Vorgang des Loslassens... Taste hebt sich
RELEASED	die Taste ist losgelassen...

Mit dem mitgelieferten Beispielprogramm „Demo_LMB_Status.bas“ kann man dieses Statuswechsel mal mit der linken Maustaste ausprobieren.

Zum Mausrad:
siehe „Demo_Wheel.bas“

***

GFX

Beschreibung:
sGUI benötigt ein paar grafische Voreinstellung. Werden diese nicht getätigt,
kommt es zu Fehldarstellungen der Controls. InitGFX übernimmt dies und muß gleich
nach dem SCREEN-Befehl aufgerufen werden.

Routinen:
InitGFX()

***
sub InitGFX (clearscreen as integer=1)

Beschreibung:

Parameter:
clearscreen   (optional) wenn 1 (voreingestellt) dann wird ein cls ausgeführt, mit 0 wird dies unterdrückt

Rückgabewert:
keiner

Hinweise:
keine

***

EventHandle

Beschreibung:
Ein existierender Eventhandle mit seinen Methoden stellt das Herzstück des Ganzen dar.
Das Erzeugen ist relativ einfach:
[code]
dim as EventHandle ptr event 'Dimensionierung einer EventHandle Variable
event=new EventHandle   'Erzeugen eines EventHandles
[/code]

Das Löschen zum Schlußdes Programms erfolgt so:
[code]
DELETE event   'Löschen eines EventHandles
[/code]

Die bisherigen Routinen dafür sind entfallen.

Da es sich um eine Pointer(Zeiger)-Variable handelt, sind alle Methoden und Informationen
referenziert aufzurufen/abzufragen. Wenn die EventHandle (PTR) Variable z.B. wie oben "event" heißt,
so muß der Aufruf von "xSLEEP" so erfolgen "event->xSLEEP". Über die Methode "->xSLEEP" kontrollieren
wir die Controls ;-) .Das können Buttons oder das PullDownMenü sein.
Desweiteren merken uns wichtige Systemereignisse wie MausPosition, Zustand der MausTasten, Tastatureingaben.

Wichtig!!! Ohne den Aufruf von "->xSLEEP" innerhalb einer Schleifen-Konstruktion funktioniert sGUI nicht.

Welche Informationen stellt der EventHandle noch zur Verfügung:
->GADGETMESSAGE   Gadgetnummer (genauer ein Gadget Pointer) des Gadgets, welchesgrad eine Aktion meldet
->MENUMESSAGE   ist >0 (enthält dann die integer gecastete Adresse des Eintrages) wenn ein Menüpunkt angewählt wurde
->MENUNUMBER   Menünummer in dem sich der angewählte Menüeintrag befindet
->ITEMNUMBER   Eintragnummer des angewählten Menüeintrages
->EXITEVENT   Abbruchsignal beim Klicken auf CloseButton

Routinen:
->xSleep()

***
sub xSleep (eventmode as integer,XButton as integer=1)

Beschreibung:

Parameter:
eventmode   1= kein SLEEP 1, 0= mit SLEEP 1, 1 mit SLEEP Programm wartet in xSLEEP bis ein Ereignis auftritt
XButton   (optional) wenn 1(voreingestellt) dann wird die Abfrage des Closebuttons unterstützt

Rückgabewert:
keiner

Hinweise:
keine

***

Activation

Beschreibung:
Dieser Modus beschreibt die Sichtbar- und Benutzbarkeit eines Controls. Es werden dabei 3 Zustände
unterschieden: OFF-> ein Control ist nicht sichtbar und auch nicht benutzbar. Dies ist auch der Zustand nach
der Erzeugung durch einen der ADD... Befehle. ON-> das Control ist sichbar und benutzbar. SLEEP-> das Control
ist zwar sichbar (etwas blasser) aber nicht benutzbar.

Routinen:
GadgetOn(), GadgetOff(), GadgetSleep(), RestoreActivation()

***
sub GadgetOn overload (gad as _Gadget ptr)
sub GadgetOn (gadblockstart as _Gadget ptr, gadblockend as _Gadget ptr)

Beschreibung:
Schaltet die Aktivierung eines Controls in den ON Modus, d.h. es ist sichtbar und benutzbar.
Mit der überladenen Variante ist es möglich, einen kompletten Bereich der vom Eventhandle verwalteten
Controlliste in den entsprechenden Aktivierungsmodus zu versetzen.

Parameter:
gad   Zeiger auf ein Control
gadblockstart   Zeiger auf ein Control

```
gadblockend         Zeiger auf ein Control


Rückgabewert:
keiner

Hinweise:
keine
```

***

```
sub GadgetSleep overload (gad as _Gadget ptr)
sub GadgetSleep (gadblockstart as _Gadget ptr, gadblockend as _Gadget ptr)

Beschreibung:
Schaltet die Aktivierung eines Controls in den SLEEP Modus, d.h. es ist sichtbar aber nicht benutzbar.
Mit der überladenen Variante ist es möglich, einen kompletten Bereich der vom Eventhandle verwalteten
Controlliste in den entsprechenden Aktivierungsmodus zu versetzen.

Parameter:
gad                 Zeiger auf ein Control
gadblockstart       Zeiger auf ein Control
gadblockend         Zeiger auf ein Control

Rückgabewert:
keiner

Hinweise:
keine
```

***

```
sub GadgetOff   overload (gad as _Gadget ptr)
sub GadgetOff (gadblockstart as _Gadget ptr, gadblockend as _Gadget ptr)

Beschreibung:
Schaltet die Aktivierung eines Controls in den OFF Modus, d.h. es ist unsichtbar und nicht benutzbar.
Mit der überladenen Variante ist es möglich, einen kompletten Bereich der vom Eventhandle verwalteten
Controlliste in den entsprechenden Aktivierungsmodus zu versetzen.

Parameter:
gad                 Zeiger auf ein Control
gadblockstart       Zeiger auf ein Control
gadblockend         Zeiger auf ein Control

Rückgabewert:
keiner

Hinweise:
keine
```

***

```
sub RestoreActivation overload (gad as _Gadget ptr)
sub RestoreActivation (gadblockstart as _Gadget ptr, gadblockend as _Gadget ptr)

Beschreibung:
Ruft den Aktivierungsmodus vor der letzten Änderung auf. Also ein UNDO der letzten Aktivierungsänderung.
Mit der überladenen Variante ist es möglich, einen kompletten Bereich der vom Eventhandle verwalteten
Controlliste in den entsprechenden Aktivierungsmodus zurück zuversetzen.

Parameter:
gad                 Zeiger auf ein Control
gadblockstart       Zeiger auf ein Control
gadblockend         Zeiger auf ein Control

Rückgabewert:
keiner

Hinweise:
keine
```

***

```
-------------------------------------------------------------------------------------
Selection
-------------------------------------------------------------------------------------
Beschreibung:
Alle Controls können unterscheiden zwischen selektiertem oder nicht selektiertem Zustand. Aber nicht bei
allen ist dieses "Feature" sinvoll und wird auch benutzt. Ein normaler Button zB. wechselt während des Klickens
darauf nur kurz in den selektierten Zustand, andere zB. CheckMarkGadget verharren in ihrer Selektion
bis zum nächsten Klick. Einem Label sind wiederum beide Zustände völlig egal.

Routinen:
GetSelect(), SetSelect()
```

***

```
function GetSelect (gad as _Gadget ptr) as integer

Beschreibung:
Liefert den Selektionszustand eines Controls.

Parameter:
gad                 Zeiger auf ein Control

Rückgabewert:
1= Control ist selektiert, 0= Control ist nicht selektiert

Hinweise:
keine
```

***

```
sub SetSelect (gad as _Gadget ptr,Selection as integer)

Beschreibung:
Setzt den Selektierungszustand eines Controls
```

Parameter:
gad   Zeiger auf ein Control
Selection   0=nicht sektiert, 1=selektiert

Rückgabewert:
keiner

Hinweise:
Nur sinnvoll bei Controls die toggeln können. zB. Togglegadget, CheckMarkGadget, RadioButton

***

Draw & Update

Beschreibung:
Zum einen kann eine zerstörte Grafik, wie auch immer dies geschehen kann, wieder hergestellt werden.
Zum anderen kann es erforderlich sein, die "innere Mathematik" mancher Controls nach bestimmten Veränderungen
wieder auf den aktuellsten Stand zu bringen.

