… eines vorweg:
Auch wenn sehr viel Code wiederverwertet wurde und durchaus verwandtschaftliche Zusammenhänge bestehen, sGUI2 ist leider absolut inkompatibel zum Vorgänger v0.8.4!

Ein Umstieg bei bestehenden Projekten sollte also wohl überlegt sein und kann mit viel Aufwand verbunden sein.

sGUI2

sGUI2 ist eine kleine Bibliothek die dem Nutzer ermöglicht, eine „simple“ Bedienoberfläche auf dem nativen FB-Screen zu erstellen.

Geschrieben und getestet wurde unter **Windows 11**.
auch unter **Linux Mint 21** konnte ich funktionierende Binaries erzeugen.

sGUI2 bietet bisher folgende Features:
* simples Fenstersystem
* Menüsystem
* einfache Buttons
* CheckMarkGadgets, RadioButtons, ToggleGadgets
* Labels
* Scrollbars
* ListBoxen
* DropDownListBoxen (ehem. ComboBox)
* StringGadgets (Texteingabefeld)
* einfaches TextField (mehrzeiliger Text, nach wie vor nur rudimentäre Editiermöglichkeit)
* FileRequester (Dateidialog)
* änderbares Farbschema
* BitmapFont-Unterstützung

sGUI2 benutzt nun einen **zweiseitigen Screen**, die GUI selbst ist sozusagen eine Art Overlay.
Der Nutzer kann für eigene Ausgaben, sei es Text oder Grafik, den Screen „ganz normal“ weiter benutzen.

Einschränkungen bei FB-Befehlen

Folgende Befehle/Funktionen sind nicht oder nur eingeschränkt benutzbar:

*SCREEN/SCREENRES
*SCREENSET
*SCREENEVENT
*INKEY

Für einige Befehle gibt ein entsprechendes Replacement, falls sGUI2 benutzt werden soll. Für die Funktionen gibt es sGUI-Systemvariablen die dann stattdessen abgefragt werden müssen.

Fenstersystem

Nun gut, die Fenster haben eine feste Grösse. Vorrangig sollten sie Controls enthalten. Es sind aber auch eigene Grafiken als Hintergrund-Layer möglich.

Big Screens @ High DPI

sGUI ist grafisch nicht direkt skalierbar. Es bietet lediglich die Möglichkeit auch auf grossen Screens etwas Les- und Bedienbares zu erstellen. Abhängig von der Gadgetart sind die Verfahren recht unterschiedlich und leider nicht so ganz einheitlich.

Der erste Schritt ist mit Sicherheit das Benutzen eines großen Bitmap-Fonts. sGUI unterstützt dies nun.
Buttons, ToggleGadgets, RadioToggles, ListBoxen, DropDownListBoxen sind Gadgets denen bei Erzeugung eine eindeutige Grösse in Breite und Höhe mitgegeben wird, unabhängig vom benutzten Font. Man macht sich die Dinger einfach so groß wie man sie braucht, fertig :).

StringGadgets und Labels sind in ihrer Höhe an den benutzten/geladenen Font gebunden.

Ursprüglich mit einer festen Breite und/oder Höhe programmierte Gadgets wie Arrows, Scrollbars, RadioButtons oder das CheckMark werden nun, abhängig vom geladenen Font, skaliert.

Zumindest sind nun einige „Features“ in sGUI enthalten, die das Arbeiten auch mit höheren Auflösungen möglich machen.

Inhaltsverzeichnis

Wie üblich folgt nun ein kleines einfaches Beispiel, um einen ersten Eindruck zu bekommen, wie das ganze hier so funktioniert:

<code>

```
'1)
#include "sGUI\sGUI.bas"
#include once "sGUI\Gadget_SimpleToggle.bas"

'2.)
using sGUI

'3.)
sGUIScreen 17
InitGUI
color Colors.Pen,Colors.BackGround
cls

'4.)
dim as Gadget ptr mybutton,button2
dim as sGUIWindow ptr mywindow,childwindow
mywindow=AddWindow(0,50,75,300,100,"Fenster",WFLAG_DRAGABLE or WFLAG_FRAMED)
childwindow=AddWindow(mywindow,135,20,150,50,"ChildFenster",WFLAG_DRAGABLE or WFLAG_FRAMED)

mybutton=AddSimpleGadget(0,10,10,120,40,"Click Me!")
button2=AddSimpleGadget(mywindow,10,10,120,40,"Click Here!")

ShowWindow(mywindow)
ShowWindow(childwindow)
GadgetOn(mybutton)
GadgetOn(button2)

'5.)
do
  sleep 1
  MasterControlProgram
  if GADGETMESSAGE then
    select case GADGETMESSAGE
      case mybutton
        print "mybutton geklickt"

      case button2
        print "button2 geklickt"

    end select
  end if
loop until SCREENCLOSEBUTTON
```

</code>

Wie man sieht, auf den ersten Blick scheint es nicht viele Unterschiede zur „alten sGUI“ zu geben, warten wir es ab...

Der sGUI Ordner

In diesem Ordner sind alle Dateien enthalten, die man braucht um sGUI2 benutzen zu können. Am besten man kopiert der kompletten Ordner in seinen Projektordner. Mehr ist eigentlich nicht zu tun.
***

Die Includes

Um das Ganze nutzen zu können bedarf es einiger Incudes. Dies sollte gleich zum Anfang includiert werden, naja, wie wohl bei den meisten anderen Bibliotheken auch :).

Die erste zu includierende Datei ist immer die "sGUI\sGUI.bas", die „GrundBasis“ !

Danach folgen die control-spezifischen Includes. Diese sollten immer mit dem Schlüsselwort „ONCE“ includiert werden. Welche Datei brauche ich für welches Control? Hier eine Liste:

"sGUI\Gadget_ScrollBar.bas"	Scrollbar
"sGUI\Gadget_DropDownListBox.bas"	DropDownListBox
"sGUI\Gadget_Label.bas"	Labels
"sGUI\Gadget_SimpleToggle.bas"	einfacher Button, ToggleButton
"sGUI\Gadget_RadioCheck.bas"	RadioButton, CheckBox, RadioToggle
"sGUI\Gadget_EditBox.bas"	einfache TextBox
"sGUI\Gadget_String.bas"	StringGadget
"sGUI\Gadget_FileRequester.bas"	DateiDialog

***

Der Namespace

Das gesamte System besteht (leider!) aus sehr vielen globalen (SHARED) Variablen bzw. Konstanten. Das ist ein nicht gerade sehr schöner Programmierstil.

Um Kollisionen usw. zu vermeiden, ist die sGUI in einen NAMESPACE namens „sGUI“ verpackt.
Ein „USING sGUI“ ist darum empfehlenswert.

***

Der (sGUI)SCREEN

sGUI benötigt zum „Betrieb“ einen Screen mit 2 Seiten. Damit dies sicher gestellt ist, gibt es sGUIScreen und sGUIScreenRes.

***

InitGUI

Einige „Dinge“ (Imagezeugs) sind erst realisierbar, wenn wir einen Screen erzeugt haben.
Dafür ist der Befehl „InitGUI“ gedacht.
Zum einen erzeugt dieser Befehl ein sogenanntes „RootWindow“, eine Art Einstiegspunkt, Zum anderen werden die GUI-Farben festgelegt.
Wer Screenhintergrund und Stiftfarbe (für PRINT) an die GUI anpassen will, sollte folgende Zeilen **direkt nach** InitGUI einfügen:

<code>
color Colors.Pen,Colors.BackGround
cls
</code>

In einem späteren Kapitel wird noch auf die Möglichkeit eingegangen, eigene Farbschemen zu erstellen und zu nutzen.

***

Controls und Fenster

Im allgemeinen werden nun Fenster und Controls erzeugt. In manchen Fällen können auch (Daten-)Listen erforderlich sein.
Um mit den Objekten arbeiten zu können, werden typisierte Pointervariablen benötigt:
Gadget Ptr für Controls, **sGUIWindow Ptr** für Fenster und **sGUIList Ptr** für Listen.

Im Abschnitt 4.) des oben gezeigten Code-Beispiels sieht man eigentlich recht gut wie das Erzeugen der GUI-Elemente vonstatten geht.

Der erste Parameter ist, sowohl bei Fenstern als auch bei Controls, immer ein Zeiger auf ein „Eltern“- Fenster. In diesem Eltern-Fenster wird dann das Objekt dargestellt.
Die entsprechenden Befehle dafür werden noch näher erläutert.

Die erzeugten Fenster und Controls sind nach dem Erzeugen nicht sichtbar und müssen „aktiviert“ werden. Auch zu diesen Befehlen später mehr.

***

Die Ereignis-Schleife

sGUI benötigt eine Ereignis-Schleife in der permanent über den Befehl „MasterControlProgram“ der Zustand der GUI abgefragt und auf dem Screen dargestellt wird (Abschnitt 5.) des Code-Beispiels).
Die Prozessorauslastung kann und sollte innerhalb dieser Schleife mit „SLEEP“ reguliert werden.

