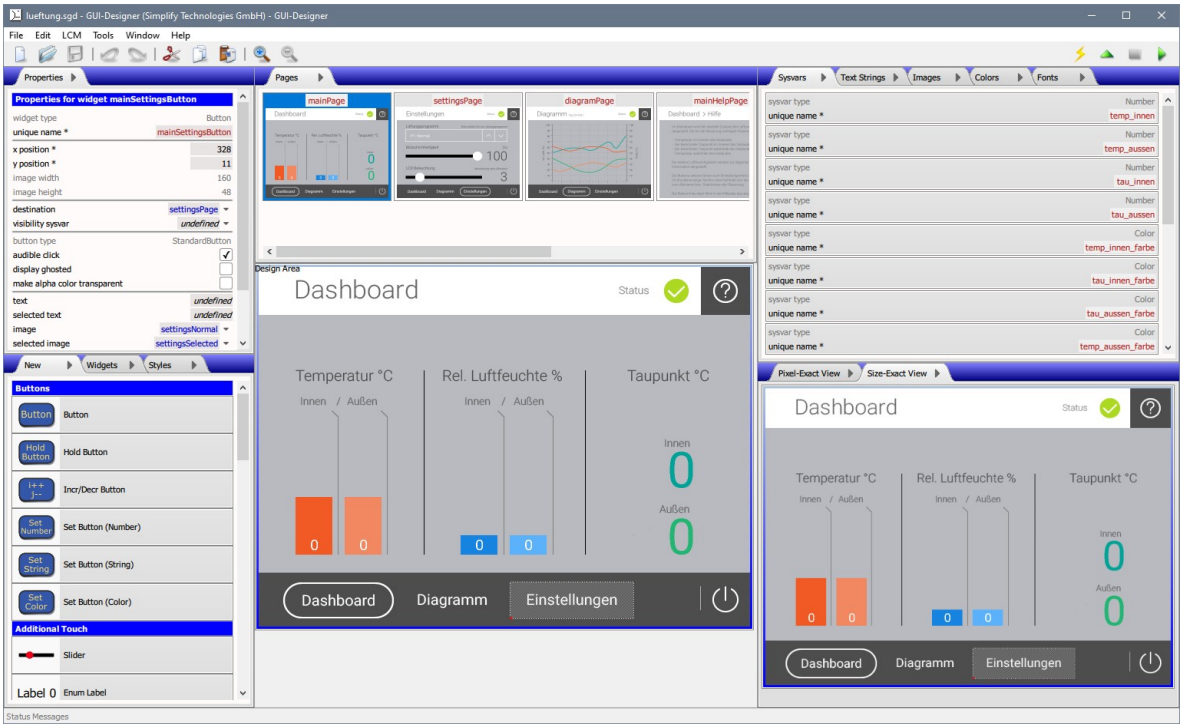
Simplify Technologies

We interface users

Arbeiten mit dem

Simplify Technologies

GUI-Designer



Handbuch

Schanzenfeldstraße 2
D-35578 Wetzlar
Tel.: +49 (0)6441 210390
www.simplify-technologies.de

Änderungsdatum: 2026-01-22

Inhaltsverzeichnis

1. Einleitung

Smart Displays:
Die Displays verstehen direkt komplexe Befehle (z.B. zur Darstellung von Buttons) und können leicht von einem externen Micro-controller angesprochen werden.

In dieser Anleitung beschreiben wir, wie Sie mit dem GUI-Designer schnell ansprechende Bedienoberflächen mit unseren LCM-Display-Modulen erstellen können. Diese Module sind sogenannte **Smart Displays**, d. h. sie enthalten über die bloße Anzeige hinaus eine Vielzahl fertiger Funktionen.

Dabei gehen wir zweistufig vor:
- In der Regel wird zuerst der Weg gezeigt, der am schnellsten zum Ziel führt.
- An Stellen, an denen das nützlich erscheint, werden erweiterte Erklärungen mit zusätzlichen Details gegeben, die man aber im ersten Schritt überspringen kann. Wenn ein tieferes Verständnis gewünscht wird, kann man diese Details dort nachschauen. Diese befinden sich immer am Ende eines Kapitels in einem weiteren Abschnitt, dessen Name mit „Details ...“ anfängt.

In diesem Handbuch sind für eine einfachere Orientierung folgende Darstellungen gewählt:
- Links zur Verzweigung innerhalb des Dokuments sind dunkelblau.
- Begriffe, die zur Beschreibung eines grundsätzlichen Konzepts verwendet werden, sind **fett** gedruckt. Diese Begriffe sind im Handbuch genau definiert, im Anhang C: Verzeichnis der Definitionen führen Links dort hin.
- Bedienelemente des GUI-Designers sind *kursiv* dargestellt.
- Aktionen über die Tastatur oder Maus sind in einer Serifen-Schrift angelegt.
- Wenn **Eigenschaften** (**Properties**) benannt werden, die im GUI-Designer gesetzt werden können, dann werden diese Namen und auch ihre Werte in einer `Computer-Schrift` dargestellt.
- Diese `Computer-Schrift` wird auch verwendet, um alle anderen Dinge zu beschreiben, die spezifisch für eine Kundenanwendung sind, also z. B. Source-Code.

1.1 Motivation - Was soll mit dem GUI-Designer erreicht werden ?

Das Erstellen einer guten Benutzeroberfläche macht mehr Arbeit, als man oft auf den ersten Blick vermutet.

Bei komplexeren Anwendungen ist es bereits aufwändig, diese genau zu spezifizieren und festzulegen, wie diese vom Benutzer bedient werden sollen. Diese Arbeit muss sinnvollerweise von jemandem gemacht werden, der die Anwendung und die Anwender am besten kennt, oft in Zusammenarbeit und in Diskussion mit anderen, insbesondere mit den Nutzern.

Das Ziel und der Sinn des GUI-Designers ist, bei der Realisierung und Umsetzung Zeit zu sparen. Mit ihm kann das Design intuitiv auf dem PC erstellt und dann im LCM-Display-Modul genutzt werden, ohne unnötigen zusätzlichen Aufwand. Man erreicht dadurch:

GUI-Designer:
Zeit und Geld sparen.
Intuitiv arbeiten.

- Erhebliche Reduzierung von Entwicklungszeit und -kosten
- Designarbeiten ohne Programmierung
  → Arbeitsteilung und Entlastung der Softwareentwickler von Designarbeiten
- Weniger Aufwand bei nachträglicher Optimierung und Änderungen des Designs
  → mehr Designverbesserungen
  →bessere Benutzeroberfläche

1.2 Wie erreicht man das ?

Wenn man die Anwendungen genauer betrachtet, so stellt man fest, dass viele vollständig über einen Satz von Variablen beschrieben werden können. Bei einer Heizungssteuerung werden z. B. Variablen wie `Innentemperatur` und `Außentemperatur` eine wichtige Rolle spielen. Dies sind auch die Kategorien, in denen man als Anwendungsentwickler denkt. Diese Variablen, die eine Anwendung charakterisieren, nennen wir im Folgenden **Systemvariablen**.

Systemvariable:
Eine Variable, die einen Zustand in der Anwendung beschreibt.
Beispiel: Eine Spannung, die gerade gemessen wird. Typen für Systemvariablen sind z. B. Zahlen und Textstrings.

Deshalb soll die Anwendung idealerweise – ohne sich weiter um Details der Gestaltung kümmern zu müssen – einfach mit den Display-Modulen in der Sprache dieser Systemvariablen kommunizieren können.

Für eine einfache Thermometer-Anwendung würde dies so ablaufen:

1. Die Außentemperatur ändert sich und wird von der Anwendung gemessen.
2. Da die Außentemperatur eine Systemvariable der Anwendung ist, sendet diese die Änderung an das LCM-Modul – mit der Bedeutung „Außentemperatur ist jetzt 23°C“.
3. Das LCM-Modul aktualisiert die Repräsentation der Temperatur, z.B. eine numerische oder eine Balkendarstellung.

Die Darstellung der Systemvariablen erfolgt automatisch.

Die Anwendung im Kundengerät braucht also nicht mehr um die grafische Gestaltung der Anzeigeelemente zu kümmern, sondern sie „denkt“ einfach in den Variablen, die für die Anwendung relevant und sowieso bereits vorhanden sind.

1.3 Zusammenarbeit zwischen Display-Modul und Anwendung

Controller-Device-Kommunikation

Die Kommunikation zwischen Ihrer Anwendung und dem LCM-Modul folgt immer einem einfachen Schema, das auch **Controller-Device-Kommunikation** genannt wird - die Anwendung (**Controller**) steuert das Display-Modul (**Device**):

1. Die Anwendungselektronik sendet ein Befehl an das LCM-Modul (z.B. „Zeige einen Button an“).
2. Das LCM-Modul führt den Befehl aus.
3. Das LCM-Modul sendet eine Antwort an die Anwendungs-elektronik („Befehl verstanden und ausgeführt, keine Fehler, der nächste Befehl kann kommen“)

Der Controller entscheidet also, wann er etwas vom Display-Modul möchte und was. Das Display antwortet nicht, wenn es nicht gefragt wird, denn vielleicht darf man den Controller gerade nicht stören, weil er etwas anderes machen muss.

Bemerkung: Wir benutzen den Begriff **Controller** und **Anwendung** manchmal als gleichbedeutend. Gemeint ist immer die Elektronik und Software des Kunden, die die Kontrolle über die Anwendung ausübt und das Display-Modul ansteuert. Denn diese Elektronik steuert nicht nur die Kommunikation als Controller im Controller-Device-Schema, sondern dort befindet sich das Know-How über die zu lösende Aufgabe selbst.

1.4 Details: Technischer Ansatz zur Erstellung des Anwendungsdesigns

Die folgenden Abschnitte beschreiben technische Zusammenhänge, die generell für ein tieferes Verständnis nützlich sind, die aber für den Einstieg in die Bedienung des GUI-Designers und für das Erstellen eines einfachen Projektes nicht sofort benötigt werden. Wenn Sie möchten, können Sie also diesen Abschnitt überspringen, um möglichst schnell konkret mit dem GUI-Designer zu arbeiten.

Hier geht es nun um die Tatsache, dass die Anforderungen an das Design einer Anwendung mittlerweile recht hoch geworden sind, so dass es praktikable Möglichkeiten geben muss, solche Anforderungen zu erfüllen.

1.4.1 Bisheriger Lösungsansatz für ansprechende Benutzeroberflächen

Ein gutes Design konnte bisher erstellt werden, indem die Anwendung den umfangreichen Befehlssatz der LCM-Display-Module (über 400 Befehle, davon die meisten für Grafikfunktionen) nutzte. Dies funktionierte aus der Sicht des

Display-Moduls etwa nach dem Motto:

„Anwendung, hier hast Du viele Möglichkeiten und Befehle; male Dir was schönes!“

Der **Controller** sendet genaue Grafikbefehle und das Display-Modul (**Device**) führt diese aus.

Die Struktur der Gesamtanwendung war damit, wie in Abbildung 1 skizziert:

Controller erstellt Bedienoberfläche direkt mit Grafikbefehlen.

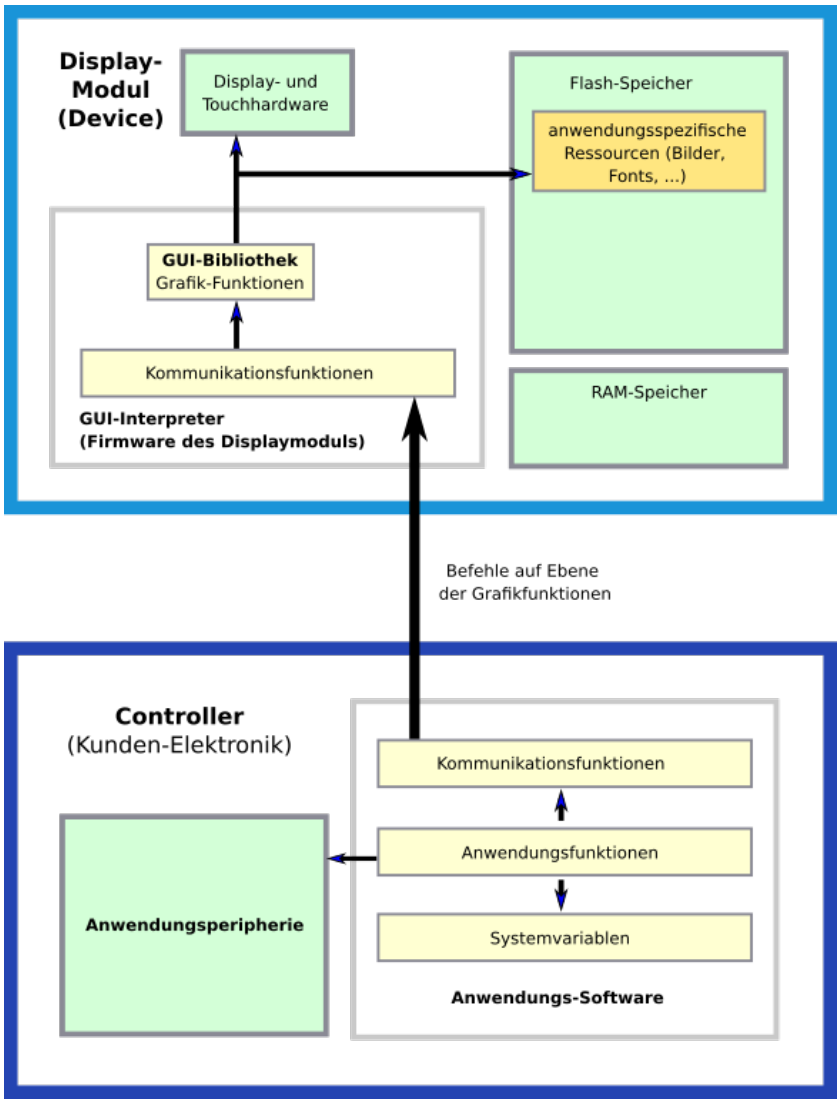


Abbildung 1: Controller sendet Grafikbefehle

Die fertigen Funktionen machten die Entwicklung bereits wesentlich einfacher, als wenn man alles „zu Fuß“ machen musste. Die Flexibilität und der Umfang der Möglichkeiten bedeuten allerdings auch einen gewissen Aufwand in der Einarbeitung in diese Funktionen und bei der Umsetzung des Designs über Funktionsaufrufe in der Anwendung.

Um diesen Aufwand weiter erheblich zu reduzieren, wurde eine Erweiterung des bisherigen Konzepts vorgenommen:

1.4.2 Erweiterter Lösungsansatz

Um die Grafikmöglichkeiten der Display-Module nutzen zu können, dabei aber den Entwicklungsaufwand erheblich zur reduzieren, wurde die Firmware erweitert. Die Struktur ist nun wie in der Abbildung 2:

Controller kommuniziert über seine Systemvariablen, Bedienoberfläche wurde durch GUI-Designer erstellt.

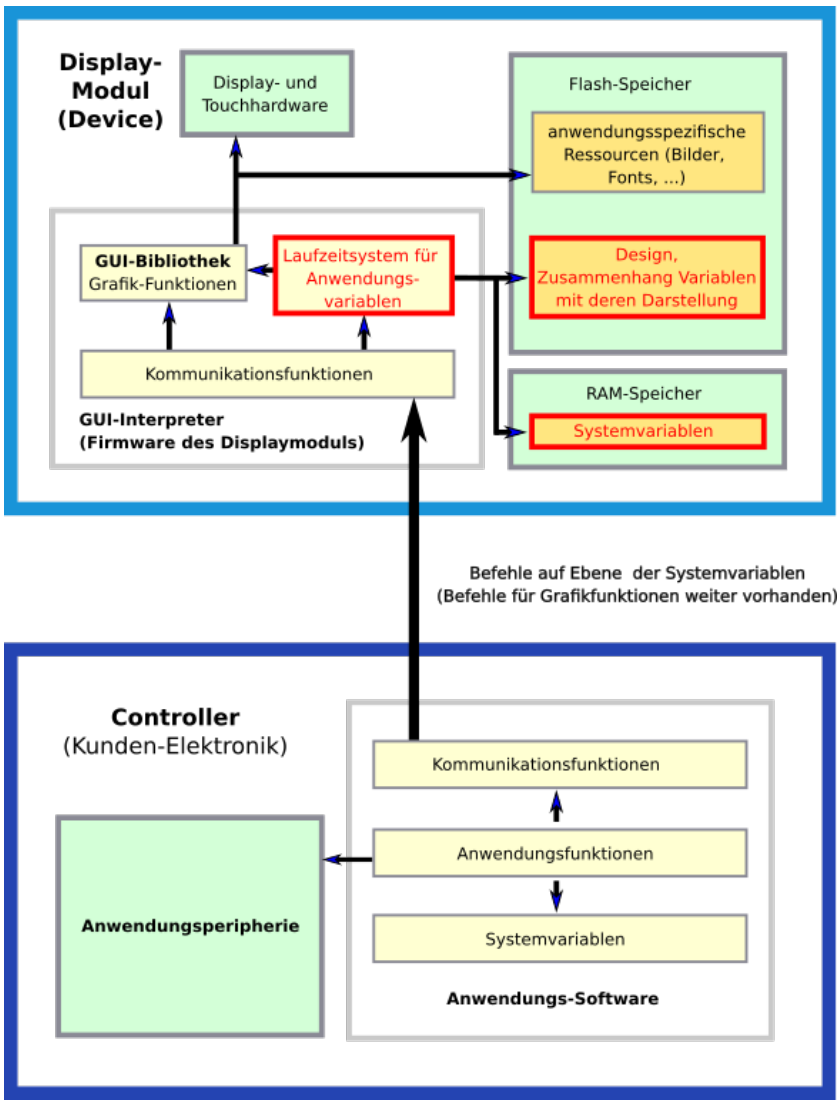


Abbildung 2: Controller sendet Systemvariablen

Das Design mit seinen grafischen Eigenschaften und der Gestaltung im Zusammenhang mit den **Systemvariablen** wird nun nicht mehr über die Programmierung in der Anwendung (im **Controller**) realisiert, sondern extern auf einem PC-Programm, dem GUI-Designer, erstellt.

Dieses Design wird dann in das Display-Modul (**Device**) hochgeladen und dort abgearbeitet. Von der Anwendung heraus kann der Wert einer Systemvariable leicht geändert werden – die Anzeige aktualisiert sich dann auf dem Display ohne weiteres Zutun des Controllers.

Hier funktioniert also alles nach dem Konzept:

„Anwendung, sage mir, was der Stand der Dinge ist – ich kümmere mich um die Darstellung!“

Kompatibilität: Durch diese Erweiterung geht die ursprüngliche Funktionalität nicht verloren. Die Befehle des **GUI-Interpreters** stehen weiter zur Verfügung und erlauben es dem Controller, von dort beliebige grafische Eingriffe auf dem Display zu machen. Dies erlaubt es, ganz spezielle und auch unübliche grafische Elemente zu generieren, die speziell zu Ihrer Anwendung passen.

2. Los geht's: Starten mit dem GUI-Designer - ein Beispiel

Tipp: Die GUI-Designer Referenzkarte gibt eine Übersicht über das Programmfenster und die verwendeten Begriffe.

Der GUI-Designer erleichtert das Entwickeln einer benutzerfreundlichen Bedienoberfläche für Ihre Anwendung mit LCM-Display-Modulen. An die Stelle des Programmierens der Benutzeroberfläche mit zahlreichen C-Funktionen tritt nun weitgehend ein Konfigurieren, so dass viel Entwicklungszeit gespart werden kann und auch in der Entwurfsphase leicht Prototypen erstellt werden können.

Im folgenden soll in einem kleinen Beispiel gezeigt werden, wie die Benutzeroberfläche einer Anwendung mit dem GUI-Designer gestaltet werden kann.

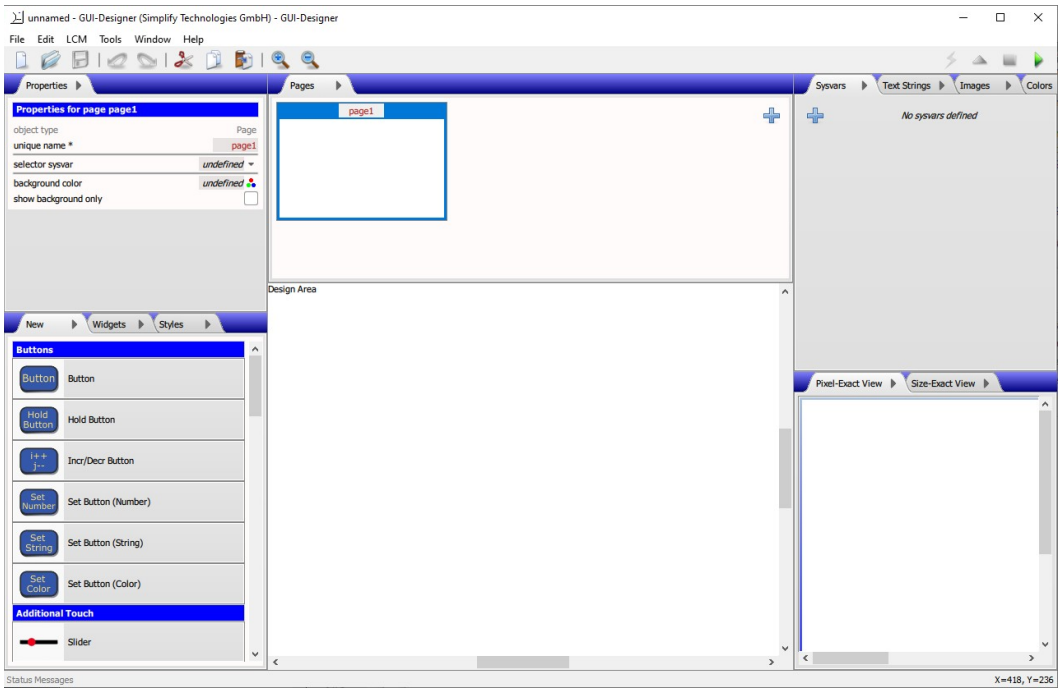
Beispiel: eine Temperatur-Anzeige

Es soll eine Anzeige für die Innen- und Außentemperatur eines Gebäudes gestaltet werden. Dabei sollen beide Temperaturen sowohl als Bargraph als auch als Zahl angezeigt werden. Ein kleiner Hilfe-Screen soll erläutern, was auf dem Haupt-Screen zu sehen ist.

Da eine einfache Möglichkeit für das Design statischer Objekte die Verwendung eines Hintergrundbildes ist, sollen für beide Seiten jeweils ein Hintergrundbild verwendet werden.

2.1 Starten des GUI-Designers, Anlegen des Projekts

Der GUI-Designer braucht nicht installiert zu werden. Ein Doppelklick auf die Datei `gui-designer.exe` startet den GUI-Designer mit einem neuen, leeren Projekt:



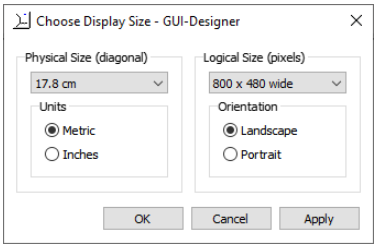


Abbildung 3: Display-Größe bestimmen – LCM > Display Size

Als erstes wird definiert, was für ein Display eingesetzt werden soll, damit der GUI-Designer die Auflösung und das Seitenverhältnis der Anwendung kennt. Stellen Sie hier im Menüpunkt *LCM > Display Size* (Abbildung 3) ein:

- *Units = Metric*
- *Physical Size (diagonal) = 17.8 cm*
- *Orientation = Landscape*
- *Logical Size (pixels) = 800x480 wide.*

Name der ersten Seite ändern.

Die erste Seite (**Page**) ist bereits angelegt. Wir ändern den Namen, indem wir auf den Namen (`page1`) klicken, und dort den neuen Namen `mainpage` eintragen. Auch möglich: auf den Namen im Tab *Properties* (Eigenschaft `unique name`) klicken und dort den Namen ändern.

Was ist ein **unique name**?

Bemerkung: Alle Objekte, die mit dem GUI-Designer verwendet werden, benötigen einen **eindeutigen Namen** (`unique name`), so dass auf sie von anderen Objekten Bezug genommen werden kann. Diese Benennung erfolgt im GUI-Designer und hat nichts mit eventuellen Dateinamen von Bildern oder anderen verwendeten **Ressourcen** zu tun. Zunächst vergibt der GUI-Designer immer einen Standardnamen. Ein eindeutiger Name unterliegt den Regeln eines Variablennamens, z.B. er darf keine Leerzeichen oder Umlaute enthalten.

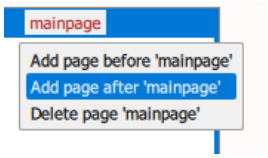


Abbildung 4: Neue Seite anlegen

Als nächstes legen wir eine weitere Seite an. Dies kann entweder durch Drücken des großen ✚ -Zeichens in der Ansichtsliste der Seiten (*Pages*-Tab) geschehen oder durch Drücken der rechten Maustaste irgendwo auf der Seite, um das **Kontextmenü** aufzuklappen und *Add page after 'mainpage'* auszuwählen (Abbildung 4). Diese Seite wird in `helppage` umbenannt.

Projektdatei speichern.

Über das Menü *File > Save* sichern wir die neue Projektdatei unter einem passenden Namen, z. B. `getting_started.sgd`. Wir öffnen den Ordner, wo `gui-designer.exe` liegt. Dort öffnen wir den bereits existierenden Unterorder `GettingStarted` und legen dort die neue Projektdatei an.

2.2 Hinzufügen von Bildern

Abbildung 5: Neues Bild (Image) dem Projekt hinzufügen

Um die Bilder für die Hintergründe und Buttons verwenden zu können, fügen wir sie dem Projekt hinzu. Dazu drücken wir, ähnlich wie beim Anlegen einer neuen Seite, das ✚ -Zeichen auf der Liste der *Images* (Tab *Images*) Dadurch erscheint ein neues **Image** Objekt in der Liste (Abbildung 5).

Durch Drücken des Ordner-Symbols erscheint ein *Datei-auswahldialog*, mit dem das gewünschte Bild zum Projekt hinzugefügt werden kann. Durch Klick auf den Standardnamen image1 kann hier ein passenderer Name eingegeben werden.

Die Eigenschaft `alpha color` kann hier undefiniert bleiben, da für dieses Beispiel keine Transparenz benötigt wird.

Die für diese Beispiel verwendeten Bilder befinden sich bereits im Unterordner `images` des Projekts. Fügen Sie nun die restlichen im Beispiel verwendeten Bilder hinzu, und vergeben Sie passende Namen, bis die Liste der *Images* so aussieht, wie in Abbildung 6.

Abbildung 6: Alle Bildressourcen

An dieser Stelle ist noch nicht definiert, wie die Bilder zum Einsatz kommen, sie stellen jetzt lediglich **Bildressourcen** dar, die im Projekt verwendet werden können.

Entsprechend würde mit Zeichensätzen (**Font-Ressourcen**) verfahren. Für unser Beispiel genügen die Standardzeichensätze, die bereits enthalten sind, so dass wir hier keine Zeichensätze hinzufügen müssen.

2.3 Anlegen der Hintergrundbilder

Die **Hintergrundbilder** werden, im Vergleich zu anderen Bilder, separat auf dem Hintergrund (Background) angelegt, damit garantiert wird, dass die Bilder immer zuerst dargestellt und von allen andern Objekte überlappt werden. Normalerweise gibt es ein einziges Hintergrundbild pro Seite, das die gesamte Fläche der Seite bedeckt. Der Hintergrund darf aber auch aus beliebig viele Bilder zusammengestellt werden.

Um Hintergrundbilder hinzuzufügen, müssen die Seiten in den **Hintergrund-Modus** umgestellt werden. Hierzu wird im Tab *Properties* die Eigenschaft `show background only` angewählt (Abbildung 7). Wenn diese Eigenschaft aktiv ist, sieht man auf jeder Seite nur deren Hintergrundbilder und es ist nur möglich **statische Bilder** (*Image*-Widgets) zu erzeugen. Alle andere **Widget-Typen** verschwinden aus dem Tab *New*.

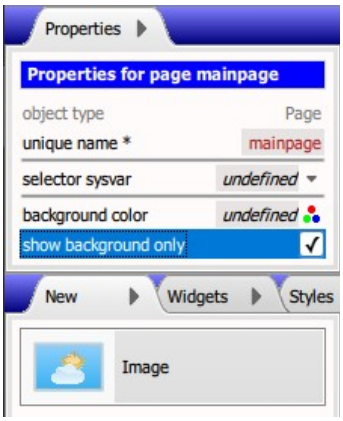


Abbildung 7: Hintergrund-Modus aktivieren

Um ein Hintergrundbild zu erstellen, zieht man aus dem Tab *New* ein *Image*-Widget auf die *Design Area* (wobei zunächst die `mainpage` ausgewählt sein sollte).

Es erscheint ein Standardbild und wenn dieses per Maus ausgewählt wird, kann man im Tab *Properties* die Eigenschaften des Bildes sehen (Abbildung 8).

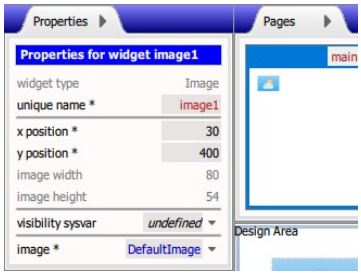


Abbildung 8: Eigenschaften eines Image-Widgets

Bemerkung: Hier wird zum ersten mal ein Objekt auf eine Seite gezogen und dort positioniert. Diese Objekte nennen wir **Widgets**. Sie haben weitere **Eigenschaften** (**Properties**), insbesondere eine Position auf der Seite und eine Größe. Das Hintergrundbild ist also ein solches **Widget**. Jedes Widget hat ebenfalls einen eigenen, **eindeutigen Namen** (`unique name`).

Bei den *Properties* wird nun eingegeben:

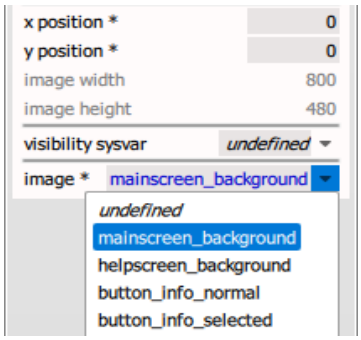


Abbildung 9: Eigenschaften des Widgets anpassen

- `x position = 0`
- `y position = 0`
- das in der Eigenschaft `image` angegebene `DefaultImage` wird geändert, indem man über den Pfeil (▼) rechts davon die Liste der verfügbaren **Image**-Objekte ausklappt und das entsprechende auswählt (siehe Abbildung 9).

Das Hintergrundbild erscheint nun in der aktuell editierten Seite.

Analog wird mit der `helppage` verfahren.

Im Ergebnis ist nun in Abbildung 10 folgendes zu sehen (die Liste der *Pages* zeigt entsprechende Vorschauen):

Beide Seiten haben jetzt Hintergrundbilder. Die Eigenschaften des zweiten Hintergrundbilds wurden im Tab *Properties* bereits angepasst.

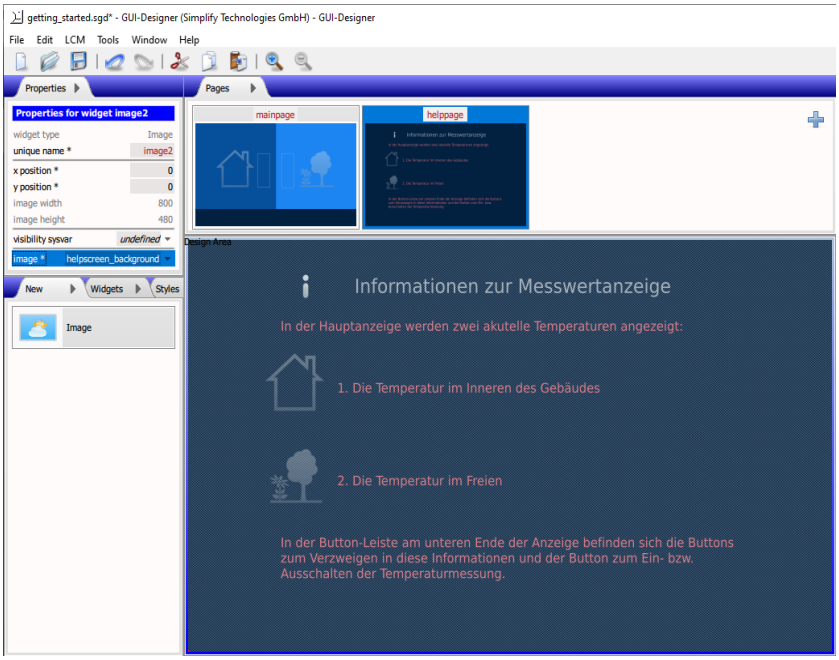


Abbildung 10: Eigenschaften des Hintergrundbilds der Seite helppage

Die Eigenschaften einer Seite anzeigen.

Wir klicken in die Titelleiste der Vorschau `mainpage`, um auf diese zu wechseln. Wir können eigentlich überall in die Vorschau klicken, um die Seite zu wechseln, aber wenn wir in die Titelleiste klicken, erscheinen im Tab *Properties* wieder die **Page-Eigenschaften** und nicht mehr die Eigenschaften des zuletzt selektierten Widgets. Hier deaktivieren wir `show background only`, da wir mit der Hintergrundgestaltung fertig sind. Im Tab New werden jetzt wieder alle **Widget-Typen** verfügbar.

2.4 Anlegen eines *Buttons*

Einen Standard-Button erstellen und ausprobieren.

Wir erstellen jetzt einen Button mit dem in die `helppage` verzweigt werden soll, indem wir im Tab *New* im Abschnitt *Touch* ein *Button*-Widget auf die *Design Area* ziehen (d.h. in diesem Fall auf die Oberfläche der `mainpage`). Ein **Standard-**

Button erscheint, den man bereits auf den Ansichten *Pixel-Exact View* und *Size-Exact View* betätigen kann.

Tipp: Position durch das Ziehen des Buttons in der *Design Area* verändern, oder genauer im Tab *Properties* einstellen.

In den seltensten Fällen wird man den Button so belassen wollen, wie er standardmäßig erstellt wird. Durch Änderungen in den *Properties* kann man ihn modifizieren. Dazu wählt man den Button in der *Design Area* mit der Maus aus. Probieren Sie aus, was passiert, wenn Sie die Position, die Abmessungen und die Beschriftung des Buttons ändern!

Standard-Abbildung eines Buttons durch Stile (Styles) anpassen.

Der hier verwendete Button ist weitgehend parametrierbar. Er lässt sich relativ schnell im Aussehen über seine Eigenschaften und die Eigenschaften seiner **Stile** (**Styles**) ändern. Oft wird ein Designer mit den Möglichkeiten der Parametrierung alleine nicht alle seine Vorstellungen realisieren können. Aus diesem Grund gibt es weitere Optionen für die Buttons, insbesondere die Möglichkeit, den Button und seine Zustände – `normal`, `selected` und `ghosted` – durch Bilder darzustellen.

Standard-Abbildung eines Buttons durch Bilder ersetzen.

Das Aussehen des Buttons soll jetzt durch verschiedene Bilder realisiert werden. Dazu wird der Button selektiert und dann in den *Properties* für das `image` und `selected image` eines der im **Aufklappmenü** angezeigten Bilder ausgewählt (siehe Abbildung 11).

Wählen Sie für dieses Beispiel

- `image` = `button_info_normal` und
- `selected image` = `button_info_selected`.

Man kommt dann zu folgender Ansicht:

Tipp: Wenn möglich, Eigenschaften über das Aufklappmenü setzen, um Tippfehler zu vermeiden.

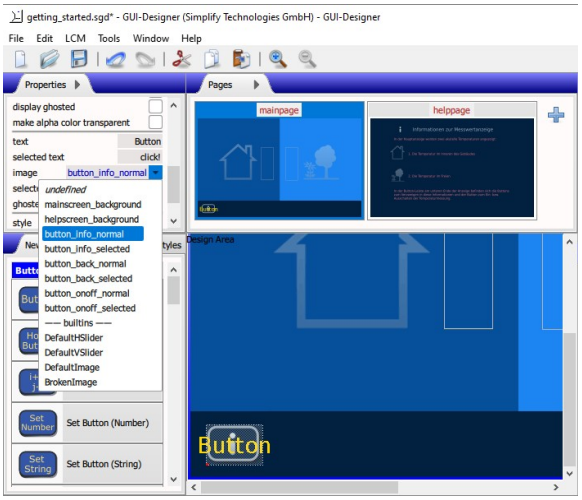


Abbildung 11: Eigenschaft `image` wird über das Aufklappmenü gesetzt

Da Buttons zusätzlich zur Grafik auch eine Text-Beschriftung enthalten können, sind jetzt noch die Beschriftungen des Buttons sichtbar, jeweils für den Normal-Zustand und für den ausgewählten Zustand. In diesem Beispiel ist die zusätzliche

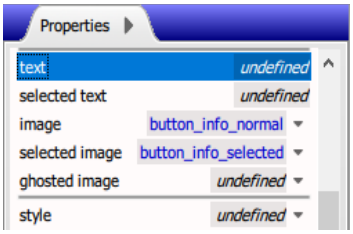


Abbildung 12: Bei Eigenschaften mit gelöschtem Wert erscheint `undefined`

Beschriftung überflüssig und wird durch Löschen der Texte in den Eigenschaften `text` und `selected text` entfernt. Es soll jetzt das Wort *`undefined`* als Wert für beide Eigenschaften erscheinen (Abbildung 12). Hiermit wird die grafische Darstellung des Buttons abgeschlossen.

2.5 Navigation durch die *Pages* mit *Buttons*

Da Buttons oft verwendet werden, um in einer Anwendung von einer Seite zur nächsten zu gelangen, wird dies bereits vom GUI-Designer unterstützt.

Die Eigenschaft `destination` setzen.

Ein **Standard-Button** hat hierzu die Eigenschaft `destination`. Durch Druck auf den Pfeil (▼) rechts erscheint eine Liste aller Seiten als Ziel. Wir wählen die Seite `helppage` aus der Liste.

Wenn Sie nun in einer Ansicht (*Pixel-Exact View* oder *Size-Exact View*) den Button drücken, wird automatisch auf die angegebene Zielseite verzweigt.

Eine Navigation durch verschiedene Seiten lässt sich leicht realisieren und ausprobieren.

Um von dort wieder zurück zur Hauptseite zu gelangen, legen wir dort ebenfalls einen Standard-Button an (unter Verwendung der Bilder `button_back_normal` und `button_back_selected`) und geben als `destination` die `mainpage` an (siehe Abbildung 13). Nun können wir per Button zwischen den beiden Seiten hin- und herschalten.

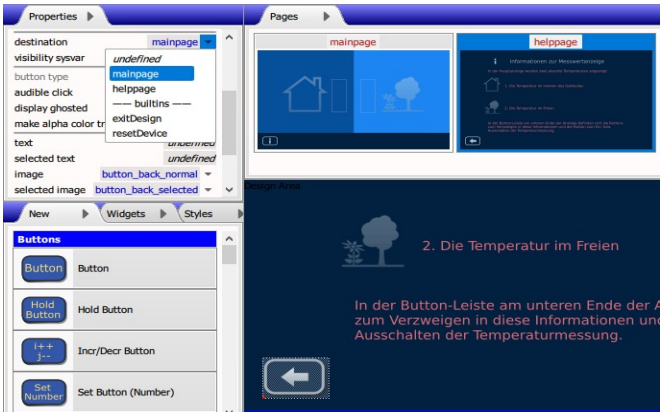


Abbildung 13: Die Button-Eigenschaft `destination` wird gesetzt, um zurück zur Hauptseite zu gelangen

2.6 Anlegen eines *Hold-Buttons*

Einen Hold-Button erstellen.

Um die Temperaturmessung über das Display-Modul ein- und ausschalten zu können, erstellen wir auf dem `mainpage` noch einen Button. Hierzu brauchen wir allerdings einen **Hold-Button**, d.h. ein *Hold-Button*-Widget im Tab *New* im Abschnitt *Touch*. Wie beim **Standard-Button** ersetzen wir den Text durch Bilder, in diesem Fall `button_onoff_normal` und `button_-`

`onoff_selected`. Das Verhalten eines Hold-Buttons darf ebenfalls auf den Ansichten *Pixel-Exact View* und *Size-Exact View* ausprobiert werden.

2.7 Systemvariablen und ihre Darstellung

In jedem Projekt gibt es Größen aus der „realen“ Welt, die auf dem Display-Modul dargestellt werden soll. Diese Größen können durch **Systemvariablen** (gekürzt: **Sysvars**) abgebildet werden. Wir legen Systemvariablen im Tab *Sysvars* an.

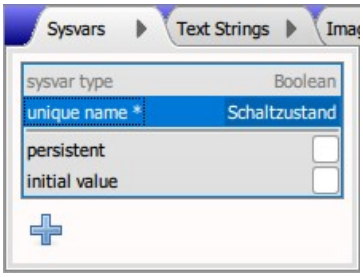


Abbildung 14: Systemvariable Schaltzustand anlegen

Wir definieren zuerst eine Systemvariable vom Typ `Boolean` und nennen sie `Schaltzustand` (Abbildung 14). Diese Systemvariable wird nun dem **Hold-Button** zugeordnet. Wir selektieren wieder den Hold-Button und ändern in *Properties* die Eigenschaft `sysvar`. Hierzu wählen wir im **Aufklappmenü** die neue Variable `Schaltzustand` aus (Abbildung 15). (Bemerkung: Es werden bei einem Hold-Button nur die `Boolean`-Sysvars angeboten.)

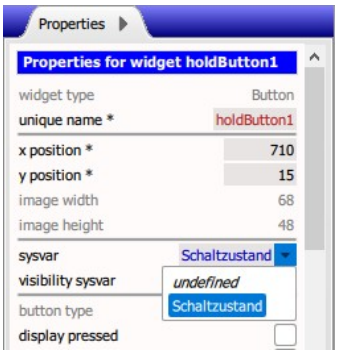


Abbildung 15: Systemvariable dem Hold-Button zuweisen

Wenn wir nun den Hold-Button in einer Ansicht betätigen, schaltet der Button immer noch zwischen den Zuständen um und gleichzeitig ändert sich automatisch auch die Eigenschaft `initial value` in der Systemvariable. Umgekehrt können wir diese Eigenschaft an- bzw. abhaken, um den Zustand des Buttons zu testen. Im aktuellen Betrieb informiert die Systemvariable die Anwendung darüber, ob der Benutzer die Temperaturmessung ein- bzw. auschalten will. (Dies geschieht durch **Events**, im Handbuch Abschnitt Zusammenarbeit zwischen Design und Controller beschrieben.)

Wir legen für unsere Anwendung zwei weitere Systemvariablen vom Typ `Number` an und nennen sie `Aussentemperatur` und `Innentemperatur`.

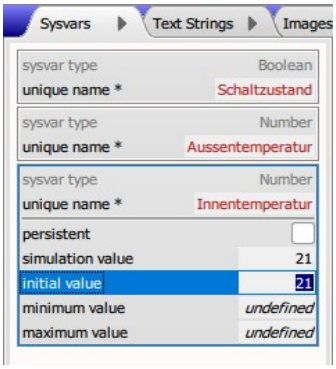


Abbildung 16: Numerische Systemvariablen anlegen.

Es lässt sich für diese Variablen neben anderen Eigenschaften sowohl ein `initial value` als auch ein nur zum Testzweck verwendeten `simulation value` einstellen, die später angezeigt werden können. Wir stellen für beide Temperaturen plausible Werte ein, wie in Abbildung 16 dargestellt.

Die zwei neue Systemvariable müssen nun eine Darstellung auf dem Display bekommen.

2.7.1 Darstellung von Variablen als Zahlen

Was ist ein *Number*-Widget?

Eine mögliche Repräsentation für eine numerische Systemvariable ist eine **Zahl** (*Number*-Widget). Dieses stellt eine Systemvariable als Zahl dar, zusammen mit einem String für eine Einheit (z. B. %, `°C`, €, `m`).

Number-Widget erstellen.

Das *Number*-Widget findet sich im Abschnitt *Text* im *New*-Tab. Wir ziehen also ein *Number*-Widget auf die Oberfläche der Seite unter dem Rechteck neben dem Haus.

Ein *Number*-Widget passt sich immer den aktuellen Wert seiner `sysvar` an.

Nun muss bestimmt werden, welche Systemvariable vom *Number*-Widget anzuzeigen ist. Das geschieht, indem man die entsprechende Sysvar in den Eigenschaften des Widgets auswählt. Hier stellen wir die Eigenschaft `sysvar` auf dem Systemvariable `Innentemperatur`. Das *Number*-Widget zeigt sofort den Simulationswert der Systemvariable an.

Die Maßeinheit `unit` stellen wir von `%` auf `°C` um.
Dann werden Position und Größe angepasst. (Bemerkung: Die Größe kann nicht automatisch bestimmt werden, da zu diesem Zeitpunkt nicht klar ist, wie groß die Zahl und damit der zur Darstellung notwendige Raum evtl. wird.)

Das Aussehen des Textes wird ebenfalls angepasst, wie in Abbildung 17 gezeigt:

Die gewünschte Systemvariable und grafische Eigenschaften des *Number*-Widgets einstellen.

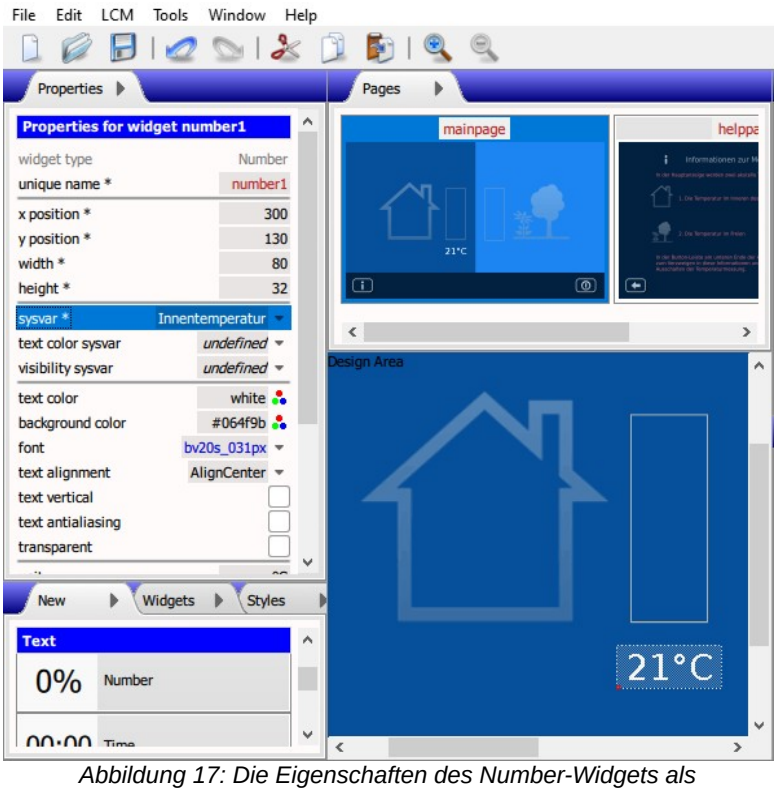


Abbildung 17: Die Eigenschaften des Number-Widgets als Repräsentation der Systemvariable `Innentemperatur`

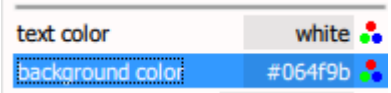


Abbildung 18: Farbeingenschaften mit RGB-Symbol

Tipp: Um Farben leichter einzustellen, gibt es ein **RGB-Symbol** ( ) rechts neben dem Farbwert der Eigenschaft (Abbildung 18). Wenn die Maus darüber schwebt, erscheint die aktuelle Farbe als Farbmuster links daneben. Wenn man darauf klickt, öffnet sich das **Farbdialog**. Hier kann man z.B. die Hintergrundfarbe mit der Schaltfläche *Pick Screen Color* (*Farbe vom Bildschirm wählen*) einfach vom Hintergrundbild übernehmen

(Abbildung 19).

