



# wxWidgets, ein Monster will gebändigt werden



## Vorwort

Das Potential von BlitzMax blickt über den Tellerrand der Spielentwicklung hinaus, auch professionelle Anwendungen lassen sich damit entwickeln. Für 30\$ USD bekommt man das MaxGUI Modul, es bietet das Nötigste an Widgets und ist leicht zu bedienen. Ich möchte hier jedoch ein Framework vorstellen, das weit darüber hinaus geht und völlig kostenlos ist.

## Was ist wxWidgets?

wxWidgets ist ein Framework zum entwickeln plattformunabhängiger Software. Alleine die Argumente kostenlos und Open Source (leicht modifizierte LPGL) sollten einen dazu bewegen einen Blick auf wxWidgets zu werfen. Die anfängliche Skepsis, ausgehend vom großen Bibliotheksumfang (.EXE Dateien unter 3 MByte sind nicht drin), und der etwas umständlichen Installation des Moduls verfliegen schnell, nachdem erste Samples kompiliert sind. Das Herumblättern im User Manual gleicht einem Einkauf mit Papas Kreditkarte. Hier eine Übersicht, was in den Regalen zu finden ist:

- alle möglichen Dialoge (Colorpicker, Fontdialog, File open/save, Print)
- große Auswahl an Widgets (Button, Textbox, Treeview, Slider, Tab, Listbox, Progressbar, Scrollbar, Spinbutton)
- Menüs und Toolbars
- MDI Unterstützung, Splashscreen, Trayicon
- XML Parser, Logger, Mehrsprachenunterstützung, PDF- und Druckerausgabe
- Streams für Zlib, gzip und ZIP Kompression/ Dekompression sowie Umgang mit TAR-Archiven
- Netzwerkfunktionalität für FTP und HTTP
- Threadverwaltung und Interprocesskommunikation
- RichText- und Scintilla Unterstützung so-

wie einen eigenen HTML Renderer für Hilfetexte

Alles ist ausreichend dokumentiert, meist ist die Dokumentation sogar überflüssig, da Klassen- und Membernamen ein einheitliches und prägnantes Namensschema besitzen.

## Installation

wxMax hält gleich mehrere Barrieren bereit. Es wird ein C++ Compiler (GCC resp. MinGW) und ein SVN Client benötigt. Stellvertretend soll hier die Installation unter Windows gezeigt werden.

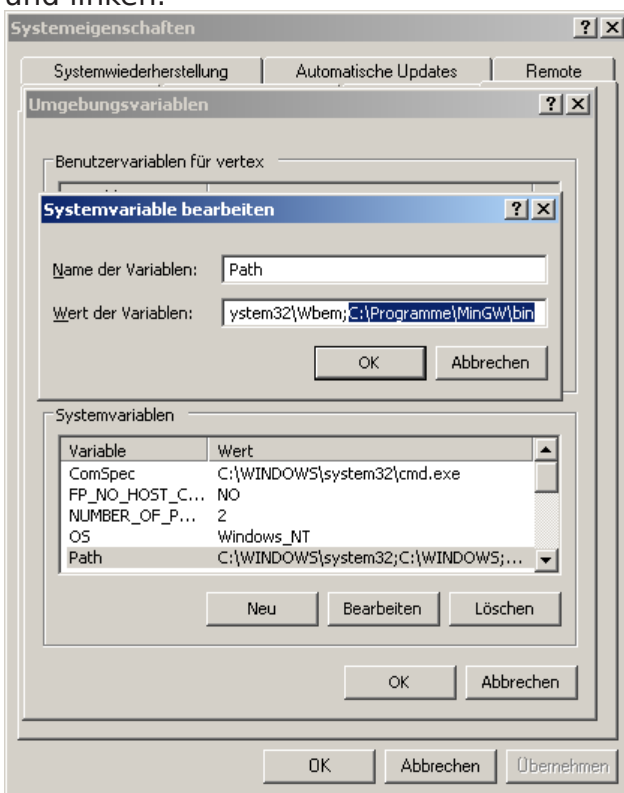
Zuerst wird die aktuelle BlitzMax Version 1.28 benötigt, da nur diese die nötigen Erweiterungen, mit denen die Module von Brucey A. Henderson kompiliert werden können, enthält.

Den C++ Compiler MinGW findet man auf der SourceForge Seite unter [Link](#). Hier muss der Automated MinGW Installer für das Release MinGW-1.5.3 heruntergeladen werden. Nach starten von MinGW-5.1.3.exe wählt man Download and install, bestätigt die Lizenzvereinbarungen, wählt current Package und aktiviert MinGW Base tools sowie den g++ compiler. Als Installationsort nehmen wir C:\Programme\MinGW. Danach werden alle Pakete automatisch heruntergeladen und installiert.

**Link** [http://sourceforge.net/project/showfiles.php?group\\_id=2435](http://sourceforge.net/project/showfiles.php?group_id=2435)

Unter Windows Vista ist es zudem erforderlich den Inhalt von C:\Programme\MinGW\libexec\gcc\mingw32\3.4.2 nach C:\Programme\MinGW\bin zu kopieren.

Damit BlitzMax MinGW erkennt, fehlt noch ein Eintrag in die Umgebungsvariable Path. Man findet sie über Start, Einstellungen, Systemsteuerung, System, Erweitert, Umgebungsvariablen, Systemvariablen. Getrennt durch ein Semikolon wird Path um C:\Programme\MinGW\bin erweitert. Von jetzt an lassen sich auch C/C++ Quellcodes in ein BlitzMax Program kompilieren und linken.



wxMax wird über SVN auf den neusten Stand gehalten. Einen kostenlosen SVN Client kann man sich unter [www.subversion.tigris.org](http://www.subversion.tigris.org), Downloads, Windows herunterladen. Hinter svn-1.4.5-setup.exe steckt ein einfacher Installer. Nach bestätigen der Lizenzvereinbarungen und Wählen des Installationsortes (hier C:\Programme\Subversion) installiert sich der Client automatisch. Der Projektmanager der Standard IDE unterstützt von nun an auch das Verwalten von SVN Projekten. Jetzt kann wxMax heruntergeladen werden. Dazu öffnet man eine Konsole über Start, Ausführen, cmd und wechselt in das BlitzMax Modul Verzeichnis via cd

C:\ [Enter] cd Programme\BlitzMax\mod [Enter]. Von hier aus wird mit svn checkout <http://wxmax.googlecode.com/svn/trunk/wx.mod/> [Enter] Das Modul vom GoogleCode Repository heruntergeladen. Im Modul Ordner sollte danach ein weiterer Ordner wx.mod existieren. Die fehlenden Header und Bibliotheken bekommt man unter [Link](#). wxwidgets\_2.8.7\_headers.zip muss ins wx.mod\include und wxwidgets\_2.8.7\_static\_win32b.zip ins wx.mod\lib Verzeichnis extrahiert werden. Um die Module zu kompilieren muss man das bmk Tool benutzen. Dazu in der Konsole bmk makemods wx[Enter] eintippen. Aufgrund des Umfangs wird das Kompilieren ca. eine halbe Stunde dauern. Kaffee trinken ist also angesagt. Praktisch ist auch die BlitzMax Dokumentation von wxMax um schnell Methoden nachschlagen zu können. Hierfür muss noch makedocs [Enter] in der Konsole aufgerufen werden. wxMax ist fertig installiert. Ob alles geklappt hat, weiß man nach dem Kompilieren und Starten von hello\_world.bmx im Ordner wx.mod\samples. Sollte es zu der Fehlermeldung »cannot find -lwinpool« kommen, müssen die 3 Bibliotheken libwinpool.a, librpcrt4.a und libodbc32.a von MinGW\lib in wx.mod\lib\win32 kopiert werden. Danach sollte sich das Programm fehlerfrei kompilieren lassen.