...Und das wars auch schon :)

***

Referenz zu den wichtigsten Befehlen

sGUIScreen() / sGUIScreenRes()

erzeugen einen zum Betrieb von sGUI benötigten Screen

*** Syntax:
sub sGUIScreen (scrmode as integer, flags as integer=0, refreshrate as integer=0)
sub sGUIScreenRes (scrwidth as integer, scrheight as integer, flags as integer=0, refreshrate as integer=0)

*** Parameter:

scrmode	alle hier aufgeführten Parameter werden an entsprechender Stelle an
flags	die FB-Befehle SCREEN/SCREENRES übergeben.
refreshrate	siehe dort... :)
scrwidth	
scrheight	

*** Rückgabewert:
keiner

*** Hinweise:
beide Befehle sollen lediglich die Erzeugung eines 32bit-Screens mit 2 Seiten sicherstellen.
Wer dies berücksichtigt, kann natürlich auch Screen/ScreenRes verwenden.

InitGUI

schafft die nötigen Voraussetzungen für den Betrieb von sGUI

*** Syntax:
sub InitGUI

*** Parameter:
keine

*** Rückgabewert:
keiner

*** Hinweise:
muß immer nach einem Screen(),ScreenRes() bzw sGUIScreen()/sGUIScreenRes() Befehl aufgerufen werden.

BitmapFont

SGUI bietet nun intern die Möglichkeit BitmapFonts zu benutzen, alternativ oder auch parallel zum FB-eigenen Font.
Es gibt drei „Slots“ in denen ein BitmapFont geladen werden kann(!). Diese drei werden in sGUI entsprechend unterschiedlich genutzt.

Der **FixedFont-Slot** wird ausschließlich in StringGadgets und TextFields benutzt.
Wichtig hierbei ist: der geladene BitmapFont **muss nicht** unbedingt ein monospaced Font sein, er wird aber in jedem Fall monospaced dargestellt werden!

Der **MenuFont-Slot** wird ausschließlich in allen Menü-Varianten benutzt.

Der **ProportionalFont-Slot** ist für den Rest, also allem oben noch nicht erwähnten… Fenstertitel, Buttons, Labels usw.
Auch hier sei gesagt: der geladene BitmapFont **muss nicht** unbedingt ein proportionaler Font sein, er wird so „benutzt“ wie er ist!

Generell sei gesagt, die unterstützen BitmapFonts müssen mit meinem FontConverter **Charset2FBFont** erstellt worden sein.

LoadProportionalFont

*** Syntax:
sub LoadProportionalFont (filename as string)

*** Parameter:
filename Dateiname eines BitmapFonts

*** Rückgabewert:
/

*** Hinweise:
/

LoadFixedFont

Läd einen Bitmap-Font in den entsprechenden Slot. Dieser BitmapFont sollte monospaced sein.
Der hier geladene Font wird ausschliesslich im StringGadget und im TextField verwendet.

*** Syntax:
sub LoadFixedFont (filename as string)

*** Parameter:
filename Dateiname eines BitmapFonts

*** Rückgabewert:
/

*** Hinweise:
Falls der geladene Font nicht monospaced ist, kann die Laufweite (feste Breite) nachträglich mit SetFixedWidth() eingestellt werden

SetFixedWidth

Falls via LoadFixedFont() kein monospaced Font geladen wurde, lässt sich mit diesem Befehl die voreingestellte feste Laufweite der Zeichen nachträglich verändern. Es hilft nur probieren :)

*** Syntax:
sub SetFixedWidth(w as integer)

*** Parameter:
w Laufweite oder feste Breite der Zeichen des BitmapFonts im „monospaced Font Slot“

*** Rückgabewert:
/

*** Hinweise:
/

LoadMenuFont

Läd einen Bitmap-Font in den entsprechenden Slot für die Darstellung der Menüs.

*** Syntax:
sub LoadMenuFont (filename as string)

*** Parameter:

filename	Dateiname eines BitmapFonts

*** Rückgabewert:
/

*** Hinweise:
/

BitmapFont Printroutinen

Es gibt die Möglichkeit die geladenen BitmapFonts auch für die Textausgabe zu nutzen. Zum einen gibt es die sGUI-interne Printroutine, zum anderen ist in sGUI die „BitmapFont.bi“ eingebunden. Dies ermöglicht das Einbinden und Benutzen weiterer BitmapFonts, unabhängig von sGUI.

DrawString

Dies ist die sGUI-interne Printroutine. Mit entprechenden Parametern aufgerufen kann sie alle drei geladene Fonts darstellen.
Sollte ein „Slot“ leer sein, benutzt diese Printroutine den FB-internen Font.

*** Syntax:
sub DrawString (img as FB.Image ptr=0, xpos as integer, ypos as integer, txt as string, PenColor as ulong=&H000000, mode as integer=0)

*** Parameter:

img	ein Zeiger auf ein FB.Image, in den „hineingeschrieben“ werden soll. Wenn 0 dann erfolgt die Ausgabe auf dem Screen. Falls die Ausgabe in einem **Fenster** erfolgen soll, ist vorher mittels **GetUserImage** ein entprechendes Image für das Fenster bereitzustellen
xpos ypos	Positionierung des Textes, es gelten jeweiligen die „lokalen“ Koordinaten der Ausgabe
txt	der aus zugebene Text, es werden nur Strings untestützt, Zahlen müssen entsprechen konvertiert übergeben werden
PenColor	Farbe des Textes, kann als HEX-Wert oder dem RGB-Makro übergeben werden
mode	=0 es wird für die Ausgabe der ProportionalFont benutzt =1 der FixedFont (monospaced) wird benutzt =2 der MenuFont wird benutzt

*** Rückgabewert:
/

*** Hinweise:
/

UDT BitmapFont

Dies ist ein UDT(User Defined Type) mit Ausgabe-Methoden für BitmapFonts. Es ermöglicht das Benutzen weiterer BitmapFonts, unabhängig von sGUI. Dieses UDT ist Bestandteil der in sGUI eingebundenen „BitmapFont.bi“, die aus dem CharSet2FBFont-Paket stammt. Die schon oben beschriebene Ausgaberoutine „DrawString“ basiert letztlich auch diesem UDT

Eine kleine Demo hierfür:

<code>
```
#include "sGUI\sGUI.bas"
using sGUI

sGUIScreenRes 1280,800,32
InitGUI
```

```
dim as BitmapFont myfont                                    'Erzeugen eines BitmapFonts
myfont.LoadFont ("Fonts\ArtDeco\ArtDeco.bmp")               'Font laden

'Ausgabe konfigurieren
myfont.PenColor=&HFF3300                                    'Stiftfarbe setzen

'Ausgabe
'Syntax: [font].Print (img as FB.Image ptr=0, posx as integer, posy as integer, text as string)
'img            Zeiger auf ein Image, 0 wenn auf Screen
'posx,posy      Position des Textes
'text           Auszugebener Text, nur Strings erlaubt, Zahlen entprechend konvertiert übergeben
myfont.Print (0,10,10, "Text")                              'Ausgabe auf Screen

'Ausgabe konfigurieren
myfont.PenColor=&H00AAFF                                    'Stiftfarbe setzen
myfont.BackGround=&H550000                                  'Hintergrundfarbe setzen
myfont.UseBackGround=1                                      'Hintegrundfarbe 0=aus, 1=ein

'Ausgabe
myfont.Print (0,10,40, "Solid Background Output")

'Ausgabe konfigurieren
'0=keine feste Laufweite, Ausgabe wie in Bitmapfont definiert
'hier zB. 30=alle Zeichen sind 30 Pixel breit
myfont.monospaced=30
myfont.UseBackGround=0                                      'Hintegrundfarbe wieder ausschalten

'Ausgabe
myfont.Print (0,10,70, "Fixed Width Output")

'Konfig bleibt bei weiteren Ausgaben so bestehen
myfont.Print (0,10,100, "Still Fixed Width Output")
sleep
```
</code>
...sieht dann so aus:

MasterControlProgram

Das MCP ist das "Herzstück" der sGUI.
Es sammelt System-Ereignisse (Tastatur und Maustasten und Mausbewegungen) und verknüpft sie mit den erzeugten Controls und Fenstern der GUI.

Das MCP sorgt für den grafischen Aufbau der GUI, kopiert Screenseite 0 nach Screenseite 1, legt die GUI als "Overlay" auf Seite 1 und zeigt schlussendlich Seite 1 auf dem Screen an.

Das MCP muß in einer Schleifenkonstruktion aufgerufen werden.

Innerhalb der Schleifenkonstruktion stehen einige FB-Funktionen nur eingeschränkt zur Verfügung (INKEY, SCREENEVENT). In den Hinweisen sind alle sGUI-System-Ereignisse zu finden, die dann anstatt der FB-Funktionen ausgewertet/benutzt werden müssen.