Das **Farbdialog** erleichtert die Auswahl einer Farbe.

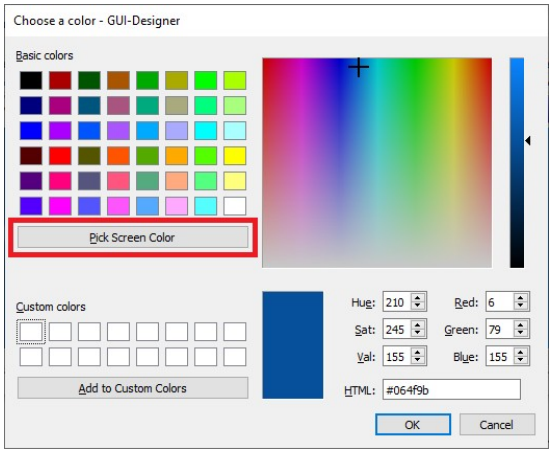


Abbildung 19: Hintergrundfarbe per Knopf vom Bildschirm wählen

Ein Widget kopieren und einfügen, um ein neues Widget zu erzeugen.

Für die Außentemperatur verfahren wir ähnlich. Um Zeit zu sparen wählen wir das *Number*-Widget für die Innentemperatur an und erzeugen ein weiteres mittels Kopieren und Einfügen (*Edit > Copy Widget* und *Edit > Paste Widget* bzw. die übliche Tastenkombinationen). Das neue *Number*-Widget landet direkt über dem alten und bekommt automatisch einen neuen **eindeutigen Namen**. Es müssen dann für diese neue Repräsentation nur die `x position` (`420`), die `sysvar`-Zuordnung und die Hintergrundfarbe (weil sie nun rechts im Bild liegt!) angepasst werden.

2.7.2 Die Reihenfolge der Widgets ist wichtig!

Wichtig! Die Reihenfolge der *Widgets* bestimmt die Überlappung!

Auf einer Seite werden verschiedene Elemente (**Widgets**) dargestellt, die sich auch überlappen können, wie im Beispiel die beide Kopien des *Number*-Widgets. Die Reihenfolge, in der sie gezeichnet werden und damit die Art, wie sie überlappen, wird zunächst durch die Reihenfolge ihrer Erstellung definiert.

Reihenfolge der *Widgets* durch **Drag & Drop** verändern.

Man kann die Reihenfolge ändern, indem man im Tab *Widgets* die Reihenfolge der Widgets durch Ziehen und Loslassen mit der Maus (Drag & Drop) verändert. Das Widget, das zuerst in der Liste steht, wird als erstes gezeichnet und dann unter Umständen von den folgenden Widgets überdeckt.

2.7.3 Darstellung einer Zahl als Balkenanzeige

Eine weitere mögliche Repräsentation für eine numerische Systemvariable ist ein Balken, dessen Höhe sich mit dem Wert der Variable ändert. Für die Temperaturanzeige möchten wir nicht nur die numerische Darstellung (**Zahl**), sondern auch eine **Balkenanzeige** mithilfe vom *BarIndicator*-Widget.

Balkenanzeige anlegen und mit einer `sysvar` verbinden.

Das *BarIndicator*-Widget findet sich im Abschnitt *Additional Dynamic Visuals* im *New*-Tab. Wir ziehen ein solches

BarIndicator-Widget auf das Rechteck neben dem Haus und verbinden es mit der gewünschten Systemvariable `Innentemperatur` (Abbildung 20). Die vorab eingestellte Innentemperatur wird sofort übernommen und dargestellt.

Beim Verbinden mit einer Systemvariable wird sofort den aktuellen Wert der `sysvar` dargestellt.

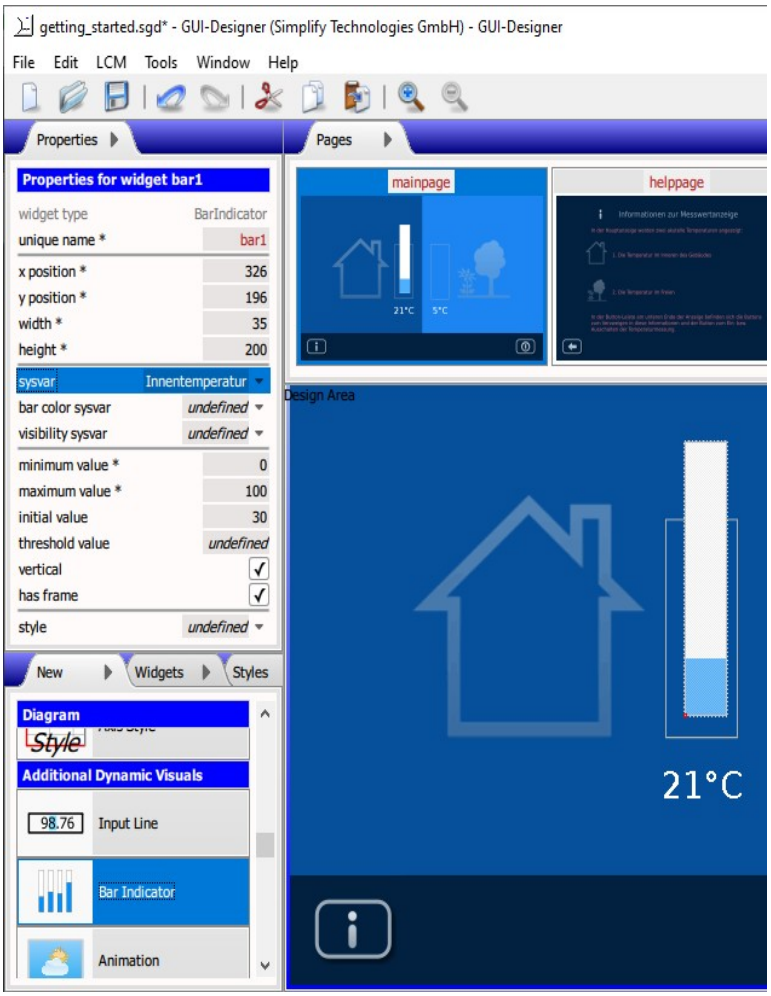


Abbildung 20: Die Balkenanzeige wird mit der Systemvariable `Innentemperatur` verbunden

Eigenschaften der Balkenanzeige anpassen.

Anschließend passen wir die Position und Größe der Balkenanzeige so an, dass sie in das vorgezeichnete Rechteck passt (in diesem Fall: `x position` = `310`, `y position` = `180`, `width` = `60`, `height` = `160`).

Um den Wertebereich, in dem die Balkenanzeige arbeitet, einzustellen, setzen wir deren Eigenschaften `minimum value` und `maximum value`. Sie bezeichnen die Werte, die einer minimalen bzw. maximalen Auslenkung der Balkenanzeige entsprechen. Für ein Thermometer wären hier z. B. `-20` und `50` (°C) sinnvoll.

Zweite Balkenanzeige mittels Kopieren und Einfügen erzeugen.

In gleicher Weise, wie beim *Number*-Widget, kopieren wir die Balkenanzeige in die Zwischenablage und fügen diese Kopie wieder ein. Die zweite Balkenanzeige liegt direkt über die erste.

Wir ändern die `x position` auf `430`, um sie an der richtigen Stelle zu bringen. Die zweite Balkenanzeige soll noch mit dem `sysvar` `Aussentemperatur` verknüpft werden. Alle andere Eigenschaften sind schon korrekt. Das sieht dann in Abbildung 21 so aus:

Die Eigenschaften der zweiten Balkenanzeige weichen nur in der X-Position und `sysvar`-Verknüpfung von der ersten Balkenanzeige ab.

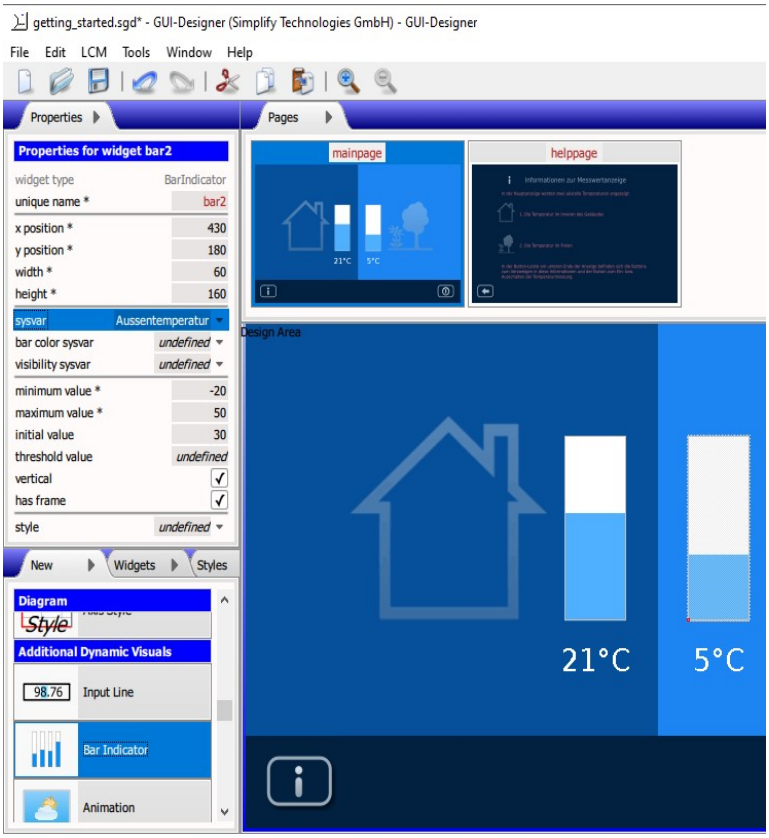


Abbildung 21: Die Eigenschaften der Balkenanzeige für die Systemvariable `Aussentemperatur`

`sysvar`-Verknüpfungen testen

Sobald man im Tab *Sysvars* den `simulation value` einer Systemvariable ändert, ändert sich die Darstellung auf dem Display. Hiermit kann die Darstellung getestet werden. (Später erfolgt die Änderung nicht über den GUI-Designer, sondern wird von der Anwendung selbst in das Display-Modul übertragen – mit demselben Resultat).

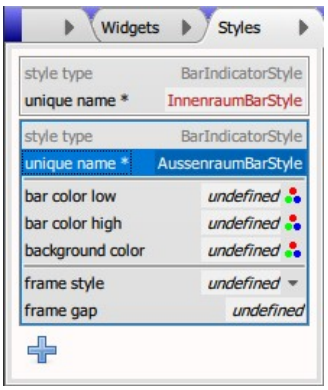


Abbildung 22: Zwei Stile für die Balkenanzeigen anlegen

2.8 Stile und Farben am Beispiel der Balkenanzeige

In den meisten Fällen wird man das Aussehen, z.B. die Farbe der **Balkenanzeige** (und auch anderer **Widgets**) verändern wollen.

Damit dies konsistent möglich ist, und man nicht z.B. jeden einzelnen Button manuell anpassen muss, sind solche Eigenschaften in sogenannten **Stile** (**Styles**) abgelegt. Diese können zunächst unabhängig von einem Widget erstellt werden und dann beliebig vielen Widgets des entsprechenden Typs

zugewiesen werden.

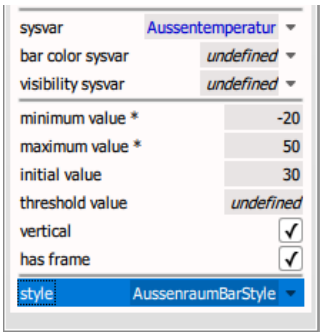


Abbildung 23: Einen Stil einer Balkenanzeige zuweisen

Im vorliegenden Beispiel möchten wir unterschiedlich aussehende Balkenanzeigen, dies wird mit zwei Stilen erreicht. Dazu werden im Tab *Styles* zwei neue **Stil**-Objekte vom Typ `BarIndicatorStyle` angelegt. Wir benennen diese Stile entsprechend, z. B. `InnenraumBarStyle` und `AussenraumBarStyle`, wie in Abbildung 22.

In den Balkenanzeigen weisen wir über die Eigenschaft `style` diesen die neuen Stile zu, wie in Abbildung 23 dargestellt.

Nun können wir in den Eigenschaften des Stils z.B. die Farben einstellen. Es gibt verschiedene Möglichkeiten eine Farbe zu bestimmen:

Es gibt mehrere Methoden eine Farbe zu bestimmen.

- Durch einen Klick auf das **RGB-Symbol** ( ) können Farben über das **Farbdialog** ausgewählt werden.
- Im Textfeld kann der Farbwert direkt als Hex-Zahl (mit vorgestelltem '#') eingetippt oder kopiert werden.
- Im Textfeld kann ein Standardname eingetippt oder kopiert werden, wie er vom W3C als anerkannte **CSS-Color-Keyword** definiert wurde. Viele Farben haben einfache englische Namen wie z.B. `red`, `blue`, `teal` usw. (für eine Liste der Standardnamen siehe https://www.w3.org/wiki/CSS/Properties/color/keywords).
- Wenn das Projekt **benutzerdefinierte Farben** enthält, kann eine benutzerdefinierte Farbe aus dem **Aufklappmenü** ausgesucht werden.

Benutzerdefinierten Farben werden im Tab *Colors* erstellt.

Bemerkung: Das Aufklappmenü der letzten Methode wird nur dann sichtbar, wenn benutzerdefinierte Farben tatsächlich vorhanden sind. In diesem Beispiel-Projekt ist dies (noch) nicht der Fall. Um das Feature auszuprobieren, können ein paar benutzerdefinierte Farben im Tab *Colors* erstellt werden.

Für unser Beispiel stellen wir in den Stilen die Eigenschaften `bar color low` und `background color` ein und testen diese. Um `bar color high` auszuprobieren, muss zunächst die Eigenschaft `threshold value` (Grenzwert) in der Balkenanzeige gesetzt werde. Der aktuellen Wert der verknüpften Systemvariable muss diesen Grenzwert überschreiten, damit diese Farbe ebenfalls erscheint. Die Wirkung der Änderungen werden sofort sichtbar, wie in Abbildung 24 am Aussenraum-Balken dargestellt.

Änderungen der Stilfarben wirken sich sofort auf den/die zugewiesenen Widget/s.

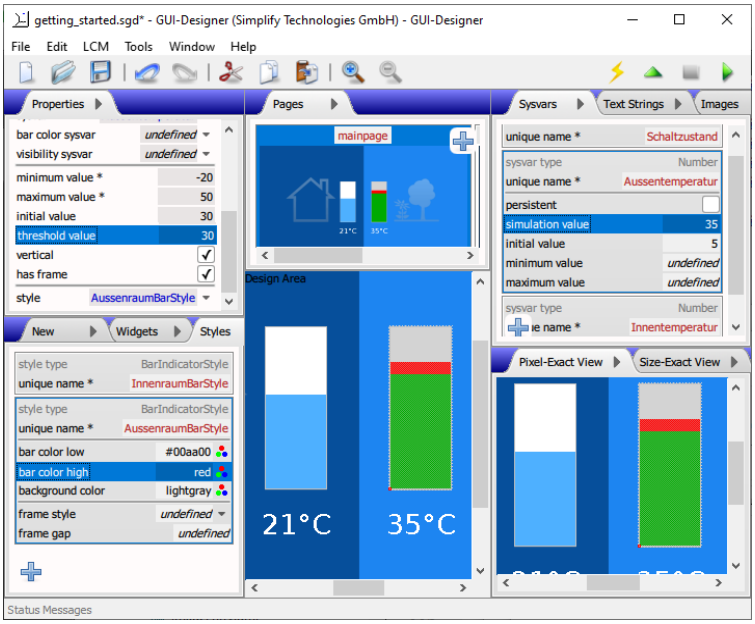


Abbildung 24: Spielerei mit Farben, Grenzwert und Systemvariablen

Alternativ-Verfahren: Farbe der Balkenanzeige über die Anwendung bestimmen.

Bemerkung: Es ist auch möglich, die Farbe der Balkenanzeige von der Anwendung aus über eine Systemvariable einzustellen. Hier wird dann `bar color low` (im Stil) evtl. nur als Standardwert verwendet. Zum ausprobieren: eine Systemvariable vom Typ `Color` erzeugen und diese der Balkenanzeige-Eigenschaft `bar color sysvar` zuweisen. Die Systemvariable kann, wie immer, durch das Setzen der Eigenschaft `simulation value` getestet werden.

Stilgruppe durch das Ziehen auf Widgets erzeugen.

Abschließende Bemerkung über Stile: Ein Standardsatz von Stile (d.h. eine **Stilgruppe**) für einen bestimmten **Widget-Typ** kann auch angelegt werden, indem man diese aus dem *New*-Tab (Abschnitt *Style Groups*) auf die Oberfläche der *Design Area* zieht. Wenn man sie direkt auf ein passendes Widget zieht, werden die erzeugte Stile automatisch diesem Objekt zugewiesen und ein zum Widget passender Name gewählt.

2.9 Fertig

Text-Widget anlegen und anpassen.

Zum Schluss fügen wir noch eine Überschrift hinzu, um die Anwendung zu vervollständigen. Dazu ziehen wir aus dem *Text*-Abschnitt im *New*-Tab ein **Text** (*Text*-Widget) auf die *Design Area*. Im Tab *Properties* setzten wir die Eigenschaften, damit der *Text*-Widget so aussieht, wie in Abbildung 25 abgebildet:

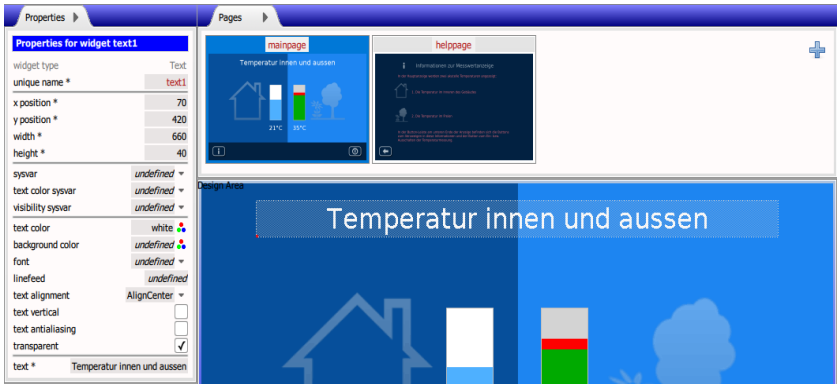


Abbildung 25: Eigenschaften des Text-Widgets anpassen

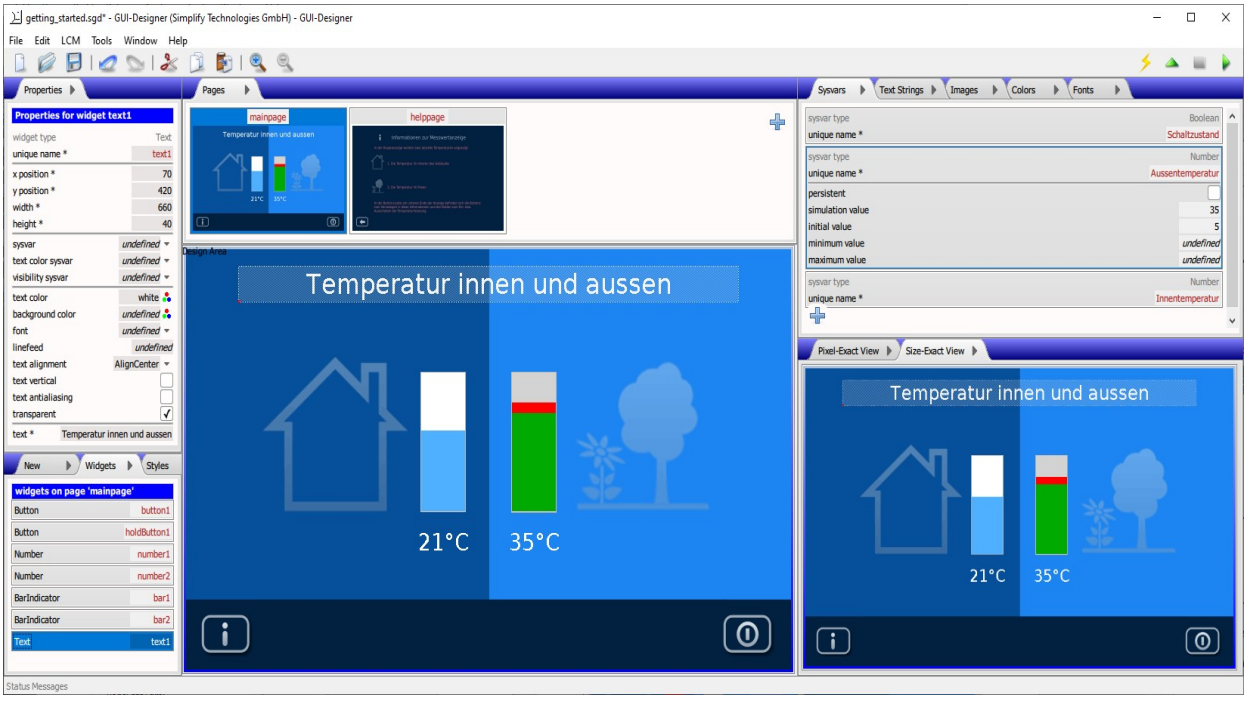
Inhalt und Farbe des *Text*-Widgets evtl. über die Anwendung bestimmen.

Bemerkung: beim *Text*-Widget können sowohl der Textinhalt als auch die Textfarbe durch **Systemvariable** von der Anwendung aus gesteuert werden. Für die Textfarbe verfahren wir, wie bei der **Balkenanzeige**. Für den variablen Textinhalt brauchen wir eine Systemvariable vom Typ `String` und weisen die Variable der *Text*-Widget-Eigenschaft `sysvar` zu. Hier soll darauf geachtet werden, dass das *Text*-Widget für alle vorgesehene Inhalte entsprechend groß angelegt wird.

Die **Mehrsprachigkeit** wird auch unterstützt!

Abschließende Bemerkung über Text: Wer auch alle Texte mehrsprachig darstellen möchte, lässt am einfachsten alle Textinhalte über **Text-Strings** darstellen (Tab *Text Strings*). Mehr dazu im Handbuch Kapitel **Mehrsprachigkeit**.

Zum Schluss sieht unser Design dann so aus:



Binär-Image-Datei für das LCM-Displaymodul erzeugen.

Mit dem Menüpunkt *File > Create getting_started_image_v2.bin* (bzw. ..._*v1.bin*) oder beim Klicken des Blitzsymbols (rechts in der Symbolleiste) exportiert der GUI-Designer eine Binärdatei zum Laden in die LCM-Display-Modulen (ab LCM3), sowie eine C/C++-Header-Datei mit Definitionen für Ihre Anwendung (davon werden meist nur die Sysvar-**Objekt-IDs** benötigt) und eine `csv`-Datei für die Unterstützung der **Mehrsprachigkeit**. Es erscheint auch ein Button links neben dem Blitzsymbol, der den Status der Binärdatei (OK bzw. Fehler und Warnungen) angibt. Mit einem Mausklick auf diesem Button oder mit dem Menüpunkt *File > View getting_started.log* können Details des Ablaufs dieses Prozesses inspiziert werden.

3. Verbindungen: Design + Display-Modul + Controller

Das **Design** ist die Benutzeroberfläche, bestehend aus mehreren Seiten (**Pages**).

Als **Design** (oder auch **Projekt**) wird im Folgenden eine Benutzeroberfläche bezeichnet, die mit dem GUI-Designer entworfen wurde. In der Regel besteht das Design aus mehreren Seiten, die als **Pages** bezeichnet werden.

Der **GUI-Interpreter** ist die Firmware auf dem Display-Modul, die u.a. den Austausch zwischen **Design** und **Controller** regelt.

Die Hauptaufgabe des GUI-Designers ist es ein Design aus **Basiskomponenten** (**Widgets**) zu erzeugen, das dann vom **GUI-Interpreter** (die Firmware auf den Display-Modulen) verstanden werden kann. Beispiele für Widgets sind: **Buttons**, **Schieberegler**, **animierte Bilder**, **Tastaturen**, aber auch einfache Elemente wie **Rechtecke** oder **statische Bilder**. Jedes Widget hat einen Satz von **Eigenschaften** (**Properties**), die optimal angepasst werden können, um die Bedürfnisse der speziellen Anwendung zu erfüllen.

Der **Controller** darf die grafische Darstellung und Touch-Detektion dem **Design/GUI-Interpreter** komplett überlassen.

Der GUI-Designer macht es einfach Seiten aufzubauen, zwischen ihnen zu navigieren, Benutzereingaben zu ermöglichen und Daten zwischen dem Design und der Anwendung mit Hilfe von **Systemvariablen** (**Sysvars**) auszutauschen. Hierdurch kann das direkte Erstellen der grafischen Benutzeroberfläche und die Detektion von Touch-Eingaben durch die Anwendung vermieden werden.

Testebene 1: Ein Design entwerfen und im GUI-Designer testen.

Viele Aspekte des Designs können während der Entwurfsphase getestet werden, indem man mit den Widgets in den Ansichten *Pixel-Exact View* und *Size-Exact View* des GUI-Designers „herumspielt“ oder die Systemvariablen (welche normalerweise durch die Anwendung verändert werden würden) im GUI-Designer direkt manipuliert.

Testebene 2: Der GUI-Designer mit einem Display-Modul verbinden, das Design hochladen und ohne Controller testen.

Natürlich muss man das Design trotzdem noch auf einem Display-Modul testen. In den Abschnitten Hardware anschließen sowie Kommunikation zwischen Hardware und GUI-Designer wird eine Verbindung zum Display-Modul aufgebaut. Ferner wird kurz das Konzept des **Startup-Objekts** erläutert.

Wenn diese Vorbereitungen einmal getroffen wurden, dann vereinfacht der GUI-Designer

- die Konvertierung des Designs in die Formate, die vom GUI-Interpreter und der Anwendung benötigt werden,
- das Hochladen des Designs auf ein Display-Modul und
- den Start bzw. den Test des Designs (eingeschränkter Ablauf des Designs ohne den Controller).

Diese drei Schritte werden im Abschnitt Design auf das Display-Modul übertragen diskutiert.

Testebene 3: Mit dem Controller testen.

Am wichtigsten ist die Kommunikation zwischen dem Controller und dem Design/GUI-Interpreter auf dem Display-Modul. Dies wird beschrieben im Abschnitt Zusammenarbeit zwischen

Design und Controller.

Zum Schluss gibt es einige das ganzen Design betreffenden Eigenschaften, die im File > Document Properties bestimmt werden können.

3.1 Hardware anschließen

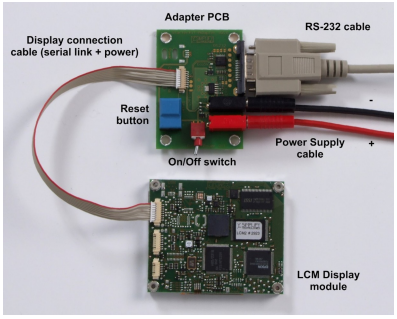


Abbildung 26: Adapter-Platine

Schließen Sie vor der Inbetriebnahme das Display-Modul an, wie in Abbildung 26 gezeigt:

1. Verbinden Sie das 8-polige Flachbandkabel der Adapterplatine mit dem entsprechenden Steckverbinder auf dem Display-Modul.

2. Verbinden Sie die Spannungsversorgungsanschlüsse der Adapterplatine mit einer geeigneten Spannungsquelle. Die benötigte Spannung beträgt 5,5V – 7,0V, die Stromaufnahme beträgt bspw. für ein 8-Zoll-Modul 1A. Kleinere Module benötigen deutlich weniger Strom.

   Die Adapterplatine ist mit einem Verpolungsschutz ausgestattet, die eine Beschädigung des Moduls bei Verpolung verhindert. Bitte achten Sie dennoch darauf, dass die Polarität korrekt ist und mit der Bezeichnung unten auf der Adapterplatine übereinstimmt, da sonst das Modul nicht funktioniert.

   Beachten Sie bitte, dass das Modul selbst keinen Verpolungsschutz besitzt.

   Bitte schalten Sie die Spannungsquelle erst ein, nachdem das Modul angeschlossen ist (kein „Hot-Plugging“).

3. Verbinden Sie die Sub-D-Buchse mit dem RS232-Kabel (das ist ein 1:1-Kabel) zur seriellen Schnittstelle des PCs. Wenn Ihr PC keine RS-232-Schnittstelle hat, können Sie auch einen USB-RS232-Wandler verwenden. Dadurch wird die Geschwindigkeit etwas herabgesetzt, verglichen mit einer direkten seriellen Verbindung. Dies liegt an Eigenschaften des USB-Protokolls und der spezifischen Implementierung der Windows-Treiber.

Der Taster auf der Adapterplatine dient zum Zurücksetzen des Moduls. Mit dem Kippschalter können Sie die Versorgungs-spannung ein- bzw. ausschalten.

3.2 Kommunikation zwischen Hardware und GUI-Designer

Damit der GUI-Designer mit dem Display-Modul kommunizieren kann, müssen passende Einstellungen auf beiden Seiten festgelegt werden. Die Parameter für den GUI-Designer (als

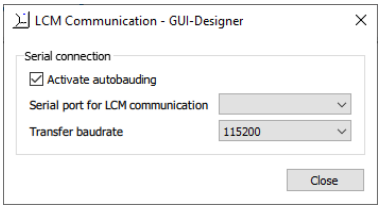


Abbildung 27: LCM > LCM Communication

Wechselwirkung der Einstellungen des **Startup-Objekts** mit denen des Dialogfensters *LCM Communication* beachten!

Pseudo-Controller) werden über den Menüpunkt *LCM > LCM Communication* eingestellt (Abbildung 27). Hier wird sowohl die serielle Schnittstelle (*Serial port for LCM communication*) als auch die gewünschte Übertragungsbaudrate (*Transfer baudrate*) des PCs gesetzt. Das Autobauding darf hier auch aktiviert oder deaktiviert werden (*Activate autobauding*).

Auf der anderen Seite arbeitet das Display-Modul standardmäßig mit **Autobauding**. Wer davon abweichen will, soll ein **Startup-Objekt** zum Design hinzufügen. Das Startup-Objekt wird im *File > Document Properties* aktiviert und konfiguriert – Details unter Startup-Objekt. Hier konzentrieren wir auf die Wechselwirkung der Einstellungen des Startup-Objekts mit denen des Dialogfensters *LCM Communication*.

Über das Startup-Objekt kann eine feste Baudrate für das Display-Modul gewählt werden. Soll dann mit einem solchen Modul kommuniziert werden, muss das Autobauding im Dialogfenster *LCM Communication* deaktiviert werden. Es gibt auch spezielle Varianten von Display-Modulen, die fest auf eine Baudrate eingestellt sind. Auch hier muss man das Autobauding deaktivieren.

Die Übertragungsbaudrate entspricht die Geschwindigkeit der Kommunikation zwischen dem GUI-Designer (als Pseudo-Controller) und dem Display-Modul. Die Eigenschaft `startup baudrate` im Startup-Objekt bezeichnet die Baudrate des Display-Moduls nach dem Einschalten. Die beiden Baudrates sollen nicht miteinander verwechselt werden.

Im Startup-Objekt gibt es auch ein **Autostart-Flag**. Wenn Autostart aktiviert ist, wird das Design direkt nach dem Einschalten gestartet. Hierzu wird kein Autobauding verwendet sondern nur die im Startup-Objekt angegebene `startup baudrate` (standardmäßig 115200).

3.2.1 Details: Autobauding

Normalerweise ist die Baudrate der Display-Module nicht festgelegt. Durch eine spezielle **Autobauding**-Sequenz, die vom Controller vorgegeben wird, passen diese sich an die Baudrate des Controllers an.

Es ist jedoch möglich das Autobauding der Module zu deaktivieren und eine feste Baudrate zu wählen. Das Display-Modul startet dann mit der eingestellten Baudrate. Ein Durchführen der Autobauding-Sequenz ist auf Seiten des Controllers nicht nötig. Man muss allerdings noch mit dem `zBIN`-Kommando in den Binärmodus schalten.

3.3 Design auf das Display-Modul übertragen

Nachdem die Hardware angeschlossen und mit dem GUI-Designer verbunden wurde, sowie evtl. ein **Startup-Objekt** konfiguriert worden ist, darf das **Design** auf das Display-Modul übertragen und dort getestet werden.

Um ein Design auf ein Display-Modul zu übertragen müssen folgende Schritte durchgeführt werden. Zu jedem Schritt gibt es ein Menüpunkt und ein entsprechendes Symbol auf der rechte Seite der Symbolleiste (Toolbar), wie hier abgebildet. Die Bezeichnung *<nnn>* wird durch den Namen des Projekts und das Fragezeichen (*?*) durch *1* oder *2* ersetzt. Zu v1 und v2 siehe auch den Abschnitt Tools-Menü.

1. Damit der **GUI-Interpreter** das mit dem GUI-Designer erzeugte Design verstehen kann, muss dieses in ein binäres Format umgewandelt werden. Hierzu erzeugt der GUI-Designer eine `bin`-Datei, in der alle benötigten Informationen zusammengefasst sind. Die Umwandlung erfolgt über das Blitzsymbol oder das Menüpunkt *File > Create <nnn>_image_v?.bin*. Gleichzeitig erzeugt der GUI-Designer eine `h`-, eine `csv`- und eine `log`-Datei (siehe auch Details: Alle erzeugte Dateien des Designs).

2. Die neue `bin`-Datei wird in den Flash-Speicher des Display-Moduls übertragen. Dies erfolgt über den Menüpunkt *LCM > Upload <nnn>_image_v?.bin* (bzw. *File > Upload <nnn>_image_v?.bin*) oder das nach oben zeigenden Dreiecksymbol. Man kann auch die LCMToolBox benutzen, um die `bin`-Datei in den Flash des Display-Moduls hoch zu laden.

3. Zum Testen kann das Design mit dem GUI-Designer (Menüpunkt *LCM > Start Design in LCM* oder das nach rechts zeigenden Dreiecksymbol) gestartet werden. Bei aktiviertem **Autostart-Flag** (im **Startup-Objekt**) startet das Design im Display-Modul bereits nach dem Einschalten. Eine weiter typische Variante zum Starten des Designs erfolgt über den Controller-Befehl `LCM_INIT_DESIGN` bzw. `LCM_START_DESIGN2` (siehe hierzu auch Details: Objekt-IDs und Benutzer-IDs).

4. Der (Pseudo-)Controller übernimmt nun die Kontrolle des Designs mit der Abfrage der **Events**. Mehr dazu unter Zusammenarbeit zwischen Design und Controller. Im GUI-Designer kann der Testablauf mit *LCM > Cancel Command* (bzw. *File > Cancel Command*) oder das Stop-Quadrat abgebrochen werden.

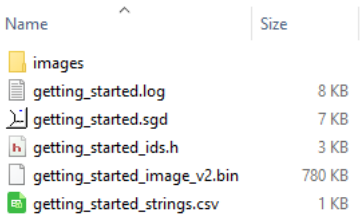


Abbildung 28: Alle erzeugte Dateien des Designs

3.3.1 Details: Alle erzeugte Dateien des Designs

Der GUI-Designer speichert das **Design** als `sgd`-Datei ab, hierzu wird intern das JSON-Format verwendet. Eine `sgd`-Datei kann im Prinzip mit einem Texteditor geöffnet und verändert werden, obwohl dieses Vorgehen nicht empfohlen wird.

Damit ein Design vom **GUI-Interpreter** verstanden und dargestellt werden kann muss dies in ein binäres Format umgewandelt werden. Der GUI-Designer führt diesen Schritt ebenfalls durch: Aus der `sgd`-Datei werden folgende Dateien generiert (siehe auch Abbildung 28):

- Eine Binärdatei (`bin`-Datei), die das aufbereitete Design, alle verwendeten externen **Ressourcen** (Bilder und Fonts) sowie **Text-Strings** und evtl. Zusatzinfos zum Startup (**Startup-Objekt**) enthält. Die Binärdatei wird einmal in den Flash-Speicher des LCM-Moduls übertragen und vom GUI-Interpreter abgearbeitet. Es gibt hier zwei Versionen: v1 wird nur dann benötigt, wenn der Firmware-Version älter ist. Die Version v2 braucht die Firmware-Version 7 oder höher (siehe auch den Abschnitt Tools-Menü).
- Eine C/C++-Header-Datei (h-Datei), die alle **Objekt-IDs** und **Benutzer-IDs** (`user IDs`) der im Design benutzen Objekte (**Widgets**, Bilder, Fonts, Text-Strings, **Sysvars** usw.) enthält (Abbildung 29). Die Objekt-IDs (oder evtl. Benutzer-IDs) werden bei der Erstellung eines eigenen Controllers benötigt, um auf die Objekte zugreifen zu können. Mehr dazu unter Details: Objekt-IDs und Benutzer-IDs.
- Eine `log`-Datei, die während des Umsetzen der `sgd`-Datei erzeugt wird. Die `log`-Datei darf über *File > View <nnn>.log* geprüft werden. Es erscheint auch ein Button links neben dem Blitzsymbol in der Symbolleiste, der den Status der Binärdatei (OK bzw. Fehler und Warnungen) angibt und ebenfalls mit einem Mausklick die `log`-Datei öffnet.
- Eine `csv`-Datei, die alle Text-Strings mit ihren eventuell vorhandenen Übersetzungen enthält. Diese `csv`-Datei kann durch weitere Übersetzungen ergänzt und wieder ins Design eingebunden werden. Mehr dazu unter Übersetzungen vorbereiten.

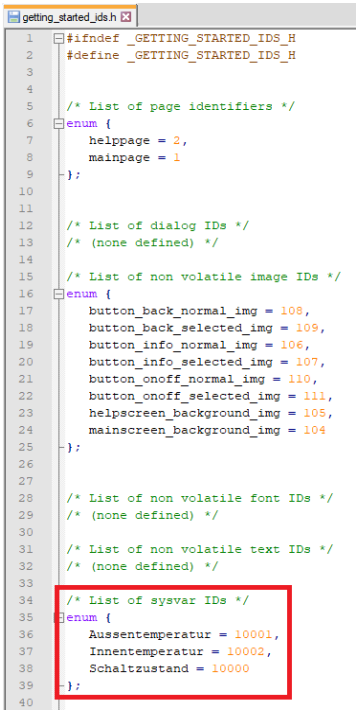


Abbildung 29: Teil einer Header-Datei mit Objekt-IDs

3.3.2 Details: Objekt-IDs und Benutzer-IDs

Im Flash-Speicher des Display-Moduls liegen die externe **Ressourcen** (Bilder und Fonts) und alle **Text-Strings** (mit ihre Übersetzungen). Jedes hat schon eine eigene **Objekt-ID**. Die Definition des Designs und das **Startup-Objekt** sind ebenfalls Objekte im Flash-Speicher. Auch die Übersetzungstabelle (siehe weiter unten) ist ein Objekt im Flash-Speicher. Man kann sich die Verteilung der Objekte im Flash-Speicher anschauen, indem man sich den Flash-Inhalt mit der LCMToolBox herunter lädt und dort im Objekteditor öffnet (siehe dort).

Beim Initialisieren und Starten des Designs werden die **Widgets** und **Sysvars** im RAM-Speicher des Moduls angelegt und bekommen auch ihre eindeutige Objekt-ID zugewiesen.

Wichtig! Den Controller stets mit der neuesten Header-Datei aktualisieren!

Wichtig! Änderungen im Design können zu Änderungen der Objekt-IDs führen. Deshalb ist es wichtig nach jeder Konvertierung des Design ins Binärformat, den Controller neu zu erzeugen und hierbei die ebenfalls neue Header-Datei zu benutzen. Hierdurch wird der Controller die eventuell geänderten Objekt-IDs benutzen und kann weiterhin die verschiedene Elemente wie Sysvars und Widgets richtig ansprechen.

Benutzer-IDs haben den Vorteil, dass sie immer fest bleiben. Der Controller kann bei Design-Änderungen oft weiter arbeiten, ohne neu erzeugt zu werden.

Manchmal ist es aber aus technischen Gründen nicht immer möglich oder wünschenswert den Controller bei jeder Design-Änderung neu zu erzeugen. Deswegen gibt es auch kunden-spezifische **Benutzer-IDs** als feste Stellvertreter für kritische Objekt-IDs. Eine Benutzer-ID (`user ID`) wird als optionale ganzzahlige Eigenschaft des entsprechenden Objekts (z.B. des Widgets, der Systemvariable, der Ressource, usw.) festgesetzt und muss – über alle Benutzer-IDs des Designs – einzigartig bleiben. Ansonsten ist der Wert dem Entwickler frei überlassen. Wo erlaubt, liegt die Eigenschaft `user ID` immer direkt unter der Eigenschaft `unique name`. Beim Erzeugen der Binärdatei wird eine Übersetzungstabelle ebenfalls erzeugt, die alle definierten Benutzer-IDs mit ihren (evtl. geänderten) Objekt-IDs verknüpfen. Vorteil: Der Controller kann meist mit den Benutzer-IDs einfach weiter arbeiten, ohne neu erzeugt zu werden, auch wenn die Objekt-IDs dazu sich geändert haben.

In der Header-Datei bekommt eine Benutzer-ID den gleichen Namen wie die dazu verknüpfte Objekt-ID aber mit dem Zusatz `_uid`.

Wichtig! Die neuen Benutzer-ID kompatiblen Kommandos (z.B. `LCM_GOTO_PAGE2` anstatt `LCM_GOTO_PAGE`) sind zu bevorzugen, stehen aber erst ab der Firmware Version 7.2 zur Verfügung!

Die Benutzer-IDs haben einen Offset von einer Milliarde, um nicht mit den Objekt-IDs zu kollidieren. (Dies passiert automatisch beim Generieren der bin- bzw. h-Datei.) Dafür wird einen `uint32` Parameter (anstatt bisher `uint16`) benötigt. Aus diesem Grund gibt es bei manchen Kommandos neue Varianten mit einem angehängten **2** (z.B. `LCM_GOTO_PAGE2`), die nicht nur Objekt-IDs sondern auch Benutzer-IDs bearbeiten dürfen. Diese neuen Varianten sind zu bevorzugen, es sei denn, die Firmware Version ist zu alt (<7.2) und kennt die neuen Kommandos nicht. In diesem Handbuch werden nur die neuen Kommandos erwähnt, aber die alten Kommandos sind nach wie vor möglich, wenn keine Benutzer-IDs verwendet werden.

3.4 Zusammenarbeit zwischen Design und Controller

In der Regel wird der GUI-Designer nicht alle Möglichkeiten des GUI-Interpreters abdecken. Man muss also auch verschiedene Problemstellungen durch Befehle abbilden, die vom **Controller** zum **GUI-Interpreter** gesendet werden.

Der GUI-Interpreter wird in einem eigenen Handbuch und Referenz erläutert.

Der Befehlssatz und die allgemeine Kommunikation zwischen dem Controller und dem GUI-Interpreter wird im Handbuch und Referenz des GUI-Interpreters ausführlich erläutert. Aber in der Zusammenarbeit mit einem Design entsteht folgendes Problem, wenn der Controller das Erscheinungsbild der **Seiten** (**Pages**) anpassen möchte:

Der Controller muss wissen, wann eine bestimmte Page aufgerufen wird, um „seine“ Elemente zu zeichnen. Dies wird durch **Events** gelöst, die beim Seitenwechsel generiert werden. Für den Controller ist es dann einfach zusätzliche **Linien**, **Texte** usw. zu zeichnen.

Schwieriger wird es bei **Buttons** oder ähnlichen Touch-empfindlichen Widgets, da hier auch beim Verlassen der Page Aufräumarbeiten zu erledigen sind (z.B. das Deaktivieren des Controller-eigenen Buttons).

Der GUI-Interpreter wurde aber um einige Befehle und Events erweitert.

Die neue oder veränderte Events sowie die neue Seitenbefehle, um auf einem Seitenwechsel ausreichend zu reagieren werden in den nächsten Abschnitten beschrieben.

3.4.1 Details: Events des GUI-Interpreters

Der **GUI-Interpreter** besitzt eine interne **Event**-Queue, deren Einträge vom Controller regelmäßig abgefragt werden müssen, um entsprechend reagieren zu können. Je nach Anwendung könnte das Intervall zwischen 0,1s bis einige Sekunden betragen.

Während der Darstellung eines Designs erzeugt der GUI-Interpreter eine ganze Reihe von Events, die hauptsächlich durch die Interaktion mit dem Benutzer entstehen. Wichtige Events sind:

Neue Events für Sysvars und Seitenwechsel.

- `SYSVAR_CHANGE_EVENT` wird erzeugt, wenn sich eine **Systemvariable** (**Sysvar**) ändert. Ein mit einer Sysvar verknüpften Widget erzeugt nur noch dieses Event. D.h. andere widget-spezifische Events wie `BUTTON_PRESSED_EVENT` werden überflüssig und daher unterdrückt.
- `PAGE_CHANGE_EVENT` wird beim Wechsel von einer Seite auf die nächste erzeugt.
- `EXIT_DESIGN_EVENT` entsteht, wenn das laufende Design beendet wird.

Alte Events die teils durch `SYSVAR_CHANGE_EVENT` abgelöst werden.

- `BUTTON_PRESSED_EVENT` wird erzeugt, wenn man einen *Button*-Widget drückt, der mit **keiner** Sysvar verknüpft ist.
- `SLIDER_CHANGE_EVENT` wird erzeugt, wenn man einen *Slider*-Widget verschiebt, der mit **keiner** Sysvar verknüpft ist.

Die Events enthalten noch weitere Informationen, um gezielt

reagieren zu können (z.B. die **Objekt-ID** der Sysvar, die sich gerade geändert hat).

3.4.2 Details: Neue Seiten-Befehle des GUI-Interpreters

Bei Designerweiterungen (d.h. vom Controller erzeugten grafischen Elementen) muss auf Seitenwechsel-Events reagiert werden!

Wenn ein Design durch eigene Controller-Befehle modifiziert werden soll müssen verschiedene Szenarien berücksichtigt werden. Der Controller muss beim Starten des Designs oder beim Seitenwechsel an der richtigen Stelle und in der richtigen Reihenfolge seine Modifikationen einfügen. Hierzu können die oben beschriebenen **Events** und entsprechende Interpreter-Befehle benutzt werden:

`LCM_GOTO_PAGE2`

- Wenn auf das automatische bzw. Button-gesteuerte Umschalten der Seiten verzichtet wird und der Controller diese Aufgabe übernimmt, dann wird ein Interpreter-Befehl benötigt, mit dem auf die jeweilige Seite umgeschaltet werden kann (`LCM_GOTO_PAGE2`). Der Controller weiß dann immer, welche Seite gerade angezeigt wird und kann seine Zeichen- und Darstellungsoperationen entsprechend anpassen.