Routinen:
DrawGadget(),UpdateGadget()

***
sub DrawGadget(gad as _Gadget ptr)

Beschreibung:
Zeichnet ein Control neu auf den SCREEN

Parameter:
gad   Zeiger auf ein Control

Rückgabewert:
keiner

Hinweise:
keine

***
sub UpdateGadget(gad as _Gadget ptr)

Beschreibung:
Führt ein Update eines Controls aus

Parameter:
gad   Zeiger auf ein Control

Rückgabewert:
keiner

Hinweise:
keine

***

Rename

Beschreibung:
Ändert die Beschriftung einiger Controls.

Routinen:
SetCaption()

***
sub SetCaption(gad as _Gadget ptr,Text as string)

Beschreibung:
Ändert die Beschriftung einiger Controls.

Parameter:
gad   Zeiger auf ein Control
text   neuer Text

Rückgabewert:
keiner

Hinweise:
Bei folgenden Controls sinnvoll einzusetzen: SimpleGadget, ToggleGadget, RadioButton, CheckMarkGadget, Label.
Auf andere Controls hat dies keine Auswirkung.

***

Umlaute

Beschreibung:
Getestet nur unter Windows(Vista).Welche Codepages oder ISO-Normen bezüglich der Zeichenkodierung wann,
wo und von wem benutzt werden, keine Ahnung! Wie es aussieht arbeiten hier Betriebssystem, die IDEs
(ich benutze einen einfachen Editor) und das compilierte Programm (was ich hier mal als "in FreeBasic" bezeichne)
jeweils mit anderen Zeichenkodierungen bzw Normen.... und das führt leider zu Problemen.
Es geht mir hier auch nur um die Beschreibung des Problems und (m)einen Versuch es zu lösen.
Inwieweit das ganze unter Linux notwendig ist und ob es dort überhaupt funktioniert,
entzieht sich ebenfalls meiner Kenntnis.

Zum Problem:
Werden in einem Freebasic-Quellcode einer String-Variablen Umlaute oder
das "Eszett" zugewiesen (zB einer Stringvariablen: a="äöüÄÖÜß"), so kommt es zur
Falschdarstellung dieser Zeichen. Das gleiche Problem tritt auch auf wenn ein Programm eine
beliebige Text-Datei öffnet um deren Inhalt in Freebasic anzuzeigen.Auch hier gibt es die gleichen
Probleme.Mit Uimport() lässt sich ein String so "umformen" das String-Variablen mit Umlauten in Freebasic
richtig dargestellt werden.Werden zur Laufzeit eines Programms String-Variablen über INKEY
oder INPUT Umlaute(o. Eszett) zugewiesen, so werden diese in FreeBasic korrekt dargestellt.

Das UmkehrProblem:
Jetzt hat man nun Text mit Umlauten den man wunderbar in FreeBasic anzeigen kann...
Soweit so gut...Werden diese Strings jedoch in eine Textdatei geschrieben, um sie in anderen
Programmem, zB. Editoren oder Textverarbeitungen, weiter zu nutzen so werden diese
Umlaute d o r t wiederum falsch dargestellt. Mit UExport() lässt sich ein String mit Umlauten wieder
zurück in ein für die "Aussenwelt" lesbares Format umwandeln.

Kleiner Tipp nebenbei:
Wer mit CHDIR in ein existierendes Verzeichnis wechseln will, das Umlaute im Namen hat,
wird Probleme bekommen. Den Pfadnamen mit UExport behandeln, kann da helfen, ein CHDIR "Frühstück"
wird nicht klappen, ein CHDIR UExport("Frühstück") schon.

Letztlich ist dies hier Brechstangen-Prinzip, es sollte sich wohl überlegt werden, ob in wo man dies
hier einsetzt

Routinen:
UImport(), UExport()

***
function UImport(s as string) as string

Beschreibung:
Importiert Text so, daß die Umlaute im FBScreen angezeigt werden

Parameter:
s   ein beliebiger String
Rückgabewert:
keiner

Hinweise:
keine

***
function UExport(s as string) as string

Beschreibung:
Exportiert Text so, daß die Umlaute außerhalb von FB angezeigt werden

Parameter:
s   ein beliebiger String
Rückgabewert:
string

Hinweise:
keine

***

Controls

Beschreibung:
Hier nun die Beschreibung der Controls und deren "Spezialprozeduren", mit denen das Control
abgefragt oder auch manipuliert werden kann.
Alle Controls basieren auf dem gleichen UDT(mit Ausnahme des Menüsystems), was bedeutet,
daß die oben erwähnten "Spezialprozeduren" auch auf Controls anwendbar sind, für die sie
nicht gedacht sind! Davon ist einfach mal abzuraten.

Um überhaupt Controls erzeugen zu können, sind eine Variable als "GADGET ptr" und ein
existierender EventHandle erforderlich. Denn zu jeder Control-Art existiert eine
spezielle "Add.....()" Prozedur die das Control erzeugt. Deren erster Parameter ist immer
ein Zeiger auf einen EventHandle, dem das Control dann zugeordnet wird und dort als Liste verwaltet wird

[code]
```
dim as EventHandle ptr event  'Dimensionierung einer EventHandle Variable
event=NEW EventHandle         'Erzeugen eines EventHandles

dim as Gadget ptr button      'Dimensionierung einer Gadget Variable
button=AddSimpleGadget(event,20,20,140,20,"Klick mich")'Erzeugen eines SimpleGadgets
```
[/code]

Ping:
Controls "melden sich", sobald sie korrekt bedient oder verändert wurden.
Jede Art auf seine eigene Weise. Ich nenne das mal einen "Ping" von sich geben.
Ist dies der Fall, dann wird die Variable ->GADGETMESSAGE des EventHandles ungleich
Null sein und uns den (Gadget)Pointer auf eben jenes Control liefern,welche hier für Unruhe gesorgt hat.

Routinen:
solang nichts anderes behauptet wird, alles was ab jetzt im Text folgt :)...

***

Arrow

Beschreibung:
Arrow und SmallArrow sind Controls die ein Pfeilsymbol für 4 mögliche Richtungen beinhalten.
Im Unterschied zum SimpleGadget fängt dieses Control bei HOLD nach einer Pause(ca. 0.8 sek.) an zu triggern.

Ping:
erfolgt bei HIT und HOLD.