*** Syntax:
sub MasterControlProgram (mode as integer=1)

*** Parameter:
mode wenn 1 wird die GUI auf den Screen "gezeichnet"
      wenn 0 wird die Ausgabe der GUI unterdrückt und muß via DrawScreen() erfolgen.

*** Rückgabewert:
keiner

*** Hinweise:
Nach Aufruf des MCPs stehen folgende sGUI-System-Ereigisse zur Verfügung:

***Keyboard**

KEY	as string	liefert das Zeichen
ASCCODE	as integer	entstpricht der Ausgabe der FB-Funktion "ASC(KEY)"
EXTENDED	as integer	manche Keys, z.B. Pfeiltasten, "liefern" ein vorangestelltes zweites Zeichen. Ist dies der Fall hat EXTENDED den Wert 255, sonst 0

***Maus**

MOUSEMOVED	as integer	ist 1 wenn Maus bewegt wurden, anderenfalls 0
MOUSEX	as short	X-Koordinate der Maus im Screen
MOUSEY	as short	Y-Koordinate der Maus im Screen
MOUSEDELTAX	as short	falls Bewegung der Maus in X-Richtung erfolgte, Differenz zur Position der Maus vom vorhergehenden Frame

MOUSEDELTAY	as short	falls Bewegung der Maus in Y-Richtung erfolgte, Differenz zur Position der Maus vom vorhergehenden Framem
LMB	as integer	Maustastenstatus, hier der linken(LMB) Maustaste. Es werden 4 Status unterschieden: HIT =2 Taste grad frisch gedrückt HOLD =3 Taste gehalten RELEASE =1 Taste grad losgelassen RELEASED =0 Taste ist losgelassen Der jeweilige Status hängt vom Status des vorhergehenden Frames ab. Ein HIT und ein RELEASE kann nur einen Frame dauern.
MMB	as integer	siehe LMB
RMB	as integer	siehe LMB
LMB_DOUBLE_CLICK	as integer	Doppelklick der linken Maustaste
MMB_DOUBLE_CLICK	as integer	Doppelklick der mittleren Maustaste
RMB_DOUBLE_CLICK	as integer	Doppelklick der rechten Maustaste
WHEEL	as integer	ist -1 wenn das Mausrad vom User weg bewegt wird ist 1 wenn das Mausrad zum User hin bewegt wird ist 0 falls keine Radbewegung erfolgte

***sGUI Ereignisse**

GADGETMESSAGE	as Gadget ptr	Zeiger des Gadgets, welches eine Aktion meldet
WINDOWMESSAGE	as sGUIWindow ptr	Zeiger des Fensters, in welchem sich das Gadget befindet, welches eine GADGETMESSAGE „gesendet“ hat
MENUID	as integer	ID des Menüeintrages, welcher angewählt wurde
MENUBASE	as sGUIList ptr	Zeiger auf die MENUBASE, also der Liste des angewählten Menüeintrages

***Screen**

SCREENCLOSEBUTTON	as integer	ist 1 falls vom User der CloseButton des Screens angeklickt wurde

DrawScreen

"normalerweise" übernimmt das MasterControlProgram auch die gesamte grafische Ausgabe auf dem Screen.
Folgenden Nachteil hat dies: alle grafischen Ausgaben ausserhalb von sGUI sind erst immer einen Frame (oder Schleifendurchlauf) später sichtbar.
Während die GUI also immer "frame-aktuell" ist, stammt jedoch das Aussehen des UserScreens (Screenseite 0) noch vom vorherigen Frame.
Darum gibt es die Möglichkeit, die Grafikausgabe getrennt vom MasterControlProgram (dieses dann aufgerufen mit dem Parameter 0) erfolgen zu lassen.
Erst wenn im aktuellem Frame alles(grafische) getan ist, wird die Ausgabe zusammengeschmiedet und dargestellt, via DrawScreen !!!

*** Syntax:
sub DrawScreen

*** Parameter:
/

*** Rückgabewert:
/

*** Hinweise:
/

AddWindow

Diese Funktion erzeugt ein Fenster

*** Syntax:
function AddWindow (parent as any ptr, PosX as integer, PosY as integer, WinWidth as integer, WinHeight as integer, Text as string="", WinFlags as integer=0) as sGUIWindow ptr

*** Parameter:

win	ein Zeiger auf ein Parent-Fenster, in welchem das zu erzeugende Fenster erscheinen soll falls das Fenster auf dem Screen erscheinen soll, muss 0 übergeben werden
PosX PosY	ist die Position dieses Fensters im Parent-Fenster
WinWidth WinHeight	dies bestimmt die Breite und Höhe des Fensters, genauer des Contentbereiches. Hat ein Fenster einen Rahmen so ist das gesamte Fenster größer
Text	der Titeltext des Fensters
Winflags	über die Flags werden bestimmte Eigenschaften des Fensters eingestellt wie Aussehen, Verhalten im Fensterstapel usw. Mehrere Flags werden entweder addiert oder mit OR logisch verknüpft, siehe Hinweise.

*** Rückgabewert:
Zeiger vom Typ sGUIWindow ptr auf das erzeugte Fenster

*** Hinweise:

ein übergebener Titeltext des Fensters setzt automatsch das GFX Flag WFLAG_TITLEBAR

zu den Flags
1.) FensterTyp

WFLAG_IMAGE	ein reines Imagefenster, ignoriert den Titeltext und schließt eventuell gesetzte Flags in 2.) und 3.) aus bzw. hebt diese auf

2.) Buttons & Menü

WFLAG_DRAGABLE	Fenster ist über die Titelzeile verschiebbar, setzt zusätzlich das GFX Flag WFLAG_TITLEBAR
WFLAG_CLOSEABLE	Fenster hat einen CloseButton, setzt zusätzlich das GFX Flag WFLAG_TITLEBAR
WFLAG_WARNIGNORE	reserviert, Funktion noch nicht realisiert

3.) GFX Elemente

WFLAG_FRAMED	das Fenster besitzt einen Rahmen
WFLAG_TITLEBAR	das Fenster besitzt eine Titelzeile, falls ein Titeltext definiert worden ist, wird diese Flag automatisch gesetzt
WFLAG_MENUBAR	das Fenster besitzt eine Menüzeile, notwendig wenn ein Menü hinzugefügt werden soll
WFLAG_MINIMISABLE	reserviert, Funktion nicht realisiert
WFLAG_NOSHADOW	schaltet den Schatteneffekt um das Fenster herum aus

4.) Stapelverhalten, GUI-Lock
nur **eines** der folgenden Flags **muss bzw. darf** gesetzt sein, anderenfalls gilt WFLAG_STACKABLE

WFLAG_STACKABLE	Fenster kann sich im Fensterstapel bewegen, dies hängt natürlich vom Stapelverhalten der anderen Fenster ab
WFLAG_BACKDROP	Fenster ist immer "unten" im Stapel
WFLAG_ALWAYSONTOP	Fenster ist immer oben auf dem Fensterstapel
WFLAG_WARN	Fenster ist oben und auch nur dieses Fenster hat Funktionalität, andere Fenster und deren „Inhalt“ sind gesperrt. Derzeit können nicht alle Gadgetarten in einem solchen Fenster dargestellt werden, als Beispiel hier die DropDownListBox: deren PullDown ist selbst ein Fenster, welches aber nun nicht mehr funktioniert, auf Grund des WARN-Flags seines Parentfensters.
WFLAG_WARNIGNORE	reserviert, Spezialflag, noch nicht funktionsfähig

5.) Focus oder Selektion
Vorab sein gesagt, die Selektion ist in sGUI2 eher als eine grafische Eigenschaft zu sehen:
wenn ein Mausklick in einem Fenster erfolgt, ist dieses in der Regel selektiert oder hat im weitesten Sinn auch den Focus.
Es kann nur immer ein Fenster den Focus haben, unabhängig in welcher Hierachie-Ebene es sich befindet.
Falls nicht durch entsprechende Flags beeinflusst und ein Rahmen vorhanden ist, werden Fenster bei Selektion farblich hervorgehoben dargestellt.
Dies kann nun optisch zu "unschönen Effekten" führen, beispielsweise verlieren Fenster ihren Focus wenn ein (rahmenloses) Child-Fenster in seinem inneren angeklickt wurde usw...
Um etwas besseren Einfluss darauf zu haben gibt es nun folgende Flags...

5.1) "Focus"
folgende Flags beeinflussen direkt die Selektion bei einem MausKlick.