`PAGE_CHANGE_EVENT`

- Wenn auf den Seiten nur einfache Zeichenoperationen (**Linien**, **Bilder**, **Kreise** usw.) vom Controller ausgeführt werden müssen, dann wird ein neues Event beim Seitenwechsel generiert werden (`PAGE_CHANGE_EVENT`). Diesem Event wird die aktuelle Page-Nummer mitgegeben. Hierdurch weiß der Controller welche Seite gerade angezeigt wird und kann seine Zeichenbefehle danach durchführen.

`LCM_SET_ENTER_PAGE_MACRO2`
`LCM_SET_LEAVE_PAGE_MACRO2`

- Will der Controller auf den Seiten eigene Touchable-Widgets zur Darstellung bringen, so müssen diese beim Eintritt auf die Seite aktiviert und beim verlassen der Seite deaktiviert werden. Jeder Seite wird hierzu ein Makro jeweils zum Betreten und Verlassen der Seite zugeordnet. Hierzu gibt es die Interpreter-Befehle `LCM_SET_ENTER_PAGE_MACRO2` und `LCM_SET_LEAVE_PAGE_MACRO2` (siehe das GUI-Interpreter Handbuch zur Erstellung von Makros).

Wichtig! Touchable-Widgets nur in den Makros (de-)aktivieren!

Wichtig! Widgets dürfen nur in den `ENTER/LEAVE` Makros aktiviert bzw. deaktiviert werden, nicht „per Hand“ in der Anwendung selbst! Vgl. die Sichtbarkeit als Alternative.

`LCM_INIT_DESIGN`

- Ferner benötigt man noch einen Befehl `LCM_INIT_DESIGN`, der ein Design initialisiert, aber nicht startet.

Wichtig! Reihenfolge beachten!

Wichtig! Der Controller muss in der folgenden Reihenfolge vorgehen um seine Design-Erweiterungen zu erzeugen:

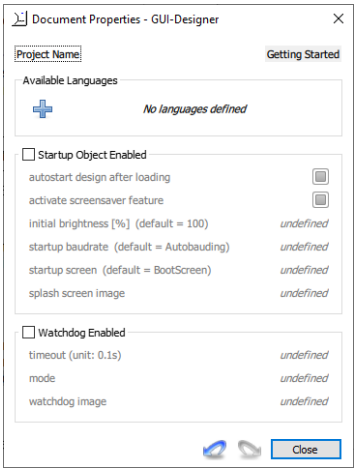


Abbildung 30: Dokument Eigenschaften mit gesetztem Projektnamen

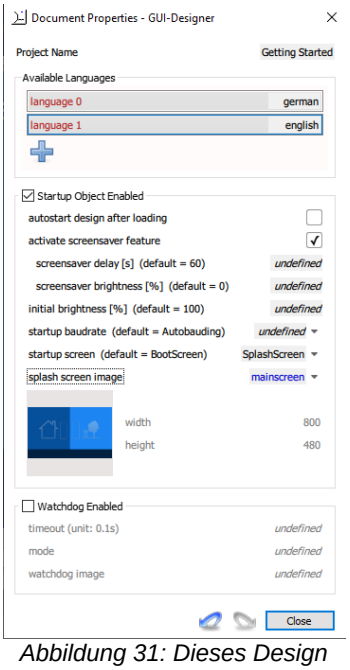


Abbildung 31: Dieses Design unterstützt 2 Sprachen, ein Bildschirmschoner und einen Splash-Screen

1. Zunächst das Design initialisieren ohne es zu starten (`LCM_INIT_DESIGN`).
2. Neue Touchable-Widgets (z.B. **Buttons** oder **Schieberegler**) definieren.
3. Mit `LCM_SET_ENTER_PAGE_MACRO2` und `LCM_SET_LEAVE_PAGE_MACRO2` für jede Seite entsprechende Makros definieren mit denen diese Touchable-Widgets aktiviert und deaktiviert werden. Die Makros können natürlich auch die einfachen Widgets wie **Texte** und Linien enthalten.
4. Mit `LCM_GOTO_PAGE2` die gewünschte Startseite anspringen.
5. In der Event-Loop des Controllers auf das `PAGE_CHANGED_EVENT` achten, um eventuell eigene Zeichenbefehle ausführen zu können (d.h. wenn sie nicht bereits im Schritt 3 gemacht worden sind).

Im einfachsten Fall – d.h. wenn keine Controller-Erweiterungen nötig sind – darf das Design auch mit `LCM_START_DESIGN2` gestartet werden. Das Kommando kombiniert einfach `LCM_INIT_DESIGN` mit `LCM_GOTO_PAGE2`.

3.5 *File > Document Properties*

Jedes Design hat einige globale Eigenschaften, die unter dem Menüpunkt *File > Document Properties* eingestellt werden können.

3.5.1 Projektname

Der Projektname kann frei gewählt werden. Er erscheint in der Projektdatei und hat für den GUI-Designer oder das Display-Modul keine weitere Bedeutung (siehe Abbildung 30).

3.5.2 Sprachen

Unter *Available Languages* wird eine Liste der verwendeten Sprachen definiert (Abbildung 31). Wie bei den meisten Listen, kann man mit Drag & Drop die Liste umsortieren, mit dem Plussymbol ( ) einen neuen Eintrag anlegen, oder über das **Kontextmenü** eines Eintrags den Eintrag duplizieren bzw. löschen. Mehr dazu unter Mehrsprachigkeit.

3.5.3 Startup-Objekt

Das Display-Modul hat beim Systemstart ein Standardverhalten, das aber teilweise konfigurierbar ist. Wenn das Standardverhalten nicht gewünscht wird, muss das **Startup-Objekt** zunächst aktiviert werden (*Startup Object Enabled* abhaken), um die andere Eigenschaften editieren zu können.

Normalerweise startet das Display-Modul mit dem **Boot-Screen**, d.h. mit einem weißen Hintergrund und den Firmen-

Informationen von Simplify Technologies (Eigenschaft `startup screen`). Es ist möglich, diese Informationen wegzulassen (`EmptyScreen`) oder durch einen eigenen **Splash-Screen** zu ersetzen. Das Bild muss vorab als **Image** in der *Images*-Liste angelegt werden, um dann hier der Eigenschaft `splash screen image` zugewiesen zu werden.

Man kann auch festlegen, dass das Display-Modul sofort nach dem Einschalten das Design startet (`autostart design after loading` = **Autostart-Flag**) und die Baudrate schon auf einen festen Wert eingestellt ist (`startup baudrate`).

Wichtig! Es gibt besondere Bedingungen bei der Verwendung des **Autostart-Flags**!

Achtung! Ein Display-Modul, welches mit einem Startup-Objekt mit aktiviertem Autostart-Flag ausgestattet wurde kann nicht mehr ohne weiteres durch Flashen mit einem neuen Design versorgt werden. Zunächst muss der Flash-Speicher gelöscht werden. Der Controller kann dazu den Befehl `LCM_INIT_FLASH` aufrufen. Zum Löschen des Flash-Speichers mit dem GUI-Designer muss zunächst im Menü *LCM > LCM Communication* das Autobauding deaktiviert und die Transferbaudrate auf die Baudrate des Startup-Objekts gesetzt werden. Das Löschen erfolgt dann mit dem Menüpunkt *LCM > Reset to Factory Settings*.

Display-Helligkeit beim Startup bestimmen

Darüber hinaus kann die Anfangshelligkeit der Hintergrundbeleuchtung festgelegt werden (`initial brightness`). Es ist übrigens auch möglich, die Helligkeit der Hintergrundbeleuchtung über eine **Systemvariable** zu steuern. Mehr dazu im Abschnitt Display-Helligkeit steuern.

Bildschirmschoner einsetzen

Ein **Bildschirmschoner** (**Screensaver**) kann hier ebenfalls aktiviert werden (`activate screensaver feature`). Der Bildschirmschoner reduziert die Hintergrundbeleuchtung des LCDs auf die angegebene Prozentzahl (`screensaver brightness`, Standardwert: `0%`), wenn für die vorgegebene Zeit (`screensaver delay`, Standardwert: `60s`) der Touch nicht gedrückt wurde. Die Hintergrundbeleuchtung wird wieder „normal“, wenn der Touchscreen berührt wird. Mehr hierzu im Abschnitt Details: Bildschirmschoner (Screensaver).

Beim Umwandeln des Designs in das binäre Format zum Upload auf das Display-Modul, wird das Startup-Objekt hinzugefügt.

3.5.4 Watchdog

Ein **Watchdog** reagiert, wenn die Kommunikation zwischen Controller und Display-Modul abbricht.

Es gibt eine **Watchdog**-Funktion, die die Kommunikation zwischen Display-Modul und Controller überwacht. Der Watchdog muss zunächst aktiviert werden (*Watchdog Enabled* abhaken), um die andere Eigenschaften editieren zu dürfen.

`timeout` wird in Zehntel-sekunden angegeben!

Dort kann die Zeit bis zum Timeout des Watchdogs in Einheiten von 0,1s eingestellt werden (`timeout`). Innerhalb dieser Zeit muss eine Kommunikation zwischen Controller und LCM-Modul stattfinden. Andernfalls wird die Abarbeitung des Designs unterbrochen und das Modul reagiert auf zwei verschiedenen Arten (`mode`):

`mode = Reset`

- `Reset`-Modus: Nach dem Timeout, wird ein Reset durchgeführt und anstatt der normalen Einschaltmeldung ein vorher definiertes Bild angezeigt. Dieses könnte z.B. eine Warnung o.ä. sein. Der Controller muss danach das Display-Modul mit **Autobauding**, Starten des Designs usw. neu initialisieren.

`mode = Redraw`

- `Redraw`-Modus: Hier wird angenommen, dass die Kommunikation nur kurzzeitig unterbrochen war und nach einer gewissen Zeit die Befehle das Display-Modul wieder erreichen. Nach dem Timeout wird der bisherige Bildschirm-inhalt gespeichert und anstatt dessen ein vorher definiertes Timeout-Bild gezeigt. Wenn die Kommunikation wieder einsetzt, wird zum ursprünglichen Bildinhalt zurück-geschaltet und weitere Befehle des Controllers normal ausgeführt.

Das entsprechende Bild (`watchdog image`), je nach Modus, muss bereits als **Image** in der *Images*-Liste vorliegen und hier ausgewählt werden.

3.5.5 Details: Bildschirmschoner (Screensaver)

Der Screensaver-Modus dient dazu die Display-Hintergrund-beleuchtung abzuschalten oder herunterzuregeln (erst ab **GUI-Interpreter** Version 5.5) wenn das Display-Modul längere Zeit nicht betätigt wurde. Hierdurch wird die Lebensdauer des Displays verlängert. In der Regel benutzt man das **Startup-Objekt**, um den **Screensaver** zu konfigurieren, es gibt jedoch auch folgende Interpreter-Befehle um hier direkt einzugreifen:

Der **Screensaver** kann auch vom Controller direkt konfiguriert werden.

- `LCM_SET_SCREENSAVER`: Setzt die Wartezeit bis zum Abschalten der Hintergrundbeleuchtung, `0` bedeutet Deaktivierung der Screensaver-Funktionalität.
- `LCM_RESET_SCREENSAVER`: Setzt den Zähler für die Screensaver-Wartezeit zurück. Hierdurch kann auch bei Geräten ohne Touch der Screensaver deaktiviert werden.

4. Design Aufteilung: Seiten (Pages)

Eine **Seite** (**Page**) bestimmt, was der Kunde auf dem Display-Modul zu einem bestimmten Zeitpunkt von der Anwendung sieht.

Eine Anwendung enthält normalerweise mehr Informationen oder Optionen, als auf dem Display-Modul gleichzeitig dargestellt werden können. Daher besteht das Design aus mehreren **Seiten** (**Pages**), einschließlich spezieller Seiten (**Dialoge**) für die Tastatureingabe. Der GUI-Designer enthält auch einen automatischen Mechanismus zum Wechseln zwischen den Seiten, so dass der **Controller** sich nicht damit befassen muss.

Mit Hilfe des **GUI-Interpreters** können Umschaltvorgänge zwischen Seiten erkannt und auch ausgelöst werden. Einzelheiten finden Sie unter Zusammenarbeit zwischen Design und Controller im vorherigen Kapitel.

Im Kapitel wird gezeigt, wie Seiten erstellt, angezeigt und bearbeitet werden.

Dieses Kapitel konzentriert sich stattdessen darauf, wie Seiten mit dem GUI-Designer erstellt, angezeigt und bearbeitet werden. Im Wesentlichen ist die Seite das grundlegende Anzeigeelement, das vom GUI-Designer verwendet wird. Zunächst muss die Seitengröße bestimmt werden. Dann gibt es eine Übersicht aller Seiten (Pages-Tab), um die Seiten zu verwalten. Der Inhalt einer Seite wird in der Design Area bearbeitet. Die Endergebnisse können in den Tabs Pixel-Exact View und Size-Exact View angezeigt und teilweise getestet werden.

Schließlich gibt es einen speziellen Seitentyp, der als **Dialog** bezeichnet wird. Dadurch kann der Benutzer die Werte von **Systemvariablen** (**Sysvars**) eingeben, um Informationen an den Controller zu übergeben. Da dies eng mit **System-variablen** zusammenhängt, wird im nächsten Kapitel, im Abschnitt Eingabe-Seite (Dialog), darauf eingegangen.

Die Seitengröße muss mit dem Display-Modul übereinstimmen!

Wichtig: Die Größe jeder Seite muss der Auflösung des Display-Moduls entsprechen, auf dem das Design installiert wird. Dies sollte vor dem Hinzufügen von Widgets zum Design festgelegt werden. Dieser Vorgang wird daher als Nächstes beschrieben.

4.1 Seitengröße bestimmen

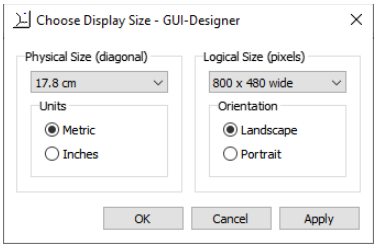


Abbildung 32: Dialogfenster, um die Seitengröße zu bestimmen

Eine Seite hat immer die gleiche Größe wie die Auflösung des Display-Moduls. Deswegen soll als Erstes die Größe Ihres Display-Moduls dem GUI-Designer bekannt gegeben. Dies geschieht über die Menüauswahl *LCM > Display Size*. Daraufhin wird das Dialogfenster *Choose Display Size* (Abbildung 32) geöffnet, in dem sowohl die Diagonale (*Physical Size (diagonal)*) als auch die Auflösung (*Logical Size (pixels)*) des Displays festgelegt werden soll. Die Aufklappmenüs ermöglichen alle aktuell verfügbaren LCM-Display-Modul-Größen.

Die Diagonale (linke Spalte) wird im GUI-Designer verwendet, um die Diagonale der *Size-Exact View* zu bestimmen, hat jedoch ansonsten keine besondere Auswirkung. Der Einfachheit halber dürfen die Einträge dieses Aufklappmenüs wahlweise in cm (*Metric*) oder *Inches* dargestellt werden.

Die Auflösung in Pixel (rechte Spalte) ist wiederum von entscheidender Bedeutung, und muss unbedingt mit der Auflösung des Display-Moduls übereinstimmen. Zunächst ist die Ausrichtung (*Orientation*) auf Querformat (*Landscape*) oder Hochformat (*Portrait*) einzustellen. Diese Entscheidung vertauscht nach Bedarf die Breite und Höhe der Einträge des Aufklappmenüs, um die endgültige Auflösung zu bestimmen.

Die korrekte Auflösung ist entscheidend für das ordnungsgemäße Funktionieren des Designs mit einem Display-Modul!

Die entgültige Auflösung bestimmt die Auflösung der *Design Area* und der *Pixel-Exact View* sowie die relativen Abmessungen der *Size-Exact View* und der **Miniaturbilder** auf dem *Pages*-Tab. Sie ist auch für die korrekte Übertragung und Darstellung des Designs auf dem Display-Modul wesentlich.

Wichtig! Die Pixel-Auflösung des Display-Moduls sollte vor dem Hinzufügen von Widgets zum Design korrekt eingestellt werden. Insbesondere wenn Sie die Auflösung ändern, nachdem eine Seite mit Widgets gefüllt wurde, kann dies zum Absturz des GUI-Designers führen.

4.2 Seiten verwalten (*Pages*-Tab)

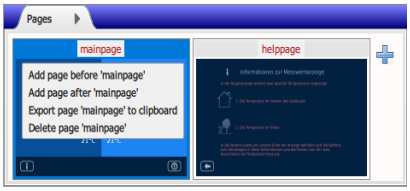


Abbildung 33: Pages-Tab mit Kontextmenü (rechte Maustaste)

Der *Pages*-Tab befindet sich standardmäßig im oberen mittleren Fenster des GUI-Designers. Darin werden alle bisher definierte **Seiten** (**Pages**) des Designs als eine waagerechte Liste von **Miniaturbilder** dargestellt. Die aktuelle Seite wird in blau hervorgehoben und ist dann in der *Design Area*, *Pixel-Exact View* und *Size-Exact View* detailliert sichtbar.

Neue Seite mit einem klick anlegen und die Name der Seite editieren.

Der obere Rand eines Miniaturbildes ist etwas dicker, fungiert als Titelleiste und enthält den **eindeutigen Namen** der Seite. Der Name kann mit einem einfachen Klick darauf editiert werden. Ein **Kontextmenü** (rechte Maustaste) erlaubt, dass eine neue Seite entweder vor oder nach der aktuellen Seite angelegt bzw. die aktuelle Seite gelöscht wird (Abbildung 33). Es ist auch möglich, eine neue Seite am Ende der Liste mit dem Plussymbol ( ) rechts im *Pages*-Tab anzulegen.

Zusätzlich (wiederum im Kontextmenü) darf die Seite samt Inhalt (den Widgets und deren Bezugsobjekte, z.B. Stile, Sysvars, Text-Strings, usw.) in die Zwischenablage „exportiert“ werden (*Export page to clipboard*) als sogenannte „Subdesign“. Der Inhalt der Zwischenablage kann in einem beliebigen Design mit Strg+v (Steuerungstaste+v) wieder „importiert“ (eingefügt) werden. Die Zwischenablage darf auch per Hand in eine leere

Textdatei kopiert werden (empfohlene Erweiterung: .sgdi) als eine Art Schnipsel für den späteren Gebrauch.

Seiten können leicht umsortiert werden aber die erste Seite ist wichtig!

Die Liste der Seiten kann jederzeit mit Ziehen und Loslassen (Drag & Drop) eines Miniaturbilds umsortiert werden. Die Reihenfolge der Seiten ist an sich unbedeutend bis auf der ersten Seite. Die erste Seite ist die Seite, die beim Starten des Designs im Display-Modul als Erste dargestellt wird.

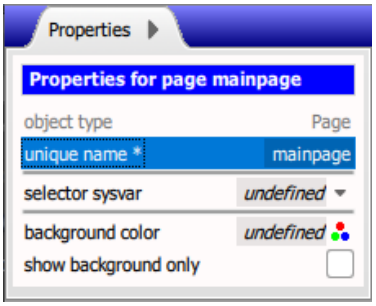


Abbildung 34: Eigenschaften der aktuellen Seite

Jede Seite hat mehr Eigenschaften, als im Miniaturbild zu sehen sind. Wenn keine **Widgets** selektiert sind, werden im Tab *Properties* alle **Page-Eigenschaften** der aktuellen Seite angezeigt (Abbildung 34). Es wird ebenfalls auf die Page-Eigenschaften gewechselt, wenn auf dem leeren Teil der Titelleiste eines Miniaturbilds geklickt wird.

Im *Properties*-Tab erscheint nochmal die Eigenschaft `unique name`, die auch in der Titelleiste des Miniaturbilds zu sehen ist. Es gibt ferner die Eigenschaft `selector sysvar`, die für spezielle **Eingabe-Seiten** (**Dialoge**) reserviert ist. Schließlich gibt es Eigenschaften für die Hintergrundgestaltung, wie unten erklärt.

4.2.1 Details: Hintergrund-Modus und Hintergrundfarbe

Hintergrund-Modus mit `show background only`

Sind die Eigenschaften einer Seite zu sehen kann hier in den **Hintergrund-Modus** umgeschaltet werden. Hierzu wird die Pseudo-Eigenschaft `show background only` im *Properties*-Tab aktiviert. Sie gilt als Pseudo-Eigenschaft, weil es sich nicht um eine normale gespeicherte Page-Eigenschaft sondern um einen temporären Zustand des GUI-Designers handelt, die global allen Seiten betrifft.

Was ändert sich im Hintergrund-Modus?

Im Hintergrund-Modus zeigen alle Ansichten nur die Hintergründe der verschiedenen Seiten. Im *Pages*-Tab sieht man die verschiedenen Miniaturbilder der Hintergründe. Der *Widgets*-Tab zeigt nur die **statischen Bilder** (*Image*-Widgets) aus denen der Hintergrund der jeweiligen Seite besteht. Im *New*-Tab dürfen nur neue statische Bilder (**Widget-Typ** *Image*) zu einer Seite hinzugefügt werden. Hierdurch ist es einfacher den Hintergrund einer Seite zu editieren ohne durch die anderen Widgets abgelenkt oder verwirrt zu werden.

Hintergrundbilder immer im Hintergrund-Modus anlegen!

Wichtig! Es wird empfohlen, die Hintergrundbilder immer im Hintergrund-Modus anzulegen. Dies verhindert, dass ein Hintergrundbild andere Widgets im Vordergrund aus versehen überdeckt!

Wie kann ich alle Widgets wieder sehen?

Wenn alle **Hintergrundbilder** angelegt worden sind, soll der Hintergrund-Modus (`show background only`) wieder deaktiviert werden. Jetzt zeigen die Ansichten und die Miniaturbilder wieder alles, d.h. die Hintergrundbilder sowie alle

anderen bereits angelegten Widgets. Der *New*-Tab erlaubt wieder alle Widget-Typen und der *Widgets*-Tab listet alle Widgets der aktuellen Seite auf, die nicht dem Hintergrund gehören.

Brauche ich eine Hintergrundfarbe?

In manchen Fällen wird ein Hintergrundbild nicht gewünscht oder das Bild deckt nicht den gesamten Hintergrund ab. In diesem Fall darf eine Hintergrundfarbe (Eigenschaft `background color`) für diese Seite bestimmt werden (Standardwert: `white`). Die Hintergrundfarbe ist eine echte Page-Eigenschaft, gilt nur für die aktuelle Seite und wird im Design mitgespeichert.

4.3 *Pixel-Exact View* und *Size-Exact View*

Der GUI-Designer enthält zwei Tabs mit der Bezeichnung *Pixel-Exact View* und *Size-Exact View*. Diese sind normalerweise im rechten unteren Fenster des GUI-Designers sichtbar und zeigen den Inhalt der aktuell ausgewählten Seite an. Es gibt zwei verschiedene Ansichten, da der Computermonitor im Allgemeinen eine andere Pixelgröße als das Display-Modul hat.

Vorteile der *Pixel-Exact View*

Die *Pixel-Exact View* ignoriert diesen Größenunterschied und ordnet einen Pixel auf dem Display-Modul einem Pixel auf dem Computermonitor zu. Mit anderen Worten, was Sie sehen, ist das, was Sie erhalten (WYSIWYG), aber die Ansicht kann aufgrund des Unterschieds in der Pixelgröße größer oder kleiner als das Display-Modul sein.

Vorteile der *Size-Exact View*

Die *Size-Exact View* skaliert die Seite hingegen so, dass sie genau dieselbe Diagonale wie das Display-Modul hat. Einige Pixel werden dabei möglicherweise verzerrt, und die Seite sieht vielleicht nicht so scharf aus, aber man kann erkennen, ob der Text lesbar ist und die Berührungselemente groß genug sind, wenn sie später auf das Display-Modul übertragen werden.

Mit diesen Ansichten kann die Touch getestet werden.

Beide Ansichten sind so konzipiert, dass sie weitgehend wie bei der Interaktion mit einem Display-Modul funktionieren, indem Sie das Widget mit der Maus (Klicken oder Ziehen) anstelle eines Fingers „berühren“. Trotz einiger Einschränkungen, können viele Eigenschaften des Designs vor dem Hochladen auf das Display-Modul getestet werden.

4.4 *Design Area*

Die *Design Area* befindet sich im unteren mittleren Fenster des GUI-Designers. Es ist das einzige Fenster, das keine Tabs enthält. Im nicht-gezoomten Zustand zeigt das Fenster genau den gleichen Inhalt wie die pixelgenaue Ansicht. Mit anderen Worten, es zeigt eine 1:1-Pixel-Darstellung der aktuell ausgewählten Seite des Designs, die je nach Pixelgröße Ihres Computermonitors größer oder kleiner als das Display-Modul

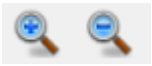
sein kann.

Die *Design Area* wird zum Bearbeiten des Designs verwendet.

In jeder anderen Hinsicht unterscheidet sich die *Design Area* jedoch völlig von der *Pixel-Exact View*. Kurz gesagt, in der *Design Area* wird das Design erstellt und geändert, während in der *Pixel-Exact View* (und in der *Size-Exact View*) das Design angezeigt und getestet wird.

Features der *Design Area*

Einige Funktionen der *Design Area*:



- Wenn Sie auf dem *Pages*-Tab auf eine beliebige Stelle der Miniaturansicht einer Seite klicken, zentriert sich der Designbereich an diesem Punkt auf dieser Seite (stoppt jedoch, wenn er eine Kante erreicht).
- Die *Design Area* kann entweder über das Menü (*Window* > *Zoom In* oder *Zoom Out*) oder über die Tastatur (Strg+ or Strg-) oder die Symbolleiste (Lupensymbole) vergrößert bzw. verkleinert werden. Dabei wird die Größe bei jedem Klick verdoppelt bzw. halbiert.
- Neue **Widgets** können zum Design hinzugefügt werden, indem **Widget-Typen** vom *New*-Tab in die *Design Area* gezogen werden.
- Wenn Sie auf ein Widget klicken, wird umgeschaltet, ob es ausgewählt ist oder nicht (vgl. das Klicken auf eine Schaltfläche in der *Pixel-Exact View* zum Testen des Widgets). Strg+Klick ermöglicht die **Mehrfachauswahl**.
- Durch Ziehen eines Widgets wird seine Position geändert (vgl. das Ziehen eines **Schiebereglers** in der *Pixel-Exact View* zum Testen des Reglers). Wenn das Widget Teil einer **Mehrfachauswahl** ist, werden alle ausgewählten Widgets um dieselbe Entfernung verschoben.
- Wenn ein einziges Widget ausgewählt ist, darf die Größe des Widgets durch Ziehen einer Kante oder Ecke geändert werden. Der Cursor ändert sich, um diese Möglichkeit anzudeuten. Bei **Linien** dürfen die Endpositionen ebenfalls in dieser Weise einzeln verschoben werden.

- Die Bearbeitungsbefehle – *Cut Widgets*, *Copy Widgets*, *Paste Widgets*, *Delete Widgets* und *Select All Widgets* - sind an dieses Fenster gebunden und werden zum Kopieren oder Löschen von ausgewählten Widgets verwendet.

Hinweis zum Kopieren: Wenn Widgets auf dieselbe Seite kopiert werden, werden sie direkt auf die Originale kopiert. Ziehen Sie die neuen (ausgewählten!) Widgets eine kurze Entfernung, um sowohl die Kopien als auch die Originale darunter zu sehen. Es ist auch möglich, Widgets in die Zwischenablage zu kopieren (oder auszuschneiden) und sie dann wieder auf einer anderen Seite einzufügen.

5. Systemvariablen (Sysvars)

Systemvariablen werden auch **Sysvars** genannt.

Systemvariablen (**Sysvars**) beinhalten Parameter oder Messwerte einer Anwendung. Bei einer Temperaturregelung bspw. wäre der Temperaturwert und die Stellgröße zwei Systemvariablen.

Ein **Widget** berücksichtigt bei seiner Darstellung stets die aktuellen Werten aller relevanten **Sysvars.**

Systemvariablen werden im GUI-Designer in den Sysvars-Tab angelegt. Diese Systemvariablen werden dann anderen **Widgets** über die Widget-Eigenschaft `sysvar` (manchmal genauer benannt, z.B. `visibility sysvar`) zugewiesen. Das Widget berücksichtigt bei seiner Darstellung stets den aktuellen Wert der Systemvariable.

Eine Sysvar kann über ein **berührbares Widget** oder eine **Eingabe-Seite** des Designs gesetzt werden.

Manchmal kann der Wert einer Systemvariable über ein **berührbares Widget** gesteuert werden, z.B. einen **Schiebe-regler** (*Slider*-Widget). Aber oft werden genaue numerische Werte oder Texte durch den Benutzer eingetippt. Hierzu kann eine spezielle Seite angelegt werden: die Eingabe-Seite (Dialog).

Der **Controller** bekommt solche Änderungen über ein `SYSVAR_CHANGE_EVENT` samt **Objekt-ID** mit.

Jede angelegte Systemvariable im Design bekommt eine eigene **Objekt-ID**. Dadurch ist sie vom **Controller** direkt ansprechbar. Im Idealfall beschränkt sich die Kommunikation zwischen Controller und Display-Modul auf die Übertragung von Systemvariablen. Siehe Neue Sysvar-Befehle des GUI-Interpreters. Normalerweise kommen aber noch **Events** hinzu, insbesondere `SYSVAR_CHANGE_EVENTs`. In speziellen Fällen werden darüber hinaus weitere Befehle an das Display-Modul gesendet, um besondere Anforderungen der Benutzer-oberfläche zu realisieren. Siehe Zusammenarbeit zwischen Design und Controller.

5.1 Neue Sysvar-Befehle des GUI-Interpreters

Der **Controller** hat Befehle, um **Sysvars** selbst lesen oder setzen zu können.

Es gibt fünf Typen von **Sysvars**: `Number`, `Color`, `Boolean`, `String`, und `Float`. Die ersten drei Typen werden intern durch `Int32`-Werte dargestellt, für die es Controller-Befehle zum Registrieren, Setzen/Lesen und Setzen/Lesen der Grenzen der Sysvars gibt.

Wenn der Entwickler die Sysvars in der *Sysvars*-Liste des GUI-Designers anlegt braucht er sich um die Registrierung nicht mehr zu kümmern. Der **GUI-Interpreter** erledigt dies beim Starten des Designs.

Alle benötigten Sysvars müssen mit dem GUI-Designer angelegt werden!

Wichtig! Es ist dann allerdings nicht mehr möglich weitere Sysvars mit dem Controller anzulegen (z.B. mit `AU_DATASERV_REGISTER_INT32` bei `Int32`-Sysvars). Alle Sysvars müssen in der *Sysvars*-Liste des GUI-Designers angelegt werden.

`Number`-, `Color`-, und `Boolean`-Sysvars werden als `Int32`-Werte behandelt.

Zum Setzen/Lesen lauten die Befehle für `Int32`-Werte:

- `AU_DATASERV_SETVAR_INT32`
- `AU_DATASERV_GETVAR_INT32`
- `AU_DATASERV_SETVAR_INT32_LIMITS`
- `AU_DATASERV_GETVAR_INT32_LIMITS`

`Uint32` ist ebenfalls möglich, aber im Wertebereich eingeschränkt.

Die `Int32`-Werte der Sysvar-Typen `Number`, `Color`, und `Boolean` dürfen im Bereich zwischen $(-2^{31} + 2)$ und $(2^{31} - 2)$ liegen (die Endwerte sind für den GUI-Designer aus technischen Gründen reserviert). Es gibt auch entsprechende Funktionen für `Uint32` zum Lesen und Schreiben der Sysvars, aber bei den oben durchgeführten Konventionen, ist deren Wertebereich von 0 bis ebenfalls $(2^{31} - 2)$ eingeschränkt.

`String`-Sysvars werden auf der kleinstmöglichen Länge gehalten, um effizienter zu bleiben.

Bei `String`-Sysvars wird je nach Länge (Eigenschaft `max characters`) eine der folgenden Typen vom Design gewählt: `String10`, `String20`, `String40`, `String80` (Standardwert). Längere Strings werden nicht unterstützt. Die Kommunikationsbefehle werden nach dem selben Muster wie bei den ganzzahligen Sysvar-Typen gebildet.

`Float`-Sysvars stellen Fließkommazahlen vom Typ `Float32` dar und werden lediglich in **Diagramm-Container** (*Diagram*-Widgets) verwendet. Die Kommunikationsbefehle werden nach dem selben Muster wie bei den ganzzahligen Sysvar-Typen gebildet.

Das `persistent`-Flag

Die numerische Sysvar-Typen können dauerhaft im EEPROM-Speicher des Display-Moduls abgelegt werden, dazu muss das `persistent`-Flag gesetzt werden. Bei `String`-Sysvars ist dies nicht möglich.

5.2 Reservierte Sysvar-Namen

Reservierte Namen für die Helligkeit und Temperatur des Displays sowie für die Sprachverwaltung.

Einige **Sysvar**-Namen werden durch das System benutzt und haben reservierte Namen:

- `SYSVAR_DISPLAY_BRIGHTNESS` (siehe Display-Helligkeit steuern)
- `SYSVAR_LCM_TEMPERATURE` enthält die Temperatur des Display-Moduls in Celsius (wird alle 2min automatisch aktualisiert).
- `SYSVAR_NUMBER_OF_LANGUAGES` (siehe Bemerkung)
- `SYSVAR_CURRENT_LANGUAGE` (siehe Bemerkung)

Bemerkung: Weitere Infos unter Details: Reservierte Sysvars für die Mehrsprachigkeit.

5.3 Display-Helligkeit steuern

Die Helligkeit des Displays durch eine Sysvar steuern.

Die Helligkeit des Display-Moduls kann durch das Anlegen einer reservierten `Number`-**Sysvar** mit dem Namen `SYSVAR_DISPLAY_BRIGHTNESS` gesteuert werden. Der Bereich der Sysvar muss von `0` bis `100` gehen und repräsentiert direkt die Display-Helligkeit in Prozent. Man kann diese Sysvar bspw. mit einem **Schieberegler** (*Slider*-Widget) verbinden und so direkt die Helligkeit kontrollieren.

5.4 Sysvars verwalten (*Sysvars*-Tab)

Der *Sysvars*-Tab befindet sich standardmäßig im oberen rechten Fenster des GUI-Designers. Darin werden alle bisher definierte **Systemvariablen** (**Sysvars**) des Designs als eine senkrechte Liste von umrahmten Einträgen dargestellt. Der aktuelle Eintrag wird blau umrahmt und aufgeklappt, damit alle Eigenschaften sichtbar sind. Alle andere Einträge werden minimal dargestellt (nur Typ und Name sind sichtbar).

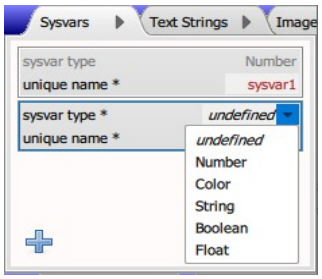


Abbildung 35: Die erlaubte Typen für Systemvariablen

Ein neuer Eintrag wird minimal dargestellt bis die Eigenschaft `sysvar type` bestimmt wird. Um den Typ zu bestimmen, wird auf dem Pfeil (▼) geklickt und im **Aufklappmenü** eine der fünf Typen ausgewählt: `Number`, `Color`, `String`, `Boolean` oder `Float` (Abbildung 35). Danach kann der Typ nicht mehr geändert werden und das Aufklappmenü verschwindet.

Wie bei den meisten Listen, kann man mit Drag & Drop die Liste umsortieren, mit dem Plussymbol ( ) einen neuen Eintrag anlegen, oder über das **Kontextmenü** eines Eintrags den Eintrag duplizieren bzw. löschen. Hier ist zu beachten, dass duplizierte Einträge auch den Typ erben! Wenn ein anderer Typ gewünscht wird muss stattdessen das Plussymbol verwendet werden, um eine zunächst „typlose“ Sysvar zu erzeugen.

In allen Fällen bekommen neue Einträge automatisch einen **eindeutigen Namen** (`unique name`). Verbindungen zu einer Sysvar erfolgen immer direkt durch diesen Namen bzw. für den **Controller** durch die daraus erzeugte **Objekt-ID**. Es wird empfohlen der Sysvar einen sinnvollen Namen zu geben.

Neben Typ und Namen, die mögliche Eigenschaften einer **Sysvar**.

In Abhängigkeit des Typs erscheinen weitere Eigenschaften. Der Simulationswert (`simulation value`) ist für den Testzweck im GUI-Designer da. Er wird im Design (`sgd`-Datei) zwar mitgespeichert, wird aber nicht auf dem Display-Modul übertragen. Der Anfangswert (`initial value`) wird auf dem Display-Modul verwendet bis der Bediener oder der Controller einen neuen Wert setzt und soll unbedingt angegeben werden (um Warnungen zu vermeiden). Eine `String`-Sysvar hat eine maximale Größe (`max characters`). Eine `Number`- bzw. `Float`-Sysvar hat einen kleinsten bzw. größten Wert (`minimum`

`value` bzw. `maximum value`). Die vier numerische Typen dürfen auch nach Bedarf dauerhaft im EEPROM-Speicher des Display-Moduls (Checkbox `persistent`) abgelegt werden.

Bei `Color`-Sysvars hat der Controller auch Zugang zu den **benutzerdefinierten Farben** im Design.

Bemerkung: Aus technischen Gründen ist es verboten, die Eigenschaften `simulation value` und `initial value` einer `String`-Sysvar auf einem **Text-String** der *Text-Strings*-Liste zu setzen. Bei `Color`-Sysvars ist aber ein Verweis auf eine **benutzerdefinierten Farbe** in der *Colors*-Liste durchaus zulässig. Konstanten für die benutzerdefinierten Farben stehen in der Header-Datei auch dem Controller zur Verfügung.

`Number`-Sysvars sind immer ganzzahlig!

Wichtig: `Number`-Sysvars sind immer ganzzahlig. Es findet in der *Sysvars*-Liste keine Konvertierungen (z.B. Millimeter nach Meter) statt. Solche Konvertierungen finden nach Bedarf immer dort statt, wo die Sysvar dargestellt wird (z.B. in einem *Number*-Widget).

5.5 Eingabe-Seite (Dialog)

Ein **Dialog** erlaubt die Eingabe von **Sysvar**-Werten durch den Benutzer.

Eingabe-Seiten (**Dialoge**) im Design dienen dazu den Wert von verschiedenen **Systemvariablen** (**Sysvars**) eines Typs einzugeben. Hierbei ändert sich das Aussehen der Seite je nach dem, welche Systemvariable eingegeben werden soll.

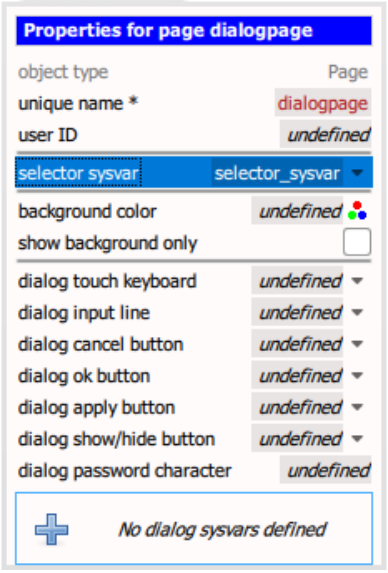


Abbildung 36: Das Setzen der Eigenschaft `selector sysvar` verwandelt eine Seite in einen Dialog mit weiteren Eigenschaften

Ein Dialog wird von einer normalen **Page** abgeleitet, in dem eine spezielle Systemvariable vom Typ `Number` als **Selektor-Sysvar** definiert und der **Page-Eigenschaft** `selector sysvar` zugewiesen wird. Die Seite wird dadurch in einen Dialog verwandelt (Abbildung 36).

Die Eigenschaften des Dialogs werden um eine zunächst leere Liste von sogenannten **Dialog-Sysvars** erweitert. Außerdem enthält ein Dialog noch weitere Eigenschaften, die Ok-, Cancel- und Apply-**Buttons**, eine **Tastatur** und eine **Eingabezeile** bestimmen können. Wenn der Dialog für Kennwörter verwendet wird, soll ein Symbol zum Verdecken des Kennworts (`dialog password character`) definiert werden. Dazu passt ein Button zum Zeigen/Verstecken des Kennworts (`dialog show/hide button`). Solche Widgets werden, wie bei jeder anderen Seite auch, zuerst auf der Seite platziert und konfiguriert, und dann den obengenannten Eigenschaften je nach Funktion zugewiesen. Alle Widgets sind theoretisch optional; es muss aber möglich sein, den Dialog abzubrechen, entweder durch einen Cancel-Button oder durch eine berührbare Tastatur mit einer Taste vom Typ `Esc`.

Abbildung 37: Beispielliste von Dialog-Sysvars

Die **Selektor-Sysvar** (Page-Eigenschaft `selector sysvar`) und die **Dialog-Sysvars** sind die wichtigsten Eigenschaften. Die Selektor-Sysvar entscheidet, welche Systemvariable editiert werden soll, indem er einfach die Nummer (=Index) eines

Eintrags in der *Dialog-Sysvar*-Liste angibt. Jeder Eintrag der Liste enthält eine Verbindung zu einer Systemvariable (Eigenschaft `sysvar`), die mit diesem Dialog editiert werden soll (Abbildung 37), sowie weitere Eigenschaften zur Darstellung der Systemvariable (siehe Details: Eingeschaften einer Dialog-Sysvar).

Der spezielle *Set-Button (Number)* springt den Dialog an.

Der Aufruf des Dialogs erfolgt über ein **Set-Button** vom Typ `Number` auf einer beliebigen Seite. Der Set-Button setzt die Selektor-Sysvar auf die Nummer der zu editierenden Systemvariable und springt die Eingabe-Seite an. Hierzu wird die Eigenschaft `destination` auf die Eingabe-Seite gesetzt, `sysvar` auf die Selektor-Sysvar und `sysvar value (Number)` auf den Index der zu editierenden Dialog-Sysvar.

Ein *EnumLabel*-Widget erleichtert variable Überschriften usw.

Man kann durch die Benutzung von **indizierten Textauswahlen** (*EnumLabel*-Widgets), die ebenfalls mit der Selektor-Sysvar verknüpft sind, passende Überschriften, Einheiten o.ä. auf der Eingabe-Seite darstellen. In dieser Weise können viele Systemvariable mit nur eine Eingabe-Seite unterschiedlich dargestellt und editiert werden.

5.5.1 Details: Dialog Beispiel

Ein einfaches Beispiel erklärt, wie der **Dialog** (die **Eingabe-Seite**) korrekt konfiguriert und aufgerufen wird.

1. Anlegen einer **Selektor-Sysvar**.

Es gibt drei **Sysvars** für `Breite`, `Länge`, `Höhe` eines Quaders. Angenommen man will mit dem Dialog die Abmessungen dieses Quaders eingeben, dann erzeugt man eine **Selektor-Sysvar**, deren Wertebereich von `0` bis `2` gehen kann (`0`: Sysvar der `Breite` soll editiert werden, `1`: Sysvar der `Länge` soll editiert werden, `2`: Sysvar der `Höhe` soll editiert werden).

2. Anlegen eines **Dialogs** mit *TouchKeyboard-*, *InputLine-* und *Button-*Widgets.

Nun erzeugt man eine neue **Page** und setzt deren Eigenschaft `selector sysvar` auf die eben definierte Selektor-Sysvar. Die Page wird dadurch in einen Dialog umgewandelt. Im Tab *Properties* erscheinen dann Eigenschaften für die Dialog-spezifischen **Widgets** (z.B. `dialog cancel button`). Die gewünschten Widgets müssen auf dem Dialog erzeugt werden. Wir ziehen eine **Tastatur** (*TouchKeyboard*), eine **Eingabezeile** (*InputLine*) und zwei **Standard-Buttons** (Cancel und OK) auf die Seite und weisen sie den entsprechenden Eigenschaften zu (Abbildung 38).

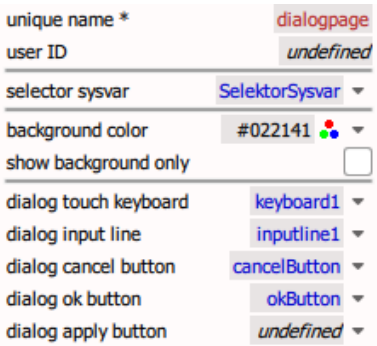


Abbildung 38: Dialog-Eigenschaften des Beispiels

Bemerkung: Bei den Buttons muss die Eigenschaft `destination` leer bleiben. Im LCD-Display springt der Dialog von sich aus immer auf die Seite des aufrufenden **Set-Buttons** zurück, wenn der OK- oder Cancel-Button betätigt wird.

3. Anlegen einer *Dialog-Sysvars*-Liste.

Ferner legt man eine *Dialog-Sysvars*-Liste an, deren Einträge aus den Sysvars für `Breite`, `Länge`, `Höhe` bestehen

(Abbildung 37). In diesem Beispiel erlauben wir nur ganzzahlige Eingaben. Die zusätzliche Eigenschaften der Einträge werden darum nicht benötigt.

4. Anlegen und Ausprobieren der drei **Set-Buttons**.

Auf irgendeiner anderen Page gibt es nun jeweils einen **Set-Button** (vom Typ `Number`) für `Breite`, `Länge` und `Höhe`, bei dem die Eigenschaft `sysvar value (Number)` auf `0`, `1` oder `2` gesetzt wird, und dessen `sysvar`-Eigenschaft auf die Selektor-Sysvar sowie `destination`-Eigenschaft auf den Dialog gesetzt wird (Abbildung 39).

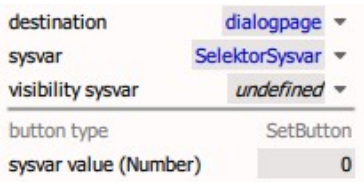


Abbildung 39: SetButton-Eigenschaften für die Breite

Drückt der Benutzer später z.B. auf den Set-Button für die `Länge`, dann wird die Selektor-Sysvar auf `1` gesetzt und der Dialog angesprungen. Dort wird dann die Länge editiert. Das Anspringen kann schon im *Pixel-Exact View* oder *Size-Exact View* ausprobiert werden. Im Selektor-Sysvar (*Sysvars*-Tab) wird sichtbar, ob `0`, `1` oder `2` als `simulation value` gesetzt wurde.

5. Anlegen eines *EnumLabel*-Widgets.

Damit es klar wird, was gerade editiert wird, erweitern wir unser Beispiel und legen auf der Eingabe-Seite auch noch eine **indizierte Textauswahl** (*EnumLabel*-Widget) mit drei Einträgen (*labels*) an. Die Einträge sollen folgende Texte enthalten (Abbildung 40):

- *label 0*: `text` = „`Breite`“,
- *label 1*: `text` = „`Länge`“,
- *label 2*: `text` = „`Höhe`“.

Abbildung 40: Einträge eines EnumLabel-Widgets

Die Eigenschaft `sysvar` des *EnumLabel*-Widgets wird ebenfalls auf die Selektor-Sysvar gesetzt. Jetzt wird bspw. beim Drücken auf den Set-Button für die `Länge` wieder der Dialog angesprungen, nun steht jedoch zusätzlich auf dem *EnumLabel*-Widget der passenden Text „`Länge`“.