### Das erste Programm

Unser erstes Programm soll einfacher Weise »Hello, world!« in einer MessageBox ausgeben.

[Link http://code.google.com/p/wxmax/downloads/list](http://code.google.com/p/wxmax/downloads/list)



```

SuperStrict

Framework wx.wxApp
Import wx.wxControl

Global myApplication : TMyApplication

myApplication = New TMyApplication
myApplication.Run()

Type TMyApplication Extends wxApp
    Method onInit:Int()
        wxMessageBox("Hello, welt!" , "Nachricht für dich!")
    End Method
End Type

```

wxWidgets ist komplett objektorientiert. Typisch für wxWidgets ist das Ableiten der Klassen und Überladen von Methoden. Eine wxWidgets Anwendung ist daher immer von *wxApp* abgeleitet und überlädt die Methode *onInit*.

Als nächstes wollen wir ein Fenster mit Menü und Statusbar erstellen. Hierfür benötigen wir die Klassen *wxFrame*, *wxMenuBar* und *wxMenu*. Zuerst muss eine Fensterklasse, abgeleitet von *wxFrame*, erstellt werden. In der überladenen *onInit* Methode wird dann das Menü und die Statusbar erstellt. Typischerweise werden in der *onInit* - Methode auch alle Widgets geladen. Aber dieses Beispiel kommt ohne aus.

```

SuperStrict

Framework wx.wxApp
Import wx.wxFrame
Import wx.wxMenu

Global myApplication : TMyApplication

myApplication = New TMyApplication
myApplication.Run()
End

Type TMyApplication Extends wxApp
    Field myFrame : TMyFrame

    Method onInit:Int()
        myFrame = New TMyFrame
        myFrame.Create()
        myFrame.Show()

        Return True
    End Method
End Type

Type TMyFrame Extends wxFrame
    Method onInit()
    End Method
End Type

```

Dieser Code erzeugt ein leeres Fenster. Wohlgemerkt wird hier die *onInit* - Methode der Klasse *TMyApplication* benutzt, um das Fenster auf elegante Art zu erzeugen.

Als nächstes widmen wir uns der Statusbar. Dazu wird die Statusbar in einem Member *statusBar* referenziert. Die Methode *CreateStatusBar* ist in der Klasse *wxFrame* enthalten und hat als Rückgabebetyp *wxStatusBar*. Die Anzahl der Felder in der Statusbar kann eingestellt werden. Typischerweise werden 2 Felder verwendet. Im ersten Feld steht der Name und im zweiten Feld eine Kurzhilfe zum Widget, über das sich aktuell der Mauszeiger befindet.

```

Type TMyFrame Extends wxFrame
    Field statusBar : wxStatusBar

    Method OnInit()
        statusBar = CreateStatusBar(2)
        statusBar.SetStatusText("Text 1", 0)
        statusBar.SetStatusText("Text 2", 1)
    End Method
End Type

```

Auch das Anbringen eines Menüs ist recht einfach gemacht:

```

' ...
Field menuBar : wxMenuBar
Field menuFile : wxMenu
Field menuEdit : wxMenu
Field menuHelp : wxMenu

Method OnInit()
    menuBar = New wxMenuBar.Create()
    menuFile = New wxMenu.Create()
    menuEdit = New wxMenu.Create()
    menuHelp = New wxMenu.Create()

    menuFile.Append(wxID_NEW, "Neu")
    menuFile.Append(wxID_OPEN, "Öffnen")
    menuFile.Append(wxID_SAVE, "Speichern")
    menuFile.AppendSeparator()
    menuFile.Append(wxID_EXIT, "Beenden")

    menuBar.Append(menuFile, "Datei")
    menuBar.Append(menuEdit, "Bearbeiten")
    menuBar.Append(menuHelp, "Hilfe")
    SetMenuBar(menuBar)
' ...

```

Für Standard-Menüs, wie Neu, Öffnen usw. bietet wxWidgets vordefinierte IDs. Braucht man bspw. ein Menüeintrag »Bild einfügen« muss man sich eine eigene ID ausdenken, bspw. `Const ID_MENU_INSERT_PICTURE : Int = 100`. Die IDs sind später wichtig wenn es um Events geht, damit zwischen den Einträgen unterschieden werden kann. Tastenkürzel (Hotkeys) sind auch möglich, sie werden vom Menütext durch Tabulator ~t getrennt - so könnte der Eintrag »Öffnen« auch über [Strg+O] erreichbar sein:

```
menuFile.Append(wxID_OPEN, „Öffnen~tCtrl+O“)
```

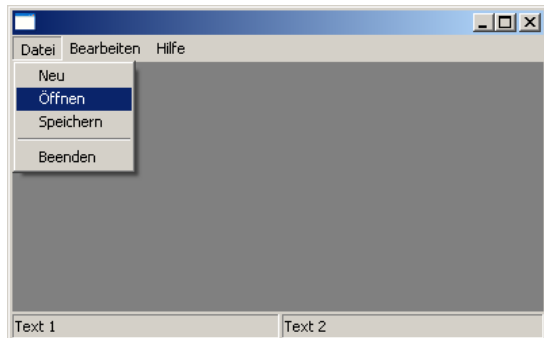
Über die Methoden *AppendSubMenu*, *AppendCheckItem* und *AppendRadioItem* lassen sich zudem Untermenüs, CheckItems (Menüeinträge mit Häkchen) und RadioItems (Menüeinträge mit Mehrfachauswahl) erstellen.

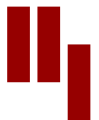
## Form Designer

Fast unverzichtbar ist ein Form Designer bei Großprojekten. Die Auswahl ist nicht besonders groß:

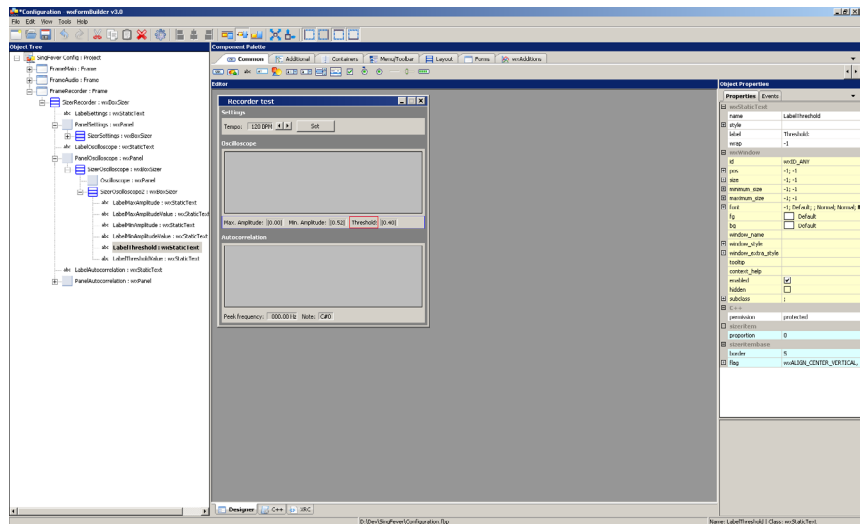
- Kommerziell: *wxDesigner*, *DialogBlocks*
- Kostenlos: *Almas Dialog Designer* (leider nicht für Windows erhältlich), *wxGlade* (nur für Python), *wxDev-C++*, *wxFormBuilder*

*wxDev-C++* und *wxFormBuilder* sind sich recht ähnlich, doch bevor man sich eine kom-





plette IDE für C++ herunterlädt, greift man besser zum wxFormBuilder. Download unter [www.wxformbuilder.org](http://www.wxformbuilder.org). Man merkt gleich, dass das kein überladenes Programm ist. Alles ist schön übersichtlich platziert. Links der Object Tree mit allen Frames und Widgets, oben die Toolbar zum Ausrichten der Widgets, darunter eine Toolbar mit allen Widgets, rechts Object Properties und im Zentrum eine Vorschau des jeweiligen Frames.