Routinen:
AddArrow(), AddSmallArrow()

***
function AddArrow (event as EventHandle ptr,PosX as integer,PosY as integer,DirArrow as integer) as Gadget ptr
function AddSmallArrow (event as EventHandle ptr,PosX as integer,PosY as integer,DirArrow as integer) as Gadget ptr

Beschreibung:
Erzeugt einen Pfeilbutton

```
Parameter:
event               Zeiger auf einen EventHandle
PosX                Abstand vom linken Screenrand in px
PosY                Abstand vom oberen Screenrand in px
DirArrow            Richtung des Pfeilsymbols 0=links,1=rechts,2=hoch,3=runter

Rückgabewert:
Zeiger auf ein erzeugtes Control, bei Fehlschlag 0

Hinweise:
keine

***
------------------------------------------------------------------------------------------
BMPSimpleGadget
------------------------------------------------------------------------------------------
Beschreibung:
Dies ist eine einfacher Button aus Bitmaps aufgebaut,für klassische Aufgaben wie "OK", "Weiter" oder "Abbruch".

Ping:
RELEASE

Routinen:
AddBMPSimpleGadget(), bedingt auch SetCaption()

***
function AddBMPSimpleGadget (event as EventHandle ptr,PosX as integer,PosY as integer,FileUnselect as string,FileSelect as
string="",FileMask as string="") as Gadget ptr

Beschreibung:
Erzeugt einen einfachen Button aus mind. einer Bitmap.

Parameter:
event               Zeiger auf einen EventHandle
PosX                Abstand vom linken Screenrand in px
PosY                Abstand vom oberen Screenrand in px
FileUnselect        Bitmap die das Control im unselektierten Zutand zeigt
FileSelect          (optional)Bitmap die das Control im selektierten Zutand zeigt
FileMask            (optional)Bitmap die eine Maske des Control darstellt, um "formsensitives" Klicken zu realisieren

Rückgabewert:
Zeiger auf ein erzeugtes Control, bei Fehlschlag 0

Hinweise:
keine

***
------------------------------------------------------------------------------------------
CheckMarkGadget
------------------------------------------------------------------------------------------
Beschreibung:
Dies ist eine CheckBox, die nach Anklicken jeweils zwischen selektiert und nicht selektiert wechselt.
Sie hat eine feste Größe.

Ping:
RELEASE.

Routinen:
AddCheckMarkGadget(), SetSelect(), GetSelect(), SetCaption

***
function AddCheckMarkGadget (event as EventHandle ptr,PosX as integer,PosY as integer,SwtchSel as integer,Text as string="") as
Gadget ptr

Beschreibung:
Erzeugt ein CheckMarkGadget

Parameter:
event               Zeiger auf einen EventHandle
PosX                Abstand vom linken Screenrand in px
PosY                Abstand vom oberen Screenrand in px
SwtchSel            Selektierung bei Erzeugung 0/1
Text                Text rechts neben der Box

Rückgabewert:
Zeiger auf ein erzeugtes Control, bei Fehlschlag 0

Hinweise:
Mit definiertem Text ist der "sensible" Bereich des Controls um die Textbreite erweitert.
Auch das Klicken auf den Text löst ein Ping aus. Die Höhe des Controls ist abhängig von der
benutzten Fonthöhe und kann 12,14 oder 16 px betragen.

***
------------------------------------------------------------------------------------------
ComboBox
------------------------------------------------------------------------------------------
Beschreibung:
Einklappbare Options- u AuswahlBox. Durch den Klick auf das Pfeilsymbol wird der Selektor geöffnet.
In ihm kann man Scrollen und seine Auswahl treffen. Wenn sich der Mauszeiger über dem Control befindet, wird  das Mausrad
unterstützt.

Ping:
bei Auswahl eines neuen Eintrages im Selektor

Routinen:
AddComboBox(),SetCombolBoxVal(), GetComboBoxVal(),
die TO_Befehle, UpdateGadget()

***
function AddComboBox(event as EventHandle ptr,PosX as integer,PosY as integer,NumChars as integer, NumRows as integer=5) as Gadget
ptr
```

Beschreibung:
Erzeugt eine ComboBox

Parameter:
event Zeiger auf einen EnventHandle
PosX Abstand vom linken Screenrand in px
PosY Abstand vom oberen Screenrand in px
NumChars Breite der Box in Anzahl Zeichen, Faustformel für Breite in px= NumChars*8 + 6
NumRows (optional) Höhe der erscheinenden Selectbox in Anzahl Zeilen, voreingestellt sind 5 Zeilen

Rückgabewert:
Zeiger auf ein erzeugtes Control, bei Fehlschlag 0r

Hinweise:
benutzt ein TextObject für die Verwaltung der Einträge

***
sub SetComboBoxVal(gad as Gadget ptr,i as integer)

Beschreibung:
Setzt einen Eintrag, sodas er in der CB angezeigt wird

Parameter:
gad Zeiger auf ein Control
i Index(Zeilennummer des TOs in dem die Einträge verwaltet werden) des Eintrages >0

Rückgabewert:
keiner

Hinweise:
keine

***
function GetComboBoxVal(gad as Gadget ptr) as integer
Beschreibung:
Gibt den Index des selektierten Eintrag aus

Parameter:
gad Zeiger auf ein Control

Rückgabewert:
Index selektierter Eintrag, 0 wenn keine Selektion vorhanden

Hinweise:
keine

***

FileRequester

Beschreibung:
Eine kleine Nachbildung eines OpenFileDialoges.
Der FileRequester ist im Grunde ein in einer Funktion gekapselter EventHandle mit eigenen Controls und einem eigenen EventLoop.
Die Bedienung sollte selbst erklärend sein. Wenn sich der Mauszeiger über dem Control befindet, wird das Mausrad unterstützt.

Ping:
keiner

Routinen:
FileRequester()

***
function FileRequester(PosX as integer,PosY as integer,BtnText as string,Path as string, File as string="",FExistMode as integer=0)
as string

Beschreibung:
Öffnet einen "OpenFileDialog"

Parameter:
PosX Abstand vom linken Screenrand in px
PosY Abstand vom oberen Screenrand in px
BtnText Text des "Aktionsbuttons" und der Kopfzeile des Fakewindows
Path (optional) selbstdefinierter Pfad
File (optional) selbstdefinierte Datei
FExistMode (optional) wenn 1 dann wird Pfad+Datei auf Gültigkeit überprüft

Rückgabewert:
wenn erfolgreich ein STRING der den Pfad+Dateinamen enthält, bei Fehlschlag einen Leerstring

Hinweise:
keine

***

Label

Beschreibung:
Anklickbarer Text

Ping:
RELEASE

Routinen:
AddLabel(), SetCaption(), SetLabelColor()

***
function AddLabel (event as EventHandle ptr,PosX as integer,PosY as integer,Text as string,TextColor as integer=TextColor) as Gadget
ptr

Beschreibung:
Erzeugt ein Label

```
Parameter:
event               Zeiger auf einen EventHandle
PosX                Abstand vom linken Screenrand in px
PosY                Abstand vom oberen Screenrand in px
Text                Text des Labels
TextColor Farbe des Textes

Rückgabewert:
Zeiger auf ein erzeugtes Control, bei Fehlschlag 0

Hinweise:
keine
```

***

```
sub SetLabelColor (gad as Gadget ptr,TextColor as integer)

Beschreibung:
ändert die Textfarbe eines Labels

Parameter:
gad                 Zeiger auf ein Control
TextColor neuer Farbwert, auch das RGB() Makro ist hierfür benutzbar

Rückgabewert:
keiner

Hinweise:
keine
```