Reihenfolge der *Dialog-Sysvars*-Liste ist wichtig!

Wichtig! die Reihenfolge der *Sysvars*-Liste (*Sysvars*-Tab) ist beliebig, aber die Reihenfolge der Einträge der *Dialog-Sysvars*-Liste ist nicht beliebig! Die Reihenfolge muss sowohl mit den Einträgen eines passenden *EnumLabel*-Widgets als auch mit der Eigenschaft `sysvar value (Number)` aller zu diesem Dialog springenden Set-Buttons abgestimmt werden.

Siehe auch Beispiel-Design `quader_wvga.sgd`

Ein Beispiel-Design namens `quader_wvga.sgd` ist bereits im Ordner GettingStarted vorhanden. Im Beispiel-Design werden nicht nur die Eingabe-Seite sondern auch Mehrsprachigkeit und die Verwendung von **benutzerdefinierten Farben** vorgeführt.

5.5.2 Details: Eingeschaften einer Dialog-Sysvar

Ein **Dialog** enthält als **Page-Eigenschaft** die *Dialog-Sysvars*-Liste. Eine **Dialog-Sysvar** ist ein Eintrag dieser Liste und ermöglicht das Editieren einer einzigen **Systemvariable**

(**Sysvar**) der *Sysvars*-Liste. Die Haupteigenschaft (`sysvar`) weist auf die gewünschte Systemvariable hin (Abbildung 41).

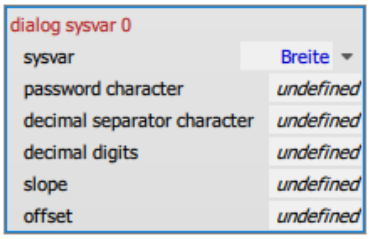


Abbildung 41: Eigenschaften einer Dialog-Sysvar

Wenn nicht alle Dialog-Sysvars für Kennwörter verwendet werden, ist es möglich ein Symbol zum Verdecken des Kennworts (`password character`) individuell bei den relevanten Dialog-Sysvars zu vergeben (und die übergeordnete Eigenschaft `dialog password character` undefiniert zu lassen).

Um Bruchteile zu erlauben, ist es immer möglich, eine `Float`-Sysvar zu verwenden. Aber meist ist es für den Controller effizienter, mit Ganzzahlen zu arbeiten und diese nach Bedarf zu transformieren. Die unten beschriebenen Transformationen sind hauptsächlich für Ganzzahlen (`Number`-Sysvars) gedacht, dürfen aber auch mit `Float`-Sysvars verwendet werden.

Wenn es um eine `Number`-Sysvar handelt, gibt es weitere Eigenschaften, die die Eingabe erleichtern sollen. `Number`-Sysvars sind immer ganzzahlig, der Benutzer denkt jedoch oft in abgeleiteten Einheiten. Z.B. ist die Sysvar in Millimeter (ganzzahlig) und der Benutzer möchte seine Daten in Meter eingeben.

Transformation einer Ganzzahl mithilfe von `slope` und `offset`.

Hierzu gibt es die Eigenschaften `decimal digits` (Anzahl der Stellen hinter dem Komma) und `slope` (Steigung). Möchte man Millimeter in Meter darstellen (bzw. eingeben lassen), setzt man `decimal digits` = `3` und `slope` = `0.001`. Möchte der Benutzer anstatt eines Punktes ein Komma als Trennzeichen verwenden, so kann dies über die Eigenschaft `decimal separator character` erreicht werden.

Wichtig: Die Eigenschaft `decimal separator character` gilt nur für die reine Darstellung auf dem Display-Modul. Die numerischen Eigenschaften des GUI-Designers `slope` und `offset` müssen immer mit Punkt-Trennung eingegeben werden!

Man darf natürlich auch einen `offset` (Y-Achsenabschnitt) eingeben, um einen linearen Zusammenhang abzubilden. Eine Systemvariable könnte z.B. die Temperatur in Celsius beinhalten, der Benutzer in den USA soll aber den Wert in Fahrenheit lesen bzw. eingeben dürfen: Hier wird `slope` = `1.8` und `offset` = `32` gesetzt.

`offset` und `slope` sind Eigenschaften, die Bruchteile enthalten dürfen.

Wichtig: Der `offset` ist immer in Benutzereinheiten und nicht in Sysvar-Einheiten einzugeben (in unserm Beispiel, in Fahrenheit und nicht in Celsius). D.h. konkret, dass der `offset`, wie der `slope`, Bruchteile enthalten darf.

5.6 Mathematische Formeln (*Expressions*-Tab)

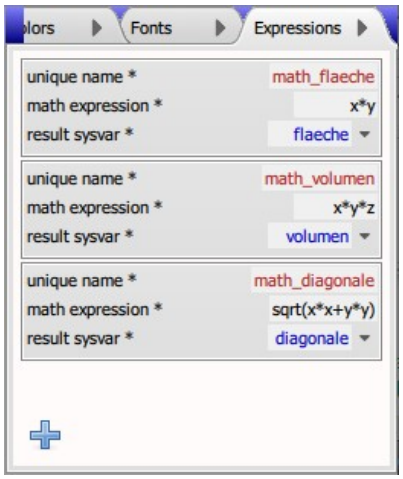


Abbildung 42: Expressions-Tab mit drei Formeln

Es ist möglich, mehrere Systemvariablen in einer mathematischen Formel automatisch auszuwerten und das Ergebnis einer weiteren Systemvariable zuzuweisen. Das wird über den *Expressions*-Tab eingerichtet.

Der *Expressions*-Tab liegt standardmäßig im oberen rechten Fenster und enthält eine Liste von Formeln, deren Variablen und Ergebnis allesamt mit Systemvariablen verbunden werden. In Abbildung 42 gibt es drei Formeln, die in der Lage sind, die Fläche, Volumen und Diagonale eines Quaders zu rechnen und die Ergebnisse an den Systemvariablen `flaeche`, `volumen` und `diagonale` weiterzugeben.

Eine Formel enthält neben dem **eindeutigen Namen** (`unique name`) Eigenschaften für die mathematische Formel selbst (`math expression`) und die Systemvariable, die das Ergebnis empfangen soll (`result sysvar`). Diese Systemvariable darf entweder eine `Number`-Sysvar oder eine `Float`-Sysvar sein.

Achtung! Das Ergebnis wird bei `Number`-Sysvars entsprechend gerundet. `Float`-Sysvars entsprechen lediglich 32-Bit-Floatwerte. Arithmetischer Überlauf oder Unterlauf wird nicht geprüft. Es ist Sache der Kunde, robuste Formeln und plausible Eingabewerte sicherzustellen, z.B. über die Minimal- und Maximalwerte jeder Systemvariable.

Es folgt eine Liste Variablen, die die schriftliche Variablen in der Formel (`variable name`) mit Systemvariablen verbinden (`sysvar`). Diese Systemvariablen dürfen ebenfalls entweder `Number`- oder `Float`-Sysvars sein. In Abbildung 43 wird in der Formel `x*y` die Variable `x` mit der Systemvariable `laenge` und die Variable `y` mit der Systemvariable `breite` verbunden.

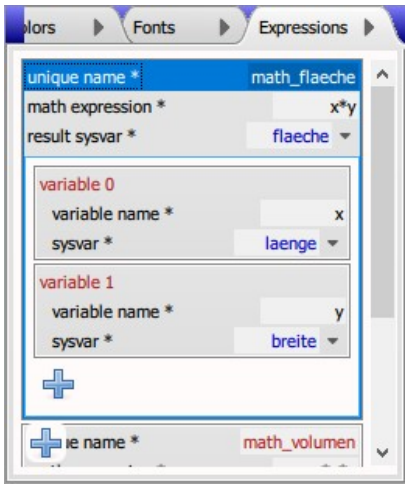


Abbildung 43: Formel mit zwei Variablen für x und y

Wenn einer der „Variable“-Sysvars sich ändert, wird automatisch das Ergebnis neu berechnet und ans „Ergebnis“-Sysvar weiter gegeben.

5.6.1 Unterstützte Operatoren und Funktionen

Folgende Operatoren werden unterstützt:

\+ – * / sowie % (Modulo) und ^ (Potenz).

Es gibt auch die Konstanten „pi“ und „e“.

Es werden auch einige einfache Funktionen unterstützt, die in den nachfolgenden Tabellen kurz erläutert werden.

Funktionen mit einem Parameter

abs(x)	Absolutwert
acos(x)	Arkuskosinus
asin(x)	Arkussinus
atan(x)	Arkustangens
ceil(x)	Aufrundung
cos(x)	Kosinus
cosh(x)	Hyperbelkosinus
exp(x)	e-Funktion
fac(x)	Fakultät
floor(x)	Abrundung
ln(x)	natürlicher Logarithmus
log(x)	Zehnerlogarithmus
sin(x)	Sinus
sinh(x)	Hyperbelsinus
sqrt(x)	Quadratwurzel
tan(x)	Tangens
tanh(x)	Hyperbeltangens

Funktionen mit zwei Parameter

atan2(a, b)	Arkustangens von b/a
equal(a, b)	1 wenn a == b, sonst 0
geq(a, b)	1 wenn a >= b, sonst 0
greater(a, b)	1 wenn a > b, sonst 0
leq(a, b)	1 wenn a <= b, sonst 0
less(a, b)	1 wenn a < b, sonst 0
ncr(n, k)	Binomialkoeffizient n über k
neq(a, b)	1 wenn a != b, sonst 0
npr(n, k)	Variation ohne Zurücklegen [n!/(n-k)!]
pow(a, b)	Potenz a^b

6. Basiskomponenten: Widgets

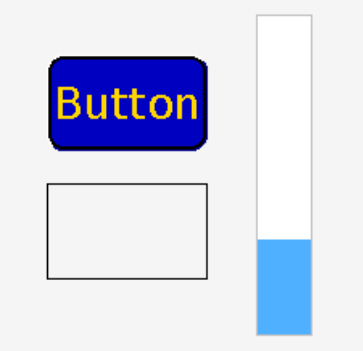


Abbildung 44: Widgets: Button, Rechteck und Balkenanzeige (Standardeinstellungen)

Die graphische Bausteine eines jeden Designs sind die **Widgets** (Abbildung 44). Widgets werden in drei Hauptgruppen unterteilt: die Berührbare Widgets (z.B. ein **Button** oder **Schieberegler**), die Repräsentations-Widgets, die zwar nicht berührbar aber trotzdem über eine **Systemvariable** dynamisch gesteuert werden (z.B. eine **Balkenanzeige**), und die Statische Widgets, die sich nicht verändern (z.B. ein **Rechteck** oder ein **statisches Bild**). Diese Gruppen und die Widgets darin werden in separaten Kapiteln in Detail erläutert. In diesem Kapitel besprechen wir die allgemeine Merkmale der Widgets.

Die Hauptmerkmal eines Widgets ist sein **Widget-Typ**, z.B. *Button*, *Image* oder *EnumLabel*. Jeder Widget-Typ hat entsprechende **Eigenschaften**. Manche Widget-Typen haben auch entsprechende **Stile** (**Styles**), damit eine Gruppe grafischer Eigenschaften (z.B. Farben) einmal definiert und für mehrere gleichartige Widgets verwendet werden können.

Neue Widgets werden über den New-Tab oder auch durch Kopieren und Einfügen erzeugt. Danach können sie über den Widgets-Tab umsortiert werden. Hier kann ein Widget über ein **Kontextmenü** (rechte Maustaste) dupliziert, gelöscht oder exportiert werden. Das Duplizieren bzw. Löschen eines Widgets geht aber auch über das Edit-Menü bzw. die entsprechenden Hotkeys. Ebenfalls im *Widgets*-Tab oder direkt auf der Oberfläche der *Design Area* werden auch Widgets selektiert, um diese zu bearbeiten.

Die Eigenschaften und Sichtbarkeit des zuletzt selektierten Widgets erscheinen im Properties-Tab und evtl. auch im Styles-Tab. Die Farbeigenschaften des Widgets oder des Stils können im Klartext oder als **benutzerdefinierte Farben** (Colors-Tab) bestimmt werden. Die obengenannte Tabs werden in diesem Kapitel näher erläutert. Es gibt auch Eigenschaften die auf **Sysvars**, **Images**, **Fonts** oder **Text-Strings** verweisen. Damit sind ziemlich alle Tabs der GUI-Designer Oberfläche in verschiedenerlei Hinsicht mit den Widgets verbunden!

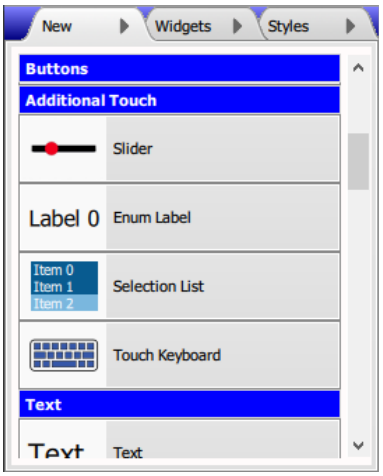


Abbildung 45: Ein Teil des New-Tabs (Additional Touch)

6.1 Widgets erzeugen (*New*-Tab)

Der *New*-Tab liegt standardmäßig im unteren linken Fenster und enthält eine Liste aller möglichen **Widget-Typen**, sowie die gebündelten **Stilgruppen**. Die Widget-Typen werden in mehrere Abschnitte unterteilt: *Buttons* und *Additional Touch* (**berührbare Widgets** – Abbildung 45), *Text*, *Diagram*, und *Additional Dynamic Visuals* (**Repräsentations-Widgets**), und *Static Visuals* (**statische Widgets**). Im Gegensatz zu anderen Listen können die Elemente dieser Liste nicht umsortiert werden, aber einzelne Abschnitte können aus- bzw. eingeblendet werden, indem man auf den Abschnittsnamen klickt oder mit

den Tasten Bild-↓ (PgDn) und Bild-↑ (PgUp) von einem Abschnitt zum nächsten bzw. vorigen Abschnitt schreitet.

Es gibt für jeden Widget-Typ ein Element in der Liste, aber der **Button** bildet eine Ausnahme. Das *Button*-Widget hat so viele verschiedene Funktionsmöglichkeiten, dass dieser Typ wiederum in Subtypen unterteilt worden ist. Deswegen haben die Buttons einen eigenen Abschnitt, indem mehrere Buttons zur Verfügung stehen, die nicht miteinander verwechselt werden dürfen. Auch der **Diagramm-Container** (*Diagram*-Widget) hat einen eigenen Abschnitt, damit seine **Plots** samt **Stile** sowie Stile für die **Achsen** (*axes*) einfacher erzeugt und mit dem Diagramm-Container verbunden werden können.

Bemerkung: Im **Hintergrund-Modus** dürfen nur *Image*-Widgets auf dem Hintergrund platziert werden. In diesem Fall verschwinden alle andere Widget-Typen bzw. Stilgruppen aus der Liste, bis der Hintergrund-Modus wieder verlassen wird.

Alternativverfahren: mit der Maus oder mit Tab- und Pfeiltasten einen Widget-Typ im *New*-Tab selektieren, dann mit der Eingabetaste ein neues Widget mittig im *Design Area* positionieren.

Um ein Widget zu erzeugen, wird mit Drag & Drop ein Widget-Typ in der Liste ausgewählt und auf die *Design Area* gezogen und dort abgelegt. Ein „Standard“-Widget des Typs wird an der Drop-Position erzeugt. D.h. alle erforderlichen Eigenschaften (z.B. die Größe) werden bestimmt (Ausnahme: `sysvar`) und alle optionalen Eigenschaften bleiben undefiniert (bzw. nehmen Standardwerte an), sozusagen als Ausgangspunkt einer gewünschten Darstellung.

Nachdem die Eigenschaften eines Widgets zufriedenstellend angepasst worden sind, ist es meist einfacher das angepasste Widget zu kopieren anstatt immer von neuem anzufangen. Die *Edit*-Menü Kommandos *Cut Widgets*, *Copy Widgets*, *Delete Widgets* und *Select All Widgets* wirken auf die (selektierten) Widgets der aktuellen Seite. Widgets in der Zwischenablage können mit *Edit > Paste Widgets* wieder auf der gleichen oder einer anderen Seite eingefügt werden. Das kopierte Widget bekommt einen neuen **eindeutigen Namen** (`unique name`); ansonsten bleiben alle andere Eigenschaften gleich dem alten Widget. Beim Einfügen auf der gleichen Seite heißt das: Das neue Widget landet genau über dem alten und muss zuerst zur Seite geschoben werden, damit beide Widgets sichtbar sind.

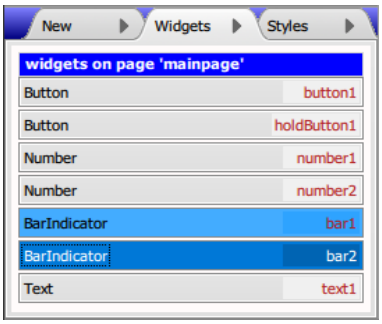


*Abbildung 46: Widgets-Tab aus dem Beispiel getting_started.sgd – mit **Mehrfachauswahl***

6.2 Reihenfolge bestimmen (*Widgets*-Tab)

Ebenfalls im unteren linken Fenster ist standardmäßig der *Widgets*-Tab. Der *Widgets*-Tab enthält alle Widgets der aktuellen Seite (Abbildung 46). Hier werden nur der **Widget-Typ** und der **eindeutige Name** angezeigt; alle andere Eigenschaften stehen im Tab *Properties*. Sowohl hier als auch im Tab *Properties* ist der eindeutige Name editierbar. Der Name darf keine Leerzeichen, Umlaute oder Sonderzeichen (außer dem Unterstrich) enthalten. Ferner darf er nicht mit einem Ziffer anfangen.

Wie in anderen Listen, können hier die Elemente mit Drag & Drop umsortiert werden. Die Position der Widgets in der Liste hat allerdings eine wichtige Bedeutung: Sie entscheidet über die Reihenfolge in der die Widgets gezeichnet werden. Wenn es zu Überschneidungen kommt, wird das zuletzt gemalte Widget das zuerst gemalte Widget teils oder ganz überdecken. Die Reihenfolge darf daher bewusst eingesetzt werden, um z.B. ein ausgefülltes **Rechteck** oder einen **freistehenden Rahmen** (*Frame*-Widget) zu malen und danach **Buttons** oder andere Widgets darauf zu platzieren.

Wichtig! Die Reihenfolge im *Widgets*-Tab entscheidet über die Reihenfolge des Zeichnens!

Nur in dieser Liste dürfen mehrere Widgets gleichzeitig ausgewählt werden (**Mehrfachauswahl** – siehe Abbildung 46). Mit ⇧+Klick (Umschalttaste + Mausklick) kann ein zusammenhängender Bereich markiert werden. Mit Strg-Klick (Steuerungstaste + Mausklick) können einzelne Widgets dazu (de-)selektiert werden. Die *Edit*-Kommandos *Cut Widgets*, *Copy WIdgets* und *Delete Widgets* wirken auf alle selektierten Widgets. Das Ziehen eines der selektierten Widgets mit der Maus in der *Design Area* (um dessen Position zu verschieben) zieht alle anderen selektierten Widgets mit.

Nur im *Widgets*-Tab gibt es die **Mehrfachauswahl**!

Der *Widgets*-Tab ist auch nützlich, wenn ein Widget unsichtbar oder überdeckt ist. Es ist in diesem Fall immer möglich, das Widget aus der *Widgets*-Liste zu selektieren, um dessen Eigenschaften anzuzeigen bzw. anzupassen.

Tipp! Im *Widgets*-Tab können auch unsichtbare Widgets selektiert werden!

Es gibt auch hier ein **Kontextmenü**, das ein einziges Widget sich duplizieren, löschen oder exportieren lässt. Das Duplizieren bzw. Löschen lassen sich auch über das Edit-Menü bzw. die entsprechenden Hotkeys machen, aber das Exportieren geht nur über das Kontextmenü. In diesem Fall wird das Widget samt allen benötigten Stile, Sysvars, TextStrings, usw. ebenfalls in die Zwischenablage kopiert (als sogenanntes „Subdesign“). Das Subdesign kann dann in einem anderen Design mit Strg+v (Steuerungstaste+v) eingefügt werden. Die Zwischenablage kann natürlich per Hand in eine leere Textdatei kopiert werden (empfohlene Erweiterung: .sgdi) als eine Art Schnipsel für die mehrfache Verwendung in anderen Designs.

Tipp! Ein Widget samt alle Bezugsobjekte kann als „Subdesign“ in die Zwischenablage exportiert und dann in anderen Designs eingefügt werden.

6.3 Eigenschaften ändern (*Properties*-Tab)

Die **Eigenschaften** (**Properties**) des zuletzt selektierten **Widgets** erscheinen im *Properties*-Tab (Abbildung 47), wo sie den Bedürfnissen der Anwendung angepasst werden können. Der *Properties*-Tab ist standardmäßig im oberen linken Fenster zu sehen. Wenn kein Widget selektiert ist, werden in diesem Tab die **Page-Eigenschaften** der aktuellen Seite angezeigt.

Als erstes sieht man immer den **Widget-Typ** und den **eindeutigen Namen**. Letzteres kann sowohl hier oder auch im

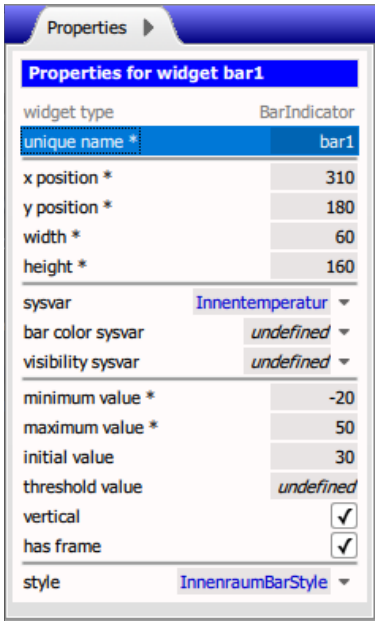


Abbildung 47: Properties-Tab zeigt die Eigenschaften einer Balkenanzeige

Widgets-Tab umbenannt werden.

In den weiteren Zeilen stehen die Position und Größe des Widgets, wobei manche Widgets ihre Größe (auch teilweise) selbst ermitteln (z.B. *Image*-Widgets). Die Größe wird meist durch die Eigenschaften `width` und `height` bestimmt (beim **Kreis** gibt es hier die Eigenschaft `radius`).

Im dritten Abschnitt werden die **Sysvar**-Verbindungen bestimmt, wobei es immer mindestens eine Systemvariable für die **Sichtbarkeit** (`visibility sysvar`) geben kann. Die **statischen Widget-Typen** haben z.B. keine weitere Sysvar-Verbindungen. Das **Datum** (*Date*-Widget) hat dagegen drei weitere Sysvar-Verbindungen jeweils für Tag, Monat und Jahr. Da den Wert einer Systemvariable leicht vom **Controller** zu ändern ist, nehmen solche Änderungen automatisch Einfluss auf die Darstellung des Widgets, ohne dass sich der Controller selbst um die grafische Darstellung kümmern muss.
Danach kommen weitere Bereiche mit typspezifische Eigenschaften. Am Ende der einfachen Eigenschaften kommen, falls zutreffend, Verbindungen zu einem oder vielleicht auch mehrere **Stilen** (**Styles**) der Styles-Tab mit weiteren gebündelten Eigenschaften.

Manche Widgets verwalten außerdem eine untergeordnete Liste, die hinter allen anderen Eigenschaften erscheint. Siehe Details: Untergeordnete Listen.

6.3.1 Details: Editieren der Eigenschaften

Das Editieren einer Eigenschaft verhält sich unterschiedlich, je nach dem, was für einen Datentyp sich hinter der Eigenschaft verbirgt:

Abbildung 48: Numerische Eigenschaften

Abbildung 49: Boolesche Eigenschaften

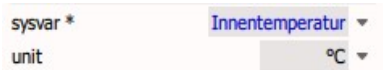


Abbildung 50: String-Eigenschaften

- Numerische Eigenschaften (Abbildung 48) werden mit der Eingabe- oder F2-Taste, sowie per Mausklick im Textfeld als Text direkt editiert. Bis auf wenigen Ausnahmen (z.B. `slope` und `offset`) sind diese Eigenschaften immer ganzzahlig. Viele sind auch immer positiv (oder Null). Achtung! Illegale Werte werden ignoriert, aber unwahrscheinliche Werte werden oft akzeptiert.
- Boolesche Eigenschaften (Abbildung 49) werden als *Checkbox* dargestellt und mit der Eingabetaste, Spacebar, oder per Mausklick umgeschaltet.
- String-Eigenschaften (Abbildung 50) dürfen mit der F2-Taste oder per Mausklick im Textfeld direkt eingegeben werden. Meistens aber gibt es auch ein **Aufklappmenü**, um den Wert vorzugsweise aus einer Liste zu wählen. Die Eingabetaste oder ein linker Mausklick auf den Pfeil (▼) klappt das Menü auf. Gewählt wird durch die Cursortasten (und abgeschlossen mit der Eingabetaste) oder per Mausklick. In diesem Fall erscheint der String oft in blau

Tipp bei Bezugsobjekte: Ein rechter Mausklick auf dem Pfeil des Aufklappmenüs springt zum Bezugsobjekt! D.h. der entsprechende Tab wird in den Vordergrund gebracht und das Objekt wird selektiert.

Abbildung 51: Farbeigenschaften

anstatt in schwarz. Das bedeutet, dass der String sich auf einem anderen Objekt bezieht, und korrekterweise den **eindeutigen Namen** dieses Objekts wiedergibt. Einfache Klartext wird in schwarz dargestellt. Wenn ein Bezug erwartet wird, sollte hier geprüft werden, ob ein Tippfehler vorliegt oder ob das Bezugsobjekt inzwischen gelöscht wurde.

- Farbeigenschaften (Abbildung 51) können ebenfalls mit der F2-Taste oder per Mausklick im Textfeld direkt eingegeben werden. Hier gibt es aber zusätzlich ein **RGB-Symbol** ( ), um das **Farbdialog** zu öffnen. Falls es **benutzerdefinierte Farben** gibt, steht auch ein Aufklappmenü zur Verfügung. Die Eingabetaste öffnet vorzugsweise das Aufklappmenü, wenn vorhanden, ansonsten das Farbdialog. Weitere Darstellungsmerkmale: → Benutzerdefinierte Farben werden in blau, alle andere Werte mit schwarz dargestellt. → Wenn die Maus über das RGB-Symbol schwebt, wird der aktuelle Farbton als Muster links eingeblendet. → Bei benutzerdefinierten Farben kann auch mit einem rechten Mausklick auf den Pfeil (▼) zur Farbe im *Colors*-Tab gesprungen werden.

Weitere allgemeine Merkmale des Editierens:

- Die Eingabe- und F2-Taste werden nur dann wirksam, wenn die Eigenschaft hervorgehoben („aktiv“) und der Name der Eigenschaft (links) mit einer gestrichelten Linie umringt ist.
- Bei direkter Texteingabe werden Änderungen erst dann in der *Design Area* sichtbar, nachdem die Eingabe abgeschlossen und akzeptiert wird, entweder mit der Eingabetaste oder durch einen Mausklick woanders.
- Um Textfelder zurück zu stellen, kann der Inhalt des Feldes einfach gelöscht werden. Im Fall eines leeren Textfeldes erscheint automatisch das Wort *undefined* (kursiv!) und ein Standardwert wird benutzt.

6.3.2 Details: Untergeordnete Listen

Manche Widgets brauchen eine variable Anzahl von Eigenschaften, die in einer untergeordneten Liste verwaltet werden.

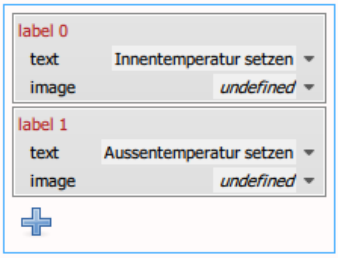


Abbildung 52: Liste aller Einträge eines EnumLabel-Widgets

Manche Widgets verwalten neben den einfachen Eigenschaften auch eine untergeordnete Liste. Diese Liste enthält eine beliebig Anzahl von Einträgen mit weiteren Eigenschaften. Als Beispiel nehmen wir die **indizierte Textauswahl** (*EnumLabel*-Widget). Sie hat eine untergeordnete Liste, deren Einträge u.a. die `text` und `image` Eigenschaften für jeden möglichen Index der Auswahl enthält (Abbildung 52).

Die Einträge in dieser Liste haben meist keinen **eindeutigen Namen** (Ausnahme: **Plots**). Sie sind lediglich indiziert und werden über den Index verwaltet. Wieder am Beispiel der indizierten Textauswahl: ein *EnumLabel*-Widget wird mit einem numerischen **Systemvariable** verknüpft, die den aktuellen Index vorgibt.

Untergeordnete Liste funktionieren weitgehend wie andere Liste, sind aber lediglich indiziert (d.h. die Einträge haben keinen **eindeutigen Namen**).

Untergeordnete Listen verhalten sich ansonsten wie die meisten anderen Listen. Der aktuelle Eintrag wird blau umrahmt und aufgeklappt, damit alle Eigenschaften sichtbar sind. Alle andere Einträge werden minimal dargestellt (nur der Index und die Haupteigenschaften sind sichtbar). Man kann mit Drag & Drop die Liste umsortieren, mit dem Plussymbol () einen neuen Eintrag anlegen, oder über das **Kontextmenü** eines Eintrags den Eintrag duplizieren bzw. löschen.

Hierzu muss beachtet werden, dass irgendeine gekoppelte Indexverwaltung die tatsächliche Anzahl der Einträge nicht übersteigt.

6.4 Sichtbarkeit von Widgets (Visibility)

Die **Sichtbarkeit** eines **Widgets** (**Button**, **Schieberegler**, **Kreis**, **Linie** usw.) kann durch den Wert einer **Systemvariable** (**Sysvar**) beeinflusst werden.

Jedes Widget hat vier Eigenschaften (im *Properties*-Tab editierbar), die die Sichtbarkeit beeinflussen:

Die Sichtbarkeit eines Widgets wird über eine Sysvar, eine Maske und zwei Grenzwerte bestimmt

- `visibility sysvar`: Der Name einer Systemvariable (d.h. eine Sysvar der *Sysvars*-Liste), die die Sichtbarkeit kontrolliert.
- `16-bit mask`: Eine binäre Maske, mit der die Sysvar maskiert werden kann (Und-Verknüpfung). Standardwert: 65535 (0xffff), d.h. es wird nichts maskiert.
- `low limit`: Untere Grenze des maskierten Sysvar-Werts für die Sichtbarkeit des Widgets. Standardwert: 1
- `high limit` : Obere Grenze des maskierten Sysvar-Werts für die Sichtbarkeit des Widgets. Standardwert: 65535 (0xffff).

Die oberste Eigenschaft (`visibility sysvar`) ist immer im dritten Abschnitt des *Properties*-Tabs sichtbar. Die anderen Eigenschaften erscheinen etwas eingerückt erst dann, wenn diese gesetzt wird.

Wie wird die Maske und die Grenzwerte eingesetzt?

Der Test auf Sichtbarkeit erfolgt dann bei der Darstellung in zwei Schritten:

- Zunächst wird die Sysvar mit der Maske verknüpft und man erhält den maskierten Sysvar-Wert:
  `value = (sysvar & mask)`.
- Im zweiten Schritt werden `low limit` (`lo`) und `high limit` (`hi`) betrachtet und dadurch drei Fälle unterschieden:
  1. `lo == hi`
     Wenn nun `value = lo` ist, dann ist das Widget sichtbar.
  2. `lo < hi`

Wenn `lo <= value <= hi`, dann ist das Widget sichtbar.

Sonderfall! Der untere Grenzwert darf über den oberen Grenzwert liegen!

3. `lo > hi` (Sonderfall!)
   Wenn `value >= lo`, dann ist das Widget sichtbar.
   Wenn `value <= hi`, dann ist das Widget sichtbar.
   D.h. es gibt einen Bereich zwischen `high limit` und `low limit` bei dem das Widget **unsichtbar** ist.
   Bsp.: `high limit = 3` und `low limit = 5`, dann ist das Widget nur beim Wert `value = 4` unsichtbar und ansonsten sichtbar.

Im einfachen Fall reichen die Standardwerte und eine numerische Sysvar mit dem Wert 0 oder 1.

Dieses Verfahren erlaubt ziemlich komplizierte Kombinationen von sichtbaren und unsichtbaren Widgets. Aber für den einfachen Fall reicht es, eine numerische Sysvar zu definieren, deren Wert entweder `0` oder 1 sein kann. Ein Widget, das sein `visibility sysvar` mit dem Sysvar verknüpft (und die anderen Eigenschaften auf die Standardwerte belässt), wird unsichtbar, wenn die Sysvar auf 0 steht, ansonsten sichtbar.

6.5 Eigenschaften bündeln (*Styles*-Tab)

Stile (**Styles**) enthalten das Look-and-Feel eines Widgets, damit es von mehrere Widgets benutzt werden kann.

Stile (**Styles**) beinhalten eine Sammlung Eigenschaften, die zu einer einheitliche Darstellung der **Widgets** beiträgt. Ein **Stil** oder eine **Stilgruppe** wird einmal definiert und kann dann von mehreren Widgets verwendet werden. Stile enthalten meist die Farbgebung, Textmerkmale und etwaige Rahmen, d.h. das Look-and-Feel des Widgets.

Stile werden im GUI-Designer in den Tab *Styles* angelegt. Diese Stile werden dann anderen Widgets über die Widget-Eigenschaft `style` (manchmal genauer benannt, z.B. `selected style`) zugewiesen. Das Widget benutzt die Eigenschaften des Stils für seine Darstellung.

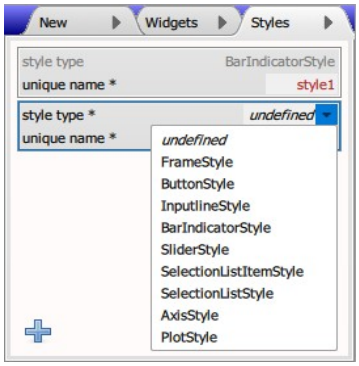


Abbildung 53: Die erlaubte Typen für Stile

Der *Styles*-Tab befindet sich standardmäßig im unteren linken Fenster des GUI-Designers. Darin werden alle bisher definierten Stile des Designs als eine Liste von Einträgen dargestellt. Der aktuelle Eintrag wird blau umrahmt und aufgeklappt, damit alle Eigenschaften sichtbar sind. Alle andere Einträge werden minimal dargestellt (nur **Stiltyp** und Name sind sichtbar – Abbildung 53).

Ein neuer Eintrag wird minimal dargestellt bis der Stiltyp (die Eigenschaft `style type`) festliegt. Um den Stiltyp zu bestimmen, wird auf den Pfeil (▼) geklickt und im **Aufklapp-menü** eine der Stiltypen ausgewählt. (Abbildung 53). Danach kann der Stiltyp nicht mehr geändert werden und das Aufklapp-menü verschwindet.

Wie bei den meisten Listen, kann man mit Drag & Drop die Liste umsortieren, mit dem Plussymbol ( ) einen neuen Eintrag

anlegen, oder über das **Kontextmenü** eines Eintrags den Eintrag duplizieren bzw. löschen. Hier ist zu beachten, dass duplizierte Einträge auch den Stiltyp erben. Wenn eine anderer Stiltyp gewünscht wird, muss stattdessen das Plussymbol verwendet werden, um einen zunächst „typlosen“ Stil zu erzeugen.

Zusätzlich (wiederum im Kontextmenü) darf der Stil samt untergeordneten Stile und weitere Bezugsobjekte (z.B. Benutzerdefinierte Farben (Colors-Tab)) in die Zwischenablage „exportiert“ werden (*Export style to clipboard*) als sogenannte „Subdesign“. Der Inhalt der Zwischenablage kann in einem beliebigen Design mit Strg+v (Steuerungstaste+v) wieder „importiert“ (eingefügt) werden. Die Zwischenablage darf auch per Hand in eine leere Textdatei kopiert werden (empfohlene Erweiterung: .sgdi) als eine Art Schnipsel für den späteren Gebrauch.

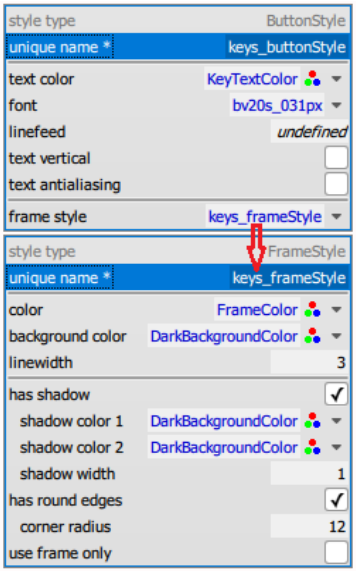


Abbildung 54: ButtonStyle mit gebundenem FrameStyle

In allen Fällen bekommen neue Einträge automatisch einen **eindeutigen Namen** (`unique name`). Verbindungen zu einem Stil erfolgen immer direkt durch diesen Namen. Man sollte die automatisch generierten Namen durch sinnvolle zum Projekt passende Namen ersetzen.

Abhängig vom Stiltyp erscheinen weitere Eigenschaften. Darunter kann es sogar die Eigenschaft `frame style` geben, die auf einem weiteren untergeordneten **Rahmenstil** (Stiltyp `FrameStyle`) verweist (Abbildung 54). Das heisst, der vollständige Stil besteht eigentlich aus zwei gebundenen Stile. Trotzdem ist die Reihenfolge der Stile beliebig. Mehrere Hauptstile dürfen den gleichen Rahmenstil benutzen, um eine einheitliche Rahmendarstellung zu erreichen.
Für manche Widgets ist es wichtig, dass es verschiedene Stile gibt – z.B. bei **Buttons** ist es meist ratsam, dass der normale (nicht gedrückte) Zustand sich vom gedrückten Zustand unterscheidet. Man definiert deshalb bei Buttons die Eigenschaften für (normal) `style`, `selected style`, und sogar `ghosted style`.

Abbildung 55: Standard Touch-Keyboard lediglich durch Stile umgestaltet

Wenn die verschiedene Zustände eines Buttons durch Bilder dargestellt werden (wie in Anlegen eines Buttons in Kapitel 2), bräuchten wir keine `ButtonStyles` (samt `FrameStyles`). Es wäre aber durchaus möglich, ähnlich aussehende Buttons komplett über Stile zu erzeugen. Ein zweites Beispiel im GettingStarted Ordner – `getting_started_dialog.sgd` – zeigt, wie eine numerische **Tastatur** (*TouchKeyboard*-Widget) durch geschickt-definierten Stile seine Tasten dem Stil der anderen Bilder-Buttons anpassen kann (Abbildung 55). Der `ButtonStyle`- und `FrameStyle`-Stile für die Tasten (wie hier abgebildet) sind im Abbildung 54 genauer zu sehen.

6.5.1 Rahmenstil (`FrameStyle`)

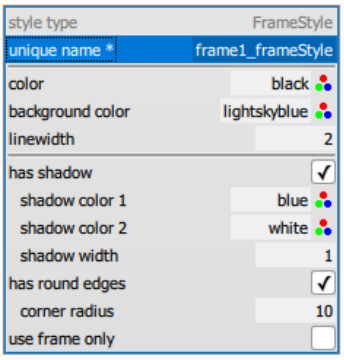


Abbildung 56: Standardstil eines Frame-Widgets

Ein Grundstil ist der **Rahmenstil** (**Stiltyp** `FrameStyle`). Er ist sowohl der Hauptstil des **freistehenden Rahmens** (*Frame*-Widget – Abbildung 56) und der **Tastatur** (*TouchKeyboard*-Widget) als auch ein untergeordneter Stil anderer Stile (z.B. `ButtonStyle`).

Der Rahmenstil gibt die Randfarbe (`color`) und Dicke (`linewidth`), sowie die Hintergrundfarbe (`background color`) der inneren Fläche des Rahmens vor.

Der Rand kann mit gerundeten Ecken gezeichnet werden (`has round edges` und `corner radius`). Der Rahmen darf auch einen Schatten haben (`has shadow`), mit unterschiedliche Schattenfarben unten rechts (`shadow color 1`) und oben links (`shadow color 2`) und mit einer eigenen Breite (`shadow width`).

Tipp! Buttons dreidimensional über **Rahmenstile** gestalten

Tipp! Um das Drücken eines Buttons dreidimensional zu gestalten, werden zwei Rahmenstile – normal und selektiert – definiert, wobei im ersten Stil die dunklere Schattenfarbe unten rechts ist, im zweiten Stil aber oben links. Der erste Rahmenstil wird dann im normalen, der zweite Rahmenstil im selektierten `ButtonStyle` verwendet.

Es ist auch möglich einen ungefüllten Rahmen (`use frame only`) zu definieren. Ein ungefüllter Rahmen überdeckt den Hintergrund innerhalb des Randes nicht. Er darf aber keine gerundeten Ecken haben.

Manche **Widget-Typen** benutzen zwar einen (untergeordneten) Rahmenstil, unterstützen aber nicht unbedingt alle Features des Rahmens. Als Beispiel: Ein „normaler“ **Schieberegler** (*Slider*-Widget) mit Rahmen aber ohne Buttons verträgt keine gerundeten Ecken. Solche Widgets ignorieren unpassenden Eigenschaften.

6.5.2 Weitere Stiltypen

Zusätzlich zum **Rahmenstil** des **freistehenden Rahmens** (*Frame*-Widget) bzw. der **Tastatur** (*TouchKeyboard*-Widget) – wie im vorigen Abschnitt beschrieben – verlagern folgende **Widget-Typen** ihre Farben-, Text- und Rahmeneigenschaften in entsprechenden Stilen ab: **Buttons**, **Eingabezeilen** (*InputLine*), **Balkenanzeigen** (*BarIndicator*), **Schieberegler** (*Slider*), **Auswahllisten** (*SelectionList*) samt deren Einträge (*items*), sowie die **Achsen** (*axes*) und **Plots** (*plots*) eines **Diagramm-Containers** (*Diagram*-Widget). Spezifische Eigenschaften werden in der jeweiligen Widget-Beschreibung weiter unten erläutert. Hier werden aber einige allgemeine Aussagen über die Stileigenschaften gemacht.

Einige Widgets stellen u.a. Texte dar. Das Aussehen der Texte

Abbildung 57: Allgemeine Text-eigenschaften eines Stils

Abbildung 58: Text-Widget links ohne und rechts mit Antialiasing

wird für manche Widgets direkt im Widget festgelegt, für andere Widgets in deren dazugehörenden Stilen. Letzteres trifft für Buttons, Eingabezeilen, die Einträge (*items*) einer Auswahlliste und die Achsen (*axes*) eines Diagramm-Containers zu.

Zu den allgemeinen **Texteigenschaften** im Stil (Abbildung 57) gehören z.B. der gewünschte **Font** (`font`), die Textfarbe (`text color`), evtl. auch eine Hintergrundfarbe (`background color`). Der Text kann auch mit **Antialiasing** (`text antialiasing`) dargestellt werden, wobei der Text deutlich schöner (Stichwort Kantenglättung) aber dafür nur halb so groß erscheint (Abbildung 58). Weitere Texteigenschaften finden Sie bei den Widget-Beschreibungen.

Die **Stiltypen** `ButtonStyle`, `InputlineStyle`, `BarIndicatorStyle`, und `SliderStyle` haben einen untergeordneten Rahmenstil (Eigenschaft `frame style`). Wenn die Eigenschaft undefiniert bleibt, werden für den Widget-Typ sinnvolle Einstellungen gemacht. D.h. der Standardrahmen eines Buttons unterscheidet sich deutlich vom Standardrahmen einer Eingabezeile.

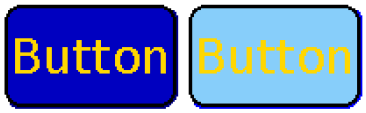


Abbildung 59: Links: Button ohne Rahmenstil (Standardwerte eines Buttons); Rechts: Button mit leerem Rahmenstil (Standardwerte eines Frame-Widgets)

Es gibt aber einen wichtigen Unterschied zwischen der vollkommen undefinierten Eigenschaft `frame style` und einem angehängten Rahmenstil mit eigenen undefinierten Eigenschaften. Wenn ein neuer Rahmenstil hinzugefügt und ohne weiteres einem anderen Stil (z.B. `ButtonStyle`) untergeordnet wird, werden alle im Rahmenstil undefinierten Eigenschaften immer die Standardeinstellungen eines freistehenden Rahmens (*Frame*-Widgets) annehmen und nicht die des Buttons. Hierdurch erhält der Button ein anderes Aussehen (Abbildung 59)!

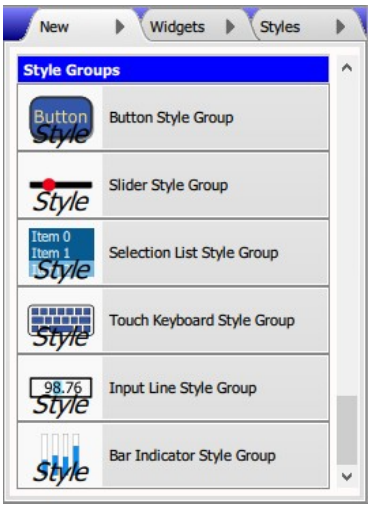


Abbildung 60: Die Stilgruppen im New-Tab

Das Anlegen von **Stilgruppen** (im *New*-Tab – Abbildung 60) adressiert sowohl dieses (unerwünschte) Verhalten als auch die Tatsache, dass manche Widgets mehrere Stile gleichzeitig verwalten müssen.

6.5.3 Details: Stilgruppen erzeugen (*New*-Tab)

Im *New*-Tab gibt es einen Abschnitt namens *Style Groups*. Hier steht für jeden **Stiltyp** ein Eintrag, der die Stil-Bedürfnisse des korrespondierenden **Widget-Typs** völlig abdeckt (Abbildung 60). Das heißt, dass nicht nur ein Stil sondern mehrere zusammenhängende Stile in einem Schritt erzeugt werden. (Bemerkung: Wo nur einen einzigen Stil erzeugt wird, steht diese Stil nicht unter *Style Groups* sondern beim entsprechenden Widget, etwa *Frame Style*, *Axis Style* oder *Plot plus Style*.)

Am effizientesten wird eine solche **Stilgruppe** direkt auf ein bereits angelegtes Widget gezogen und abgelegt (Abbildung 61). Die nötigen Stile werden erzeugt und gleich mit dem

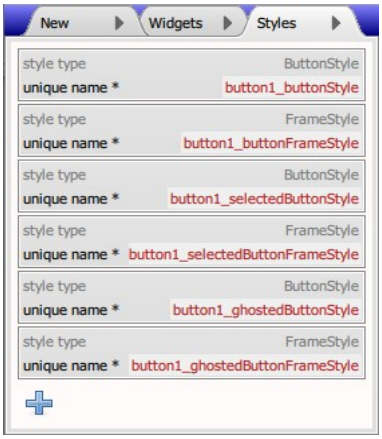


Abbildung 61: Button-Stilgruppe, auf button1 gezogen, erzeugt gleich sechs Stile

Widget verbunden. Hierbei baut die automatische Namensgebung den Namen des Widgets in den Stilnamen hinein, um diese Verwandtschaft zu betonen. In der Stilgruppe werden alle Stileigenschaften mit den gleichen Standardwerten belegt, die ein Widget ganz ohne Stile als Standardwerte verwenden würde. Damit haben Sie einen guten Ausgangspunkt, um die Stile für Ihren Zwecken anzupassen.