WFLAG_PARENTSELECT	bei einem MausKlick ins Fenster wird nicht dieses selbst, sondern dessen Parent-Fenster selektiert. Die Selektion wird "nach oben durchgereicht", sinnvoll bei Child-Fenstern
WFLAG_NOSELECT	bei einem MausKlick ins Fenster erfolgt keine Selektion. Eine möglich bestehende Selektion eines anderen Fensters bleibt weiterhin erhalten

5.2.) Darstellung
folgende Flags beeinflussen das Aussehen, unabhängig davon ob das Fenster selekiert/unselektiert ist

WFLAG_APPEARSSELECTED	Fenster wird permanent selektiert dargestellt
WFLAG_APPEARSUNSELECTED	Fenster wird permanent unselektiert dargestellt

6.) sonstige

WFLAG_HIDECLICKEDOUTSIDE	bei einem Klick **ausserhalb** des Fenster schließt es, anderenfalls bleibt es offen.
WFLAG_HIDECLICKEDINSIDE	bei einem Klick **ins** Fenster schließt es, anderenfalls bleibt es offen.
WFLAG_DELETEONCLOSE	löscht das Fenster und dessen Child-Objekte wenn sein CloseButton angeklickt wurde. Vorsicht, nicht ausgiebig getestet!

ShowWindow

Dieser Befehl „aktiviert“ ein Fenster oder einfacher gesagt, macht es sichtbar und benutzbar.
Generell gilt, Fenster sind nach der Erzeugung nicht sichtbar und müssen mit diesem Befehl sichtbar gemach werden.

*** Syntax:
sub ShowWindow (win as sGUIWindow ptr)

*** Parameter:

win	ein Zeiger auf ein Fenster, welches nun sichtbar sein soll

*** Rückgabewert:
/

*** Hinweise:
/

HideWindow

Dieser Befehl „deaktiviert“ ein Fenster oder einfacher gesagt, macht es unsichtbar.

Das Fenster kann jederzeit mit ShowWindow wieder sichtbar gemacht werden, es wird also nicht gelöscht

*** Syntax:
sub HideWindow (win as sGUIWindow ptr)

*** Parameter:
win ein Zeiger auf ein Fenster, welches nun unsichtbar sein soll

*** Rückgabewert:
/

*** Hinweise:
/

GetUserImage

Normalerweise sind Fenster nur mit Controls gefüllt, so ist es zumindest gedacht.

Falls man neben den Controls auch eigene grafische Ausgaben im Fenster tätigen will, benötigt man fürs Fenster ein weiteres Image, auf dem sich der User „austoben“ darf

GetUserImage liefert den Zeiger auf ein solches FB.Image.

Ein mit WFLAG_IMAGE erzeugtes Fenter besitzt dagegen nur ein Layer. Gadgets sollten in ihm nicht platziert werden, es ist ein reines „User Grafik Fenster“.
Es ist mehr oder weniger ein „mit einer Fensterstruktur umwickeltes Image“.

So kann das Image ein paar Fenstereigenschaften „erben“ und verhält sich entsprechend auch wie ein Fenster.

***Syntax
function GetUserImage (win as _sGUIWindow ptr=0) as FB.Image ptr

*** Parameter:
win ein Zeiger auf ein Fenster, welches nun unsichtbar sein soll

*** Rückgabewert:
ein Zeiger auf ein FB.Image

*** Hinweise:
/

SimpleGadget

Ein einfacher Button

*** Syntax:
function AddSimpleGadget (win as sGUIWindow ptr, PosX as integer, PosY as integer, GadWidth as integer=0, GadHeight as integer=0, Text as string) as Gadget ptr

*** Parameter:
win ein Zeiger auf ein Fenster, in welchem das SimpleGadget erscheinen soll
falls das SimpleGadget auf dem Screen erscheinen soll, muss 0 übergeben werden

PosX ist die Position des SimpleGadgets im Parent-Fenster
PosY

GadWidth dies beschreibt die Breite und Höhe des Simplegadgets, falls hier keine Werte (=0) übergeben werden,
GadHeight bestimmt der Text die Breite bzw. Höhe der Box

Text ein Text der im Control angezeigt wird

*** Rückgabewert:
Zeiger vom Typ Gadget ptr auf das erzeugte Gadget

*** Hinweise:
Einfache Buttons bieten die Möglichkeit einer alternativen Farbgebung!
[gad]→xtd.btn.UseButtonColors wenn diese Eigenschaft =1 dann wird die Zeichenroutine nicht die „globalen“ Farben benutzen, sondern die folgenden vier im Control hinterlegten Farben.
Diesen sollten dann natürlich auch noch RGB-Werte zugewiesen werden.
[gad]->xtd.btn.GadBody
[gad]->xtd.btn.GadText
[gad]->xtd.btn.GadBodySelected
[gad]->xtd.btn.GadTextSelected

[gad]->xtd.btn.Fire Hiermit wird das Klickverhalten geändert: wenn Fire=1 triggert das Control solang es gehalten wird
[gad]->xtd.btn.BtnWindow Hier kann ein beliebiges Fenster ans Gadget "gebunden" werden, es wird bei einem Klick aufs Gadget geöffnet. Mehr passiert dabei nicht.
Entspricht also einem **ShowWindow([gad]→xtd.btn.BtnWindow)**

ToggleGadget

Ein Options-Button der, bis zum nächsten Klick darauf, seinen Selektionszustand beibehält

*** Syntax:
function AddToggleGadget (win as SGUIWindow ptr, PosX as integer, PosY as integer, GadWidth as integer=0, GadHeight as integer=0, Selection as integer, Text as string) as Gadget ptr

*** Parameter:

win	ein Zeiger auf ein Fenster, in welchem das ToggleGadget erscheinen soll falls das ToggleGadget auf dem Screen erscheinen soll, muss 0 übergeben werden
PosX PosY	ist die Position des ToggleGadgets im Parent-Fenster
GadWidth GadHeight	dies beschreibt die Breite und Höhe des Togglegadgets, falls hier keine Werte (=0) übergeben werden, bestimmt der Text (Text) die Größe der Box
Text	ein Text der im Control angezeigt wird

*** Rückgabewert:
Zeiger vom Typ Gadget ptr auf das erzeugte Gadget

*** Hinweise:
Das Gadget ist nach Erzeugung im unselektierten Zustand und muß, wenn gewünscht, mit SetSelect() entsprechend konfiguriert werden

Auch hier gibt es die Möglichkeit einer alternativen Farbgebung!

[gad]→xtd.btn.UseButtonColors	wenn diese Eigenschaft =1 dann wird die Zeichenroutine nicht die „globalen“ Farben benutzen, sondern die folgenden vier im Control hinterlegten Farben. Diesen sollten dann natürlich auch noch RGB-Werte zugewiesen werden.
[gad]->xtd.btn.GadBody	
[gad]->xtd.btn.GadText	
[gad]->xtd.btn.GadBodySelected	
[gad]->xtd.btn.GadTextSelected	

CheckMarkGadget

Vergleichbar und im Verhalten gleich dem ToggleGadget. Ein Options-Button der, bis zum nächsten Klick darauf, seinen Selektionszustand beibehält.

*** Syntax:
function AddCheckmarkGadget (win as sGUIWindow ptr, PosX as integer, PosY as integer, Text as string="", Textleft as integer=0) as Gadget ptr

*** Parameter:

win	ein Zeiger auf ein Fenster, in welchem das CheckMarkGadget erscheinen soll falls das CheckMarkGadget auf dem Screen erscheinen soll, muss 0 übergeben werden
PosX PosY	ist die Position des CheckMarkGadgets im Fenster
Text	ein Text der im Control angezeigt wird, der sensible klickbare Bereich der Gadgets erweitert sich dann um den Textbereich
Textleft	wenn 1 dann wird der Text links vom Symbol angezeigt

*** Rückgabewert:
Zeiger vom Typ Gadget ptr auf das erzeugte Gadget

*** Hinweise:
Das Gadget ist nach Erzeugung im unselektierten Zustand und muß, wenn gewünscht, mit SetSelect() entsprechend konfiguriert werden

RadioButton

RBs sind Optionsbuttons wobei immer nur ein Button einer Gruppe selektiert sein kann

*** Syntax:
function AddRadioButton (win as sGUIWindow ptr, PosX as integer, PosY as integer, Text as string="", Head as Gadget ptr, Textleft as integer=0) as Gadget ptr

*** Parameter:

win	ein Zeiger auf ein Fenster, in welchem das RadioButton erscheinen soll falls das RadionButton auf dem Screen erscheinen soll, muss 0 übergeben werden
PosX	ist die Position des RadionButtons im Parent-Fenster

PosY

Text ein Text der im Control angezeigt wird, der sensible klickbare Bereich erweitert sich um die Textbreite

Head wird 0 übergeben so stellt diese RadioButton den "Kopf" einer Radiobuttongruppe dar. In ihm wird eine Liste angelegt welche alle weiteren Mitglieder der Gruppe enthält. Soll ein weiterer RadioButton einer Gruppe hinzugefügt werden, so gibt man bei Head den Gadget Pointer zum ersten RadioButton der Gruppe an.

Textleft ist dieser Parameter 1 dann wird der Text links vom Symbol angezeigt

*** Rückgabewert:
Zeiger vom Typ Gadget ptr auf das erzeugte Gadget

*** Hinweise:
Nach Erzeugung einer RB-Gruppe ist keines der RButtons vorselektiert!
Das Selektieren eines RadioButtons dieser Gruppe muß im Anschluß mit SetSelect() erfolgen.