***

```
-------------------------------------------------------------------------------------------
ListBox
-------------------------------------------------------------------------------------------
Beschreibung:
Stellt eine Art Auswahlbox dar. Im Unterschied zur ComboBox ist diese hier nicht "einklappbar". Wenn sich der Mauszeiger über dem
Control befindet, wird  das Mausrad unterstützt.

Ping:
HIT

Routinen:
AddListBox(), GetListBoxVal(), SetListBoxVal
alle TO_Befehle, UpdateGadget()
```

***

```
function AddListBox (event as EventHandle ptr,PosX as integer,PosY as integer,NumChars as integer,NumRows as integer,DisplayMode as
integer=0,ScrollBarMode as integer=0) as Gadget ptr

Beschreibung:
Erzeugt eine ListBox

Parameter:
event               Zeiger auf einen EventHandle
PosX                Abstand vom linken Screenrand in px
PosY                Abstand vom oberen Screenrand in px
NumChars            Breite der Box in Anzahl Zeichen, Faustformel für Breite in px= NumChars*8 + 6
NumRows             Höhe der Box in Anzahl Zeilen
DisplayMode         (optional)Darstellungsvariante, wenn 1 ist es möglich Labels und Einträge in der Box zu erzeugen
                    Dies erfordert aber einen speziellen Syntax der Einträge im Textobjekt (siehe Hinweise)

Rückgabewert:
Zeiger auf ein erzeugtes Control, bei Fehlschlag 0

Hinweise:
Syntax der Einträge für den DisplayMode 1: Label "LBL:Labelname", Item "ITM:123:Itemname".
Ein Label beginnt immer mit "LBL:" und anschließend ein Name des Labels.
Ein Item beginnt immer mit "ITM:" gefolgt von einer dreistelligen ID mit Doppelpunkt und abschließend ein Name.
Die dreistellige ID ist dann das "Erkennungsmerkmal" des Items wenn es ausgewählt wurde bzw. durch das Programm im
Verlauf gesetzt wird.
```

***

```
function GetListBoxVal(gad as Gadget ptr) as integer

Beschreibung:
Liefert den Index des selektierten Eintrages bzw. ItemID wenn DisplayMode 1 sein sollte

Parameter:
gad                 Zeiger auf ein Control

Rückgabewert:
Index des selektierten Eintrages/ItemID

Hinweise:
keine
```

***

```
sub SetListBoxVal(gad as Gadget ptr,i as integer)

Beschreibung:
setzt einen bestimmten Eintrag als selektiert und scrollt zu diesem hin

Parameter:
gad                 Zeiger auf ein Control
i                   Index des Eintrages, ItemID

Rückgabewert:
keiner

Hinweise:
keine
```

***

```
--------------------------------------------------------------------------------
MessageBox
--------------------------------------------------------------------------------
Beschreibung:
Dies ist eine Nachbildung einer MessageBox. Sie ist, wie der FileRequester,
ein in einer Funktion verpackter eigener EventLoop mit eigenen Controls.

Ping:
keiner

Routinen:
MessageBox()

***
function MessageBox(PosX as integer,PosY as integer,MsgText as string="",TitleText as string="",MsgType as integer=1) as integer

Beschreibung:
Zeigt eine MessageBox

Parameter:
PosX                Abstand vom linken Screenrand in px
PosY                Abstand vom oberen Screenrand in px
MsgText             Wichtige Botschaft für den User, mehrere Zeilen sind diese im String durch "|" zu trennen
MsgType             beschreibt die Art der MessageBox, wieviele und welche Buttons sichtbar sind(siehe Hinweise)

Rückgabewert:
je nach gedrücktem Button liefert die MB folgende Rückgabewerte:
MBButton_OK=1
MBButton_CANCEL=2
MBButton_RETRY=3
MBButton_IGNORE=4
MBButton_YES=5
MBButton_NO=6

Hinweise:
msgtype bestimmt das Erscheinungsbild der Box.
Je nach erforderlicher Situation können so dem User unterschiedliche
Reaktionsmöglichkeiten geboten werden. Folgende Typen sind möglich:
MBType_OK=1                   (OK)
MBType_YESNO=2                (Ja/Nein)
MBType_CANCELRETRYIGNORE=3    (Abbruch,Wiederholen,Ignorieren)
MBType_OKCANCEL=4             (OK,Abbruch)
MBType_RETRYCANCEL=5          (Wiederholen,Abbruch)
MBType_YESNOCANCEL=6          (Ja,Nein,Abbruch)

***
--------------------------------------------------------------------------------
MultiLineEditBox / TextViewBox
--------------------------------------------------------------------------------
Beschreibung:
Dies ist eine Box, die es ermöglicht, mehrzeiligen Text zu editieren bzw.
diesen auch nur darzustellen. Sie ist mit zwei ScrollBars ausgestattet um
sich komfortabler im Text bewegen zu können.
Derzeitig  unterstützt die Box nur die DELETE und die BACKSPACE Tasten.
BlockOperationen(CUT,COPY,PASTE) sind noch immer nicht implementiert (ja ich weiß :/).

Ping:
kein

Routinen:
AddMultiLineEditBox(), alle TO_Befehle, UpdateGadget()

***
function AddMultiLineEditBox (event as EventHandle ptr,PosX as integer,PosY as integer,NumChars as integer,NumRows as
integer,DisplayMode as integer=0) as Gadget ptr

Beschreibung:
Erzeugt einMultiLineEditBox

Parameter:
event               Zeiger auf einen EventHandle
PosX                Abstand vom linken Screenrand in px
PosY                Abstand vom oberen Screenrand in px
NumChars            Breite der Box in Anzahl Zeichen
NumRows             Höhe der Box in Anzahl Zeilen
DisplayMode         (optional)Darstellungsvariante, wenn 0 (voreingestellt) Editormodus, 1 TextViewmodus

Rückgabewert:
Zeiger auf ein erzeugtes Control, bei Fehlschlag 0

Hinweise:
keine

***
--------------------------------------------------------------------------------
RadioButton
--------------------------------------------------------------------------------
Beschreibung:
RadioButton treten in der Regel in Gruppen auf. Die Selektion eines Buttons der Gruppe schließt
die Selektion der anderen aus.

Ping:
RELEASE

Routinen:
AddRadioButton(), SetSelect, GetSelect(), SetCaption()

***
function AddRadioButton (event as EventHandle ptr,PosX as integer,PosY as integer,SwtchSel as integer,Text as string="",Head as
Gadget ptr) as Gadget ptr

Beschreibung:
```

Erzeugt ein RadioButton

Parameter:
event Zeiger auf einen EventHandle
PosX Abstand vom linken Screenrand in px
PosY Abstand vom oberen Screenrand in px
SwtchSel Initalselektion 0=nicht selektiert, 1=selektiert
Text Text rechts neben dem Button
Head Wenn 0 ist dieses RB der Kopf einer Gruppe, weitere Mitglieder der Gruppe übergeben
  hier dann den Gadget PTR des RBs das den Kopf darstellt

Rückgabewert:
Zeiger auf ein erzeugtes Control, bei Fehlschlag 0

Hinweise:
keine

***

ScrollBar

Beschreibung:
Stellt auch einen Schieberegler dar und soll Bildlaufleisten simulieren. Die Knobgröße passt sich dabei
proportional den entsprechenden Scrollgrößen an. Das "Klickverhalten" ist mit dem des TrackBars identisch.

Ping:
Ein Ping erfolgt bei HOLD durch Verschieben des Knobs wenn sich der Scrollwert ändert.
Beim Klicken neben den Knob wird der Pufferwert um die Pagegröße erhöht oder verringert.
Auch bei dieser Veränderung erfolgt ein Ping

Routinen:
AddScrollBar(), ModifyScrollBar(), GetScrollBarVal()

***
function AddScrollBar(event as EventHandle ptr,PosX as integer,PosY as integer,SBLength as integer,Minimum as integer,Maximum as integer,ScrollValue as integer,PageSize as integer,Orientation as integer,DisplayMode as integer=0) as Gadget ptr

Beschreibung:
Erzeugt ein Scrollbar

Parameter:
event Zeiger auf einen EventHandle
PosX Abstand vom linken Screenrand in px
PosY Abstand vom oberen Screenrand in px
SBLength Länge des SBs in px
Minimum untere Grenze den ein Scrollwert haben kann
Maximum obere Grenze den ein Scrollwert (unter umständen) haben kann
ScrollValue Intitialwert des Scrollwertes
PageSize Seitengröße
Orientation 0=horizontal, 1=verikal
DisplayMode (optional) voreingestellt 0= mit Arrows, 1= ohne Arrows

Rückgabewert:
Zeiger auf ein erzeugtes Control, bei Fehlschlag 0

Hinweise:
Eine kleine Übung vorweg:
man nehme einen beliebigen Editor (oder eine IDE) welcher die Zeilennummern anzeigen kann.
Des weiteren benötigen wir einen Text der exakt 50 Zeilen lang ist.
Nun verändert man die Höhe des Editorfensters so, daß man nur 20 Zeilen sehen kann!!!
Wenn man jetzt mit dem ScrollBar des Editors nach oben an den Textanfang "fährt" ist
erwartungsgemäß in der obersten Zeile (die TopLine des Editors) die Zeile 1 zu sehen.
Scrollt man hingegen an das Ende des Textes so sollte in der TopLine die 31. Zeile zu sehen sein.
Diese Situation übertragen auf das sGUI ScrollBar sieht dann so aus:

*Minimum=1 (Text beginnt bei 1. Zeile)
*Maximum=50 (Text endet mit 50. Zeile)
*ScrollValue=1 (den Text von der 1. Zeile ab betrachten)
*PageSize=20 (20 Zeilen sind gleichzeitig zu sehen)

Mit GetScrollBarVal() kann man den aktuellen Scrollwert abfragen.
Beim oben genannten Beispiel liefert GetScrollBarVal() Werte von 1 bis 31, also immer jene Zeile,
die sich in der TopLine befindet!!!
Bereich, den der Scrollwert haben kann (bei PageGrösse >1): Minimum =< Scrollvalue <= (Maximum - Pagesize + 1)
Wenn die PageGrösse=1 ist dann funktioniert das ScrollBar genauso wie ein TrackBar,
jeder Wert zwischen Minimum und Maximum ist "anscrollbar".

***
sub ModifyScrollBar overload(gad as Gadget ptr,ScrollValue as integer)
sub ModifyScrollBar (gad as Gadget ptr,Minimum as integer,Maximum as integer)
sub ModifyScrollBar (gad as Gadget ptr,Minimum as integer,Maximum as integer,ScrollValue as integer,PageSize as integer=0)

Beschreibung:
Verändert die entsprechenden Größen im SB und führt ein Update aus
je nach Umfang der gewünschten Änderungen stehen drei überladene Varianten zur Verfügung

Parameter:
Minimum untere Grenze den ein Scrollwert haben kann
Maximum obere Grenze den ein Scrollwert (unter umständen) haben kann
ScrollValue Intitialwert des Scrollwertes
PageSize Seitengröße

Rückgabewert:
keiner

Hinweise:
keine

***
function GetScrollBarVal (gad as Gadget ptr) as integer

Beschreibung:
Liefert den aktuellen Scrollwert des SBs

Parameter:
gad Zeiger auf ein Control

Rückgabewert:
Scrollwert

Hinweise:
keine

***

SimpleGadget

Beschreibung:
Ist eine simpler Button für so klassische Aufgaben wie "Yo Man", "Okay" oder "Nope" oder "Lade" .....

Ping:
RELEASE

Routinen:
AddSimpleGadget(), SetCaption

***
function AddSimpleGadget (event as EventHandle ptr,PosX as integer,PosY as integer,GadWidth as integer,GadHeight as integer,Text as string) as Gadget ptr

Beschreibung:
Erzeugt ein SimpleGadget

Parameter:
event Zeiger auf einen EventHandle
PosX Abstand vom linken Screenrand in px
PosY Abstand vom oberen Screenrand in px

Rückgabewert:
Zeiger auf ein erzeugtes Control, bei Fehlschlag 0

Hinweise:
keine

***

StringGadget

Beschreibung:
einzeiliges Texteingabefeld

Ping:
bei CHR(13), als der RETURNtaste

Routinen:
AddStringGadget(), SetString(), GetString(), alle TO_Befehle, UpdateGadget()

***
function AddStringGadget(event as EventHandle ptr,PosX as integer,PosY as integer,NumChars as integer,Text as string="",MaxChars as integer=0 ,CharLimitation as integer=0,ShowEnd as integer=0,Focus as integer=0) as Gadget ptr

Beschreibung:
Erzeugt ein StringGadget

Parameter:
event Zeiger auf einen EventHandle
PosX Abstand vom linken Screenrand in px
PosY Abstand vom oberen Screenrand in px
NumChars Länge des StringGadgets in Anzahl Zeichen
Text (optional) Initialtext
MaxChars (optional) maximale Anzahl der einzugebenden Zeichen
CharLimitation (optional) Zeichenbegrenzung
  0=alle Zeichen (voreingestellt)
  1=nur Ziffern für Integerwerte
  2=nur Ziffern und ein(!) Punkt für Fließkommazahlen
  3=nur die Ziffern 0 und 1 für Binärzahlen
  4=nur Ziffern und die Zeichen A bis F für Hexadezimalzahlen
  5=nur Ziffern und die Zeichen A bis F und Punkt für IP Adressen(also noch die alten ;))
ShowEnd (optional) wenn 1 dann wird der String im beim Verlassen (Return) "rechts-endig" (Ende ist zu sehen)
  dargestellt wenn der String länger als c ist. Die Grundeinstellung ist #links-endig#(Anfang ist zu sehen)
Focus (optional) wenn 1 dann behält das Control auch nach einem RETURN den Focus und bleibt aktiv

Rückgabewert:
Zeiger auf ein erzeugtes Control, bei Fehlschlag 0

Hinweise:
da das StringGadget für die Verwaltung ein TextObject benutzt, sind auch Manipulationen über TO_Befehle
möglich. Benutzt wird nur die erste Zeile eines TOs

***
function GetString (gad as Gadget ptr) as string

Beschreibung:
Liefert den Inhalt eines StringGadgets

Parameter:
gad Zeiger auf ein Control

Rückgabewert:
ein String

```
Hinweise:
keine
```

***

```
sub SetString (gad as Gadget ptr,Text as string)

Beschreibung:
Setzt den Inhalt eines StringGadgets

Parameter:
gad                 Zeiger auf ein Control
Text                neuer Inhalt

Rückgabewert:
keiner

Hinweise:
keine
```

***

```
-------------------------------------------------------------------------------------
ToggleGagdet
-------------------------------------------------------------------------------------
Beschreibung:
Dies ist ein Textoptionsbutton, von Funktion und Verhalten dem CheckMark ähnlich

Ping:
RELEASE

Routinen:
AddToggleGadget(), SetSelect(), GetSelect(), SetCaption()
```

***

```
function AddToggleGadget (event as EventHandle ptr,PosX as integer,PosY as integer,GadWidth as integer,GadHeight as integer,s as
integer,txt as string) as Gadget ptr

Beschreibung:
Erzeugt ein ToggleGadget

Parameter:
event               Zeiger auf einen EventHandle
PosX                Abstand vom linken Screenrand in px
PosY                Abstand vom oberen Screenrand in px
GadWidth            Breite des Buttons in px
GadHeight           Höhe des Buttons in px

Rückgabewert:
Zeiger auf ein erzeugtes Control, bei Fehlschlag 0

Hinweise:
keine
```

***

```
-------------------------------------------------------------------------------------
Menüystem
-------------------------------------------------------------------------------------
Beschreibung:
Das Menüsystem wurde erst spät in die sGUI integriert und hat so seine Besonderheiten.
Das "Handling", zumindest beim Erstellen von Menüs, soll ein wenig an die Menüprogrammierung
unter AmigaBasic erinnern. Einige Handhabungen, zb. die des Checkmarks, sind noch sehr suboptimal.

Routinen:
MenuOn(), MenuOff(), Menu(), GetChecked(), PullUpMenu()
```

***

```
sub MenuOn(event as EventHandle ptr)
sub MenuOff(event as EventHandle ptr)

Beschreibung:
Dies beiden Befehle schalten das Menu entweder ein oder aus. Selbst wenn noch kein Menü erstellt wurde
erscheint beim Einschalten zumindest der Menübalken.

Parameter:
event               Zeiger auf einen EventHandle

Rückgabewert:
keiner

Hinweise:
keine
```

***

```
function Menu(event as EventHandle ptr,menuID as integer,entryID as integer,status as integer,mname as string="") as integer

Beschreibung:
Dies ist der "komplizierteste" Befehl im Menüsystem.
Erzeugt ein MenuHeader bzw. MenuItem. Dient zur Manipulation des Menüs.
Bei der Steuerung des CheckHäkchens ist dies sogar erforderlich.

Parameter:
event               Zeiger auf einen EventHandle
menuID              definiert die Nummer des Menü-Strips der erzeugt/manipuliert werden soll
entryID             ist die entryID=0 dann wird der MenuHeader des Menüs mit der Nummer menuID
                    erzeugt/manipuliert. Also das was als Menüname in der Menüleiste zu sehen ist.
                    ist die entryID >1  so wird ein Menü-Eintrag des Menüs mit der Nummer menuID
                    erzeugt/manipuliert.
status              hiermit lassen sich die Art der Einträge manipulieren, aber auch der komplette Menü-Strip
                    aktivieren/deaktivieren
                    Folgende Statuskonstanten sind definiert:
                    MS_INACTIVE=0
                    MS_ACTIVE=1
                    MS_CHECKMARK=2
```

```
                    MS_SEPARATOR=4
                    Mit ihnen lassen sich durch einfaches Addieren bzw bitweise OR Verknüpfung
                    folgende Header/Items erzeugen:
                    *0(MS_INACTIVE)     >> inaktiver Header / inaktives Item
                    *1(MS_ACTIVE)       >> aktiver Header / aktives Item
                    *2(MS_CHECKMARK)    >> inaktives Item mit CheckMark
                    *3(MS_ACTIVE OR MS_CHECKMARK) >> aktives Item mit CheckMark
                    *4(MS_SEPARATOR)    >> Trennlinie
                    unlogische Statuswerte werden ignoriert
mname               Text eines MenuHeaders/Entrys, sollte status=4(Separator) sein so wird dieser Parameter ignoriert

Rückgabewert:
eine auf integer geCASTete Adresse eines Menüeintrages

Hinweise:
keine

***
function GetChecked (event as EventHandle ptr,menuID as integer,entryID as integer) as integer

Beschreibung:
Überprüft einen Entry ob ein CheckMark gesetzt ist oder nicht. Dies soll die Manipulierbarkeit des CheckMarks eines
Eintrages im EventLoop erleichtern und auch nur auf den betreffenden Menü-Eintrag angewendet werden!

Parameter:
event               Zeiger auf einen EventHandle
menuID              Hmmm, ID des Menüs in dem sich der zu untersuchende Eintrag befindet
entryID             ID des zu untersuchenden Eintrages

Rückgabewert:
wenn 1 CheckMark gesetzt, 0 nicht gesetzt

Hinweise:
keine

***
sub PullUpMenu (event as EventHandle ptr)

Beschreibung:
Ist ein Notausstieg um ein Menü hoch zu fahren und zu schließen. Da das Menü direkt in den SCREEN
gezeichnet wird, kann es eventuell zu Grafik-Problemen kommen wenn an gleicher Stelle andere grafische
Aktionen auf dem Screen erfolgen sollen. Zugegeben: eine Krücke!!! :-/ Erfüllt aber seinen Zweck.

Parameter:
event               Zeiger auf einen EventHandle

Rückgabewert:
keiner

Hinweise:
keine

***
-------------------------------------------------------------------------------------
Menüystem Programmbeispiele
-------------------------------------------------------------------------------------
Erscheinungsformen:
Folgender Code erzeugt alle möglichen "Erscheinungsformen" des Menüsystems
(ohne Auswertung im EventLoop):

[code]
'Compileroption -s console
#include "sGUI\sGUI.bas"
#include "sGUI\Menu.bas"
using sGUI

screen 19,32
InitGFX

dim as EventHandle ptr event=NEW EventHandle

Menu (event,1,0,1,"Menue A")           'Header, also das was im MenuBar als Menüname zu sehen ist
Menu (event,1,1,1,"Eintrag A")         'normaler Eintrag, aktiv
Menu (event,1,2,0,"Eintrag B")         'normaler Eintrag, inaktiv
Menu (event,1,3,4,"")                  'Separator(Trennlinie)
Menu (event,1,4,3,"Option A")          'Eintrag mit CheckMark, aktiv
Menu (event,1,5,2,"Option B")          'Eintrag mit CheckMark, inaktiv

Menu (event,2,0,1,"")                  'Menü 2 und sein Strip wird nicht dargestellt, da dessen Header einen Leerstring ist
Menu (event,2,1,1,"Eintrag E")         '
Menu (event,2,2,0,"Eintrag F")         '

                                       'da Menü 2 nicht dargestellt wird, rückt Menü 3 nach links auf

Menu (event,3,0,0,"Menue C")           'wenn Header inaktiv ist, dann ist auch der komplette Strip inaktiv, egal welche
                                       'Aktivierungen...
Menu (event,3,2,1,"Eintrag D")         'die einzelnen Einträge haben( hier aktiv)...
Menu (event,3,1,0,"Eintrag C")         '(und hier inaktiv).

MenuOn(event)

dim as integer ff=freefile
open cons for output as ff

do
  event->xSleep(1)
loop until event->EXITEVENT

close ff

DELETE event
[/code]
```

***
Abfragen:
Die Abfrage eines selektierten Menüpunktes erfolgt natürlich im Eventloop.
Der EventHandle hält dafür die Variablen ->MENUMESSAGE, ->MENUNUMBER und ->ITEMNUMBER bereit.
Sobald ->MENUMESSAGE > 0 ist, wurde ein Menueintrag angewählt. Welches Menü und welcher Eintrag
gewählt wurde ist dann mit ->MENUNUMBER und ->ITEMNUMBER zu erfragen.

[code]
```
'Compileroption -s gui
#include "sGUI\sGUI.bas"
#include "sGUI\Menu.bas"
using sGUI

screen 19,32
InitGFX

dim as EventHandle ptr event=NEW EventHandle

dim as integer itema,itemb

Menu(event,1,0,1,"File")
Menu(event,1,1,1,"Laden")
Menu(event,1,2,1,"Sichern")

MenuOn(event)

do
  event->xSleep(1)
  if event->MENUMESSAGE then
    select case event->MENUNUMBER
    case 1
      select case event->ITEMNUMBER
      case 1
        beep
      case 2
        beep:beep
      end select
    case 2
    '........
    end select
  end if
loop until event->EXITEVENT

DELETE event
```
[/code]

***
Der Menu() Befehl ist eigentlich eine Funktion. Wenn erfolgreich ein Item/Header erstellt wurde,
ist das Ergebnis dieser Funktion ein Integerwert >0 (genauer gesagt ist es die geCASTete Adresse
des reservierten Speichers des erzeugten Items/Headers).
->MENUMESSAGE liefert bei Anwahl eines Items auch genau diesen Integerwert!
Und dies kann man nutzen um eine vereinfachte Abfrage zu erstellen. Voraussetzung ist,
daß das Ergebnis beim Erstellen der Items durch den Menu() Befehl in einer Variable festgehalten wird:

[code]
```
'Compileroption -s gui
#include "sGUI\sGUI.bas"
#include "sGUI\Menu.bas"
using sGUI

screen 19,32
InitGFX

dim as EventHandle ptr event=NEW EventHandle

dim as integer itema,itemb

Menu(event,1,0,1,"File")
itema=Menu(event,1,1,1,"Laden")
itemb=Menu(event,1,2,1,"Sichern")

MenuOn(event)

do
  event->xSleep(1)
  if event->MENUMESSAGE then
    select case event->MENUMESSAGE
    case itema
         beep
    case itemb
         beep:beep
    end select
  end if
loop until event->EXITEVENT

DELETE event
```
[/code]

***
Header oder Item ändern:
Um einen Header oder ein Item zu verändern, bedarf es nur eines weiteren Aufrufes
des MENU() Befehles mit entsprechend neuen Parametern. Wenn beispielsweise ein Item
im weiteren Programmverlauf deaktiviert werden muß (analog dem GadgetSleep).
Im Fall des Items mit Checkmark ist eine Veränderung bei Auswahl durch den User sogar erforderlich.
Folgender Code zeigt eine Möglichkeit, das CheckMark verschwinden und wieder erscheinen zu lassen,
wenn der User dieses Item anwählt:

[code]
```
'Compileroption -s console
#include "sGUI\sGUI.bas"
#include "sGUI\Menu.bas"
```

```
using sGUI

screen 19,32
InitGFX

dim as EventHandle ptr event=NEW EventHandle

Menu (event,1,0,1,"Menue A")
Menu (event,1,3,3,"Option")
MenuOn(event)

dim as integer ff=freefile
open cons for output as ff

do
  event->xSleep(1)
  if event->MENUMESSAGE then
    select case event->MENUNUMBER
      case 1
        select case event->ITEMNUMBER
          case 3
            if GetChecked(event,1,3) then
              Menu (event,1,3,1,"Option")
              print #ff,"CheckMark nicht gesetzt"
            else
              Menu (event,1,3,3,"Option")
              print #ff,"CheckMark gesetzt"
            end if
       end select
    end select

  end if
loop until event->EXITEVENT

close ff

DELETE event)
[/code]

***
--------------------------------------------------------------------------------
TextObject
--------------------------------------------------------------------------------
Beschreibung:
Das Textobjekt dient der Verwaltung eines mehrzeiligen Textes.
Es kann Zeilen Löschen, Splitten, Zusammenfügen, neue Erzeugen.
Desweiteren kann durch einen internen Cursor die Zeile editiert werden.
Das TO verhält sich passiv. Wenn eine Aktion mit dem Text passieren soll, so
ist der entsprechende Befehl aufzurufen. Das TO zeigt den Text nicht selbst an.
Das TO fragt nicht selbst die Tastatur ab.
Alles was mit dem Text passieren soll/kann muß von außen ausgelöst werden.

Der Zugriff auf ein TO und dessen Befehle erfolgt über die so genannten TO_Befehle.
Dadurch ist sichergestellt, das auf das richige TO zugegriffen wird bzw werden
Versuche abgefangen ein TO eines Controls zu verändern, das keins hat!

Routinen:
... öhhmmm!!! Viele!!!

***
function TO_KeyAddChar(gad as _Gadget ptr,c as string)as integer

Beschreibung:
Fügt an der Cursorposition das Zeichen c ein

Parameter:
gad                 Zeiger auf ein Control
c                   einzufügendes Zeichen (1 Byte)

Rückgabewert:
wenn erfolgreich 1, sonst 0

Hinweise:
keine

***
function TO_KeyBackspace(gad as _Gadget ptr)as integer

Beschreibung:
Löscht das Zeichen vor dem Cursor. Je nach Cursorposition kann auch eine Zeile gelöscht werden
Parameter:
gad                 Zeiger auf ein Control

Rückgabewert:
wenn erfolgreich 1, sonst 0

Hinweise:
keine

***
function TO_KeyDelete(gad as _Gadget ptr)as integer

Beschreibung:
Löscht das Zeichen auf(!) dem der Cursor steht. Je nach Cursorposition kann auch eine Zeile gelöscht werden

Parameter:
gad                 Zeiger auf ein Control

Rückgabewert:
wenn erfolgreich 1, sonst 0

Hinweise:
```

keine

***

function TO_KeyReturn(gad as _Gadget ptr)as integer

Beschreibung:
fügt eine neue Zeile nach der Cursorposition ein und, in Abhängigkeit von der aktuellen
Cursorposition, verschiebt/splittet oder belässt den Inhalt der alten Zeile

Parameter:
gad Zeiger auf ein Control

Rückgabewert:
wenn erfolgreich 1, sonst 0

Hinweise:
keine

***

function TO_SetCursor overload(gad as _Gadget ptr,i as integer,p as integer)as integer

Beschreibung:
Setzt den Cursor in die Zeile mit dem Indes i, an die Position p. Gegebenfalls erfolgt jedoch eine Anpassung
an ein mögliche Position

Parameter:
gad Zeiger auf ein Control
i Index einer Zeile
p (gewünschte) Position des Cursors

Rückgabewert:
wenn erfolgreich 1, sonst 0

Hinweise:
keine

***

function TO_SetCursor(gad as _Gadget ptr,l as Textline ptr,p as integer)as integer

Beschreibung:
Setzt den Cursor in die Zeile mit der Adresse l, an die Position p. Gegebenfalls erfolgt jedoch eine Anpassung
an ein mögliche Position

Parameter:
gad Zeiger auf ein Control
l Adresse einer Zeile
p (gewünschte) Position des Cursors

Rückgabewert:
wenn erfolgreich 1, sonst 0

Hinweise:
keine

***

function TO_CursorUp(gad as _Gadget ptr)as integer

Beschreibung:
Verschiebt den Cursor eine Zeile nach oben. Es wird versucht die Position beizubehalten. Sollte dies nicht möglich sein erfolgt
eine Anpassung der Cursorposition

Parameter:
gad Zeiger auf ein Control

Rückgabewert:
wenn erfolgreich 1, sonst 0

Hinweise:
keine

***

function TO_CursorDown(gad as _Gadget ptr)as integer

Beschreibung:
Verschiebt den Cursor eine Zeile nach unten. Es wird versucht die Position beizubehalten. Sollte dies nicht möglich sein erfolgt
eine Anpassung der Cursorposition

Parameter:
gad Zeiger auf ein Control

Rückgabewert:
wenn erfolgreich 1, sonst 0

Hinweise:
keine

***

function TO_CursorLeft(gad as _Gadget ptr)as integer

Beschreibung:
Verschiebt den Cursor eine Position nach links. Sollte der Cursor vor dieser Aktion am Zeilenanfang gestanden
haben, befindet er sich nun am Ende der vorhergehenden Zeile

Parameter:
gad Zeiger auf ein Control

Rückgabewert:
wenn erfolgreich 1, sonst 0

Hinweise:
keine

***
function TO_CursorRight(gad as _Gadget ptr)as integer

Beschreibung:
Verschiebt den Cursor eine Position nach rechts. Sollte der Cursor vor dieser Aktion am Zeilenende gestanden haben, befindet er sich nun am Anfang der nächsten Zeile

Parameter:
gad Zeiger auf ein Control

Rückgabewert:
wenn erfolgreich 1, sonst 0

Hinweise:
keine

***
function TO_CursorKeyPos1(gad as _Gadget ptr)as integer

Beschreibung:
Setzt den Cursor an den Zeilenanfang

Parameter:
gad Zeiger auf ein Control

Rückgabewert:
wenn erfolgreich 1, sonst 0

Hinweise:
keine

***
function TO_CursorKeyEnd(gad as _Gadget ptr)as integer

Beschreibung:
Setzt den Cursor ans Zeilenende

Parameter:
gad Zeiger auf ein Control

Rückgabewert:
wenn erfolgreich 1, sonst 0

Hinweise:
keine

***
function TO_GetCursorLine(gad as _Gadget ptr) as integer

Beschreibung:
Liefert die Cursorlinie

Parameter:
gad Zeiger auf ein Control

Rückgabewert:
integer

Hinweise:
keine

***
function TO_GetCursorPosition(gad as _Gadget ptr) as integer

Beschreibung:
Liefert die Cursorposition

Parameter:
gad Zeiger auf ein Control

Rückgabewert:
integer

Hinweise:
keine

***
function TO_GetLines (gad as _Gadget ptr) as integer

Beschreibung:
Liefert Gesamtzahl der derzeitig existierenden Zeilen

Parameter:
gad Zeiger auf ein Control

Rückgabewert:
0 wenn keine Zeilen existieren, sonst >0

Hinweise:
keine

***
function TO_GetLineAddr(gad as _Gadget ptr,i as integer) as TextLine ptr

Beschreibung:
Liefert die Adresse einer Zeile auf Basis ihres Indexes

Parameter:
gad Zeiger auf ein Control
i Index einer Zeile

```
Rückgabewert:
Adresse einer Zeile als TextLine ptr

Hinweise:
keine
```

***

```
function TO_GetLineIndex(gad as _Gadget ptr,r as TextLine ptr) as integer

Beschreibung:
Liefert den Index einer Zeile auf Basis der Adresse der Zeile

Parameter:
gad                 Zeiger auf ein Control
r                   Zeiger auf eine Zeile

Rückgabewert:
wenn erfolgreich >0

Hinweise:
keine
```

***

```
function TO_GetLongestLine(gad as _Gadget ptr) as integer

Beschreibung:
Liefert die Länge der längsten Zeile. Ob diese noch existiert, ist nicht gegeben.
Ist mehr eine Art "Schleppzeiger"

Parameter:
gad                 Zeiger auf ein Control

Rückgabewert:
integer

Hinweise:
keine
```

***

```
function TO_LoadText (gad as _Gadget ptr,filename as string,linebreak as string=chr(13,10)) as integer

Beschreibung:
Lädt eine Datei ins TO

Parameter:
gad                 Zeiger auf ein Control
filename            Dateiname der zu ladenen Datei
linebreak           Zeichen oder Zeichenkette die den Zeilenumbruch in der zu ladenen Datei darstellt, voreingestellt chr(13,10)

Rückgabewert:
wenn erfolgreich 1, sonst 0

Hinweise:
keine
```

***

```
function TO_SetText (gad as _Gadget ptr,s as string,linebreak as string=chr(13,10)) as integer

Beschreibung:
Lädt den String ins TO

Parameter:
gad                 Zeiger auf ein Control
s                   ein String der den gesamten Text enthält
linebreak           Zeichen oder Zeichenkette die den Zeilenumbruch im String darstellt, voreingestellt chr(13,10)

Rückgabewert:
wenn erfolgreich 1, sonst 0

Hinweise:
keine
```

***

```
function TO_SaveText (gad as _Gadget ptr,filename as string, linebreak as string=chr(13,10))as integer

Beschreibung:
Schreibt den gesamten Text in eine Datei, Zeilenumbruch definierbar

Parameter:
gad                 Zeiger auf ein Control
filename            Dateiname
linebreak           Zeichen oder Zeichenkette die den Zeilenumbruch in der zu schreibenden Datei darstellt, voreingestellt chr(13,10)

Rückgabewert:
wenn erfolgreich 1, sonst 0

Hinweise:
keine
```

***

```
function TO_GetText (gad as _Gadget ptr,linebreak as string=chr(13,10)) as string

Beschreibung:
Gibt alle Zeilen als ein String aus

Parameter:
gad                 Zeiger auf ein Control
linebreak           Zeichen oder Zeichenkette die den Zeilenumbruch im Ausgabestring darstellt, voreingestellt chr(13,10)

Rückgabewert:
String der den kompletten Text enthält, bei Fehlschlag oder keinem Text ein Leerstring!
```

Hinweise:
keine

***

function TO_GetLineContent overload(gad as _Gadget ptr,i as integer) as string

Beschreibung:
Liefert den Inhalt einer Zeile

Parameter:
gad Zeiger auf ein Control
i Index eine Zeile

Rückgabewert:
String, Inhalt einer Zeile

Hinweise:
keine

***

function TO_GetLineContent (gad as _Gadget ptr,l as TextLine ptr) as string

Beschreibung:
Liefert den Inhalt einer Zeile

Parameter:
gad Zeiger auf ein Control
l Zeiger auf eine Zeile

Rückgabewert:
String, Inhalt einer Zeile

Hinweise:
keine

***

function TO_SetLineContent overload(gad as _Gadget ptr,i as integer,t as string)as integer

Beschreibung:
Ersetzt den Inhalt einer Zeile mit dem neuen aus dem String t

Parameter:
gad Zeiger auf ein Control
i Index eine Zeile
t neuer Text

Rückgabewert:
wenn erfolgreich 1, sonst 0

Hinweise:
keine

***

function TO_SetLineContent (gad as _Gadget ptr,l as TextLine ptr,t as string)as integer

Beschreibung:
Ersetzt den Inhalt einer Zeile mit dem neuen aus dem String t

Parameter:
gad Zeiger auf ein Control
l Zeiger auf eine Zeile
t neuer Text

Rückgabewert:
wenn erfolgreich 1, sonst 0

Hinweise:
keine

***

function TO_AppendLine (gad as _Gadget ptr,t as string) as integer

Beschreibung:
Hängt eine neue Zeile, die den Inhalt des Strings t hat, an den bestehenden Text

Parameter:
gad Zeiger auf ein Control
t Text

Rückgabewert:
wenn erfolgreich 1, sonst 0

Hinweise:
keine

***

function TO_ClearText (gad as _Gadget ptr) as integer

Beschreibung:
Löscht alle Zeilen, also den ganzen Text. GetLines() liefert danach 0

Parameter:
gad Zeiger auf ein Control

Rückgabewert:
wenn erfolgreich 1, sonst 0

Hinweise:
keine

***