Eine Stilgruppe darf auch auf eine leere Fläche in der *Design Area* gezogen werden. In diesem Fall wird ebenfalls die ganze Stilgruppe erzeugt und die übergeordneten Stile mit etwaigen untergeordneten **Rahmenstile** verbunden. Die automatische Namensgebung ist allerdings hier etwas anders aufgebaut und es gibt noch keine Verbindung mit irgendeinem Widget. Diese Methode ist aber trotzdem sinnvoll, vor allem wenn eine Stilgruppe für viele Widgets (z.B. ähnlich aussehende Buttons) verwendet werden soll.

Stilgruppen werden in einer vorgegebenen Reihenfolge am Ende der *Styles*-Liste erzeugt. Allerdings hat diese Reihenfolge keine Bedeutung und die Elemente der Liste dürfen umsortiert werden. Es ist auch möglich, dass gewisse Elemente nicht gebraucht werden, z.B. eine **Balkenanzeige** ohne Rahmen braucht keinen untergeordneten Rahmenstil. Solche Stile dürfen gelöscht werden. Es soll nur darauf beachtet, dass der übergeordnete Stil ebenfalls seine Eigenschaft `frame style` löscht. Noch ein Beispiel: Ein **Button** braucht vielleicht den optionalen `ghosted style` gar nicht. Hier kann in der Stilgruppe der `ghosted`-Stil vom Stiltyp `ButtonStyle` samt seinem untergeordneten Rahmenstil gelöscht werden. Im Button selbst bleibt dann die Eigenschaft `ghosted style` undefiniert.

6.6 Benutzerdefinierte Farben (*Colors*-Tab)

Mehrfach verwendete Farben über die *Colors*-Liste bestimmen.

Es gibt die Möglichkeit **benutzerdefinierte Farben** in der *Colors*-Liste (*Colors*-Tab) anzulegen. Hiermit können Sie abstrakte, zweckorientierte, mehrfach verwendete Farben erstellen (z.B. eine Farbe namens `background_color`), die dann überall im Design als Wert einer passende Farbeigenschaft eingesetzt werden dürfen.

Farbe einmal ändern und alles passt sich der neuen Farbe an!

Durch das ändern des Farbwerts eines Eintrags der *Colors*-Liste, ändern Sie gleichzeitig die Farbe aller Eigenschaften in Design, die diese benutzerdefinierte Farbe verwendet haben. Damit können Sie sehr leicht mit dem Aussehen Ihrer Anwendung experimentieren.

Abbildung 62: Farben der AussenraumBarStyle

Als Beispiel nehmen wir die Farben der `AussenraumBarStyle` aus Stile und Farben am Beispiel der Balkenanzeige in Kapitel 2. Hier haben wir eine RGB-Farbe (`#00aa00`) sowie zwei **CSS-Color-Keywords** (`red` und `lightgray`), nochmal

hier in Abbildung 62 abgebildet.

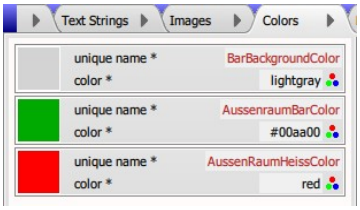


Abbildung 63: Benutzer-definierte Farben anlegen

Wir verschieben diese Farben in die *Colors*-Liste. Dort legen wir drei Farben mit den Namen `BarBackgroundColor`, `AussenraumBarColor` und `AussenraumHeissColor` an, wie in Abbildung 63 dargestellt. Jeder Eintrag hat den entsprechenden Wert, wie im `AussenraumBarStyle`. Hierzu kann man den Wert kopieren, direkt als Text eintippen oder auf dem **RGB-Symbol** ( ) klicken, um mit dem **Farbdialog** eine Farbe auszusuchen.

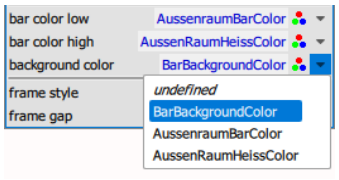


Abbildung 64: Benutzer-definierte Farben anwenden

Wir gehen zurück zum `AussenraumBarStyle`. Jetzt steht neben dem RGB-Symbol ( ) auch noch ein **Aufklappmenü** (▼). Hier kann man aus der Auswahlliste den Namen der benutzerdefinierten Farbe aussuchen (Abbildung 64). Eine Änderung der `BarBackgroundColor` in der *Colors*-Liste wird sich dann auf den `AussenraumBarStyle` sofort auswirken.

Abbildung 65: Über das RGB-Symbol schweben, um das Farbmuster einzublenden

Benutzerdefinierte Farben werden in blau dargestellt. Damit wird sofort erkennbar, ob die Farbe benutzerdefiniert ist oder etwa ein CSS-Color-Keyword darstellt wird. Um die aktuelle Farbe der Eigenschaft zu prüfen, kann man auch bei benutzer-definierten Farben mit der Maus über das RGB-Symbol ( ) schweben. Es wird das aktuelle Farbmuster links daneben eingeblendet, wie im Abbildung 65. Man kann auch mit der rechten Maustaste auf den Pfeil (▼) klicken, um zur benutzer-definierten Farbe in der *Colors*-Liste zu springen.

7. Berührbare Widgets (*Touch*)

Besonders wichtig für ein Touch-Display sind die berührbaren Elementen, damit der Anwender Informationen an die Anwendung weiterreichen kann. Sie befinden sich im Tab *New* in den Abschnitten *Buttons* und *Additional Touch*.

Hierzu gibt es Buttons, einen Schieberegler (*Slider*-Widget), eine Tastatur (*TouchKeyboard*-Widget), und zwei Möglichkeiten, um eine Auswahl zu treffen: die schon bekannte Auswahlliste (*SelectionList*-Widget) und die Indizierte Textauswahl (*EnumLabel*-Widget).

7.1 Buttons (*Button*-Widgets)

Abbildung 66: Button (Standardeinstellungen)

Eine Grundfunktionalität eines Touch-Displays ist **Buttons** anzuzeigen und deren Betätigung zu registrieren. Hierzu gibt es verschiedene Typen von Buttons, die mit unterschiedlichen Arten von **Systemvariablen** (Eigenschaft `sysvar`) zusammenarbeiten. Im *New*-Tab werden gleich sechs Button-Typen im Abschnitt *Buttons* zur Auswahl gestellt.

Abbildung 67: Die verschiedene Button-Typen

Es folgt eine kurze Zusammenfassung der verschiedenen Button-Typen mit ihren besonderen Eigenschaften (siehe auch Abbildung 67):

- **(Standard-)Button** (Abbildung 66): Durch Betätigen dieses Buttons kann der Anwender auf die in der Eigenschaft `destination` angegebene Zielseite (**Page**) springen. Der Standard-Button wird nicht mit einer Systemvariable gebunden. Er wird auch oft – ohne `destination` – verwendet, wenn die Anwendung das Verhalten selbst verwalten soll.

- **Hold-Button** (oder Schalter): Ein Betätigen dieses Buttons schaltet den Zustand der verbundenen booleschen Systemvariable um. Der Button bleibt gedrückt, bis er wieder betätigt wird, bzw. er ändert seinen Zustand, wenn die Systemvariable über andere Wege (z.B. durch die Anwendung) umgeschaltet wird. Wenn eine **Sysvar**-Verbindung fehlt, gibt es ersatzweise die Checkbox `display pressed`, um den Button beim Starten bereits in gedrücktem Zustand darzustellen.

- **Radio-Button**: Ein Betätigen dieses Buttons schaltet den Wert der verbundenen numerischen Systemvariable auf einem festen Wert (`sysvar value`), wie bei einem numerischen Set-Button. Aber der Button bleibt gedrückt, wie beim Hold-Button, bis die Systemvariable über andere Wege einen neuen Wert erhält, meist durch das Betätigen eines anderen Radio-Buttons, der mit dem gleichen Systemvariable verbunden ist.

- **Incr/Decr-Button**: Der Button erhöht bzw. erniedrigt den Wert einer numerischen Systemvariable um die angegebene Schrittweite (`step size`), die auch negativ sein darf. Wenn der Maximum- bzw. Minimumwert der Systemvariable erreicht wird, erfolgt keine weitere Änderung mehr. Durch die Eigeschaft `loop` springt die Systemvariable automatisch zum anderen Ende der Skala, wenn der Maximum- bzw. Minimumwert überschritten wird.

- Bei **Set-Buttons** für `Number`-, `Float`–, `String`- und `Color`-Systemvariablen wird diese auf einen festen Wert (`sysvar value`) gesetzt und dann optional auf eine Zielseite (`destination`) gesprungen. Eine Hauptaufgabe für numerischen Set-Buttons ist es, die **Selektor-Sysvar** einer **Eingabe-Seite** (**Dialog**) zu setzen und dann dorthin zu springen.

Wenn der Button weder `sysvar`- noch `destination`-Verbindungen hat, wird beim Betätigen des Buttons einfach dem Controller einen `BUTTON_PRESSED_EVENT` zugeschickt und die Anwendung entscheidet, was zu tun ist. Ansonsten agieren die Buttons komplett selbstständig. Die Anwendung braucht nur etwaige Änderungen der Systemvariablen oder Page-**Events** zu beachten.

Abbildung 68: (Standard) Button in normalem (links) und deaktiviertem Zustand

Alle weiteren Eigenschaften sind den Button-Typen gemein. Sie haben die Standardeigenschaften für Position, Größe und **Sichtbarkeit**. Ein Button kann beim Berühren einen hörbaren Ton von sich geben (`audible click`). Er kann anfänglich deaktiviert gezeichnet werden (`display ghosted`), wobei hierzu entweder ein zusätzliches Bild (`ghosted image`) oder ein spezieller Stil (`ghosted style`) erforderlich ist (Abbildung 68).

Mit `make button transparent` verwandelt sich der Button in ein touch-sensitives Rechteck (vgl. auch die Sichtbarkeit über die Eigenschaft `visibility sysvar`). Ein transparenter Button wird nicht gezeichnet, der Button ist aber aktiv.

Abbildung 69: Ein selektierter Button links; rechts der Gleiche aber durchsichtig (`alpha color` = #b11500)

Wenn ein Button aber mit einem Bild (`image`) gezeichnet wird, gibt es stattdessen die Eigenschaft `make alpha color transparent`. Hiermit betrachtet der Button die **Alphafarbe** (`alpha color`) der jeweiligen **Image** (`image`, `selected image` oder `ghosted image`) als durchsichtig. Alles andere ist weiterhin sichtbar (Abbildung 69). Images ohne definierte Alphafarbe bleiben unberührt. Es gibt hier Einschränkungen – mehr dazu unter Details: Alphafarbe.

Ein Button hat bis zu drei Zustände: normal (nicht gedrückt), selektiert (gedrückt), und deaktiviert (ausgegraut). Es gibt für jeden Zustand die Möglichkeit, die Texte, Bilder, und Stile

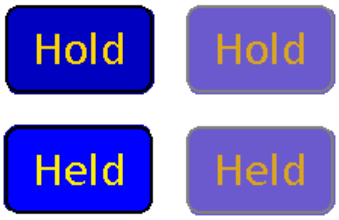


Abbildung 70: Hold-Button in normalem (oben) und selektiertem Zustand (unten), rechts jeweils deaktiviert

anzupassen. Für den normalen Zustand gibt es die Eigenschaften `text`, `image`, und `style`. Für den selektierten Zustand gibt es `selected text`, `selected image`, und `selected style`. Für den deaktivierten Zustand gibt es `ghosted image` und `ghosted style` (ein separater Textinhalt dazu wird nicht unterstützt). Als Beispiel zeigt Abbildung 70 einen **Hold-Button** mit den Standard-einstellungen.

Ein Button kann Texte, Bilder oder beides haben. Der Text darf entweder Klartext sein oder sich auf einen **Text-String** der *Text-Strings*-Liste beziehen. Enthält der Button zusätzlich zum Text auch ein Bild, so wird der Text immer vor dem Bild gezeichnet. Hierdurch lassen sich verschiedene Buttons mit dem gleichen Hintergrund ausstatten. Die Bilder für die verschiedene Zustände sollten die gleiche Größe haben. Hier ist zu beachten, dass das Hauptbild (`image`), wenn vorhanden, die Gesamtgröße des Buttons bestimmt. D.h. konkret: die Eigenschaften für die Breite und Höhe (`width` und `height`) werden durch die nicht mehr editierbare `image width` und `image height` ersetzt (Abbildung 71).

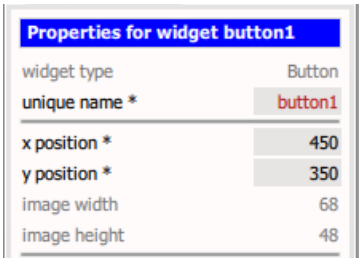


Abbildung 71: Button mit gesetzter `image`-Eigenschaft; Breite und Höhe sind nicht mehr editierbar

Bemerkung: Der GUI-Designer ändert nie die Pixelgröße eines Bildes. Solche Anpassungen müssen extern – z.B. mittels eines Bildbearbeitungsprogramms – vorab durchgeführt werden.

Wenn Buttons nicht ausschließlich über Bilder gezeichnet werden, benutzen sie Stile vom Typ `ButtonStyle` für den normalen Zustand (`style`), den gedrückten Zustand (`selected style`) und den deaktivierten Zustand (`ghosted style`). Jeder dieser Button-Stile hat zusätzlich einen untergeordneten **Rahmenstil** (Eigenschaft `frame style`). Ein Beispiel ist in Abbildung 72 zu sehen. Es ist ratsam, eine **Stilgruppe** *Button Style Group* aus dem Tab *New* (Abschnitt *Style Groups*) anzulegen, wobei alle sechs Stile auf einmal angelegt und miteinander verbunden werden. Eine solche Stil-gruppe kann dann für mehrere oder gar alle Buttons verwendet werden, um ein gleichartiges Aussehen zu erreichen und spätere Stiländerungen zu erleichtern.

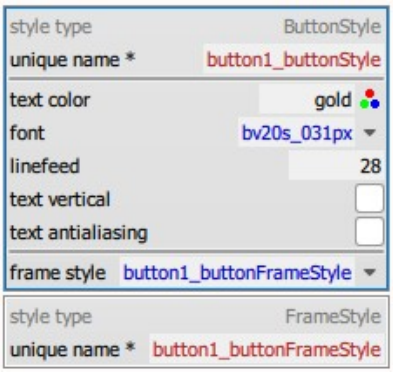


Abbildung 72: ButtonStyle für den normalen Zustand (Standardeinstellungen)

In jedem Stil vom Typ `ButtonStyle` werden die üblichen **Text-eigenschaften** `font`, `text color`, und `text anti-aliasing` bestimmt. Ferner kann der Text senkrecht orientiert dargestellt werden (`text vertical`). Die Hintergrundfarbe des Textes wird ausnahmsweise nicht hier angegeben. Stattdessen wird die Hintergrundfarbe des untergeordneten Rahmenstils (`frame style`) verwendet.

Buttons haben auch die Besonderheit, dass sie mehrzeiligen Text darstellen können. Mehrzeilige Texte benutzen keinen

Abbildung 73: Buttons mit mehrzeiligen Texten: a) korrekt! (linefeed im Stil gesetzt), b) wie (a), aber ohne linefeed im Stil, sodass die 1. Zeile überdeckt wird

automatischen Zeilenumbruch, sondern müssen an allen Umbruchstellen die Buchstabenkette „\n“ (einen Backslash und die Buchstabe ‚n‘) enthalten, die in einen echten Zeilenumbruch bei der Darstellung umgewandelt wird. Hierzu legt die Eigenschaft `linefeed` (im Stil) den Zeilenabstand fest. Der Zeilenabstand wird in Pixel zwischen den Grundlinien der Zeilen gemessen.

Wichtig! Bei mehrzeiligen Texten (und nur dann) ist die Eigenschaft `linefeed` zwingend! Wenn `linefeed` undefiniert bleibt (entspricht dem Wert `0`), werden die Zeilen direkt übereinander geschrieben (vgl. Abbildung 73a und b). Die Eigenschaft `linefeed` muss in allen Stilen des Buttons (`style`, `selected style`, `ghosted style`) gesetzt werden.

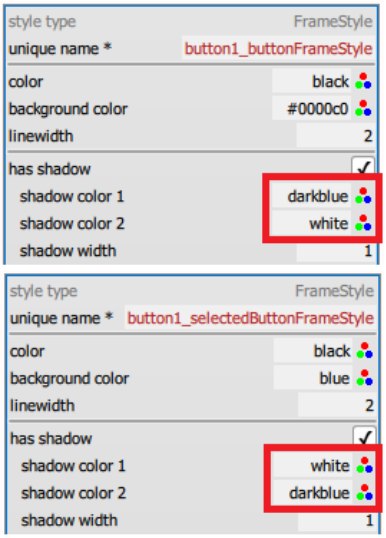


Abbildung 74: Schattenfarben des normalen und selektierten FrameStyles werden vertauscht (Standardeinstellungen)

Tipp: Die Eigenschaften des `Button`- bzw. `FrameStyles` für den gedrückten Zustand sollten so gewählt werden, dass man die entsprechende Zustände auch erkennen kann. (Tipp: Hierzu werden oft die Schattenfarben vertauscht – siehe Abbildung 74.) Die Rahmenbreite (`linewidth`) bzw. Schattenbreite (`shadow width`) des Rahmens können auch unterschiedlich gesetzt werden, um den Eindruck zu erwecken, dass der Button beim Drücken größer oder kleiner wird. Ebenfalls wird empfohlen, die Farben des Stils für den deaktivierten Zustand so zu wählen, dass sie einen „ausgegraute“ Eindruck erwecken. Falls der Button nie deaktiviert wird, können die Stile für diesen Zustand entfallen (`ghosted style` bleibt undefiniert).

Wichtig! Die OK- und Apply-Buttons einer Eingabe-Seite sollen immer einen sinnvollen `ghosted style` haben. Diese Buttons werden automatisch deaktiviert, wenn der Eingabewert ausserhalb des erlaubten Bereichs liegt.

Alle Änderungen in den Stilen können bereits im GUI-Designer ausprobiert werden. Buttons können in den Tabs *Pixel-Exact View* und *Size-Exact View* mit der Maus „gedrückt“ werden. Sogar das Deaktivieren kann ausprobiert werden, indem im Button die Eigenschaft `display ghosted` abgehakt wird.

7.2 Schieberegler (*Slider*-Widget)

Abbildung 75: Schieberegler (Standardeinstellungen)

Ein **Schieberegler** (*Slider*-Widget) erlaubt es, eine numerische **Systemvariable** (`sysvar`) zwischen einem Minimal- und Maximalwert (`minimum value` bzw. `maximum value`) zu variieren (Abbildung 75). Man kann auch einen Anfangswert (`initial value`) direkt vorgeben, oder der Anfangswert (ebenfalls `initial value`) der mit dem Regler verbundenen Systemvariable wird übernommen. Bei einem Schieberegler ohne Verbindung mit einer Systemvariable steht dieser auf dem Minimalwert.

Bemerkung: Die minimalen und maximalen Werten sind Pflichteigenschaften des Schiebereglers. Empfohlen wird, dass sie mit den gleichnamigen Eigenschaften der Systemvariable übereinstimmen.

Der Schieber kann mit dem Finger hin und her gezogen werden. Alternativ dazu kann zu jeder Seite des Schiebers mit dem Finger „geklickt“ werden, um in größeren Abständen zu springen (`step size`). Dieser Abstand wird in der Einheit des Schiebereglers und nicht etwa in Pixel angegeben. Standardmäßig wird eine Sprung um etwa 20% des Schiebebereichs errechnet. Es gibt auch eine Konfiguration, bei der Buttons an den beiden Enden des Schiebereglers betätigt werden können, um in Einzelschritten den Regler weiter zu schieben.

Ein Schieberegler hat die Standardeigenschaften für Position, Größe und **Sichtbarkeit**. Er darf waagerecht oder senkrecht (Checkbox `vertical`) gemalt bzw. betätigt werden.

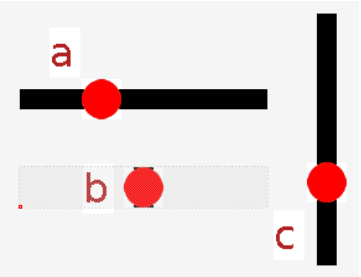


Abbildung 76: 3 Schieberegler:
a = Standard;
b = a + vertical-Flag;
c = b mit Breite und Höhe vertauscht

Bemerkung: eine Schaltung zwischen senkrecht und waagerecht bestimmt nur, ob der Schieberegler von unten nach oben oder von links nach rechts geschoben wird (vgl. Abbildung 76a und b). Die Größe des Schiebereglers ändert sich nicht und muss manuell angepasst werden (Abbildung 76c).

Der Standardschieberegler (Abbildung 76a) hat ein eingebautes Schiebebild (`indicator image`) und eine automatisch erzeugte schwarze Schiebenut (Checkbox `bar mode`). Das Bild darf durch eine eigene **Image** aus der *Images*-Liste ersetzt werden. Ein Hintergrundbild (`background image`) – ebenfalls aus der *Images*-Liste – kann die Schiebenut unterlegen oder gar ersetzen (`bar mode` abwählen). Ferner kann die Schiebenut mit einer Hintergrundfarbe unterlegt werden (Checkbox `use background`). Ansonsten ist der Hintergrund durchsichtig.

Die weitere Eigenschaften der Schiebenut (Dicke und Farben) werden in einem optional angehängten Stil (`style`) vom Typ `SliderStyle` festgelegt, wobei nur die erste vier Eigenschaften relevant sind. Die Dicke der Schiebenut (`bar thickness`), die Farben der Schiebenut unterhalb (`bar color low`) und oberhalb (`bar color high`) des Schiebers, sowie die Hintergrundfarbe (`background color`) – falls aktiviert – dürfen hier angepasst werden.

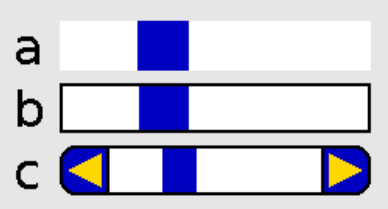


Abbildung 77: Schieberegler ohne Bilder (und ohne Stil):
a) ohne Rahmen,
b) mit Rahmen,
c) mit Rahmen und Buttons

Es ist auch möglich, Schieberegler ohne Bilder zu erzeugen (Abbildung 77). Wenn kein Schiebebild definiert wird (`indicator image` = *undefined*), erscheint zunächst ein Schieberegler als gefülltes Rechteck mit rechteckigem Schieber (Abbildung 77a). Hierzu gibt es einen optionalen Rahmen (`has`

`frame`) mit oder ohne Buttons (`has buttons`). Die Größe des Schiebers (`indicator size`) – wiederum in der Einheit des Schiebereglers – ersetzt die Eigenschaft `step size` für die größeren Sprünge. Diese Art von Schieberegler hat keine eingeblendete Schiebenut.

Die Farbeigenschaften und evtl. **Rahmenstil** des Schiebereglers ohne Bilder werden ebenfalls in dem optional angehängten Stil (`style`) vom Typ `SliderStyle` festgelegt, wobei nur die letzte vier Eigenschaften relevant sind. Die Füllfarbe des Rechtecks (`background color`), die Farbe des Schiebers (`indicator color`) sowie die Farbe der Pfeilen (`arrow color`) (falls diese durch die Benutzung von Buttons vorhanden sind) können hier angepasst werden. Wenn der Schieberegler einen Rahmen hat (erforderlich, um Buttons zu verwenden!), kann auch einen Rahmenstil (`frame style`) noch angehängt und entsprechend modifiziert werden.

Bemerkung: Der Rahmenstil kann gerundete Ecken einschalten, aber diese Eigenschaft greift nur dann, wenn auch Buttons verwendet werden (vgl. Abbildung 77b und c).

7.3 Indizierte Textauswahl (*EnumLabel*-Widget)

Abbildung 78: EnumLabel (Standardeinstellungen)

Die **indizierte Textauswahl** (*EnumLabel*-Widget – Abbildung 78) verwaltet mit minimalem Platzbedarf eine einfache Auswahl aus einer untergeordneten Liste von Einträgen (*labels*), der jeweils ein Text (`text`), ein Bild (`image`) oder beides enthalten darf (Abbildung 79). Der Text darf entweder Klartext sein oder sich auf einem **Text-String** der *Text-Strings*-Liste beziehen. Der aktuelle, in einer numerischen **Systemvariable** (`sysvar`) gespeicherte Index bestimmt, welcher Eintrag aus der Liste gerade erscheint.

Abbildung 79: Liste der Einträge (Standardeinstellungen)

Wichtig! Die Systemvariable (`sysvar`) ist Pflicht! Um die Entwicklung zu erleichtern, wird im GUI-Designer das *EnumLabel*-Widget auch ohne die Verbindung mit einer numerischen Sysvar sichtbar und den Wert auf 0 gesetzt. Aber ohne `sysvar` wird auf dem Display-Modul gar nichts gezeichnet!

Ein typischer Fall für die indizierte Textauswahl wurde schon bei der Erstellung einer **Eingabe-Seite** (siehe Details: Dialog Beispiel) vorgestellt. Das *EnumLabel*-Widget wurde mit einer **Selektor-Sysvar** verbunden und die Einträge im *EnumLabel* gaben an, welche Systemvariable gerade editiert wurde. In diesem Beispiel war dann das *EnumLabel* ein reines **Repräsentations-Widget**.

Ein *EnumLabel* darf aber auch berührt werden und imitiert dann ein oder zwei **Incr/Decr-Buttons** (`touch option`). Bei der

Option `OneButtonBounceMode` wird bei der Berührung immer der nächste Eintrag in der Liste angezeigt und die Liste läuft von einem Ende zum anderen hin und her. Bei der Option `OneButtonLoopMode` läuft die Liste immer nur aufsteigend und springt vom Ende direkt wieder zum Anfang. Bei der Option `TwoButtonsDecrFirst` wirkt die Berührung der linken Hälfte des Widgets absteigend, und die Berührung der rechten Hälfte aufsteigend. Die Option `TwoButtonsIncrFirst` verhält sich umgekehrt.

Tipp: Alternativ dazu, kann das Berührungs-Feature (`touch option`) deaktiviert und durch zwei **Incr/Decr-Buttons** daneben ersetzt werden, den einen aufsteigend, den anderen absteigend, die mit der selben **Systemvariable** verknüpft wird, die auch für das *EnumLabel* zuständig ist.

Tipp: Bei den Optionen mit zwei Buttons, soll das Design des *EnumLabels* so sein, dass man versteht, dass zwei Buttons hinterlegt sind. Zum Beispiel, bei reinen Texteinträgen kann der Text durchsichtig (`transparent`) gemacht werden damit ein (immer zentriertes) Hintergrundbild (`background image`) mit beispielsweise zwei Pfeilen durch den Text durchscheint.

Tipp: *EnumLabel* verwenden, um die Mehrsprachigkeit zu unterstützen.

Ein gutes Beispiel eines berührbaren *EnumLabels* wären Einträge, die auf Bilder von Staatsflaggen verweisen, die die unterstützten Sprachen der Anwendung wiedergeben. Das *EnumLabel* wäre mit der reservierten Systemvariable `SYSVAR_CURRENT_LANGUAGE` verbunden, um das Umschalten der Oberfläche in eine andere Sprache zu unterstützen (siehe Mehrsprachigkeit).

Ein *EnumLabel* hat die Standardeigenschaften für Position, Größe und **Sichtbarkeit**. Die Hintergrundfarbe des Rechtecks (`background color`) gilt allgemein, darf aber in den einzelnen Einträgen durch eine andere Farbe (ebenfalls `background color`) ersetzt werden. Falls die Einträge auf Bilder (**Images**) verweisen, dürfen diese Bilder linksbündig (`AlignLeft`), rechtsbündig (`AlignRight`), oder zentriert (`AlignCenter` = Standardwert) positioniert werden (`image alignment`). Falls ein Image auch eine **Alphafarbe** (`alpha color`) hat, wird diese Farbe automatisch durchsichtig, d.h. man sieht anstelle der Farbe die aktuelle Hintergrundfarbe.

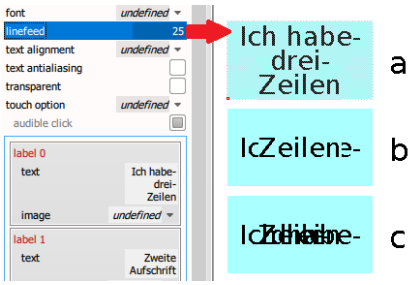


Abbildung 80: EnumLabels mit mehrzeiligen Texten:
a) korrekt! (mit linefeed),
b) wie (a), aber ohne linefeed,
c) wie (b), aber transparent

Die restliche Eigenschaften werden dann wirksam, wenn die Einträge Texte enthalten. Hierzu zählen die übliche **Texteigenschaften** `font`, `text color`, und `text antialiasing`. Der **Font** und die Textfarbe können aber in den einzelnen Einträgen (`font` bzw. `color`) nach Bedarf ersetzt werden. Texte können zudem linksbündig (`AlignLeft`), rechtsbündig (`AlignRight`), oder zentriert (`AlignCenter` = Standardwert) positioniert werden (`text alignment`). Es darf ferner bestimmt werden, ob der Text mit der Hintergrundfarbe unterlegt werden soll oder nicht (Checkbox `transparent`). Dies zeigt nur dann Wirkung, wenn der Text entweder die Grenzen des Widgets sprengt oder ein im gleichen Eintrag enthaltenes Bild überschreibt, da die Hintergrundfarbe das Rechteck bereits ausfüllt (siehe auch Abbildung 80c).

Wie bei **Buttons**, können mehrzeilige Texte verwendet werden. Mehrzeilige Texte benutzen keinen automatischen Zeilenumbruch, sondern müssen an allen Umbruchstellen die Buchstabenkette „\n“ (einen Backslash und die Buchstabe ‚n‘) enthalten, die in einer echten Zeilenumbruch bei der Darstellung umgewandelt wird. Hierzu legt die Eigenschaft `linefeed` den Zeilenabstand – d.h. die Anzahl in Pixel zwischen den Grundlinien der Zeilen – fest.

Wichtig! Bei mehrzeiligen Texten (und nur dann) ist die Eigenschaft `linefeed` zwingend (Abbildung 80a)! Wenn `linefeed` undefiniert bleibt (entspricht dem Wert `0`), werden die Zeilen direkt übereinander geschrieben (Abbildung 80b und c).

Tipp: Wie bekommt ein **EnumLabel** eine Umrahmung?

Tipp: Das *EnumLabel* hat keinen eigenen Rahmen. Wer eine Umrahmung des *EnumLabels* wünscht, kann einfach ein separates **Rechteck** (*Rectangle*-Widget) oder einen **freistehenden Rahmen** (*Frame*-Widget) erzeugen. Bitte beachten: Im *Widgets*-Tab muss dieser Rahmung unbedingt vor das *EnumLabel*-Widget verschoben werden, damit die Elemente in der richtigen Reihenfolge gezeichnet werden.

7.4 Auswahlliste (*SelectionList*-Widget)

Abbildung 81: Auswahlliste, Item 2 ausgewählt (Standardeinstellungen)

Eine **Auswahlliste** (*SelectionList*-Widget) verwaltet eine Liste von Einträgen (*items*), wobei meist nur eine Untermenge (`visible lines`) der gesamten Liste sichtbar ist (Abbildung 81).

Jeder Eintrag darf ein linksbündiges Bild (**Image**) aus der *Images*-Liste ([`selected`] `image`), einen linksbündigen Text ([`selected`] `left label`) und einen rechtsbündige Text ([`selected`] `right label`) enthalten (es sind auch alle drei möglich). Der Text kann entweder Klartext sein oder sich auf einem **Text-String** der *Text-Strings*-Liste beziehen. In der Regel bestimmt der aktuelle, in einer numerischen **Systemvariable** (`sysvar`) gespeicherte Index, welcher Eintrag aus der Liste gerade ausgewählt ist.

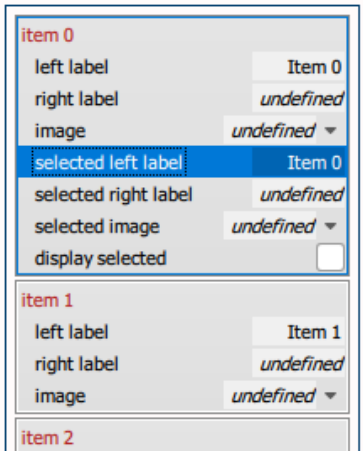


Abbildung 82: Erste Einträge der Auswahlliste (Standardeinstellungen)

Wichtig! Für jedes gewünschte Element (`image` oder `label`) ist auch die ausgewählte (`selected`) Variante des Elements anzupassen, sonst sieht der Eintrag im ausgewählten Zustand unerwartet anders oder gar leer aus (Abbildung 82)!

Mehrere Checkboxen entscheiden, wie die Auswahlliste verwendet wird. Die Liste kann entweder als ein reines **Repräsentations-Widget** erscheinen oder berührbar sein (`touch control`). Ersteres ist nur von informativer Natur und

muss durch die Anwendung verwaltet und nach Bedarf gescrollt werden. Letzteres erlaubt das Scrollen der Liste durch das Wischen mit dem Finger (wobei die Wischrichtung durch `invert wipe direction` einstellbar ist) und/oder durch den optionalen Schieberegler (`slider`), der einen Rahmen (`has frame`) mit oder ohne Buttons (`has buttons`) haben darf. Das „Klicken“ eines Eintrags mit dem Finger bewirkt die einfache oder evtl. auch mehrfache Auswahl (`multi select` – siehe Bemerkung) durch den Anwender, mit oder ohne hörbaren Ton (`audible click`). Eine Auswahl durch den Anwender wirkt sich unmittelbar auf den aktuellen Wert der damit gebundenen Systemvariable aus.

Bemerkung: Die mehrfache Auswahl (`multi select`) und die Möglichkeit, einzelne Einträge vorab auszuwählen (*Item*-Eigenschaft `display selected`) sind Sonderfunktionen, die nur sichtbar werden, wenn die Auswahlliste (noch) nicht mit einer Systemvariable gebunden ist. Die Verwaltung durch eine Systemvariable erlaubt nur eine einfache Auswahl. Hierzu entscheidet der Simulationswert (`simulation value`) oder Anfangswert (`initial value`) der Systemvariable, mit welchem ausgewählten Eintrag die Auswahlliste zuerst erscheint (zur Not mit dem ersten Eintrag).

Es gibt die mehrfache Auswahl, aber nur als Sonderfunktion.

Es gibt noch die Checkbox `alternate row coloring`. Wenn diese Checkbox abgehakt ist, werden die Einträge in alternierenden Hintergrundfarben dargestellt. Die aktuelle Hintergrundfarbe wird bei jeder zweiten Eintrag etwas dunkler oder heller – basierend auf der ursprünglichen Helligkeit der Hintergrundfarbe – dargestellt.

Wo die Auswahlliste von anderen Widgets abweicht, ist in der Bestimmung der Größe des Widgets. Es gibt hier (wie sonst) eine feste Breite, aber die Höhe (`total height`) ist nicht editierbar und wird stattdessen aus der obengenannten Anzahl der gleichzeitig sichtbaren Einträge (`visible lines`) und der Höhe eines einzigen Eintrags (`line height`) intern errechnet. Beide sind Pflichtangaben.

Bemerkung: Die evtl. verwendete Bilder werden nicht skaliert. Daher sollten sie auch bei der Wahl der Zeilenhöhe (`line height`) berücksichtigt werden.

Die letzten Eigenschaften sind die Stile. Eine Auswahlliste hat einen Hauptstil vom Typ `SelectionListStyle` (`style`) sowie drei Stile für die Einträge vom Typ `SelectionListItemStyle` für den normalen Zustand (`item style`), den ausgewählten Zustand (`selected item style`) und den fokussierten Zustand (`focused item style`). Der Hauptstil hat ebenfalls einen untergeordneten Stil vom Typ `SliderStyle`, der wiederum einen untergeordneten

Rahmenstil besitzt.

Tipp! Stilgruppe *Selection List Style* verwenden, um die Stile anzupassen.

Hier ist es (wie bei **Buttons**) überaus ratsam, eine **Stilgruppe** *Selection List Style Group* aus dem Tab *New* (Abschnitt *Style Groups*) anzulegen, wobei gleich alle sechs Stile auf einmal angelegt und miteinander verbunden werden. Eine solche Stilgruppe kann dann gleich für mehrere oder gar alle Auswahllisten verwendet werden, um ein gleichartiges Aussehen zu erreichen und spätere Stil-Änderungen zu erleichtern.

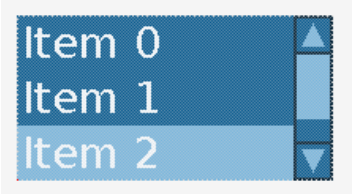


Abbildung 83: Auswahlliste, deren Schieberegler mit einem Rahmen und Buttons dargestellt wird (Standardfarben)

Der Hauptstil (`style`) gibt an, wie weit der Text vom linken oder rechten Rand bzw. vom Bild eingerückt wird (`frame gap`). Er bestimmt zudem den Stil (`slider style`) und die Breite (`slider width`) des Schiebereglers, falls aktiviert. Der Schieberegler wird immer in der Variante ohne Bilder dargestellt, d.h. es sind in seinem Stil nur die untere vier Eigenschaften von Bedeutung (`background color`, `indicator color`, `arrow color`, sowie einen angehängten Rahmenstil (`frame style`). Die Stileigenschaften `frame style` bzw. `arrow color` greifen nur dann, wenn die Widget-Eigenschaften `has frame` bzw. `has buttons` aktiviert worden sind (Abbildung 83). Mehr dazu im Abschnitt Schieberegler (Slider-Widget).

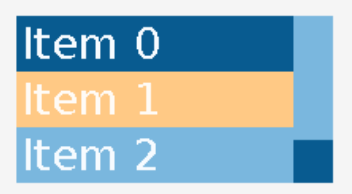


Abbildung 84: Die Maus (der Finger) bewegt sich gerade über Item 1 (im fokussierten Zustand mit den Standardfarben dargestellt)

Die drei Eintragsstile legen die übliche **Texteigenschaften** der verschiedenen Zustände fest. Hier gibt es neben den normalen und ausgewählten Zuständen auch den sogenannten fokussierten Zustand. Letzterer trifft zu, wenn der Finger auf den Einträgen der Auswahlliste hin und her bewegt wird aber man sich noch nicht für einen Eintrag entschieden hat. Der Zustand ist nur vorübergehend aktiv und kann in den Tabs *Pixel Exact View* und *Size Exact View* durch das Ziehen der Maus (Drag) ausprobiert werden (Abbildung 84). Wenn dieses Verhalten nicht erwünscht wird, kann es durch den Widget-Eigenschaft `ignore focused item style` unterdrückt werden.

7.5 Tastatur (*TouchKeyboard*-Widget)

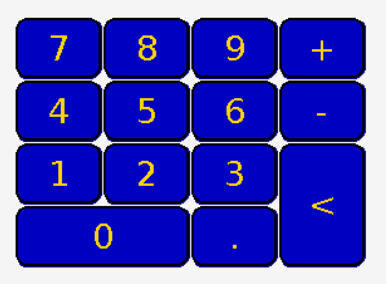


Abbildung 85: Tastatur (Standardeinstellungen)

Eine berührbare **Tastatur** (*TouchKeyboard*-Widget) verwaltet eine Liste von Tasten (*keys*), bei der jede Taste wie ein vereinfachter **Button** konfiguriert wird (Abbildung 85). Eine einzelne Taste hat einen zusätzlichen Tastentyp (`key type`) sowie eine oder mehrere Erscheinungsformen (*appearances*), die automatisch ausgetauscht werden, wenn eine entsprechende Taste wie Shift, Ctrl, Alt oder Capslock gedrückt wird (im folgenden *Modifier*-Taste genannt).

Die verschiedenen Tastentypen (`key types`) decken die typische Funktionalität einer normalen Tastatur ab. Folgende Optionen werden unterstützt:

- `Normal` – der Typ für die meisten Tasten.
- `ShiftModifier` (Umschalttaste) – bis zum nächsten Tastendruck schalten alle Tasten auf ihre Erscheinungsform vom Typ `ShiftModifier` um. Tasten ohne diese Erscheinungsform werden leer dargestellt.
- `CapsLockModifier` (Feststellttaste) – alle Tasten schalten nach Möglichkeit auf eine der priorisierten Erscheinungsformen um. Vorrang hat die Erscheinungsform vom Typ `CapsLockModifier`, dann `ShiftModifier`, dann `Normal`. Die Tasten bleiben in diesem Form, bis wieder eine Modifier-Taste gedrückt wird. Tasten, die keine der obengenannten Erscheinungsformen haben, werden leer dargestellt.
- `AltModifier` (Alt-Taste) – bis zum nächsten Tastendruck schalten alle Tasten auf ihre Erscheinungsform vom Typ `AltModifier` um. Tasten ohne diese Erscheinungsform werden leer dargestellt.
- `CtrlModifier` (Steuerungstaste) – bis zum nächsten Tastendruck schalten alle Tasten auf ihre Erscheinungsform vom Typ `CtrlModifier` um. Tasten ohne diese Erscheinungsform werden leer dargestellt.
- `Enter` (Eingabetaste) – schließt die Eingabe erfolgreich ab; verhält sich auf einer **Eingabe-Seite** wie ein OK-Button.
- `Esc` (Escape-Taste) – bricht die Eingabe ab; verhält sich auf einer **Eingabe-Seite** wie ein Cancel-Button.
- `Delete` (Löschtaste) – löscht die Buchstabe, wo der Cursor sich gerade befindet.
- `Backspace` (Rücktaste) – löscht die Buchstabe direkt vor dem Cursorposition.
- `CursorLeft` – bewegt den Cursor eine Buchstabe nach links
- `CursorRight` – bewegt den Cursor eine Buchstabe nach rechts
- `CursorUp` – nur der Vollständigkeit halber, da die damit verbundene **Eingabezeile** nur einzeilig sein darf.
- `CursorDown` – nur der Vollständigkeit halber, da die damit verbundene **Eingabezeile** nur einzeilig sein darf.
- `ClearScreen` – löscht das gesamte Eingabefeld.

Wie bei den meisten berührbaren Tastaturen, werden die Modifier-Tasten (`*Modifier`) nicht verwendet, um Tastaturkombinationen zu ermöglichen. Stattdessen schalten diese Tasten die Tastatur auf einen Alternativsatz von Buchstaben, z.B. Großbuchstaben, Ziffern, Sonderzeichen, Umlaute, usw., um die Tastatur kompakter zu halten. Diese Alternativbuchstaben werden in den Erscheinungsformen der Tasten abgelegt.

Ein *TouchKeyboard*-Widget hat die Standardeigenschaften für Position, Größe und **Sichtbarkeit**. Die Tastatur kann auch mit einem Bild (`image`), einem Rahmen (`has frame`) oder beidem

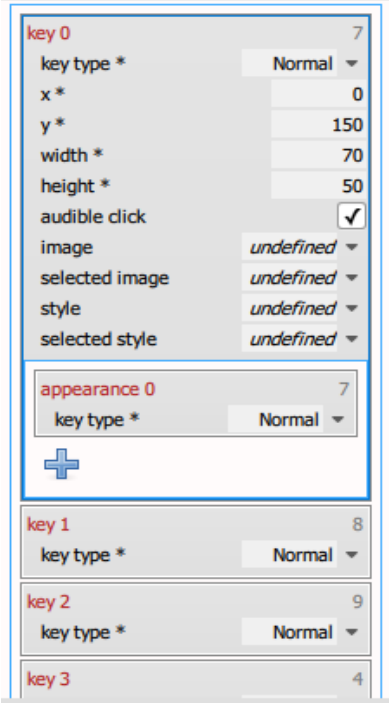


Abbildung 86: Anfang der Tastenliste (Standardeinstellungen)

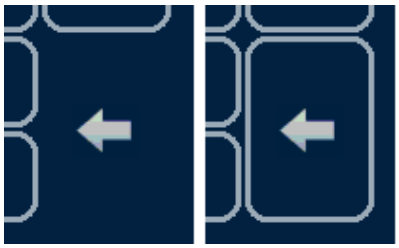


Abbildung 87: Tastaturtaste mit kleines Bild, links ohne und rechts mit Rahmen

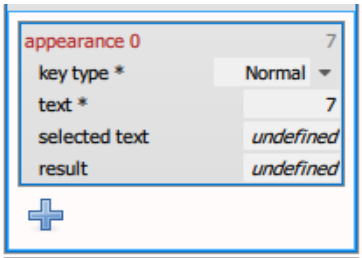


Abbildung 88: Die Erscheinungsformen für die erste Taste (Standardeinstellungen)

hinterlegt werden. Der Rahmen kann einen eigenen Stil (`frame style`) anhängen. Es gibt ebenfalls allgemeine Stile vom Typ `ButtonStyle` für den normalen (`key style`) und gedrückten (`selected key style`) Zustand der Tasten, jeweils mit einem untergeordneten Stil vom Typ `FrameStyle`.

Hier ist es (wie bei **Buttons**) überaus ratsam, eine **Stilgruppe** *Touch Keyboard Style Group* aus dem Tab *New* (Abschnitt *Style Groups*) anzulegen, wobei gleich alle fünf Stile auf einmal angelegt und miteinander verbunden werden.

Als letzte Eigenschaft gibt es eine Liste aller Tasten (*keys*). Die Tasten verhalten sich ähnlich wie Buttons. Jede Taste (Abbildung 86) hat ebenfalls die Standardeigenschaften für Position und Größe, mit dem wichtigen Unterschied, dass die Position auf den Nullpunkt der Tastatur bezogen wird (und nicht auf der Nullpunkt des LCD-Displays). Eine Taste kann beim Berühren einen hörbaren Ton von sich geben (`audible click`). Sie erbt die Tastenstile der Tastatur, darf sie aber durch eigene Stile ersetzen (`style` bzw. `selected style`).

Eine Taste kann mit Bildern ([`selected`] `image`) hinterlegt werden. In diesem Fall erscheint eine zusätzliche Checkbox, um das Zeichnen des Rahmens zu bestimmen (`has frame`). In der Regel wird der Rahmen vom Bild überdeckt und daher nicht gebraucht. Aber im Gegensatz zu Buttons werden die Dimensionen der Taste nicht entsprechend der Größe des Bildes angepasst. Ein Bild darf insbesondere wesentlich kleiner als die Taste ausfallen. In diesem Fall wird ein Rahmen evtl. wünschenswert (Abbildung 87).