Bei Bedienung eines RadioButtons durch den User erfolgt automatisch das Anpassen der anderen Mitglieder der Gruppe.
Ergibt sich allerdings im Programmverlauf, dass eine bestimmte Selektion "erzwungen" werden muss, dann muss dies mit **allen** Mitgliedern einer Gruppe **einzeln** per SetSelect() erfolgen.
Nochmal mit anderen Worten: SetSelect() kann am zu „manipulierenden“ Control nicht erkennen, ob es sich dabei um ein RadioButton oder gar um eine RB-Gruppe handelt, es setzt lediglich **dessen (und nur dessen!)** Selektions-Status.

RadioToggle

Erzeugt ein RadioToggle bzw eine RadioToggleGruppe.
RTs sind Optionsbuttons wobei immer nur ein Button einer Gruppe selektiert sein kann.
Eine RadioToggleGruppe soll bei der Erstellung eines Property-Dialoges helfen und soll dort die Funktion der Seiten-Reiter erfüllen. An jedes der Rts einer Gruppe kann ein Fenster gekoppelt werden.

*** Syntax:
function AddRadioToggle (win as sGUIWindow ptr, PosX as integer, PosY as integer, GadWidth as integer, GadHeight as integer, Text as string, Head as Gadget ptr) as Gadget ptr

*** Parameter:
win ein Zeiger auf ein Fenster, in welchem das RadioToggle erscheinen soll
falls das RadioToggle auf dem Screen erscheinen soll, muss 0 übergeben werden

PosX ist die Position des RadioToggles im Parent-Fenster
PosY

GadWidth diese beschreiben die Breite und Höhe des RadioToggles
GadHeight

Text ein Text der im Control angezeigt wird

Head wird 0 übergeben so stellt dieses RadioToggle den "Kopf" einer RadioTogglegruppe dar. In ihm wird eine Liste angelegt welche weitere Mitglieder der Gruppe enthält.
Soll ein weiteres RadioToggle einer Gruppe hinzugefügt werden, so gibt man bei Head den Gadget Pointer zum ersten RadioToggle der Gruppe an.

*** Rückgabewert:
Zeiger vom Typ Gadget ptr auf das erzeugte Gadget

*** Hinweise:
[gad]->xtd.rbc.RTWindow =[win]
Ihr kann ein Zeiger zu einem Fenster übergeben werden.
Beim Klicken auf ein RadioToggle wird dann das so gekoppelte Fenster angezeigt, andere in der Gruppe gekoppelte Fenster werden geschlossen.

Bei Bedienung eines RadioToggles durch den User erfolgt automatisch das Anpassen der anderen Mitglieder der Gruppe
Wie man aber an den Parametern sieht, ist eine Vorselektierung nicht vorgesehen.
Es muß also nach der Erstellung einer Gruppe die Selektierung eines Mitgliedes mit SetSelect() erfolgen.

Generell gilt: ergibt sich im Programmverlauf daß eine bestimmte Selektion "erzwungen" werden muß, dann muß dies mit allen Mitgliedern einer Gruppe einzeln per SetSelect() erfolgen.
Die Sichtbarkeit der gekoppelten Fenster muss dann auch „händisch“ mit ShowWindow()/HideWindow() erfolgen.

GetSelect

Mit dieser Funktion erhält man den Selektionsstatus eines Gadgets.
Eine sinnvolle Verwendung ist nur bei folgenden Gadgettypen möglich:
*ToggleGadget
*CheckMark
*Radiobutton
*RadioToggle

***Syntax:
function GetSelect (gad as _Gadget ptr) as integer

*** Parameter:
gad ein Zeiger auf ein Gadget

*** Rückgabewert:
0 oder 1 0= das Gadget ist nicht selektiert, 1=es ist selektiert

*** Hinweise:

SetSelect

Mit dieser Funktion setzt man den Selektionsstatus eines Gadgets.
Eine sinnvolle Verwendung ist nur bei folgenden Gadgettypen möglich:
*ToggleGadget
*CheckMark
*Radiobutton
*RadioToggle

***Syntax:
sub SetSelect (gad as _Gadget ptr, Selection as integer=1)

*** Parameter:
gad ein Zeiger auf ein Gadget
Selection 0=das Gadget soll nicht selektiert sein, 1=es soll selektiert sein (voreingestellt)

*** Rückgabewert:

*** Hinweise:

Arrow

Arrows sind einfache Buttons mit einem Pfeilsymbol. Beim Halten triggern sie.

*** Syntax:
function AddArrow (win as sGUIWindow ptr, PosX as integer, PosY as integer, DirArrow as integer) as Gadget ptr

*** Parameter:
win ein Zeiger auf ein Fenster, in welchem der/das Arrow erscheinen soll
falls der/das Arrow auf dem Screen erscheinen soll, muss 0 übergeben werden

PosX ist die Position der/des Arrows im Parent-Fenster
PosY

DirArrow bestimmt die Richtung des Pfeilsymbols
0=links
1=rechts
2=hoch
3=runter

*** Rückgabewert:
Zeiger vom Typ Gadget ptr auf das erzeugte Gadget

*** Hinweise:

StringGadget

StringGadget ist ein einzeiliges Eingabefeld.

*** Syntax:
function AddStringGadget(win as sGUIWindow ptr, PosX as integer, PosY as integer, GadWidth as integer, MaxChars as integer=0, CharLimitation as integer=0, ShowEnd as integer=0, Focus as integer=0, Mode as integer=0) as Gadget ptr

*** Parameter:
win ein Zeiger auf ein Fenster, in welchem das StringGadget erscheinen soll
falls das StringGadget auf dem Screen erscheinen soll, muss 0 übergeben werden

PosX ist die Position des StringGadgets im Parent-Fenster
PosY

GadWidth dies beschreibt die Breite des StringGadgets, die Höhe ist abhängig vom benutzten Font

MaxChars eine Begrenzung der Anzahl der Zeichen die eingegeben werden kann, 0 entspricht keiner Längenbegrenzung

CharLimitation hierüber kann die Art der "zugelassenen" Zeichen begrenzt werden
0= keine Beschränkung, voreingestellt
1= nur Zeichen die dezimale Integerzahlen darstellen
2= nur Zeichen die dezimale Fließkommazahlen darstellen
3= nur Zeichen die Binärzahlen darstellen

	4= nur Zeichen die Hexadezimalzahlen darstellen 5= alte(!) IP Adressen 6= „freier“ Zeichenfilter, mittels einer zusätzlichen Routine können selbst die erlaubten Zeichen definiert werden
ShowEnd	hier kann bei "überlangem" Inhalt bestimmt werden, welches "Ende" wichtiger ist und angezeigt wird, solang nicht editiert wird 0= Anfang des Strings, voreingestellt 1= Ende des Strings
Focus	normalerweise ist nach einem "Return" die Eingabe in ein StringGadget beendet. Bei Focus=1 kann weiter editiert werden, solang kein anderes Gadget (oder Fenster) angeklickt wurde.
Mode	0=Editiermodus, Text kann bearbeitet werden 1=ViewModus, Text wird nur angezeigt

*** Rückgabewert:
Zeiger vom Typ Gadget ptr auf das erzeugte Gadget

*** Hinweise:
Generell ist zu beachten das die verwaltete Eingabe vom Typ immer ein String ist. Wer also ein StringGadget ausliest und den Inhalt als Zahlentyp benötigt, muss entsprechende Konvertierungen vornehmen.
Im umgekehrten Fall gilt dies genauso, ein Zahlenwert muß zu einem String konvertiert werden.
Nach Erzeugung ist das Gadget leer.
Für das Setzen und Auslesen des StringGadgets stehen die Routinen GetString() und SetString() zur Verfügung

SetString

Setzt einen Inhalt eines StringGadgets

***Syntax:
sub SetString (gad as Gadget ptr, Text as string)

*** Parameter:

gad	ein Zeiger auf ein StringGadget
Text	Text, der im StringGadget angezeigt werden soll

*** Rückgabewert:
/

*** Hinweise:
/

GetString

Liefert den Inhalt eines StringGadgets

***Syntax:
function GetString (gad as Gadget)

*** Parameter:

gad	ein Zeiger auf ein StringGadget

*** Rückgabewert:
ein String, Inhalt des StringGadgets

*** Hinweise:
/

SetAllowedChars

Setzt einen selbst definierten Zeichenfilter eines StringGadgets

***Syntax:
sub SetAllowedChars (gad as Gadget ptr, allowed as string)

*** Parameter:

gad	ein Zeiger auf ein StringGadget
allowed	ein String der alle Zeichen enthält, die ins StringGadget eingegeben werden dürfen

*** Rückgabewert:
ein String, Inhalt des StringGadgets

*** Hinweise:
Das StringGadget muss beim Parameter CharLimitation mit Modus 6 “betrieben“ werden

TextField

Das TextField ist im Grunde ein ganz einfacher Editor, man kann mehrzeiligen Text eingeben und rudimentär bearbeiten.

*** Syntax:
function AddTextField(win as sGUIWindow ptr, PosX as integer, PosY as integer, BoxWidth as integer, BoxHeight as integer, Mode as integer=0) as Gadget ptr

*** Parameter:

win	ein Zeiger auf ein Fenster, in welchem das TextField erscheinen soll falls das TextField auf dem Screen erscheinen soll, muss 0 übergeben werden
PosX PosY	ist die Position des TextField im Parent-Fenster
BoxWidth BoxHeight	dies beschreibt die Breite und Höhe des TextFields
Mode	Darstellungsmodus 0=Editiermodus, Text kann bearbeitet werden 1=ViewModus, Text wird nur angezeigt

*** Rückgabewert:
Zeiger vom Typ Gadget ptr auf das erzeugte Gadget

*** Hinweise:
… hier muss ich nun etwas weiter ausholen, also …
Das TextField enthält, wie auch das StringGadget, ein UTD(User Defined Type) mit ein paar Methoden(Routinen). Diesen UDT verwaltet den mehrzeiligen Text, kann ihn editieren und enthält Routinen für das Laden und Speichern des Textes. Auch sGUI selbst „kommuniziert“ nur über diese Methoden mit dem UDT, hier sind es dann die Methoden für die Cursorsteuerung und das Editieren.
Wie komme ich nun an diese Methoden?

[gad]→xtd.Tcontainer→(Methode(Parameter))

[gad] stellt den Zeiger auf unser TextField dar, dazu kommt etwas Referenzierungszauber plus den Namen der Methode und möglichen Parametern.

Hier die wichtigsten Methoden:
Laden ...
aus Datei
[gad]→xtd.Tcontainer→LoadText (filename as string, linebreak as string=chr(13,10))

filename	Dateiname der zu ladenen Datei
linebreak	falls die Datei eine „exotische“ Zeichenkombination als Zeilenumbruch benutzt, hiermit übergeben

aus einem String
[gad]→xtd.Tcontainer→SetText (s as string, linebreak as string=chr(13,10))

s	ein String der irgendeinen Text enthält
linebreak	falls der String eine „exotische“ Zeichenkombination als Zeilenumbruch benutzt, hiermit übergeben

Speichern ...
in eine Datei
[gad]→xtd.Tcontainer→SaveText (filename as string, linebreak as string=chr(13,10))

filename	Dateiname der zu speichernde Datei
linebreak	falls die Datei eine „exotische“ Zeichenkombination als Zeilenumbruch benutzen soll, hiermit übergeben

in einen String (als Funktion)
[gad]→xtd.Tcontainer→GetText (linebreak as string=chr(13,10)) as string

linebreak	falls die Datei eine „exotische“ Zeichenkombination als Zeilenumbruch benutzen soll, hiermit übergeben

Abfrage Zeileninhalt als Funktion
[gad]→xtd.Tcontainer→GetRowContent (i as integer) as string

i	Zeilennummer

'Ersetzen des Zeileninhaltes einer bestimmten Zeile
[gad]→xtd.Tcontainer→SetRowContent (i as integer, t as string)

i	Zeilennummer
t	String(Text) der den bisherigen ersetzen soll/wird

neue Zeile (mit Inhalt) am Textende anhängen
[gad]→xtd.Tcontainer→AppendRow (t as string="")

t	Text, den die neue Zeile enthalten soll

Text komplett löschen
[gad]→xtd.Tcontainer→ClearText

****

Wer mehr Methoden nutzen möchte sollte sich die public Methoden des UDTs „sGUIText“ in der Datei „sGUI\external\ sGUIText.bi“ genauer ansehen.

ListBox

Die ListBox ist eine statische AuswahlBox mit Einträgen. Je nach Konfiguration können ein oder mehrere Einträge darin selektiert/deselektiert werden.

*** Syntax:
function AddListBox(win as sGUIWindow ptr, PosX as integer, PosY as integer, BoxWidth as integer, BoxHeight as integer, DisplayMode as integer=0, SelectionMode as integer=0) as Gadget ptr

*** Parameter:

win	ein Zeiger auf ein Fenster, in welchem die ListBox erscheinen soll falls die ListBox auf dem Screen erscheinen soll, muss 0 übergeben werden
PosX PosY	ist die Position der ListBox im Parent-Fenster
GadWidth GadHeight	dies beschreibt die Breite und Höhe der ListBox
DisplayMode	hierüber wird die Darstellung der Einträge in der ListBox konfiguriert 0= selektierte Eintrage werden farblich hervorgehoben, Text=Colors.Colors.GadTextSelected und der Hintergrund=Colors.GadBodySelected 1= selektierten Einträgen wird ein Häkchen vorangestellt 2= optimiert für den Fall daß die ListBox den Inhalt eines Festplattenverzeichnisses darstellen soll. Setzt zusätzliche Daten innerhalb der Einträge voraus. 3= hier ist es möglich, Einträge als nicht anklickbare Labels zu definieren. Setzt zusätzliche Daten innerhalb der Einträge voraus
SelectionMode	hierüber wird das Selektionsverhalten eingegestellt 0= Single Select: es kann vom User nur ein Eintrag selektiert werden. Das Anklicken eines anderen Eintrages führt zum Wechsel der Selektion. Eine Deselektion durch nochmaliges Anklicken ist nicht möglich. Solang die ListBox vom Program-User benutzt wird, ist also immer ein Eintrag selektiert. Wobei erwähnt sein sollte das auch in diesem Modus ein "Nichts ist selektiert"-Zustand möglich ist. 1= Single Deselectable: es kann vom User nur ein Eintrag selektiert werden. Das Anklicken eine anderen Eintrages führt zum Wechsel der Selektion. Wird ein bereits selektierter Eintrag nochmalig angeklickt, wird dessen Selektion aufgehoben. Es kann sowohl keiner oder maximal ein Eintrag selektiert sein. 2= Multi Deselectable: es können vom User mehrere Einträge selektiert werden. Wird ein bereits selektierter Eintrag nochmalig angeklickt, wird dessen Selektion aufgehoben. Es können beliebig viele Eintrage selektiert sein.

*** Rückgabewert:
Zeiger vom Typ Gadget ptr auf das erzeugte Gadget

*** Hinweise:
Wenn ListBox und Einträge erstellt worden sind, sollte generell eine entsprechende (Vor-)Selektierung mit SetListBox() erfolgen.

SetListBox

Selektiert/Deselektiert einen Eintrag einer ListBox

*** Syntax:
sub SetListBox(gad as Gadget ptr, i as integer,Selection as integer=1)

*** Parameter:

gad	ein Zeiger auf eine ListBox
i	Indexnummer des Eintrages dessen Selektion geändert werden soll
Selection	1=Eintrag soll selektiert sein 0=Eintrag soll nicht selektiert sein

*** Rückgabewert:
/

*** Hinweise:
Nach Erzeugung einer ListBox und deren Einträge ist gegebenenfalls eine entsprechende Selektion mit diesem Befehl vorzunehmen

GetListBox

Liefert einen (genauer den ersten) oder weitere Indizes selektierter Einträge einer ListBox
Dient also der Abfrage einer ListBox

*** Syntax:

function GetListBox(gad as Gadget ptr, index as integer=0) as integer

*** Parameter:
gad ein Zeiger auf eine ListBox

index wenn>0 dann stellt dies die Indexnummer selektierten Eintrages dar.

*** Rückgabewert:
eine Indexnummer eines selektierten Eintrages. Falls kein Eintrag selektiert ist wird eine 0 rückgegeben

*** Hinweise:
Diese einfache Schleifenkonstruktion fragt Mehrfachselektionen ab:
<code>
index=GetListBox(lbx)
do
	'if index then
	'dann mach dies oder das
	'end if

	index=GetListBox(lbx,index)
loop until index=0
</code>
Wenn index=0 dann wurde kein (weiterer) selektierter Eintrag gefunden und der Loop wird verlassen

SetEntryActivation

Setzt die Activation eines Eintrages

*** Syntax:
sub SetEntryActivation overload(gad as Gadget ptr, Index as integer, Activation as integer=1)

*** Parameter:
gad ein Zeiger auf eine ListBox oder DropDownListBox

Index Index des zu ändernden Eintrages

Activation neue Aktivation, 0= Eintrag nicht anwählbar, 1= Eintrag anwählbar

*** Rückgabewert:
/

*** Hinweise:
/

DropDownListBox

Erzeugt eine DropDownListBox(früher ComboBox genannt, aber nach Wiki-Definition keine ist) mit Einträgen. Es kann nur ein Eintrag darin selektiert sein.

*** Syntax:
function AddDropDownListBox (win as sGUIWindow ptr,PosX as integer,PosY as integer,GadWidth as integer,GadHeight as integer,ListBoxWidth as integer=200,ListBoxHeight as integer=100,mode as integer=0) as Gadget ptr

*** Parameter:
win ein Zeiger auf ein Fenster, in welchem die DropDownListBox erscheinen soll
falls die DropDownListBox auf dem Screen erscheinen soll, muss 0 übergeben werden

PosX ist die Position der DropDownListBox im Parent-Fenster
PosY

GadWidth diese beschreiben die Breite und Höhe der DropDownListBox
GadHeight

ListBoxWidth diese beschreiben die Breite und Höhe der aufgeklappten ListBox.
ListBoxHeight

mode Bedienkonzept des Controls. Zum einen gibt es die Möglichkeit eines Auswahlfensters, in dem man den entsprechende Eintrag auswählt, "very common" denke ich.
Die andere Bedienvariante besteht in einer Art von rotierenden Eintragsliste.
Wer noch das Amiga-OS kennt, wird sich an das Cycle-Gadget erinnern :).
Man muß so oft ins Control zu klicken, bis der entsprechende Eintrag erscheint. Dies ist nur sinnvoll bei relativ wenigen Einträgen.

0 (voreingestellt) ein Auswahlfenster wird aufgeklappt und einer der Einträge kann nun (angescrollt und..) ausgewählt werden

1 Mix aus beidem...
links: Auswahlfenster
rechter Symbolbereich: rotierende Liste

2 noch ein Mix aus beidem...
links: rotierende Liste
rechter Symbolbereich: Auswahlfenster

3 nur rotierende Liste

*** Rückgabewert:
Zeiger vom Typ Gadget ptr auf das erzeugte Gadget

*** Hinweise:
Wenn DropDownListBox und Einträge erstellt worden sind, sollte generell eine entsprechende (Vor-)Selektierung mit SetDDListBox() erfolgen.
Falls gewünscht, kann die Box allerdings auch einen "nichts ist selektiert" Zustand haben.

SetDDListBox

Selektiert/Deselektiert einen Eintrag einer ListBox

*** Syntax:
sub SetDDListBox(gad as Gadget ptr,i as integer, Selection)

*** Parameter:
gad ein Zeiger auf eine ListBox

i Indexnummer des Eintrages dessen Selektion geändert werden soll

Selection 1=Eintrag soll selektiert sein
0=Eintrag soll nicht selektiert sein

*** Rückgabewert:
/

*** Hinweise:
Nach Erzeugung einer ListBox und deren Einträge ist gegebenenfalls eine entsprechende Selektion mit diesem Befehl vorzunehmen.

GetDDListBox

Liefert den Index des selektierten Eintrages einer DropDownListBox

*** Syntax:
function GetDDListBox(gad as Gadget ptr) as integer

*** Parameter:
gad ein Zeiger auf eine ListBox

*** Rückgabewert:
eine Indexnummer eines selektierten Eintrages. Falls kein Eintrag selektiert ist, wird eine 0 rückgegeben

*** Hinweise:
/

SetEntryActivation

Setzt die Activation eines Eintrages

*** Syntax:
sub SetEntryActivation overload(gad as Gadget ptr, Index as integer, Activation as integer=1)

*** Parameter:
gad ein Zeiger auf eine ListBox oder DropDownListBox

Index Index des zu ändernden Eintrages

Activation neue Aktivation, 0= Eintrag nicht anwählbar, 1= Eintrag anwählbar

*** Rückgabewert:
/

*** Hinweise:
/

FileRequester

Filerequester sind in sGUI als Gadget(mit eigenem Fenster) konzipiert und können somit auch wie ein Gadget in einem laufenden Eventloop abgefragt werden.
Es ist dadurch möglich mehrere unterschiedlich konfigurierte Filerequester zu erzeugen

*** Syntax:
function AddFileRequester (doit as string="doit", cancel as string="cancel", Mode as integer=0) as Gadget ptr

*** Parameter:

doit	Text des "AktionsButtons"
cancel	Text des AbbruchButtons
Mode	...genauer FEXISTS-Mode! wenn =1 wird der Pfad+Dateiname über die FBeigene FEXISTS() Funktion auf deren Gültigkeit überprüft. Der FileRequester kann nur über mit einem gültigen Dateipfad oder dem Abbruchbutton verlassen werden. Im Falle einer Ungültigkeit "poppt" ein einfaches Warnfenster auf! wenn =0 wird Pfad+Dateiname nicht überprüft

*** Rückgabewert:
Zeiger vom Typ Gadget ptr auf das erzeugte Gadget

*** Hinweise:

ShowFileRequester

Das Anzeigen des FileRequesters erfolgt über den Befehl ShowFileRequester.
Ein über diesen Befehl geöffneter Filerequester benötigt einen laufenden
Eventloop mit dem MasterControlProgram

Der Vorteil dieses Aufrufs liegt in der Möglichkeit trotz geöffnetem Filerequesters weitere
GUI externe Sachen im Eventloop ablaufen zu lassen

*** Syntax:
sub ShowFileRequester (gad as Gadget ptr,Path as string="",Filename as string="")

*** Parameter:

gad	Zeiger auf einen Filerequester
Path	Pfad eines Verzeichnisses das im Filerequester angezeigt werden soll
Filename	möglicher voreingesteller Name einer Datei

*** Rückgabewert:
/

*** Hinweise:
/

GetFileRequester

nach Verlassen eines Filerequesters wir ein entsprechendes Ereignis (GADGETMESSAGE) erzeugt, auf welches man dann reagieren kann
Mit GetFileRequester() wird der Filerequester ausgelesen.

*** Syntax:
function GetFileRequester (gad as Gadget ptr) as string

*** Parameter:

gad	Zeiger auf einen Filerequester

*** Rückgabewert:
ein String der entweder einen kompletten Pfad+Dateiname oder einen Leerstring enthält

*** Hinweise:
/

OpenFileRequester

ein Filerequester als Funktion
Die Erzeugung bzw. dadurch auch die Grundkonfiguration eines oder mehrerer(!) FileRequester mittels AddFileRequester() ist notwendig. Dies ist im Grunde genommen "nur" ein in eine Funktion gekapselter EventLoop in dem das MasterControlProgram aufgerufen wird.

Diese Version ist für Anwender gedacht, die lediglich einen Filerequester in ihrem Programm benötigen, aber sonst weiter nichts mit sGUI zu tun haben möchten.
Um einen "SGUI Screen" kommt man allerdings nicht herum!

*** Syntax:
function OpenFileRequester(gad as Gadget ptr,Path as string=curdir,Filename as string="") as string

*** Parameter:
gad ein Zeiger auf ein zuvor erzeugten FileRequester

Path ein Dateipfad den der Filerequester anzeigen soll

Filename eine voreingestellter Dateiname

*** Rückgabewert:
liefert einen String der, bei erfolgreichem Beenden des FRs, einen Pfad+Dateiname enthält. Anderenfalls ist der zurückgegebene String leer.

*** Hinweise:

'/

PopUpMenü / PullDownMenü

Diese Features sind bisher noch nicht komplett programmiert worden, aber Teile davon existieren bereits und sind auch benutzbar!
Wie erstellt man Menüs?
Zuerst benötigt man die MenüBasis. Dies ist in erster Linie eine Liste, die eine Art „Beschreibung“ über den Aufbau des Menüs enthält. In ihr werden alle Einträge, CheckMarks und SubMenüs und deren Eigenschaften definiert.
Diese Liste muss einem bestimmten Syntax entsprechen.
Aus dieser Liste, wenn man so will - der Menübeschreibung, wird dann das eigentliche Menü mittels entsprechenden Befehl erstellt.
Wichtig hierbei ist: jedes Menü hat seine eigene MenüBasis, anders gesagt: aus einer MenüBasis darf auch nur ein Menü erstellt werden!

NewMenuBase

Erzeugt eine sGUIListe welche später mittels entsprechender Einträge eine Menübeschreibung enthalten wird

*** Syntax:
function NewMenuBase as sGUIList ptr

*** Parameter:
/

*** Rückgabewert:
ein Zeiger vom Typ sGUIList ptr

*** Hinweise:
Dies ist eine Funktion. Man sollte also eine entsprechend dimensionierte Variable bereitstellen:
<code>
dim as sGUIList ptr meinMenu=NewMenuBase
</code>
Die Beschreibung des Menüs darauffolgend benötigt den Zeiger auf diese Liste

Item, CheckMark, RadioButton, Separator, OpenMenuStrip, CloseMenuStrip

Diese Befehle erzeugen entsprechende Einträge in einer MenuBase.

*** Syntax:
function Item(menulist as sGUIList ptr, txt as string, activation as integer=1) as integer
function CheckMark(menulist as sGUIList ptr, txt as string, activation as integer=1, selection as integer) as integer
function RadioButton (menulist as sGUIList ptr, txt as string, activation as integer=1, selection as integer, HeadID as integer) as integer
function Separator(menulist as sGUIList ptr) as integer
function OpenMenuStrip(menulist as sGUIList ptr, txt as string,activation as integer=1) as integer
function CloseMenuStrip(menulist as sGUIList ptr) as integer

*** Parameter:
menulist ein Zeiger auf eine MenuBase, dem dieser Eintag hinzugefügt wird

txt falls ein Eintrag ein Text enthalten kann, hier definieren

activation wenn 1=anwählbar, wenn 0=nicht anwählbar

selection wenn 1=selektiert , wenn 0=nicht selektiert dargestellt

HeadID dient im Falle eines RadioButtons der Zuordnung zu einer RB-Gruppe. 0 bedeutet dieser Radiobutton ist der erste RB einer Gruppe, also dessen „Kopf“. Sollen weiterere RBs der Gruppe zugeordnet werden, so muss HeadID die ID des Kopf-Radiobuttons enthalten. Die Art der Definition einer Gruppe ist im Prinzip vergleichbar mit dem Verfahren bei normalen RadioButtons.

*** Rückgabewert:

ein integer Wert, 1 wenn funktion erfolgreich ausgeführt wurde, anderenfalls 0

*** Hinweise:
zu den möglichen Einträgen:
Ein Item ist ein ganz normaler Menüpunkt
Ein CheckMark ist ein Item, das zwischen selektiertem oder nicht selektiertem Zustand wechseln kann.
Ein Separator ist im weitesten Sinn einfach nur ein Trennbalken.
Mit OpenMenuStrip/CloseMenuStrip lassen sich hierarchische Strukturen, wenn man so will Menübäume, definieren.

Es gibt, je nach Verwendungszweck, zwei Syntax-Varianten:

* **PullDownMenüs:** hier müssen Item/CheckMark/Separatoren **immer** von einem **Open-Close-Konstrukt** umschlossen sein. Ausserhalb sind diese Einträge nicht erlaubt.
Durch Verschachtelung von **Open-Close-Konstrukte** können UnterMenüs erzeugt werden.

* **PopUpMenüs:** hier müssen lediglich UnterMenüs mit einem **Open-Close-Konstrukt** definiert werden.

CreatePopUpMenu

Erzeugt aus einer MenüBase (mit entprechenden Einträgen) ein PopUpMenü. Es werden im Hintergrund Fenster und Gadgets auf Grundlage der MenuBase herstellt.

*** Syntax:
sub CreatePopUpMenu(menulist as sGUIList ptr)

*** Parameter:
menulist ein Zeiger auf eine MenuBase, aus der ein PopUpMenü erstellt werden soll

*** Rückgabewert:
/

*** Hinweise:
/

OpenPopUpMenu

Dieser Befehl öffnet ein an eine MenuBase gekoppeltes Menü an einer bestimmten Stelle auf dem Screen.
Unter welchen Bedingungen sich dieses öffnet, bleibt letztlich dem Programmierer überlassen, das kann ein Rechtsklick mit der Maus sein, oder ein bestimmter Tastendruck.

*** Syntax:
sub OpenPopUpMenu (menulist as sGUIList ptr, PosX as integer, PosY as integer)

*** Parameter:
menulist ein Zeiger auf eine MenuBase

PosX X/Y-Koordinaten auf dem Screen
PosY

*** Rückgabewert:
/

*** Hinweise:

CreateWindowMenu

Erzeugt aus einer MenüBase (mit entprechenden Einträgen) ein PullDownMenu und fügt dieses einem Fenster hinzu

*** Syntax:
sub CreateWindowMenu(menulist as sGUIList ptr, win as sGUIWindow ptr)

*** Parameter:
menulist ein Zeiger auf eine MenuBase, aus der ein PopUpMenü erstellt werden soll
win ein Zeiger auf ein Fenster, dem ein PullDownMenü hinzugefügt werden soll

*** Rückgabewert:
/

*** Hinweise:
das Fenster **muss** mit dem Flag **WFLG_MENUBAR** erzeugt worden sein.

Menüabfrage

In einer MenuBase haben alle Einträge ein ID-Nummer (Index). Im Grunde genommen werden alle Einträge einfach nur durchgezählt.
Wichtig hierbei ist, auch Einträge welche nur der Hierarchie Organisation dienen, werden mitgezählt und erhalten eine ID!

```
dim as sGUIList ptr menulist=NewMenuBase
Item(menulist,"Square",1)                    'ID=1
Item(menulist,"Triangle",1)                  'ID=2
OpenMenuStrip (menulist,"Round Things…")     'ID=3
        Item(menulist,"Circle",1)            'ID=4
        Item(menulist,"Ellipse",1)           'ID=5
CloseMenuStrip(menulist)                     'ID=6
Separator(menulist)                          'ID=7
CheckMark(menulist,"Exit active",,0)         'ID=8
Item(menulist,"Exit",0)                      'ID=9
```

Für die Abfrage des Menüs in einer Ereignisschleife stehen zwei sGUI Variablen zur Verfügung:
MENUBASE und **MENUID.**

Wenn ein Menüeintrag angewählt wurde, enthalten MENUID dessen oben beschriebene ID und MENUBASE enthält die Liste, in welcher sich der angewählte Menüeintrag befindet.
Wenn man mehrere Menüs hat, ist somit die MENUBASE ein wichtiges Unterscheidungsmerkmal

```
dim as integer selection
do
  sleep 1
  MasterControlProgram
  if RMB=RELEASE then OpenPopUpMenu(menulist,MOUSEX, MOUSEY)

  if MENUBASE then
    select case MENUBASE
      case menulist
        if MENUID then

        print "MenuBase: " & MENUBASE & ", MenuID: " & MENUID
        select case MENUID

          case 8'CheckMark
            GetMenuEntry(menulist,8,,selection)
            if selection then
              print "CheckMark selected, 'Exit' now selectable"
              SetMenuEntry(menulist,9,1)
            else
              print "CheckMark not selected, 'Exit' not selectable"
              SetMenuEntry(menulist,9,0)
            end if

          case 9
            SCREENCLOSEBUTTON=1'leave the main loop
        end select
      end if
    end select
  end if
loop until SCREENCLOSEBUTTON
```

Menüeinträge auslesen/verändern

Es kann zur Laufzeit notwendig werden dass Menüeinträge nicht auswählbar sein dürfen, oder man muss den Selektionsstatus eines CheckMarks im Menü ändern. Dafür gibt es folgende Routinen:

GetMenuEntry

„holt“ Informationen aus einem Menüeintrag

*** Syntax:
sub GetMenuEntry (menulist as sGUIList ptr, Index as integer, byref Activation as integer=-1, byref Selection as integer=-1)

*** Parameter:

menulist	ein Zeiger auf eine MenuBase
Index	eine ID eines Eintrages
Activation	eine Variable die nach Aufruf von GetMenuEntry die Activation dieses Eintrages enthält
Selection	eine Variable die nach Aufruf von GetMenuEntry die Selection dieses Eintrages enthält

*** Rückgabewert:
/

*** Hinweise:

hier muss man vorher zwei integer Variablen dimensionieren und diese dann der Routine als Parameter übergeben.
Diese werden dann durch die Routine verändert (siehe Schlüsselwort BYREF in FB Referenz).

SetMenuEntry

„schreibt“ Informationen in eine Menüeintrag

*** Syntax:
sub SetMenuEntry (menulist as sGUIList ptr, Index as integer, Activation as integer=-1, Selection as integer=-1)

*** Parameter:

menulist	ein Zeiger auf eine MenuBase
Index	eine ID eines Eintrages
Activation	setzt die Activation 0=anwählbar, 1=anwählbar
Selection	setzt die Selection 0=nicht selektiert, 1=selektiert

*** Rückgabewert:
/

*** Hinweise:
/

Kleine Helferlein

Hier sollen in loser Folge Tipps und Hinweise kommen…

GetHoveredObjects

*** Syntax:
sub GetHoveredObjects (byref win as sGUIWindow ptr, byref gad as Gadget ptr)

*** Parameter:

win	eine vorher entsprechend erzeugte Zeigervariable
gad	eine vorher entsprechend erzeugte Zeigervariable

*** Rückgabewert:
die der Routine übergebenen Variablen werden nach Aufruf der Routine gesetzt und danach weiterverarbeitet werden.
Je nach dem wo sich der Mauszeiger befindet, kann nur der Zeiger des Fenster bzw beide Zeiger, vom Fenster und Gadget, ermittelt werden.

*** Hinweise:
/

sGUIMainSize

Diese Variable definiert die Größe der Symbole der **Checkbox** und des **RadioButtons** sowie die Dicke des **Scrollbars**.
Geladene BitmapFonts beeinflussen diese Variable.

Wem also, **nach den geladenen Fonts(!)**, die oben genannten Elemente zu groß erscheinen, kann dieser Variable einen neuen Wert zuweisen:

```
 <code>
sGUIMainSize=14
</code>
```

sGUIMainGap

Diese Variable ist der Standard-Abstand innerhalb der GUI. Ausserdem entspricht ist dies auch die Dicke des Fensterrahmens. Auch Buttons denen keine Breite/Höhe übergeben wurden, arbeiten mit diesem Abstand.

Nach InitGUI kann auch dieser Variable ein neuer Wert übergeben werden:

```
 <code>
sGUIMainGap=5
</code>
```