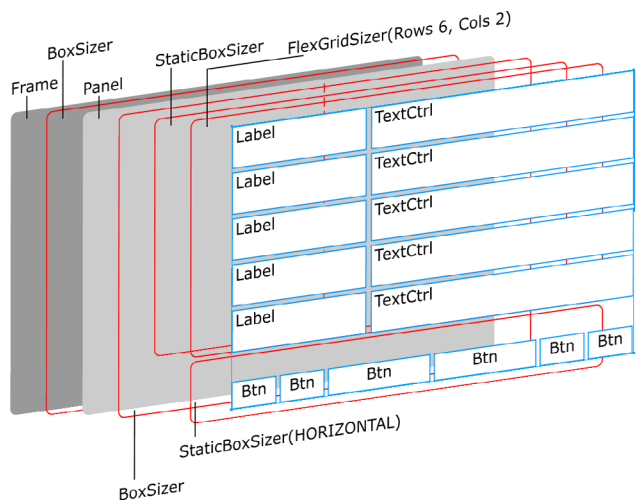
Der Designer erzeugt zum einem C++ Code und zum anderen XRC Code. XRC ist ein XML basierendes Resource System, warüber sich auch zur Laufzeit Frames laden und anzeigen lassen.

### Das Prinzip der Sizer

Bevor es weitergehen kann, müssen wir zuerst verstehen was Sizer sind und wie sie funktionieren.

Das Problem ist, dass Widgets von Plattform zu Plattform unterschiedlich aussehen und dadurch auch unterschiedliche Größen haben. Ein Button der mit 40 Pixel Breite gut unter Windows aussieht kann unter Mac OS X schon zu klein sein. Besser ist es, wenn die Größe sich dynamisch anpasst. Hier kommen die Sizer zum Einsatz. Ihre Größe wird durch ihren Inhalt bestimmt. wxWidgets verfolgt ein ähnliches Layoutkonzept wie CSS im Webdesign.

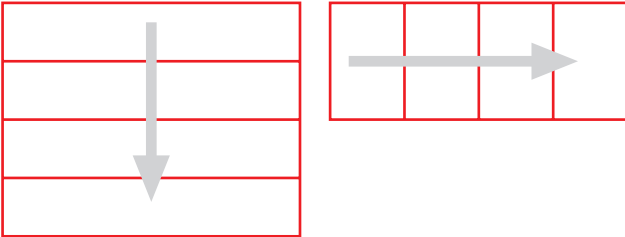
Eine typische Anwendung ist das Verwalten von Mitarbeitern in einer Datenbank. Alle Widgets sind in Sizer verschachtelt. An der Wurzel ist der Frame. Die zwei StaticBoxSizer für Person und Eintrag stehen untereinander, sind also vertikal angeordnet. Dafür bietet sich ein BoxSizer mit `orient = wxVERTICAL` an. Alle Childs werden somit untereinander angeordnet. Nicht nur Widgets sondern auch Spacer (Abstandhalter) und Sizer selbst können Childs in einem Sizer sein. Die Buttons mit *vor*, *zurück* usw. sind horizontal auf einer Linie angeordnet. Dafür kam ein BoxSizer mit `orient = wxHORIZONTAL` zum Einsatz. Für die Personendaten bot sich ein FlexGridSizer an.



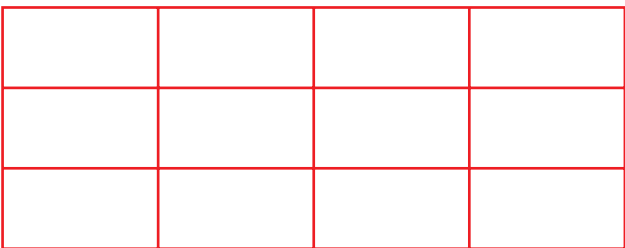
## Sizer in der Übersicht

wxWidgets bietet unterschiedliche Sizer für unterschiedliche Aufgaben:

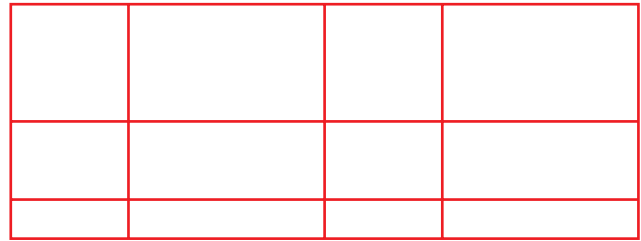
- **BoxSizer**  
Abhängig von seiner Orientierung, werden seine Childs entweder untereinander wxHORIZONTAL oder nebeneinander wxVERTICAL angeordnet.



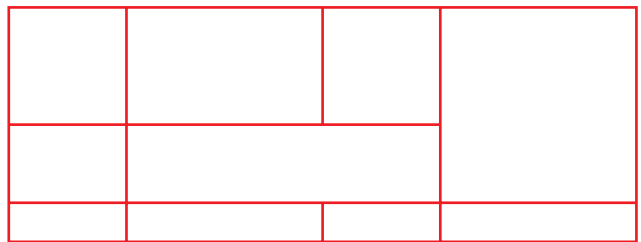
- **StaticBoxSizer**  
Wie der Name schon sagt, verhält er sich wie ein BoxSizer mit dem Unterschied, dass er zusätzlich eine Überschrift (Label) und einen umfließenden Rahmen besitzt. In anderen Frameworks ist häufig von einer *GroupBox* die Rede.
- **GridSizer**  
Diesen Sizer darf man sich als eine zweidimensionale Tabelle vorstellen. Die Anzahl an Spalten (*cols*) und Zeilen (*rows*) sind beliebig. Alle Zellen besitzen die selbe Größe. Zusätzlich lassen sich die vertikalen (*vgap*) und horizontalen (*hgap*) Abstände der Zellen angeben. Ein gutes Beispiel für einen GridSizer ist die Tastatur eines Taschenrechners oder eines Telefons.



- **FlexGridSizer**  
Auch der FlexGridSizer ist aufgebaut wie eine zweidimensionale Tabelle mit dem Unterschied, dass die Spalten und Zeilen nicht einheitlich groß sind. Das erlaubt mehr Flexibilität. Dazu kann man angeben, welche Spalten (*AddGrowableCol*) und Zeilen (*AddGrowableRow*) mit deren Inhalt wachsen dürfen (später dazu mehr im Abschnitt über Größe).



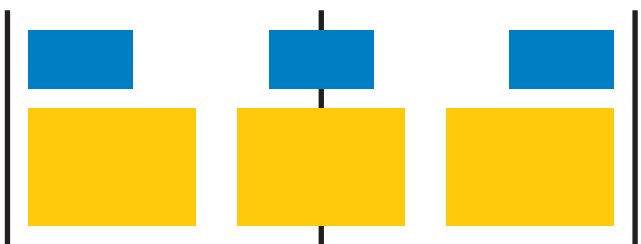
- **GridBagSizer**  
Dieser Sizer geht noch einen Schritt weiter und ermöglicht es Zeilen und Spalten zu verbinden. Wer Ahnung von HTML hat, wird sich hier an *colspan*- und *rowspan*-Attribute einer Tabelle erinnern fühlen.