Schließlich verwaltet die Taste eine Liste von Erscheinungsformen (*appearances* – Abbildung 88). Jede Erscheinungsform hat ebenfalls einen Tastentyp (`key type`), wobei diese Tastentyp die Erscheinungsform mit einer Modifier-Taste verbindet. D.h. konkret: hier werden nur `Normal`, `ShiftModifier`, `CapsLockModifier`, `AltModifier` und `CtrlModifier` erlaubt. Eine Erscheinungsform gibt an, welcher Text (`text`) auf der Taste erscheinen soll, wenn die entsprechende (oder gar keine) Modifier-Taste aktiviert worden ist. Während des Berührens der Taste erscheint stattdessen der ausgewählte Text (`selected text`). Nach dem Loslassen der Taste wird das Ergebnis (`result`) an die **Eingabezeile** weitergereicht. Normalerweise sind alle drei Werte gleich. Daher dürfen `selected text` und `result` undefiniert bleiben; sie erben automatisch den Wert der Eigenschaft `text`.

Bemerkung: Die Sondertasten (`Delete`, `ShiftModifier`, usw.) brauchen in ihre Erscheinungsformen keinen Ergebnistext, da der Tastentyp selbst das Verhalten der Taste vollständig bestimmt.

8. Repräsentations-Widgets (*Dynamic Visuals*)

Es gibt mehrere Möglichkeiten, die Daten in den **System-variablen** auf dem LCM-Modul darzustellen. Die **berührbaren Widgets** wurden bereits erklärt. Hier geht es nun um Widgets, die Werte darstellen aber keine Touch-Funktionalität haben, d.h. die **Repräsentations-Widgets**, die sich im Tab *New* in den Abschnitten *Text*, *Diagram* und *Additional Dynamic Visuals* befinden. Es gibt hierzu fünf Arten von Text-Repräsentationen (der Text, die Zahl, das Datum, die Uhrzeit und die 7-Segment-Zahl), ein Diagramm-Container samt Unterkomponenten, eine Eingabezeile, eine Balkenanzeige und ein Animiertes Bild.

Diese **Widget-Typen** werden fast immer mit einer oder mehreren **Systemvariablen** verbunden, damit die Darstellung mit jeder Änderung der Systemvariable dynamisch aktualisiert wird.

8.1 Text-Repräsentationen

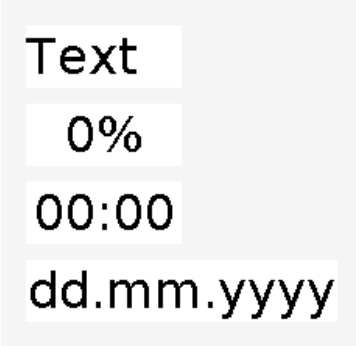


Abbildung 89: Text-, Number-, Time-, und Date-Widgets (v.o. mit Standardeinstellungen)

Es gibt mehrere Möglichkeiten, variable Texte darzustellen (Abbildung 89). Sie stehen im Tab *New* im Abschnitt *Text*. Allgemeine Zeichenketten werden als Text (*Text*-Widget) dargestellt, entweder in Verbindung mit einer `String`-**Sysvar** oder als statischen Text. Die Zahl (*Number*-Widget) oder die 7-Segment-Zahl (*SevenSegmentDisplay*-Widget) erlaubt die transformierte Darstellung einer `Number`- oder `Float`-Systemvariable. Schließlich gibt es spezialisierte Widgets für die Uhrzeit und das Datum (*Time*- und *Date*-Widgets).

Ausnahme! Die 7-Segment-Zahl verwendet eigene zifferbezogenen Eigenschaften anstatt der normalen Texteigenschaften. Sie werden in ihr Abschnitt gesondert beschreiben.

Alle **Text-Repräsentationen** haben die Standardeigenschaften für Position, Größe und **Sichtbarkeit**. Bis auf die 7-Segment-Zahl verwenden alle die üblichen **Texteigenschaften**, angefangen von `font`, `text color`, `background color` und `text antialiasing`. Sie dürfen senkrecht dargestellt werden (`text vertical`). Sie können linksbündig (`AlignLeft`), rechtsbündig (`AlignRight`), oder zentriert (`AlignCenter`) positioniert werden (`text alignment`), wobei hier *Text*-Widgets `AlignLeft`, die andere Widgets aber `AlignCenter` als Standardwert verwenden. Es darf ferner bestimmt werden, ob der Text mit der Hintergrundfarbe unterlegt werden soll oder nicht (`transparent`). Dies kann wichtig sein, wenn die Ober- oder Unterlänge des Fonts etwas größer als erwartet ausfällt und andere Widgets oder den Hintergrund unerwünscht überdeckt. Andererseits kann eine Hintergrundfarbe den Text besser lesbar machen.

Die weitere Eigenschaften sind vom **Widget-Typ** abhängig, wie in den weiteren Abschnitten beschrieben.

8.1.1 Text (*Text*-Widget)

Abbildung 90: Text-Widget (Standardeinstellungen)

Ein **Text** (*Text*-Widget) wird standardmäßig statisch dargestellt (Abbildung 90). Er wirkt dann wie andere **statische Widgets**. Neben den allgemeinen Eigenschaften, gibt die Eigenschaft `text` den Textinhalt vor, und darf entweder Klartext sein oder sich auf einen **Text-String** der *Text-Strings*-Liste beziehen. Durch die Verwendung von Text-Strings wird auch die **Mehrsprachigkeit** unterstützt, trotz der statischen Darstellung.

Tipp! Wenn es um eine Liste fester Meldungen handelt, kommt auch das EnumLabel-Widget in Betracht. Es erlaubt sowohl den variablen Inhalt als auch die Verwendung von Text-Strings (d.h. die Mehrsprachigkeit).

Es ist aber auch möglich mit dem *Text*-Widget einen dynamischen Textinhalt darzustellen, der durch eine `String`-**Sysvar** bestimmt wird. Wenn die Eigenschaft `sysvar` auf einer `String`-Sysvar zeigt, verschwindet die Eigenschaft `text`. Der Textinhalt kommt jetzt von der **Systemvariable**, d.h. die Anwendung muss selbst festlegen, was dargestellt wird. Diese Methode ist für abwechselnden Textinhalt gedacht, z.B. zur Laufzeit zusammengesetzten Texte. Für die Mehrsprachigkeit wird aber die Verwendung von Text-Strings bevorzugt.

Bei der Verwendung von Text-Strings oder eine String-Sysvar kann es dazu kommen, dass der Text nicht mehr im Widget hinein passt. Im Normalfall überragt dann der Text die Ränder des Widgets. Die Eigenschaft `text clipping` unterbindet das Überragen des Textes und trimmt ihn passend.

Ob statischer oder variabler Textinhalt, darf die Textfarbe ebenfalls durch den **Controller** verändert werden. Die Eigenschaft `text color sysvar` enthält in diesem Fall den Namen einer `Color`-Sysvar. Die Eigenschaft `text color` wird solange verwendet, bis die `text color sysvar` mit Werten vom Controller geliefert wird (oder selbst einen Standardwert hat).

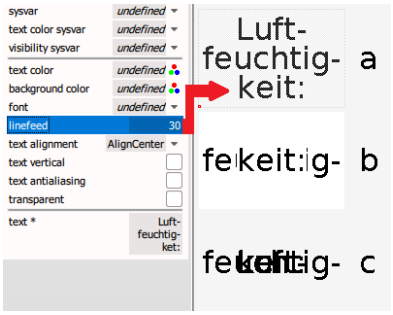


Abbildung 91: Text-Widgets mit mehrzeiligen Texten: a) korrekt! (mit linefeed), b) wie (a) aber ohne linefeed, c) wie (b) aber transparent

Schließlich darf das *Text*-Widget, wie bei **Buttons**, mehrzeilige Texte darstellen. Mehrzeilige Texte benutzen keinen automatischen Zeilenumbruch, sondern müssen an allen Umbruchstellen die Buchstabenkette „\n“ (einen Backslash und die Buchstabe ‚n‘) enthalten, die in einer echten Zeilenumbruch bei der Darstellung umgewandelt wird. Hierzu legt die Eigenschaft `linefeed` den Zeilenabstand – d.h. die Anzahl in Pixel zwischen den Grundlinien der Zeilen – fest.

Wichtig! Bei mehrzeiligen Texten (und nur dann) ist die Eigenschaft `linefeed` zwingend (Abbildung 91a)! Wenn `linefeed` undefiniert bleibt (entspricht dem Wert `0`), werden die Zeilen direkt übereinander geschrieben (Abbildung 91b und c).

8.1.2 Zahl (*Number*-Widget)

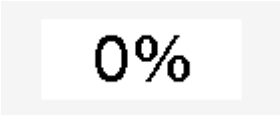


Abbildung 92: Number-Widget (Standardeinstellungen)

Eine **Zahl** (*Number*-Widget) erlaubt die (evtl. transformierte) Darstellung einer `Number`- oder `Float`-**Sysvar** (`sysvar`) samt Einheit (`unit`), wie `cm` oder `%` (Abbildung 92). Es verwendet neben der allgemeinen Eigenschaften auch spezielle Eigenschaften, zur besseren Darstellung eines numerischen Wertes.

Wichtig! Die Systemvariable (`sysvar`) ist Pflicht! Um die Entwicklung zu erleichtern, wird im GUI-Designer das *Number*-Widget auch ohne die Verbindung mit einer numerischen Sysvar sichtbar und den Wert auf 0 gesetzt. Aber ohne `sysvar` wird auf dem Display-Modul gar nichts gezeichnet!

Tipp! Wenn das Widget mit einer Float-Sysvar verbunden wird, darf ebenfalls mittels `slope` und `offset` den Wert für die Darstellung transformiert werden.

Obwohl `Number`-Sysvars immer ganzzahlig sind, darf ein *Number*-Widget in beliebig abgeleiteten Einheiten dargestellt werden. Hierzu gibt es die Eigenschaften `decimal digits` (Anzahl der Stellen hinter dem Komma) und `slope` (Steigung). Möchte der Benutzer anstatt eines Punktes ein Komma als Trennzeichen verwendet, so kann dies über die Eigenschaft `decimal separator character` erreicht werden. Man darf natürlich auch einen `offset` (Y-Achsenabschnitt) eingeben, um einen linearen Zusammenhang abzubilden. Mehr hierzu unter Details: Eingeschaften einer Dialog-Sysvar.

Wichtig: Der `offset` ist immer in Benutzereinheiten (`unit`) und nicht in Sysvar-Einheiten einzugeben. D.h. konkret, dass der `offset`, wie der `slope`, Bruchteile enthalten darf.

Wie bei einem **Text** darf die Textfarbe einer Zahl durch eine `Color`-Sysvar (`text color sysvar`) geregelt werden. Hiermit übernimmt die Anwendung die Farbgebung. Es gibt aber den einfacheren Fall, wobei die Nummer in einer Signalfarbe gezeichnet wird, wenn eine Grenze über- bzw. unterschritten wird. Diese Aufgabe übernimmt das Widget durch die `highlight`-Eigenschaften. Die Signalfarbe wird mit `highlight color` (Standardwert = `rot`) festgelegt. Dieser Wert ersetzt die Textfarbe, wenn der numerischen Wert zwischen `highlight low limit` und `highlight high limit` (Endpunkte mitgezählt) liegt. Die Farbe ändert sich automatisch im gegebenen Bereich, ohne dass die Anwendung sich darum kümmern muss.

Wichtig! Beim `highlight low` bzw. `high limit` handelt es sich um Sysvar-Einheiten – d.h. die Werte vor der Umwandlung durch `slope` und `offset` – und nicht um Benutzereinheiten (`unit`)! Beide Eigenschaften dürfen aber trotzdem Bruchteile enthalten, da das Widget evtl. mit einem `Float`-Sysvar verbunden ist.

Wichtig! Die Signalfarbe hat oberste Priorität! Wenn die

Anwendung die Textfarbe mittels einer Systemvariable (`text color sysvar`) setzt, wird diese Farbe im Bereich der Signalfarbe ignoriert und stattdessen die Signalfarbe verwendet!

8.1.3 Uhrzeit und Datum (*Time*- und *Date*-Widgets)

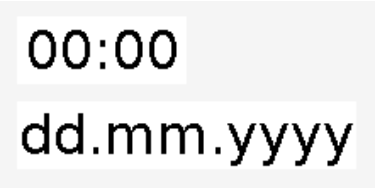


Abbildung 93: Time- und Date-Widgets (Standardeinstellungen)

Es gibt zwei spezielle Widgets um die **Uhrzeit** und das **Datum** auf dem LCD-Modul darzustellen (Abbildung 93). Sie bestehen jeweils aus den allgemeinen Eigenschaften zusammen mit einigen Checkbox-Eigenschaften und einen Trennzeichen, um die Darstellung individuell anzupassen. Sie werden jeweils mit mehreren **Systemvariablen** (**Sysvars**) verbunden.

Die **Uhrzeit** (*Time*-Widget) stellt standardmäßig Stunden und Minuten dar. Es gibt aber die `show`-Checkboxen (`show hours`, `show minutes`, `show seconds`), um stattdessen eine ausgewählte Kombination von Stunden, Minuten und Sekunden darzustellen. Wer keine 24-Stunden- sondern lieber 2-mal-12-Stunden-Zählung verwenden will, darf auch den Zusatz `am` bzw. `pm` dazuschalten (`show am/pm`). Schließlich darf auch das Trennzeichen zwischen den Teilen (`time separator character`, Standardwert = Doppelpunkt) bestimmt werden.

Das *Time*-Widget wird mit bis zum drei verschiedenen `Number`-Sysvars verbunden, jeweils für die Stunden (`hours sysvar`), Minuten (`minutes sysvar`) und Sekunden (`seconds sysvar`). Hier ist zu empfehlen, in der jeweiligen Systemvariable entsprechende Minimum- und Maximumwerte einzugeben, z.B. Stunden dürfen zwischen 0 und 23 laufen.

Wichtig! Die Sysvar für Stunden (`hours sysvar`) ist Pflicht, auch wenn die Stunden nicht dargestellt werden! Um die Entwicklung zu erleichtern, wird im GUI-Designer das *Time*-Widget auch ohne diese Sysvar sichtbar und die Stunden mit `00` gefüllt. Aber ohne `hours sysvar` wird auf dem Display-Modul gar nichts gezeichnet!

Die Anwendung ist für das Aktualisieren der Stunden, Minuten und Sekunden verantwortlich. Im Display-Modul fängt die Zeit mit den Anfangswerten (`initial value`) der Systemvariablen an, falls vorhanden, ansonsten mit Nullen. Es wird nicht automatisch im Display bzw. im Widget weitergezählt.

Wichtig! Die Anwendung soll immer 24-Stunden-Zählung verwenden! Falls `show am/pm` aktiviert wird, werden alle Zeiten ab Mittag automatisch umgewandelt (Beispiel: wenn die Systemvariable für Stunden und Minuten auf `19:30` stehen, wird automatisch `7:30 pm` angezeigt).

Das **Datum** (*Date*-Widget) wird ebenfalls mit drei verschiedenen `Number`-Systvars verbunden, jeweils für den Tag (`day sysvar`), den Monat (`month sysvar`) und das Jahr (`year sysvar`). Hier ist wiederum zu empfehlen, in der jeweiligen Systemvariable entsprechende Minimum- und Maximumwerte einzugeben, z.B. der Monat darf zwischen 1 und 12 laufen.

Wichtig! Es wird das aktuelle Datum erst richtig gezeigt, wenn alle drei Verbindungen hergestellt und die Systemvariablen mit gültigen (Standard-)Werten gefüllt werden. Aber die Systemvariable für den Tag (`day sysvar`) ist Pflicht! Um die Entwicklung zu erleichtern, wird im GUI-Designer das *Date*-Widget auch ohne diese Sysvar sichtbar, allerdings nur als Formatmuster, wie z.B. `dd.mm.yyyy`. Aber ohne `day sysvar` wird auf dem Display-Modul gar nichts gezeichnet!

Tipp! Das aktuell gewählte Format wird sichtbar, wenn das Datum ohne Ziffern dargestellt wird, etwa dd für den Tag, mm für den Monat, yyyy für das Jahr.

Die Anwendung ist für das Aktualisieren des Tages, Monats und Jahres verantwortlich. Im Display-Modul fängt das Datum mit den Anfangswerten (`initial value`) der verbundenen Systemvariablen an, falls vorhanden, ansonsten mit dem aktuellen Format, z.B. `dd.mm.yyyy`. (Bemerkung: Bei fehlenden Sysvars können auch mal Elementen des Datums 01.12.1970 in der Ausgabe vorkommen.)

Abbildung 94: Das typische US-Datumsformat ist auch möglich

Es gibt zwei Checkboxen um ein paar gängige Ländernormen abzudecken (Abbildung 94). Der Monat darf vor dem Tag dargestellt werden (`show month before day`) oder das Jahr darf mit nur 2 Ziffern ausgegeben werden (`show year as 2 digits`). Beide Optionen sind standardmäßig abgewählt. Weiter darf das Trennzeichen zwischen den Teilen (`date separator character`, Standardwert = Punkt) bestimmt werden.

8.1.4 7-Segment-Zahl (*SevenSegmentDisplay*-Widget)

Abbildung 95: SevenSegmentDisplay-Widget (Standardeinstellungen)

Eine **7-Segment-Zahl** (*SevenSegmentDisplay*-Widget) wird fast genauso definiert wie eine Zahl (*Number*-Widget). Sie weicht hauptsächlich in der Darstellung und durch einige Einschränkungen ab.

Die Darstellung besteht aus nur 7-Segment-Ziffern und evtl. einen Punkt als Trennzeichen. Andere Trennzeichen werden nicht erlaubt. Daher entfällt die Eigenschaft `decimal separator character`. Eine 7-Segment-Zahl kann auch keine Einheiten darstellen, was die Eigenschaft `unit` ebenfalls überflüssig macht. Es soll bei Bedarf einen separaten Text (*Text*-Widget) für die Einheit verwendet werden.

Ziffern verwenden standardmäßig folgende Pixelwerte: Ziffergröße = 40x80 (BxH), Segment-Halbbreite = 3, Segment-Abstand = 2.

Die übliche **Texteigenschaften** werden weitgehend durch zifferbezogene Eigenschaften ersetzt. Es wird die Zifferbreite bzw. -höhe (`digit width` and `digit height`) in Pixel sowie die Anzahl der Ziffer (`number of digits`) festgelegt. Die Halbbreite eines einzigen Segments (`segment half width`) und sein Abstand zum nächsten Segment (`segment gap`) – ebenfalls in Pixel – legen fest, wie dick und nahe zueinander die Segmente gemalt werden sollen. Aus diesen Eigenschaften werden die genaue Höhe und Breite des Widgets berechnet. Deswegen ist die Größe des Widgets nicht editierbar. Das Widget wird immer waagerecht dargestellt.

Es bleibt lediglich die Textfarbe, die genauso wie bei der Zahl reguliert wird, ob statisch (`text color`) oder über einen `Color`-**Sysvar** (`text color sysvar`), sowie nach Bedarf durch eine Signalfarbe ersetzt (die `highlight`-Eigenschaften). Die Eigenschaft `background color` entfällt, da die Hintergrundfarbe immer von der Hintergrundfarbe der Seite entnommen wird.

8.2 Diagramm-Container (*Diagram*-Widget)

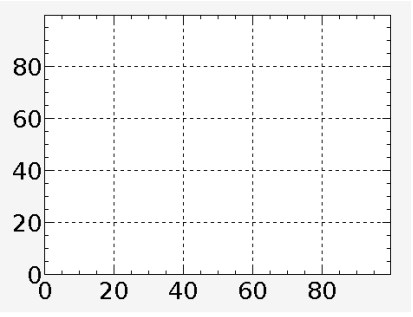


Abbildung 96: Ein zunächst leerer Diagramm-Container (Standardeinstellungen)

Ein **Diagramm-Container** (*Diagram*-Widget – Abbildung 96) verwaltet zwei **Achsen** (`XAxis` und `YAxis`) sowie eine untergeordnete Liste von bis zu acht **Plots**, die entweder Kurven oder Streudiagramme darstellen dürfen. Sowohl die Achsen als auch die Plots haben eigene **Stile** (**Stiltyp** `AxisStyle` bzw. `PlotStyle`), damit die Darstellung genau angepasst bzw. verschiedene Plots unterschieden werden können.

Ein Diagramm-Container hat die Standardeigenschaften für Position, Größe und **Sichtbarkeit**, außerdem noch eine Hintergrundfarbe (`background color`), die dieses Rechteck ausfüllt. Die Achsenbeschriftung selbst wird aber außerhalb des Diagrammbereichs gezeichnet und ist immer transparent (Abbildung 96).

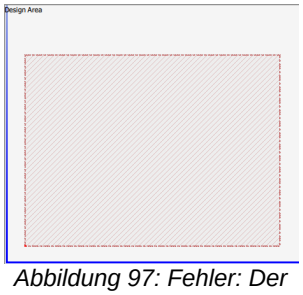


Abbildung 97: Fehler: Der Platz für die Achsen ist nicht ausreichend

Wichtig! Die Größe der X- bzw. Y-Achse werden hinzuaddiert, um zu entscheiden, ob der Diagramm-Container auf dem Display-Modul korrekt gezeichnet werden kann! Wenn der Diagramm-Container nur als fehlerhaftes Rechteck dargestellt wird (Abbildung 97), dann muss das Widget zunächst weiter vom Rand entfernt werden.

Alle anderen Eigenschaften stecken in den beiden Achsen und den Plots, sowie in deren Stilen. Die benötigten Komponenten dazu stehen im Tab *New* im Abschnitt *Diagram*.

8.2.1 Diagrammachsen (`XAxis` und `YAxis`)

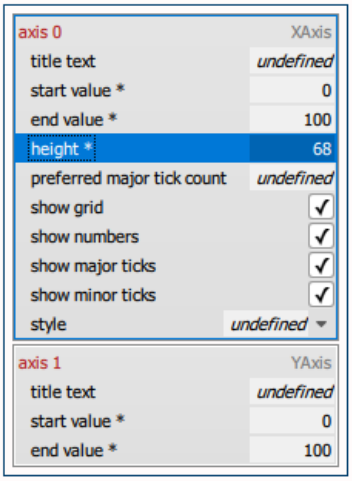


Abbildung 98: Eigenschaften der X-Achse (Standardeinstellungen)

Ausnahmefall! Die Eigenschaften `start value` und `end value` dürfen Fließkommazahlen enthalten!

Die beiden **Achsen** eines **Diagramm-Containers** werden im Tab *Properties* wie eine untergeordnete Liste dargestellt. Diese Liste hat aber immer genau zwei Einträge: `axis 0` vom Typ `XAxis` für die X-Achse und `axis 1` vom Typ `YAxis` für die Y-Achse. Bis auf die Höhe (`height`) der X-Achse bzw. die Breite (`width`) der Y-Achse, werden die Eigenschaften beider Achsen zunächst mit Standardwerten gefüllt (Abbildung 98).

Wichtig und erforderlich sind die Eigenschaften für den Startwert (`start value`) und Endwert (`end value`) des sichtbaren Achsenbereichs. Sowohl hier als auch im Datensatz eines Plots werden **Weltkoordinaten** verwendet, d.h. konkret: Fließkommazahlen werden hier ausnahmsweise erlaubt! Die beiden Eigenschaften bestimmen die Skalierung der Achsen (Länge), wobei dieser Bereich dann unterhalb (X-Achse) bzw. links (Y-Achse) des Diagrammbereichs genau passt. Auf der Y-Achse bleibt die dadurch entstandene Nummerierung unverändert. Auf der X-Achse können die Werte aber fortschreiten, falls das *Diagram*-Widget später durch den Controller gescrollt wird. Die ursprünglich errechnete Länge der X-Achse wird beim Scrollen beibehalten.

Wichtig! Der Diagramm-Container kann nur durch den GUI-Interpreter-Befehl `GR_SCROLL_DIAGRAM_GUIREP_HORIZONTAL` vom Controller gescrollt werden!

Tipp! Die Achsengröße soll so groß wie nötig und so klein wie möglich eingestellt werden, damit z.B. der Diagramm-Container näher zum Rand des Display-Moduls positioniert werden kann.

Die Höhe (`height`) der X-Achse bzw. Breite (`width`) der Y-Achse definiert die **Achsengröße**. Die Standardgrößen lassen genug Platz für die Standardnummerierung und einen Achsentitel (`title text`, dieser ist zunächst leer). Der Achsentitel wird mittig am unteren (X-Achse) bzw. linken (Y-Achse) Rand des Achsenbereichs gezeichnet. Eine Vergrößerung bzw. Verkleinerung der Achsengröße verschiebt ebenfalls den Titel.

Die bevorzugte Anzahl von Hauptunterteilungen (`preferred major tick count`, Standardwert = 8) darf hier ebenfalls festgelegt werden. Darüber hinaus gibt es Checkboxen für die Merkmale, die gezeichnet werden sollen:

- `show grid`: Automatisches Raster im Diagrammbereich entlang der Hauptunterteilstriche.
- `show numbers`: Automatische Nummerierung entlang der Hauptunterteilstriche.
- `show major ticks`: Automatische Hauptunterteilstriche.
- `Show minor ticks`: Automatische Nebenunterteilstriche.

Tipp! Bei Bedarf kann im Tab *Styles* der Achsenstil dupliziert und der anderen Achse zugewiesen werden, um die Achsen unterschiedlich zu gestalten.

Als letzte Eigenschaft steht der Name eines **Stils** (`style`). Der Stil muss vom Type `AxisStyle` sein. Im Abschnitt *Diagram* des Tabs *New* kann ein *Axis Style* auf den gewünschten Diagramm-Container gezogen werden. Hiermit werden beide(!) Achsen mit diesem Stil automatisch verbunden.

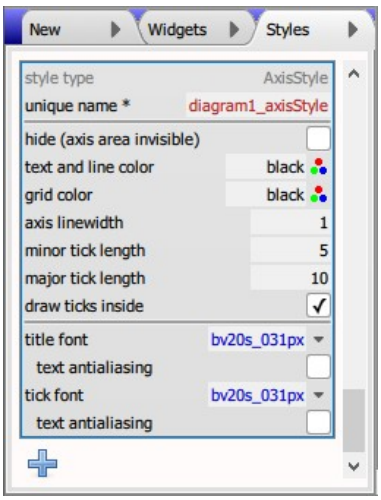


Abbildung 99: AxisStyle (Standardeinstellungen)

Im Stil wird festgelegt, wie die in der Achse ausgewählten Merkmale zu zeichnen sind (Abbildung 99). Als erstes, darf der Achsenbereich komplett ausgeblendet werden, wenn es nur auf eine qualitativen Darstellung ankommt (Checkbox `hide (axis area invisible)`). Sichtbar wird dann nur noch das Raster, falls dieses Merkmal ausgewählt ist. In diesem Fall wird die Achsengröße komplett ignoriert, und der Diagramm-Container darf bis zum Rand des Display-Moduls verschoben werden.

Ferner darf eine Farbe für die Beschriftung und die Linien (`text and line color`) und eine evtl. abweichende Farbe für das Raster (`grid color`) bestimmt werden. Die Linienbreite (`axis linewidth`) betrifft alle Linien der Achse: den Rand um den Diagrammbereich, die Unterteilungsstriche und das Raster. Die Länge (in Pixel) der Neben- bzw. Hauptunterteilungsstriche dürfen angepasst werden (`minor` bzw. `major tick length`), sowie ob die Unterteilungen außerhalb oder innerhalb der Diagrammumrandung gezeichnet werden sollen (Checkbox `draw ticks inside`).

Als **Texteigenschaften** werden lediglich **Font** und **Antialiasing** unterstützt, wobei der Titel (`title font`) anders gestaltet werden darf als die Nummerierung (`tick font`).

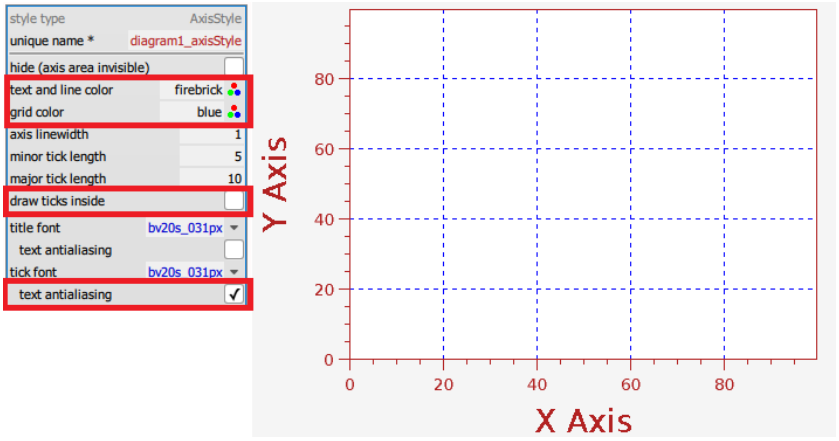


Abbildung 100: Diagramm-Container mit verändertem Achsenstil (AxisStyle)

Das Beispiel in Abbildung 100 addiert einen Titel (`title text`) zu jeder Achse und verändert ansonsten nur den Stil wie im Bild angezeigt, um Farbe, Unterteilungen und Text etwas anders zu gestalten (vgl. Abbildung 96).

8.2.2 Diagramm-Plots

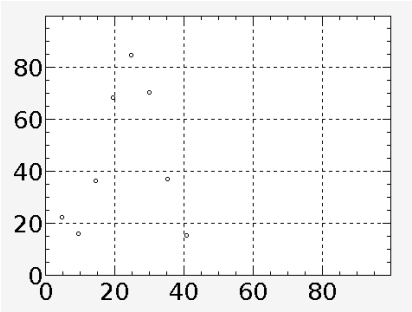


Abbildung 101: Diagramm-Container mit einem Plot aus Pseudodaten (Standardeinstellungen)

Ein **Diagramm-Container** kann bis zu acht **Plots** gleichzeitig verwalten. Sie werden im Tab *Properties* als eine zunächst leere untergeordnete Liste hinter den beiden **Achsen** dargestellt. Anders als die anderen untergeordneten Listen, muss jeder der maximal acht Plots auch einen **eindeutigen Namen** (`unique name`) haben.

Die Eigenschaften eines Plots beschreiben, wie der zugehörige Datensatz mit einer festgelegten Größe und gefüllt mit **Weltkoordinaten** auf dem Diagrammbereich dargestellt werden soll. Es wird hier ebenfalls definiert, wie neue Datenpunkte dem Datensatz hinzugefügt bzw. dargestellt werden.

Pseudodaten gelten nur als Hilfestellung und werden nicht auf das Display-Modul übertragen!

Wichtig! Um das Gestalten eines Plots einfacher zu machen werden im GUI-Designer zunächst für jeden Plot Pseudodaten erzeugt und dargestellt (Abbildung 101). Die Pseudodaten gelten nur als Hilfestellung und werden nicht auf das Display-Modul übertragen. Die echten Daten müssen vom Controller kommen.

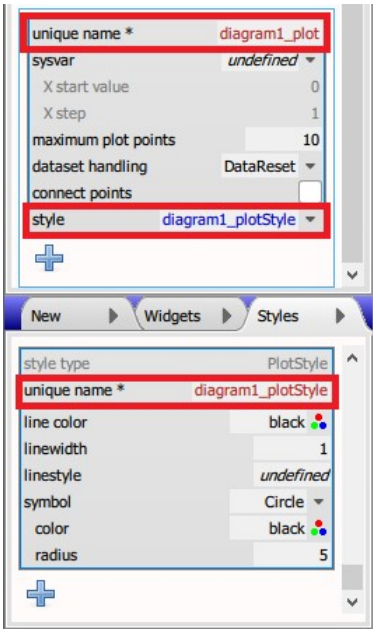


Abbildung 102: Plot plus Style (Tab New) auf diagram1 ziehen, um einen Plot (in diagram1) mit eigenem Stil zu erzeugen (Standardeinstellungen).

Jeder Plot darf anders gestaltet werden, damit die Plots voneinander zu unterscheiden sind. Es wird selten sein, dass nur die Standardeinstellungen verwendet werden oder das mehrere Plots den gleichen **Stil** haben. Um den dazu benötigten Vorgang zu erleichtern kann im Abschnitt *Diagram* des Tabs *New* ein *Plot plus Style* auf den gewünschten Diagramm-Container gezogen werden. Hiermit wird sowohl ein Plot mit einem eindeutigen Namen erzeugt und dem Diagramm-Container zugeordnet als auch ein neuer Stil vom **Stiltyp** `PlotStyle` dem Tab *Styles* hinzugefügt und dem Plot (Plot Eigenschaft `style`) automatisch zugewiesen (Abbildung 102).

Ungewöhnlich für die Einträge einer untergeordneten Liste, hat der Plot-Eintrag sowohl einen eindeutigen Namen (`unique name`) als auch die Eigenschaft `sysvar`. Ist der Plot mit einer `Float`-Sysvar verbunden, dann besteht er aus gleichmäßigen X-Schritten. Der Controller schickt lediglich Y-Werte in Weltkoordinaten an die genannte Systemvariable und der X-Wert wird automatisch erhöht. Hierzu wird zusätzlich die X-Position des ersten Datenpunktes (`x start value`) und der X-Abstand der Punkten (`X step`) benötigt, beide ebenfalls in Weltkoordinaten. Wenn X – und somit der Plot – endlos weiter laufen soll, dann gibt es ein Datensatz-Modus (`dataset handling`) namens `DataContinue`. Achtung: Die Daten verschwinden am Ende des dargestellten Diagrammbereichs, es sei denn, der Bereich wird durch den GUI-Interpreter-Befehl `GR_SCROLL_DIAGRAM_GUIREP_HORIZONTAL` regelmäßig weiter gescrollt.

`GR_SCROLL_DIAGRAM_GUIREP_HORIZONTAL`

Wichtig! Der Diagramm-Container kann nur durch den GUI-Interpreter-Befehl `GR_SCROLL_DIAGRAM_GUIREP_–`

`HORIZONTAL` vom Controller gescrollt werden!

`GR_PLOT_ADD_XY_DATA`
`GR_PLOT_CLEAR_DATA`

Wenn der Plot mit keiner Systemvariable verbunden ist, dann wird er zu einem XY-Plot. D.h. die Daten werden als Punkte (X und Y, beide in Weltkoordinaten) vom Controller über den GUI-Interpreter-Befehl `GR_PLOT_ADD_XY_DATA` verschickt. Hierzu wird die **Objekt-ID** des Plots verwendet, die aus dem eindeutigen Namen erzeugt wurde. Die Objekt-ID wird auch im Befehl `GR_PLOT_CLEAR_DATA` verwendet, der die bisherigen Punkte im Datensatz verwirft.

Die maximale Anzahl von Punkten legt die Größe des Datensatzes fest!

Wie schon erwähnt erfolgt das Hinzufügen von Datenpunkten durch setzen einer Sysvar bzw. durch den Befehl `GR_PLOT_ADD_XY_DATA`. Wird die maximale Anzahl von Punkten (`maximum plot points`) erreicht, wird der Datensatz wieder von vorne gefüllt und die bisherigen Daten gehen (nach und nach) verloren.

Es gibt aber mehrere Möglichkeiten, wie der Sprung zum Beginn ders Datensatzes interpretiert werden kann. Die gewünschte Wirkung wird durch den Datensatz-Modus (`dataset handling`) festgelegt:

- DataReset: Alle bisherigen Daten werden verworfen, d.h. der Plot wird dann auch zuerst gelöscht.
- DataOverwrite: Die bisherigen Daten werden nach und nach überschrieben. Es entsteht so etwas wie beim Herzmonitor (das Diagrammbereich wird nicht gescrollt).
- DataContinue: Die Daten werden zwar von vorne überschrieben, aber es wird hinter dem letzten Datenpunkt weitergeschrieben. Wird nun das Diagramm dabei gescrollt, dann kann der Plot ewig weiterlaufen. Beim erneuten Zeichnen des Diagrammbereichs (z.B. nach dem Umschalten einer Page) sieht man dann möglicherweise die alten Daten nicht mehr unbedingt, da diese ja nicht mehr im Datensatz stecken.

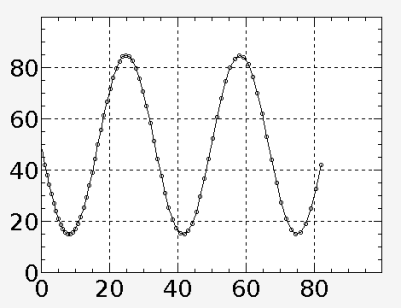


Abbildung 103: Plot mit maximal 80 verbundenen Punkte (Pseudodaten)

Standardmäßig werden die einzelne Punkte nicht durch Linien verbunden (Checkbox `connect points` nicht abgehakt), aber bei Kurven ist dies oft wünschenswert (Abbildung 103). Die Eigenschaften dieser Linien werden im Stil festgelegt.

Als letzte Eigenschaft steht der Name eines **Stils** (`style`). Der Stil muss vom Type `PlotStyle` sein. Im Abschnitt *Diagram* des Tabs *New* wurde hoffentlich über *Plot plus Style* ein Stil zum Plot bereits angelegt. Wenn aber *Plot plus Style* nicht über einen bestehenden Diagramm-Container gezogen, sondern anderswo abgelegt wurde, entsteht ein alleinstehender `PlotStyle` (ohne Plot-Zuweisung).

Im Stil können verschiedene Elemente des Plots angepasst werden. Wenn die Datenpunkte durch Linien verbunden werden sollen, dürfen Farbe (`line color`), Pixelbreite (`linewidth`)

und Linienstil (`linestyle`) der Linien angepasst werden. Eine genaue Beschreibung dieser Eigenschaften steht unter Rechteck (*Rectangle*). Das untere Beispiel in Abbildung 104 verwendet `linewidth = 2`, `line color = #ff55ff` und `symbol = NoSymbol`.

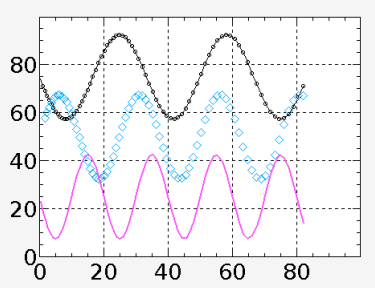


Abbildung 104: Drei Pseudoplots mit jeweils maximal 80 Punkten aber verschiedenen Stilen

Die Datenpunkte selbst dürfen durch ein (oder auch kein) Symbol (`symbol`) hervorgehoben werden. Die möglichen Symbole sind: kein Symbol (`NoSymbol`), Kreis (`Circle` – Standardwert), Quadrat (`Square`), Raute (`Diamond`), Stern (`Star`), Dreieck (`Triangle`), und Pixel. Das Symbol hat eine eigene Farbe (`color`) und einen Radius (`radius`). Letzteres entspricht der halbe Symbolgröße in Pixel, damit der Datenpunkt in der Mitte des Symbols liegt. Bei `Pixel` findet `radius` und bei `NoSymbol` beide Eigenschaften keine Verwendung. Das mittlere Beispiel in Abbildung 104 verwendet `symbol = Diamond` mit `color = #00aaff` und `radius = 10`. Die Datenpunkte sind hier nicht durch Linien verbunden.

8.3 Eingabezeile (*InputLine*-Widget)

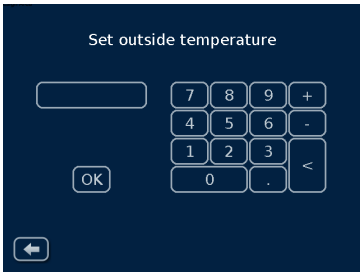


*Abbildung 105: **Dialog** mit Eingabezeile und numerischer Tastatur (Beispiel)*

Eine **Eingabezeile** (*InputLine*-Widget) ist meist nur in Zusammenhang mit einer berührbare **Tastatur** (*TouchKeyboard*-Widget) auf einer **Eingabe-Seite** (**Dialog**) sinnvoll (Abbildung 105). Hier werden die gedrückten Tasten der Tastatur automatisch auf der damit verbundenen Eingabezeile angezeigt. Es wäre natürlich auch möglich, die Eingabezeile komplett über den Controller zu verwalten, z.B. in Zusammenhang mit einer externen Tastatur. Aber im Normalfall wird erwartet, dass alle Eingaben über einen Dialog gemacht werden.

Abbildung 106: Standard-Eingabezeile mit silbernem Rahmen

Bemerkung: Die Eingabezeile hat ausnahmsweise keine eigene `sysvar`-Eigenschaft. Stattdessen wird sie über die Dialog-Eigenschaften mit der Tastatur verbunden und ihr Inhalt wird ebenfalls durch den Dialog an die richtige Systemvariable übergeben. Dies erlaubt, dass eine einzige Eingabezeile bzw. -seite für beliebig viele Systemvariablen verwendet werden kann.

Die Eingabezeile (Abbildung 106) hat die Standardeigenschaften für Position, Größe und **Sichtbarkeit**. Sie darf mit oder ohne Rahmen gezeichnet werden (`has frame`). Es darf auch einen Cursortyp (`cursor type`) bestimmt werden. Der Standardwert ist keinen Cursor zu verwenden (`NoCursor`). Alternativ kann der Cursor als Rechteck (`SolidCursor`), Unterstrich (`UnderlineCursor`) oder senkrechte Linie (`VerticalCursor`) erscheinen. Alle weiteren Eigenschaften samt Rahmengestaltung werden im angehängten Stil (`style`) definiert.

Tipp! Der Cursor kann auf dem LCD-Modul mit dem Finger positioniert werden!

Der Stil muss vom Typ `InputlineStyle` sein. Er enthält die allgemeinen **Texteigenschaften**. Die Farbe des Cursors (`cursor color`) wird ebenfalls hier bestimmt. Außerdem gibt es einen untergeordneten **Rahmenstil** (`frame style`) sowie einen Abstand (in Pixel) links und rechts zwischen dem Rahmen und dem Text (`frame gap`). Wird die Eingabezeile ohne Rahmen gezeichnet (s.o.), so spielen diese beide Eigenschaften keine Rolle.

Abbildung 107: Textfläche (hellblau) einer Eingabezeile mit gerundeten Kanten

Bemerkung: wenn der Rahmen gerundete Kanten hat, wird der Textbereich automatisch so verkleinert, dass er nicht in den gerundeten Bereich fällt (Abbildung 107). In diesem Fall ist `frame gap = 0` sinnvoll.

8.4 Balkenanzeige (*BarIndicator*-Widget)

Eine **Balkenanzeige** (*BarIndicator*-Widget) ist eine der Repräsentationsmöglichkeiten einer numerischen **Systemvariable**. Die Länge des Balkens variiert mit dem aktuellen Wert der Systemvariable.

Eine Balkenanzeige hat die Standardeigenschaften für Position, Größe und **Sichtbarkeit**. Sie darf waagerecht oder senkrecht (Checkbox `vertical`) sowie mit oder ohne Rahmen gezeichnet werden (Checkbox `has frame`).

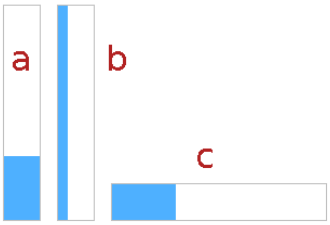


Abbildung 108: 3 Balkenanzeige: a = Standard (vertical); b = a ohne vertical-Flag; c = b mit Breite und Höhe vertauscht

Bemerkung: Eine Umschaltung zwischen senkrecht und waagerecht bestimmt nur, ob der Balkenwert von unten nach oben oder von links nach rechts gezeichnet wird (vgl. Abbildung 108 a und b). Die Größe der Anzeige ändert sich nicht und muss manuell angepasst werden (Abbildung 108c).

Die Eigenschaft `sysvar` enthält den Namen der numerischen Systemvariable. Es müssen auch immer die minimalen und maximalen Werte (`minimum value` und `maximum value`) des Balkens angegeben werden, damit der aktuelle Wert korrekt abgebildet werden kann. Wie bei der Systemvariable, sind diese Werte und alle weitere numerische Werte ganzzahlig.

Bemerkung: Die minimalen und maximalen Werte müssen nicht mit den gleichnamigen Eigenschaften der Systemvariable übereinstimmen. Der abgesteckte Bereich der Balkenanzeige darf kleiner oder größer ausfallen.

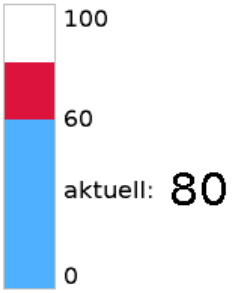


Abbildung 109: Standard Balkenanzeige mit Schwellenwert = 60, Anfangswert = 80

Optional dazu kommt ein Anfangswert (`initial value`), der solange verwendet wird, bis die Systemvariable mit Werten vom Controller beliefert wird (oder selbst einen Standardwert hat).

Es gibt auch ein optionalen Schwellenwert (`threshold value`). Der Schwellenwert gibt einen Grenzwert vor, ab dem der Balken in einer anderen (Signal-)Farbe gezeichnet wird (Abbildung 109).

Gewünscht wird allerdings auch, dass bei Übertreten eines Schwellenwerts der ganze Balken in der Signalfarbe bemalt wird. Vielleicht soll sogar die Farbe je nach aktuellem Wert kontinuierlich angepasst werden. Dies kann so realisiert werden, indem der Controller diese Aufgabe mit Hilfe einer weiteren Systemvariable steuert. Die Eigenschaft `bar color sysvar` enthält in diesem Fall den Namen einer `Color`-Sysvar. Der Controller schickt einen neuen Balkenwert an die numerische Systemvariable zu und ändert ebenfalls die gewünschte Farbe durch die `Color`-Sysvar. Der Balken passt seine Darstellung den beiden Werten an. Es wird empfohlen in diesem Fall die Eigenschaft `threshold value` undefiniert zu lassen.

Alle weiteren Eigenschaften samt Rahmengestaltung werden mit dem angehängten Stil (`style`) definiert. Der Stil muss vom Typ `BarIndicatorStyle` sein. Er definiert einige Farben und den Rahmen (falls `has frame` in der Balkenanzeige abgehakt ist).

Die Hintergrundfarbe des Stils (`background color`) wird auf der „unbenutzten“ innere Fläche der Balkenanzeige verwendet. Diese Farbe muss evtl. mit der Hintergrundfarbe im Rahmen abgestimmt werden, weil sie unterschiedliche Flächen überdecken (s.u.).

Die Eigenschaft `bar color low` bestimmt die Farbe des Balkens unterhalb des Schwellwerts (`threshold value`). Sie wird solange verwendet, bis eine etwaige `bar color sysvar` (s.o.) mit Werten vom Controller geliefert wird (oder selbst einen Standardwert hat).

Die Eigenschaft `bar color high` wird nur dann benutzt, wenn in der Balkenanzeige ein Schwellwert (`threshold value`) definiert ist, und der aktuelle Wert des Balkens darüber liegt. Alles was über diesem Schwellwert liegt, wird in dieser Signalfarbe ausgefüllt. Wenn die Farbgestaltung des Balkens komplett über den Controller laufen soll, soll der `threshold value` undefiniert bleiben. In diesem Fall hat `bar color high` keine Bedeutung.

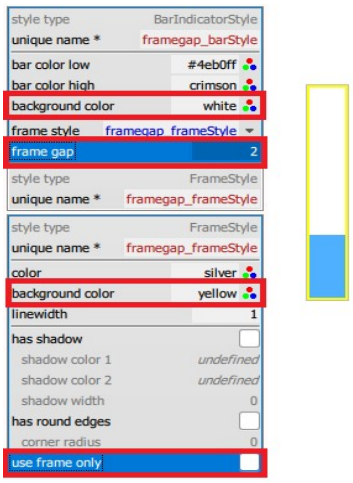


Abbildung 110: Der Abstand (frame gap) wird in der Hintergrundfarbe des Rahmenstils gezeichnet.

Außerdem gibt es einen untergeordneten **Rahmenstil** (`frame style`) sowie einen Abstand (in Pixel) rund um den Rahmen und der inneren Fläche (`frame gap`). Wenn die Balkenanzeige ohne Rahmen gezeichnet wird, spielen diese beiden

Eigenschaften keine Rolle.

Bemerkung: Wenn ein Abstand (`frame gap`) definiert ist, wird der Zwischenraum anders gezeichnet als die „ungenutzte“ innere Fläche des Balkens. Der Zwischenraum beachtet die Rahmenstil-Eigenschaft `use frame only`: wenn abgehakt, wird in diesem Bereich nichts gezeichnet (transparent), ansonsten wird mit der Hintergrundfarbe (`background color`) des Rahmenstils(!) gearbeitet (Abbildung 110).

8.5 Animiertes Bild (*Animation*-Widget)

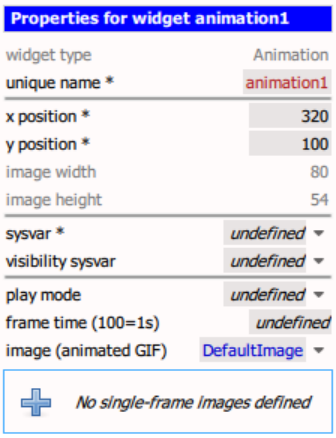


Abbildung 111: Eigenschaften eines animierten Bildes (Standardeinstellungen)

Ein **animiertes Bild** (*Animation*-Widget) erlaubt selbst-laufende Animation auf dem LCD-Display, entweder durch eine animierte GIF-Datei (`image (animated GIF)`) oder eine Liste von Einzelbilder (*single-frame images*). Die Einzelbilder dürfen als Bitmap (`.bmp`), JPEG (`.jpg`) oder GIF (`.gif`) formatiert werden.

Das animiertes Bild wird zunächst mit einem Standardbild (`DefaultImage`) erzeugt (Abbildung 111). Das Bild ist keine animierte GIF-Datei sondern eine einfache Bitmap, nur um etwas darzustellen. Diese Bitmap soll unbedingt ersetzt werden.

Ein animiertes Bild hat die Standardeigenschaften für Position und **Sichtbarkeit**, allerdings wird die Größe automatisch von der animierten GIF-Datei bzw. der ersten Datei in der Liste von Einzelbilder übernommen (`image width`, `image height`) und ist daher nicht editierbar.

Das Widget gilt als dynamisch, weil es durch eine boolesche **Systemvariable** (`sysvar`) gestartet und gestoppt wird.

Tipp! Zum Testen auf dem Display-Modul ohne Controller kann ein „Ein/Aus“ Hold-Button ebenfalls mit der Systemvariable verbunden werden, um die Animation einzuleiten und abzubrechen.

Wichtig! Die Systemvariable (`sysvar`) ist Pflicht! Um die Entwicklung zu erleichtern, wird im GUI-Designer das animierte Bild auch ohne die Verbindung mit einer booleschen Sysvar sichtbar und bei jedem Neumalen des Designs automatisch um einen Rahmen weiter gespult. Aber ohne `sysvar` wird auf dem Display-Modul gar nichts gezeichnet!

Durch die Eigenschaft `play mode` wird entschieden, ob beim Start die Animation nur einmal (`PlayOnce`), oder in eine Schleife (`Loop`) oder hin und her (`Bounce`) läuft. Die Geschwindigkeit des Abspielens wird durch die Rahmendauer (`frame time`) geregelt, wobei die Einheit hier eine Hundertstelsekunde ist. Der Standardwert ist 50, d.h. die Bilder tauschen sich alle Halbsekunde regelmäßig aus.

Wer Einzelbilder (*single-frame images*) verwenden möchte, muss unbedingt den Eigenschaft `image (animated GIF)` löschen (*undefined*), sonst werden die Einzelbilder ignoriert. Die tatsächlichen Bilder werden, wie alle Bilder, erst als **Images** in der *Images*-Liste aufgenommen. Erst dann dürfen sie als Einzelbilder hier angelegt werden. Jeder Eintrag enthält der Name des Images (`image`) sowie die Möglichkeit, dieses Einzelbild mit einem abweichenden Rahmendauer (`frame time`) zu versehen. Hiermit kann die Animation sehr detailliert gesteuert werden.

Wer eine animierte GIF-Datei verwenden möchte, soll wissen, dass die Rahmendauern in der Datei ignoriert werden. Es wird nur die hier angegebene Rahmendauer verwendet; d.h. Abweichungen pro Rahmen werden bei GIF-Dateien nicht unterstützt. Es wird auch immer die Bilder aufeinander gezeichnet, auch wenn in der Datei ein Austausch der Bilder gewünscht wird. Im konkreten Fall sind animierte GIF-Dateien mit einem Transparenzkanal (Alphakanal) für das Display-Modul ungeeignet.

Wichtig! Weder Einzelbilder noch animierte GIF-Dateien sollen einen Alphakanal enthalten, weil sie sich bei der Animation nicht richtig dargestellt werden. Mehr dazu unter Details: Alphafarbe vs. Alphakanal.

9. Statische Widgets (*Static Visuals*)

Die letzte Kategorie von Widgets sind die statische Darstellungen (**statische Widgets**). Bis auf die **Sichtbarkeit** können diese Widgets nicht über **Systemvariablen** verändert werden. Sie befinden sich im Tab *New* im Abschnitt *Static Visuals*.

Hierzu zählt ein Statisches Bild (*Image*-Widget), ein Freistehender Rahmen (*Frame*-Widget), oder die standard Geometrische Formen wie Rechteck (*Rectangle*-Widget), Linie (*Line*-Widget) oder Kreis (*Circle*-Widget).

9.1 Statisches Bild (*Image*-Widget)

Abbildung 112: Statisches Bild (Standardeinstellungen)

Ein **statisches Bild** (*Image*-Widget) legt ein **Image** der *Images*-Liste (`image`) an einer beliebigen Stelle auf dem Display. Das Bild darf als Bitmap (`.bmp`), JPEG (`.jpg`) oder GIF (`.gif`) formatiert werden.

Ein **statisches Bild** hat die Standardeigenschaften für Position und **Sichtbarkeit**, allerdings wird die Größe automatisch vom **Image** übernommen (`image width`, `image height`) und ist daher nicht editierbar.

Das Widget wird zunächst mit einem Standardbild (`DefaultImage`) erzeugt (Abbildung 112), damit das Widget überhaupt sichtbar wird. Das Standardbild soll durch ein **Image** aus der *Images*-Liste (`image`) ausgetauscht werden. Falls das **Image** auch eine **Alphafarbe** bestimmt, wird diese auch automatisch beachtet. Siehe hierzu Details: Alphafarbe.

Wichtig! Die Alphafarbe darf nicht mit dem Alphakanal eines Bildformats verwechselt werden! Insbesondere werden Bitmaps mit einem Alphakanal nicht unterstützt. Wer ein Alphakanal verwenden möchte, soll die Bitmap in einer GIF-Datei verwandeln. Mehr unter Details: Alphafarbe vs. Alphakanal.

9.2 Freistehender Rahmen (*Frame*-Widget)

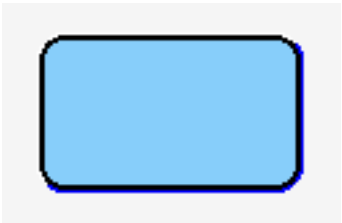


Abbildung 113: Freistehender Rahmen (Standardeinstellungen)

Ein **freistehender Rahmen** ist gestalterisch vielseitiger als ein einfaches **Rechteck**. Er wird verwendet, um z.B. andere Widgets zu umrahmen. Er darf gerundete Ecken und Schattierung haben (Abbildung 113). Die Breiten der Rahmenlinie und der Schattierung dürfen auch einzeln bestimmt werden. Die innere Fläche darf transparent sein oder eine andere Farbe haben als die Rahmenlinie selbst.

In den meisten Fällen wird ein freistehender Rahmen gar nicht nötig, da der Rahmen als angehängter **Rahmenstil** bereits

vorhanden ist. Dies gilt für alle Widgets mit Stilen, z.B. **Buttons**, **Balkenanzeigen** (*BarIndicator*), **Schieberegler** (*Slider*), **Tastaturen** (*TouchKeyboard*), usw.

Es gibt aber Widgets, die keine Stile haben, z.B. die **indizierte Textauswahl** (*EnumLabel*) oder **Zahl** (*Number*-Widget). Hier wird ein passender Rahmen vielleicht auch wünschenswert. Dazu gibt es den freistehenden Rahmen (*Frame*) als eigenes Widget.

Der freistehender Rahmen hat die Standardeigenschaften für Position, Größe und **Sichtbarkeit**. Es bleibt lediglich ein Verweis auf einem Stil (`style`) vom Typ `FrameStyle`, d.h. einem **Rahmenstil**, wo die weitere Eigenschaften – Farbgebung, Liniendicke, usw. – festgehalten werden. Gleich hinter dem Eintrag *Frame* im Tab *New* steht ein Rahmenstil (*Frame Style*) mit den Standardeinstellungen zur Verfügung, die als Startmuster verwendet werden kann. Mehr zu diesen Eigenschaften unter Rahmenstil (FrameStyle).

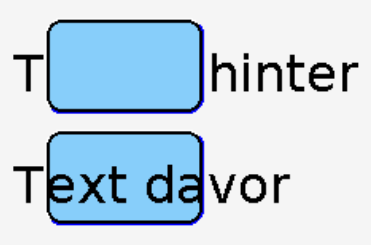


Abbildung 114: Der Rahmen wird (oben) nach und (unten) vor dem Text gezeichnet

Wichtig! Bei freistehenden Rahmen ist zu beachten, dass die Widget-Reihenfolge stimmt. Der Rahmen muss zuerst gezeichnet werden und erst danach das in den Rahmen erscheinenden Widget, wie in Abbildung 114 (unten). Mehr unter Reihenfolge bestimmen (Widgets-Tab).

9.3 Geometrische Formen

Es gibt ein paar **Widget-Typen**, die einfache geometrische Formen darstellen: das **Rechteck** (*Rectangle*-Widget), die **Linie** (*Line*-Widget) und der **Kreis** (*Circle*-Widget).

Alle geometrische Formen haben die Standardeigenschaften für Position, und **Sichtbarkeit**. Das Rechteck hat auch die Standardeigenschaften für Größe. Aber die Größe des Kreises wird durch den Radius (`radius`) bestimmt, und die Linie definiert eine Endposition (`x` bzw. `y end position`), d.h. sie wird lediglich durch zwei Punkte bestimmt.

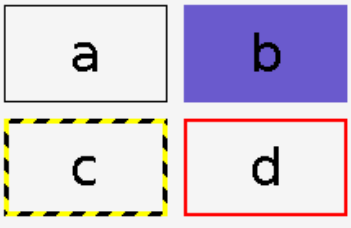


Abbildung 115: Rechteckbeispiele:
a) Standardeinstellungen
b) color = slateblue, solid fill
c) linewidth = 3, linestyle = 255 und background color = yellow
d) linewidth = 2, color = red

9.3.1 Rechteck (*Rectangle*-Widget)

Ein **Rechteck** wird standardmäßig als eine durchgehende, ein Pixel breite, schwarze Linie gezeichnet (Abbildung 115a). Die Farbe (`color`), die Pixelbreite (`linewidth`) und der Linienstil (`linestyle` – s.u.) der Linie können variiert werden. Es darf aber auch mit dem Checkbox `solid fill` das Rechteck in der gegebene Farbe (`color`) komplett ausgemalt werden (Abbildung 115b). In diesem Fall sind die weitere Eigenschaften ohne Belang.

Der Linienstil (`linestyle`) spielt eine besondere Rolle. Der Wert dieser Eigenschaft entspricht einem 16-bit Muster (d.h.

von `0` bis `65535` = `0xffff`), wobei jede Eins mit der Farbe (`color`) und jede Null entweder mit der Hintergrundfarbe (`background color`) oder durchsichtig (`transparent`) dargestellt wird. Im Abbildung 115c wird mit `linestyle = 255` acht Bits in der Farbe schwarz und acht Bits in der Hintergrundfarbe gelb abwechselnd gezeichnet. Wenn `linestyle` undefiniert bleibt, wird die Linie durchgehend (=65535) gezeichnet und die Eigenschaften `background color` und `transparent` sind ohne Belang.

Wichtig! Die Eigenschaft linestyle soll immer als Dezimalzahl angegeben werden! Hexzahlen werden nicht verstanden.

9.3.2 Linie (*Line*-Widget)

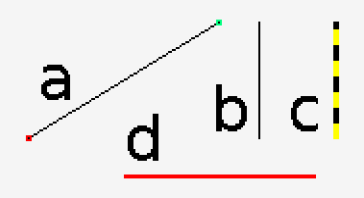


Abbildung 116: Linienbeispiele: a) Standardeinstellungen und selektiert (aber mit Gitterfarbe = springgreen) b) senkrecht (X-Werte gleich) c) wie (b) plus Abbildung 115c d) waagerecht (Y-Werte gleich) plus Abbildung 115d

Als weitere geometrische Form gibt es die **Linie** (*Line*-Widget). Sie wird durch zwei Punkte definiert: als Startpunkt die Position (`x` bzw. `y position`) und als Endpunkt die Endposition (`x` bzw. `y end position`).

Ein *Line*-Widget wird standardmäßig als eine durchgehende, ein Pixel breite, schwarze schräge Linie gezeichnet (Abbildung 116a). Normalerweise werden aber meist waagerechte (beide Y-Werte gleich) oder senkrechte (beide X-Werte gleich) Linien gewünscht (Abbildung 116b-d). Deswegen gibt es auch im Tab *New* Einträge für waagerechte und senkrechte Linien (*Horizontal* bzw. *Vertical Line*). Die Orientierung entspricht lediglich den Anfangszustand. Die Linien unterliegen keinen Einschränkungen und dürfen nachträglich anders orientiert werden.

Der Anfang einer selektierten Linie wird in der **Positionsfarbe**, das Ende in der **Gitterfarbe** markiert (siehe Abbildung 116a). Jedes Ende darf mit der Maus gezogen werden (Drag) um die Position, Länge und Orientierung zu ändern. Wer an einer beliebigen Stelle der Linie zieht, verschiebt stattdessen die ganze Linie (beide Positionen gleichmäßig), was einem Positionswechsel – wie bei anderen Widgets – gleichkommt. Das gleiche passiert bei der Verschiebung von Linien in einer **Merfachauswahl**.

Die anderen Eigenschaften des *Line*-Widgets funktionieren genauso wie beim Rechteck. Die Hauptmerkmale sind Farbe (`color`) und Linienbreite (`linewidth`). Wenn ein Linienstil (`linestyle`) anders als durchgehend (Standardwert `65535` (=`0xffff`)) verwendet wird, spielt auch die Hintergrundfarbe (`background color`) bzw. Durchsichtigkeit (`transparent`) eine Rolle. Es fehlt lediglich das Konzept `solid fill`. Um diese Ähnlichkeit darzustellen, wird in Abbildung 116c die gleiche Werte verwendet wie in Abbildung 115c. Ebenfalls mit Abbildung 116d und Abbildung 115d.

9.3.3 Kreis (*Circle*-Widget)

Abbildung 117: Kreisbeispiele:
a) Standardeinstellungen
b) solid fill, color = slateblue
c) color = red

Als letzte geometrische Form gibt es den einfachen **Kreis** (*Circle*-Widget). Der Kreis wird standardmäßig als eine schwarze Linie dargestellt (Abbildung 117a). Die Linie ist immer durchgehend und ein Pixel breit. Nur die Farbe (`color`) darf variiert werden (Abbildung 117c). Es ist auch möglich, wie bei Rechtecken, den Kreis auszufüllen (`solid fill` – Abbildung 117b.

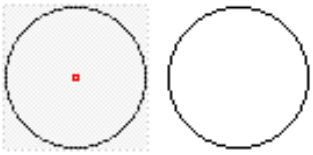


Abbildung 118: Zwei gleiche Kreise, links selektiert mit sichtbarer Position mittig

Außergewöhnlich beim Kreis ist, dass die Position nicht in der unteren linken „Ecke“ sondern in der Mitte des Kreises liegt. Beim selektierten Kreis ist dies als roter Punkt mittig sichtbar (Abbildung 118). Der Kreis hat auch keine Breite (`width`) oder Höhe (`height`) sondern lediglich einen Radius (`radius`). Wer am Rand oder an einer Ecke des selektierten Bereiches der *Design Area* mit der Maus zieht, ändert damit lediglich den Radius.

10. Mehrsprachigkeit

Der GUI-Designer unterstützt **Text-Strings** in mehreren Sprachen sofern diese in der *Text-Strings*-Liste angelegt sind.

Im Menü *File > Document Properties* müssen die verwendeten Sprachen eingestellt werden (Sprachen einstellen (File > Document Properties)).

Alle Texte des Designs werden im Tab *Text-Strings* jeweils mit einem **eindeutige Namen** (`unique name`) und einem Standardwert (`default text`) angelegt (Texte verwalten (Text-Strings-Tab)). Die weiteren sprachabhängige Eigenschaften können vorerst leer bleiben.

Einige **Widgets** haben einen oder mehrere String-Eigenschaften (z.B. `text` oder `unit`). Wird hier statt des Klartextes der Namen eines Eintrags der *Text-Strings*-Liste angegeben, kann eine automatische Sprachumschaltung des Strings erfolgen.

Im Abschnitt Design auf das Display-Modul übertragen wird erwähnt, dass auch eine `csv`-Datei erzeugt wird. Dieser Datei enthält alle Strings in der *Text-Strings*-Liste und deren aktuellen Übersetzungen. Diese `csv`-Datei dient dazu, die Strings in einem Tabellenkalkulationsprogramm zu laden und dann – z.B. durch ein Übersetzungsbüro – übersetzen zu lassen (siehe Übersetzungen vorbereiten). Die dadurch vervollständigte Tabelle wird wieder in eine `csv`-Datei zurückkonvertiert und im Menü *Tools > Load Translations* eingelesen, um die *Text-Strings*-Liste zu aktualisieren (Übersetzungen einlesen (Tools > Load Translations)).

10.1 Sprachen einstellen (*File > Document Properties*)

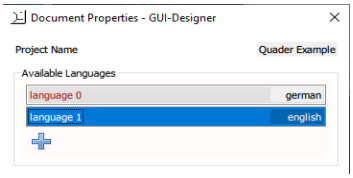


Abbildung 119: Beispiel mit zwei angelegten Sprachen

Der Menüpunkt *File > Document Properties* öffnet das Dialogfenster *Document Properties*. Dort unter *Available Languages* wird eine einfache Liste von verwendeten Sprachen definiert. Links steht der Index der Sprache. Rechts darf der Name der Sprache editiert werden. Dieser Name erscheint dann als neue Eigenschaft bei allen Einträgen der *Text-Strings*-Liste. Die Sprachnamen werden innerhalb dieser Liste automatisch eindeutig gehalten und unterliegen die gleiche syntaktische Einschränkungen wie andere **eindeutige Namen**. Abbildung 119 zeigt ein Beispiel mit zwei angelegten Sprachen.

Wichtig! Die Sprachen müssen zuerst im *File > Document Properties* festgelegt werden, bevor man im Menüpunkt *Tools > Load Translations* die passende Übersetzungen dazu aus einer `csv`-Datei laden kann. Mehr dazu im

Abschnitt Übersetzungen einlesen (Tools > Load Translations).

Wie bei den meisten Listen, kann man mit Drag & Drop die Liste umsortieren, mit dem Plussymbol ( ) einen neuen Eintrag anlegen, oder über das **Kontextmenü** eines Eintrags den Eintrag duplizieren bzw. löschen.

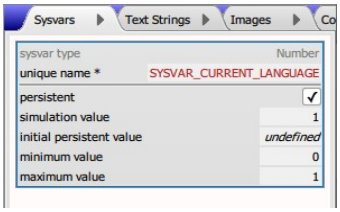


Abbildung 120: Automatisch angelegte reservierte Sysvar

Tipp: Nach einer Umschaltung der Sprache, Reset des LCMs durchführen

Beim Anlegen der erste Sprache erscheint die reservierte **Systemvariable** `SYSVAR_CURRENT_LANGUAGE` automatisch in der Liste der Systemvariablen (*Sysvars*-Tab), wie in Abbildung 120 dargestellt. Diese **Sysvar** wird auch auf die Sprache aktualisiert, die in der Sprachliste ausgewählt wird.

Tipp: Diese Systemvariable kann z.B. mit einem *EnumLabel*-Widget oder mit Set-Buttons gesteuert werden, um die Sprache in der Anwendung umzuschalten. Allerdings muss die Anwendung danach neu gestartet werden (d.h. man muss einen Reset des Display-Moduls durchführen), damit alle Text-Strings in der neugewählten Sprache dargestellt werden.

10.1.1 Details: Reservierte Sysvars für die Mehrsprachigkeit

Es gibt zwei System-**Sysvars**, die vom GUI-Designer für die Mehrsprachigkeit benutzt werden und deren Namen reserviert sind:

- `SYSVAR_NUMBER_OF_LANGUAGES`: Anzahl der Sprachen. Diese Sysvar wird automatisch erzeugt, erscheint aber in der Oberfläche des GUI-Designers nicht. Der Benutzer verändert diese Sysvar nur in Ausnahmefällen. Wenn er diese Sysvar in der Liste der Systemvariablen erzeugt, kann er die Anzahl der verwendeten Sprachen einschränken: Bei drei definierten Sprachen, kann durch das Setzen von `SYSVAR_NUMBER_OF_LANGUAGES = 2` die letzte Sprache deaktiviert werden.
- `SYSVAR_CURRENT_LANGUAGE`: Index der gerade benutzten Übersetzung (startet bei 0). Durch Ändern dieser Sysvar ändert sich auch die angezeigte Sprache. Die Sysvar wird vom GUI-Designer automatisch erzeugt. Das `persistent`-Flag wird gesetzt, damit die aktuelle Sprache „reset-sicher“ gespeichert wird.

10.2 Texte verwalten (*Text-Strings*-Tab)

Der *Text-Strings*-Tab enthält eine Liste von sogenannten **Text-Strings**. Wie bei den meisten Listen, kann man mit Drag & Drop die Liste umsortieren, mit dem Plussymbol ( ) einen neuen Eintrag anlegen, oder über das **Kontextmenü** eines Eintrags den Eintrag duplizieren bzw. löschen.

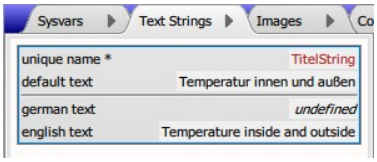


Abbildung 121: Beispiel Text-String

Tipp: Die Hauptsprache darf sich auf den Standardwert verlassen.

Jeder Text-String hat einen **eindeutigen Namen** (`unique name`) und einen Standardwert (`text` bzw. `default text`). Sind im *File* > *Document Properties* Einträge in der Liste der verwendeten Sprachen definiert, steht hier für jede Sprache eine entsprechende Eigenschaft (z.B. `german text`, `english text`), wie im Beispiel links (Abbildung 121).

Tipp: Der Standardwert wird immer dann verwendet, wenn keine passende Übersetzung vorhanden ist. D.h. wenn als Standardwert ein deutscher Text steht, kann die Eigenschaft `german text` undefiniert bleiben. Es wird für die deutsche Sprache einfach der Standardwert übernommen.

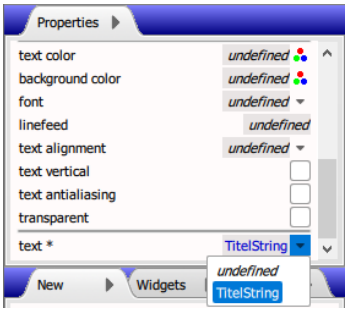


Abbildung 122: Den Text-String einer String-Eigenschaft zuweisen

Einige **Widgets** haben einen oder mehrere Strings als Eigenschaften. Wird hier die Zeichenkette direkt angegeben, dann ändert sich diese nicht mit dem Umschalten der Sprache. Gibt man statt dessen einen Namen aus der *Text-Strings*-Liste an (wie in Abbildung 122), kann eine automatische Sprachumschaltung erfolgen.

Tipp: Das Anlegen eines Text-Strings bringt beim Starten des Display-Moduls einen Geschwindigkeitsvorteil, und es wird weniger Arbeitsspeicher verbraucht.

10.2.1 Details: String-Eigenschaften und die *Text-Strings*-Liste

Ein **Text-String** muss immer zuerst angelegt und danach als Wert einer String-Eigenschaft des Widgets zugewiesen werden. Im Widget auf das ►-Symbol der String-Eigenschaft mit der linken Maustaste klicken, um den Namen des Text-Strings aus der Liste zu wählen. Die blaue Darstellung des Text-String-Namens dient zur Unterscheidung zu normalem Klartext.

Wenn eine String-Eigenschaft einen (blauen) Text-String-Name enthält, ist es ebenfalls möglich mit der rechte Maustaste auf das ►-Symbol zu klicken, um zum Text-String in der *Text-Strings*-Liste zu „springen". Anschließend könnte dort der Textwerte kontrolliert oder angepasst werden.

Bemerkung: Wenn die *Text-Strings*-Liste leer ist, gibt es kein ►-Symbol bei den String-Eigenschaften, da (noch) keine Auswahlmöglichkeiten bestehen. Werden keine Text-Strings im Design verwendet, wirkt das Erscheinungsbild etwas aufgeräumter.

10.3 Übersetzungen vorbereiten

Es ist möglich, Übersetzungen aller Einträge der *Text-Strings*-Liste durch ein Übersetzungsbüro erstellen zu lassen, um sie dann unter dem Menüpunkt *Tools* > *Load Translations* wieder

einzulesen.

1. csv-Datei mit File/Create nnn_image.bin erzeugen.

Die dazu benötigte `csv`-Datei wird durch den Menüpunkt *File > Create <nnn>_image_v?.bin* (oder mit dem Blitzsymbol aus der Symbolleiste) nebenbei erzeugt. Die Datei enthält alle Einträge der *Text-Strings*-Liste mit Spalten für den **eindeutigen Namen** (`strings name`), Standardtext (`default`) sowie für alle bisher definierten Sprachen (s.o. Sprachen einstellen). Die Spalten werden durch Kommas getrennt und jeder String mit Anführungszeichen umrahmt (Abbildung 123).

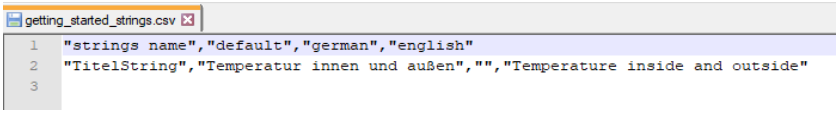


Abbildung 123: Inhalt einer automatisch erzeugten csv-Datei

Achtung! Die Datei verwendet das `Unicode (UTF-8)` Encoding.

2. csv-Datei mit einem Tabellenkalkulations-programm laden.

Die `csv`-Datei soll zunächst mit Hilfe einer Tabellenkalkulation (z.B. Excel, LibreOffice o.ä.) geladen werden. Hierzu ist es wichtig, dass das Komma als Trennzeichen, das Anführungs-zeichen als Zeichenketten-Trenner und vor allem das Zeichen-satz auf `Unicode (UTF-8)` gesetzt wird (Abbildung 124).

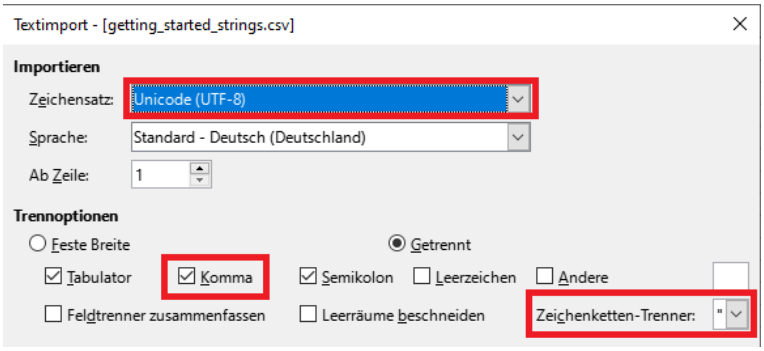


Abbildung 124: Wichtige Einstellungen beim Importieren [Beispiel aus LibreOffice]

3. Daten prüfen und als Tabelle abspeichern.

Im Tabellenkalkulationsprogramm angekommen, sollte jetzt die korrekte Lesbarkeit der geladenen Texte korrekt geprüft werden. Es dürfen keine seltsamen Buchstaben, unerwartete Fragezeichen oder ähnliches enthalten sein. Dies sind Hinweise auf ein falsches Encoding. Es dürfen auch keine Anführungs-zeichen um die Einträge zu sehen sein. Vor allem müssen die Einträge in separaten Spalten auftauchen (Abbildung 125). Wenn alles korrekt ist, kann diese Datei als normales Tabellen-dokument abgespeichert werden (nicht als `csv`-Datei!).

	A	B	C	D	E
1	strings name	default	german	english	
2	TitelString	Temperatur innen und außen		Temperature inside and outside	
3					

Abbildung 125: csv-Datei im Tabellenkalkulationsprogramm geladen

4. Tabelle an ein Übersetzungsbüro weiter geben.

Die erzeugte Tabelle aus dem Tabellenkalkulationsprogramm kann jetzt an eins oder mehrere Übersetzungsbüros zur Vervollständigung weitergegeben werden. Es ist ratsam, eine normale Tabelle (und nicht die `csv`-Datei) weiterzugeben, da es sonst zu Schwierigkeiten mit dem Format oder Encoding kommen kann. Jedes Übersetzungsbüro lädt die Tabelle und übersetzt nur die für sich relevanten Spalten. Die anderen Spalten bleiben leer.

5. Übersetzte Tabelle wieder laden und als csv-Datei mit UTF-8 Encoding abspeichern.

Wenn eine Tabelle mit Übersetzungen wieder eintrifft, muss diese Tabelle wieder in eine `csv`-Datei konvertiert werden. Hierzu wird zunächst die Tabelle in die Tabellenkalkulationsprogramm geladen. Hier kann sicherheitshalber die Korrektheit der Texte geprüft werden. Jetzt muss die Tabelle als `csv`-Datei (Text CSV) im `UTF8`−Encoding gespeichert werden. Empfohlen wird das Komma als Trennzeichen und Anführungszeichen als Text-Delimiter (Abbildung 126). Allerdings können hier nach Bedarf auch andere Symbole verwendet werden.

Tipp: Um ans Export-Dialogfenster zu gelangen, wird es evtl. nötig sein, ein Flag im Dialogfenster *Speichern unter...* anzuwählen (etwa *Filtereinstellungen bearbeiten*).

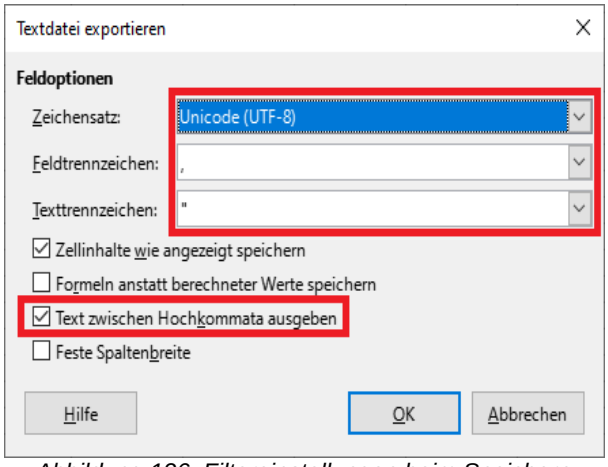


Abbildung 126: Filtereinstellungen beim Speichern unter... (Text CSV) [Beispiel aus LibreOffice]

Jede Sprachspalte kann separat übersetzt und eingelesen werden.

Es ist **nicht** nötig, unterschiedliche Tabellen erst zusammen zu führen, um die Sprachen alle in einer Tabelle zu vereinen. Jede Tabelle kann separat in eine eigenen `csv`-Datei gewandelt und dann einzeln in den GUI-Designer geladen werden. Leere Spalten bzw. Zellen werden beim erneuten Laden der Datei ignoriert. Die Anzahl oder Reihenfolge der Sprachen (d.h. ab der 3. Spalte) ist beliebig. Hier ist die Liste der Sprachen im Menüpunkt *File > Document Properties* maßgebend. Selbst die Anzahl oder Reihenfolge der Zeilen (d.h. ab der 2. Zeile) ist beliebig. **Text-Strings** werden durch die erste Spalte (`strings name`) explizit gesucht und bearbeitet.

Wichtig! Das obengenannte Verhalten führt dazu, dass weder neue Sprachen noch neue Text-Strings allein durch

Wichtig! Dem Design unbekannten Sprachen oder Text-String-Namen in der csv-Datei werden ignoriert!

die `csv`-Datei erzeugt werden dürfen! Bereits existierende Sprachen oder Text-Strings werden auch nicht gelöscht. Die Liste der erlaubten Sprachen ist allein durch den Menüpunkt *File > Document Properties* zu verwalten (Sprachen einstellen). Ebenfalls soll die *Text-Strings*-Liste verwendet werden, um Elemente zu löschen bzw. zu erstellen (Texte verwalten (Text-Strings-Tab)). Die Text-String-Eigenschaft `unique name` muss zu einem Eintrag der Spalte `strings name` in der `csv`-Datei passen, bevor man die `csv`-Datei einliest.

10.4 Übersetzungen einlesen (*Tools > Load Translations*)

Um die übersetzten Texte einer `csv`-Datei wieder einzulesen, gibt es den Menüpunkt *Tools > Load Translations* im GUI-Designer, der dann das Dialogfenster *Load Translations* öffnet.

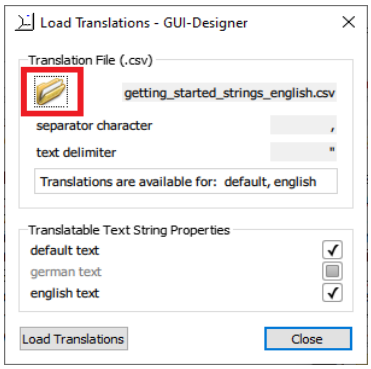


Abbildung 127: Dialogfenster zum Laden der Sprachübersetzungen – Ordnersymbol klicken, um die csv-Datei zu bestimmen

Im oberen Rahmen wird der Namen und die Eigenschaften der `csv`-Datei bestimmt. Lädt man eine gültige `csv`-Datei (auf die *Ordnersymbol* klicken, um die Datei auszusuchen), werden zusätzliche Infos über die enthaltenen Sprachen eingeblendet (siehe Abbildung 127). Leere Spalten erscheinen hier nicht (z.B. die Spalte für die Sprache, die sowieso als `default text` schon vordefiniert ist). Falls die `csv`-Datei ein anderes Trennzeichen (`separator character`) bzw. Zeichenketten-Trenner (`text delimiter`) verwendet, kann dies hier angepasst werden.

Achtung! Die `csv`-Datei muss mit dem `UTF8`-Encoding erzeugt werden. Hierzu bietet der GUI-Designer keine Konvertierung an.

Im unteren Rahmen steht mindestens `default text` sowie weitere Zeilen für die anderen Sprachen, die im Menüpunkt *File > Document Properties* definiert wurden. Nur die Sprachen, die auch Übersetzungen in der `csv`-Datei enthalten, sind hier aus- bzw. abwählbar. Die restlichen Sprachen werden ausgegraut.

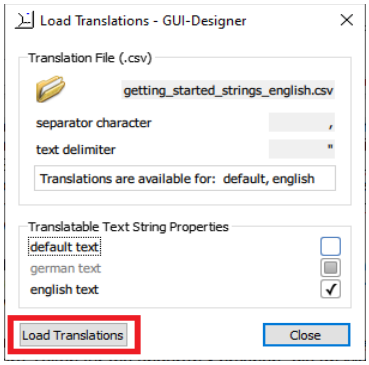


Abbildung 128: Einstellungen angepasst, um nur englische Texte zu laden

Tipp: Da `default text` auch in dieser Liste steht, ist es tatsächlich möglich nicht nur Fremdsprachen sondern auch die Eigenschaft `default text` auszutauschen.

Nach dem Durchführen der gewünschten Einstellungen, lädt die Schaltfläche *Load Translations* dann die entsprechenden Sprachen in die passenden Einträge der *Text-Strings*-Liste (siehe Beispiel in Abbildung 128). Wie schon erwähnt, ist die Reihenfolge der Einträge bzw. Sprachen in der `csv`-Datei vollkommen beliebig. Es wird nur nach dem **eindeutigen Namen** sowie der Sprache gesucht und entsprechend ausgetauscht.

11. Externe Ressourcen: Bilder und Fonts

Ein Design wird in JSON-Format gespeichert und beinhaltet alle Listen, Objekte und deren Eigenschaften. Ein Design braucht aber zusätzlich externe **Ressourcen**, z.B. Bilder- sowie Font-Dateien. Diese externe Ressourcen müssen im Design festgelegt und im Binärdatei mit eingebunden, damit sie auf dem Display-Modul hochgeladen und verwendet werden können. Hierzu gibt es im GUI-Designer den Tab *Images* für **Bildressourcen** und den Tab *Fonts* für **Font-Ressourcen**.

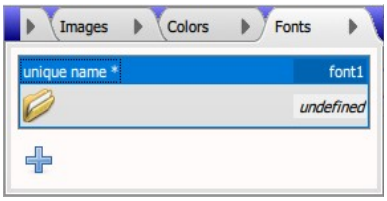


Abbildung 129: Eintrag für eine (noch undefinierte) Font-Ressource

Die Ressourcen werden jeweils in eine senkrechte Liste von umrahmten Einträgen aufgeführt (Abbildung 129). Der aktuelle Eintrag wird blau umrahmt. Wie bei den meisten Listen, kann man mit Drag & Drop die Liste umsortieren, mit dem Plussymbol ( ) einen neuen Eintrag anlegen, oder über das **Kontext-menü** eines Eintrags den Eintrag duplizieren bzw. löschen.

In allen Fällen bekommen neue Einträge automatisch einen **eindeutigen Namen** (`unique name`). Verbindungen zu einer Ressource erfolgen immer direkt durch diesen Namen bzw. für den **Controller** durch die daraus erzeugte **Objekt-ID**. Es wird empfohlen dem Eintrag einen sinnvollen Namen zu geben.

Abbildung 130: Das Ordnersymbol öffnet einen Dateiauswahldialog, um eine Ressource auszusuchen

Die Hauptaufgabe beider Tabs ist es, den Dateipfad und -name einer externen Ressource über einen *Dateiauswahldialog* (das *Ordnersymbol* – Abbildung 130) zu bestimmen, so dass auf sie von anderen Objekten über den eindeutigen Namen Bezug genommen werden kann. Mit dieser Vorgehensweise können Dateien leicht ausgetauscht werden, ohne die Bezüge alle manuell nachziehen zu müssen, da die Verbindungen durch den eindeutigen Namen bestehen bleiben.

In der Oberfläche des GUI-Designers sieht man die Dateinamen ohne Pfade, d.h. der Pfad ist zwar vorhanden, wird aber in der Oberfläche ausgeblendet. Beim klicken auf das *Ordnersymbol* wird der normale *Dateiauswahldialog* mit dem Ordner geöffnet, indem die Datei aktuell liegt, ohne das der Benutzer sich darum kümmern muss.

Es macht aber doch einen Unterschied, wie die Bilder und Fonts im Dateisystem abgelegt werden. Mehr dazu unter Details: Dateipfade von Bilder und Fonts.

11.1 Im Design verwendeten Bilder (*Images*-Tab)

Der *Images*-Tab befindet sich standardmäßig im oberen rechten Fenster des GUI-Designers. Darin werden alle bisher geladene externe **Bildressourcen** des Designs als Einträge (**Images**) der *Images*-Liste aufgeführt.

Abbildung 131: Eintrag einer ausgefüllten Bildressource

In jedem Image (Abbildung 131) wird links eine skalierte Abbildung der externen Bildressource dargestellt. Rechts davon ist das *Ordnersymbol*, um die *Dateiauswahldialog* aufzuklappen. Die Dateiname der externen Bildressource erscheint rechts daneben. Zusätzlich zu den **eindeutigen Namen** (`unique name`) gibt es auch die optionale Eigenschaft **Alphafarbe** (`alpha color`), im Abschnitt Details: Alphafarbe genauer erklärt.

Erlaubte Formate: Bitmaps, JPEGs und GIFs

Als **Bildressourcen** kommen Bitmaps (`.bmp`), JPEGs (`.jpg`) und GIFs (`.gif`) in Frage. Für **animierte Bilder** können hier auch animierte GIFs geladen werden. Es werden aber nicht alle Bildformate unterstützt, insbesondere werden Bitmaps mit einem Alphakanal nicht unterstützt. Siehe hierzu Details: Alphafarbe vs. Alphakanal.

Tipp: Auf die Größe des Bildes achten und evtl. auf einer geringeren Farbtiefe oder eines anderen Formats konvertieren.

Es soll auch auf die Größe der Bilder beachtet werden, da alle **Bild-** und **Font-Ressourcen** sowie **Text-Strings** gleichzeitig im Speicher des Display-Moduls reinpassen müssen! Evtl. können Bilder mit einer 24-bit Farbtiefe mit einem Bildverarbeitungsprogramm auf 256-Farbe reduziert werden, um Platz zu sparen. Es spart vielleicht auch Platz, wenn Bitmaps auf JPEG- oder GIF-Format konvertiert werden, wobei JPEG-Dateien merklich länger brauchen, um gezeichnet zu werden.

11.1.1 Details: Alphafarbe

Manche Widgets unterstützen eine **Alphafarbe**. Beim Zeichnen eines Bildes mit Alphafarbe werden alle Pixels in der Alphafarbe einfach übersprungen und bleiben daher durchsichtig für das, was im Hintergrund vorher gemalt wurde.

Wichtig! Alphafarbe nicht mit Alphakanal verwechseln!

Wichtig! Die Alphafarbe darf nicht mit dem Alphakanal eines Bildformats verwechselt werden! Mehr unter Details: Alphafarbe vs. Alphakanal.

Die Alphafarbe kann verwendet werden, um Bilder mit z.B. unregelmäßigen Umrandung zu zeichnen, damit alles außerhalb des Randes (oder innerhalb eines Loches im Bild) durchsichtig bleibt, obwohl ein Bild eigentlich immer als Rechteck gemalt wird (Abbildung 132).

Abbildung 132: EnumLabel-Eintrag mit Bild oben ohne und unten mit Alphafarbe

Die Alphafarbe, falls definiert, findet bei **statischen Bilder**

(*Image*-Widgets) und **indizierten Textauswahlen** (*EnumLabel*-Widgets) automatisch Verwendung. Bei **Buttons** (*Button*-Widgets) wird die Alphafarbe nur dann umwandelt, wenn explizit aktiviert (Checkbox `make alpha color transparent`). Die andere Widgets mit Bilder (z.B. **Schieberegler**, **Auswahllisten**, **animierte Bilder** und **Tastaturen**) unterstützen diese Funktion nicht; die Alphafarbe wird nicht durchsichtig.

Abbildung 133: Buttons mit Alphafarbe auf farbigem Hintergrund, 1 und 3 mit dem selben Button-Stil (Alphafarbe von 1 geht verloren), 2 mit eigenem Button-Stil (alles OK)

Besondere Sorgfalt wird bei der Verwendung von Alphafarben in Zusammenhang mit **Buttons** empfohlen. Die gewünschte Alphafarbe des Bildes wird intern an dem entsprechenden Button-**Stil** (**Stiltyp** `ButtonStyle`) gehängt und dort verwaltet. D.h. die Alphafarbe des **Images** für die Eingeschaft `image` wird an dem Button-Stil der Eigenschaft `style` angehängt. Ebenso für die `selected` und `ghosted` Eigenschaften. Falls mehrere Buttons die selben Stile verwenden, kann es dazu kommen, dass der nächste Button die Alphafarbe eines Stils überschreibt. Dies wirkt dann auf alle Buttons, die den Stil ebenfalls verwenden. Der letzte Button „gewinnt“ (Abbildung 133). Solange alle Bilder die selbe Alphafarbe verwenden, ist das überhaupt kein Problem. Aber bei Bilder mit einer abweichenden Alphafarbe soll gesonderte Button-Stile angelegt werden. Alle Buttons, die diese abweichende Alphafarbe brauchen, sollen auch die gesonderte Button-Stile verwenden, damit keine Alphafarbe-Überlappungen stattfinden.

Tipp: Ein Design mit unterschiedlichen Alphafarben gut auf dem Display-Modul testen, da Überlappungen nur dort sichtbar werden.

Technische Bemerkung: Im GUI-Designer werden die Stile der Einfachheit halber pro Widget neu erzeugt und dann wieder verworfen, so dass ein Problem mit überlappenden Alphafarben in der Entwicklungsumgebung zunächst ausbleibt. Dies ist im **GUI-Interpreter** auf dem Display-Modul nicht der Fall, da alle Stile des Designs nur einmal angelegt und in ihrem letzten Zustand festgehalten werden. Die Benutzung mehrerer Alphafarben mit Buttons sollen deswegen immer am Display-Modul gut überprüft werden!

11.1.2 Details: Alphafarbe vs. Alphakanal

Die **Alphafarbe** (Eigenschaft `alpha color`) in den **Images** der *Images*-Liste soll nicht mit dem Alphakanal (alpha channel) eines Bildformats verwechselt werden.

Die Alphafarbe erlaubt eine selbst-hergestellte Durchsichtigkeit bei bestimmten **Widget-Typen** (*Image*, *EnumLabel*, *Button*). Die angegebene Farbe wird als „durchsichtig“ behandelt und nicht gemalt.

Der Alphakanal wird von der GUI-Designer und GUI-Interpreter nur bedingt unterstützt!

Ein Alphakanal wiederum ist ein separater Kanal, der „keine“ Farbe hat und die Transparenz darstellt, bei 16-bit Farbtiefe nur „ein“ oder „aus“, bei 32-bit Farbtiefe von 0 bis 255 abstufbar. Der Alphakanal wird von der GUI-Designer und **GUI-Interpreter** nur bedingt unterstützt, wie unten erläutert!

Tipp! Eine Bitmap mit einem Alphakanal kann evtl. in einer GIF-Datei konvertiert werden.

Eine Bitmap (`.bmp`) mit einem Alphakanal lässt sich weder in der GUI-Designer noch auf dem Display-Modul zeichnen. Nur Monochrom, 16- und 256-Farben (unkomprimiert bzw. RLE-komprimiert) sowie 24-bit (True Color) Bitmaps werden unterstützt, alle Formate ohne Alphakanal. Eine Bitmap mit Alphakanal soll umwandelt werden, in dem z.B. der Alphakanal durch eine andere Farbe ausgetauscht und die Bitmap ohne Alphakanal gespeichert wird. Die ausgetauschte Farbe soll sich am Hintergrund der Anwendung orientieren (sinnvoll für Widgets, die eine Alphafarbe nicht unterstützen), darf aber auch ganz fremd sein. Letzteres würde sich dann für die Alphafarbe (`alpha color`) eignen.