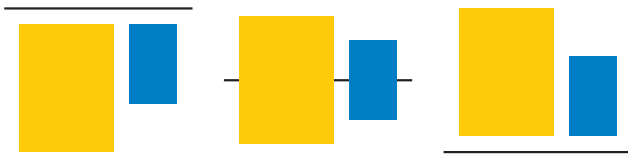
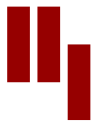
- **StdDialogButtonSizer**  
Anfangs wirkt er deplaziert hat aber auch seine Berechtigung. Wer eigene Dialogboxen erstellen möchte hat das Problem, dass bspw. der Hilfe Button mal rechts (Windows) mal links (Mac OS X) positioniert wird. Um hier größten Komfort für den Benutzer zu garantieren, nimmt man diesen Sizer um plattformunabhängig die Standardbuttons zu platzieren.

### Richtig platzieren

Das Anordnen von Elementen innerhalb eines Sizers erfolgt über die Alignflags. Man muss hier die Ausrichtung des Sizers berücksichtigen. In einem BoxSizer mit vertikaler Orientierung lassen sich die Childs entweder links *wxALIGN\_LEFT*, mittig *wxALIGN\_CENTER\_HORIZONTAL* oder rechts *wxALIGN\_RIGHT* anordnen. Das ist

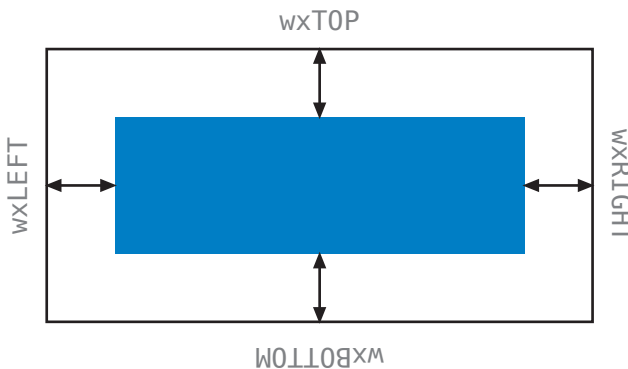


bspw. bei Eingabemasken sinnvoll. In einem BoxSizer mit horizontaler Orientierung hingegen sind nur oben *wxALIGN\_TOP*, mittig *wxALIGN\_CENTER\_VERTICAL* oder unten *wxALIGN\_BOTTOM* zugelassen:



Bei den Sizern mit Zellen (GridSizer, FlexGridSizer und GridBagSizer) ist auch eine Kombination aus vertikalen und horizontalen Flags möglich. Ein Button wird bspw. mit `wxALIGN_RIGHT|wxALIGN_BOTTOM` nach unten rechts verfrachtet. `wxALIGN_CENTER` ist eine Kombination aus `wxALIGN_CENTER_HORIZONTAL` und `wxALIGN_CENTER_VERTICAL`.

Auch der äußere Abstand der Childs lässt sich bestimmen. Man verpasst dem Widget einen unsichtbaren Rahmen und kann festlegen ob er oben `wxTOP`, unten `wxBOTTOM`, links `wxLEFT` oder rechts `wxRIGHT` zum tragen kommt. Eine Kombination der einzelnen Flags ist möglich. Standardmäßig wird `wxALL` genommen - eine Kombination aus allen Flags. Die Außenrahmen werden übrigens zusammenaddiert. Stehen sich zwei Buttons mit 10 Pixel Außenrahmen gegenüber so beträgt ihr Gesamtabstand 20 Pixel. Der Webdesigner wird hier an *margin-left*, *margin-right* usw. denken.

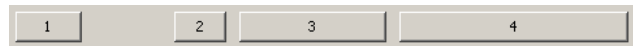


### Größe

Anfangs wirkt das Bestimmen der Größe etwas kompliziert, man hat sich allerdings auch hier nach kurzer Zeit reingearbeitet.

Alle Childs teilen sich einen endlichen Raum. Wie dieser aufgeteilt wird, bestimmt die Eigenschaft *Proportion*. Ist sie auf 0 (Standard) gesetzt, nimmt das Child die optimale Größe an (bei `StaticText` ist die abhängig von Texthöhe und Breite) der restliche Raum wird dann auf die anderen Childs aufgeteilt. Mit *Proportion* kann man

dabei die einzunehmenden Anteile bestimmen. Bspw. teilen sich zwei Buttons den Raum in einem `BoxSizer` (mit horizontaler Orientierung). Der erste Button bekommt 7 Anteile der Zweite 3 Anteile. Ist der `BoxSizer` 200 Pixel breit, bekommt der erste Button eine Breite von 140 Pixel und der Zweite eine Breite von 60 Pixel zugesprochen; also 7 von 10 und 3 von 10. Bei einem `BoxSizer` mit vertikaler Orientierung wird statt der Breite die Höhe anteilmäßig aufgeteilt. Auf `Grid-`, `FlexGrid-` und `GridBagSizer` hat diese Eigenschaft übrigens keine Auswirkung, da sich nur ein einziges Child im Raum (der Zelle) befindet. Anhand eines einfachen Beispiels mit vier Buttons und einem `Spacer` soll das veranschaulicht werden:



Zu sehen sind vier Buttons und einem `Spacer` mit unterschiedlichen *Stretchfactors*. Button 1 mit *Proportion* = 1, der `Spacer` mit *Proportion* = 1, Button 2 mit *Proportion* = 0, Button 3 mit *Proportion* = 2 und Button 4 mit *Proportion* = 3. Button 2 hat eine feste Breite von 40 Pixel. Der restliche Raum wird auf die anderen Buttons und den `Spacer` aufgeteilt. Button 1 bekommt davon 1/7, der `Spacer` 1/7, Button 3 2/7 und Button 4 3/7.

Des weiteren gibt es noch die Eigenschaft *Expand*. In `BoxSizern` mit vertikaler Orientierung dehnt es in die Breite, mit horizontaler Orientierung in die Höhe aus. In den Zellen basierenden Sizern wird der in Breite und Höhe der volle Platz genutzt.

### Projekt wxID3 Editor

Nach der ganzen Theorie können wir unser Wissen gleich auf die Probe stellen. Es soll ein Editor von `ID3v1` und `ID3v1.1` - Tags in MP3 Dateien werden. ID3 steht für *Identify an MP3*. Diese Tags speichern Informationen über Interpret, Titel, Genre usw. des Stücks. Ein `ID3v1` Tag ist 128 Byte groß und ist immer am Ende einer MP3 Datei zu finden. Der Tag wird von MP3 Playern ignoriert, da der MPEG Frame Header sich mit 11 gesetzten Bits synchronisiert. »TAG« ist in binär Schreibweise **01010100 01000001 01000111** und widerspricht somit den Synchronisationsbits.

Der Aufbau ist folgender:

**Offset Länge Beschreibung**

Offset	Länge	Beschreibung
0	3	Signatur »TAG«
3	30	Titel
33	30	Interpret
63	30	Album
93	4	Erscheinungsjahr
97	30	Kommentar
127	1	Genre

Die Länge von 30 Byte beim Feld *Interpret* bspw. ist nicht zwingend. Sollte er kürzer ausfallen, werden die restlichen Bytes mit Null-Zeichen aufgefüllt.

Das Feld *Genre* gibt den Index in einer Genreliste an.

In Version v1.1 hat ein zusätzliches Feld *Tracknummer* Einzug gehalten. Dazu wurde das Kommentarfeld um zwei Byte gekürzt. Es folgt ein Null-Zeichen und anschließend ein Byte Tracknummer. Mit dem Null-Zeichen wurde die Rückwärtskompatibilität gesichert.

Die entsprechende Klasse *TID3v1Tag* ist schnell geschrieben und soll auch nicht Thema bezogen weiter interessant sein. Sie stellt zwei Methoden *Read* und *Write* zur Verfügung an denen ein Dateistream übergeben wird.

Die Oberfläche wird in *wxFormBuilder* entwickelt. Die Datei *wxID3Editor.fbp* enthält das gesamte Projekt.



Um nicht stundenlang den Aufbau des GUIs zu besprechen nur ein paar Besonderheiten:

- Es ist nicht nötig, dass das Fenster *frame-*

*Main* vom Benutzer vergrößert / verkleinert werden kann. Dazu wurde der Standardstyle *wxDEFAULT\_FRAME\_STYLE* auf *wxCAPTION*, *wxCLOSE\_BOX*, *wxMINIMIZE\_BOX*, *wxSYSTEM\_MENU* abgeändert. Die vier Flags garantieren eine Titelleiste mit Mnimieren- und Beendenbutton.

- Bei *bitmapLogo* wurde der relative Pfad *Logo.bmp* verwendet. Standardmäßig wird aber ein absoluter Pfad von *wxFormBuilder* vorgegeben. Damit kommt aber weder *wxFormBuilder* zurecht, noch sollte man mit absoluten Pfaden arbeiten.
- Die Metadaten wurden in einem *FlexGridSizer* realisiert. Die zweite Spalte wurde als *growable* festgelegt. In der ersten Zeile der zweiten Spalte wurde ein *BoxSizer* mit horizontaler Orientierung benutzt um die zwei Widgets *textTrack* sowie *checkVersion* auf einer Zeile unterzubringen. Das gilt analog für die vorletzte Zeile, welche aus Platzgründen *labelGenre* und *choiceGenre* noch untergebracht wurden.
- Um die Länge der Eingabe zu begrenzen, wurde bspw. bei *textInterpret* das Attribut *maxlength* auf 30 gesetzt.

Im Ordner *wx.max/tools* gibt es ein Tool *wxCodeGen*, womit man aus *wxFormBuilder*-Projekte *BlitzMax Code* generieren lassen kann. Es ist von Vorteil, wenn das Erstellen der ganzen Widgets in eine separate Datei ausgelagert wird. So wurde der generierte Code leicht modifiziert in die Datei *Designer.bmx* ausgelagert. In der Methode *onInit* der Klasse *TFrameMain* wird sie dann wieder mit `Include "Designer.bmx"` eingebunden. Neben dem Festlegen der maximalen Länge des *TextCtrls* gibt es noch Validatoren, die die Eingabe in Echtzeit prüfen. Für die Widgets *textTrack* und *textYear* wurde ein numerischer Validator gewählt. Der verhindert, dass Buchstaben oder andere Zeichen eingegeben werden dürfen. Dazu wurde `New wxTextValidator.Create(wxFILTER_NUMERIC)` als Parameter bei der *Create*-Methode übergeben. Bis zu diesem Punkt ist alles Schnittstellendesign, hat also mit Programmieren recht wenig am Hut.

Die eigentliche Funktionalität des Pro-



gramms kommt jetzt. Man stelle sich vor, das Programm wird zum ersten mal gestartet; eine MP3 ist noch nicht ausgewählt. Bei diesem Szenario sollten alle Widgets, bis auf den Ladebutton, deaktiviert sein um den Benutzer nicht zu verwirren. Die Klasse *wxWindow*, von der unter anderem auch *wxTextCtrl* und *wxButton* abgeleitet sind, bietet die Methoden *Enable* und *Disable* zum aktivieren und deaktivieren des Widgets. Wir werden uns für *TFrameMain* zwei Methoden *disableAll* und *enableAll* programmieren. Erste deaktiviert alle Widgets, bis auf den Ladebutton, und letztere aktiviert sie. *textTrack* kann aber nicht trüdem aktiviert werden. Das muss in Abhängigkeit von *checkVersion* geschehen. Den aktuellen Stand dieser *CheckBox* erfährt man mit *checkVersion.GetValue()*, das *True* für *checked* und *False* für *unchecked* zurückliefert.

```

Method disableAll()
    buttonSave.Disable()
    textTrack.Disable()
    checkVersion.Disable()
    textTitle.Disable()
    textInterpret.Disable()
    textAlbum.Disable()
    textYear.Disable()
    choiceGenre.Disable()
    textComment.Disable()
End Method

Method enableAll()
    buttonSave.Enable()
    If checkVersion.GetValue() Then textTrack.Enable()
    checkVersion.Enable()
    textTitle.Enable()
    textInterpret.Enable()
    textAlbum.Enable()
    textYear.Enable()
    choiceGenre.Enable()
    textComment.Enable()
End Method

```

Zu Beginn muss also *disableAll* aufgerufen werden, damit es nicht zum Missverständnis mit dem Benutzer kommt. Bleiben wir gleich bei der *CheckBox* *checkVersion*. Möchte der Benutzer Version 1.1 benutzen hat das 2 Konsequenzen: *textTrack* muss aktiviert werden und *textComment* kann nur noch 28 statt 30 Zeichen entgegen nehmen. Und hier kommen nun Events ins Spiel.

Bei jeder Aktion, sei es hervorgerufen durch den Benutzer oder durch das Betriebssystem, wird ein Event ausgelöst. So werden Events

beim klicken auf einem Button, ändern eines Textes, Größe ändern eines Fensters, Drag 'n' Drop von Dateien uvm. ausgelöst. Auf alle Events muss nicht reagiert werden. Beim Button *buttonLoad* interessiert uns, bis auf das Klick-Event, kein anderes. Ob sich der Mauszeiger über den Button bewegt spielt keine Rolle für die Programmfunktionalität. Deswegen werden Events auch explizit über die Methoden *Connect* resp. *ConnectAny* mit Funktionen verbunden. Events werden also in Funktionen verwaltet, genauer in Call-back-Funktionen. Ist das Klick-Event von *checkVersion* durch

```
ConnectAny(wxEVT_COMMAND_CHECKBOX_CLICKED, onVersionClick)
```

mit *onVersionClick* verbunden, wird die Funktion von nun an von *wxWidgets* immer dann aufgerufen, wenn der Benutzer auf die *CheckBox* klickt. Die Funktionsprototypen für das Eventhandling sind immer gleich: *Function Funktionsname(e:wxEvent)*. Weiterhin gibt es mehrere Eventklassen die von *wxEvent* abgeleitet sind. Bei *wxEVT\_COMMAND\_CHECKBOX\_CLICKED* wird bspw. *wxCommandEvent* ausgelöst. Diese hat, neben anderen Erweiterungen, die Methode *IsChecked* um abzufragen, ob die *CheckBox* aktiviert oder deaktiviert wurde. Das geht auch mit *CheckBox*-Menüeinträgen.

```

Function onVersionClick(e:wxEvent)
    Local event : wxCommandEvent, ..
        _self : TFrameMain

    event = wxCommandEvent(e)
    _self = TFrameMain(event.Parent)

    If event.IsChecked() Then
        _self.textTrack.Enable()

```

```

        _self.textComment.SetMaxLength(28)
    If _self.textComment.GetLabelText().Length > 28 Then ..
        _self.textComment.SetLabel(_self.textComment.GetLabelText()[..28])
    Else
        _self.textTrack.Disable()
        _self.textComment.SetMaxLength(30)
    EndIf
End Function

```

Leider ist es nicht möglich Methoden für das Eventhandling zu benutzen. Stattdessen kommen statische Funktionen zum Einsatz. Um dennoch Übersicht zu wahren, erstellt man die Funktionen in den Klassen. Über das Casten von *event.parent* in den Klassentyp (siehe *\_self*) kann man dann auch auf die Member der Instanz zugreifen.

Die Funktionalität von *buttonLoad* besteht darin, einen Dateidialog anzuzeigen in dem der Benutzer die MP3 wählt, die Datei als Stream zu öffnen und den ID3-Tag, falls vorhanden, auszulesen und anzuzeigen. Dazu könnte man die Klasse *wxFileDialog* benutzen, es geht aber über die Funktion *wxFileSelector* etwas bequemer.

```

Function onLoadClick(e:wxEvent)
    Local event : wxCommandEvent, ..
        _self : TFrameMain, ..
        file : String

    event = wxCommandEvent(e)
    _self = TFrameMain(event.Parent)

    file = wxFileSelector("Öffne MP3 Datei",,, ".mp3", ..
        "MP3 Dateien (*.mp3)|*.mp3|Alle Dateien (*.*)|*.*", ..
        wxFD_FILE_MUST_EXIST, _self)

    If file Then
        Local stream : TStream, ..
            success : Int

        stream = ReadStream(file)
        If Not stream Then Throw("Konnte Datei nicht öffnen")
        success = id3Tag.Read(stream)
        stream.Close()

        _self.textFile.SetLabel(file)
        _self.textFile.SetSelection(-1, -1)

        If success Then
            If id3Tag.Track > 0 Then
                _self.textTrack.SetLabel(id3Tag.Track)
                _self.checkVersion.SetValue(True)
                _self.textComment.SetMaxLength(28)
            Else
                _self.textTrack.SetLabel("")
                _self.textTrack.Disable()
                _self.checkVersion.SetValue(False)
                _self.textComment.SetMaxLength(30)
            EndIf
            _self.textTitle.SetLabel(id3Tag.Title)
            _self.textInterpret.SetLabel(id3Tag.Interpret)
            _self.textAlbum.SetLabel(id3Tag.Album)
            _self.textYear.SetLabel(String.FromInt(id3Tag.Year))
            _self.choiceGenre.SelectItem(id3Tag.Genre)
            _self.textComment.SetLabel(id3Tag.Comment)
        Else
            _self.textTrack.SetLabel("")
            _self.checkVersion.SetValue( )
            _self.textComment.SetMaxLength(28)
            _self.textTitle.SetLabel("")
        EndIf
    EndIf
End Function

```



```

        _self.textInterpret.SetLabel("")
        _self.textAlbum.SetLabel("")
        _self.textYear.SetLabel("")
        _self.choiceGenre.SelectItem(0)
        _self.textComment.SetLabel("")
    EndIf
    _self.enableAll()
EndIf
End Function

```

»Öffne MP3 Datei« wird in der Titelleiste des Dialogs angezeigt. Die Parameter *defaultPath* und *defaultFileName* lassen wir außer Acht - sie sind aber nützlich, wenn man zu Anfang bspw. den Ordner *Meine Bilder* für ein Grafikprogramm benutzen möchte. Der Parameter *wildc* für die Dateierweiterungen folgt einer eigenen Syntax. Alle Einträge werden durch ein | getrennt. Anzeigetext|Erweiterung;Erweiterung|Anzeigetext|Erweiterung;Erweiterung|... Bei einem Grafikprogramm könnte *wildc* bspw. »JPEG Bild(\*.jpg;\*.jpeg)|\*.jpg;\*.jpeg|GIF Bild(\*.gif)|\*.gif|PNG Bild(\*.png)|\*.png|Alle Dateien|\*.\*« lauten. `wxFILE_MUST_EXIST` stellt sicher, dass die Datei auch existiert (nur wenn *Abbrechen* geklickt wurde, liefert die Funktion `Null` zurück).

Den umgekehrten Weg geht *buttonSave*. Hier müssen zunächst alle Einträge aus den Widgets in den ID3-Tag übernommen und anschließend in die Datei geschrieben werden.

```

Function onSaveClick(e:wxEvent)
    Local event : wxCommandEvent, ..
           _self : TFrameMain, ..
           stream : TStream

    event = wxCommandEvent(e)
    _self = TFrameMain(event.Parent)

    If _self.checkVersion.GetValue() Then
        id3Tag.Track = _self.textTrack.GetLabelText().ToInt()
    Else
        id3Tag.Track = 0
    EndIf
    id3Tag.Title      = _self.textTitle.GetLabelText()
    id3Tag.Interpret  = _self.textInterpret.GetLabelText()
    id3Tag.Album      = _self.textAlbum.GetLabelText()
    id3Tag.Year       = _self.textYear.GetLabelText().ToInt()
    id3Tag.Genre      = _self.choiceGenre.GetSelection()
    id3Tag.Comment    = _self.textComment.GetLabelText()

    stream = OpenStream(_self.textFile.GetLabelText())
    If Not stream Then Throw("Konnte Datei nicht zum Speichern öffnen")
    id3Tag.Write(stream)
    stream.Close()
End Function

```

Die vollständige `onInit` Methode sieht folgendermaßen aus:

```

Method onInit()
    Include „Designer.bmx“
    disableAll()

    Connect(wxID_OPEN, wxEVT_COMMAND_BUTTON_CLICKED, onLoadClick)
    Connect(wxID_SAVE, wxEVT_COMMAND_BUTTON_CLICKED, onSaveClick)
    ConnectAny(wxEVT_COMMAND_CHECKBOX_CLICKED, onVersionClick)
End Method

```

Die Widgets *buttonLoad*, *buttonSave* und *checkVersion* sind nun mit ihren Callback-Funktionen verbunden. Als letzte Methode soll es noch eine Art Konstruktor geben, der den richtigen Fenstertitel und die richtige Titelleiste sicherstellt:

```

Method Construct:TFrameMain(parent:wxWindow = Null, id:Int = -1,
                             x:Int = -1, y:Int = -1)
    Super.Create(,,"wxID3 Editor",,,,, wxCAPTION ..
                | wxCLOSE_BOX ..
                | wxMINIMIZE_BOX ..
                | wxSYSTEM_MENU)
    SetIcon(wxIcon.CreateFromFile("Icon.ico", wxBITMAP_TYPE_ICO))

    Return Self
End Method

```

Zugleich wird über *SetIcon* das geladene Icon gesetzt. Es findet sich anschließend in der Titelleiste wieder.

Sollte ein Fehler auftreten, können wir diesen auch gleich in eine *MessageBox* anzeigen lassen.

```

Try
    application.Run()
Catch Exception:Object
    wxMessageBox(Exception.ToString(), "Fehler", wxICON_ERROR)
End Try

```

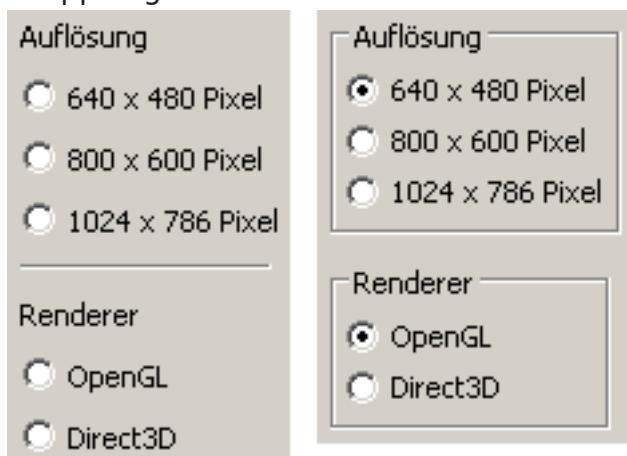
Erweitern ließe sich das Programm dahingehend, dass es Drag 'n' Drop von MP3s zulässt oder Kopieren und Einfügen des Tags in und aus der Zwischenablage unterstützt. Das ID3 v1.1 längst veraltet ist, sollte klar sein. Es bot sich aber wegen des einfachen Aufbaus für dieses Projekt an. Wer sicher gerne näher mit dem Thema befassen möchte, erhält weitere Informationen unter [www.id3.org](http://www.id3.org).

### Ausgewählte Widgets im Überblick

- **RadioButton und RadioBox**

Wann immer eine Auswahl von mehreren Einträgen getroffen werden soll, kommen *RadioButtons* zum Einsatz.

Ob ein Eintrag aktiv ist, wird mit *myRadioButton.GetValue()* abgefragt. Welche *RadioButtons* zu einer Gruppe gehören bestimmt das Flag *wxRB\_GROUP* und muss immer beim ersten Eintrag aktiviert sein. So wurde das Flag im Beispiel bei »640 x 480 Pixel« und »OpenGL« gesetzt, sonst würden die fünf *RadioButtons* als eine einzige Gruppe agieren.



Komfortabler ist hingegen eine *RadioBox*. Das ist ein *StaticBoxSizer* mit einer Liste von Auswahloptionen. Er gruppiert die Ra-

*dioButtons* von alleine. Welche Option gewählt wurde, erfährt man über *myRadioBox.GetSelection()*.

- **Slider und Gauge**

*Slider* sind Schiebepotentiometern nachempfunden. Man gibt ihnen einen *minvalue* und *maxvalue* vor und der Benutzer bewegt den Schieber im Bereich dieser beiden Werte. Welcher Wert gewählt wurde erfährt man über *mySlider.GetValue()*. *Gauge* ist vllt. besser bekannt als *ProgressBar*. Hier gibt man einen Bereich von 0 bis Max (siehe *range*) vor und in diesem Bereich kann man mit *myGauge.SetValue(x)* den Stand festlegen. *Slider*- und *Gaugewid*get lassen sich entweder horizontal oder vertikal ausrichten über ihr *Styleattribut*.



- **ListCtrl**

Eine *ListBox* besteht aus *String*-Einträgen, die in einer Liste mit *Scrollbar* angeordnet sind. Standardmäßig kann nur ein Eintrag gewählt werden, wenn das Flag *wxLB\_MULTIPLE* gesetzt ist, sind auch mehrere Einträge wählbar. Das geschieht über die

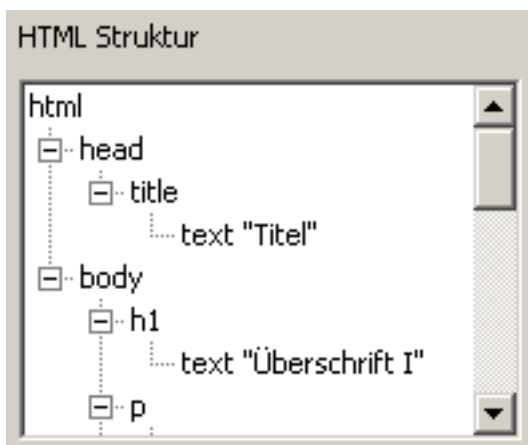


[Shift] Taste oder, bei gesetztem `wxLB_EXTENDED`-Flag, durch einfaches Anklicken des Eintrags.



Die Klasse `wxListBox` als auch `wxComboBox` und `wxChoice` sind abgeleitet von `wxControlWithItems`. Diese bietet unter anderem die Methoden

- Clear, Löschen der gesamten Einträge
- Append, Anhängen eines Eintrags an die Liste
- Insert, Einfügen eines Eintrags in die Liste an bestimmter Position
- Delete, Löschen eines Eintrags
- Select, zum Auswählen eines Eintrags
- GetSelection(s), zum Abfragen der Auswahl.
- TreeCtrl

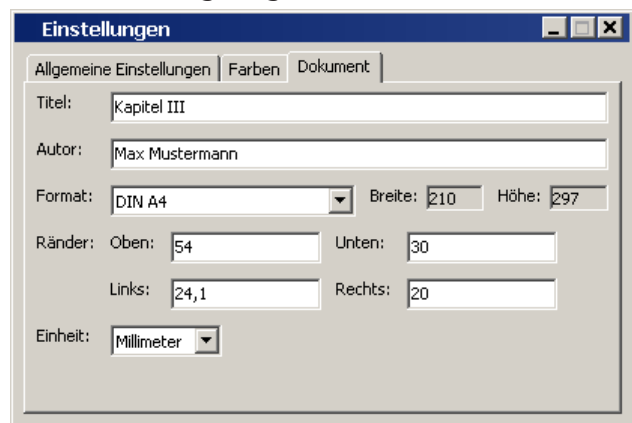


Um hierarchische Daten darzustellen, eignet sich das `TreeCtrl`. Man denke an den Explorer, welcher Ordner und Unterordner in einer Baumstruktur darstellt. Über `AddRoot` kann das Wurzelement hinzugefügt werden und über `AppendItem` lassen sich dann einem Element mehrere Unter-elemente anfügen. Über `GetSelection` resp. `GetSelections` (bei gesetztem `wxTR_`-

`MULTIPLE`-Flag) erfährt man, welches Element vom Benutzer ausgewählt ist. Weitere Methoden für das Bearbeiten der Baumstruktur sind `GetRootItem`, `GetFirstChild`, `GetLastChild` und `GetNextSibling`.

- Notebook

In einem Notebook werden Inhalte in Pages gruppiert und sind über Tabs anwählbar. Am Besten geschieht dies über ein Panel, was dann über `myNotebook.AddPage(myPanel)` als Tab hinzugefügt wird.



### Tipps und Tricks

Um seinen Programmen den letzten Schliff zu verleihen oder sich das lästige Suchen in der Dokumentation zu ersparen, folgen ein paar ausgewählte Tipps und Tricks:

- TrayIcon

TrayIcons lassen sich über die Klasse `wxTaskBarIcon` realisieren. Sie bietet die Methoden `SetIcon` und `RemoveIcon`. `RemoveIcon` sollte unbedingt zum Programmende aufgerufen werden, sonst bleibt das Icon noch eine Weile anstandslos in der TrayBar stehen.



Wenn man eine eigene Klasse von `wxTaskBarIcon` ableitet, kann man die Methode `CreatePopupMenu` überschreiben. In ihr kann man ein Popupmenü erstellen und es als Rückgabewert benutzen. Klickt der Benutzer nun mit der rechten Maustaste aufs TrayIcon, bekommt er dieses Popupmenü angezeigt. `CreatePopupMenu` ist die zu bevorzugende Variante statt auf ein `KlickEvent` der rechten Maustaste zu reagieren.

```

' ...
icon = wxIcon.CreateFromFile("Icon.ico", wxBITMAP_TYPE_ICO)
If Not icon.IsOk() Then Throw("Konnte Icon.ico nicht laden")

trayIcon = New TTrayIcon
trayIcon.Create()
trayIcon.SetIcon(icon, "wxID3 Editor")
' ...

Type TTrayIcon Extends wxTaskBarIcon
    Method CreatePopupMenu:wxMenu()
        Local menu : wxMenu

        menu = New wxMenu.Create()
        menu.Append(wxID_OPEN, "Programm öffnen")
        menu.Append(wxID_EXIT, "Programm beenden")

        Return menu
    End Method
End Type

```

- SplashScreen

Von großen Anwendungen, wie Photoshop oder Illustrator kennt man sicher die Splash-Screens zu Anfang. Diese kann man recht simpel über die Klasse *wxSplashScreen* erstellen:

```

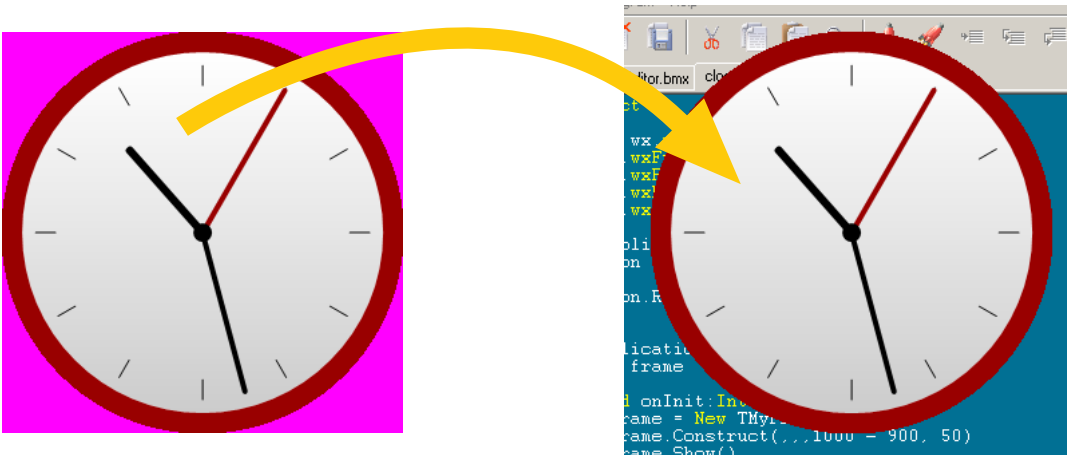
bitmap = New wxBitmap.Create()
bitmap.LoadFile("Splash.bmp", wxBITMAP_TYPE_BMP)
If Not bitmap.IsOk() Then Throw("Konnte Splash.bmp nicht laden")
splashScreen = New wxSplashScreen.Splash(..
    bitmap, wxSPLASH_CENTRE_ON_SCREEN|wxSPLASH_TIMEOUT, 5000, myFrame)

```

In diesem Beispiel wird der Splashscreen 5 Sekunden lang angezeigt, falls der Benutzer nicht vorher auf ihn klickt.

- Regions

Bei manchen Programmen ist es sinnvoll, nicht das Theme des Betriebssystems zu benutzen. Die blaue Fenstertitelleiste würde bei Winamp bspw. nicht gut aussehen. Mittels Regions lässt sich eine Maske definieren, die bestimmt, welcher Teil des Fensters gezeichnet werden soll. Die Region kann dabei aus einer Bitmap erstellt werden, wobei die Maske der Bitmap benutzt wird.



```

Type TClock Extends wxFrame
    Field bitmap      : wxBitmap
    Field region     : wxRegion
    Field background : wxStaticBitmap

    Method Construct:TClock(parent:wxWindow = Null, id:Int = -1, ..

```



```

        title:String = "", x:Int = -1, y:Int = -1)
    Create(parent, id, title, x, y,,, wxFRAME_SHAPED)
    Return Self
End Method

Method OnInit()
    bitmap = New wxBitmap.Create()
    bitmap.LoadFile("Clock.bmp", wxBITMAP_TYPE_BMP)
    If Not bitmap.IsOk() Then Throw("Konnte Clock.bmp nicht laden")
    bitmap.SetMask(New wxMask.Create(bitmap, ..
        New wxColour.Create(255, 0, 255)))

    region = New wxRegion.CreateWithBitmap(bitmap)

    Local sizer : wxBoxSizer
    sizer = New wxBoxSizer.Create(wxVERTICAL)

    background = New wxStaticBitmap.Create(Self, wxID_ANY, bitmap)
    sizer.Add(background, 0, wxALL, 0)

    SetSizer(sizer)
    Layout()
    sizer.Fit(Self)

    SetShape(region)
End Method
End Type

```

Wichtig ist, dass das Flag `wxFRAME_SHAPED` gesetzt wurde. Als Transparenzfarbe wurde, wie aus Spielen mit Sprites gewohnt, Magenta benutzt. Und ja, die Maske wird aus dem Bitmap erstellt und diese dann als Maske des Bitmaps verwendet.

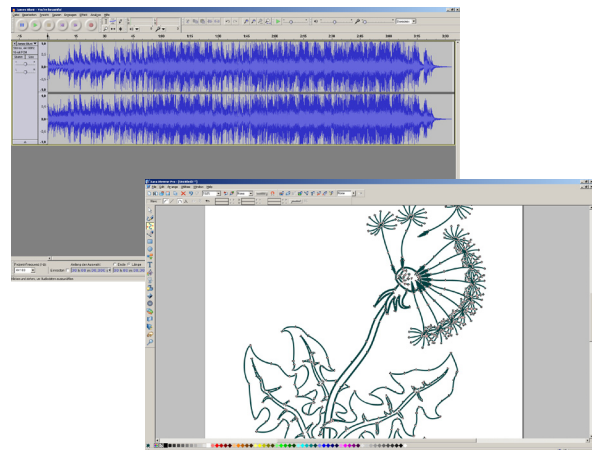
- Auflösung abfragen

Um ein Pop-upfenster an die gewünschte Position zu platzieren, ist es erforderlich, die aktuelle Auflösung zu erfragen. wxWidgets bietet dazu die Klasse `wxSystemSettings`: `width = wxSystemSettings.GetMetric(wxSYS_SCREEN_X)` und `height = wxSystemSettings.GetMetric(wxSYS_SCREEN_Y)`. Um Standardschriften oder Standardfarben abzufragen, gehören die Funktionen `GetFont` und `GetColour` zu ihren Funktionsumfang.

### Fazit

wxWidgets deckt den Bereich einfacher Konfigurationstools bis hin zu professionelle Softwarelösungen ab. Dabei ist es kinderleicht zu handhaben. Vom Industriestandard kann man zwar noch nicht sprechen, aber das Audacity (Audioeditor), Xara Xtreme (Vektorgrafik-Programm) und AOL Communicator (E-Mail Client) auf dieses Framework aufbauen, spricht für einen großen Einsatz. wxMax deckt leider noch

nicht den kompletten Umfang des Frameworks ab - BlitzMax als Sprache ist dahingehend einfach nicht mächtig genug um alle Aspekte der Objektorientierten Programmierung zu entsprechen. Ein kostenloser Ersatz für MaxGUI ist es aber allemal.



### Nachtrag

wxMax bekommt ihr jetzt auf der GoogleCode-Seite <http://code.google.com/p/wx-max/> schon vorkompiliert.

Das Projekt wxID3 Editor findet ihr unter: <http://irgendwas.de/pbm/wxIDE3Editor.zip>