GIF-Dateien mit Alphakanal sind möglich, funktionieren aber nicht wie erwartet bei animierten Bildern.

Eine GIF-Datei mit einem Alphakanal andererseits lässt sich zunächst korrekt darstellen. Dies funktioniert einwandfrei für z.B. ein **statisches Bild** (*Image*-Widget) oder **Button**-Bilder. Aber für das **animierte Bild** (*Animation*-Widget) ist ein Alphakanal ungeeignet. Dies gilt sowohl für eine animierte GIF-Datei als auch für GIF-Dateien als Einzelbilder (*single-frame images*). Im GUI-Designer scheint alles bestens zu funktionieren, da die Entwicklungsumgebung bei jedem Einzelbild von neuem anfängt. Der GUI-Interpreter zeichnet aber anders! Die Einzelbilder werden nicht ausgetauscht sondern übereinander gezeichnet. Durchsichtige Pixel im neuen Bild lassen die darunter liegende Farben der vorigen Bilder durchscheinen. Die Animation wirkt immer unaufgeräumter evtl. bis zum totalen Stillstand.

11.2 Im Design verwendeten Fonts (*Fonts*-Tab)

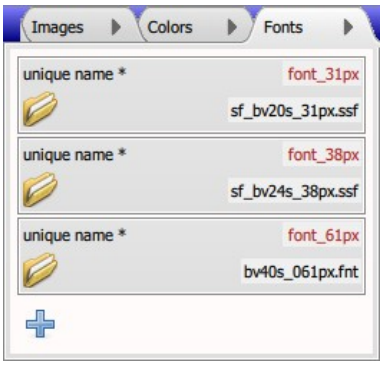


Abbildung 134: Fonts-Tab mit zwei SSF-Fonts und einer einfachen Font-Datei

Der *Fonts*-Tab befindet sich standardmäßig im oberen rechten Fenster des GUI-Designers. Darin werden alle bisher geladene externe **Font-Ressourcen** des Designs als Einträge (**Fonts**) der *Fonts*-Liste aufgeführt (Abbildung 134).

In jedem Font steht links das *Ordnersymbol*, um die *Datei-auswahldialog* aufzuklappen. Die Dateiname der externen Font-Ressource erscheint rechts daneben. Es gibt ansonsten nur den **eindeutigen Namen** (`unique name`).

Als Font-Ressourcen kommen einfache Font-Dateien (`.fnt`), und **SSF-Fonts** (`.ssf`) in Frage. SSF-Fonts sind spezielle platzsparende Fonts, um UTF8-kodierten (TrueType) Texte abzubilden. Mehr dazu unter Details: SSF-Fonts.

Tipp! Im Starterkit findet man eine Sammlung brauchbarer Fonts im Ordner `<version>\src\Misc`, z.B. bei V3.0.0 unter `3_0_0\src\Misc`.

Bemerkung: Auf die Dateigröße des Fonts soll beachtet werden, da alle Bild- und Font-Ressourcen sowie Text-Strings gleichzeitig im Speicher des Display-Moduls reinpassen müssen!

11.2.1 Details: SSF-Fonts

SSF-Fonts werden für die Mehrsprachigkeit im europäischen Raum gebraucht, um Akzente und Sonderzeichen sowie bspw. griechische und kyrillische Buchstaben abzudecken.

Ein **SSF-Font** wird für die Darstellung von UTF8 kodierten Texten (z.B. für asiatische Schriftarten) benötigt. Ein SSF-Font kann beliebig viele Zeichen enthalten und ist nicht auf 256 beschränkt wie das `fnt`-Format.

SSF-Fonts werden über den *SimplifyCharacterCompiler* (ein separates Tool) erzeugt. Man gibt diesem Tool den gewünschten TrueType-Font vor und einen Text, der alle benötigten Zeichen mindestens einmal in der UTF8-Kodierung enthält. Intern werden daraus mehrere `fnt`-Fonts mit jeweils 256 Zeichen erstellt und eine Übersetzungstabelle die jedem Unicode-Zeichen eine `fnt`-Font-Nummer und eine Zeichennummer zuordnet. Die `fnt`-Fonts und die Tabelle werden zu einem SSF-Font zusammengefügt. Dieser SSF-Font soll dann als Binärdatei abgespeichert werden. Diese Binärdatei wird dann als externe **Ressource** in der Liste der **Fonts** (*Fonts*-Tab des GUI-Designers) aufgenommen.

Es werden nur die benötigten Zeichen vermerkt, was enorm viel Speicherplatz spart!

Bei genauen Texteingaben werden eben so viel Zeichen zur Verfügung gestellt, wie im Design benötigt wird, nicht mehr und nicht weniger! Das spart enorm viel Speicherplatz gegenüber einen TrueType-Font.

Wichtig! Wenn Texte im Design geändert werden, wird es evtl. notwendig sein, den SSF-Font nochmal zu erzeugen. Dies trifft zu, wenn die neue Texte Zeichen enthalten, die im bisherigen SSF-Font (noch) nicht vorhanden sind.

11.3 Details: Dateipfade von Bilder und Fonts

Abbildung 135: Teil des Images-Tabs aus Kapitel 2; das selektierte Textfeld zeigt nicht nur den Dateinamen sondern auch das Relativpfad (images) dazu

Alle Bilder- bzw. Font-Dateien, die in oder unterhalb des Pfads der Projektdatei (`sgd`-Datei) liegen, werden relativ abgespeichert (Abbildung 135). Alle andere Bilder- bzw. Font-Dateien werden absolut abgespeichert. Wenn der Benutzer eine Projektdatei kopieren oder verschieben möchte, muss er zusätzlich alle dabei oder darunterliegende Bilder- und Font-Dateien ebenfalls kopieren oder verschieben. Sonst werden die Bilder bzw. Fonts nicht mehr gefunden.

Relative Pfade haben den Vorteil, dass man komplette „Projekte“ bestehend aus Projektdatei, Bildern und Fonts leicht zwischen verschiedenen Rechnern und Benutzern austauschen kann.

Absolute Pfade erlauben es z.B. Logos für mehrere Projekte an einer zentralen Stelle zu pflegen.

12. Sonstige Programmelemente

Der Aufbau der Fenstern mit verschiedenen Tabs, die darin enthaltenen Listen und Eigenschaften, sowie die *Design Area* zum Anzeigen des Designs wurden bereits ausführlich erklärt. Es gibt nur noch ein paar Standardkomponente eines Windows-Programms, die ebenfalls verwendet werden: Menüs, die Symbolleiste (Toolbar), und die Statuszeile (Statusbar).

12.1 Menüs

Menüs werden verwendet, um das zu verwirklichen, was nicht über die Seiten und Listen und ihre Eigenschaften festgelegt werden können, z.B. die übergeordneten Eigenschaften des Designs selbst, oder die Aufstellung einer Verbindung zum Display-Modul.

Die Menüpunkte werden hier erläutert, wobei die Details meist in dem Kapitel liegen, wo die Menüpunkte bei der Entwicklung gebraucht werden. In diesem Kapitel wird die Infos daher nicht wiederholt, sondern lediglich einen Link zum relevanten Abschnitt angeboten.

12.1.1 File-Menü

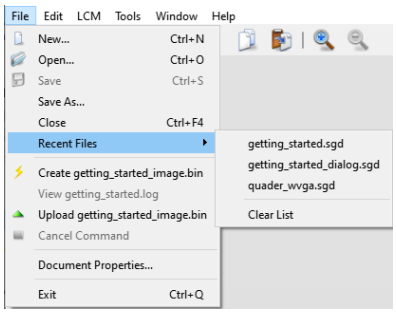


Abbildung 136: File-Menü

Der obere Teil des *File*-Menüs (Abbildung 136) wird wie üblich verwendet, um ein Design (`sgd`-Datei) zu erzeugen (*New*), öffnen (*Open*), speichern (*Save*), kopieren (*Save As*) oder schliessen (*Close*). Ein Design darf ebenfalls über eine Liste der zuletzt verwendeten Dokumente (*Recent Files*) geöffnet werden, wobei diese Liste auch bei Bedarf geräumt werden kann (*Clear List*). Auch wie üblich ist als letzte Menüpunkt das Verlassen des GUI-Designers (*Exit*) angebracht. Es gibt für die meisten dieser Kommandos entsprechende Tasten-kombinationen oder Symbole in der Symbolleiste.

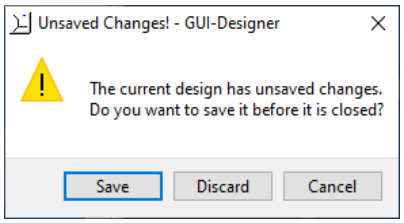


Abbildung 137: Dialogfenster, um Designänderungen zu sichern

Beim Designwechsel (*New*, *Open*, *Recent Files*) bzw. beim Schließen des Designs (*Close*) wird immer geprüft, ob das bereits geöffnete Design etwaige Änderungen aufweist. Wenn ja, wird immer erst gefragt, ob die Änderungen gespeichert (*Save*) oder verworfen (*Discard*) bzw. der Vorgang abgebrochen (*Cancel*) werden soll (Abbildung 137).

Bemerkung: Das Dialogfenster erscheint auch dann, wenn lediglich in den Tabs *Pixel Exact View* bzw. *Size Exact View* getestet wird, was wiederum die Änderung des Simulations-werts (`simulation value`) einer **Systemvariable** hervor-rufen kann. Der Simulationswert der Systemvariable wird zwar nicht auf dem LCD-Display (in der Binärdatei) übertragen, bedeutet aber trotzdem eine Änderung des Designs und wird mit dem Design gespeichert.

Im mittleren Teil des *File*-Menüs wird das Design für das Display-Modul aufbereitet und hochgeladen. Mehr zu diesem Vorgang unter Design auf das Display-Modul übertragen. Hier darf die Binärdatei des Designs zunächst angefertigt (*Create <nnn>_image_v?.bin*) und die dadurch erzeugte `log`-Datei angeschaut (*View <nnn>.log*) werden. Hier wird *<nnn>* durch den Namen des Designs und *?* durch *1* oder *2* ersetzt. Wenn alles fehlerfrei „kompiliert“, darf die Binärdatei zum Display-Modul hochgeladen werden (*Upload <nnn>_image_v?.bin*), wobei das *Upload*-Kommando auch abgebrochen werden darf (*Cancel Command*). Diese Kommandos sind auch in der Symbolleiste ganz rechts zu finden. Das *Upload*- und das *Cancel*-Kommando stehen doppelt im *LCM*-Menü.

Als letztes gibt es den Zugang zum Dialogfenster *Document Properties*, der im Abschnitt File > Document Properties ausführlich beschrieben wird.

12.1.2 Edit-Menü

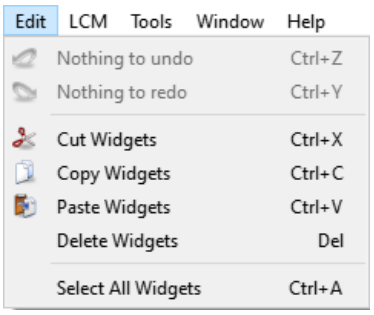


Abbildung 138: Edit-Menü

Die erste beiden Menüpunkte im *Edit*-Menü (Abbildung 138) lassen bereits erfolgte Design-Änderungen rückgängig machen (Undo) bzw. wiederherstellen (Redo). Wo sinnvoll, werden Änderungen zusammengefasst. Zum Beispiel, wenn ein Widget mehrmals hintereinander verschoben wird, rückt Undo das Widget wieder auf der Anfangsposition und Redo wieder auf der Schlussposition. Die Zwischenpositionen werden ignoriert. Das heißt konkret, dass die X- und Y-Positionen immer zusammen geändert werden. Analog dazu werden die Breite und Höhe eines Widgets (`width` und `height`) zusammen geändert. Im Allgemeinen werden aber Eigenschaftänderungen getrennt gehandelt.

Die Widgets-Menüpunkte arbeiten mit den Widgets in der *Design Area* zusammen. Ein oder mehrere selektierten Widgets dürfen ausgeschitten (*Cut Widgets*), kopiert (*Copy Widgets*), eingefügt (*Paste Widgets*) und gelöscht (*Delete Widgets*) werden. Es darf auch alle Widgets der aktuellen Seite selektiert werden (*Select All Widgets*).

Für alle Kommandos gibt es die üblichen Tastenkombinationen. Einige Kommandos befinden sich auch in der Symbolleiste.

Bemerkung: Zusätzlich zu den Tastenkombinationen in Abbildung 138 gibt es auch die traditionellen Windows-Tasten-kombinationen: *Cut Widgets* mit ⇧+Entf (Umschalt+Entf), *Copy Widgets* mit Strg+Einfg, und *Paste Widgets* mit ⇧+Einfg (Umschalt+Einfg).

Wichtig! Die Tastenkürzel dürfen auch für das Editieren in Textfelder verwendet werden; die Menüpunkte und Symbole der Symbolleiste dagegen nicht!

Wichtig! Beim Editieren in einem Textfeld werden alle Tastenkombinationen vorübergehend „umgeleitet“ und funktionieren wie gewohnt, um Buchstaben ausschneiden, kopieren, einfügen und löschen zu können. Die Menü-

punkte und Symbole der Symbolleiste arbeiten im Gegensatz dazu weiterhin nur mit ganzen Widgets.

12.1.3 LCM-Menü

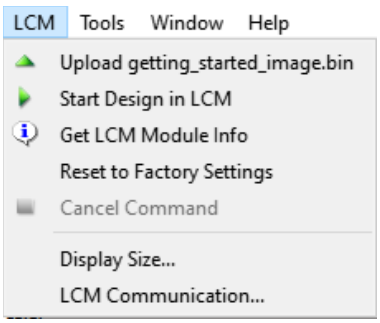


Abbildung 139: LCM-Menü

Die Menüpunkte des *LCM*-Menüs (Abbildung 139) verwalten die Eckdaten des Display-Moduls sowie die Kommunikation mit dem Display.

Hier darf das aktuelle Design als Binärdatei aufbereitet und auf dem Display-Modul hochgeladen werden (*Upload <nnn>_image_v?.bin*), wobei *<nnn>* durch den Namen des Designs und das Fragezeichen (*?*) durch *1* oder *2* ersetzt wird. Zu v1 und v2 siehe auch Tools-Menü. Mehr zu diesem Vorgang unter Design auf das Display-Modul übertragen. Es darf das Design im Display-Modul gestartet (*Start Design in LCM*) und wieder abgebrochen (*Cancel Command*) werden. (Bemerkung: hier handelt es sich immer um das zuletzt hochgeladene Design, das nicht unbedingt aktuell sein mag!) Die drei benannten Kommandos befinden sich auch in der Symbolleiste ganz rechts. Das *Upload*- und das *Cancel*-Kommando stehen doppelt im File-Menü.

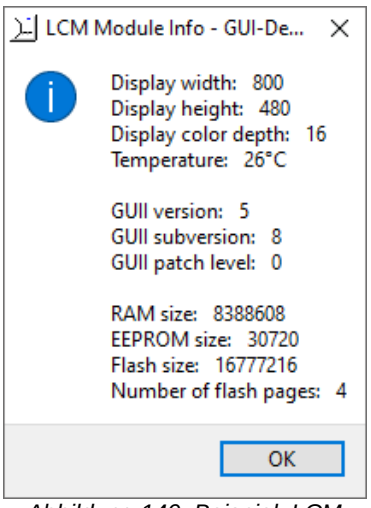


Abbildung 140: Beispiel: LCM Modulinfos

Es gibt zwei weitere nützliche Kommandos, womit der GUI-Designer mit dem Display-Modul austauschen kann. Mit *Get LCM Module Info* wird aktuellen Modulinfos vom Display-Modul geholt (Abbildung 140).

Mit *Reset to Factory Settings* wird das Display-Modul auf die Werkseinstellungen zurückgesetzt. Das Kommando wird manchmal nötig, z.B. in Zusammenhang mit einem **Startup-Objekt** mit aktiviertem **Autostart-Flag**. Das Kommando ist abhängig von der Speichergröße des Display-Moduls und kann ziemlich lang dauern. Deswegen darf das Kommando ebenfalls mit *Cancel Command* abgebrochen werden.

Mit dem Menüpunkt *Display Size* wird das Dialogfenster *Choose Display Size* aufgemacht, um die Größe des Display-Moduls zu bestimmen. Mehr hierzu unter Seitengröße bestimmen.

Wichtig! Die Pixelgröße des Display-Moduls ist von bedeutender Wichtigkeit, und soll bestimmt werden, bevor ein Design mit Widgets bestückt wird!

Mit dem Menüpunkt *LCM Communication* wird das Dialogfenster *LCM Communication* aufgemacht. Mehr hierzu unter Kommunikation zwischen Hardware und GUI-Designer.

12.1.4 Tools-Menü

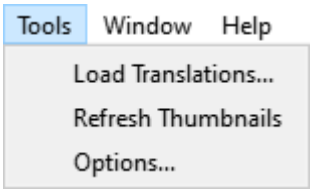


Abbildung 141: Tools-Menü

Das *Tools*-Menü (Abbildung 141) bietet einige Extras an.

Mit dem Menüpunkt *Load Translations* wird es möglich, externe Übersetzungen in das Design zu laden. Siehe Mehrsprachigkeit für ausführliche Infos, insbesondere den Abschnitt Übersetzungen einlesen (Tools > Load Translations).

Mit dem Menüpunkt *Refresh Thumbnails* werden alle **Miniaturbilder** im Tab *Pages*, die Abbildungen in Tab *Images* sowie alle auf **Bildressourcen** bezogenen Widget-Dimensionen aktualisiert.

Beim Einlesen eines Designs werden zunächst Miniaturbilder für alle Seiten erzeugt. Danach wird bei jeder Änderung des Designs das Miniaturbild der aktuellen Seite aktualisiert. Der Großteil aller Handlungen betreffen auch nur die aktuelle Seite, sodass dieses Verhalten meist ausreichend (und schnell) ist. Es gibt allerdings einige globalen Änderungen, die auf allen Seiten sofort wirksam aber nicht gleich auf allen Miniaturbildern übertragen werden, z.B. wenn man ein Farbwert im Tab *Colors* oder die aktuelle Sprache ändert. Hier wird dann *Refresh Thumbnails* sinnvoll, um die Änderung auf allen Miniaturbildern zu übertragen. Ersatzweise erwirkt das Auswählen einer Seite, z.B. durch das Klicken auf deren Miniaturbild, ebenfalls die sofortige Aktualisieren des Miniaturbildes.

Die Abbildungen im Tab *Images* können eine Aktualisierung benötigen, wenn **Bildressourcen** direkt im Dateisystem verschoben oder ausgetauscht werden. Hier erzeugt *Refresh Thumbnails* die Abbildungen neu und korrigiert nach Bedarf die aktuelle Größe aller damit verbundenen Widgets.

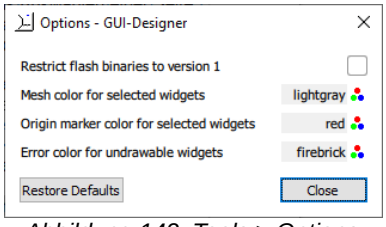


Abbildung 142: Tools > Options (Standardeinstellungen)

Mit dem Menüpunkt *Options* (Abbildung 142) gibt es weitere Anpassungen.

Falls der Firmware älter ist, darf die Binärdateien auf v1 eingeschränkt werden (*Restrict flash binaries to version 1*). Version 2 der Binärdatei kann nur bei der Firmware-Version 7 oder höher geladen werden. Version 1 funktioniert bei alle Firmware-Versionen, ist aber langsamer beim Starten des Designs auf dem LCM-Display.

Es gibt auch die Möglichkeit, einige Systemfarben anzupassen, damit sie für die Farbgebung Ihres Designs als Kontrastfarben gut wirken. Beispiele der Standardfarben werden in Abbildung 143 gezeigt.

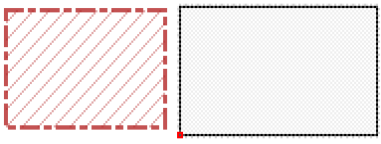


Abbildung 143: Fehlerhaftes Widget (links) und selektiertes Rechteck (rechts) (Standardfarben)

Frei wählbar sind
- die Gitterfarbe, die eines selektierten Widgets überzieht (*Mesh color for selected widgets*),
- die Positionsfarbe, die die Position (meist unten links) eines

selektierten Widgets hervorhebt (*Origin marker color for selected widgets*), und
- die Fehlerfarbe des Umrisses eines fehlerhaften, meist nicht zeichenbaren Widgets (*Error color for undrawable widgets*).

Der Knopf *Restore Defaults* stellt die Standardeinstellungen wieder her.

12.1.5 Window-Menü

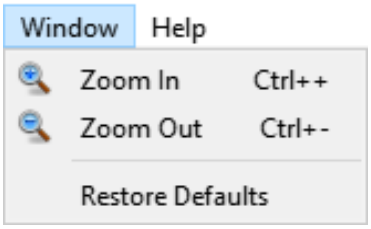


Abbildung 144: Windows-Menü

Das *Window*-Menü (Abbildung 144) beeinflusst das gesamt Layout der Fenster. Mit dem Knopf *Restore Defaults* wird die Hauptfenstergröße sowie das Verhältnis der kleineren Fenster untereinander wieder auf die Werkseinstellungen zurückgesetzt. Ebenfalls wird die Zoomfaktor der *Design Area* auf 4 gesetzt.

Die Menüpunkte *Zoom In* und *Zoom Out* beziehen sich lediglich auf das Fenster *Design Area*. Der Blick auf die aktuelle Seite des Designs darf stufenweise verdoppelt oder halbiert werden. Ein Zoomfaktor von 1, 2, 4, 8 oder 16 ist möglich. *Zoom In* erhöht die Vergrößerung, *Zoom Out* reduziert sie. Diese beide Optionen befinden sich ebenfalls als Lupen in der Symbolleiste.

12.1.6 Help-Menü

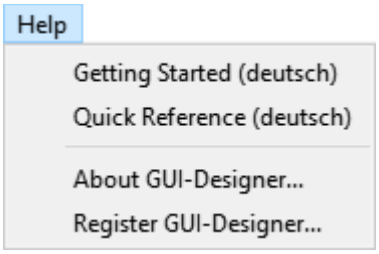


Abbildung 145: Help-Menü

Das *Help*-Menü (Abbildung 145) bietet einige Infos an und startet bzw. vollendet den Registrierungsvorgang.

Mit *Getting Started* steht Ihnen den Inhalt des Kapitels 2 dieses Handbuchs (auf deutsch) als `pdf`-Datei zur Verfügung. Die Kurzübersicht (*Quick Reference*) ist ebenfalls auf deutsch und zeigt die Oberfläche des GUI-Designers mit alle Menüs und Tabs samt Kurzbeschreibungen auf eine einzige Seite.

Mit dem Menüpunkt *About GUI-Designer* finden Sie Kurzinfos über die Software (z.B. Versionsnummer), der Stand Ihrer Registrierung, und die Kontaktdaten für Simplify Technologies GmbH. Hier gibt es auch den Knopf *Register GUI-Designer*, der das selbe Dialogfenster aufmacht wie im gleichlautenden Menüpunkt des *Help*-Menüs.

Die Software-Registrierung der Entwicklungsumgebung GUI-Designer (*Register GUI-Designer*) wird ausführlich in Anhang A: Registrierung des GUI-Designers beschrieben. Der GUI-Designer läuft ohne Registrierung als Testversion nach 30 Tage ab und wird in der Testzeit auf drei Seiten begrenzt.

Wichtig! Die Registrierung erfolgt nicht automatisch sondern nur in gegenseitigem Einvernehmen mit Simplify Technologies GmbH. Daher soll rechtzeitig vor dem Ablauf der Testversion die Registrierung beantragt werden!

12.2 Symbolleiste (Toolbar)

Die Symbolleiste (Toolbar) befindet sich direkt unterhalb der Menüleiste und vereinfacht die Ausführung einiger oft verwendeten Menü-Kommandos . Die gleiche Symbole befinden sich auch in den korrespondierenden Menüpunkten. Wenn die Maus kurz über ein Symbol verweilt, gibt ein Tooltip dazu weitere Hilfe, wie in Abbildung 146 beim Kommando *Cut Widget* dargestellt.

Bemerkung: Kommando-Details befinden sich nicht hier sondern in den angegebenen Menü-Links.

12.2.1 Standard-Kommandos

Die sogenannte Standardkommandos befinden sich links in der Symbolleiste (Abbildung 146). Die Abbildungen sind leicht zu erkennen, weil sie auch in vielen anderen Programmen zu finden sind.

Abbildung 146: Symbolleiste (links) mit geöffnetem Tooltip

Die abgebildeten Kommandos sind

- die File-Menü Kommandos *New*, *Open* und *Save*, um ein neues Design zu erzeugen (*New*), ein bereits existierendes Design zu öffnen (*Open*) bzw. die Änderungen des Designs zu speichern (*Save*);
- Die Edit-Menü Kommandos *Undo* und *Redo*, um Design-Änderungen rückgängig zu machen (*Undo*) oder wieder herzustellen (*Redo*).
- die Edit-Menü Kommandos *Cut Widgets*, *Copy Widgets* und *Paste Widgets*, um die bereits selektierten Widgets in die Zwischenablage entweder auszuschneiden (*Cut*) oder kopieren (*Copy*) bzw. aus der Zwischenablage Widgets auf die aktuelle Seite wieder einzufügen (*Paste*);
- die Window-Menü Kommandos *Zoom In* und *Zoom Out*, um die aktuelle Seite in der *Design Area* zu vergrößern (*Zoom In*) oder verkleinern (*Zoom Out*).

Wichtig! Die Symbole für *Cut/Copy/Paste* arbeiten nur mit ganzen Widgets zusammen. Sie können nicht verwendet werden, um z.B. den Text in Textfelder zu kopieren oder wieder einzufügen. In Textfelder funktionieren nur die bekannten Tastenkombinationen.

Abbildung 147: Symbolleiste (rechts) – LCM-Kommandos

12.2.2 LCM-Kommandos

Die LCM-Kommandos befinden sich rechts in der Symbolleiste (Abbildung 147). Diese Kommandos werden verwendet, um die

Binärdatei (Flash Binary) aufzubereiten, auf das Display-Modul hochzuladen und dort auszuführen. In den Beschreibungen unten wird *<nnn>* durch den Namen des Designs und die Fragezeichen (*?*) durch *1* oder *2* ersetzt. Zur v1 bzw. v2 siehe Tools-Menü).

Die abgebildeten Kommandos sind

- das File-Menü Kommando *Create <nnn>_image_v?.bin*, um eine Binärdatei des Designs aufzubereiten;
- die im File-Menü als auch im LCM-Menü aufgeführten Kommandos *Upload <nnn>_image_v?.bin* und *Cancel Command*, um die Binärdatei aufzubereiten und auf das Display-Modul hochzuladen (*Upload*) bzw. ein laufendes Kommando im Display-Modul abzubrechen (*Cancel Command*);
- das LCM-Menü Kommando *Start Design in LCM*, um mit dem **GUI-Interpreter** das Design zu starten, das sich gerade im Display-Modul befindet. Es wird mit *Cancel Command* beendet.

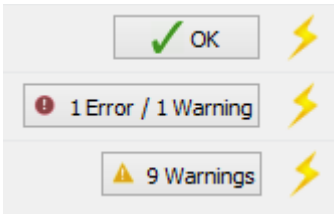


Abbildung 148: View-Log-Button mit Statusangabe – drei Beispiele

Es gibt aber auch einen weiteren Button, der erst dann links neben dem Blitzsymbol erscheint bzw. aktualisiert wird, wenn die Binärdatei erzeugt bzw. hochgeladen wird. In Abbildung 148 sind drei Beispiele, wie der Button aussehen könnte. Wenn Fehler (Errors) auftauchen, kann die Binärdatei erst dann hochgeladen werden, nachdem die Fehler entfernt worden sind. Bei Warnungen (Warnings) darf die Binärdatei hochgeladen und ausgeführt werden. Ein Klick auf dem Statusbutton führt das File-Menü Kommando *View <nnn>.log* aus, um die Warnungen oder Fehler in der `log`-Datei zu betrachten (einfach nach „Warning“ bzw. „Error“ suchen).

12.3 Statuszeile (Statusbar)

Die Statuszeile (Statusbar) befindet sich unterhalb des Arbeitsbereichs und zeigt auf der linke Seite hauptsächlich Status- bzw. Fehlermeldungen an. Meldungen bleiben nur für eine kurze Zeit, maximal bis etwa 10 Sekunde (Abbildung 149).

Abbildung 149: Statuszeile nach einem Abbruch (ESC) beim Editieren der Eigenschaft `background color`

Auf der Rechte Seite gibt es Zusatzinfos, die auch länger erscheinen können. Wenn die Maus sich in der *Design Area*, *Pixel-Exact View* oder *Size-Exact View* befindet, wird in der Statuszeile rechts die genaue Position der Maus in Design-Koordinaten angegeben (Abbildung 149). Hier befindet sich der Nullpunkt des Designs immer unten links, y läuft positiv nach oben, x positiv nach rechts.

Wenn ein LCM-Kommando gerade läuft (z.B. *Upload* oder *Start*

Design) wird das Kommando mit einem animierten Warteschleife in der Statuszeile ganz rechts dargestellt, wobei die Design-Koordinaten (falls angezeigt) etwas weiter nach links erscheinen (Abbildung 150).

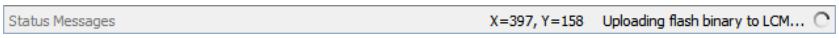


Abbildung 150: Statuszeile beim Hochladen einer Binärdatei

13. Anhang A: Registrierung des GUI-Designers

Der GUI-Designer wird zunächst als eingeschränkte Testversion ohne Lizenzschlüssel ausgeliefert. Dies liegt daran, dass der uneingeschränkte Lizenzschlüssel an die Hardware-ID Ihres Computers gebunden ist.

Die Testversion ist auf nur 3 Design-Seiten beschränkt und läuft nach 30 Tagen ab. Es ist daher wichtig, dass Sie Ihren Lizenzschlüssel so bald wie möglich nach der Installation der Software beantragen.

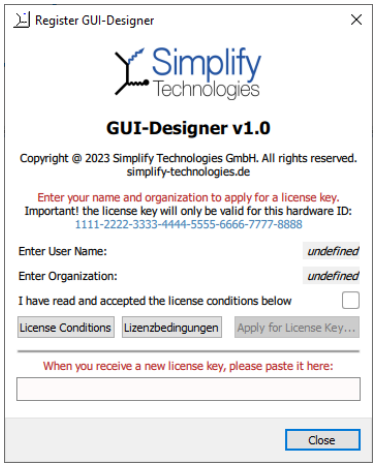


Abbildung 151: Help > Register GUI-Designer, Ausgangszustand im Trial-Modus

13.1 Lizenzschlüssel beantragen

Um einen Lizenzschlüssel zu beantragen, starten Sie den GUI-Designer und wählen Sie das Menü *Help > Register GUI-Designer*. Es öffnet sich ein Dialogfenster, das die Hardware-ID (`hardware ID`) in blau anzeigt und zusätzliche Felder zur Eingabe Ihres Benutzernamens und Ihrer Organisation enthält (Abbildung 151). Die angezeigte Hardware-ID gilt nur für diesen Computer. Die Beantragung eines Lizenzschlüssels muss daher für jeden Computer wiederholt werden, auf dem die registrierte Software verwendet wird! Weitere Informationen finden Sie unter Details: Sonderbedingungen.

Anfangs ist die Schaltfläche *Apply for License Key* deaktiviert. Sie müssen einen Benutzernamen (`user name`) und eine Organisation (`organization`) eingeben und die Lizenzbedingungen akzeptieren, bevor Sie einen Lizenzschlüssel beantragen können. Klicken Sie auf das Wort *undefined*, um das entsprechende Textfeld zu bearbeiten. Über Schaltflächen können die Lizenzbedingungen auch in englischer oder deutscher Sprache angezeigt werden. Wenn alles korrekt ausgefüllt ist, wird die Schaltfläche *Apply for License Key* aktiviert (Abbildung 152).

Abbildung 152: Help > Register GUI-Designer nach dem Ausfüllen der erforderlichen Felder

Klicken Sie auf die Schaltfläche *Apply for License Key*, um automatisch einen E-Mail-Client (mit `mailto:`) mit einer vorformatierten Nachricht zu öffnen, die die für die Registrierung der Software erforderlichen Informationen enthält. Ein Beispiel ist in Fehler: Verweis nicht gefunden dargestellt. Beachten Sie, dass diese vorformatierte Nachricht bereits den Empfänger und eine entsprechende Betreffzeile zusammen mit dem Nachrichtentext enthält. Die Meldung identifiziert die gewünschte Software und gibt die Hardware-ID an (wichtig!). Diese Informationen sind normalerweise ausreichend, um den Registrierungsprozess zu starten. Senden Sie die Nachricht von Ihrem E-Mail-Client aus und beenden Sie die Anwendung.

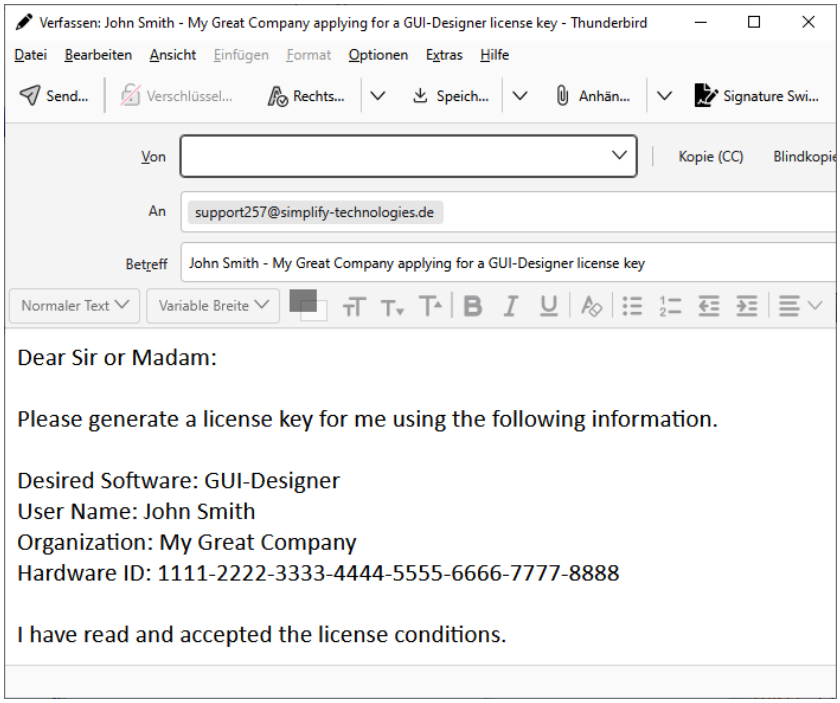


Abbildung 153: Automatisch generierte E-Mail zur Beantragung eines Lizenzschlüssels

13.1.1 Details: Sonderbedingungen

Normalerweise sollten die obigen Anweisungen ausreichen, um einen Lizenzschlüssel erfolgreich zu beantragen. Wenn es Schwierigkeiten gibt, trifft vielleicht einer der folgenden Fälle auf Sie zu:

1. Wenn Sie ein webbasiertes E-Mail-System anstelle eines lokalen E-Mail-Clients verwenden, fügen Sie den Inhalt der automatisch generierten E-Mail in eine neue E-Mail auf Ihrem Web-Client ein (einschließlich der Betreffzeile und des Empfängers - support257@simplify-technologies.de - wie im Beispiel gezeigt) und senden Sie die E-Mail von Ihrem Web-Client.
2. Wenn `mailto:` auf Ihrem Computer überhaupt nicht funktioniert (d.h. es ist kein lokaler Mail-Client installiert), wird ein Mustertext (samt der Support-Adresse und dem Betreff!) in die Zwischenablage kopiert. Den Text in einem E-Mail oder FAX entsprechend kopieren und uns zusenden. Benutzen Sie unsere E-Mail-Adresse support257@simplify-technologies.de oder die FAX-Nummer +49 (0) 6441-210399.
3. Wenn Sie nicht berechtigt sind, selbst eine Lizenz zu beantragen (z. B. wenn solche Anträge nur von der Einkaufsabteilung gestellt werden), müssen Sie der autorisierten Person die genaue Hardware-ID für jeden Computer mitteilen, der die Software verwenden wird. Jede Hardware-ID sollte mit einem eindeutigen Benutzernamen verknüpft sein, damit jeder erhaltene Lizenzschlüssel dem richtigen Computer zugewiesen werden kann. Alle

Hardware-IDs können in einer E-Mail enthalten sein, solange die Verbindung zwischen jeder Hardware-ID und einem Benutzernamen klar ist.

4. Wenn die Probezeit abgelaufen ist, lesen Sie den separaten Abschnitt Wenn die Probezeit abgelaufen ist.

13.2 GUI-Designer freischalten

Die Simplify Technologies GmbH generiert einen Lizenzschlüssel, wenn alle Voraussetzungen wie ausstehende Zahlungen, die Bereitstellung fehlender Informationen usw. erfüllt sind. Der Lizenzschlüssel wird Ihnen in einer E-Mail ähnlich dem folgenden Beispiel zugesandt (Abbildung 154).

Abbildung 154: E-Mail von Simplify Technologies GmbH mit beiliegendem Lizenzschlüssel

Abbildung 155: Erfolgreich registrierte Software

Die angehängte `txt`-Datei enthält eine langen Folge von Ziffern und Buchstaben, die über mehrere Zeilen verteilt sein können. Kopieren Sie den gesamten langen String in die Zwischenablage, starten Sie den GUI-Designer und wählen Sie im Menü *Help > Register GUI-Designer*. Es erscheint das gleiche Dialogfenster wie in Abbildung 151. Diesmal ist es nicht erforderlich, Informationen im oberen Teil des Dialogfensters einzugeben. Fügen Sie stattdessen einfach den Lizenzschlüssel in das Textfeld am unteren Rand des Dialogfensters ein. Wenn der Lizenzschlüssel für diesen Computer korrekt ist (d.h. für diese Hardware-ID), werden die anderen Felder (Benutzername usw.) automatisch ausgefüllt und können nicht mehr bearbeitet werden (Abbildung 155). Ihre Software ist jetzt registriert und Sie können den GUI-Designer ohne Einschränkungen sofort weiter verwenden.

13.3 Wenn die Probezeit abgelaufen ist

Wenn der Testzeitraum abgelaufen ist (nach 30 Tagen), wird der GUI-Designer nicht wie erwartet ausgeführt. Stattdessen wird

nach einer Meldung über das Ablaufen des Testzeitraums ein spezielles Registrierungs-Dialogfenster angezeigt (siehe Abbildung 156). Dieses Dialogfenster ähnelt dem in Abbildung 151, enthält jedoch einige zusätzliche Informationen. Gehen Sie vor wie in den Abschnitten Lizenzschlüssel beantragen und GUI-Designer freischalten beschrieben. Der Hauptunterschied besteht darin, dass Sie mit dem Dialogfenster in Abbildung 156 anstatt mit dem einfacheren Dialogfenster aus Abbildung 151 arbeiten. Wenn Sie den Lizenzschlüssel in das Textfeld einfügen, sehen Sie auch sofort, ob der Lizenzschlüssel akzeptiert wurde (wie Abbildung 155). Schließen Sie jedoch das Dialogfenster hier, wird auch das Programm beendet. Sie müssen den GUI-Designer neu starten, damit der Lizenzschlüssel wirksam wird.

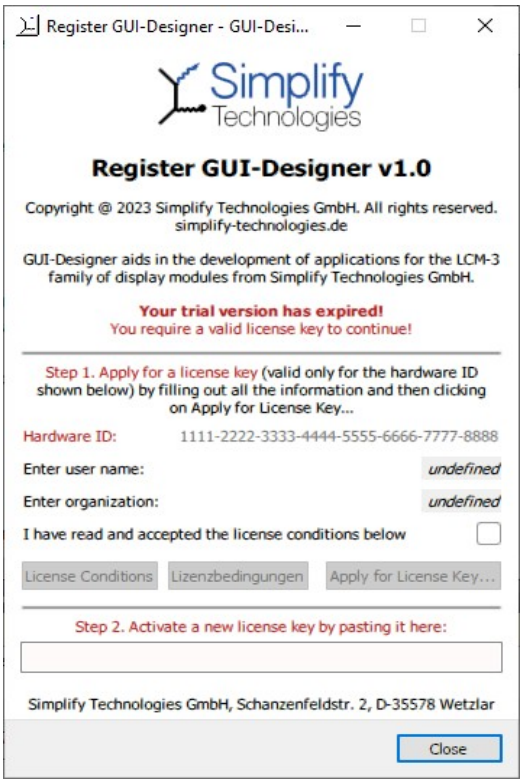


Abbildung 156: Registrierdialogfenster wenn die Probezeit abgelaufen ist

14. Anhang B: Konventionen und Einschränkungen

Es gibt einige Konventionen sowie Einschränkungen, die hier erläutert werden.

14.1 Barrierefreiheit

Die Barrierefreiheit wird angestrebt. Es ist möglich, durch die Eigenschaften und Listen lediglich mit Tasten zu navigieren.

Mit der Tabulatortaste und Umschalttaste+Tabulatortaste kann durch die Tabs und deren aktuellen Eintrag sowie durch die Symbolleiste navigiert werden. Diese beide Tasten werden auch in den Dialogboxen verwendet, um durch die Eigenschaften zu navigieren.

Ansonsten navigiert man in den Tabs vor- und rückwärts durch die einzelne Eigenschaften mit den Cursortasten.

Das **Kontextmenü** des aktuell selektierten Eintrags einer Liste ist über die Menütaste bzw. die Umschalttaste+F10 zu erreichen.

In einer Liste darf auch mit den Tasten Bild-↑ (PgUp) und Bild-↓ (PgDn) die einzelne Einträge der List durchgegangen werden. Speziell im Tab *Properties*, kann man mit Bild-↑ zur ersten Eigenschaft und mit Bild-↓ zur untergeordneten Liste (falls vorhanden) bzw. letzten Eigenschaft springen.

Ein Sonderfall bietet hier der Tab *New*, wo Bild-↓ und Bild-↑ zum nachsten bzw. vorigen Abschnitt schreitet. Dabei wird der verlassene Abschnitt automatisch aus- und der neue Abschnitt bei Bedarf eingeblendet. In Kontrast dazu, blenden die Cursortasten einen Abschnitt bei Bedarf ein aber nie aus.

Für die aktuelle Eigenschaft wird die Eingabetaste verwendet, um die Eigenschaft zu ändern. Wenn spezifisch einen neuen Text eingetippt werden soll, darf auch mit Umschalttaste+F2 das Editieren der Eigenschaft initiiert werden. Checkboxen können mit der Eingabetaste oder mit der Leertaste abgehakt werden.

14.2 Numerische und syntaktische Regeln

Numerische **Systemvariablen** (`Number`-**Sysvars**) dürfen im Bereich zwischen $-2^{31} + 2$ und $2^{31} - 2$ liegen.

Bei der Darstellung von Zahlen (z.B. in einer **Zahl** (*Number*-Widget) oder in einer **Dialog-Sysvar**) ist die Anzahl der Nachkommastellen auf neun beschränkt (siehe Eigenschaft `decimal digits`).

Bei der **Zahl** (*Number*-Widget) darf der Zahlenwert zusammen

mit der Einheit nicht länger als 20 Zeichen sein.

Die Eigenschaft `unique name` (**eindeutiger Name**) muss immer *wohlgeformt* sein, d.h. das erste Zeichen muss ein Buchstabe (`a-z` oder `A-Z`) sein und alle weiteren Zeichen müssen Buchstaben oder Ziffern sein. Umlaute, Leerzeichen und Sonderzeichen sind nicht erlaubt. Andere Eingaben werden bei `unique name` nicht zugelassen.

14.3 Besonderheiten

In seltenen Fällen (z.B. bei der Änderung einer Eigenschaft mit globaler Wirkung) entsprechen die Miniaturbilder im Tab *Pages* nicht mehr genau deren Inhalt. Hierzu gibt es die Kommando *Tools > Refresh Thumbnails*. Weitere Details im Abschnitt Tools-Menü.

Bei **Systemvariablen** wird die Eigenschaft `simulation value` nie auf das Display-Modul übertragen. Auf dem Display-Modul gilt nur den Anfangswert (`initial value` bzw. `initial persistent value`), um Werten auf dem Display-Modul ordentlich zu initialisieren. Der Simulationswert wird nur für das Testen im GUI-Designer verwendet.

Beim **Button** wird die Button-Größe dem Text nicht automatisch angepasst. Der Text könnte über die Button-Ränder hinausragen. Der Entwickler muss sicherstellen, dass die Button-Höhe bzw. Breite für den Text (evtl. in verschiedenen Sprachen!) und Font ausreichend groß ist.

Beim **Schieberegler** (*Slider*-Widget) kann der Schieberegler sowohl waagerecht als auch senkrecht dargestellt werden (`vertical`-Checkbox). Beim (de-)aktivieren der `vertical`-Checkbox werden allerdings andere Eigenschaften, vor allem die Höhe und Breite, in der Regel nicht angepasst, welches zunächst zu einer seltsamen Darstellung führen kann. Lediglich die zwei eingebauten Bilder `DefaultHSlider` bzw. `DefaultVSlider` (und nur sie!) werden automatisch ausgetauscht. **Images** werden ansonsten nie verdreht. Man muss die Bilder für den Schieberegler in der korrekten Orientierung zur Verfügung stellen.

Bei der **Balkenanzeige** (*BarIndicator*-Widget) kann die Balkenanzeige sowohl waagerecht als auch senkrecht dargestellt werden (`vertical`-Checkbox). Beim (de-)aktivieren der `vertical`-Checkbox werden allerdings andere Eigenschaften, vor allem die Höhe und Breite, in der Regel nicht angepasst. Dies führt zunächst zu einer seltsamen Darstellung.

Bei der **Auswahlliste** (*SelectionList*-Widget) ist es möglich, den

Schieberegler auszuschalten (`slider`-Checkbox). Es kann aber sein, dass auf dem Display trotzdem Platz für den Schieberegler reserviert wird. Das passiert wenn der Firmware-Version < 6.0.0 ist. Als Workaround soll zusätzlich im dazugehörigen Hauptstil (vom Typ `SelectionListStyle`) die Eigenschaft `slider width` auf `0` gesetzt werden. Damit ist der Slider in aller Versionen der Firmware komplett abgeschaltet.

Bekannter Fehler: Bei der Eingabe einer String-Eigenschaft kann es vorkommen, dass die Eingabe eines Buchstabens mithilfe der `ALT-GR`-Taste (z.B. einer eckigen oder geschweiften Klammer auf deutsche Tastaturen) den Eingabevorgang beendet. Workaround: Beim Wiedereditieren kann die Eingabe fortgesetzt werden.

Bekannter Fehler: Beim Editieren in einem Textfeld kann es vorkommen, dass der Eingabevorgang nach ein oder zwei Buchstaben einfach beendet wird. Dies passiert nur bei dem erst editierten Textfeld, nachdem man von einem anderen Programmfenster in den Gui-Designer zurück wechselt. Beim Wiedereditieren ist das Problem behoben.

15. Anhang C: Verzeichnis der Definitionen

Mit einem "Klick" auf die folgenden Begriffe gelangen Sie zu der entsprechenden Definition: