AppWizard

Wizard for creating ready-to-use emWin applications

User Guide & Reference Manual

Document: UM03003
Software Version: 1.30
Revision: 0
Date: August 4, 2022

A product of SEGGER Microcontroller GmbH

www.segger.com

Disclaimer

The information written in this document is assumed to be accurate without guarantee. The information in this manual is subject to change for functional or performance improvements without notice. SEGGER Microcontroller GmbH (SEGGER) assumes no responsibility for any errors or omissions in this document. SEGGER disclaims any warranties or conditions, express, implied or statutory for the fitness of the product for a particular purpose. It is your sole responsibility to evaluate the fitness of the product for any specific use.

Copyright notice

Trademarks

Names mentioned in this manual may be trademarks of their respective companies.

Brand and product names are trademarks or registered trademarks of their respective holders.

Contact address

SEGGER Microcontroller GmbH

Ecolab-Allee 5
D-40789 Monheim am Rhein

Germany

Tel.	+49 2173-99312-0
Fax.	+49 2173-99312-28
E-mail:	support@segger.com*
Internet:	www.segger.com

*By sending us an email your (personal) data will automatically be processed. For further information please refer to our privacy policy which is available at https://www.segger.com/legal/privacy-policy/.

Manual versions

This manual describes the current software version. If you find an error in the manual or a problem in the software, please inform us and we will try to assist you as soon as possible. Contact us for further information on topics or functions that are not yet documented.

Print date: August 4, 2022

Software	Revision	Date	By	Description
1.30	1	220804	JE	Chapter *Interactions* updated. • New job ROTATEDISPLAY added. • New job SETRANGE added.
1.28	1	220601	JE FO	Chapter *User interface* updated. • Section "Importing and exporting of content" added.
1.28	0	220428	FO	Chapter *User interface* updated. • Quick access button section enhanced. Chapter *Objects* updated. • New Wheel object added. • The following object property descriptions were moved to the objects they are used in: Auto repeat, Blend colors, Disable animation, Error correction level, Fade mode, Gradient, Keyboard layout, JPEG/GIF/BMP, Line width, Maximum length, Offset, Password mode, Persistent mode, Pixelsize, Rotate marker, Rounded value/ends, Snap position, Start/end angle, Version • Added "Untouchable" property to the objects Image, Text and Box. Chapter *Interactions* updated. • Jobs SETPERIOD, MOVETO, SETALPHA, SETSCALE, SETANGLE, INVALIDATE added. Chapter *Command line usage* added.
1.26	0	220113	JE	Chapter *Objects* updated. • New Dropdown object added. • New Listbox object added. • New Listview object added. • New property 'Content' added. Chapter *Interactions* updated. • New job ADDITEM added. • New job DELITEM added. • New job INSITEM added. • New job SETITEM added. Chapter *Resource management* updated. • Introduction updated. • Stock fonts updated. • Explanation for use of alpha channel images added. Chapter *User interface* updated. • Workspace directory added to preferences. Chapter *Animations* updated. • More information about autostart option added.
1.24	0	211116	FO	Chapter *Drawings* added. • New feature allowing custom drawings in an application.
1.22	0	210608	JE	Chapter *User interface* updated. • Sample browser added to file options. Chapter *Objects* updated. • New property 'Motion support' added. Chapter *Interactions* • New job SETSTART added. • New job SETEND added.
1.20	0	210308	FO	Chapter *User interface* updated. • Multibuffering option added to project options. Chapter *Objects* updated. • New property 'Stay on top' added. • New property 'Untouchable' added. • Vertical mode: Default bitmaps are switched automatically. Chapter *Interactions* • New signal PIDPRESSED added. • New signal PIDRELEASED added. • New signal UNPINNED added. • New signal FIXED added. • New job SETBITMAP added. • New job SHIFTWINDOW added. • New job ANIMCREATE added.

Software	Revision	Date	By	Description
				• New job ANIMSTART added. • New job ANIMSTOP added. • "Index" parameter added to job SETCOLOR. • Jobs ANIMCOORD, ANIMVALUE, ANIMRANGE and CASCADECOORD removed. • Signal ANIMCOORD removed. New chapter *Animations* added. Chapter *User Code* updated. • Added custom user code section. Chapter *Board support packages (BSPs)* updated. • Added "MultibufAvail" option.
1.14	0	210108	FO	Various screenshots updated. Chapter *Resource management* updated. • Sub-chapter 'Variable management' moved to dedicated 'Variables' chapter. Chapter *Objects* updated. • Hexadecimal mode added to Text object. • New Color property added to Image object. Chapter *Interactions* updated. • Job `SETFOCUS` added to Button, Multiedit, Rotary, Slider and Switch • New job `CALC` added. • New job `MODALMESSAGE` added. Chapter *Variables* added. Chapter *AppWizard SPY* updated. • Requirements added.
1.12	0	201106	FO	Chapter *Directory structure* updated. • SPY directory added. Chapter *User interface* updated. • AppWizard Spy window added. • Menu bar section enhanced. • Object IDs can be edited from within the hierarchic tree view. • By activating the option in the preferences dialog, object IDs can now be shown in the editor window. Chapter *Objects* updated. • Object focus section added. • Focus options property added. • Opaque property added. • Radius property added to Box and Button objects. • Object example screenshots updated. New chapter *AppWizard SPY* added. Chapter *Glossary* updated.
1.10	0	200824	FO	Chapter *Objects* updated. • Multiedit object added. • Timer object added. • Password mode property added. Chapter *Interactions* • New signal TIMER added. • New job START added. • New job STOP added.
1.08	0	200805	FO SC	Fixed several typos. Chapter *Objects* updated. • New object Progbar added. • Added 'Frame color' property to Text object. Chapter *Interactions* updated. • New signal `ENTER_PRESSED` added. • New job SETX0 added. • New job SETY0 added. • New job SETX1 added. • New job SETY1 added. Chapter *User Code* updated. • Function `APPW_GetText()` added. • Function `APPW_GetValue()` added. • Function `APPW_SetText()` added. • Function `APPW_SetValue()` added.
1.06	2	200626	FO	Chapter *Interactions* updated. • Added section about conditions.
1.06	1	200605	FO	Chapter *Objects* updated. • Added `KEYBOARD_ARA` layout for Arabic. • Added additional information about Keyboard object.
1.06	0	200602	FO	Chapter *Getting started* updated.

Software	Revision	Date	By	Description
				• AppWizard version section added. Chapter *Objects* updated. • Added Keyboard object. • Space property added. • Keyboard layout property added. • Text rotation property added. • Text wrapping property added. • Object properties reworked. Chapter *Interactions* updated. • Job ENABLEPID added. • Job SETFOCUS added. Chapter *User Code* updated. • Function `APPW_SetCustCallback()` added.
1.04	0	200408	FO	Chapter *User interface* updated. • Interaction window section enhanced. • Screenshots for positioning logic updated. Chapter *Objects* updated. • Added Gauge object. • Added QRCode object. • Error correction level property added. • Pixelsize property added. • Version property added. • Line width property added. • Rounded value/ends property added. • Start/end angle property added. Chapter *Interactions* updated. • Signal `TEXT_CHANGED` added. • Job SETLANG added. • Names of signals and slots have been shortened.
1.02	2	200323	FO	Chapter *Board support packages (BSPs)* updated. • Section 'Importing a custom BSP' updated.
1.02	1	200318	FO	Chapter *User interface* updated. • Screenshots updated. • Added information about Thai support.
1.02	0	200313	FO	Chapter *User Code* added. • Sub-chapter *Slot routines* merged. • Sub-chapter *Screen callback routines* added. • Sub-chapter *Fonts* added. • Sub-chapter *Variables* merged.
1.00	1	200306	FO	Chapter *Interactions* updated. • 'Slot routines' section updated. • 'Custom user code' section added. Chapter *Board support packages (BSPs)* updated. • Examples updated.
1.00	0	200226	JE FO	Initial release. Chapter *Board support packages (BSPs)* updated. • Section 'Preconfigured BSPs included in the shipment' updated. • Section 'Creating custom BSPs' updated. • Section 'Importing a custom BSP' added.
0.90	1	200221	FO	Chapter *Getting Started* updated. • Added note about the AppWizard Quick Start Guide. Chapter *Interactions* updated. • 'Slot routines' section updated. • Added job-specific parameters to each job.
0.90	0	200102	FO	Initial beta version.

About this document

Assumptions

This document assumes that you already have a solid knowledge of the following:

- The software tools used for building your application (assembler, linker, C compiler).
- The C programming language.
- The target processor.
- DOS command line.

If you feel that your knowledge of C is not sufficient, we recommend *C: A Reference Manual* by Harbison and Steele (ISBN 0--13--089592X). This book provides a complete description of the C language, the run-time libraries, and a style of C programming that emphasizes correctness, portability, and maintainability.

How to use this manual

This manual explains all the functions and macros that the product offers. It assumes you have a working knowledge of the C language. Knowledge of assembly programming is not required.

Typographic conventions for syntax

This manual uses the following typographic conventions:

Style	Used for
Body	Body text.
`Parameter`	Parameters in API functions.
`Sample`	Sample code in program examples.
`Sample comment`	Comments in program examples.
`User Input`	Text entered at the keyboard by a user in a session transcript.
`Secret Input`	Text entered at the keyboard by a user, but not echoed (e.g. password entry), in a session transcript.
Reference	Reference to chapters, sections, tables and figures.
Emphasis	Very important sections.
SEGGER home page	A hyperlink to an external document or web site.

Table of contents

Chapter 1

Introduction

This introduction gives some information about this document. It also gives an overview of the AppWizard’s features and its requirements.

1.1 What is the AppWizard?

The AppWizard is a tool for creating complete and ready-to-use emWin applications consisting of a number of screens.

Each screen consists of its own graphical control elements like buttons, sliders, images, text, child windows and so on. Applications are generated as a bundle of C files. Those C files are included automatically by the BSPs shipped with the AppWizard or can be used by a custom defined project.

Resources can be compiled and linked with the application code or stored externally on an SD card. Using resources at runtime from SD card is supported by the AppWizard without any additional configuration or code.

1.2 Features

Interactions

Interactions define the application behavior in case of user input. Several methods, animations or swiping can be used to switch between the application screens. To be able to extend the applications behavior user defined code can be invoked on interactions which can be edited within the AppWizard.

Conditions

A condition can be added to an interaction to determine precisely, under which circumstances an action should be executed. Conditions allow to implement a complex program logic into the application using the AppWizard.

Positioning

Positioning of objects can be done by specifying absolute coordinates or relative to already existing elements. Zooming can be used to be able to place small elements.

Resources

Resources like fonts, text and images are managed completely by the AppWizard. That means the application designer gets completely rid of resource management. Resources can be part of the created application code or generated as binary files to be stored on external media. The behavior ‘intern’ or ’extern’ can be specified for each resource separately. That makes it possible to have frequently used resources directly in the addressable ROM area and rarely used resource components on external media. The content of the resource folder is automatically managed by the AppWizard.

Variables and calculations

The user may also add variables to the project which can be manipulated from outside of the application. Variables are mostly used for interactions. For example, an interaction can be set for a variable that will be triggered after its value has changed. An interaction can also change the value of a variable.

Any form of calculation can be done using variables, just like in a C program. This provides even more opportunities for creating AppWizard applications.

Animations

Animations enable the user to animate objects in the application. Since most values within an application can be animated, such as object position and size, variables and object values, there are almost no limits to animations.

Drawings

Drawings allow to utilize a wide array of emWin’s 2D graphic drawing functions and add custom drawings to any widget object in the project. The drawing functions include e.g. standard geometric shapes such as rectangles, circles, arcs, and much more. A widget can draw the defined drawing before its standard appearance is drawn (pre-draw) or afterwards (post-draw).

Multiple languages

Multiple languages can be defined in the integrated multilingual text management system. Text can also be part of the application code or located on external media.

Font management

To be able to display the text with the right font the AppWizard contains its own font management system which is used to create emWin-fonts. The range of included codepoints can be specified for each font separately. It can be specified by custom defined pattern files, custom defined ranges or automatically by the range of characters resulting from

the application defined text. Fonts can also be located on an SD card or as part of the application code.

Integrated play mode

The internal play mode can be used for a quick check of the application's behavior without the need of compiling. Display size and color management can be changed on demand within the AppWizard.

BSPs (Board Support Packages)

The AppWizard comes with a set of BSPs which may also include precompiled libraries of emFile and embOS. In case of using a BSP the possibility of changing the color format and display size are restricted. Those ready-to-use and preconfigured BSPs make it possible to write and execute applications without any knowledge of writing applications in C code. All BSPs automatically include the generated application code, which means nothing needs to be changed or configured to be able to run the application on the target. Of course it is also possible to use custom defined BSPs with the AppWizard.

Simulation

Projects generated by the AppWizard also contain a simulation project for Microsoft Visual Studio. The difference between the integrated play mode and the simulation is that application defined code is not compiled and executed by the play mode. The simulation on the other hand also runs application defined code.

AppWizard SPY

The emWin SPY tool is fully integrated into AppWizard as AppWizard SPY. This tool allows the monitoring of memory usage in the application, as well as properties of the widgets that are present. The tool also has a logging feature and a recording feature so that certain debug cases can be rerun.

1.3 Requirements

1.3.1 Host system

The first version is available for Windows systems only, requiring Windows 7 or newer.

The recommended screen resolution is at least full HD (1920 * 1080).

1.3.2 Target system

To be able to use applications generated by the AppWizard we recommend that the target at least fulfills the following requirements:

- At least 256 KBytes of flash. *1
- At least 130 KBytes of RAM. *2
- At least a 32 bit CPU running at 100 MHz or more. *3

At the end RAM and ROM requirement depends on the application built with the AppWizard.

> **Note**
>
> *1 256 KBytes of flash memory are required for emWin and the AppWizard library. Additional flash memory or external storage is required for resources like fonts, images and text.
>
> *2 130 KBytes are required for emWin including 100 KByte of working RAM. This does not include memory required for a framebuffer.
>
> *3 Ideal would be a device with a hardware accelerator such as D/AVE 2D by Renesas or Chrom-ART Accelerator by ST.

1.3.3 Development environment

The AppWizard can be used with any IDE and any ANSI C compiler complying with at least one of the following international standards:

- ISO/IEC/ANSI 9899:1990 (C90) with support for C++ style comments (//)
- ISO/IEC 9899:1999 (C99)
- ISO/IEC 14882:1998 (C++)

1.3.4 Additional software libraries

No additional software library is required to be able to use the AppWizard. The AppWizard optionally supports resource (fonts, images and text) management from external storage. If external storage should be used for resource management a file system for reading operations is required. Any file system can be used.

Chapter 2

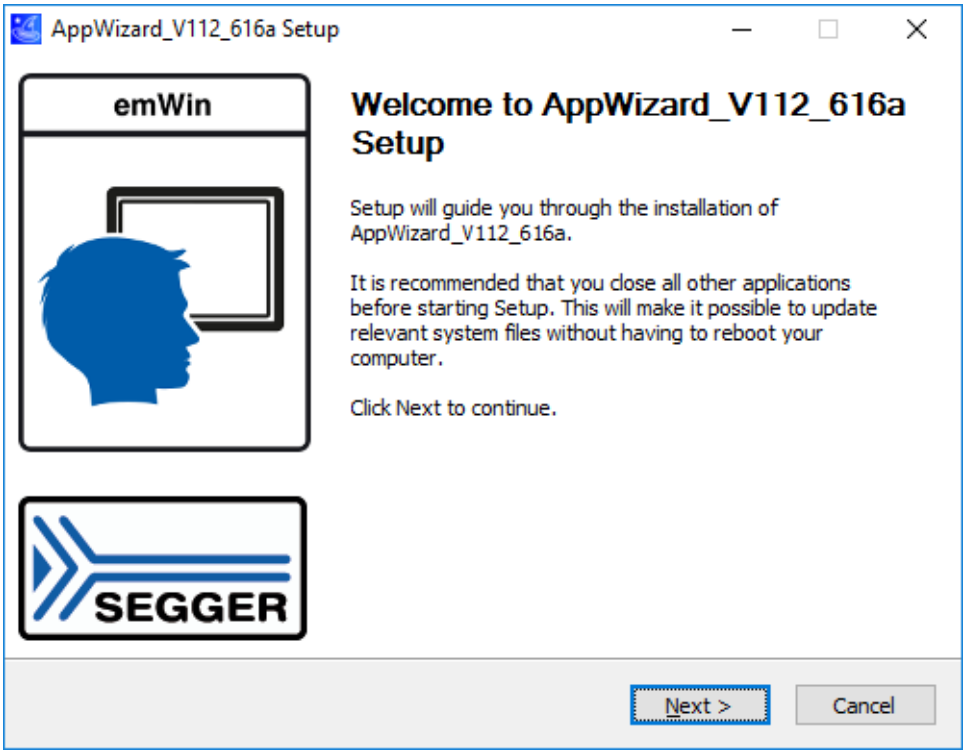
Installation

The following chapter describes how to install the AppWizard.

2.1 Microsoft Windows

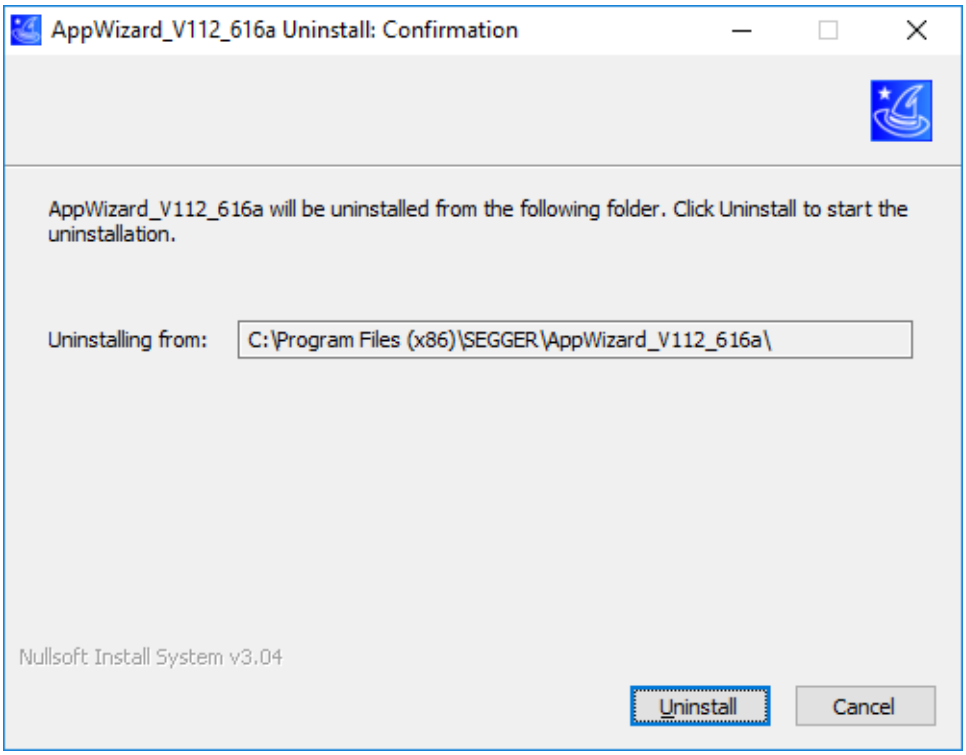
Installing the AppWizard

To install the AppWizard, simply run the setup wizard which will guide you through the installation. It comes with all required components without use of downloading and installing further tools.



Uninstalling the AppWizard

To uninstall the AppWizard, simply run the uninstaller which is located in the program directory.



Chapter 3

Getting started

The following chapter will provide an overview on how to get started with the AppWizard right after the installation has finished.

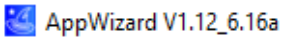
> **Note**
>
> The shipment also includes a **Quick Start Guide** to the AppWizard which provides step-by-step guides for creating example projects or performing simple actions (e.g. adding objects to the screen).
>
> The guide is located in the directory and named **AN03003_AppWizard_QuickStartGuide.pdf**.

3.1 Starting the tool

The AppWizard application (AppWizard.exe) can be started from the Windows Start menu or the installation directory.

3.2 AppWizard version

The current version of the AppWizard can be read in the top bar of the program. The version number contains the AppWizard version and the emWin version that is used.



In the above example, AppWizard version 1.12 is used in conjunction with emWin version 6.16a.

APP_Version.h

The current version of AppWizard is defined in the file `APP_Version.h`. The example below equals V1.04.

```
#define APPW_VERSION    10400
```

GUI_Version.h

The current emWin version is defined in `GUI_Version.h`. The example below equals V6.12.

```
#define GUI_VERSION    61200
```

> **Note**
>
> The used emWin libraries/code must match the corresponding emWin and AppWizard versions.

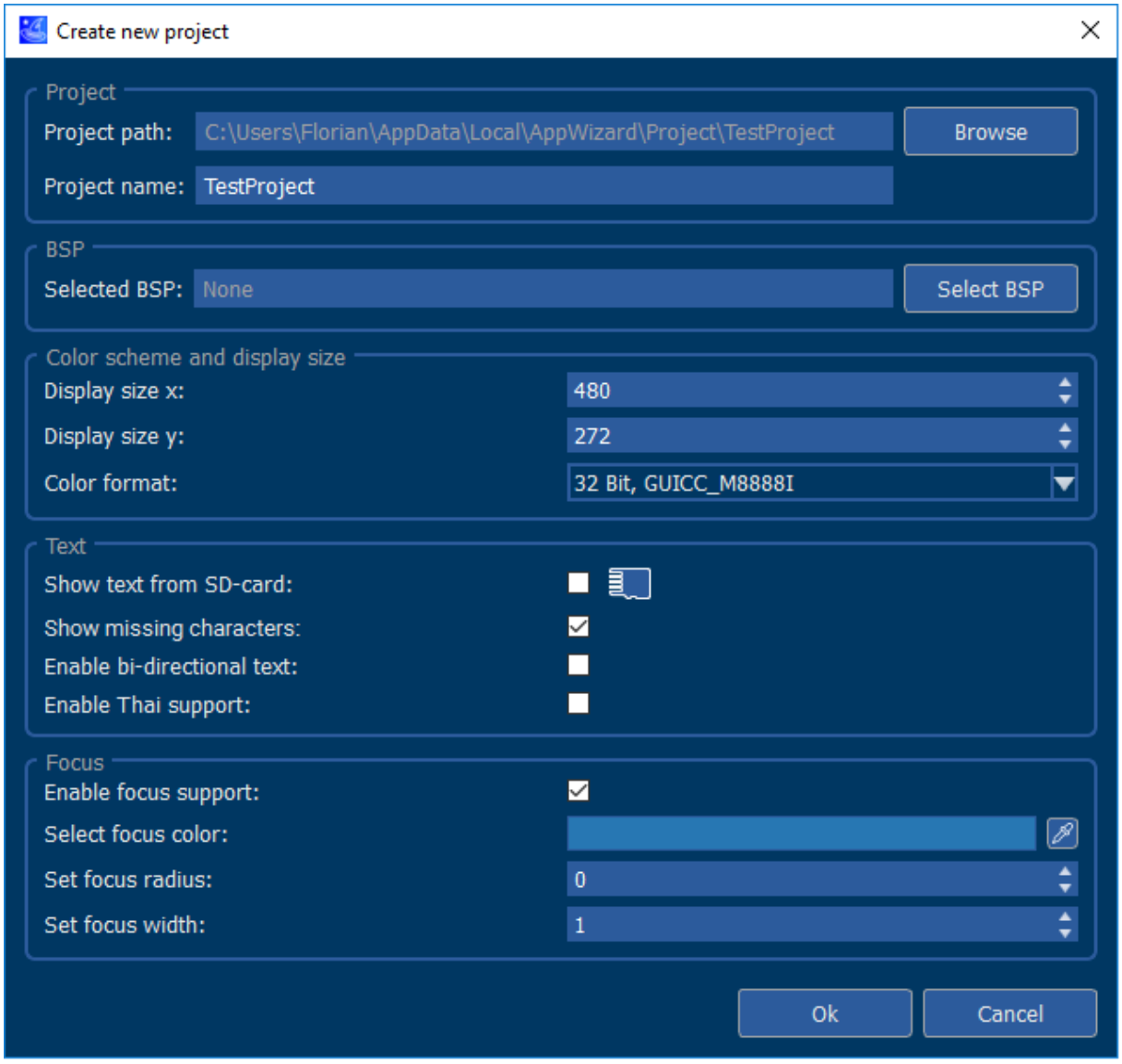
3.3 Creating a new project

The following section will guide you through the entire process of creating and running a project with the AppWizard.

❶ Create a new project

The initial step is to create a new project. Right after opening the AppWizard, the user has the option to either create a new project or open an existing one.

When creating a new project, the user can choose the project path, a name for the project, specify the target's display size and pick a color format. Alternatively a BSP can be selected, which already includes the respective display size and color format. The user also has the option to enable extern storage mode by ticking the checkbox next to the SD card image. Other options are to enable support of Thai script or bi-directional text.



When generating a project, the AppWizard also generates a simulation project in the folder `\Simulation` located in the project directory.

❷ Build up a structure

After the project has been created, the user can start to build their application by dragging objects onto the screen, adding interactions to the objects, or adding their resources to the project like bitmaps.

The first thing to add to an empty application is a screen object. This object serves as a parent object for all other objects to be added. Window objects may be added to the screen object to divide the screen into different sections, enhancing the application structure.

Objects like buttons can be placed into the screens or windows and the object's properties can be edited in the 'Properties' window to the right.

A more detailed explanation on how the user interfaces work can be found in the chapter *User interface* on page 30. To learn more about objects, see the chapter *Objects* on page 61.

❸ Run and test the application

As the user is building their application, they can run their application during this entire process of building. This makes it very easy to test the application. The application can be run by entering play mode which is done by clicking the play button in the top right corner of the editor window. More information about this can be found in *Play window* on page 42.

❹ Export and save the project

The option "File ➔ Save" (<CTRL>+S) simply saves the project file. If the user wants to save their application as C files, they are able to save and export their project by clicking "File ➔ Export & Save" (<CTRL>+<SHIFT>+E). By doing this, the AppWizard generates C files from this project.

❺ Run the simulation project

Once the project has been exported, the AppWizard generated C sources with runnable emWin code. The source files are located in the \Source directory. To run the generated code, the simulation project can be used which the AppWizard generated after the creation of the project. The exported source files are automatically linked to the simulation project, which means it is ready to be run.

❻ Compile and run on target

The chapter *Board support packages (BSPs)* on page 239 explains how to run a project on a hardware target.

3.4 Opening existing projects

The user may open existing projects either on start-up of the AppWizard by clicking the button “Open existing project” or by using the command “File ➔ Open” (<CTRL>+O).

The only files applicable for opening are AppWizard project files that have a `.AppWizard` extension. When opening an existing project, the project settings may be changed by selecting a different BSP, if needed.

Chapter 4

Directory structure

This chapter gives an overview on how the structure of an AppWizard project looks like.

4.1 Root folder

The root folder contains the project file <PROJECT_NAME>.AppWizard. It also contains the following sub-folders:

Folder	Description
/Source	Root directory for the generated application source code.
/Source/CustomCode	Directory for application source code which the user is allowed to extend by custom code.
/Source/Generated	Directory for fixed application code which should not be edited.
/Resource	Root directory for resource files.
/Resource/Image	Images used by the project including generated C and DTA files.
/Resource/Text	Text defined in the multilingual text editor.
/Resource/Font	Fonts created or referred by the project including generated C files.
/Target	Selected board support package for target hardware.
/Simulation	Simulation project.
/Spy	AppWizard SPY related files.

4.1.1 /Source

The folder /Source contains the file with the application entry point. The file is named <PROJECT_NAME>.c.

4.1.1.1 /Source/Config

The sub-folder /Config contains the following files intended to be changed/enhanced by the user:

File	Description
Application.h	Header file to be used by user defined code.
<SCREEN_ID>_Slots.c	Interaction slots to be used to invoke user defined code on interactions.

When opening a project, the AppWizard reads the user defined slot code. It can be edited within the AppWizard.

4.1.1.2 /Source/Generated

The sub-folder /Generated contains the following files not intended to be modified by the user.

File	Description
APPWConf.c	Configuration file (text and driver initialization).
Application.h	Header file to be used by user defined code.
<SCREEN_ID>.c	Screen definition(s).
Resource.c	Resource and screen information.
Resource.h	Prototypes of resource and screen information elements.

4.1.2 /Resource

The folder `/Resource` is the root directory for text, font and image resources:

4.1.2.1 /Resource/Font

The sub-folder `/Font` contains all font files referenced by the project:

File	Description
`<FONTNAME>.xbf`	Binary file of font in XBF format.
`<FONTNAME>.c`	Simple C arrays of XBF font files which are not managed on external memory.

4.1.2.2 /Resource/Image

The sub-folder `Image` contains all image files referenced by the project:

File	Description
`<IMAGENAME>.<SUFFIX>`	Image file referenced by the project.
`<IMAGENAME>.dta`	Streamed image.
`<IMAGENAME>.c`	Simple C arrays of streamed image files which are not managed on external memory.

4.1.2.3 /Resource/Text

The sub-folder `Text` contains all text defined in the project:

File	Description
`APPW_Language_<n>.txt`	Text file(s), one for each language.
`APPW_Language_<n>.c`	Simple C arrays of text file(s) if not managed on external memory.

Chapter 5

User interface

The following chapter will give an overview on the user interface of the AppWizard. The user interface of the AppWizard consists of a menu bar and a couple of windows.

The following windows exist:

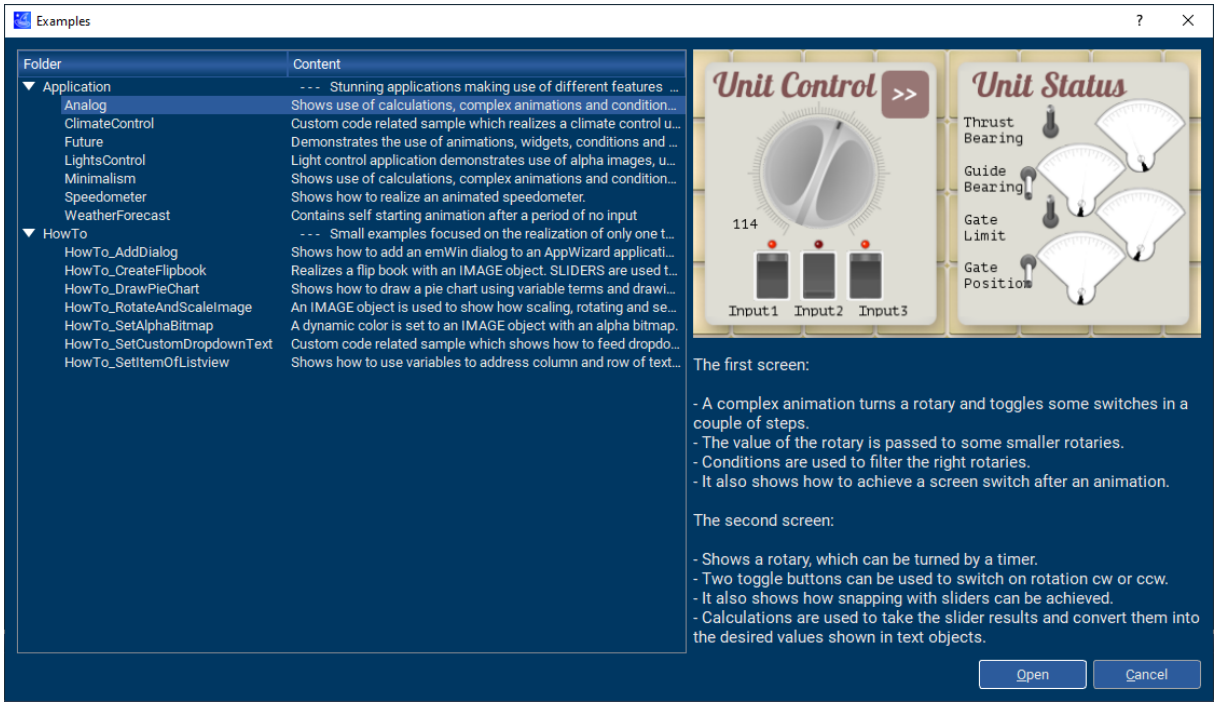
- ’Editor’ window (center/top)
- ’Interactions’ window (center/bottom)
- ’Add objects’ window (left/top)
- ’Hierarchic tree view’ window (left/bottom)
- ’Properties’ window (right)
- Quick access buttons for text, fonts, images and variables at the lower left edge

5.1 Menu bar

It consists of the following items:

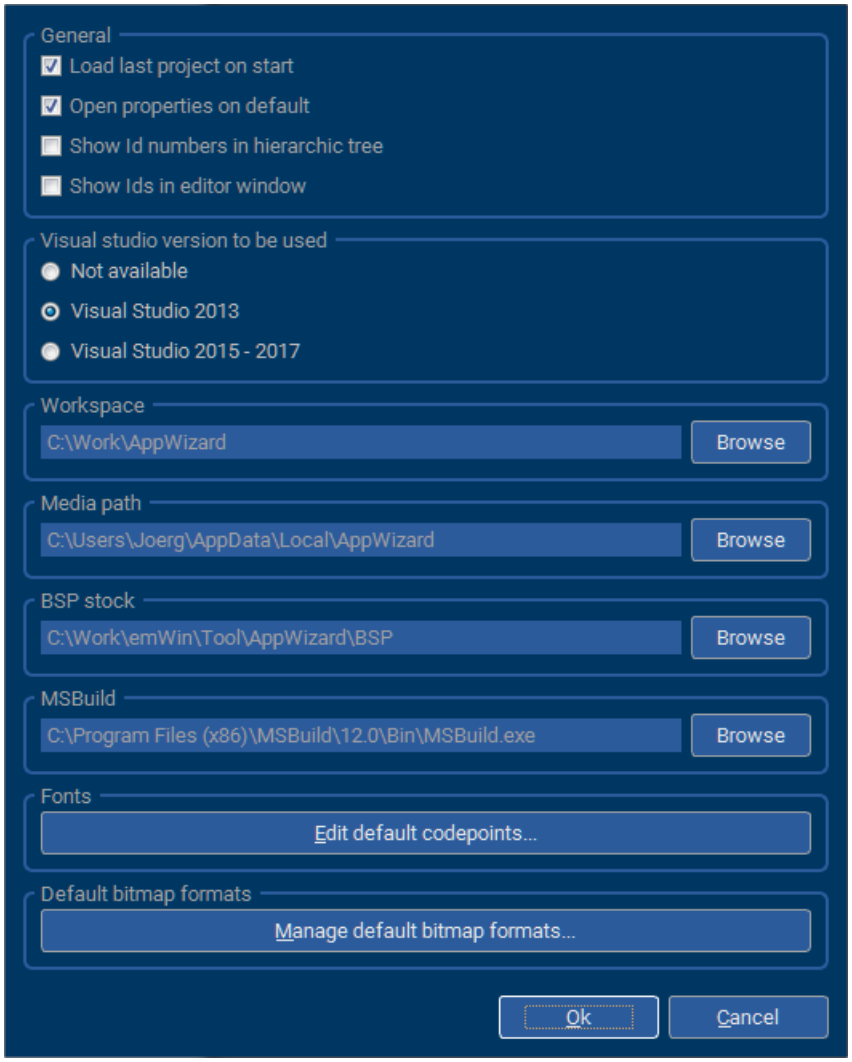
- File (New project, Open, Save, Save As, Export & Save, Rename, Close project, Open example, Import BSP, Exit, Recent files)
- Edit (Undo, Redo, Cut, Copy, Paste, Delete, Select all, Preferences)
- Project (Edit options, Play F5, Start Simulation F6, Start Spy)
- Resource (Edit Text, Edit Fonts, Edit Images, Edit Variables)
- Help (About, Open User Guide, Open Quick Start Guide, Open Release Notes)

5.1.1 File / Open example



The example browser allows browsing through the available example projects. On the left side it shows a tree with the available examples and a short description. When selecting an example it shows a screenshot and a more detailed description on the right side. When clicking the ’Ok’-button a dialog for selecting the parent folder for the sample is shown. After selecting the parent folder the example will be extracted and immediately loaded into the AppWizard.

5.1.2 Edit / Preferences



The preferences dialog has the following options:

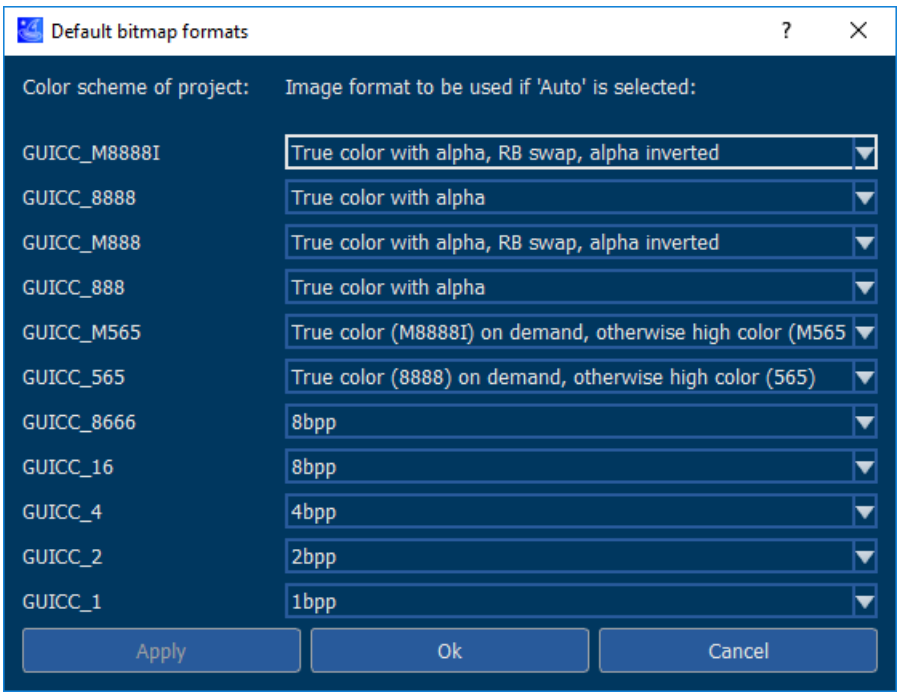
- **Load last project on start:** Enables loading the last used project after starting the application.
- **Open properties on default:** Opens all object properties by default.
- **Show Id numbers in hierarchic tree**: Shows the Window Manager Id next to the AppWizard Id.
- **Show Ids in editor window**: Displays the object's ID in the editor window.
- **Simulation project to be used**: Select which Visual Studio simulation project should be used when pressing F6. If you are using Visual Studio 2010-2013 select the first option. If using a later version select the second option.
- **Workspace:** Path to be used to open examples.
- **Media path:** Path to external media.
- **BSP stock:** Sets a path, where custom BSPs are located.
- **MSBuild:** MSBuild to be used for AppWizard SPY.
- **Edit default codepoints...:** Editing the default range of code points to be used for new fonts.
- **Default bitmap formats:** Set the default bitmap format to be used when bitmaps use the 'auto' format.

Default codepoints

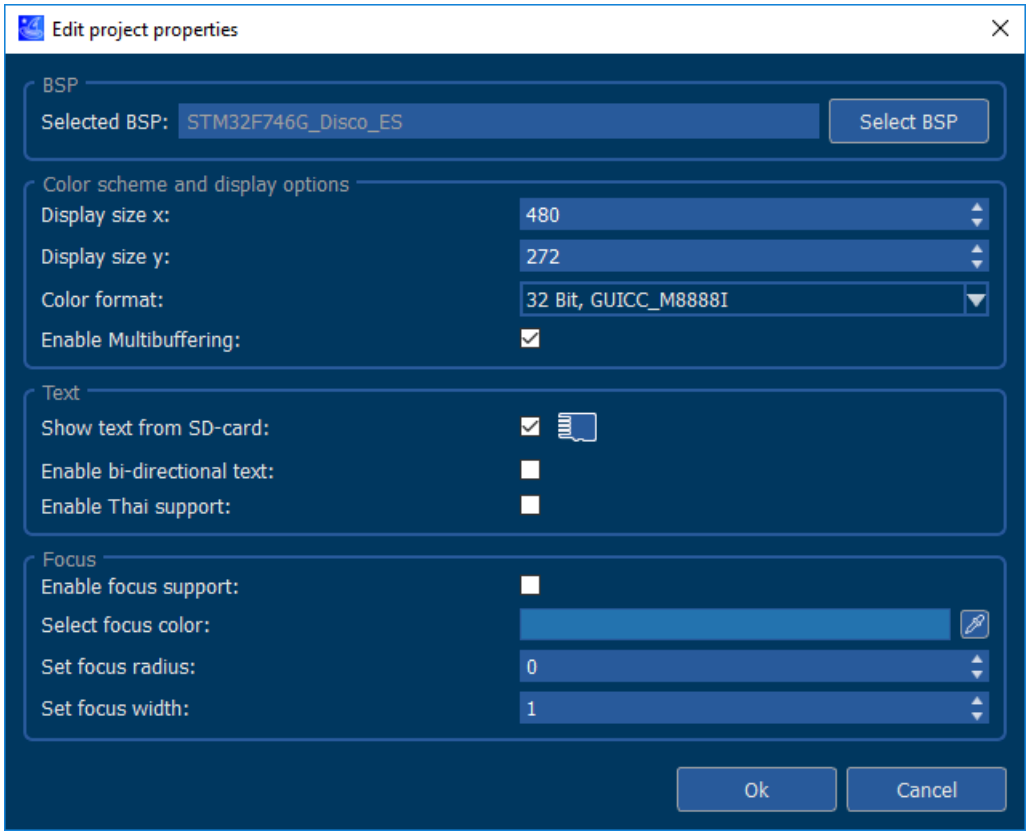
The user can define which characters should be included in newly created fonts. A more detailed description and explanation on the codepoint dialog can be found under *Font resource window* on page 45.

Default bitmap formats

The user has the option to select a default bitmap format which will be used by the ‘Auto’ format option when a bitmap is added to the project.



5.1.3 Project / Edit options



The project options dialog has the following options:

BSP

- **Selected BSP:** Desired BSP for target hardware.

Color scheme and display options

- **Display size x:** Horizontal display size.
- **Display size y:** Vertical display size.
- **Color format:** Desired color format.
- **Enable Multibuffering:** Option whether or not multi buffering should be enabled in the project. This option is only available, if the selected BSP supports multi buffering.

Text

- **Show text from SD-card:** Option to outsource the texts to external media.
- **Enable bi-directional text:** Enables support of bi-directional texts.
- **Enable Thai support:** Enables support of Thai script.

Focus

- **Enable focus support:** Enables the ability for objects to receive focus.
- **Select focus color:** Color used for drawing the focus rectangle.
- **Set focus radius:** Radius used for the corners of the focus rectangle.
- **Set focus width:** Line width of the focus rectangle.

In case of using a BSP display size and color format are fixed and come from the BSP.

Focus rectangles are explained later on in the chapter *Object focus* on page 102.

5.2 Editor window

The editor window shows the currently selected screen by drawing it directly with emWin. That makes sure that "what you see is what you get". Additionally each object has a slightly semi-transparent frame which ensures that also invisible objects give a slight optical feedback.

To be able to place graphical objects a screen has to be created at first. That is done by clicking on the screen icon in the 'Add Object' window left to the editor window. Placing controls is done in the same way. Simply drag an element from the 'Add Object' window onto an existing screen or window object in the editor window.

Independent horizontal and vertical placing

Horizontal and vertical placement of an object can be defined independently. The behavior of each axis can be defined by either a relative position and a size or two relative positions. 'Relative' means relative to its parent or relative to a sibling. That makes it possible to create screens or windows which are self-adjusting when changing the parent's or sibling's placement.

Hierarchical structure

Window elements are used to achieve a hierarchic object structure. They can be placed within a screen or an already existing window. When placing objects on a window the position of those objects can be changed by simply moving or animating the window.

Snapping

Snapping is used when moving objects with the mouse. Edges and center points of existing objects are used for snapping. When aligned with other objects the editor generates optical feedback by highlighting the according object and/or center line.

Selecting objects

Left-clicking selects the first object under the clicked coordinate. A selected object has nine drag points for modifying the coordinates, one on each edge, one on each corner and one in the center point.

With the <CTRL> key pressed multiple objects can be selected. Selected objects are getting joint into a selection group. In that case the drag points are getting placed on to the rectangle surrounding the selection group. Rectangle selection can be done by clicking with the left button in an empty area of the editor window, holding the button pressed and dragging the rectangle with the mouse. When releasing the button the objects within the rectangle will be selected.

Positioning

Objects and groups can be positioned by dragging them with the mouse. The drag points are used to modify the geometry of an object. The property window on the right hand side can also be used to modify the size, coordinates and relations of objects.

Concatenating object positions

To concatenate object coordinates, one of the edge drag points of an object has to be connected to the edge of another object using the right mouse button. This will result in when moving the object the other object was connected to, both objects will be moved synchronous on the axis of the drag point.

A concatenated object position can be cleared by selecting any of the nine positioning options. These options are explained under *Positioning logic* on page 37.

Copy/Paste

Single objects, groups or complete screens can be copied and pasted by either using the keyboard or the menu bar. IDs of copied objects are extended with the suffix '_Copy'. The AppWizard makes sure that the generated Ids are unique within the current screen.

Zooming and panning

The content of the editor window can be easily zoomed by using the '+' or '-' button, the '+' or '-' key or the mouse wheel in combination with the <CTRL> key.

The zoom level can be reset by pressing the '1:1' button.

The content of the editor window can be moved by panning, which is done by pressing the <SPACE> bar and moving the mouse while pressing the left mouse button.

Play mode

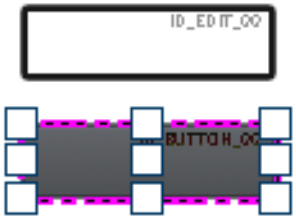
The play button in the upper right corner of the editor window opens the play window, which allows a quick check of the current application.

More information about this window can be found in the chapter *Play window* on page 42.

Object IDs

As mentioned earlier in this chapter, when the option is activated in the Preferences dialog, the object IDs can be shown in the editor window.

The ID is displayed in the upper right corner of an object. The IDs of objects are only shown when any of them is selected and is not shown at all in Play mode.



5.3 Property window

The window on the right shows the object specific properties. It consists of four areas (top to bottom):

- Id, position and size
- Positioning logic
- Coordinate and size modification
- Object specific area

5.3.1 Id, position and size

The top area shows the selected object's Id, which can be edited. Below that it shows the coordinates and size of the object.

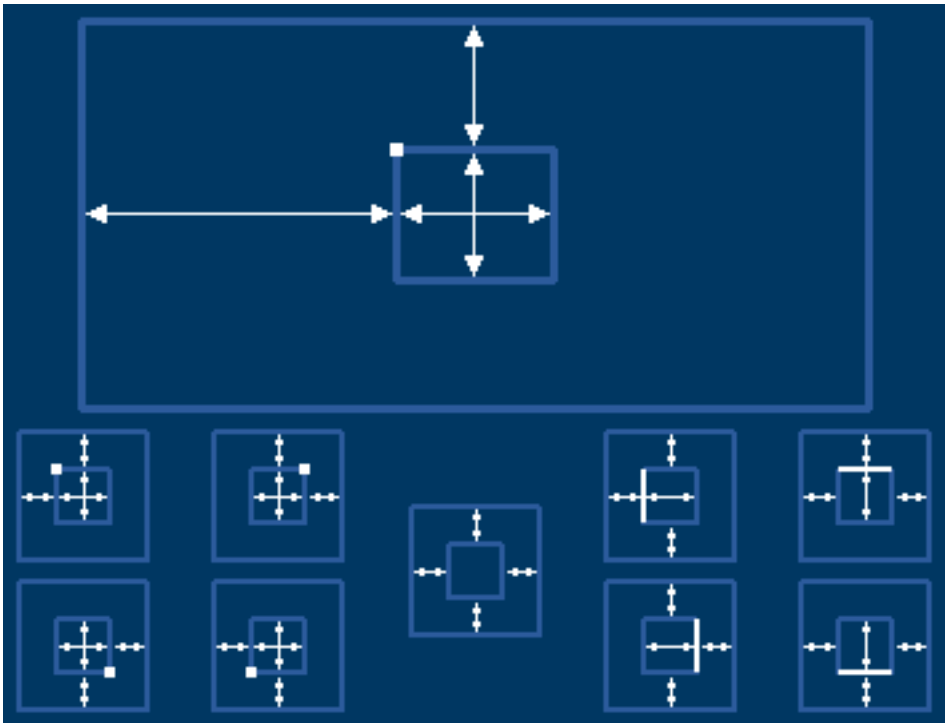
Placing details can be modified in the 'Positioning details' area below.

5.3.2 Positioning logic

The rectangle of a simple emWin window is defined by its upper left position and its X- and Y-size. To be more flexible with this, the AppWizard supports more options.

One option for example is specifying the coordinates of one of the edges and the objects X- and Y-size. That is similar to a normal emWin window except the option of using any edge and not only the top/left coordinates.

Each coordinate can be relative to an existing edge of the parent or any other sibling. For example, the top coordinate can be relative to the parent, the Y-size fixed and right and left coordinates relative to the parent.
The Y-position of the next object can then be relative to the object above and so on. This mechanism makes it possible to generate screens which are self-adjusting when the parent's size or orientation changes.
To remove a concatenated positioning logic, one of the nine options for positioning logic has to be clicked.



The top of the area shows the positioning logic of the selected object. Dimension lines are used to show coordinate and size definitions. In case of coordinates relative to existing siblings it shows the Id of the according sibling.

There are nine positioning options to choose from:

Positioning option	Description
	Top and left coordinate relative to parent. Width and height defined by given value.
	Top and right coordinate relative to parent. Width and height defined by given value.
	Bottom and right coordinate relative to parent. Width and height defined by given value.
	Bottom and left coordinate relative to parent. Width and height defined by given value.
	Top, left, bottom and right coordinate relative to parent.
	Top, left and bottom coordinate relative to parent. Width defined by given value.
	Top, left and right coordinate relative to parent. Height defined by given value.
	Top, right and bottom coordinate relative to parent. Width defined by given value.
	Left, bottom and right coordinate relative to parent. Height defined by given value.

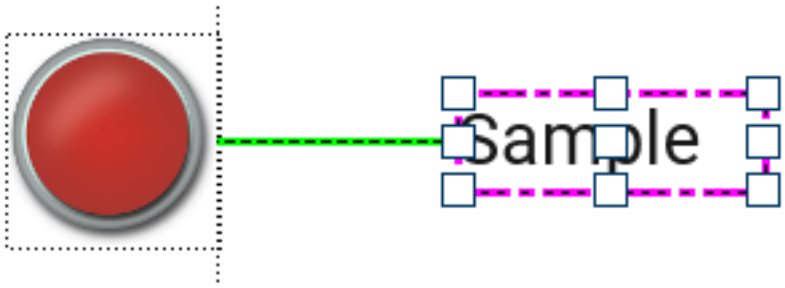
Note

The positioning logic can be changed at anytime.

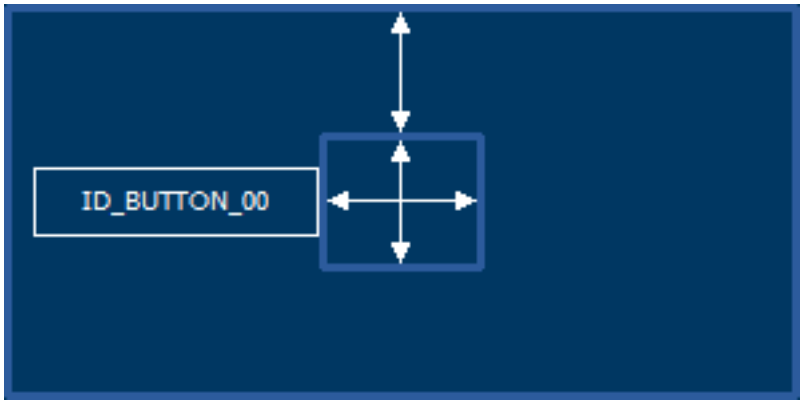
Example

In this example, the Text object's X-position shall be relative to the X-position of the Button object. To do that, one of the Text object's contact points on the X-axis has to be right-clicked. After clicking, a line appears that has to be moved to the Button's X-axis contact point.

When the line appears in a green color, the operation is valid and will be applied when releasing the right mouse button.



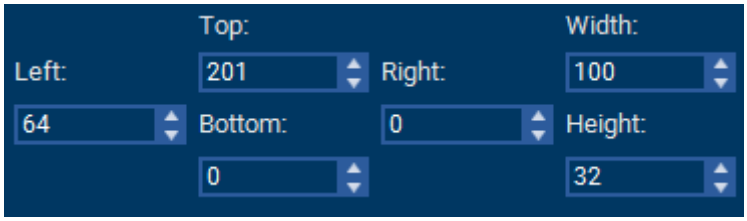
When selecting the Text object, the positioning logic property shows that its X-axis is dependent on the Button object.



To remove this positioning property, the user simply has to select one of the nine positioning options that are explained above.

5.3.3 Positioning details

This section allows setting up top, left, bottom and right coordinates and X/Y size by spin boxes, depending on the selected positioning option.



5.3.4 Object dependent details

Each object has its own properties that can be edited, they are located below the ’positioning details’ section in the ’object dependent details’ section.

Depending on which object is selected, its properties are shown. To see a list of all existing object properties, see the chapter *Object properties* on page 65.

Editing properties

Each property is shown with a text and an arrow button to the left.

To set or define a property, the arrow button should be clicked.

It opens a configuration area to be able to specify the property details. An existing property can be closed with the arrow button.

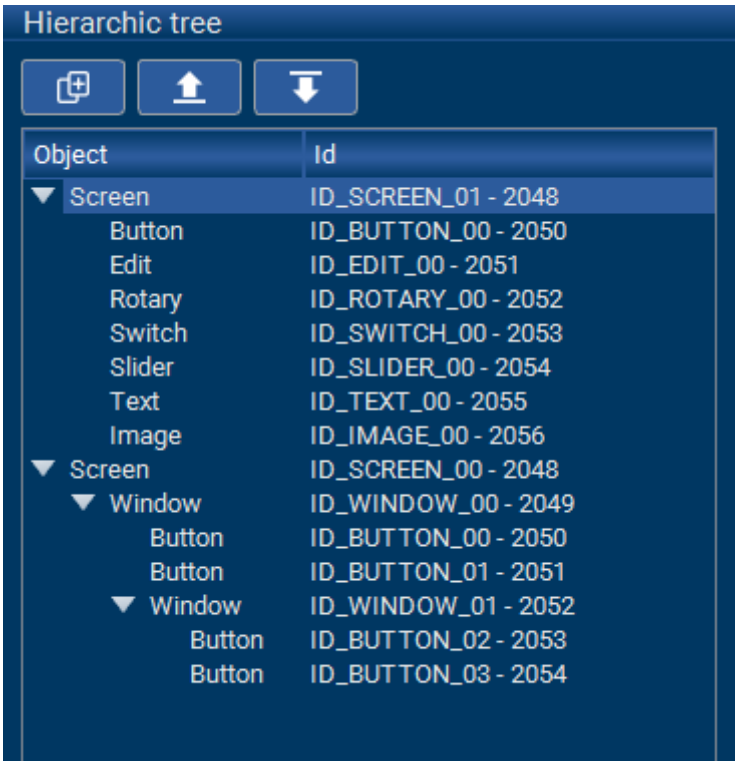
To delete an existing property, the **X** button on the right side has to be clicked.

Clicking the arrow button of an existing property opens and closes the according property definition area. The preferences dialog (Edit ➔ Preferences) allows the option to open all existing property areas per default.

Properties like text, fonts or images open the according resource management and selection dialog.

5.4 Hierarchic tree view

The hierarchic tree view gives a quick overview about the currently existing objects. It allows changing the relative position of siblings per drag and drop. Selecting an object within the tree view also selects the object in the editor window.



Duplicating objects

When clicking the duplicate object button, a copy of the selected object is inserted into the same level of the hierarchic tree.

Moving objects

Clicking the ’Move up’ button will move the currently selected object upwards.

Clicking the ’Move down’ button will move the currently selected object downwards.

> **Note**
>
> The ’Move up’ and ’Move down’ buttons can only move an object within their level of the hierarchic tree. This means an object can not be moved to another parent. To move an object to another level/parent, it has to be cut out and pasted to the new location by right-clicking it.

Editing object IDs

The ID of a selected object can be edited when the ENTER key is pressed.

5.5 Play window

The play window shows the user a ’running’ version of the current project application.

It can be opened by clicking on the play button in the upper right corner of the editor window and closed by pressing the escape key. It may also be opened and closed by pressing the <F5> key.

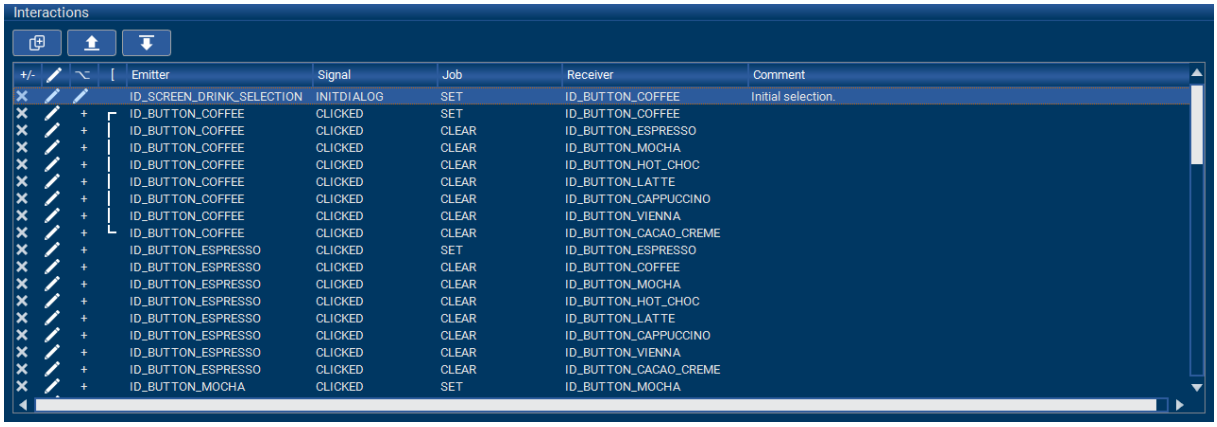
When opening the play window, a modal dialog with the resulting interactive application will be shown.

Limitations

There is one limitation to the play mode, as the AppWizard does not compile any C code, the play mode does not include any code by the user added to interactions.

5.6 Interaction window

The interaction window shows a list of all interactions associated with the selected screen. Each interaction has its own emitter, signal, job and receiver.



Creating a new interaction

Creating a new interaction is done by pressing the **+** button at the end of the list. To learn more details on how to create new interactions, see the *Introduction* section of the Interaction chapter.

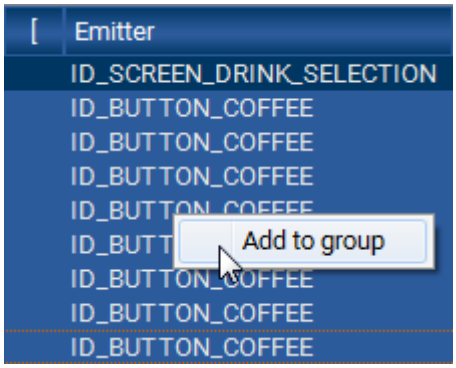
After specifying the receiver a dialog occurs which allows it to specify job dependent data and/or user defined code of the interaction slot. Clicking the pen opens a dialog for editing those parameters.

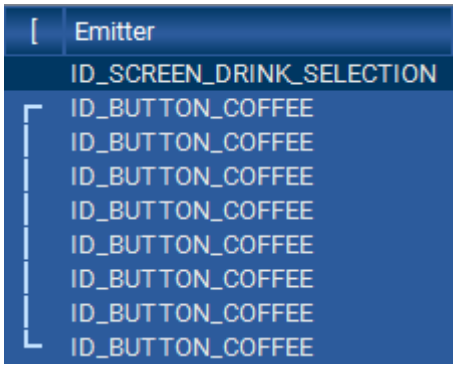
Removing an interaction is done by clicking the **X** button in the first column.

Grouping interactions

Interactions may also be grouped together. This makes sense if there is a large number of interactions present in the application and a more overseeable structure is desired. Grouped interactions are marked via the line in the "[" column.

To group interactions, select the interactions, right click them and select "Add to group". As demonstrated below, after a group has been created, all the interactions between the start and end of the line are in that group.





Now, when one item of the group is clicked, all group items are selected. To delete the group, right click and select "Clear group".

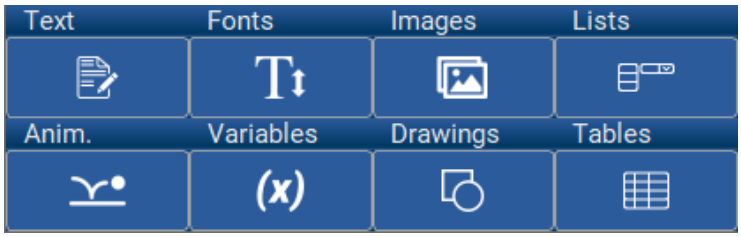
Adding a comment

A comment can be added for each interaction. To do so, simply double click the empty area next to the interaction in the "Comment" column.

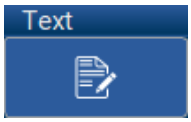
5.7 Quick access buttons

In the lower left corner there are six quick access buttons. Clicking on one of the buttons will open the corresponding window that allows management of either texts, fonts, images, variables, animations or drawings.

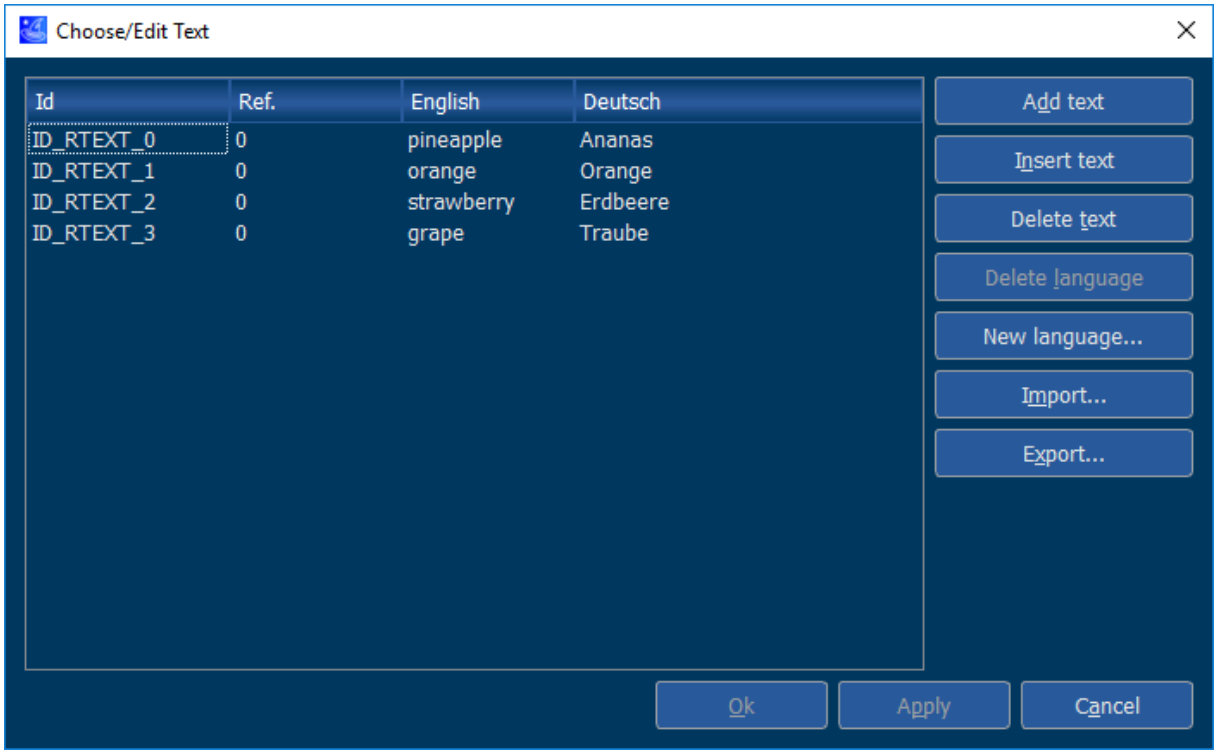
For more information about managing resources, see the chapter *Resource management* on page 54.



5.7.1 Text resource window



The text resource window makes it possible to save texts in multiple languages. The order of the languages may be changed by using drag and drop on the column header.



Each text has its own ID, that can be assigned to objects. The **Ref** column states how often the text is referenced in any objects.

Add texts or languages

Before adding texts, you need to have added at least one language first. New languages/columns are added via the **New language** button.

New texts are added via the **Add text** or **Insert text** buttons. They can be edited by clicking on the corresponding field.

Texts can be deleted via the **Delete text** button, but only when they are not being referenced by any objects.

5.7.1.1 Export and import texts in the CSV format

It is also possible to export the texts to a CSV file or import a CSV file.

CSV file rules

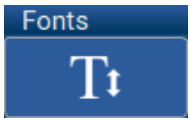
In order for the import of a CSV file to work, the following rules have to be observed:

1. The file must be UTF-8 encoded.
2. The first row has to contain the names of languages.
3. Each following row contains a text item in the given number of languages.
4. Each language string must be enclosed by quotes, e.g. “pineapple”.
5. All values are to be separated by commas.
6. After the last value there should be no comma.
7. Optional: A desired ID can be added before the first value in a row. The ID is to be separated with a comma from the values, but it must **not** be enclosed by quotes.

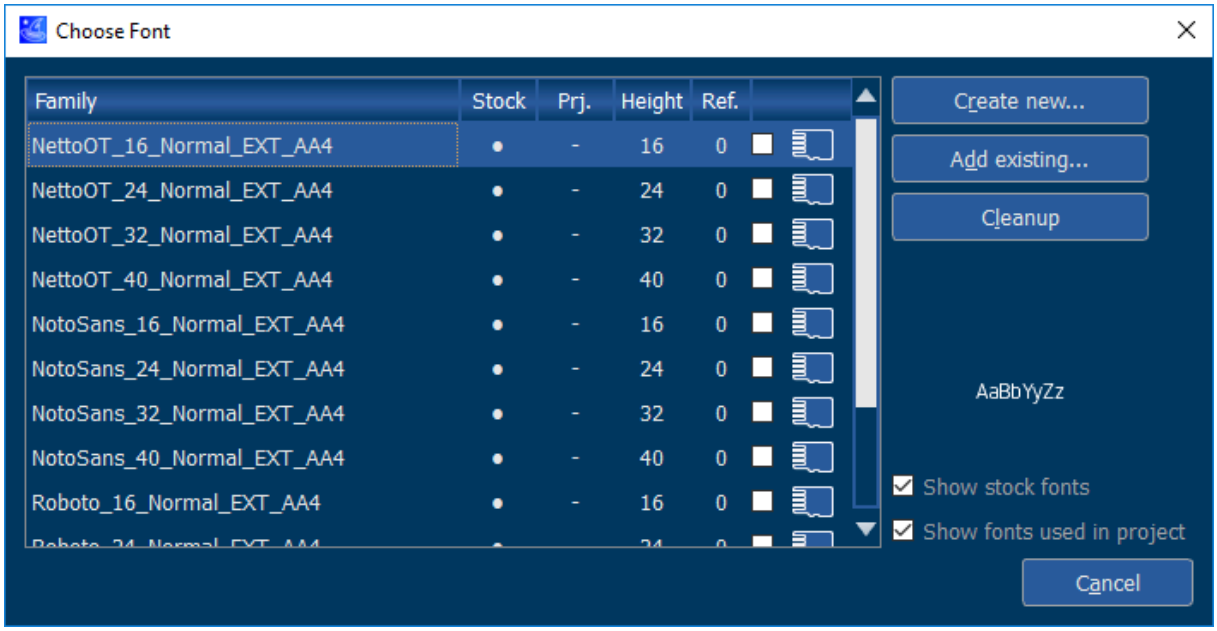
An example CSV file with all the rules applied could look like this:

```
"DE","EN","FR"
ID_RTEXT_0,"German text 0","English text 0","French text 0"
ID_RTEXT_1,"German text 1","English text 1","French text 1"
ID_RTEXT_2,"German text 2","English text 2","French text 2"
ID_RTEXT_3,"German text 3","English text 3","French text 3"
```

5.7.2 Font resource window



The font resource window allows the user to manage fonts.

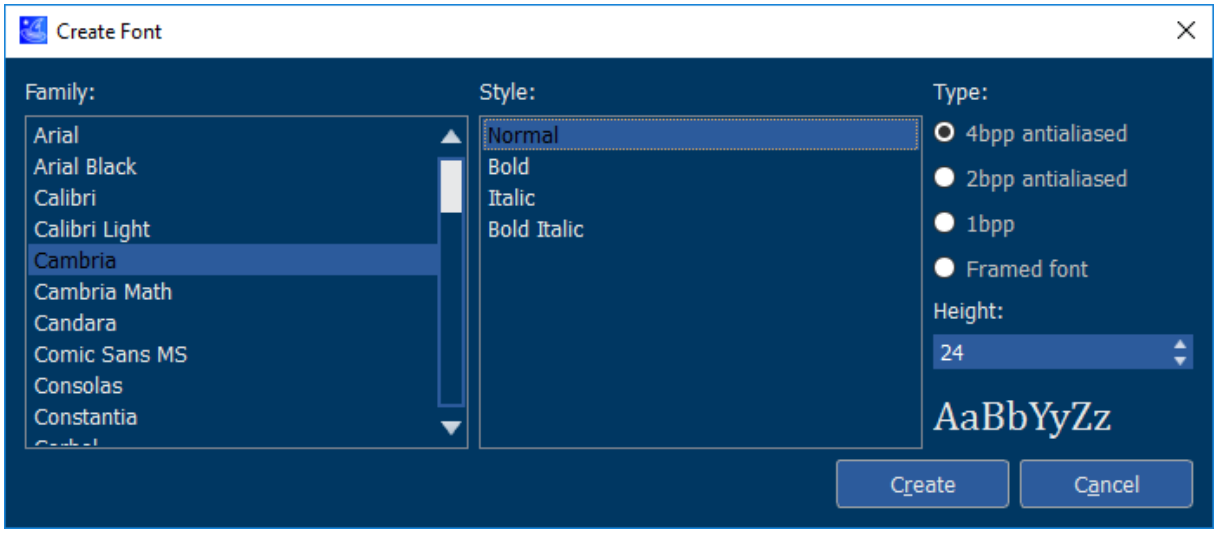


The table shows whether the font is a stock font and/or is used in the project. It also shows how often fonts are referenced in any objects and the height of the font. By ticking the checkbox next to the SD card, the font can be marked as an external resource.

> **Note**
>
> Only XBF files created with the AppWizard can be used!

Create new fonts

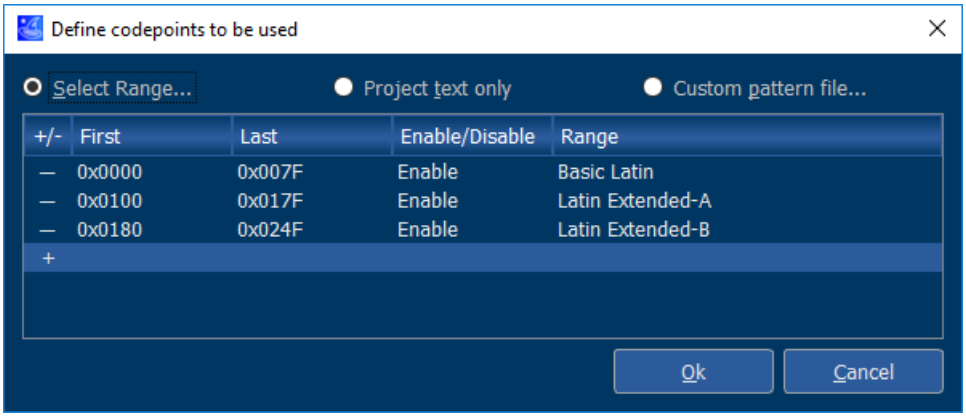
Clicking on **Create new...** allows the user to add a new font from the local installed fonts. When clicking the button, a window similar to that in the FontConverter is opened. The user has to select which font should be added, in which style and height and also select the anti-aliasing level.



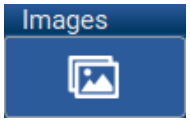
To optimize memory footprint, the user may define which characters should be present in the font. This can be done by clicking on **Codepoint range...** and either selecting a range of characters, keeping only the characters that are used in all project's texts or by parsing a pattern file.

By default, a range of characters is used for creating a font. The default range of enabled characters is:

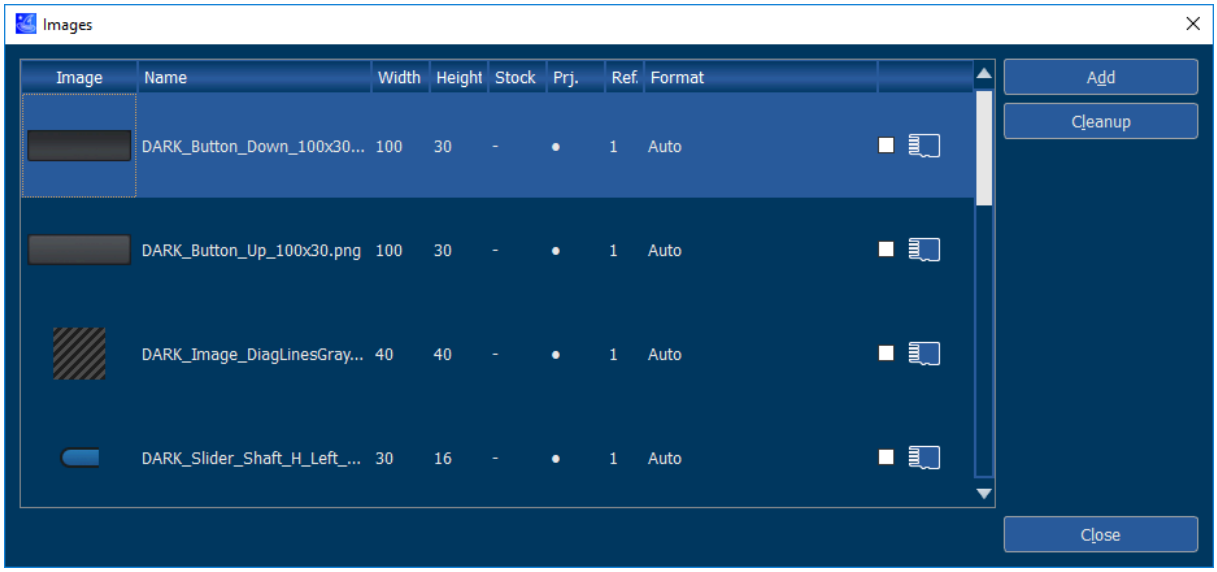
```
0x0000      to      0x007F
0x0100      to      0x017F
0x0180      to      0x024F
```



5.7.3 Image resource window



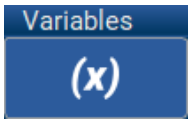
The image resource window allows the user to manage images.



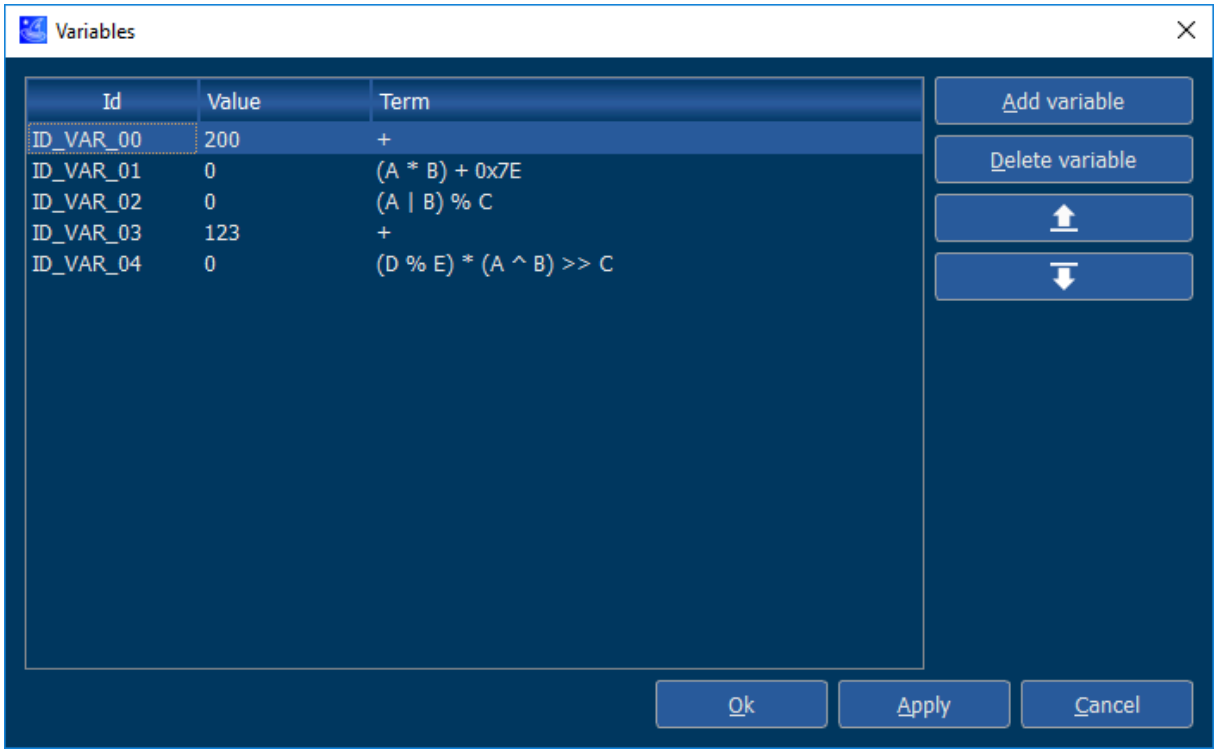
The window gives an overview of all the images used in the project, showing also their dimensions, the bitmap format applied to the image and how often it is referenced in other objects.

As in the other management windows, the user can choose if the image should be marked as an external resource. The **Add** button can be used to add another image from your local disk.

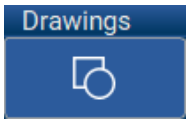
5.7.4 Variable resource window



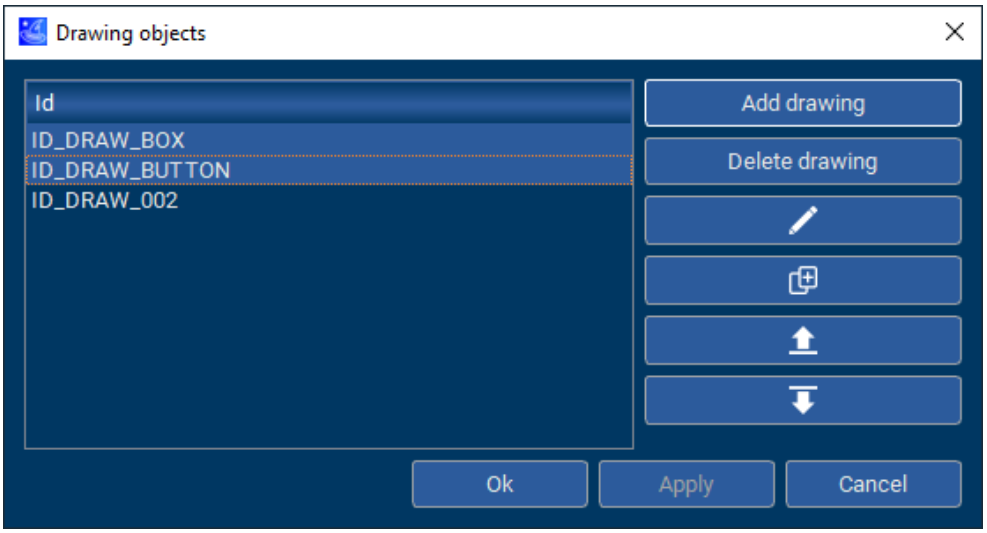
The variable window lets the user add or remove variables. More explanation on what variables are used for can be read in the chapter *Variables* on page 205.



5.7.5 Drawings window

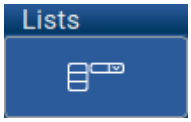


The drawing window allows the user to define custom drawings that can be used in the application.

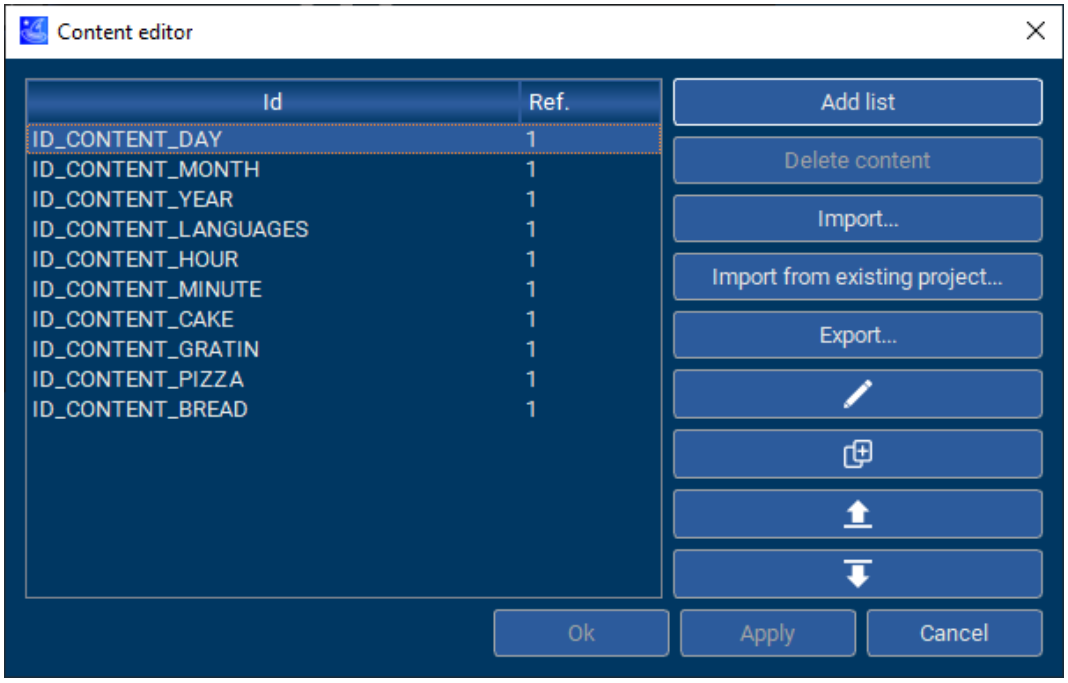


More detail about drawings can be found in the chapter *Drawings* on page 215.

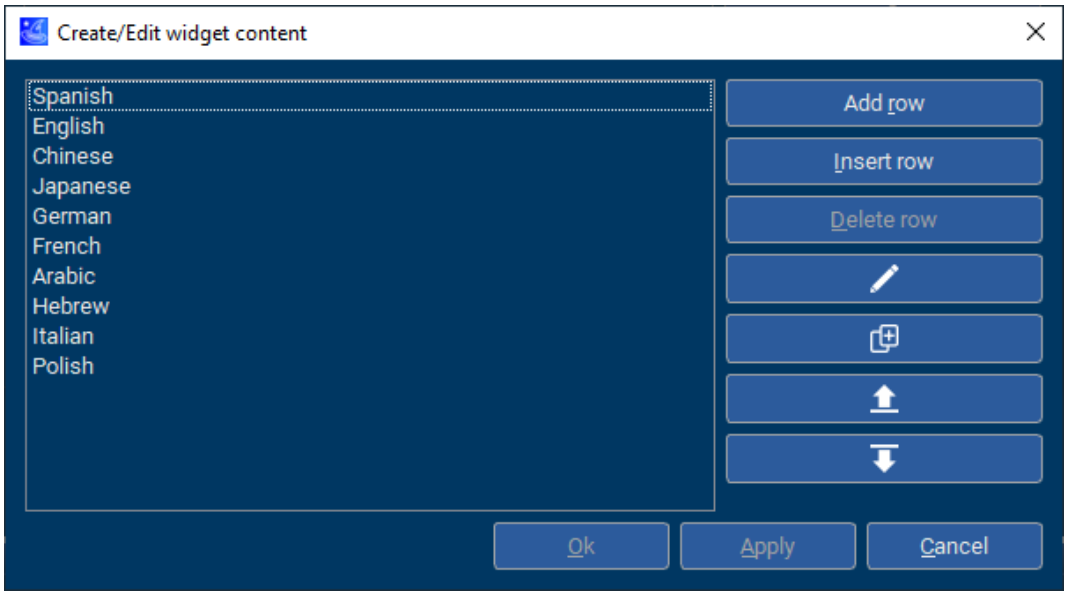
5.7.6 Lists window



The list window allows the user to add content objects in the form of lists. These lists can be used in the application by list-based objects such as Listbox, Dropdown or Wheel.



When clicking the edit button, the list can be edited and rows can be added, deleted or moved.



5.7.6.1 Importing and exporting of content

To import and export `.txt` or `.csv` files as lists or tables, click the corresponding quick access button and click on the (**Import...** or **Export...** button. Note that for importing, multiple files can be selected and imported at once.

TXT file import and export

The TXT format is only available for lists and the format being applied is one line in the file equals one row in the final list.

CSV file import and export

The import and export of `.csv` files is available for lists and tables. Below an example that demonstrates the CSV format being applied:

```
"Header column 0","Header column 1","Header column 2"
"Row 0, Column 0","Row 0, Column 1","Row 0, Column 2"
"Row 1, Column 0","Row 1, Column 1","Row 1, Column 2"
"Row 2, Column 0","Row 2, Column 1","Row 2, Column 2"
```

The following CSV rules are observed:

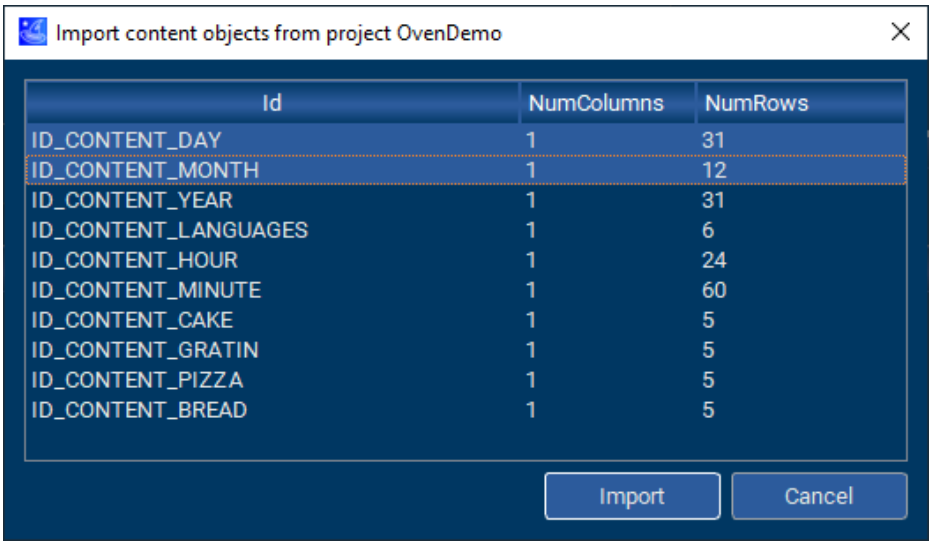
- Each string is encapsuled by quotes.
- Each value is separated by a comma, except at the end of a line.
- Each line in the text file represents a row in the table.
- The first line in the CSV file is always the header line (this is applied to lists as well).

> **Note**
>
> The CSV import and export is intended to only be used for the texts of a table. This means the tables' properties, such as column widths or alignments will not be written to the CSV file. These properties are only imported when importing a table from an existing project (see below).

Importing tables and list from other projects

To import lists or tables from other projects, click the button **Import from existing project...**.

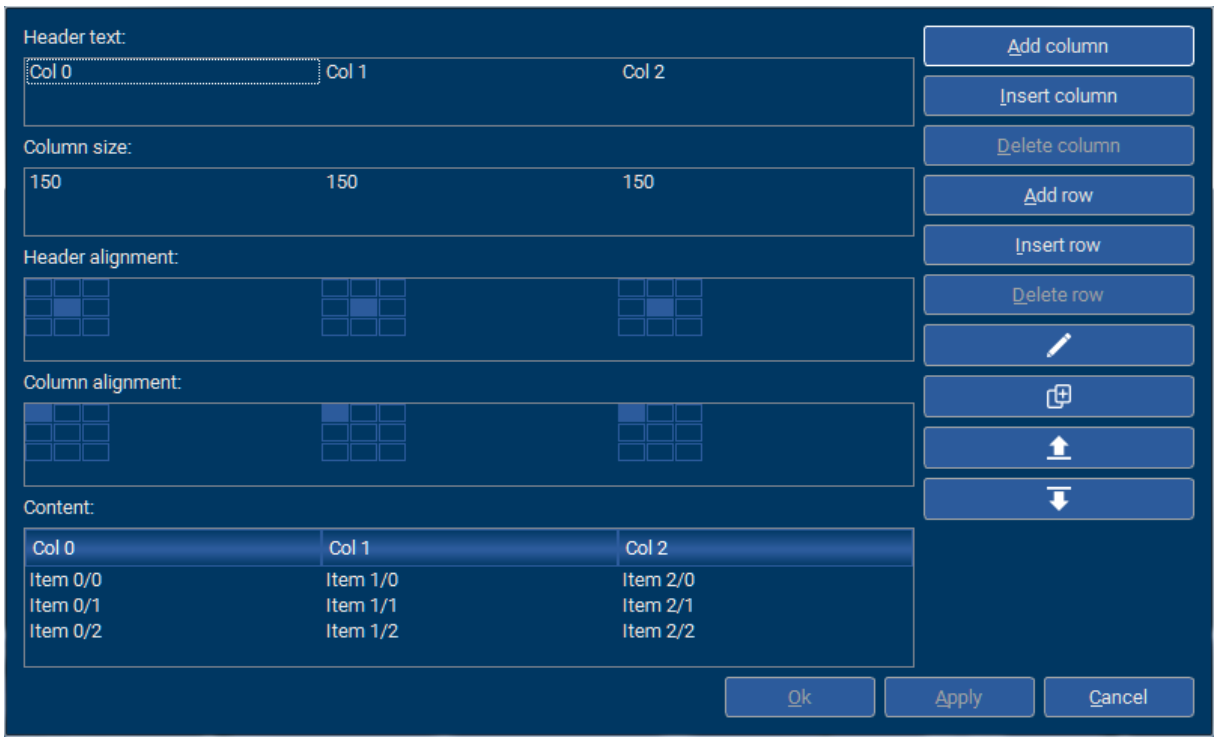
If a suitable project file has been selected, the IDs of the existing content objects will be shown. Multiple content objects can be selected for importing.



5.7.7 Tables window

The table window allows the user to add content objects in the form of tables. Tables can be used in the application by table-based objects. As of now, the only table-based object is the Listview.

The table window allows the definition of the actual table of data, but also the column sizes and alignment and the content of the header line of the table.



A more detailed explanation on how the table window works can be found under *Content* on page 70.

5.8 Starting the simulation project

By selecting menu entry `Project` ➔ `Start Simulation` or by pressing `F6` the AppWizard starts automatically the simulation project which is placed within the AppWizard project directory. Depending on the selection made under ‘Preferences’ either the VS2013 or VS2015 project gets started.

5.9 AppWizard SPY window

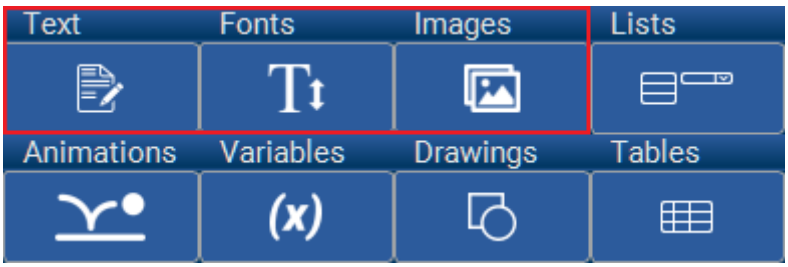
The AppWizard SPY window can be opened through the menu entry `Project` ➔ `Start Spy`. Alternatively, the `F7` key can be pressed to open the window.

More details about the features and usage of AppWizard SPY can be read in the chapter *AppWizard SPY* on page 251.

Chapter 6

Resource management

The AppWizard manages all text, fonts and images required for the application. The user gets completely rid of additional resource management like creating font files with the font converter, image files with the bitmap converter or text data to be used in the project.



Per default resources are compiled and linked into the application. For systems short on ROM, large resources or resource data which should be changeable at runtime, those resources can be managed from SD card with the file system included in the BSP.

Optionally the AppWizard manages the content of the SD card which needs to be available at runtime.

Please refer to the *Creating custom BSPs example* to learn how BSPs with or without a file system can be used.

Furthermore, there are other types of data that can also be seen as kind of ’resources’ in a broader sense. These are:

- Tables and lists
- Animations
- Variables
- Drawings

6.1 Stock resources

The AppWizard comes with a bunch of different stock fonts and images that are ready for use for any application.

> **Note**
>
> Note that as with any resources, any stock resources that have been used are saved in the exported project as well!

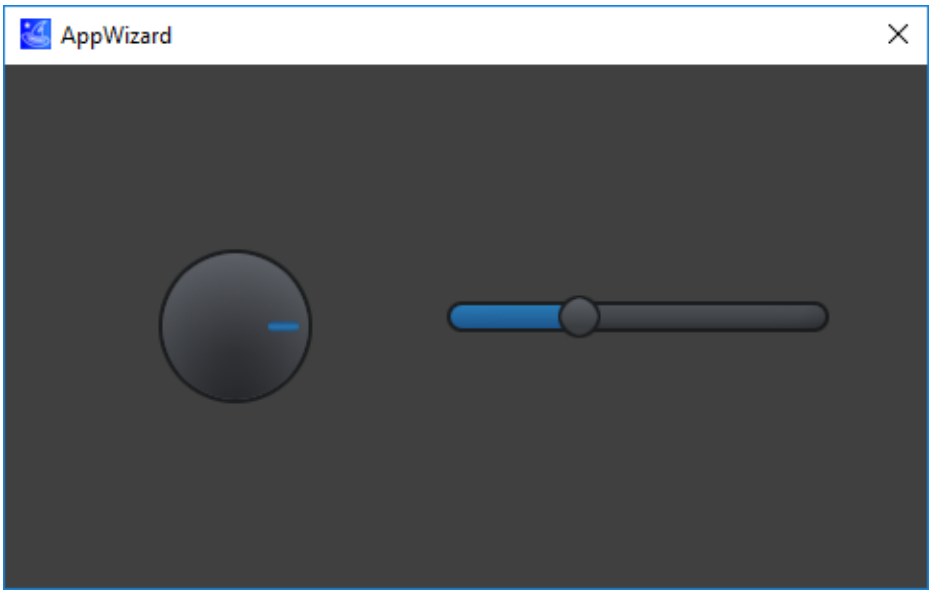
Stock fonts

The AppWizard by default supplies a few fonts, each in different sizes. All stock fonts use 4bpp anti-aliasing.

- `NettoOT_16_Normal_EXT_AA4`
- `NettoOT_24_Normal_EXT_AA4`
- `NettoOT_32_Normal_EXT_AA4`
- `NettoOT_40_Normal_EXT_AA4`
- `NotoSans_16_Normal_EXT_AA4`
- `NotoSans_24_Normal_EXT_AA4`
- `NotoSans_32_Normal_EXT_AA4`
- `NotoSans_40_Normal_EXT_AA4`
- `Roboto_16_Normal_EXT_AA4`
- `Roboto_24_Normal_EXT_AA4`
- `Roboto_32_Normal_EXT_AA4`
- `Roboto_40_Normal_EXT_AA4`

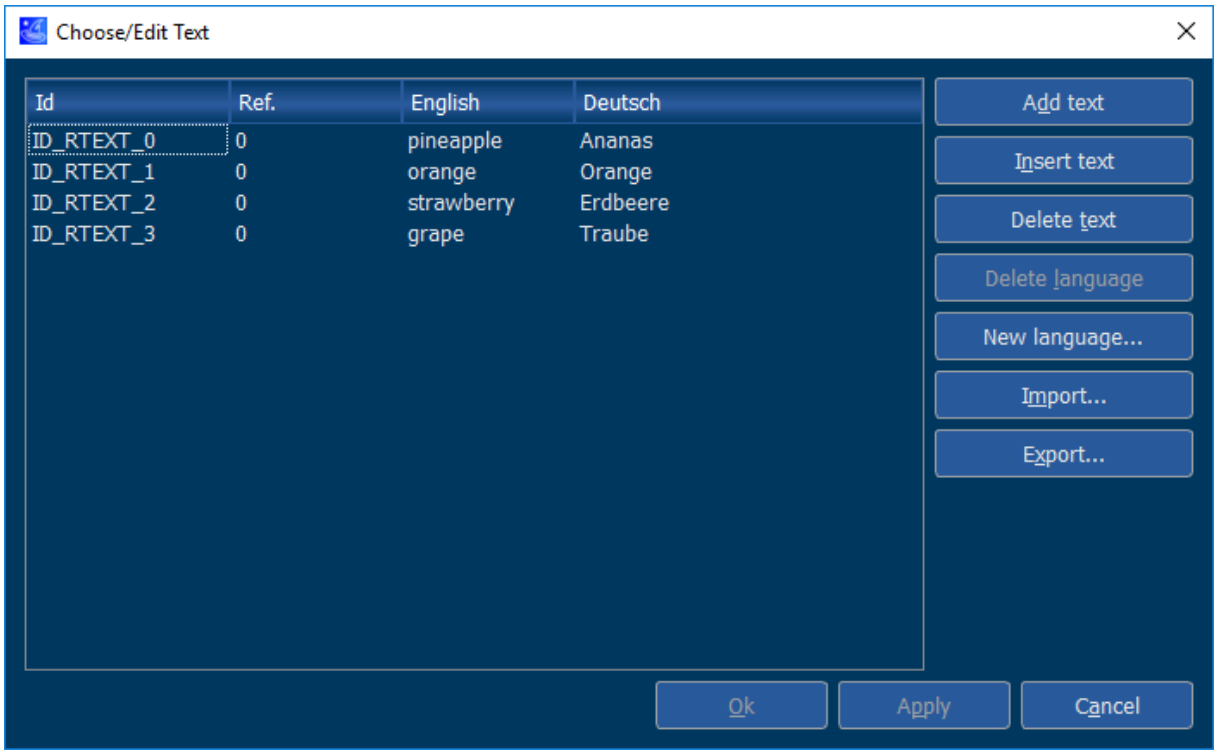
Stock images

For each object which can make use of a bitmap, the AppWizard offers a set of images. For example, bitmaps for a Rotary and its marker, or a thumb and shaft bitmap for a Slider object.



6.2 Text management

A text input dialog allows entering text in multiple languages. Text usage is based on using IDs instead of using strings directly. Text access within the application is realized by using text IDs. In combination with emWin's language module it becomes quite easy to switch between languages. More details about editing text can be found under *Text resource window* on page 44.



Linebreaks

Linebreaks in text can be achieved with the sequence `\n` in the text strings. A `\n` will be replaced by the AppWizard with a newline.

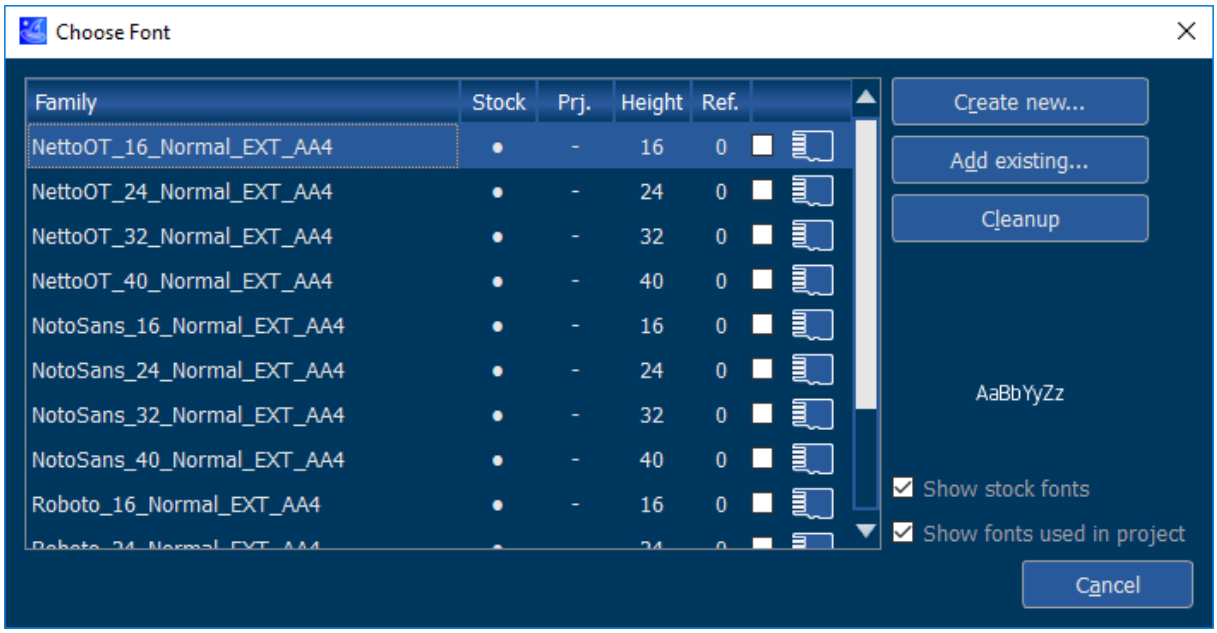
Managing text from SD card

The project property dialog has the option to enable text management directly from SD card. In that case the text is not compiled and linked with the application code.

When exporting the project, the text will then be stored in the specified media path in the directory `<Mediapath>\Resource\Text`.

6.3 Font management

The AppWizard comes with a small set of default fonts in form of XBF font files and the option for creating new fonts. The resource path contains all fonts referenced by the project. A font management dialog shows all available fonts in the project.



The following options exist:

- Show stock fonts
- Show project specific fonts

The dialog shows the following columns:

- Font family
- Stock font, means that the font is located in font stock
- Project font, means that the font is located in resource folder of the project
- Height in pixels
- Number of references
- Check box to specify SD card management for that font

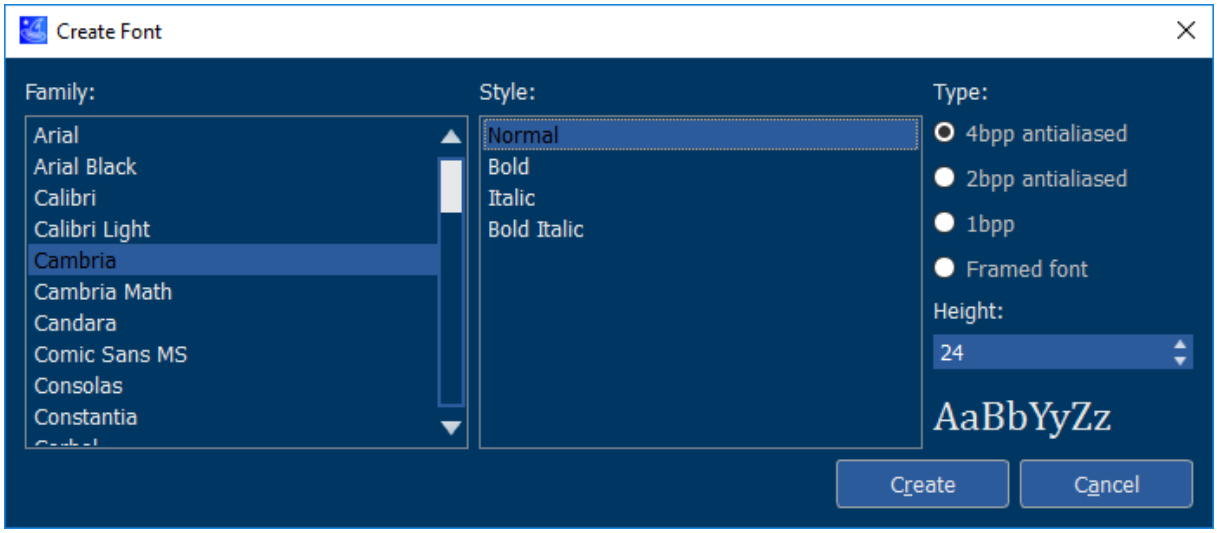
When compiling the project all referenced fonts are compiled and linked with the application per default. SD card management excludes the font from compiling and linking with the application. Those fonts are managed from SD card directly without using addressable ROM for the content of the font. When exporting the project, these fonts will be saved in a directory in the specified media path, that is `<Mediapath>\Resource\Fonts`.

The resource folder of the project contains all fonts which are used or have been used within the project. That means the folder can contain fonts which are not currently used. Those fonts are shown with zero references and can be deleted by pressing the ‘Delete’ button if not planned to be used any longer.

6.3.1 Font creation options

A font dialog offers the option for creating new fonts (by specifying family, style, type, and height in pixels) or importing existing ones from already existing projects. The available font families depend on the installed fonts of the host system.

> **Note**
>
> Fonts created with the FontConverter can not be used or imported by the AppWizard!



The following types of fonts can be created:

- 4bpp antialiased
- 2bpp antialiased
- 1bpp
- Framed fonts

6.3.2 Definition of code point ranges

Each font can have its own range of code points. The font selection dialog has the option for specifying the desired code point range.

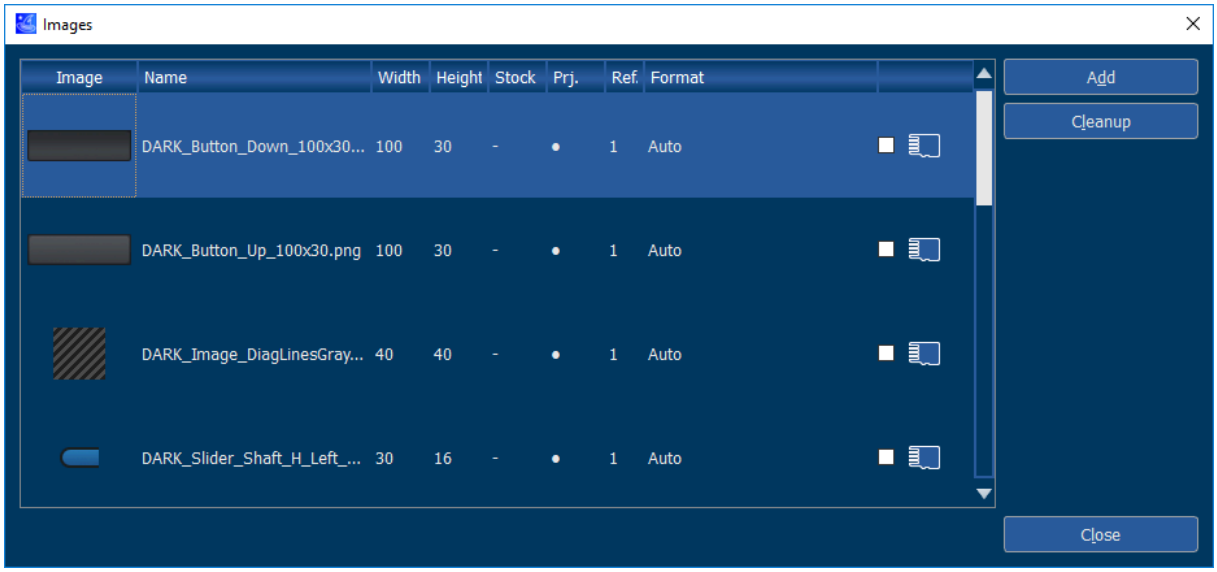
Clicking the according button opens a dialog for setting up the desired range. The following options exist:

- Setting up a list of code point ranges
- Using all code points required to draw the text defined in the application
- Using a custom pattern file which defines the code points to be used

6.4 Image management

> **Note**
>
> Please note the naming rules for image files:
> - The name must not start with a digit.
> - The only valid characters are letters, numbers and underscores.

The image management dialog shows all images located in the image stock or the resource folder of the project.



The dialog shows the following columns:

- Image preview
- File name
- Width and height in pixels
- Stock image, means image is located in image stock
- Project image, means image is located in resource folder of the project
- Number of references
- Check box to specify SD card management for that image

The following options exist:

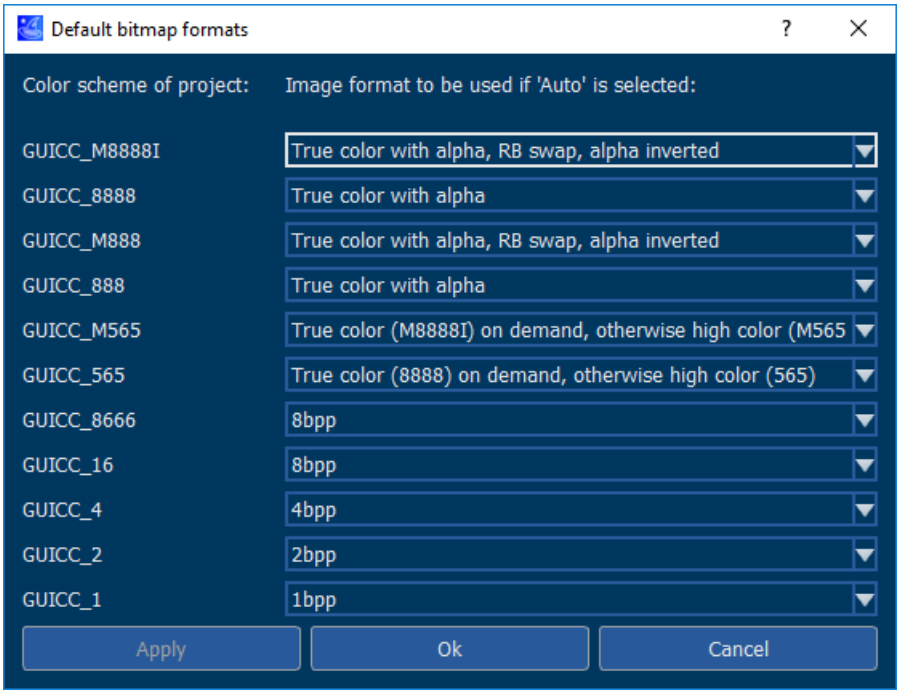
- Show stock images
- Show project specific images

> **Note**
>
> The project folder will contain all images used in the project, this applies to stock images as well.

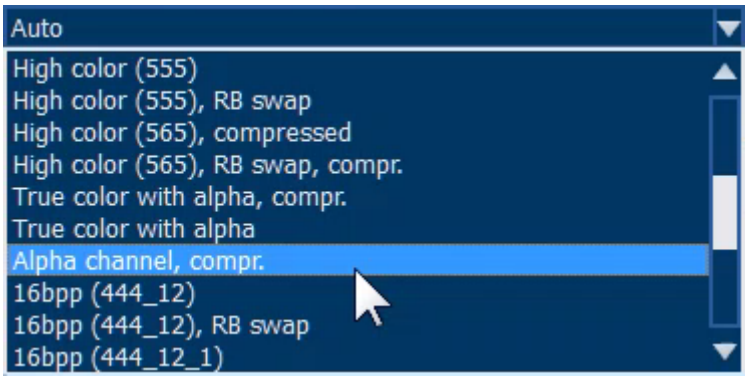
Bitmap format

When adding a new image, by default the format is set to 'Auto'. This will automatically choose the fitting bitmap format depending on which color format has been selected for the project. But the user may also select a specific bitmap format if the hardware requires it.

Additional to that the user has the option to select a completely different bitmap format by setting a format which should be used for the 'Auto' option. This can be done by opening the 'Default bitmap formats' dialog through the 'Preferences' menu.



Alpha channel bitmaps



This bitmap format allows bitmaps to be drawn in the current text color. With this format, only the alpha values of the individual pixels are stored. The RLE compression ensures an optimal use of the memory. The output is then in the text color set for the corresponding object.

Deleting images

Images can be deleted from the project by clicking the **Delete from project** button, after selecting the image that should be deleted. Images can only be deleted if the they haven't been referenced, that means the reference count shows zero.

Chapter 7

Objects

7.1 Introduction

The objects the AppWizard supports are similar to the widgets in emWin. The following table gives an overview about the currently available objects in the AppWizard.

> **Note**
>
> Some of the objects are based on emWin widgets. For more information refer to the document **UM03001 emWin User Guide & Reference Manual**.

Name	Symbol	Description
Box		Box object that can be colored or filled by a gradient and optional drawn with rounded corners.
Button		Clickable button object, can also be used as toggle button.
Dropdown		Allows the user to choose one value from a ’drop down’ list.
Edit		Edit field for user input. It also supports a decimal mode for entering numbers.
Gauge		Radial progress bar.
Image		Object that displays an image. Supports direct drawing of GIF, BMP, JPEG and Bitmap files.
Keyboard		Screen keyboard for entering text or numbers. Various predefined layouts can be used.
Listbox		Showing and selecting text from simple text lists.

Name	Symbol	Description
Listview		Table view containing multiple columns of text and a header line.
Multiedit		Edit field for multiple lines of user input.
Progbar		Progressbar to display the progression of a process.
QRCode		Displays a QR code.
Rotary		Circular object that can be rotated.
Screen		A screen serves as a parent for all other objects.
Slider		Movable slider.
Switch		Toggleable switch with two states and an optional fade mode.
Text		An object displaying text. A decimal- and a hexadecimal mode for showing numbers is also available.
Timer		Timer object.

Name	Symbol	Description
Window		Similar to screen object, serves as a parent object for other objects. Supports an opaque mode with an optional background color.
Wheel		Swipeable rotating wheel of items with texts and/or bitmaps.

7.2 Object properties

Every object has its own properties than can be edited. The following section lists all common properties that are used in multiple objects. Object-specific properties are explained in the corresponding object section.

This table lists the common properties and provides links to its corresponding chapter with more explanation.

Property	Description
Alignment	Alignment of an object.
Bitmap	Bitmap to be shown in an object.
Border size	Size of border for an object.
Color and background color	Foreground and background colors for an object.
Content	List- and table content for text based objects.
Cursor inversion	When disabling cursor inversion, the color for the cursor isn't the inverted background color.
Decimal mode	Makes an object only eligible for decimal digits.
Focus options	Disables the focus for an object or hides it.
Font	Font for the object.
Frame radius	Radius of frame around an object.
Frame size	Pixel-size of frame around an object.
Hexadecimal mode	Makes an object only eligible for hexadecimal digits.
Horizontal mode	Changes an object to be horizontal.
Motion partner	Motion settings for swiping between screens.
Motion support	Settings for swiping content.
ID	ID for the object.
Initial value	Initial value for an object.
Invert direction	Inverts direction of an object.
Opaque mode	Removes the transparency flag of an object.
Overwrite mode	Overwrite mode for text cursors of an object.
Period	Movement period.
Position and size	Position and size of an object.
Radius	Radius of an object.
Range	Range of position values of an object.
Space	Spacing.
Span of values	Set range of values of an object.
Stay on top	Makes sure a screen is displayed on top of all other screens.
Text	Text to be shown.
Text color	Color for text.
Text rotation	Rotation mode for text.
Text wrapping	Enables text wrapping.
Tiling	Tiling mode for bitmaps.
Untouchable	Screen will not be able to receive touch input.
Vertical mode	Changes an object to be vertical.

7.2.1 Alignment

Description

The alignment property allows to choose a combination of a horizontal alignment and a vertical alignments. This property can be set for bitmaps and texts.

Available objects

This property can be set for the following objects:

- *Button* object
- *Dropdown* object
- *Edit* object
- *Gauge* object
- *Multiedit* object
- *Listbox* object
- *Listview* object
- *Text* object

Usage

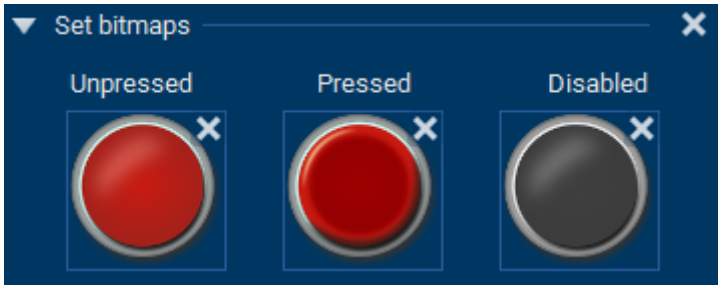
Combined with	Horizontal left	Horizontal center	Horizontal right
Vertical top			
Vertical center			
Vertical bottom			

You may also add an x and y offset to the object.

7.2.2 Bitmap

Description

The bitmap property allows to set a bitmap to a specific state of an object.



Available objects

This property can be set for the following objects:

- *Button* object
- *Dropdown* object
- *Image* object
- *Listview* object
- *Progbar* object
- *Slider* object
- *Switch* object
- *Rotary* object
- *Wheel* object

7.2.3 Border size

Description

This property sets the border size of an object in pixels. The border is the spacing between the frame and the text.

Available objects

This property can be set for the following objects:

- *Edit* object
- *Multiedit* object
- *Wheel* object

7.2.4 Color and background color

Description

The color and background color properties set the color or background color of an object.

Available objects

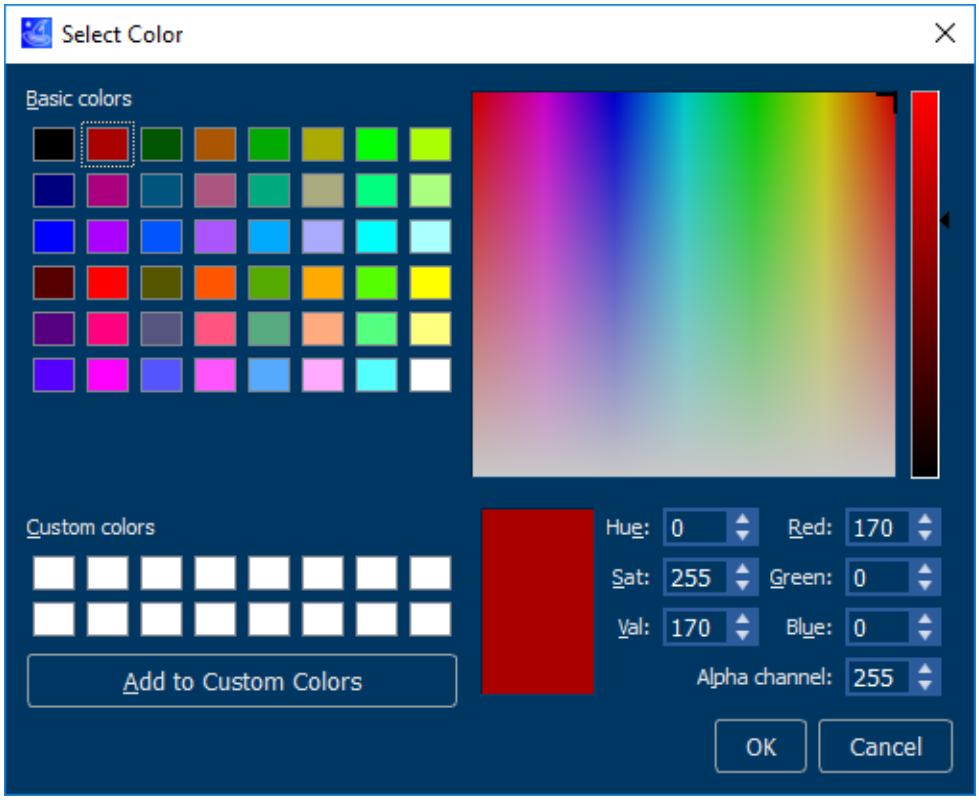
This property can be set for the following objects:

- *Box* object
- *Button* object
- *Dropdown* object
- *Edit* object
- *Gauge* object
- *Image* object
- *Keyboard* object
- *Listbox* object
- *Listview* object
- *Multiedit* object
- *Progbar* object
- *Wheel* object

Usage

When selecting the color, a dialog is opened. This dialog allows to set a specific color by setting RGB and HSV values, as well as the alpha value.

You may also save custom colors.



Related topics

- Text color

7.2.5 Content

Description

The term ’Content’ means lists and tables for text based objects. A ’list’ means a simple array of strings. A ’table’ consists of multiple columns and a header line. Column size, text alignment and content of the header line is also part of a table definition.

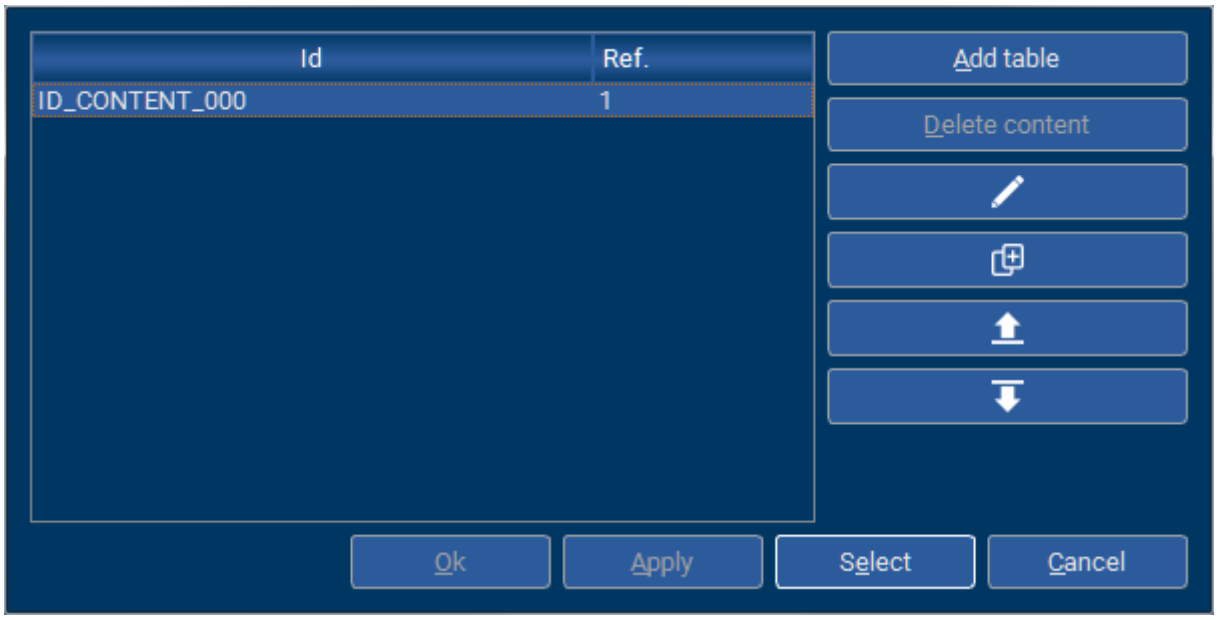
Available objects

This property can be set for the following objects:

- *Dropdown* object
- *Listbox* object
- *Listview* object
- *Wheel* object

Usage

When opening the content selection, a dialog with a list of the currently existing content is shown. It allows selecting existing or creating new content.



In dependence of the underlying object either tables or lists can be selected/created. For example LISTBOX- and DROPDOWN-objects require lists and the LISTVIEW-object requires tables. When clicking the edit pen or the ’Add table/list’ button the content definition dialog opens:

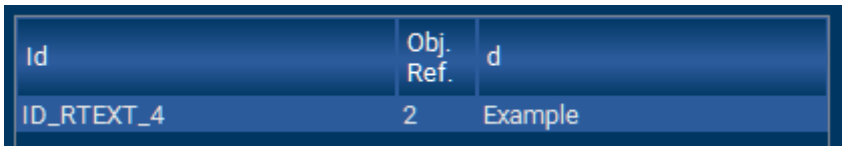
This dialog is used to edit existing content and/or to create new content. Lists consist of simple text arrays. Tables consist of several columns. For each column the width and the alignment can be defined as well as the content of the header.

Using text from text resources

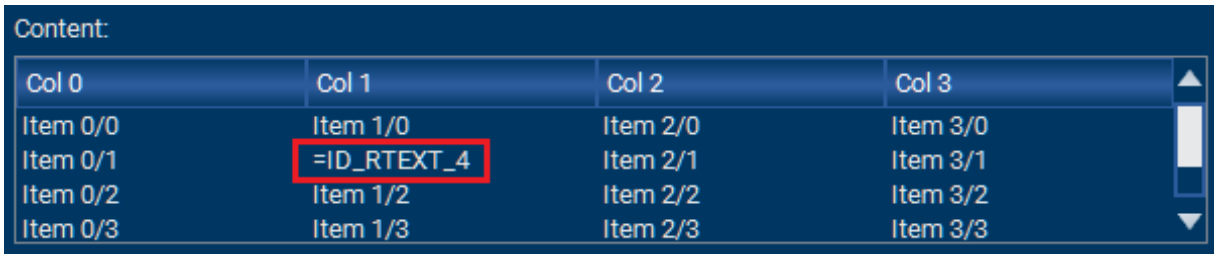
The content objects also support using text from the text resources. This can be achieved by the following mechanism: Simply enter an equal sign followed directly by the name of the desired text resource.

Example

Assuming the text resources contains an item named `ID_RTEXT_4`:



To use it in a table, in a list or in the header line of a table, use an equal sign with the resource name:



Importing and exporting content

A detailed description on how lists and tables can be imported and exported can be found under *Importing and exporting of content* on page 50.

Setting table- or list content by custom code

After creating an object containing a list or a table an `APPW_MSG_GET_CONTENT` message is send to the screen callback routine. The element 'hWinSrc' of the message structure is

(ob)used for passing the object id of the object requesting the content. To pass content to the requesting object simply set `Data.p` to an `APPW_CONTENT` structure. Here is a code excerpt which shows how to realize that:

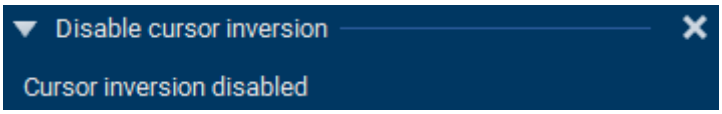
```
static const char * ID_CONTENT_Left_Col_0[] = {
  "Left Item 0",
  "Left Item 1",
  "Left Item 2",
  "Left Item 3",
  "Left Item 4",
  "Left Item 5",
  "Left Item 6",
  "Left Item 7",
  "Left Item 8",
  "Left Item 9",
};

static const char ** ID_CONTENT_Left_Text[] = {
  ID_CONTENT_Left_Col_0,
};

const APPW_CONTENT ID_CONTENT_Left_Data = {
  ID_CONTENT_Left_Text,
  GUI_COUNTOF(ID_CONTENT_Left_Col_0)
};

static const char * ID_CONTENT_Right_Col_0[] = {
  "Right Item 0",
  "Right Item 1",
  "Right Item 2",
  "Right Item 3",
  "Right Item 4",
  "Right Item 5",
  "Right Item 6",
  "Right Item 7",
  "Right Item 8",
  "Right Item 9",
};

static const char ** ID_CONTENT_Right_Text[] = {
  ID_CONTENT_Right_Col_0,
};

const APPW_CONTENT ID_CONTENT_Right_Data = {
  ID_CONTENT_Right_Text,
  GUI_COUNTOF(ID_CONTENT_Right_Col_0)
};

void cbID_SCREEN_00(WM_MESSAGE * pMsg) {
  switch (pMsg->MsgId) {
  case APPW_MSG_GET_CONTENT:
    switch (pMsg->hWinSrc) {
    case ID_DROPDOWN_00:
      pMsg->Data.p = &ID_CONTENT_Left_Data;
      break;
    case ID_DROPDOWN_01:
      pMsg->Data.p = &ID_CONTENT_Right_Data;
      break;
    }
    break;
  }
}
```

The sample library also contains a sample which shows how to set custom dropdown text.

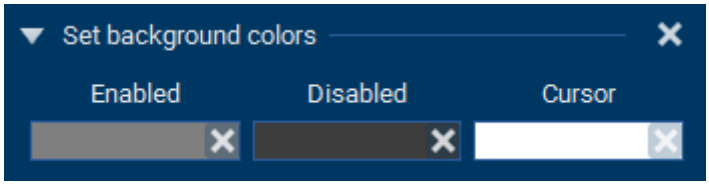
7.2.6 Cursor inversion

Description

Cursor inversion is by default activated for Edit objects. When deactivating it, the color of the cursor is no longer the inverted background color of the Edit object, but rather the user can pick a new color.



The user can set the color of the cursor with the Background color property under ’Cursor’.



Available objects

This property can be set for the following objects:

- *Edit* object
- *Multiedit* object

7.2.7 Decimal mode

Description

With decimal mode, the Edit or Text object is only eligible of holding digits instead of characters. For decimal mode, a mask of zeros has to be specified which determines how many digits are shown by the object. Also, when using decimal mode, a range property is added to the object to limit the numbers that can be entered.

Available objects

This property can be set for the following objects:

- *Edit* object
- *Text* object

Additional information

The mask being set for an Edit or Text object behaves slightly different.

The Edit object accepts either a single ’0’ or a single ’#’ as mask. A ’0’ shows leading zeros depending on the range set for this object.

The Text object shows as many leading zeros as ’0’ are being used as mask. The umber of ’#’ used indicates the maximum number of digits.

Example

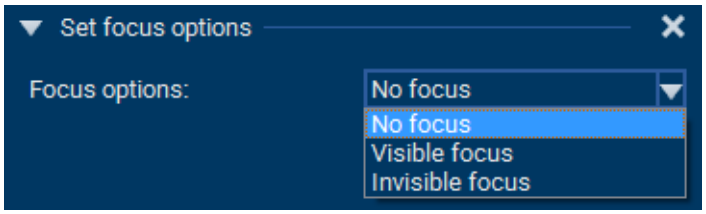
As an example, the mask `00000` for a Text object would allow a maximum of five digits and would also show the zeros if the entered number has less digits than five.

Mask	Result
	00123

7.2.8 Focus options

Description

Focus options for an individual object. The focus rectangle can be shown or hidden or the ability to receive input focus can be disabled altogether.



Available objects

This property can be set for the following objects:

- *Button* object
- *Dropdown* object
- *Edit* object
- *Listbox* object
- *Listview* object
- *Multiedit* object
- *Rotary* object
- *Slider* object
- *Switch* object

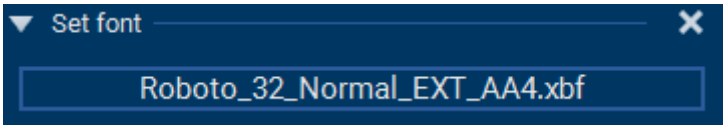
Additional information

Focus support has to be enabled in the project options. More information about input focus can be read under *Object focus* on page 102.

7.2.9 Font

Description

The font property allows to set a font to an object.



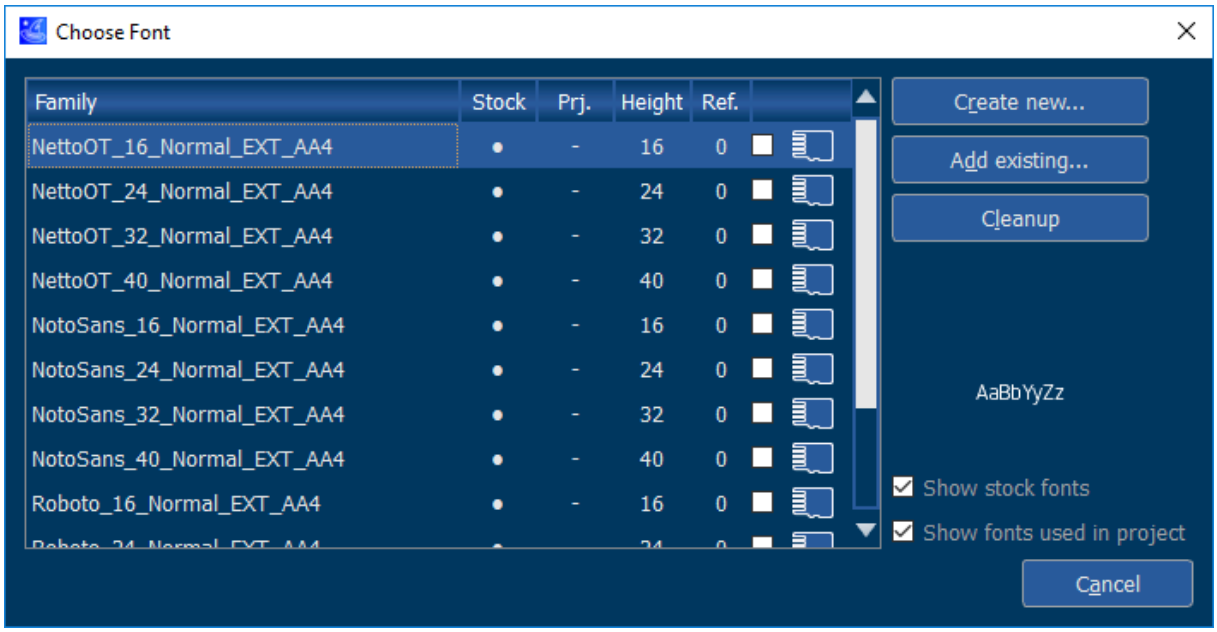
Available objects

This property can be set for the following objects:

- *Button* object
- *Dropdown* object
- *Edit* object
- *Keyboard* object
- *Multiedit* object
- *Switch* object
- *Text* object
- *Wheel* object

Usage

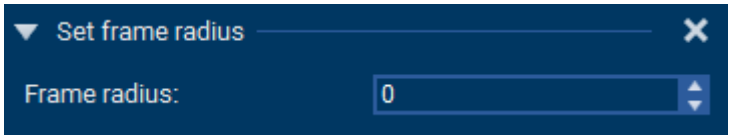
Choosing a font will open a dialog showing all fonts available in the project.



7.2.10 Frame radius

Description

This property sets the radius of the frame drawn around a given object.



Available objects

This property can be set for the following objects:

- *Edit* object
- *Listview* object
- *Multiedit* object
- *Wheel* object

7.2.11 Frame size

Description

This property sets the width of an object’s frame in pixels.

Available objects

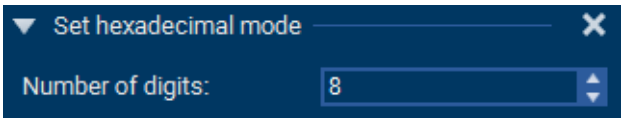
This property can be set for the following objects:

- *Dropdown* object
- *Edit* object
- *Listbox* object
- *Listview* object
- *Multiedit* object
- *Progbar* object
- *Wheel* object

7.2.12 Hexadecimal mode

Description

Hexadecimal mode makes an object only eligible for hexadecimal digits. The number of shown digits has to be set via the property.



Available objects

This property can be set for the following objects:

- *Text* object

7.2.13 Horizontal mode

Description

The horizontal mode property changes the orientation of an object to be horizontal. By default it is vertical.

Available objects

This property can be set for the following objects:

- *Wheel* object

7.2.14 Motion partner

Description

Horizontal and vertical motion allow swiping between different screens or windows.

Available objects

This property can be set for the following objects:

- *Screen* object
- *Window* object

Horizontal motion properties

Property	Description
Left partner	Screen/window that should be located left from the screen/ window.
Mode left	Mode that should be applied to the left partner. Either ’disclose’ or ’replace’.
Right partner	Screen/window that should be located right from the screen/ window.
Mode right	Mode that should be applied to the left partner. Either ’disclose’ or ’replace’.
Period	Period to be used until motion stops.

Vertical motion properties

Property	Description
Upper partner	Screen/window that should be located above the screen/window.
Mode up	Mode that should be applied to the upper partner. Either ’disclose’ or ’replace’.
Lower partner	Screen/window that should be located below the screen/window.
Mode down	Mode that should be applied to the lower partner. Either ’disclose’ or ’replace’.
Period	Period to be used until motion stops.

Disclose mode

In ’disclose mode’ the window that the user is swiping to will be disclosed. This means only the window that is swiped away moves, the other window does not.

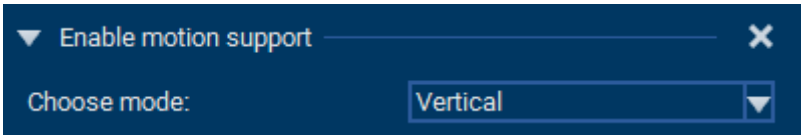
Replace mode

In ’replace mode’ the window that the user is swiping to replaces the old window as the user is swiping.

7.2.15 Motion support

Description

Horizontal and vertical motion allow swiping the content of the widget.



Available objects

This property can be set for the following objects:

- *Multiedit* object

7.2.16 ID

Description

Every object has an ID that can be set in order to identify that object.

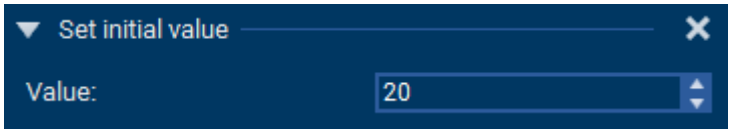
Available objects

This property can be set for all objects.

7.2.17 Initial value

Description

This property sets the initial value of an object.



Available objects

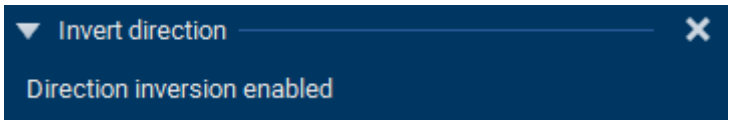
This property can be set for the following objects:

- *Gauge* object
- *Progbar* object
- *Rotary* object
- *Slider* object

7.2.18 Invert direction

Description

This property inverts the direction of an object, meaning it lowest value and initial position will be on the right instead of the left.



Available objects

This property can be set for the following objects:

- *Slider* object
- *Progbar* object

7.2.19 Opaque mode

Description

This property removes the transparency flag of an object. For example, when a transparent button is pressed, not only the button is redrawn, but also the window that is behind the button.

When an opaque button is pressed, only the button itself is redrawn, not the window that is behind the button.

Note

This property should **only** be used, if the entire area of the object is drawn. For example, if an opaque button has rounded corners, the corners of the button will not be redrawn and corrupted pixels will appear on the screen.

Available objects

This property can be set for the following objects:

- *Button* object
- *Image* object
- *Window* object

7.2.20 Overwrite mode

Description

This property sets the mode of a possible cursor to overwrite. The default mode is insert.

Available objects

This property can be set for the following objects:

- *Edit* object
- *Multiedit* object

7.2.21 Period

Description

This property sets a time period in ms how long the related operation should take until it is finished.

Available objects

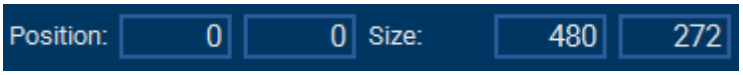
This property can be set for the following objects:

- *Keyboard* object
- *Multiedit* object
- *Progbar* object
- *Rotary* object
- *Switch* object
- *Timer* object
- *Wheel* object

7.2.22 Position and size

Description

Every object has its position and its size.



Available objects

These properties can be set for every object.

7.2.23 Radius

Description

This property sets the radius of an object or a specific part of an object. For the Rotary object, it depends on this radius where the marker bitmap will be positioned.

Available objects

This property can be set for the following objects:

- *Box* object
- *Button* object
- *Gauge* object
- *Keyboard* object
- *Progbar* object
- *Rotary* object

Rounded corners

For the Box and Button objects, this property defines the radius of rounded corners.

7.2.24 Range

Description

This property allows to define a range for an object.

Available objects

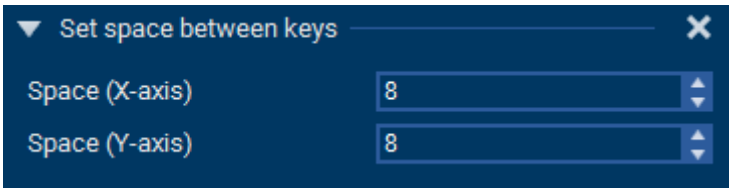
This property can be set for the following objects:

- *Edit* object (in decimal mode)
- *Progbar* object
- *Rotary* object
- *Slider* object
- *Text* object (in decimal mode)

7.2.25 Space

Description

This property defines the spacing. For instance, spacing in X- and Y-axis can be set between each key on a Keyboard object.



Available objects

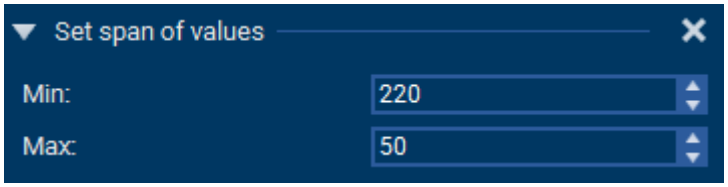
This property can be set for the following objects:

- *Edit* object
- *Keyboard* object

7.2.26 Span of values

Description

This property defines the range of numbers an object should return.



Available objects

This property can be set for the following objects:

- *Gauge* object
- *Rotary* object

7.2.27 Stay on top

Description

This property allows a screen to be shown on top of all other screens. In order for the screen to be visible, Persistent mode also has to be enabled.

If multiple screens are marked to stay on top, their order in the hierarchic tree will determine in what order they are shown.

Available objects

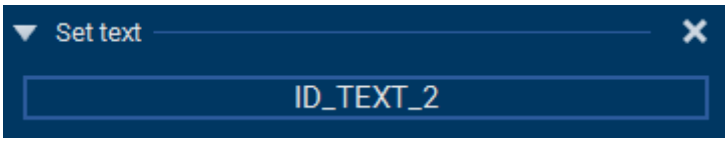
This property can be set for the following objects:

- *Screen* object

7.2.28 Text

Description

The text property allows to select a text to be shown from the text window. Only a text from the text window can be selected. For more information about the text window and how to add texts, refer to *Text resource window* on page 44.



Available objects

This property can be set for the following objects:

- *Button* object
- *Edit* object
- *Multiedit* object
- *QRCode* object
- *Switch* object
- *Text* object

7.2.29 Text color

Description

The text color property sets the text color of an object for its different states.

Available objects

This property can be set for the following objects:

- *Button* object
- *Dropdown* object
- *Edit* object
- *Listbox* object
- *Listview* object
- *Multiedit* object
- *Switch* object
- *Text* object
- *Wheel* object

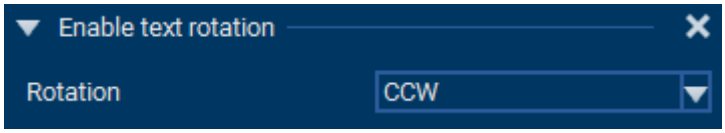
Related topics

- Color and background color

7.2.30 Text rotation

Description

Rotates the text in an object.



Available objects

This property can be set for the following objects:

- *Text* object

Available rotation modes

Rotation	Description
`CCW`	Counter-clockwise rotation (210°).
`CW`	Clockwise rotation (90°).
`180°`	180° rotation.

7.2.31 Text wrapping

Description

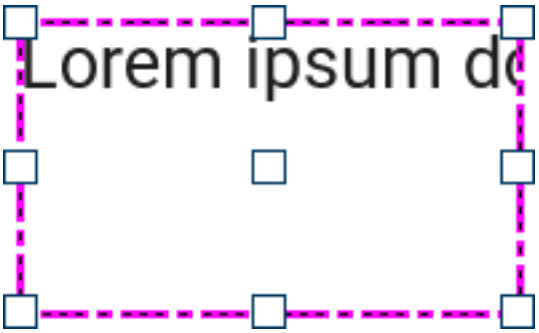
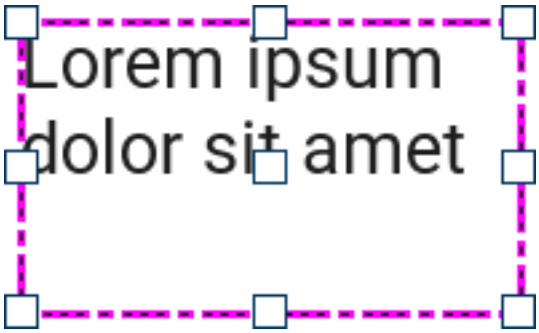
Text wrapping for text in an object.

Available objects

This property can be set for the following objects:

- *Multiedit* object
- *Text* object

Comparison

No text wrapping	Text wrapping
	

7.2.32 Tiling

Description

Tiling mode will fill the entirety of the Image object with the selected image.

It can also be used for Progbar objects to fill the entire Progbar with a narrow bitmap (usually 1 px wide).

Available objects

This property can be set for the following objects:

- *Image* object
- *Progbar* object

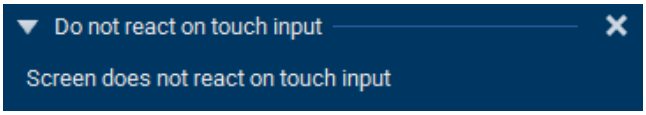
Example

See chapter *Image* on page 111 for an example.

7.2.33 Untouchable

Description

An untouchable screen is not able to receive touch input.



Available objects

This property can be set for the following objects:

- *Box* object
- *Image* object
- *Screen* object
- *Text* object

7.2.34 Vertical mode

Description

The vertical mode property changes the orientation of an object to be vertical. By default it is horizontal.

Changing a Progbar or Slider to horizontal or vertical mode will automatically change the default bitmaps accordingly.

Available objects

This property can be set for the following objects:

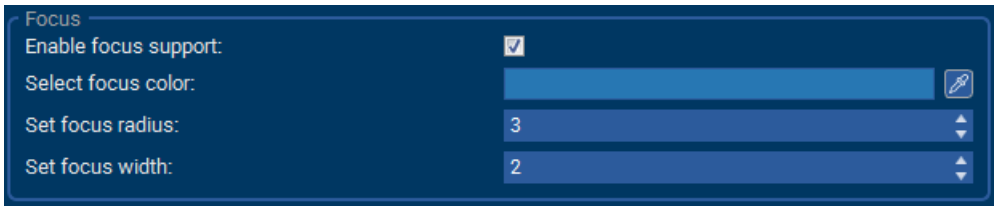
- *Progbar* object
- *Slider* object

7.3 Object focus

Since V1.12, objects are able to receive input focus. The ability for objects to receive focus must be enabled in the project options, as described earlier in the chapter *User interface* on page 30.

Enabling focus support

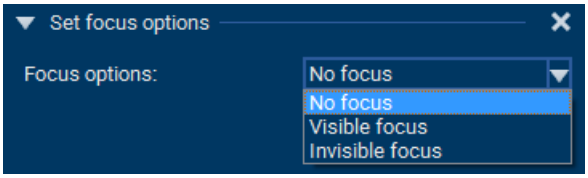
The user can define show the focus rectangle should be drawn by setting the line width, line color and corner radius in the project options.



How it works

If an object has the input focus, all key inputs are sent to the object. Using the tab key sets the focus to the next object, according to the hierarchic object tree.

Optionally, the ability of an object to receive focus or visibility of it can be modified for individual objects through the use of the Focus options property.



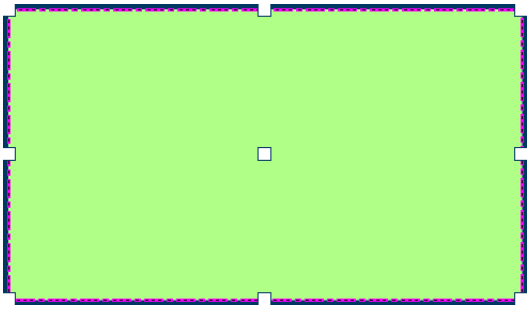
The objects that are able to receive focus are listed under the signal *GOT_FOCUS* on page 143.

Example

Below is an example Edit object that has the focus:

7.4 Box

A box object can be placed as the first object in a window/screen and simply serves for specifying a background color or a gradient. Horizontal and vertical gradients are supported. A gradient can have an unlimited number of colors. For each color the pixel position can be defined. Semi-transparent gradients are also supported.

Symbol	Example
	

> **Note**
>
> Semi-transparency is only recommended if a hardware is used which either has an accelerator for semi-transparent filling operations or is fast enough to mix up the colors per software.

Properties

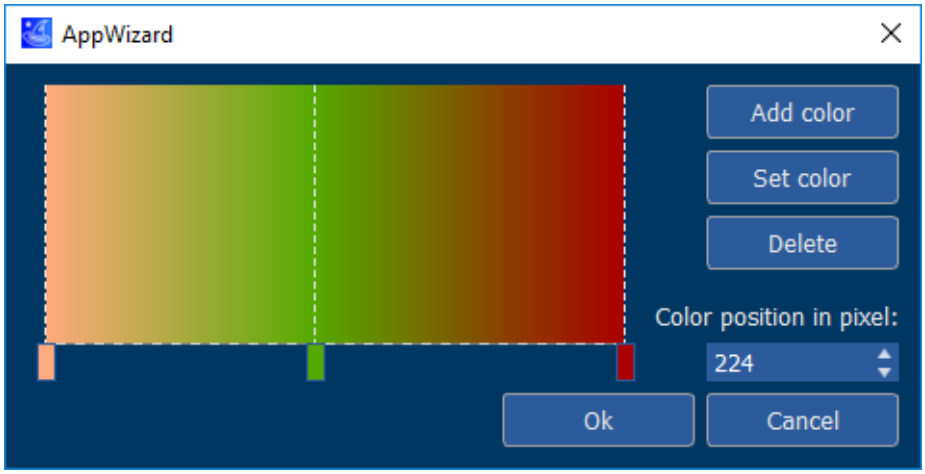
Property	Description
`Color`	Color to be shown in the box.
`Horizontal gradient`	Horizontal gradient to be shown in the box.
`Vertical gradient`	Vertical gradient to be shown in the box.
`Radius`	Radius of the rounded corners.
`Untouchable`	Sets the box object as untouchable and sends any touch input to the object below.

Gradients

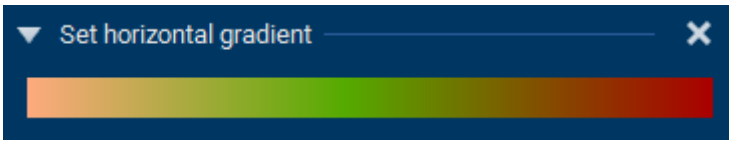
Horizontal and vertical gradients can be defined using two colors or more.

Gradient colors may be added via the **Add color** button. A gradient must contain at least two colors. The colors can be changed when the corresponding marker has been clicked. They can be edited using the **Set color** button and deleted via the **Delete button**.

The position of each color can be changed by specifying the position in the spinbox or by moving the markers.



The result of the above specified gradient looks like this below:



7.5 Button

The button object is very similar to its emWin counterpart. It is an object that can be clicked, so that its input may be processed by the application.

Symbol	Example
	

Properties

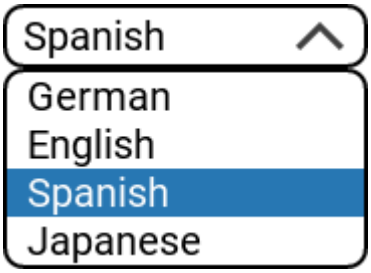
Property	Description
Text colors	• Unpressed: Text color for unpressed state. • Pressed: Text color for pressed state. • Disabled: Text color for disabled state.
Background colors	• Unpressed: Background color for unpressed state. • Pressed: Background color for pressed state. • Disabled: Background color for disabled state.
Bitmaps	• Unpressed: Bitmap for unpressed state. • Pressed: Bitmap for pressed state. • Disabled: Bitmap for disabled state.
Bitmap alignment	• Alignment: Bitmap alignment. • Offset x: Additional x-offset. • Offset y: Additional y-offset.
Auto repeat	• Start time: Starting time of auto repeating after button press. • Interval: Repeating time.
Toggle mode	By clicking the button its state is toggled between pressed and unpressed.
Text	Text to be shown.
Text alignment	• Alignment: Text alignment. • Offset x: Additional x-offset. • Offset y: Additional y-offset.
Font	Font to be used for the text.
Focus options	Disables the focus for the button or hides it.
Radius	Radius for rounded corners if the button is drawn without bitmaps.
Opaque mode	Sets the button to opaque and removes its transparency flag.

Auto repeat mode

The Button also offers an auto repeat mode. When holding the button pressed, it begins sending clicked events after the start time period in the given interval.

7.6 Dropdown

The Dropdown object is a control that allows a user to select a value from a given list of values. In inactive state it shows the currently selected item. When activating it via PID or keyboard, a 'drop-down' (or even 'drop-up') list pops with the selectable items. The list automatically disappears after selecting an item or when clicking elsewhere on the display.

Symbol	Example
	

Properties

Property	Description
Bitmaps	• Down: Down arrow to be shown. • Up: Down arrow to be shown.
Content	Content to be shown.
Text colors	• Enabled: Text color for enabled state. • Disabled: Text color for disabled state. • Cursor: Color of cursor.
Background colors	• Enabled: Background color for enabled state. • Disabled: Background color for disabled state. • Cursor: Background color of cursor.
Font	Font to be used.
Text alignment	• Alignment: Text alignment. • Offset x: Additional x-offset. • Offset y: Additional y-offset.
Opening upwards	Changes the dropDOWN to a dropUP.
Height of list	Height in pixels of the list.
Frame color	Color of frame of object.
Frame radius	Radius of rounded corners of the frame.
Frame size	Width of frame around object.
Inner gap	Border size between frame and text.
Focus options	Disables the focus for the object or hides it.

7.7 Edit

An Edit object provides, like the emWin EDIT widget, a box where the user can type text in, or numbers if decimal mode is activated.

Symbol	Example
	

Properties

Property	Description
Text	Text to be displayed initially.
Decimal mode	Mask to be used for decimal mode.
Overwrite mode	Set the cursor mode to overwrite.
Text colors	• Enabled: Text color for enabled state. • Disabled: Text color for disabled state. • Cursor: Color of cursor.
Background colors	• Enabled: Background color for enabled state. • Disabled: Background color for disabled state. • Cursor: Background color of cursor.
Frame color	Color of frame of Edit.
Cursor inversion	Disables cursor inversion mode.
Blink period	Blinking period of cursor.
Font	Font to be used.
Text alignment	• Alignment: Text alignment. • Offset x: Additional x-offset. • Offset y: Additional y-offset.
Inner gap	Spacing between frame and text.
Frame radius	Radius of rounded corners of the Edit’s frame.
Frame size	Line size of frame around Edit.
Maximum length	Maximum length of characters that can be entered into the Edit.
Focus options	Disables the focus for the Edit or hides it.

Decimal mode

With decimal mode, the Edit object is only eligible of holding digits instead of characters. For this mode, a mask of zeros has to be specified which determines how many digits are shown by the object. More details about the usage of the mask is explained under *Decimal mode* on page 74.

Also, when using decimal mode, a range property is added to the object to limit the numbers that can be entered. More on the range property can be found under *Range* on page 91.

During runtime, the cursor is highlighting the currently selected digit. When the user types in a number, the cursor will move from its current position to the right until the last digit

has been reached. If the entered number exceeds the maximum, the maximum number is put in.

The number can be increased using the <UP> and decreased using the <DOWN> key, whereas the cursor can be moved using the <LEFT> and <RIGHT> arrow keys.

7.8 Gauge

A Gauge object is similar to a progress bar, although the progress is displayed in a radial manner. The object consists of two arcs that are drawn. The relation between these two arc lines shows the progress.

Two colors can be set for a Gauge object, for the background and and foreground line.

Symbol	Example
	

Properties

Property	Description
Center alignment	• Alignment: Alignment of the Gauge within the object frame. • Offset x: Additional x-offset of the Gauge. • Offset y: Additional y-offset of the Gauge.
Initial value	Initial value of the Gauge.
Start/end angle	• Angle 0: Start angle in 10th of degrees. • Angle 1: End angle in 10th of degrees.
Span of values	• Min: Lowest value the object should return. • Max: Highest value the object should return.
Radius	Radius of the Gauge.
Colors	• Item 0: Color of the background curve. • Item 1: Color of the foreground curve.
Line width	• Width 0: Width of the background curve. • Width 1: Width of the foreground curve.
Rounded value	Enables rounded ends of the foreground (value) curve.
Rounded ends	Enables rounded ends of the background curve.
Background color	Background color of the object.

Rounded value/ends

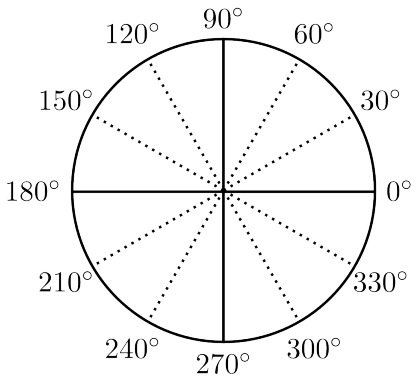
With this property, the ends of the value or end arc of a Gauge object can be set to have rounded edges.

Default	Rounded value	Rounded ends
		

Start/end angle

This property defines the angles, where the arc of a Gauge object should start and end. The values to be entered should be 10th of degrees (1800 = 180°).

The entered degree values are based off the standard angle measurement.



7.9 Image

An Image object is similar to emWin's IMAGE widget. It can be used to display any images of the file types JPEG, GIF or BMP. Alternatively, a bitmap can be chosen as well.

Symbol	Example
	

Properties

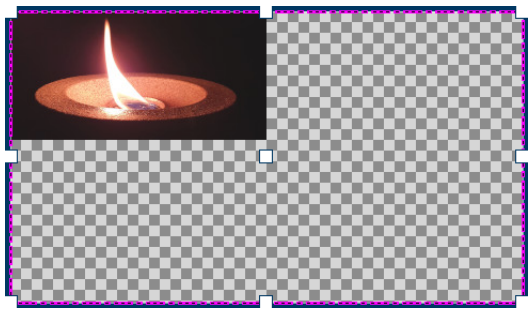
Property	Description
`Bitmap`	Sets a bitmap to the object.
`Color`	Sets the color to be used for an alpha bitmap.
`JPEG`	Sets a JPEG image to the object.
`GIF`	Sets a GIF image to the object.
`BMP`	Sets a BMP image to the object.
`Tiling`	Enables tiling mode for the object.
`Opaque mode`	Sets the image to opaque and removes its transparency flag.
`Untouchable`	Sets the image object as untouchable and sends any touch input to the object below.

Difference between bitmaps and images

In contrast to bitmaps, JPEG, GIF and BMP images are always displayed natively. Therefore JPEGs and GIFs are always decompressed before being displayed. This can lead to a notable difference in performance compared to bitmaps.

Tiling mode

Tiling mode will fill the entirety of the Image object with the selected image. In this example the purple frame surrounds the Image object.

Tiling mode disabled	Tiling mode enabled
	

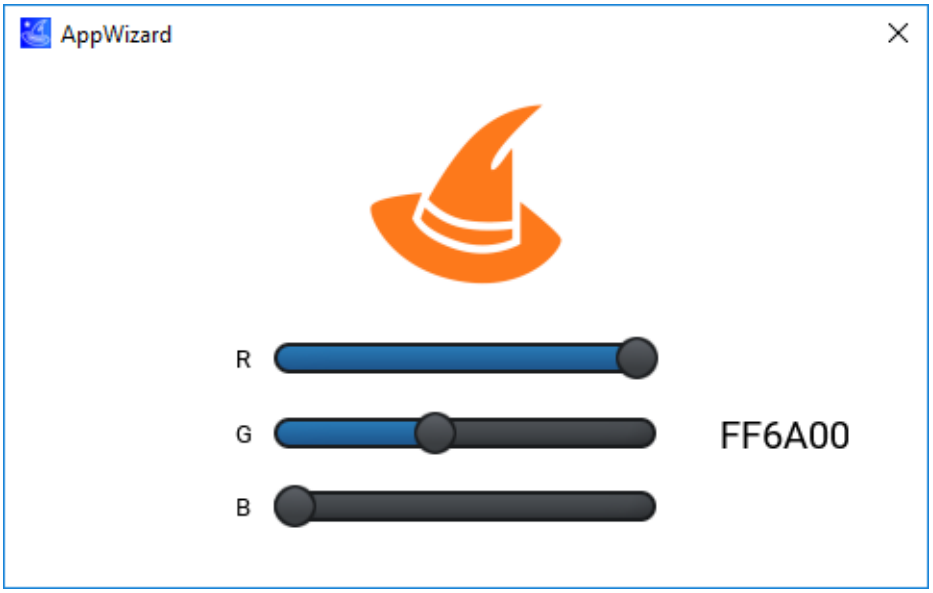
GIF support

Any GIF images are supported for this object, this includes animated GIFs.

Alpha bitmaps

To create an alpha bitmap to use it for an Image object, click the “Set bitmap” property of the Image. Then, add a new bitmap by opening the desired image. Now, select *Alpha channel, compr.* in the “Format” column to declare it as an alpha bitmap. Finally, select the bitmap for the Image.

For the alpha bitmap, a desired color can now be selected. The color can also be set to the Image by an interaction, using the SETCOLOR job.



In the example shown above R, G and B values can be entered via the slider. These three values are put together by a variable calculation to form the RGB value. Finally the RGB value can be set to the alpha bitmap via an interaction.

7.10 Keyboard

A Keyboard object can be used to enter text or numbers.

Symbol	Example
	

Properties

Property	Description
`Font for codes`	Font used for keys.
`Font for longpress codes`	Font used for longpress characters on keys.
`Keyboard layout`	Keyboard layout used for the object.
`Colors`	• Code: Color used for text shown on keys. • Long: Color used for longpress characters shown on keys. • Mark: Color used for selected character in long press dialog and locked shift key.
`Background colors`	• Key: Background color for keys. • F-Key: Background color for function keys. • Pressed: Background color for pressed keys. • BG: Background color of Keyboard object.
`Periods for backspace key`	• Start time: Period between the press of backspace and deletion of the characters. • Interval: Interval between each character deleted when holding backspace.
`Radius for key outline`	Radius used for rounded corners or the keys.
`Space between keys`	• Space (X-axis): Space between each key on X-axis. • Space (Y-axis): Space between each key on Y-axis.

Keyboard layout

The following layouts are available per default and can be set to a Keyboard object:

Layout	Description
SKEYBOARD_ARA	Layout for Arabic.
SKEYBOARD_DEU	QWERTZ layout, used for German.
SKEYBOARD_DEU_LP	QWERTZ layout with extra longpress characters.
SKEYBOARD_ENG	QWERTY layout, used for English.
SKEYBOARD_ENG_LP	QWERTY layout with extra longpress characters.
SKEYBOARD_FRA_LP	AZERTY layout, used for French.
SKEYBOARD_NUMPAD	Numpad layout.
SKEYBOARD_RUS	ЈСУКЕН/ЙЦУКЕН layout, main Cyrillic keyboard layout for the Russian language.

Files for streamed layout files are located in the project directory under `Resource\Keyboard`. The pattern files needed for specific layouts are also located in this directory.

Note that pattern files for Arabic only contain the isolated letter forms displayed on the keyboard. This excludes representation forms that are required for the display of Arabic texts.

7.11 Listbox

A list box allows selecting an item from a multiple line text box.

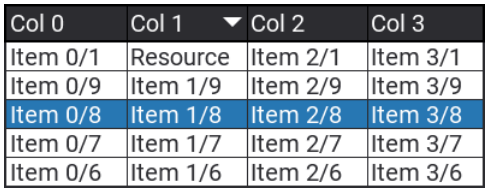
Symbol	Example
	

Properties

Property	Description
`Text colors`	• Unselected: Text color for unselected state. • Selected: Text color for selected state. • Focused: Color of cursor. • Disabled: Text color for disabled state.
`Background colors`	• Unselected: Background color for unselected state. • Selected: Background color for selected state. • Focused: Background color of cursor. • Disabled: Background color for disabled state.
`Font`	Font to be used.
`Content`	Content to be shown.
`Text alignment`	• Alignment: Text alignment. • Offset x: Additional x-offset. • Offset y: Additional y-offset.
`Frame color`	Color of frame of object.
`Frame size`	Width of frame around object.
`Focus options`	Disables the focus for the object or hides it.

7.12 Listview

A Listview object allows selecting one line of of a list with several columns. Each column has its own width and text alignment.

Symbol	Example
	

Properties

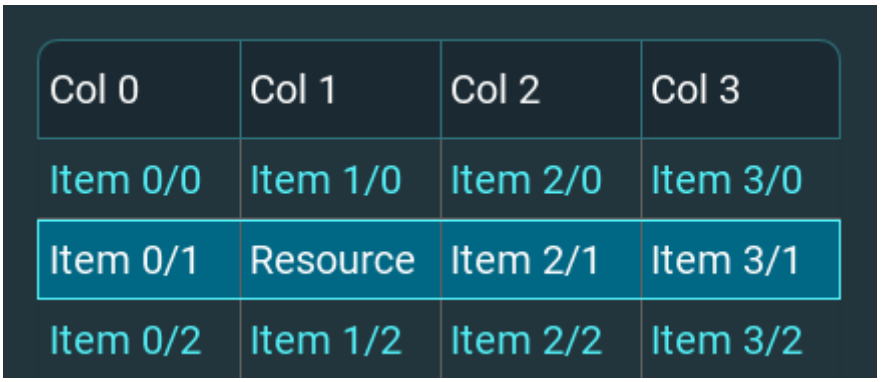
Property	Description
`Text colors`	• Unselected: Text color for unselected state. • Selected: Text color for selected state. • Focused: Color of cursor. • Disabled: Text color for disabled state.
`Background colors`	• Unselected: Background color for unselected state. • Selected: Background color for selected state. • Focused: Background color of cursor. • Disabled: Background color for disabled state.
`Header colors`	• Text: Color for header text. • Background: Color for header background. • Frame: Color for header frame.
`Colors`	• Grid: Color for grid lines. • Focus: Color for the optional focus rectangle. • Frame: Frame color of list.
`Bitmaps`	• Down: Optional descending indicator to be shown. • Up: Optional ascending indicator to be shown.
`Font`	Font to be used for content.
`Header font`	Font to be used for header. This is optional, if left empty, the same font set to the LISTVIEW is used.
`Content`	Content to be shown.
`Row height`	Row height be used for content.
`Header height`	Row height be used for header.
`Header radius`	Radius to be used for the header frame.
`Frame size`	Width of frame around the list.
`Focus rectangle width`	Width of the optional focus rectangle.
`Show header grid`	Show vertical grid lines in header.
`Show horizontal grid lines`	Show horizontal grid lines.
`Show vertical grid lines`	Show vertical grid lines.
`Enable cell selection`	Enables single cell selection instead of line selection.
`Fixed columns`	Sets the number of fixed columns.
`Column sorting`	Enables column sorting.
`Inner gap`	Horizontal border between text and cell.
`Focus options`	Disables the focus for the object or hides it.

Header radius

The upper corners of the header can be rounded with a separate radius. Further there are options for setting a separate color for frame of the header line:

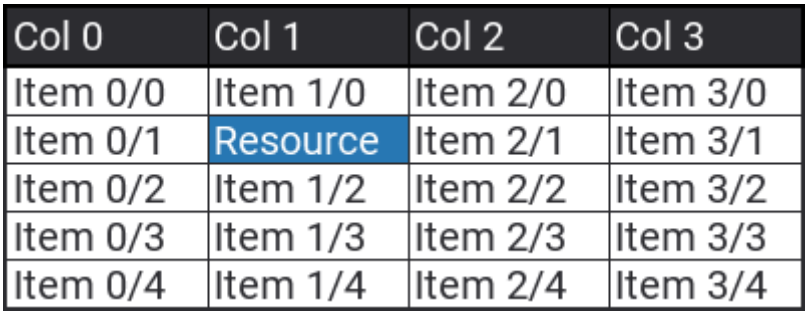
Focus rectangle width

The focus rectangle width (and color) allows showing a separate focus rectangle surrounding the currently selected row or cell:



Enable cell selection

Cell selection allows selecting single cells:

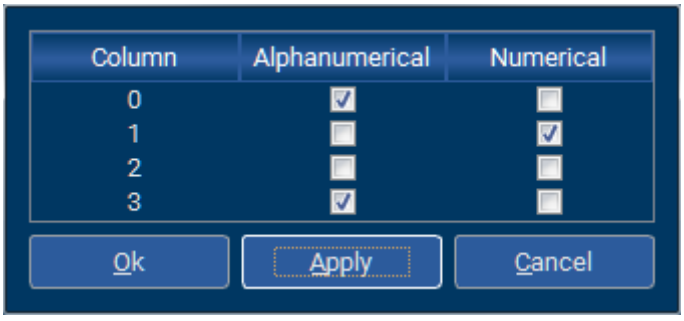


Fixed columns

Fixes the given number of columns at their horizontal positions.

Column sorting

When selecting this option a dialog occurs. It allows setting sorting options for each column:



Sorting then can be initiated by touching the header of the desired column. After touching the header a sort indicator becomes visible:

Touching the header again inverts the sorting order:

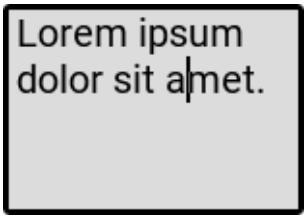
The AppWizard provides a convenient option for showing the indicators with the same color as used for the text. This can be achieved with monochrome images only. To use that feature select ’Alpha channel, compr.’ instead of the ’Auto’-option in the image dialog:

Inner gap

It defines the number of pixels between text and border of the cell. Makes sense with left or right alignment only.

7.13 Multiedit

The Multiedit object is a multi-line text input widget.

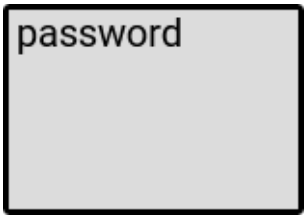
Symbol	Example
	

Properties

Property	Description
Text	Text to be displayed initially.
Overwrite mode	Set the cursor mode to overwrite.
Text colors	• Enabled: Text color for enabled state. • Disabled: Text color for disabled state. • Cursor: Color of cursor.
Background colors	• Enabled: Background color for enabled state. • Disabled: Background color for disabled state. • Cursor: Background color of cursor.
Frame color	Color of frame of Multiedit.
Cursor inversion	Disables cursor inversion mode.
Blink period	Blinking period of cursor.
Font	Font to be used.
Text alignment	Text alignment, only left and right alignment can be set to the Multiedit.
Text wrapping	Enables text wrapping.
Inner gap	Border size between frame and text.
Frame radius	Radius of rounded corners of the Multiedit's frame.
Frame size	Line size of frame around Multiedit.
Password mode	Enables password mode. See below for additional information.
Focus options	Disables the focus for the Multiedit or hides it.

Password mode

The password mode displays the text of the Multiedit as asterisks.

Password mode disabled	Password mode enabled
	

7.14 Progbar

A Progbar object visualizes the progression of an operation.

Symbol	Example
	

Properties

Property	Description
`Bitmap`	Bitmap for the “filling” and the “empty” part of the Progbar.
`Tiling`	Uses tiling for the given bitmaps. The bitmaps should be 1 pixel wide.
`Bitmap alignment`	Alignment of the set bitmaps.
`Colors`	Bitmap for the “filling” and the “empty” part of the Progbar.
`Frame size`	Size of the frame of the Progbar. If 0 no frame is displayed.
`Initial value`	Sets the initial value for the Progbar.
`Invert direction`	Inverts the direction of the Progbar (left to right or right to left).
`Period`	Sets a period which describes the duration it takes to move a Bitmap through the object.
`Radius`	Sets the radius of the edges of the Progbar. Affects also the radius of the frame. Has no effect on gradients.
`Range`	Sets the range of the Progbar.
`Vertical mode`	Changes the direction of the Progbar from horizontal to vertical.

Additional information

The period is used to animate the “filling” bitmap of the progress bar. This way it is possible to indicate a state where the progress bar is waiting for data. The user has to make sure that the bitmap has the size of the Progbar object and the left and right endings of the bitmap match each other.

7.15 QRCode

A QRCode object displays a QR code. A custom text can be set which will then be converted into a QR code.

Symbol	Example
	

Properties

Property	Description
`Error correction level`	Error correction level for QR code.
`Pixelsize`	Size in pixels of one module for QR code.
`Text`	Text used to be encoded in QR code.
`Version`	Dimensions of the code.

Error correction level

The error correction level is a specific parameter of a QR code. The higher the error correction level, the more information is saved redundantly in the QR code in order to increase the chance to be read without errors.

emWin QR codes offer four error correction levels.

Error correction level	Description
`GUI_QR_ECLEVEL_L`	About 7% or less errors can be corrected.
`GUI_QR_ECLEVEL_M`	About 15% or less errors can be corrected.
`GUI_QR_ECLEVEL_Q`	About 25% or less errors can be corrected.
`GUI_QR_ECLEVEL_H`	About 30% or less errors can be corrected.

Pixelsize

The pixelsize property defines the size in pixels of one module in a QR code.

Version

The version of a QR code indicates the overall dimensions of the code. The entered value has to be between 0 and 40. If 0 is entered, the appropriate version is chosen automatically.

7.16 Rotary

A rotary object is similar to its emWin counterpart. A Rotary object is a circular object that can be rotated. The object consists of a background and a marker, both which make use of a bitmap. When rotating the object, the marker moves along the rotary axis. Depending on how the user set the scale, values are returned for the rotated degree.

Symbol	Example
	

Properties

Property	Description
Bitmaps	• Background: Bitmap used for background of Rotary. • Marker: Bitmap used for marker of Rotary.
Initial value	Initial value of Rotary.
Range	• Positive: Starting rotation angle in 10th of degrees. • Negative: Ending rotation angle in 10th of degrees.
Span of values	• Min: Lowest value the object should return. • Max: Highest value the object should return.
Offset	This offset angle will make the Rotary appear rotated by that angle from the beginning. The offset is measured in 10th of degrees (3600 = 360°).
Radius	Radius of the Rotary.
Rotate marker	Enables marker rotation.
Marker alignment	Sets an alignment and additional offset to the marker bitmap.
Period	Period how long the marker moves when released.
Snap position	Sets snap positions on the Rotary (in 10th of degrees).
Focus options	Disables the focus for the Rotary or hides it.

Rotate marker

When activated the marker bitmap is rotated when the Rotary is moved.

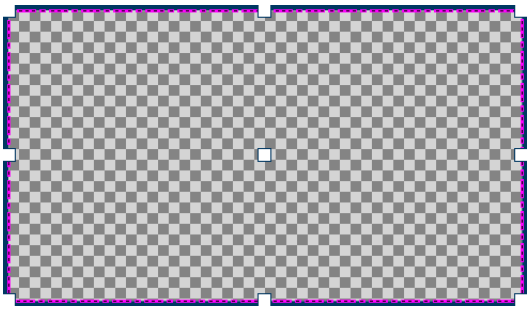
Without marker rotation	With marker rotation
	

Snap position

This property sets a position on the Rotary object at which it should snap in place. The snap position is specified in 10ths of degrees (1800 = 180°).

7.17 Screen

A screen is an invisible parent object for all other objects. An application consists of one or more screens. Interactions are also assigned to one screen.

Symbol	Example
	

Properties

Property	Description
`Horizontal motion`	Horizontal motion screen/window partner.
`Vertical motion`	Vertical motion screen/window partner.
`Persistent mode`	Enables persistent mode for screen.

Horizontal / vertical motion

In case of enabling motion without specifying a partner, the window is simply movable (horizontal and/or vertical) within the range of the parent window.

Persistent mode

The persistent mode property allows a screen to be persistent, so it does not get deleted during runtime when it is not visible anymore. It makes sense to use this mode, when the widgets in a screen are showing values which should not get deleted.

> **Note**
>
> All screens that are in persistent mode are created on start-up of the application.

7.18 Slider

A Slider object is, like the emWin SLIDER widget, a movable thumb on a shaft. By moving the thumb on the shaft, values can be selected.

Symbol	Example
	

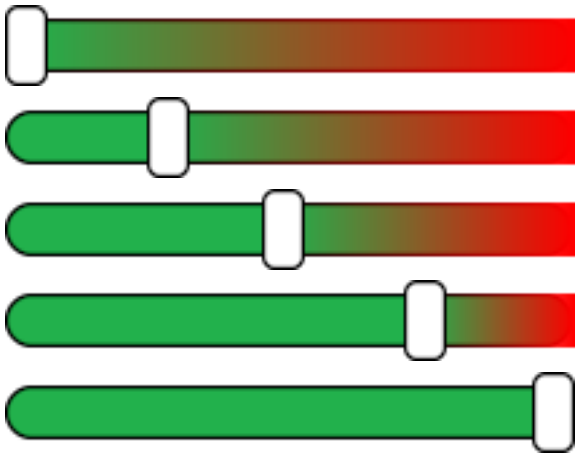
Properties

Property	Description
`Vertical mode`	Enables vertical mode.
`Invert direction`	Inverts direction of slider.
`Bitmaps`	• Shaft left: Bitmap of left part of the shaft. • Shaft right: Bitmap of right part of the shaft. • Thumb up: Bitmap of unpressed thumb. • Thumb down: Bitmap of pressed thumb.
`Blend colors`	• Shaft left: Color on left part of the shaft to be blended in. • Shaft right: Color on right part of the shaft to be blended in.
`Range`	• Min: Minimum value of slider. • Max: Maximum value of slider.
`Focus options`	Disables the focus for the Slider or hides it.

Blend colors

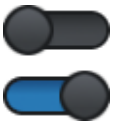
The blend colors setting makes it possible to choose a color for the left and/or right side of the shaft to blended into the corresponding bitmap.

The below example uses the green shaft bitmaps provided as stock images by the AppWizard and has the color red set as the blend color for the right side of the shaft.



7.19 Switch

A switch object works like a switch present on most modern smartphones. It has two states and can be toggled by clicking on it.

Symbol	Example
	

Properties

Property	Description
`Bitmaps`	• BG-Left: Background bitmap for left state. • BG-Right: Background bitmap for right state. • BG-Disabled: Background bitmap for disabled state. • Thumb-Left: Thumb bitmap for left state. • Thumb-Right: Thumb bitmap for right state. • Thumb-Disabled: Thumb bitmap for disabled state.
`Left text`	Text displayed in left state.
`Right text`	Text displayed in right state.
`Text colors`	• Text left: Text color in left state. • Text right: Text color in right state.
`Disable animation`	Disables the animation when toggling between the states.
`Font`	Font to be used for the object.
`Period`	Animation period when clicking the switch.
`Fade mode`	Enables fade mode which fades the background bitmaps when switching from right to left state.
`Focus options`	Disables the focus for the Switch or hides it.

Fade mode and disclose mode

By default, a Switch object uses the **disclose mode**, which means that when the switch animation is performed or when the thumb is moved, the old state bitmap will disappear while the new state bitmap will be disclosed.

When set to **fade mode**, while the switch animation is performed or when the thumb is moved, the old state bitmap will fade into the new state bitmap.

7.20 Text

A text object is similar to its emWin counterpart, it is an object displaying a text resource or a decimal value at a specified position.

Symbol	Example
	

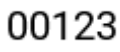
Properties

Property	Description
Text color	Text color to be used.
Background color	Background color of the object.
Framed font color	Color of the object used for framed fonts.
Text	Text to be displayed.
Decimal mode	Enables decimal mode.
Hexadecimal mode	Enables hexadecimal mode.
Text alignment	Text alignment.
Font	Font to be used.
Text wrapping	Enables text wrapping.
Text rotation	Text rotation mode.
Untouchable	Sets the text object as untouchable and sends any touch input to the object below.

Decimal mode

Just as decimal mode for the Edit object, with this setting the Text object is only eligible of holding digits instead of characters. For this mode, a mask of zeros has to be specified which determines how many digits are shown by the object. More details about the usage of the mask is explained under *Decimal mode* on page 74.

Also, when using decimal mode, a range property is added to the object to limit the numbers that can be entered. More on the range property can be found under *Range* on page 91.



7.21 Timer

A timer object represents a `GUI_TIMER` that can be set to a custom time period and optionally restarted.

The timer is an object, although unlike the other objects that represent widgets, the timer object does not have window-specific properties such as position and size. Because of that, it is also not visible on the screen.

However, it is visible in the hierarchical object tree on the left side of the AppWizard.

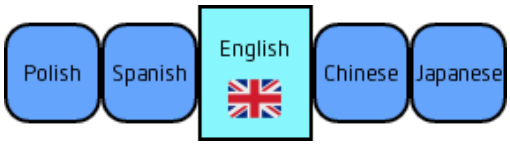
Properties

Property	Description
`Timer period`	Period of timer.
`Auto-restart mode`	When this mode is activated, the timer will be restarted again when it run out using the same time period.

7.22 Wheel

The Wheel object is a swipeable rotating list of items that can show multiple lists of texts and/or bitmaps.

There are two ways of how the Wheel can be created: **morph mode** and **plain mode**. Examples for each mode can be found further below.

Symbol	Example
	

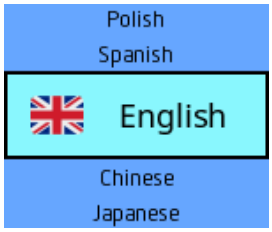
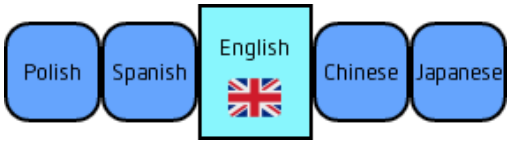
Properties

Property	Description
Morph mode	Toggles between morph mode and plain mode.
Endless mode	Toggles between endless mode and stop mode.
Horizontal mode	Sets the orientation of the widget to horizontal or vertical.
Period until stop	Period in ms it takes until the motion movement of the widget stops.
Cell- and center size	• Cell: Width/height of non-center cells. • Center: Width/height of center cells.
Colors	• Background Cell: Background color of non-center cells. • Background Center: Background color of center cells. • Background: Background color used for the widget.
Text	Allows for adding lists of texts to the Wheel.
Images	Allows for adding lists of bitmaps to the Wheel.
Border	Horizontal/vertical border that can be used to reduce the cells in horizontal/vertical size.
Overlay bitmaps	Optional bitmaps that are drawn above the Wheel. A maximum of three overlay bitmaps can be added.
Overlay alignment	Alignment of the overlay bitmaps.
Overlay colors	Colors to be used for the overlay bitmaps (alpha bitmaps only).
Center frame	• Center frame size: Frame width in pixels of the frame drawn around the center cell. • Center frame radius: Frame radius in pixels of the frame drawn around the center cell.
Color of center frame	Color used to draw the frame around the center cell.
“Morph mode”-only properties	
Size of cutaway	Amount of pixels that are “cut away” (used as spacing) from the non-center cells.
Alignment of cutaway	Alignment of the cutaway.

Morph mode and plain mode

In plain mode, when a cell is moving into the center, its properties are clipped from the non-center properties to the center properties.

In morph mode, while a cell is moving into the center, its properties (like text position, bitmaps, frame radius, …) are morphed into the different center properties.

Plain mode	Morph mode
	

Endless mode

If endless mode is active, the Wheel will not stop at the first or last cell when the user is swiping through the cells. Instead, it will scroll through the cells continuously.

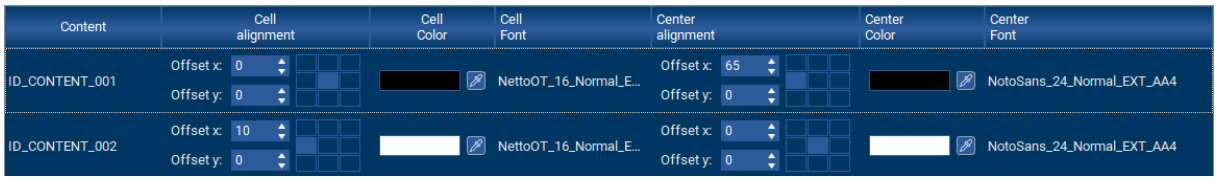
If endless mode is toggled off, stop mode will be active which means that the Wheel will stop at the first and last item of the Wheel.

Text

All previously defined content objects defined in the lists window (see *Lists window* on page 49) can be added as text to the Wheel. It is possible to add multiple lists of text.

Based on whether the Wheel is in morph or plain mode, the following properties can be set to the text:

Cell type	Plain mode	Morph mode
Non-center cell	• Text alignment • Text alignment offset • Text color • Font	• Text alignment • Text alignment • Text color • Font
Center cell	• Text alignment • Text alignment offset • Text color • Font	• Text alignment • Text color



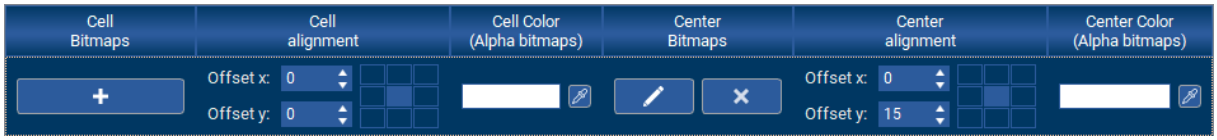
Images

Similarly to the text lists, multiple lists of bitmaps can also be added to the Wheel.

Bitmaps can be added to the non-center cells and/or the center cells.

The following properties can be set to center and non-center bitmaps:

- Bitmap alignment
- Bitmap alignment offset
- Bitmap color (only for alpha bitmaps)



Overlay

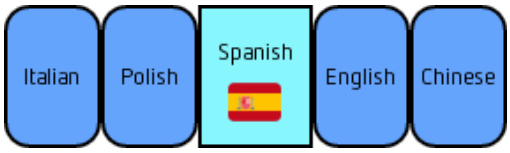
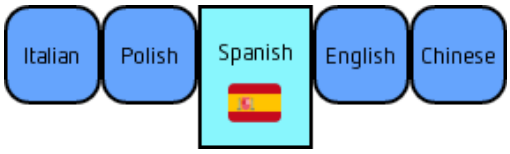
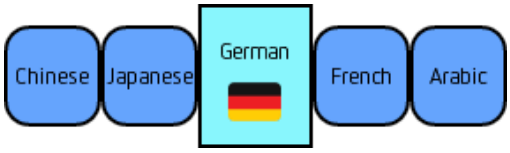
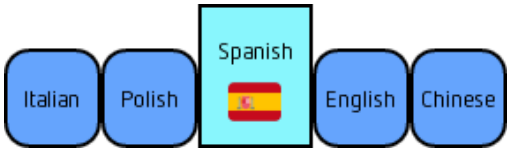
The overlay feature allows a maximum of three bitmaps to be added to the Wheel. These overlay bitmaps are drawn at a certain position above the Wheel items.

This can be used to create certain visual effects, e.g. a semi-transparent bitmap can be used as an overlay to highlight the center cell, as shown below.

Without overlay	With overlay
	

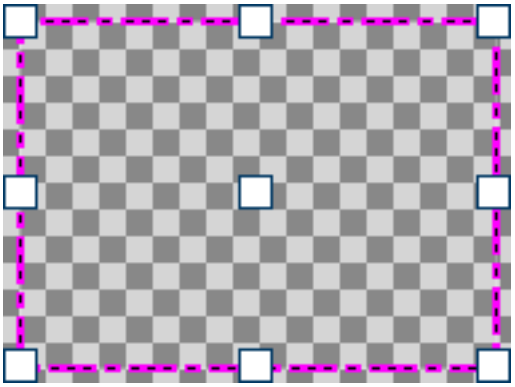
Cutaway

Cutaway is the distance that is removed from the non-center cells in morph mode. The cutaway alignment can be used to define how the cutaway distance should be distributed.

Cutaway options	Output
No cutaway	
Cutaway of 25px, top aligned	
Cutaway of 25px, center aligned	
Cutaway of 25px, bottom aligned	

7.23 Window

A window works similar to a screen. It is also invisible and serves as parent object for objects. Moving/animating the window also moves its objects. A window can have further child windows. That makes it possible to achieve a hierarchic structure for complex dialogs.

Symbol	Example
	

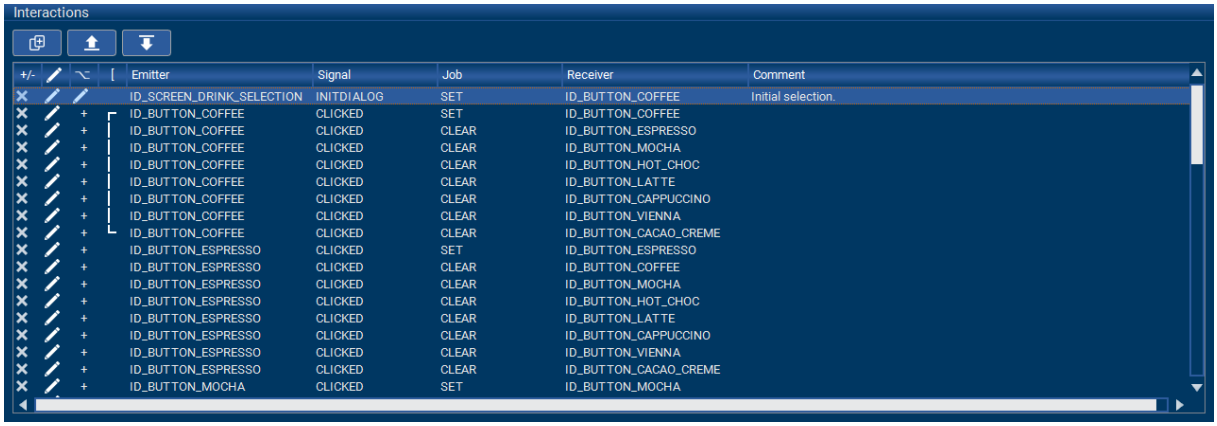
Properties

Property	Description
`Horizontal motion`	Horizontal motion screen/window partner.
`Vertical motion`	Vertical motion screen/window partner.
`Opaque mode`	Sets the button to opaque and removes its transparency flag.

Chapter 8

Interactions

The AppWizard’s interaction window makes it possible to define the application’s behavior on certain actions. Interactions are always assigned to a screen, meaning two different screens have different interactions.



8.1 Introduction

This section will explain how to set up interactions and describe the terms.

❶ Select an emitter

First, an emitter for a signal has to be selected. The **emitter** specifies the ID of the widget or variable that has to send out a certain signal in order for the interaction's job to be executed.

❷ Select the signal

The second step is to select the signal. The **signal** is the event that has to occur for the job to be executed. This could be e.g. `WM_NOTIFICATION_CLICKED`, which occurs when a widget was clicked.

For a list of all available signals, see the chapter *List of signals* on page 135.

❸ Select the job

The third step is to select a job for this interaction. The **job** specifies a certain action that will be done when the above mentioned signal has occurred. This could for example be SETTEXT to set the text of an Edit object.

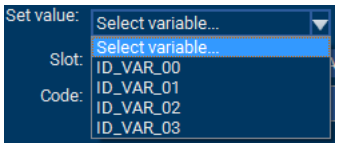
For a list of all available jobs, see the chapter *List of jobs* on page 155.

❹ Select the receiver

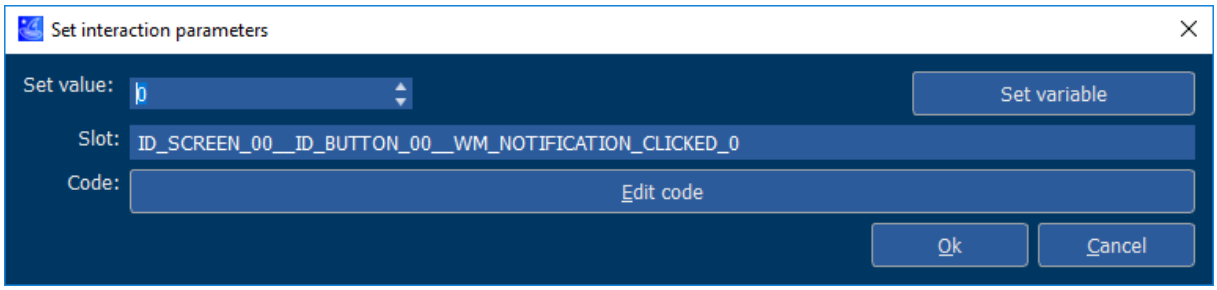
The last step is to select is a receiver for the interaction. The **receiver** specifies the ID of the widget or variable the job will be executed for. For example, if the job is SETTEXT, the receiver has to be an Edit object, whose text will then be set.

❺ Set up interaction parameters

The final step is to define what the action/job should do with the receiver. This can be done by clicking on the 'Edit' symbol of an interaction to set up interaction-specific parameters. For example for the job ADDVALUE, the user has to specify the value that will be added to the receiver.



Instead of a permanent value, the user is also able to select a variable. To do this, click the **Set variable** button and select a variable from the dropdown menu.



In the 'Slot' field, the user can see and may change the name of the slot routine. The slot routine is the routine, that will be executed for this interaction.

> **Note**
>
> The name of the slot routine must be unique! Otherwise the user code won't compile.

❻ Add a condition to the interaction (optional)

Optionally, a condition can be set up for the interaction. This condition determines whether or not the job of the interaction will be executed.

To add a condition to an interaction, click the plus symbol in the condition column. More information about conditions can be read under *Conditions* on page 202.

❼ Add custom user code to the interaction (optional)

The user may edit/insert C code that will be executed upon this interaction. The code may be added via the "Edit code" dialog or externally via an editor or IDE. More information about slot routines and where they are located can be read in the chapter *Slot routines* on page 223.

> **Note**
>
> The user must not add custom routines to the C files that contain the generated slot routines! More information about how the user can properly add their own code can be read under *Custom user code* on page 225.

8.2 List of signals

The following section will provide a list of all available signals the user can choose from for an interaction.

Signal	Description
ANIMEND	Emitted when an animation has ended.
ANIMSTART	Emitted when an animation has started.
CLICKED	When the user clicks on an object.
CREATE	Emitted when an object was created.
DELETE	Emitted when an object was deleted.
ENTER_PRESSED	Emitted when ENTER key is pressed.
FIXED	Emitted by a screen or window when a shifting operation has ended.
GOT_FOCUS	When an object gets the focus.
INITDIALOG	Emitted right after the application has started.
LOST_FOCUS	When an object lost its focus.
MOTION	Emitted when a user moves a screen by dragging it.
MOTION_STOPPED	When the motion of a Rotary object has stopped.
PIDPRESSED	Screen received touch input (pressed).
PIDRELEASED	Screen lost touch input (released).
RELEASED	Once a click on an object has been released.
TEXT_CHANGED	Emitted when the text of an object has changed.
TIMER	Emitted when a given timer has run out.
UNPINNED	Emitted by a screen or window when a shifting operation has started.
VALUE_CHANGED	If the value of an object has changed.

8.2.1 ANIMEND

Description

This signal is emitted by an object after an animation paired to the object has ended.

Emitting objects

- *Box* object
- *Button* object
- *Edit* object
- *Gauge* object
- *Image* object
- *Progbar* object
- *Rotary* object
- *Slider* object
- *Switch* object
- *Text* object
- *Variables*
- *Window* object

8.2.2 ANIMSTART

Description

This signal is emitted by an object after an animation paired to the object has started.

Emitting objects

- *Box* object
- *Button* object
- *Edit* object
- *Gauge* object
- *Image* object
- *Progbar* object
- *Slider* object
- *Switch* object
- *Text* object
- *Variables*
- *Window* object

8.2.3 CLICKED

Description

This signal is emitted when the user clicks on an object.

Emitting objects

- *Button* object
- *Edit* object
- *Rotary* object
- *Slider* object
- *Switch* object
- *Wheel* object

8.2.4 CREATE

Description

This signal is emitted right after an object has been created. The Window Manager equivalent is `WM_CREATE`.

Emitting objects

- *Screen* object

8.2.5 DELETE

Description

This signal is emitted right after an object has been deleted. The Window Manager equivalent is `WM_DELETE`.

Emitting objects

- *Screen* object

8.2.6 ENTER_PRESSED

Description

This signal is emitted when the ENTER key is pressed.

Emitting objects

- *Keyboard* object

8.2.7 FIXED

Description

This signal is emitted by screens or windows when a shifting operation has ended.

Emitting objects

- *Screen* object
- *Window* object

See also

- UNPINNED
- SHIFTSCREEN
- SHIFTWINDOW

8.2.8 GOT_FOCUS

Description

This signal is emitted when an object has gotten the focus.

Emitting objects

- *Button* object
- *Edit* object
- *Multiedit* object
- *Rotary* object
- *Slider* object
- *Switch* object
- *Wheel* object

8.2.9 INITDIALOG

Description

This signal is emitted right after the application has started. The Window Manager equivalent is `WM_INIT_DIALOG`.

Emitting objects

- *Screen* object

8.2.10 LOST_FOCUS

Description

This signal is emitted when an object lost its focus.

Emitting objects

- *Button* object
- *Edit* object
- *Multiedit* object
- *Rotary* object
- *Slider* object
- *Switch* object
- *Wheel* object

8.2.11 MOTION

Description

This signal is emitted when a screen object has been moved by the user dragging it. The Window Manager equivalent is `WM_MOTION`.

Emitting objects

- *Screen* object

8.2.12 MOTION_STOPPED

Description

This signal is emitted when the motion of a Rotary object has stopped.

Emitting objects

- *Rotary* object

8.2.13 PIDPRESSED

Description

This signal is emitted by screens when PID is pressed. The signal is sent to all existing screens in the project.

Emitting objects

- *Screen* object

8.2.14 PIDRELEASED

Description

This signal is emitted by screens when PID is released. The signal is sent to all existing screens in the project.

Emitting objects

- *Screen* object

8.2.15 RELEASED

Description

This signal is emitted once a click on an object has been released.

Emitting objects

- *Button* object
- *Edit* object
- *Rotary* object
- *Slider* object
- *Switch* object
- *Wheel* object

8.2.16 TEXT_CHANGED

Description

This signal is emitted when the text of an object has been changed.

Emitting objects

- *Edit* object
- *QRCode* object
- *Text* object

8.2.17 TIMER

Description

This signal is emitted when a given timer has run out. To run a timer, a timer object has to be created and started using the job START.

Emitting objects

- *Timer* object

8.2.18 UNPINNED

Description

This signal is emitted by screens or windows when a shifting operation has started.

Emitting objects

- *Screen* object
- *Window* object

See also

- FIXED
- SHIFTSCREEN
- SHIFTWINDOW

8.2.19 VALUE_CHANGED

Description

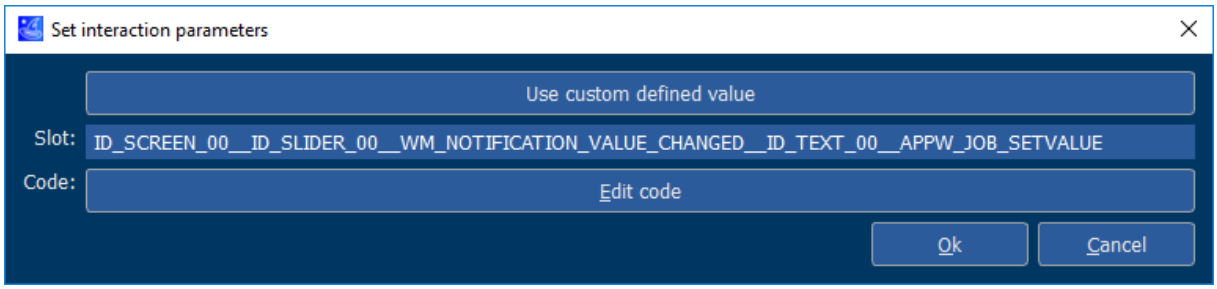
This signal is emitted when the value of an object has changed.

Emitting objects

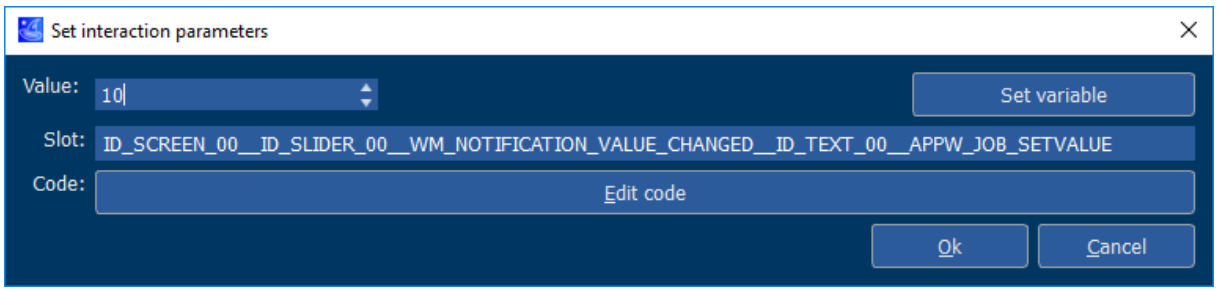
- *Button* object
- *Edit* object
- *Gauge* object
- *Progbar* object
- *Rotary* object
- *Slider* object
- *Switch* object
- *Text* object
- *Variables*
- *Wheel* object

Additional information

By default, the custom value option is disabled. This means, the value of the emitting object will be directly passed to the receiver and process the value depending on the selected job. This can be useful for jobs like SETVALUE, but it certainly does not work for all jobs.



When clicking the button **Use custom defined value**, a custom value can be entered, which will be sent to the receiver.



8.3 List of jobs

Job	Description
ADDVALUE	Adds a given increment to the given object.
ADDITEM	Appends text from object/resource to a given object.
ANIMCREATE	Creates an animation.
ANIMSTART	Starts an animation.
ANIMSTOP	Stops a running animation.
CALC	Calculates the value of a variable using the given term.
CLEAR	Clears the state of the given object.
CLOSESCREEN	Closes a given screen to go back to the screen that is behind.
DELITEM	Deletes the specified item from the given object.
ENABLEPID	Enables or disables PID input.
INSITEM	Insert text to a given object.
INVALIDATE	Invalidates a given object.
MODALMESSAGE	Creates and shows a modal dialog.
MOVETO	Moves an object's selection to a given value with an animation.
ROTATEDISPLAY	Rotates the display to the desired orientation.
SET	Sets the state of the given object.
SETALPHA	Sets the alpha value of an object.
SETANGLE	Sets the rotation angle of an object.
SETBITMAP	Sets a bitmap to an object.
SETBKCOLOR	Sets the background color of the given object.
SETCOLOR	Sets the color of the given object.
SETCOORD	Sets a coordinate.
SETENABLE	Enables the given object.
SETEND	Sets the end value/angle of the given object.
SETFOCUS	Sets focus to a given object.
SETITEM	Text transfer from/to a given object.
SETLANG	Sets the language index of an object.
SETPERIOD	Sets the period of an object.
SETRANGE	Sets the range of the given object.
SETSCALE	Sets a scaling value to an object.
SETSIZE	Sets the size of the given object.
SETSTART	Sets the start value/angle of the given object.
SETTEXT	Sets the text of the given object.
SETVALUE	Sets a value.
SETVIS	Makes the given object visible.
SETX0	Sets the x0-coordinate of an object.
SETY0	Sets the y0-coordinate of an object.
SETX1	Sets the x1-coordinate of an object.
SETY1	Sets the y1-coordinate of an object.
SHIFTSCREEN	Shifts into the given screen using the given method.

Job	Description
SHIFTWINDOW	Shifts in a window using the given method.
SHOWSCREEN	Makes the given screen visible.
START	Starts a given timer object.
STOP	Stops a given timer object.
SWAPSCREEN	Swaps the screen to the given screen.
TOGGLE	Toggles the state of the given object.
NULL	Used for only executing custom user code.

8.3.1 ADDVALUE

Description

Adds a given increment to the given object.

Receiving objects

- *Text* object
- *Progbar* object
- *Rotary* object

Interaction parameters of dialog

Parameter	Description
`Value`	Value to be added.

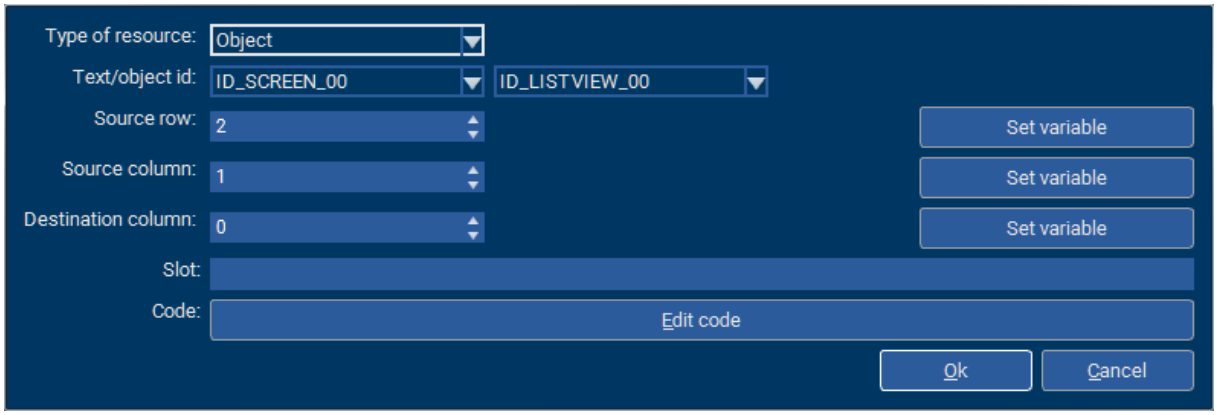
Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	Value to be added.

8.3.2 ADDITEM

Description

This job is used for the transaction of text. The source of the text to be added can be either a text from the text resources or a text-based object from any screen. It should be noted that the corresponding object must be in memory. The object specified as 'Receiver' serves as the target of the transaction. The following dialog is used to specify the job:



The fields 'Source row' and 'Source column' are not used in any case. If the source is an object with only one field, such as an EDIT or a TEXT object, both specifications are ignored. If the source is a LISTBOX or a DROPDOWN object, only 'Source row' is used. Only with a LISTVIEW object both specifications are used. The same applies to the target object. The 'Destination column' field is only needed if the target object is a LISTVIEW object.

Receiving objects

- *Dropdown* object
- *Listbox* object
- *Listview* object
- *Wheel* object

Interaction parameters of dialog

Parameter	Description
`Type of resource`	Determines if an object or a text resource should be used.
`Text/object id`	Screen- and object id of object or resource id of text.
`Source row`	Used in case of a source object with multiple rows.
`Source column`	Used in case of a source object with multiple columns.
`Destination column`	Column to add the text to. Only used if target object is a LISTVIEW.

Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	Type: 0 = text resource, 1 = object
`aPara[1].v`	0: text resource Id, 1: HB/LB: Screen Id/Object Id
`aPara[2].v`	Source: Row index (Listview, Dropdown, Listbox)
`aPara[3].v`	Source: Column index (Listview only)
`aPara[4].v`	Destination: Column index (Listview only)

8.3.3 ANIMCREATE

Description

Creates an animation that has previously been defined in the animation interface.

Interaction parameters of dialog

Parameter	Description
`Animation Id`	ID of the predefined animation.

Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	ID of the predefined animation.

Additional information

More information about how animations can be created can be read under *Animations* on page 210.

8.3.4 ANIMSTART

Description

Starts an animation that has been previously defined and created with the job ANIMCREATE.

Interaction parameters of dialog

Parameter	Description
`Animation Id`	ID of the animation.
`Number of loops`	Number of loops the animation should run. -1 if it should run endlessly.

Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	ID of the animation.
`aPara[1].v`	Number of loops the animation should run. -1 if it should run endlessly.

8.3.5 ANIMSTOP

Description

Creates an animation that has previously been defined in the animation interface.

Interaction parameters of dialog

Parameter	Description
`Animation Id`	ID of the animation.
`Delete animation`	If the animation should be deleted after it has been stopped.

Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	ID of the animation.
`aPara[1].v`	1 if the animation should be deleted after stopping, 0 if it should remain.

8.3.6 CALC

Description

Calculates the new value of a variable using the set term. If this results in a changed value, the variable will emit a `VALUE_CHANGED` signal.

Receiving objects

- *Variables*

Term calculation

A detailed description on how a calculation term can be added to a variable can be found under Calculations.

8.3.7 CLEAR

Description

Sets the state of the given object to its default state. For example, when executing this job on a Switch object, it will be set to the ’left state’.

Receiving objects

- *Button* object
- *Switch* object

8.3.8 CLOSESCREEN

Description

Closes a given screen. When the screen is closed, the screen that was behind is shown again.

Interaction parameters of dialog

Parameter	Description
Screen ID	ID of the screen to be closed.

Job-specific parameters passed to slot-routine

Parameter	Description
aPara[0].v	Screen Id.

Additional information

It should be made sure of that there is another screen behind the screen to be deleted, otherwise nothing will be shown. Therefore, the screen opening the screen that performs the CLOSESCREEN job should not be moved out via SHIFTSCREEN. Rather, the other screen should be shown using SHOWSCREEN.

8.3.9 DELITEM

Description

Deletes the given row from the given object.

Receiving objects

- *Dropdown* object
- *Listbox* object
- *Listview* object
- *Wheel* object

Interaction parameters of dialog

Parameter	Description
`Row index`	Index of row to be deleted.

Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	Item index to be deleted.

8.3.10 ENABLEPID

Description

Enables, disables or toggles PID input for the application. This job has no receiving object, since it will alter the state of PID input for the entire application.

Interaction parameters of dialog

Parameter	Description
`Enabled state`	State of PID input for the job: *On*, *Off* or *Toggle*.

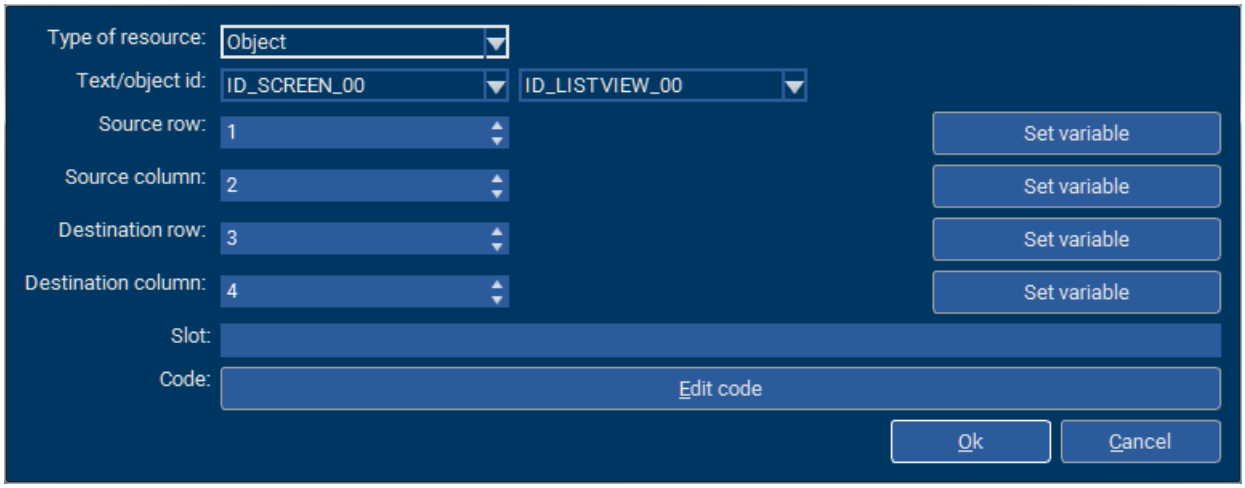
Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	PID input. 1 = On, 0 = Off.

8.3.11 INSITEM

Description

This job is used to insert text into the given object. The source of the text to be set can be either a text from the text resources or a text-based object from any screen. It should be noted that the corresponding object must be in memory. The object specified as 'Receiver' serves as the target of the transaction. The following dialog is used to specify the job:



The fields 'Source row' and 'Source column' are not used in any case. If the source is an object with only one field, such as an EDIT or a TEXT object, both specifications are ignored. If the source is a LISTBOX or a DROPDOWN object, only 'Source row' is used. Only with a LISTVIEW object both specifications are used. The same applies to the target object.

Receiving objects

- *Dropdown* object
- *Listbox* object
- *Listview* object
- *Wheel* object

Interaction parameters of dialog

Parameter	Description
`Type of resource`	Determines if an object or a text resource should be used.
`Text/object id`	Screen- and object id of object or resource id of text.
`Source row`	Used in case of a source object with multiple rows
`Source column`	Used in case of a source object with multiple columns
`Destination row`	Used in case of a destination object with multiple rows
`Destination col-umn`	Used in case of a destination object with multiple columns

Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	Type: 0 = Text resource, 1 = Object
`aPara[1].v`	0: Text resource Id, 1: HB/LB: Screen Id/Object Id
`aPara[2].v`	Source: Row index (Listview, Dropdown, Listbox)
`aPara[3].v`	Source: Column index (Listview only)
`aPara[4].v`	Destination: Row index (Listview, Dropdown, Listbox)
`aPara[5].v`	Destination: Column index (Listview only)

8.3.12 INVALIDATE

Description

Triggers a redraw (invalidates) the given object.

Receiving objects

This job can be executed for all objects.

8.3.13 MODALMESSAGE

Description

Shows a given screen above the current screen as a modal message dialog. To hide the modal screen, the job `CLOSESCREEN` has to be executed on the modal screen.

Receiving objects

- *Screen* object

Interaction parameters of dialog

Parameter	Description
`Screen ID`	ID of the screen to be shown as a modal message.

Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	Screen ID.

8.3.14 MOVETO

Description

Moves an object’s selection to a given value with an animation.

Receiving objects

- *Wheel* object

Interaction parameters of dialog

Parameter	Description
`Value`	New index.

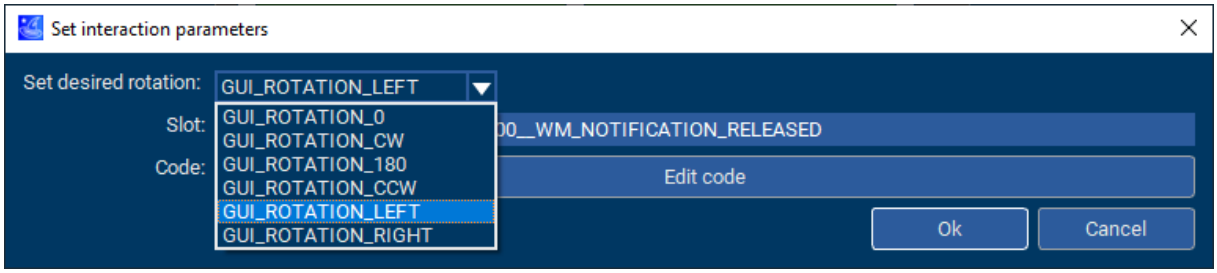
Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	New index.

8.3.15 ROTATEDISPLAY

Description

Rotates the display to the desired orientation.



The field ’Set desired orientation’ allows to set a specific orientation or turn to the display clockwise (right) or counterclockwise (left).

Interaction parameters of dialog

Parameter	Description
`Rotation`	Rotation command.

Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	Rotation command

Rotation command

The following table shows the available commands:

Command	Description
`GUI_ROTATION_0`	Default orientation
`GUI_ROTATION_CW`	Clockwise rotation 90°
`GUI_ROTATION_180`	Upside down 180°
`GUI_ROTATION_CCW`	Counterclockwise rotation 90°
`GUI_ROTATION_LEFT`	Counterclockwise rotation to the left by 90°
`GUI_ROTATION_RIGHT`	Clockwise rotation to the right by 90°

Additional information

Note that in case of using hardware accelleration like D/AVE 2D or Chrom-ART the hardware accelleration will be disabled in other rotation modes than the default orientation.

8.3.16 SET

Description

Sets the state of the given object to its “pressed” state. This means, e.g. when executed on a Button object, it will be in its pressed state and when executed on a Switch object it will be in its ‘right state’.

Receiving objects

- *Button* object
- *Switch* object

8.3.17 SETALPHA

Description

Sets the alpha value of an Image object. This job only has an effect if an alpha bitmap is set to the Image object.

Receiving objects

- *Image* object

Interaction parameters of dialog

Parameter	Description
`Alpha`	New alpha value to be set.

Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	New alpha value to be set.

8.3.18 SETANGLE

Description

Sets the rotation angle of an Image object.

Receiving objects

- *Image* object

Interaction parameters of dialog

Parameter	Description
`Angle`	New rotation angle to be set, angle in degrees * 1000 (e.g. 45000 equals 45°).

Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	New rotation angle to be set.

8.3.19 SETBITMAP

Description

Sets a bitmap to an image or button object.

Receiving objects

- *Button* object
- *Image* object

Interaction parameters of dialog

Parameter	Description
`Bitmap`	New bitmap to be set.
`Index`	Index of button bitmap. See below for more information.

Bitmap index

The bitmap index parameter is only used if the receiving object is a Button. The index is used to determine the state the bitmap is used for.

Index	Bitmap
0	Bitmap for unpressed state.
1	Bitmap for pressed state.
2	Bitmap for disabled state.

8.3.20 SETBKCOLOR

Description

Sets the background color of the given object.

Receiving objects

- *Button* object

Interaction parameters of dialog

Parameter	Description
`Background color`	New background color to be used.

Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	Background color to be used.

8.3.21 SETCOLOR

Description

Sets the color of the given object.

Receiving objects

- *Box* object
- *Button* object
- *Image* object

Interaction parameters of dialog

Parameter	Description
`Color`	New color to be used.
`Index`	Index of button state. See table below.

Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	Color to be used.
`aPara[1].v`	Index of state.

Setting the color of alpha bitmaps

If an alpha bitmap is added to an Image object, the color used for drawing the bitmap can be changed using this job. More information can be found under Image.

Color index

The color index parameter is only used if the receiving object is a Button. The index is used to determine the state the color is used for.

Index	Color
0	Color for unpressed state.
1	Color for pressed state.
2	Color for disabled state.

8.3.22 SETCOORD

Description

Sets a coordinate of an object.

Receiving objects

- *Box* object
- *Button* object
- *Edit* object
- *Image* object
- *Progbar* object
- *Rotary* object
- *Slider* object
- *Switch* object
- *Text* object
- *Window* object

Interaction parameters of dialog

Parameter	Description
`Value`	New coordinate of the object.
`Coordinate`	Axis of the coordinate to be set.

Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	Value.
`aPara[1].v`	Index of coordinate. See *Dispose indexes* on page for a list of legal values.

8.3.23 SETENABLE

Description

Sets the ’enabled’ state of a given object. The receiving object will be either enabled or disabled, depending which ’enabled’ state was specified in the interaction parameters.

Receiving objects

- *Button* object
- *Edit* object
- *Rotary* object
- *Slider* object
- *Switch* object

Interaction parameters of dialog

Parameter	Description
`Enable state`	New enable state of the object. This can be set to either on, off or toggled.

Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	Enable state. 1 = on, 0 = off.

8.3.24 SETEND

Description

Sets the end angle of a given object.

Receiving objects

- *Gauge* object

Interaction parameters of dialog

Parameter	Description
`Ang1`	End angle.

Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	End angle in 10th of degrees.

8.3.25 SETFOCUS

Description

Sets the focus onto a given object.

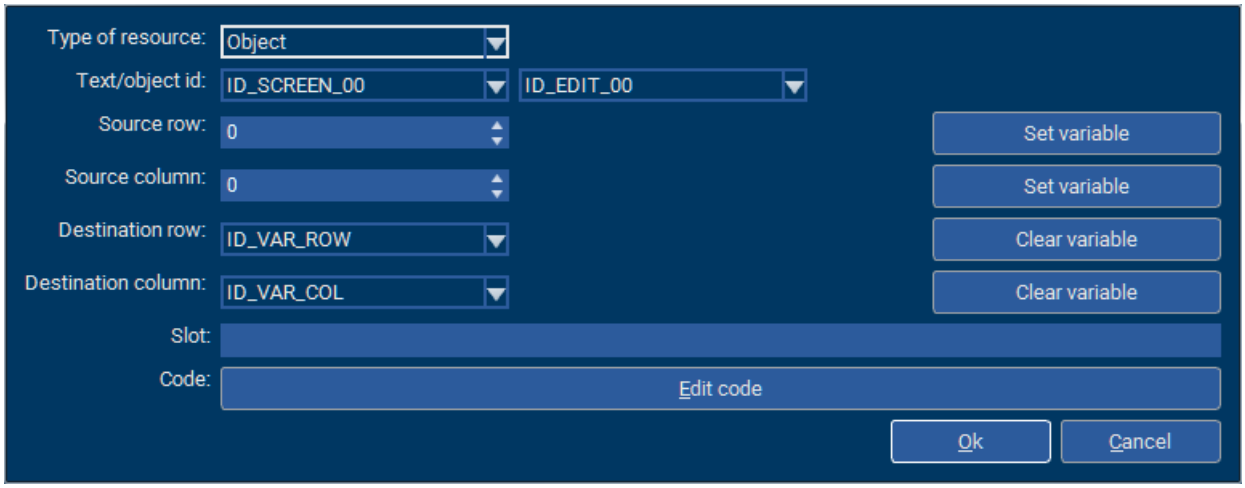
Receiving objects

- *Button* object
- *Edit* object
- *Multiedit* object
- *Rotary* object
- *Slider* object
- *Switch* object

8.3.26 SETITEM

Description

This job is used for the transaction of text. The source of the text to be set can be either a text from the text resources or a text-based object from any screen. It should be noted that the corresponding object must be in memory. The object specified as 'Receiver' serves as the target of the transaction. The following dialog is used to specify the job:



The fields 'Source row' and 'Source column' are not used in any case. If the source is an object with only one field, such as an EDIT or a TEXT object, both specifications are ignored. If the source is a LISTBOX or a DROPDOWN object, only 'Source row' is used. Only with a LISTVIEW object both specifications are used. The same applies to the target object.

Receiving objects

- *Button* object
- *Edit* object
- *Dropdown* object
- *Listbox* object
- *Listview* object
- *Multiedit* object
- *QRCode* object
- *Text* object
- *Wheel* object

Interaction parameters of dialog

Parameter	Description
`Type of resource`	Determines if an object or a text resource should be used.
`Text/object id`	Screen- and object id of object or resource id of text.
`Source row`	Used in case of a source object with multiple rows
`Source column`	Used in case of a source object with multiple columns
`Destination row`	Used in case of a destination object with multiple rows
`Destination column`	Used in case of a destination object with multiple columns

Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	Type: 0 = Text resource, 1 = Object
`aPara[1].v`	0: Text resource Id, 1: HB/LB: Screen Id/Object Id
`aPara[2].v`	Source: Row index (Listview, Dropdown, Listbox)

Parameter	Description
aPara[3].v	Source: Column index (Listview only)
aPara[4].v	Destination: Row index (Listview, Dropdown, Listbox)
aPara[5].v	Destination: Column index (Listview only)

8.3.27 SETLANG

Description

Sets the language of the application to the given index.

Interaction parameters of dialog

Parameter	Description
`Language index`	Index of the new language to be set. The index is the zero-based column number of the language seen in the text management dialog.

Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	Index of language.

8.3.28 SETPERIOD

Description

Sets the period of an object.

Receiving objects

- *Wheel* object

Interaction parameters of dialog

Parameter	Description
`Period`	New period to be set.

Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	New period to be set.

8.3.29 SETRANGE

Description

Sets the range to be used for the given object.

Receiving objects

- *Gauge* object
- *Rotary* object
- *Progbar* object
- *Slider* object

Interaction parameters of dialog

Parameter	Description
`Start`	Start value to be used.
`End`	End value to be used.

Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	Start value.
`aPara[1].v`	End value.

8.3.30 SETSCALE

Description

Sets a scaling value to an Image object.

Receiving objects

- *Image* object

Interaction parameters of dialog

Parameter	Description
`Value`	New scaling value to be set, scaling factor * 1000 (e.g. 2000 equals 200% scale).

Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	New scaling value to be set.

8.3.31 SETSIZE

Description

Sets the size of the given object.

Receiving objects

- *Box* object
- *Button* object
- *Edit* object
- *Image* object
- *Progbar* object
- *Rotary* object
- *Slider* object
- *Switch* object
- *Text* object
- *Window* object

Interaction parameters of dialog

Parameter	Description
`Value`	New size value.
`Dimension`	Either X- or Y-axis where the new size value should be applied to.

Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	Value to be used.
`aPara[1].v`	Index of axis.

Additional information

In order for this job to work, the size of the object must be editable. If all coordinates are relative, there is no size to be edited.

8.3.32 SETSTART

Description

Sets the start angle of a given object.

Receiving objects

- *Gauge* object

Interaction parameters of dialog

Parameter	Description
`Ang0`	Start angle.

Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	Start angle in 10th of degrees.

8.3.33 SETTEXT

Description

Sets the text of a given object.

Receiving objects

- *Text* object
- *Button* object

Interaction parameters of dialog

Parameter	Description
`Text`	ID of the text to be used.

Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	Text Id. Only if aPara[0].p = `NULL`.
`aPara[0].p`	Handle. Only if aPara[0].v < 0.

8.3.34 SETVALUE

Description

With this job the value of an object can be set. For most objects, this is a numerical value, except for the Text and Edit objects, where this job sets the corresponding text.

Receiving objects

- *Button* object
- *Edit* object
- *Gauge* object
- *Progbar* object
- *Rotary* object
- *Slider* object
- *Switch* object
- *Text* object
- *Variables*
- *Wheel* object

Interaction parameters of dialog

Parameter	Description
`Value`	New value or text to be set to the object.

Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	Value to be set.

Additional information

Instead of a permanent value, the user can also choose a variable.

8.3.35 SETVIS

Description

Sets the visibility of the given object to either on or off.

Receiving objects

- *Box* object
- *Button* object
- *Edit* object
- *Image* object
- *Progbar* object
- *Rotary* object
- *Slider* object
- *Switch* object
- *Text* object
- *Window* object

Interaction parameters of dialog

Parameter	Description
`Visibility`	New visibility of the object. This can be either set to on, off or toggled.

Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	Visibility flag. 1 = on, 0 = off.

8.3.36 SETX0

8.3.37 SETY0

8.3.38 SETX1

8.3.39 SETY1

Description

Sets the corresponding coordinate of an object.

Interaction parameters of dialog

Parameter	Description
`Value`	Value to be set.

Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	Value to be set.

8.3.40 SHIFTSCREEN

Description

Shifts into the given screen with an animation that the user defines.

Receiving objects

- *Screen* object

Interaction parameters of dialog

Parameter	Description
`Screen ID`	ID of the screen to be shifted in.
`Edge`	Edge the old screen should be moved to.
`Ease`	Animation style to be used. See the chapter ’Animations’ in the emWin manual for reference.
`Period`	Period in ms how long the animation will last.
`Disclose`	If disclose mode should be used.

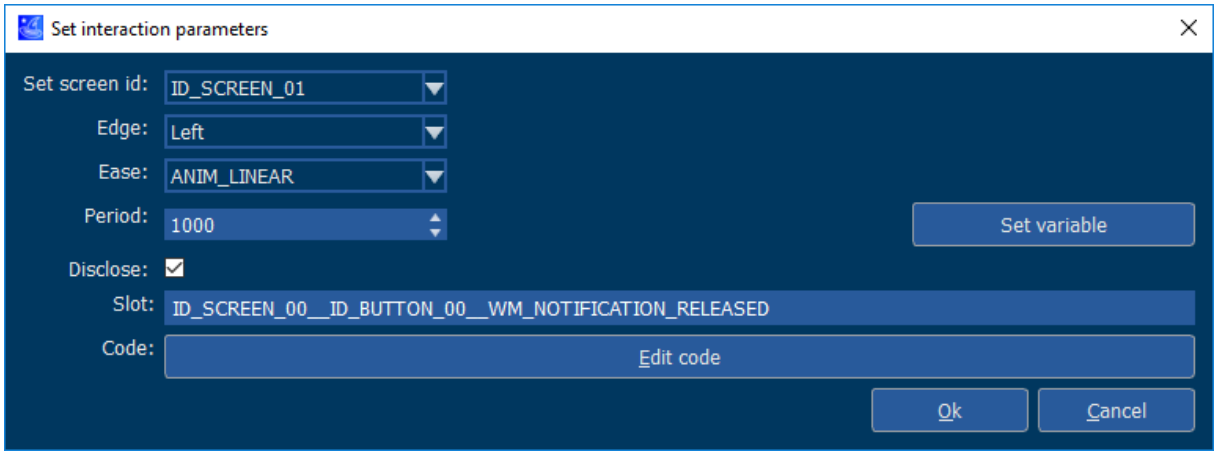
Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	Screen Id.
`aPara[1].v`	Index of edge.
`aPara[2].pFunc`	Pointer to ease function.
`aPara[3].v`	Animation period.
`aPara[4].v`	If 1, disclose mode is used.

Additional information

Note that screens that are not being marked as persistent (see *Persistent mode* on page) will be deleted after they have been faded out.

Example



8.3.41 SHIFTWINDOW

Description

Shifts a window in with a user-defined animation. This job is similar to SHIFTSCREEN.

Interaction parameters of dialog

Parameter	Description
`Window ID`	ID of the window to be shifted in.
`Edge`	Edge of the screen the window should be moved to.
`Ease`	Animation style to be used. See the chapter ’Animations’ in the emWin manual for reference.
`Period`	Period in ms how long the animation will last.
`Disclose`	If disclose mode should be used.

Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	Window Id.
`aPara[1].v`	Index of edge.
`aPara[2].pFunc`	Pointer to ease function.
`aPara[3].v`	Animation period.
`aPara[4].v`	If 1, disclose mode is used.

8.3.42 SHOWSCREEN

Description

This job makes the given screen instantly visible. There are no animation options for this job.

Receiving objects

- *Screen* object

Interaction parameters of dialog

Parameter	Description
`Screen ID`	ID of the screen to be shown.

Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	Screen Id.

Additional information

Note that screens that are not being marked as persistent (see *Persistent mode* on page) will be deleted after they have been faded out.

8.3.43 START

Description

This job starts a given timer with the set period.

Receiving objects

- *Timer* object

8.3.44 STOP

Description

This job stops a given timer.

Receiving objects

- *Timer* object

8.3.45 SWAPSCREEN

Description

Swaps the screen to the given screen without an animation.

Receiving objects

- *Screen* object

Interaction parameters of dialog

Parameter	Description
`Screen ID`	ID of the screen to be shown.

Job-specific parameters passed to slot-routine

Parameter	Description
`aPara[0].v`	Screen Id.

8.3.46 TOGGLE

Description

Toggles the ’pressed’ state of the given object. For example, when executing this job on a Switch it will toggle between its left and right state and when executing on a Button, it will toggle between its pressed and unpressed state.

Receiving objects

- *Button* object
- *Switch* object

8.3.47 NULL

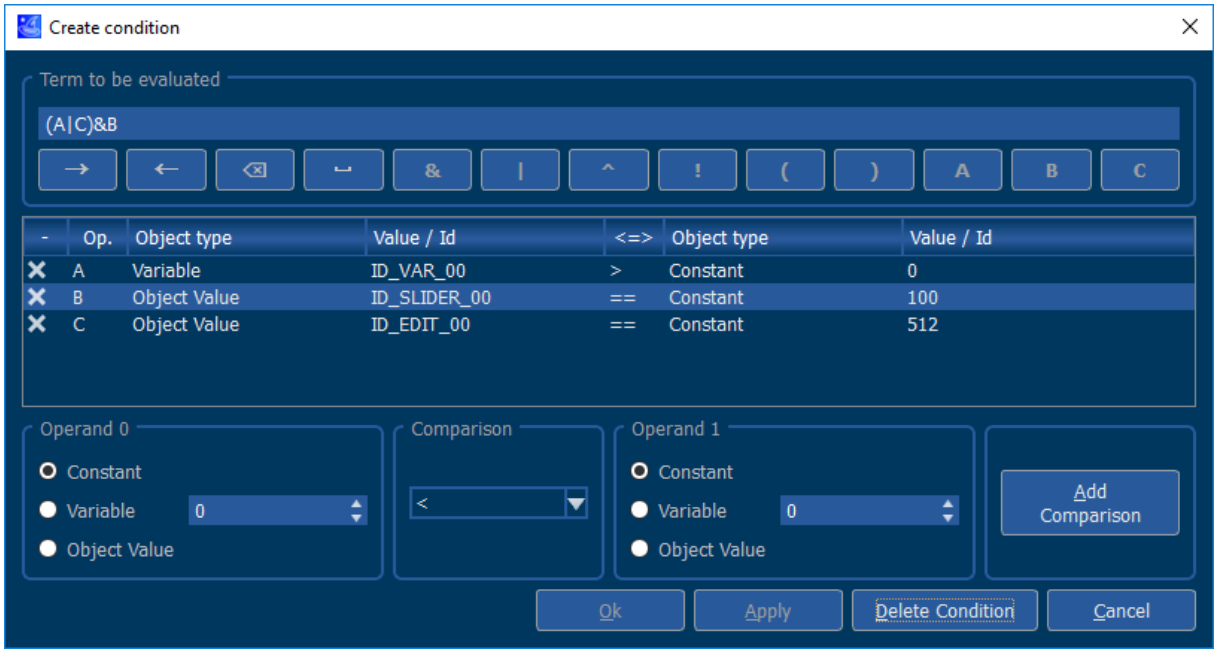
Description

Specifying a job to `NULL` gives the user the option to simply add custom code to the interaction and do nothing else.

8.4 Conditions

8.4.1 Introduction

A condition can be optionally added to an interaction. When a condition is added, the job of the interaction will only be executed, if the term of the condition is true. This allows the user to even add complex logic to the application.



Adding a condition

As hinted before, a condition can only be added to an existing interaction. A condition can be added by clicking the plus symbol in the condition column in the interaction window.

Editing or deleting a condition

A condition can be edited or removed from an interaction by clicking on the pen icon in the condition column in the interaction window.

8.4.2 Terms and operands

A term is made up from operands (such as A, B, C, …) and logical operators.

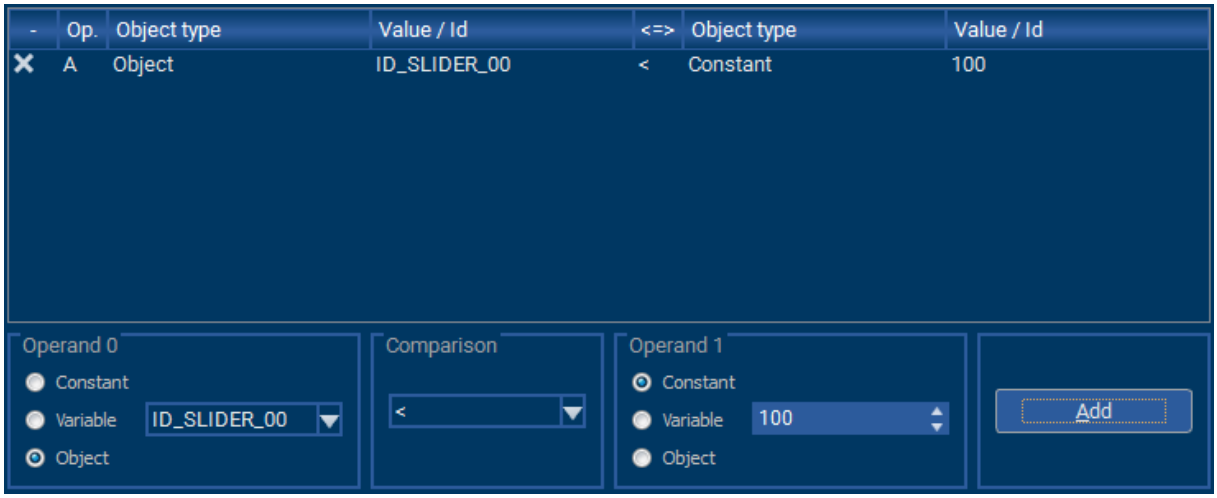
Comparisons and operands

Each operand is a validation of a comparison between two values. The values to be compared can be:

- **constants**,
- **variables** and
- **objects** (meaning objects that have a value, such as sliders, gauges, edits in decimal mode, …).

The operators for comparison are:

- **<** (less than)
- **<=** (less than or equal)
- **=** (equal)
- **>=** (greater than or equal)
- **>** (greater than)
- **!=** (not equal)



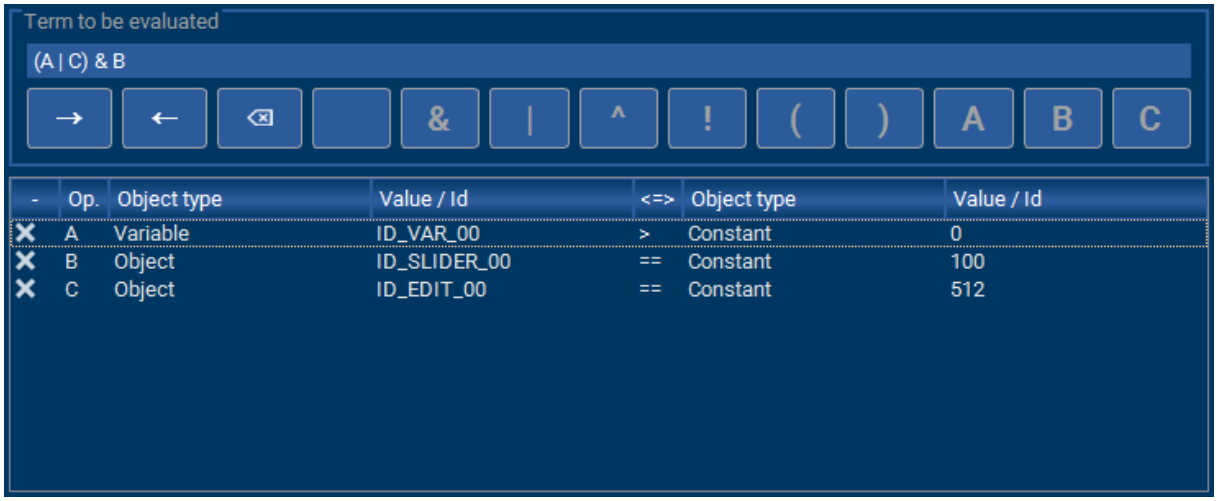
> **Note**
>
> For each comparison, an operand is added. The operand is named by a letter of the alphabet, starting with A. This means, the maximum number of operands to be added for a condition is limited to 26.

Term

When the operands have been added, a term can be set up. The term consists of the added operands and logical operators. the logical operators that can be used are:

- **&** (AND)
- **|** (OR)
- **^** (XOR)
- **!** (NOT)

Furthermore, brackets **(** and **)** can be used.



The interface allows the user to enter the term using the buttons or manually enter it via the keyboard. For each operand, an individual button is added.

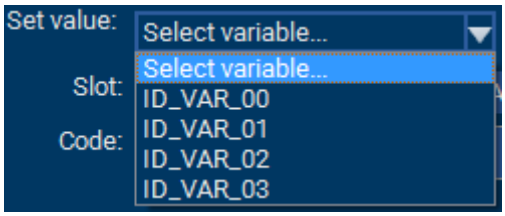
The four left-most buttons are used for moving the cursor, deleting characters and inserting spaces. Buttons with operators or operands that may not be inserted at the current position appear grayed out.

When using the keyboard to enter the term, any operators or operands not currently applicable to the term are ignored.

Chapter 9

Variables

The user can add variables to the project which can be processed by the application. Variables can also be manipulated from outside of the application.



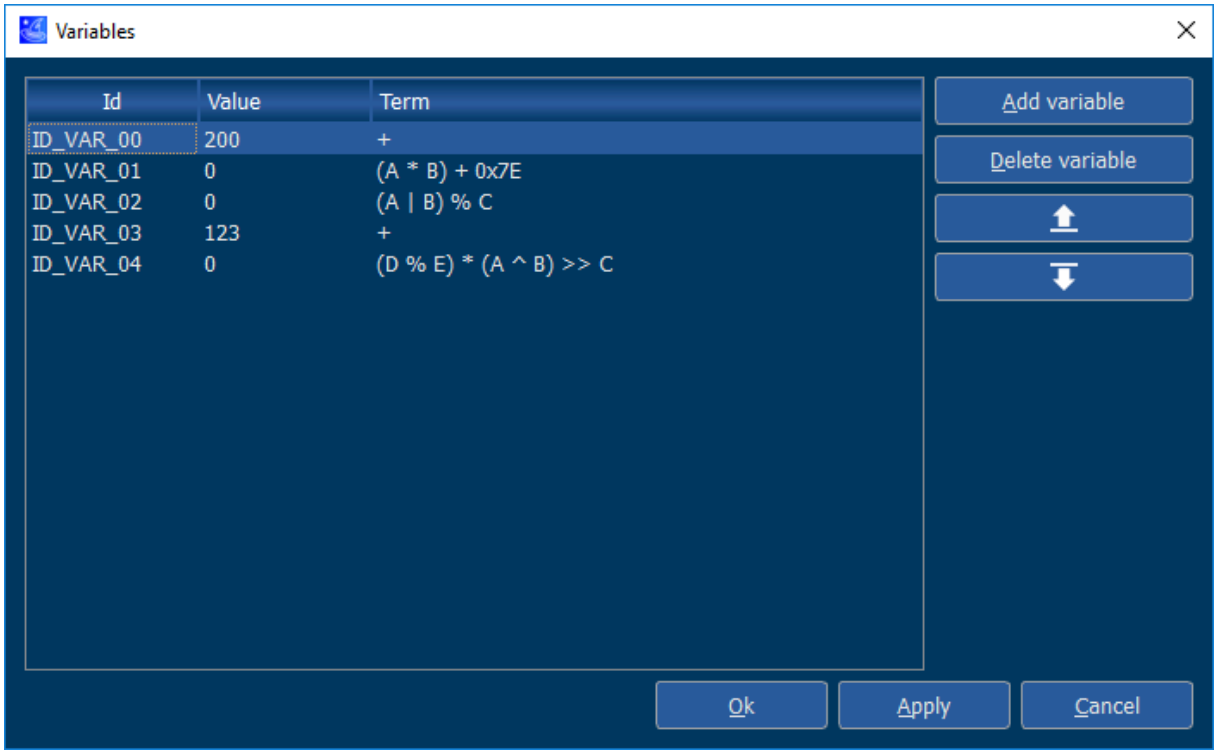
9.1 Variable management

The variable window allows the user to manage the variables for the current project. The management dialog can be opened by clicking the “Variables” button in the bottom left corner of AppWizard.

New variables can be added by pressing the **Add variable** button and they can be deleted by pressing the **Delete variable** button.

Using the buttons with the upwards and downwards arrows will move the selected variable either up or down, depending on the button.

After a variable has been created, it may be used for an interaction or can be manipulated from user code using `APPW_SetValue()`.



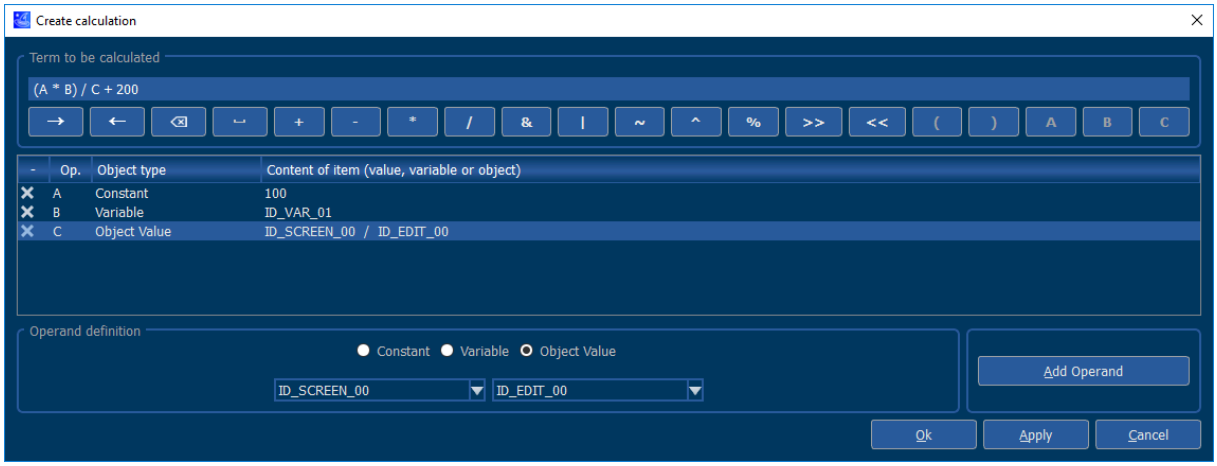
Initial value

The value in the “Value” column of a variable can be edited. This value will be assigned to the variable upon start of the application.

9.2 Calculations

Introduction

By adding a term to a variable, the value of that variable will be calculated using the given term. A term can be calculated from other variable values, object values or constant values. This allows for a much more detailed application logic inside AppWizard projects.



Adding a calculation

To add a term to a variable, click the "+" in the "Term" column of the variable management dialog.

9.2.1 Terms and operands

A term is made up from operands (such as A, B, C, …) and operators.

Operands

To create a term, operands have be added in the first place. Operands are values that can be derived from:

- **constants**,
- **variables** or
- **objects** (meaning objects that have a value, such as sliders, gauges, edits in decimal mode, …).

Operators

The following operators can be used for a calculation between the operands:

- **+** (add)
- **-** (subtract)
- ***** (multiply)
- **/** (divide)
- **%** (modulo)
- **&** (binary AND)
- **|** (binary OR)
- **^** (binary XOR)
- **~** (binary one's complement)
- **<<** (binary left-shift)
- **>>** (binary right-shift)

Creating a term

> **Note**
>
> Multiplication and division **do not** have a higher precedence level than addition and subtraction! All operations are calculated from left to right, except when brackets are used. Brackets are mandatory to indicate a higher level of precedence.
>
> **Example:**
> - 1 + 2 * 3 = 9
> - 1 + (2 * 3) = 7

When all necessary operands have been added, a calculation between the operators can be set up. Each operand is equal to a letter (such as A, B, C, …) and the operands are to be used within the term to be calculated.

> **Note**
>
> Numbers can also be used in the term. It is **not** mandatory to create an operand for a constant! Decimals are allowed, as well as hexadecimal numbers (prefixed by **`0x`**).

Calculating a variable

In order to calculate the new value of a variable with the term, the job `CALC` has to be executed with the desired variable as the receiver.

If this causes a change of the value, the variable will emit a `VALUE_CHANGED` signal.

9.3 Manipulating variables from user code

The routines `APPW_SetValue()` and `APPW_GetValue()` allow for reading and modifying variables from a project's user code. In combination with the signal **`VALUE_CHANGED`**, this feature can be utilized for various use cases.

Example

For example, in a weather forecast application, the temperature values can be stored in variables. When the user presses a button to refresh the temperature data, new data is polled and set to the variable using `APPW_SetValue()`.

By reacting on **`VALUE_CHANGED`**, the application would know when a temperature value has changed and if e.g. a different text or bitmap should be displayed.

Chapter 10

Animations

AppWizard allows the user to add complex animations to their project.

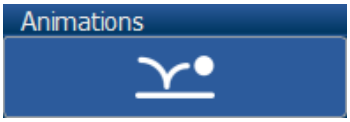
With AppWizard V1.20, animations have been completely reworked. This has been done because the previous process of defining complex animations was too complicated and not intuitive enough. Animations now support IDs which simplifies the use of animations within AppWizard and the ID makes them addressable.

With this animation rework, a couple of jobs and signals in AppWizard have been marked as obsolete. We recommend to not use these old mechanisms and to rework existing projects, eventually. Projects from previous versions with old animation interactions are still fully functional though.

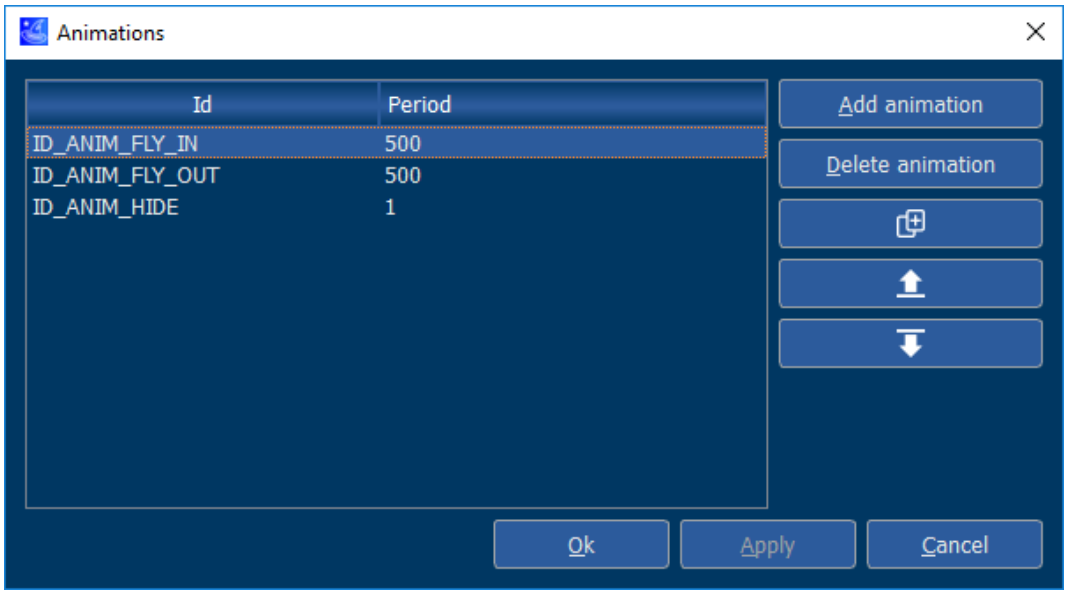
> **Note**
>
> More information about the basics of emWin animations (such as animation items, animation ease, etc.) can be read in the document **UM03001 emWin User Guide & Reference Manual**.

10.1 Pre-defining animation IDs

The first step to adding animations to an AppWizard project is opening the animation dialog by clicking the icon in the lower left corner of the tool.

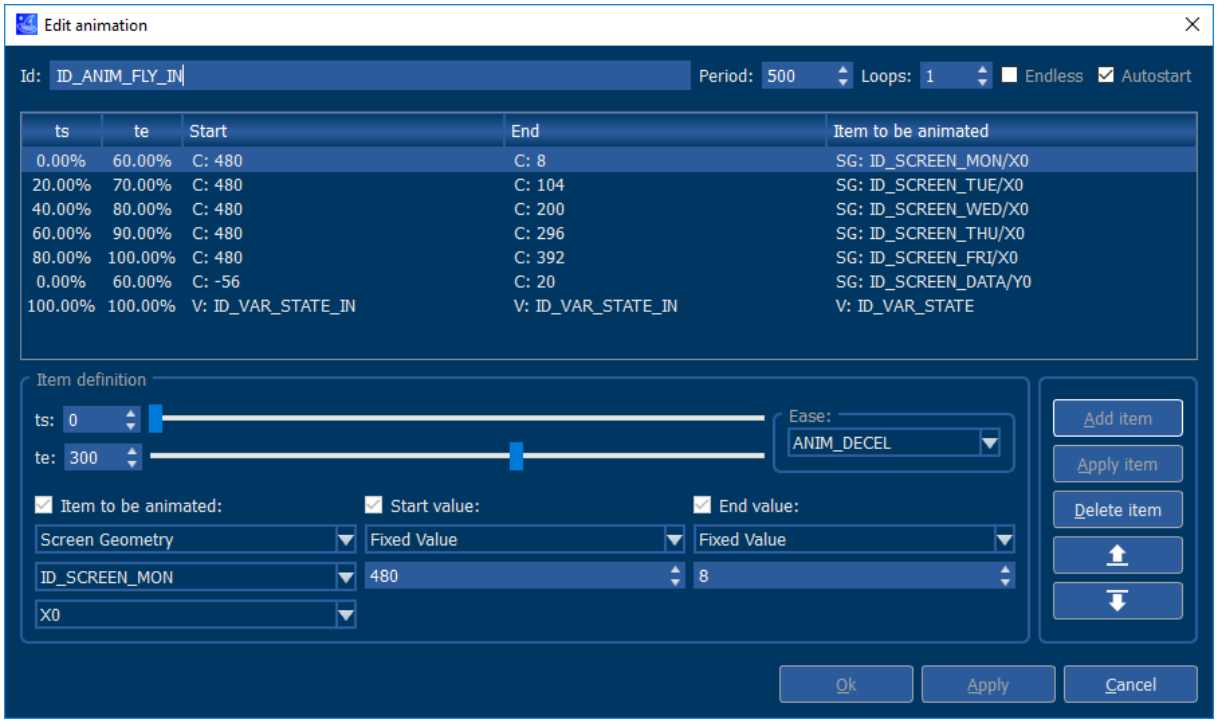


Now, the desired amount of animations can be added by clicking the "Add animation" button. The table shows the ID of the animation and it's entire duration in milliseconds.



10.2 Edit animations

To define and edit an animation, click its ID in the animation dialog. Then, the edit dialog will open.



10.2.1 Animation properties

- **Autostart**: Automatically starts the animation after executing the job ANIMCREATE.
- **Period**: Animation duration in ms. Takes effect only after creation with the autostart option.
- **Endless/Loops**: The animation runs endlessly until stopped or for a number of loops. Takes effect only after creation with the autostart option.

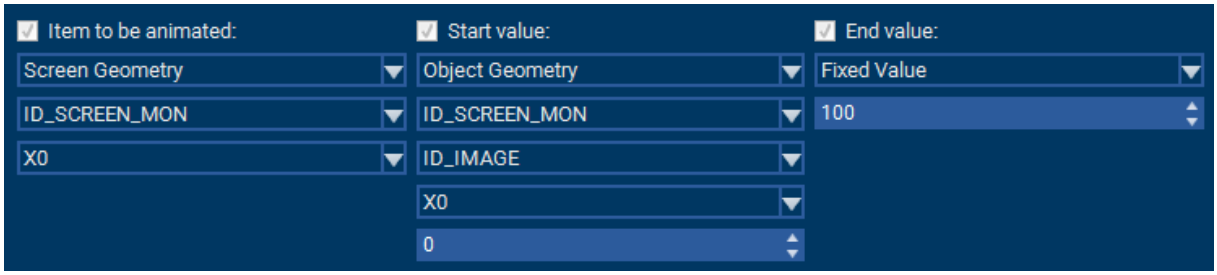
10.2.2 Start and end time of animation items

- **ts**: Start time of item in ms within the animation
- **te**: End time of item in ms within the animation

The item list shows the start and end time of each item in percent, relative to the duration of the entire animation.

ts	te
0.00%	60.00%
20.00%	70.00%
40.00%	80.00%
60.00%	90.00%
80.00%	100.00%
0.00%	60.00%
100.00%	100.00%

10.2.3 Animation values



An animation always “animates” a certain value, such as a window position.

For an animation item, three values need to be specified:

- **Start value**: Initial value of the item to be animated.
- **End value**: Final value of the item, when the animation has ended.
- **Item to be animated**: Item, that the animated value should be applied to.

The following types of values can be used for animation items:

Type	Abbreviated	Description
Object value	OV	Value of an object, such as a slider.
Variable	V	Value of an AppWizard variable.
Object Geometry	OG	Coordinate or size of an object.
Screen Geometry	SG	Coordinate or size of a screen.
Fixed Value	C	A constant value.

Object value, object geometry and screen geometry allows an additional offset in the lowermost field.

Animating object coordinates

Note that when animating object coordinates the animation has to match the anchor point of the object. For example, when an object has the anchor point in the top left corner (x0, y0), animating the coordinate x1 will not have an effect.

ANIMSTART and ANIMEND signals

When object coordinates are animated, they emit ANIMSTART and ANIMEND signals when the associated animation item starts or ends.

10.2.4 Animation ease

The ease function of an animation defines how the animated value will change over time and thus how the animation will look like. More information about this can be read in the emWin manual in the chapter *Animations*.

Ease	Description
`ANIM_LINEAR`	Animated is performed linear.
`ANIM_ACCEL`	Animation is accelerating.
`ANIM_DECEL`	Animation is decelerating.
`ANIM_ACCELDECEL`	Animation is accelerating, then decelerating.

10.3 Running animations

Starting an animation

Once an animation has been defined, it first has to be created using the interaction job ANIMCREATE.

Then, the animation can be started using the job ANIMSTART.

> **Note**
>
> Animations are only deleted automatically after they have run completely. If you want to stop and delete them by interaction, you have to activate the remove option in ANIMSTOP.

Stopping an animation

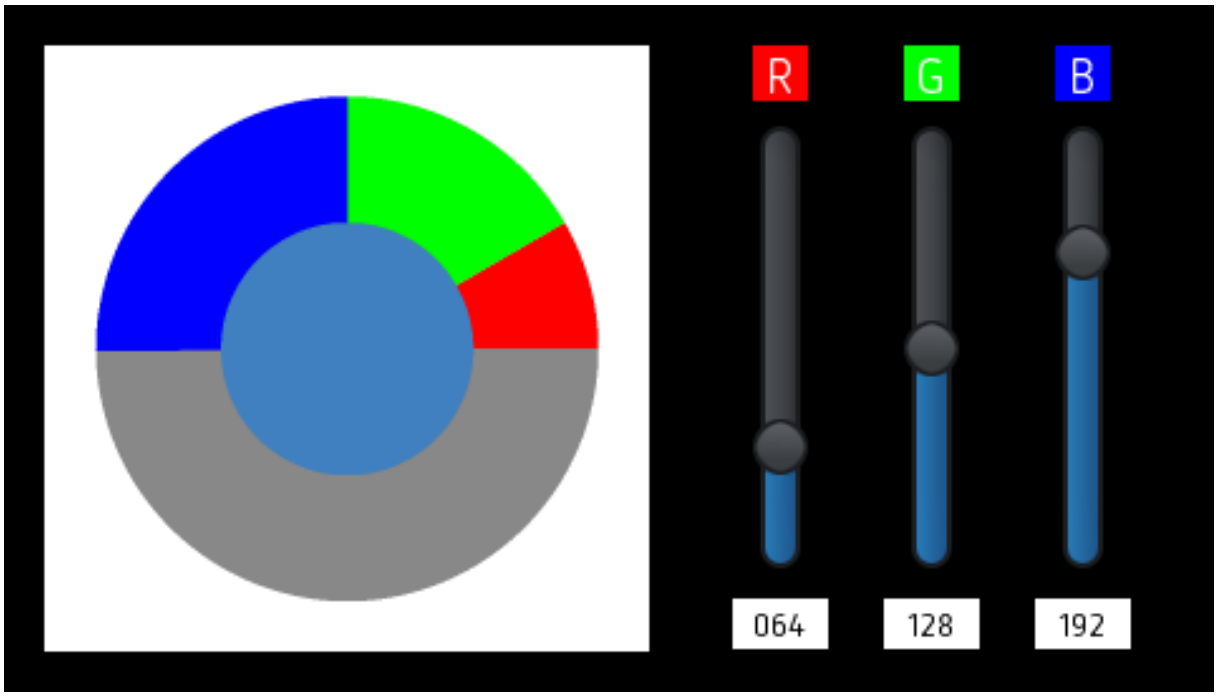
To stop an animation, the job ANIMSTOP can be used.

Optionally, the job can also remove an animation. To use it again, it would then have to be created again using ANIMCREATE.

Chapter 11

Drawings

AppWizard allows the user to add custom drawings to their application. The feature supports a wide array of emWin's drawing routines.



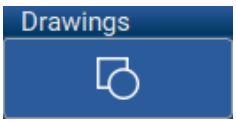
Note

This chapter will only provide information on how to use drawing routines within AppWizard. To learn more detail about a specific routine, please refer to the document **UM03001 emWin User Guide & Reference Manual**.

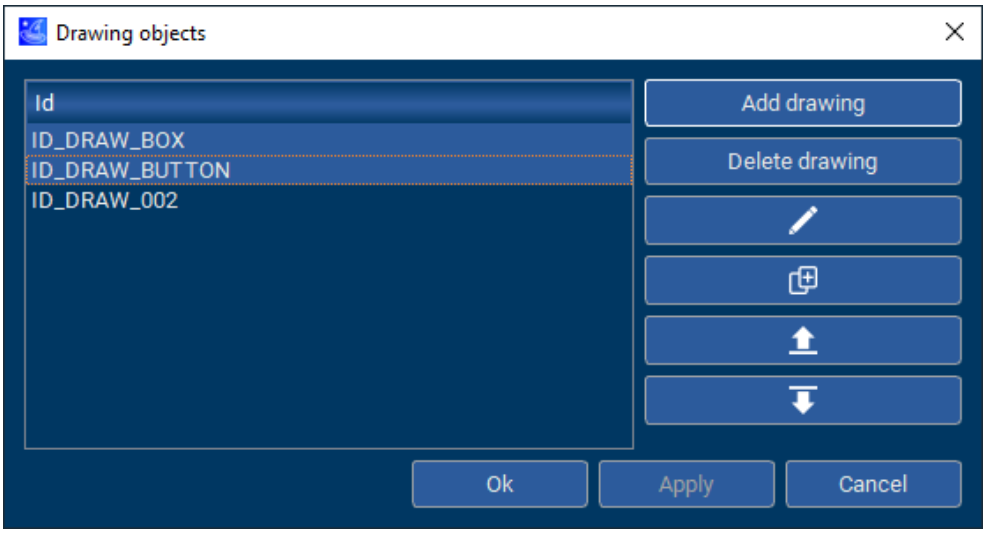
11.1 Creating a drawing object

Creating a drawing object

Before a drawing can be displayed, the first step is to create a drawing object. To do this, open the drawings dialog by clicking the quick access button in the lower left corner of AppWizard.



The dialog allows adding, editing and deleting drawing objects.

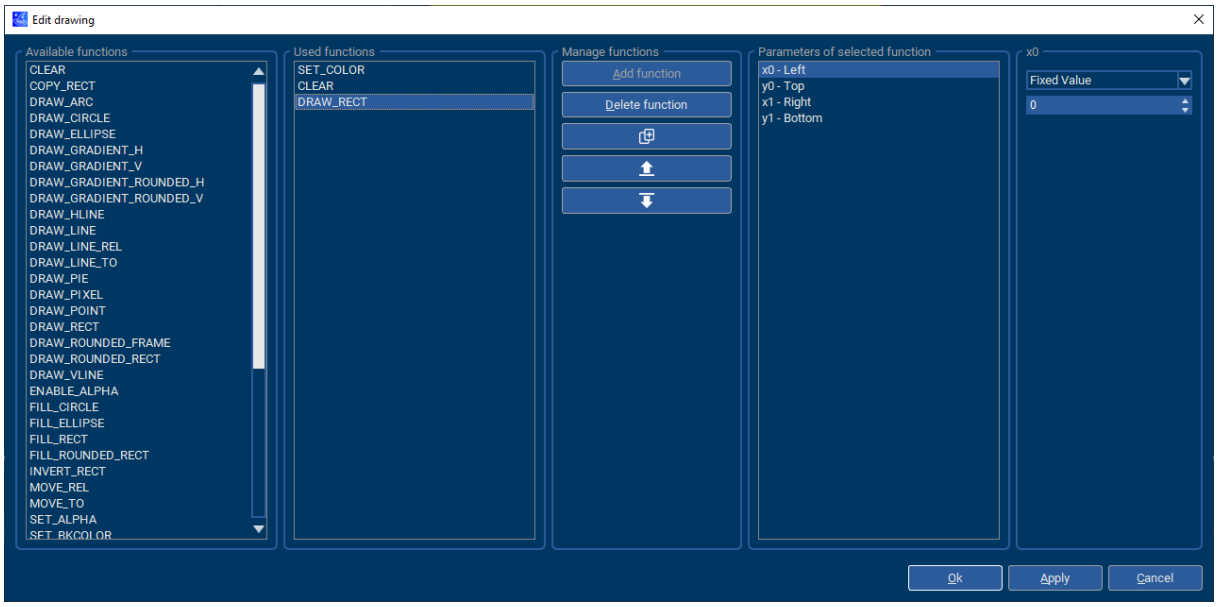


After an object has been created, it can be edited by clicking the ’Edit’ icon.

11.2 Defining a drawing

Editing a drawing

To edit a drawing, select a drawing object in the above mentioned dialog and click the ’Edit’ icon. A dialog will open, allowing to define what the drawing object should do.

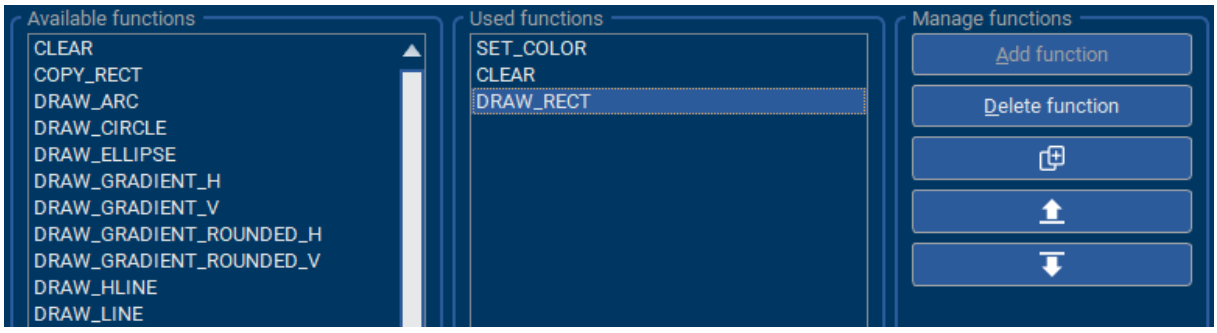


Adding functions

> **Note**
>
> A detailed list of all available functions and parameters can be found under Available drawing functions.

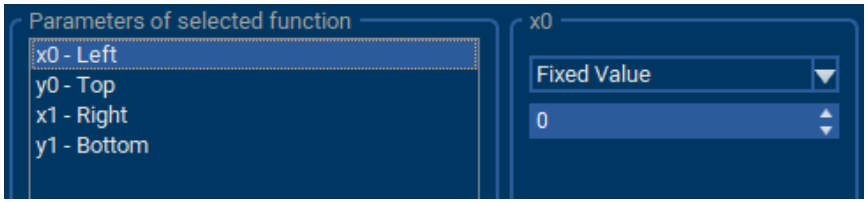
The left side of the dialog allows selecting the drawing operations that should be performed and adding them to the object. To do this, select a function under ’Available functions’ and add it using the ’Add functions’ button.

The function will then appear in the ’Used functions’ section. This shows all the drawing operations that will be performed chronologically.



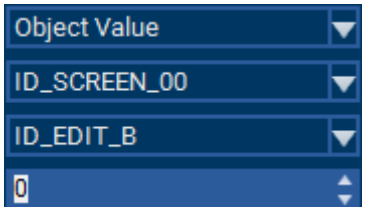
Setting the parameters

Some functions require parameters, such as coordinates for `DRAW_RECT`. To set the parameters of a function that has been added to a drawing object, select it under 'Used functions' and its corresponding parameters will be shown under 'Parameters of selected function'.



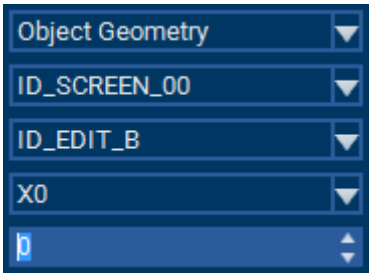
A parameter value can either be one of the following:

- an **object value**

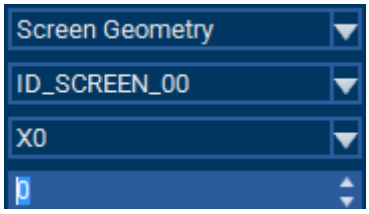


- a **variable**

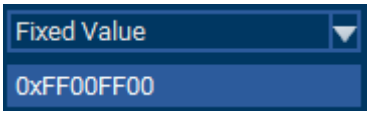
- **object geometry** (such as x0, y0, x1, y1, xSize, ySize)



- **screen geometry** (such as x0, y0, x1, y1, xSize, ySize)



- a **fixed value**



Object value, object geometry and screen geometry allows an additional offset in the lowermost field.

11.3 Displaying a drawing

A drawing object can be displayed within any given widget of the application. It can either be drawn **before** or **after** `WM_PAINT` is executed for a given widget.

When a widget is selected, its **Predraw** and its **Postdraw** property can be set to display a given drawing object.

11.4 Available drawing functions

The following table lists all available drawing functions. The parameters of the functions are largely identical to the parameters of the analogously existing emWin functions described in the emWin manual. In order to avoid redundancy, we have therefore dispensed with a detailed documentation of the drawing functions at this point.

Function	Description
CLEAR	Clears the area using the background color.
COPY_RECT	Copies a rectangular area to a new position.
DRAW_ARC	Draws an arc.
DRAW_CIRCLE	Draws a circle.
DRAW_ELLIPSE	Draws an ellipse.
DRAW_GRADIENT_H	Draws a horizontal gradient.
DRAW_GRADIENT_V	Draws a vertical gradient.
DRAW_GRADIENT_ROUNDED_H	Draws a horizontal gradient with rounded corners.
DRAW_GRADIENT_ROUNDED_V	Draws a vertical gradient with rounded corners.
DRAW_HLINE	Draws a horizontal line.
DRAW_LINE	Draws a line.
DRAW_LINE_REL	Draws a relative line from the current position.
DRAW_LINE_TO	Draws a line from the current to the new position.
DRAW_PIE	Draws a circle section.
DRAW_PIXEL	Sets a single pixel.
DRAW_POINT	Draws a point.
DRAW_RECT	Draws a rectangle.
DRAW_ROUNDED_FRAME	Draws a rounded rectangle in the given rectangle.
DRAW_ROUNDED_RECT	Draws a rounded rectangle.
DRAW_VLINE	Draws a vertical line.
ENABLE_ALPHA	Enables or disables alpha drawing mode.
FILL_CIRCLE	Fills a circle.
FILL_ELLIPSE	Fills an ellipse.
FILL_RECT	Fills a rectangle.
FILL_ROUNDED_RECT	Fills a rectangle with rounded corners.
INVERT_RECT	Inverts the color of a rectangle area.
MOVE_REL	Relative move to a new position.
MOVE_TO	Move to screen coordinates.
SET_BKCOLOR	Sets the background color.
SET_COLOR	Sets the foreground color.
SET_PENSIZE	Sets the pen size.
SET_LINESTYLE	Sets the linestyle.
AA_DISABLE_HIRES	Disables high-resolution anti-aliasing.
AA_DRAW_ARC	Draws an anti-aliased arc.
AA_DRAW_CIRCLE	Draws an anti-aliased circle.
AA_DRAW_LINE	Draws an anti-aliased line.
AA_DRAW_PIE	Draws an anti-aliased pie.

Function	Description
AA_DRAW_ROUNDED_FRAME	Draws an anti-aliased rounded rectangle in the given rectangle.
AA_ENABLE_HIRES	Enables high-resolution anti-aliasing.
AA_FILL_CIRCLE	Fills an anti-aliased circle.
AA_FILL_ELLIPSE	Fills an anti-aliased ellipse.
AA_FILL_ROUNDED_RECT	Fills an anti-aliased rounded rectangle.
AA_SET_FACTOR	Sets the anti-aliasing factor.

Chapter 12

User Code

The following chapter explains how the user may add custom code to their AppWizard application. It will also be explained how variables and fonts created within AppWizard may be utilized for custom code and how slot routines can be used.

12.1 Slot routines

Slot routines are the routines that are executed with the job of an interaction.

Where to find a slot routine

The name of a slot routine can be accessed and changed in the 'Set interaction parameters' dialog. This routine is located in the file `<ScreenID>_Slots.c` in the directory `\Custom-Code\Config\`.

Prototype

```
void <ScrID>__<EmitID>__<SignID>__<RecvID>__<JobID>(APPW_ACTION_ITEM * pAction,
                                                    WM_HWIN             hScreen,
                                                    WM_MESSAGE        * pMsg,
                                                    int               * pResult);
```

ScrID Id of the screen where the objects are on.
EmitID Id of the emitting object.
SignID Id of the signal.
RecvID Id of the receiving object.
JobID Id of the job.

Parameters

Parameter	Description
`pAction`	Pointer to an `APPW_ACTION_ITEM` structure.
`hScreen`	Handle of the screen.
`pMsg`	Pointer to a `WM_MESSAGE` structure. `pMsg->hWin` is the handle to the receiver while `pMsg->hWinSrc` is the handle to the emitter.
`pResult`	Pointer to an int containing the 'result' value. This value is explained below.

Additional information

Each interaction has job-specific parameters. The parameters can be accessed via the `aPara` element of the `APPW_ACTION_ITEM` structure which is passed to a slot routine.

The parameter of each interaction is explained under **Job-specific parameters passed to slot-routine** for each job under *List of jobs* on page 155.

The parameter `pResult` points to an integer which by default is 0. If `*pResult = 0`, the interaction will be executed by the AppWizard. If `*pResult = 1`, only the custom code is executed.

12.1.1 APPW_ACTION_ITEM

Description

This structure is passed to an interaction slot routine.

Type definition

```
typedef struct {
  int              IdSrc;
  int              NCode;
  int              IdDst;
  int              IdJob;
  void          (* pfSlot)(APPW_ACTION_ITEM * pAction,
                           WM_HWIN            hScreen,
                           WM_MESSAGE       * pMsg,
                           int              * pResult);
  APPW_PARA_ITEM   aPara[6];
} APPW_ACTION_ITEM;
```

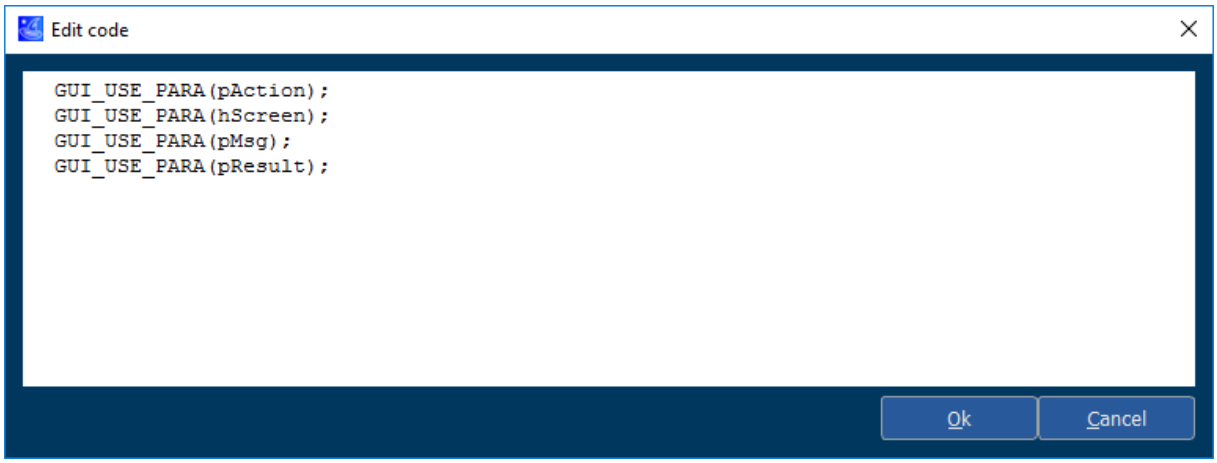
Structure members

Member	Description
IdSrc	Id of the emitter.
NCode	Id of the signal.
IdDst	Id of the receiver.
IdJob	Id of the job.
pfSlot	Function pointer to a slot routine. Prototype explained under Slot routines.
aPara	Optional job specific parameters. The parameter for each job is explained in the *List of jobs* on page 155.

12.1.2 Custom user code

Slot routines

As mentioned earlier, the user may add their own code to slot routines, either via the “Edit code” dialog in the interaction dialog, or even from any editor or IDE.



Any user code within the generated slot routines stays persistent when for example exporting the AppWizard project another time.

Custom routines

If the user wants to add their own custom routines to the application, they should create a new C file and add it to their project.

AppWizard also adds the automatically generated files `Application.c` and `Application.h` to the simulation project. These files are intended to be used for user code.

Custom routines can also be added to added to a generated slot file, however they **must** be within the user code section, otherwise they will be overwritten once the project is exported again.

```
/*** Begin of user code area ***/

static void _FooBar(void) {}

/*** End of user code area ***/
```


12.2 Screen callback routines

Every screen object has its own generated callback routine. This callback will be called additionally, this means it isn't a requirement and may be left empty.

Where to find a screen callback

The callback is named after the format `cb<ScreenID>`, e.g. `cbID_SCREEN_00`. A screen callback routine can be found in the slot routine file, located in the project directory under `\Source\CustomCode`.

How to use them

Generally, a screen callback is very similar to an emWin window callback. This means, the callback may react on all types of window messages. To learn more about the different types of window messages, refer to the document **UM03001 emWin User Guide & Reference Manual**.

> **Note**
>
> However, a screen callback **must not** have a `default` case that calls `WM_DefaultProc()`, as a normal window callback would do.

Example

When reacting on the `WM_INIT_DIALOG` case, custom windows or widgets can be added to the application upon creation of the screen object. When creating a window/widget as a child to the screen, `WM_NOTIFY_PARENT` messages obviously get sent to the parent callback.

```
/*********************************************************************
*
*       cbID_SCREEN_00
*/
void cbID_SCREEN_00(WM_MESSAGE * pMsg) {
  WM_HWIN hWin;
  int     Id, NCode;

  switch (pMsg->MsgId) {
  case WM_INIT_DIALOG:
    hWin = LISTVIEW_CreateEx(10, 10, 300, 200,
    pMsg->hWin, WM_CF_SHOW, 0, GUI_ID_LISTVIEW0);
    break;
  case WM_NOTIFY_PARENT:
    Id    = WM_GetId(pMsg->hWinSrc);
    NCode = pMsg->Data.v;
    switch(Id) {
    case GUI_ID_LISTVIEW0:
      switch(NCode) {
      case WM_NOTIFICATION_CLICKED:
        break;
      case WM_NOTIFICATION_RELEASED:
        break;
      case WM_NOTIFICATION_MOVED_OUT:
        break;
      case WM_NOTIFICATION_SCROLL_CHANGED:
        break;
      case WM_NOTIFICATION_SEL_CHANGED:
        break;
      }
      break;
    }
    break;
  }
}
```


12.3 General AppWizard API

Routine	Description
APPW_GetText()	This function stores the text of an object in the given buffer.
APPW_SetCustCallback()	Sets a function pointer for a function which is executed at the end of APPW_Exec().
APPW_GetValue()	This function returns the value of an object.
APPW_SetText()	This function sets a text to an object.
APPW_SetValue()	This function sets the value of an object.

12.3.1 APPW_GetText()

Description

This function stores the text of an object in the given buffer.

Prototype

```
int APPW_GetText(U16    IdScreen,
                 U16    IdWidget,
                 char * pBuffer,
                 U32    SizeOfBuffer);
```

Parameters

Parameter	Description
IdScreen	ID of the parent screen the object belongs to.
IdWidget	ID of the object the text should be retrieved from.
pBuffer	Pointer to a buffer the text gets stored in.
SizeOfBuffer	The size of the buffer pBuffer points to.

Return value

If the return value is 1 no handle to the object was found.

Additional information

This fuction can be used for all objects which can have a text.

Available objects

This function can be used for the following objects:

- *Button* object
- *Edit* object
- *Text* object
- *QRCode* object

12.3.2 APPW_GetValue()

Description

This function returns the value of an object.

Prototype

```
int APPW_GetValue(U16   IdScreen,
                  U16   IdWidget,
                  int * pError);
```

Parameters

Parameter	Description
IdScreen	ID of the parent screen the object belongs to.
IdWidget	ID of the object the value should be retrieved from.
pError	Out pointer being used to indicate that something went wrong.

Return value

The current value of the given object.

Additional information

This fuction can be used for all objects which can have a value. If pError is 1 no handle to the object could be found.

Available objects

This function can be used for the following objects:

- *Button* object
- *Edit* object
- *Gauge* object
- *Progbar* object
- *Slider* object
- *Switch* object

12.3.3 APPW_SetCustCallback()

Description

Sets a function pointer for a function which is executed at the end of APPW_Exec().

Prototype

```
void APPW_SetCustCallback(void ( *pFunc)());
```

Parameters

Parameter	Description
pFunc	Pointer to the function which should be called.

Additional information

This function allows the user to set a function pointer which is being called from APPW_Exec(). This allows the user to execute his own code periodically.

> **Note**
>
> It is possible to set further callback and hook functions. Please refer to chapter 'Setting hook functions' in the emWin user manual UM03001_emWin.pdf

12.3.4 APPW_SetText()

Description

This function sets a text to an object.

Prototype

```
int APPW_SetText(U16    IdScreen,
                 U16    IdWidget,
                 char * pText);
```

Parameters

Parameter	Description
IdScreen	ID of the parent screen the object belongs to.
IdWidget	ID of the object the text should be set to.
pText	Pointer to the text which should be set.

Return value

If the return value is 1 no handle to the object was found.

Additional information

This fuction can be used for all objects which can have a text.

Available objects

This function can be used for the following objects:

- *Button* object
- *Text* object
- *Edit* object
- *QRCode* object

12.3.5 APPW_SetValue()

Description

This function sets the value of an object.

Prototype

```
int APPW_SetValue(U16 IdScreen,
                  U16 IdWidget,
                  int Value);
```

Parameters

Parameter	Description
IdScreen	ID of the parent screen the object belongs to.
IdWidget	ID of the object the value should be retrieved from.
Value	The value to be set to the object.

Return value

If the return value is 1 no handle to the object was found.

Additional information

This fuction can be used for all objects which can have a value.

Available objects

This function can be used for the following objects:

- *Button* object
- *Edit* object
- *Gauge* object
- *Progbar* object
- *Slider* object
- *Switch* object

12.4 Fonts

This chapter explains how fonts created within AppWizard can be used in custom user code.

> **Note**
>
> The chapter *Font management* on page 57 explains how fonts can be created using AppWizard.

12.4.1 How to use fonts

As already explained earlier in this manual, fonts can be easily created with AppWizard and used as often as the user wants to within a project. The following section will demonstrate, how these fonts can be accessed within custom C code.

Requirements

In order to be able to use a font in custom C code, it must have been created within the project. The font also has to have been referenced by an object on a screen, this means the “Set font” property for an object must be set with the desired font.

How to use a font

The following example will demonstrate, how a font can be used in user code.

The font has to be created using `APPW_GetFont()`. The ID of the object that references the font has to be stated as second parameter, the ID of the screen the object is on as first parameter.

The function will then fill a `GUI_FONT` and `GUI_XBF_DATA` structure. The variables that hold the font data should be located in ROM, so the font data stays persistent.

```
/*********************************************************************
*
*       APP_cbWin
*/
void APP_cbWin(WM_MESSAGE * pMsg) {
  static GUI_FONT     Font;
  static GUI_XBF_DATA FontData;

  switch (pMsg->MsgId) {
  case WM_CREATE:
    APPW_GetFont(ID_SCREEN_00, ID_TEXT_00, &Font, &FontData);
    break;
  case WM_PAINT:
    GUI_SetFont(&Font);
    GUI_SetTextMode(GUI_TM_TRANS);
    GUI_DispStringAt("Test", 0, 0);
    break;
  default:
    WM_DefaultProc(pMsg);
  }
}
```

With the callback above, a window can be created. Custom window or widget callbacks should be located in the `Application.c` file and can then be used in a slot routine.

```
/*********************************************************************
*
*       cbID_SCREEN_00
*/
void cbID_SCREEN_00(WM_MESSAGE * pMsg) {
  WM_HWIN hWin;
```



```
  switch (pMsg->MsgId) {
  case WM_INIT_DIALOG:
    hWin = WM_CreateWindowAsChild(10, 10, 100, 32, pMsg->hWin,
    WM_CF_SHOW | WM_CF_HASTRANS, APP_cbWin, 0);
    break;
  }
}
```

> **Note**
>
> To learn more about slot routines and custom user code, refer to *Slot routines* on page 223.

12.4.2 Font API

The following table provides an overview of the routines related to fonts.

Routine	Description
APPW_GetFont()	Fills a font structure using the addressed setup structure.

12.4.2.1 APPW_GetFont()

Description

Fills a font structure using the addressed setup structure.

Prototype

```
int APPW_GetFont(U16            IdScreen,
                 U16            IdWidget,
                 GUI_FONT     * pFont,
                 GUI_XBF_DATA * pData);
```

Parameters

Parameter	Description
IdScreen	ID of the screen.
IdWidget	ID of the widget.
pFont	GUI_FONT structure to be filled.
pData	Pointer to a GUI_XBF_DATA structure

Return value

0 Function has succeeded.
1 Function has failed.

Example

See *How to use fonts* on page 233 for an example.

12.5 Variables

12.5.1 How to use variables

Variables in the AppWizard can be used to store a value. They can be accessed and changed by the application or from outside of the application. The application can react on a change of a variable using interactions.

Creating variables

The user can manage (add and delete) their variables via the variable resource window. This window can be accessed by clicking the lower right quick access button, located in the lower left corner of the AppWizard.

Using variables for interactions

The main purpose for variables is to use them within an interaction, whether as an emitter or as a receiver.

If the variable is an emitter of an interaction, the signal to be reacted on can be a change of that variable. If the variable is instead the receiver of a signal, the job can be to change the value of the variable.

Reading and setting variables from outside of the application

Variables created with the AppWizard can be read from outside of the application via the method `APPW_GetVarData()` and set from outside of the application via the method `APPW_SetVarData()`.

12.5.2 Variables API

The following table provides an overview of the routines related to variables.

Routine	Description
`APPW_GetVarData()`	Returns the value of a variable.
`APPW_SetVarData()`	Sets the value of a variable.

12.5.2.1 APPW_GetVarData()

Description

Returns the value of a variable.

Prototype

```
I32 APPW_GetVarData(U16   Id,
                    int * pError);
```

Parameters

Parameter	Description
Id	ID of the variable.
pError	Pointer to integer used to return error on demand.

Return value

Data value of the specified variable.

12.5.2.2 APPW_SetVarData()

Description

Sets the value of a variable.

Prototype

```
int APPW_SetVarData(U16 Id,
                    I32 Data);
```

Parameters

Parameter	Description
Id	ID of the variable.
Data	Data value to set.

Return value

0 Function has succeeded.
1 Function has failed.

Chapter 13

Board support packages (BSPs)

As already mentioned in the chapter *Requirements* on page 19 the AppWizard can be used with any ANSI C compiler without any additional software library. To make things easy it comes with a couple of preconfigured BSPs. The following chapter explains in detail which software components need to be included in a BSP, how to create custom BSPs and how to import a BSP into the repository of the AppWizard.

13.1 Preconfigured BSPs included in the shipment

The AppWizard comes with some ready to use preconfigured BSPs to be used with SEGGER Embedded Studio, but also with other IDEs. They contain the following:

- A ready-to-use display configuration
- A ready-to-use touch screen configuration (if a touch screen exists)
- A ready-to-use file system configuration (if SD card is accessible)
- A binary version of embOS
- A binary version of emFile (if SD card is accessible)

Why does a BSP include embOS?

embOS is only used to get the CPU initialized and to give emWin a time base. An operating system is basically not a requirement for AppWizard. But from emWin's standpoint it makes more sense to use the embOS code, instead of rewriting it.

The same applies to the time base for emWin. Instead of using embOS a time base can be achieved with a simple timer interrupt routine.

Why does a BSP include emFile?

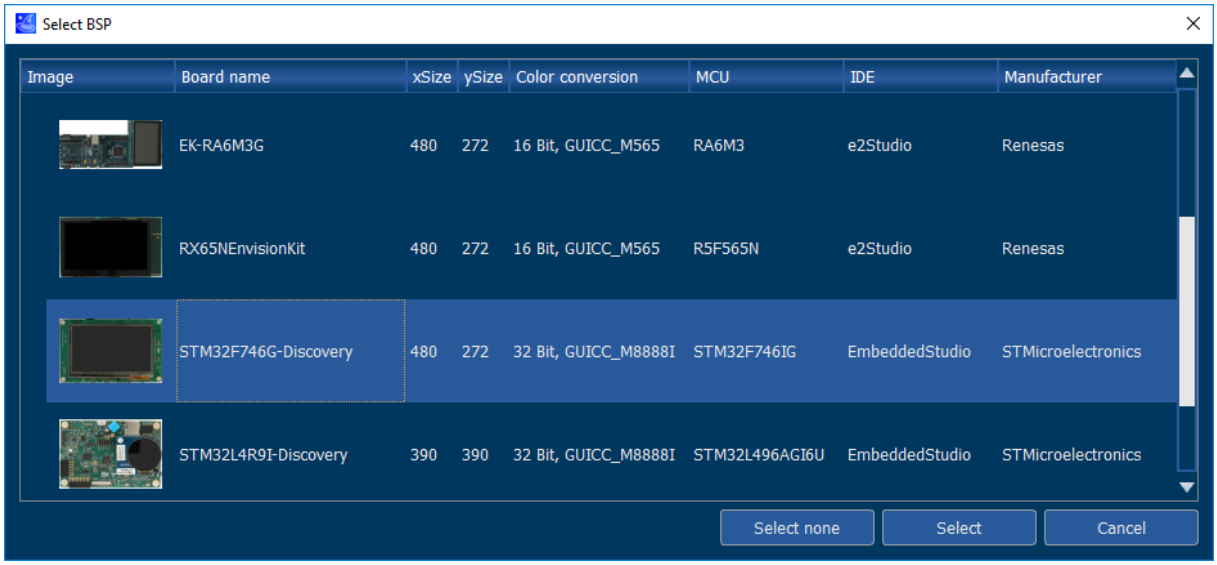
A file system is only required if resources should be outsourced to external media, for which we use the emFile system.

13.1.1 Example

The following example will explain how to open and run a project for a supported BSP on target hardware with SEGGER Embedded Studio.

13.1.1.1 Step 1: Select BSP

Select **Project ➔ Edit Options** and click **Select BSP**. Choose any BSP for Embedded Studio and click **Select**. Confirm the selection with **Ok**.



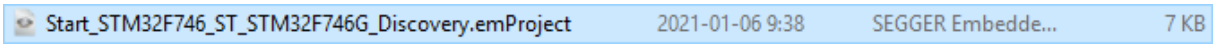
After the BSP has been selected, a new folder in the project directory named `Target` will be created. This folder contains the complete BSP.

13.1.1.2 Step 2: Generate code

Choose **File ➔ Export & Save**. By doing this the project file will be saved and the code will be exported to the sub folder `Source`.

13.1.1.3 Step 3: Run SEGGER Embedded Studio Project

The BSP contains a project file for SEGGER Embedded Studio. This project file has the suffix `.emProject` and can be found in the sub folder of the selected board under `\Target\BSP`.



Open the `.emProject` file with SEGGER Embedded Studio.

13.1.1.4 Step 4: Compile and run on target

The SEGGER Embedded Studio projects include all of the AppWizard code automatically, meaning normally no files need to be added or changed. Simply compile the project by pressing `F7` and run the project by pressing `F5`.

13.2 Creating custom BSPs

The following example shows how to create an AppWizard BSP. To be able to create a BSP for the AppWizard, we should already have an existing project with the following components:

- A ready-to-use display driver configuration
- A ready-to-use touch input configuration
- A ready-to-use time base for emWin
- A ready-to-use hardware initialization

For the sake of simplicity, we will use an already existing evaluation project for SEGGER Embedded Studio which is available on www.segger.com. This evaluation project is intended to be used for ST's STM32F429I-Discovery board.

> **Note**
>
> Although the example uses SEGGER Embedded Studio for demonstration, the following steps may also be applied to other IDEs.

13.2.1 Example

The following steps show how to create a reusable BSP for AppWizard based on that project.

13.2.1.1 Step 1: Create a project with AppWizard

After taking a look to the display configuration file of the above mentioned project, we know the display size and color conversion. Select **File ➔ New project** and enter the following data:

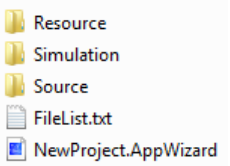
Property	STM32F429I-Discovery	Description
xSize	`240`	Target display x-size.
ySize	`320`	Target display y-size.
Color conversion	`GUICC_M8888I`	Desired color conversion.
Name	`STM32F429I-Disco`	Desired BSP name.
BSP	`None`	Must be left empty.
Enable Multibuffering	`Yes`	If the target supports multi-buffering.

13.2.1.2 Step 2: Create some elements

Fill the project with some elements such as Screen, Box and Button to make sure that there is something visible on the screen.

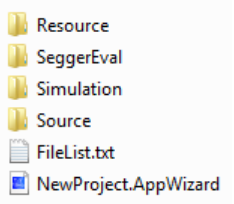
13.2.1.3 Step 3: Export & Save

Choose **File ➔ Export & Save**. After that we should find the following directory structure in the project directory:



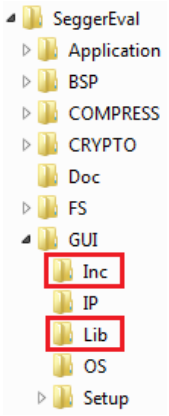
13.2.1.4 Step 4: Copy evaluation software package into project folder

By default a BSP is located in a directory named `Target` parallel to the directories `Resource`, `Simulation` and `Source`. For now, extract the evaluation software into the folder `SeggerEval`. The folder name does not matter during this step, but it must not be `Target`.



13.2.1.5 Step 5: Exchange libraries

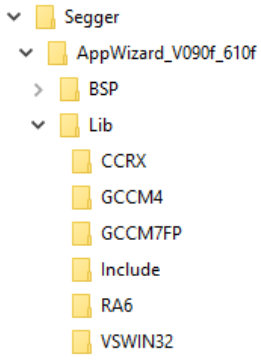
The emWin libraries of the evaluation software are located in the directories `\GUI\Lib` and `\GUI\Inc`:



> **Note**
>
> The emWin library of the evaluation package is divided into 2 directories, `\GUI\Lib` and `\GUI\Inc`. To be able to work with AppWizard, **all** files of the library have to be located in a single directory.

Delete the libraries present in the folder `\GUI\Lib` and remove the folder `\GUI\Inc`.

AppWizard comes with a couple of precompiled libraries which can be found in the program data directory of the AppWizard, as shown below. The program data directory is `C:\ProgramData\Segger\AppWizard_Vxxx_xxx`, depending on your AppWizard version.



Since we are using SEGGER Embedded Studio in this example and the MCU is an ARM Cortex-M4 device, we can use the library located in the sub-folder `GCCM4`. Copy the complete content of the `GCCM4` and the `Include` directory into the folder `\GUI\Lib` of the evaluation project.

If there is no precompiled library available, which is not very unlikely when working with your own hardware, you should create your own library. The emWin-documentation contains a detailed description how that can be achieved.

> **Note**
>
> The version number of emWin to be used to create the library must not be outdated and at least ≥ the emWin version number of the AppWizard. Otherwise the AppWizard assumes that the library of a project using the BSP which we are creating here, needs to be updated each time we open it.

With any normal project, the step of exchanging the GUI libraries would be finished at this point. But for this example, we are using an evaluation project containing multiple SEGGER products. Because of that, the files `Global.h`, `SEGGER.h` and `IP_FS.h` located in the `\GUI\Lib` directory need to be deleted. This needs to be done to avoid duplicate include files (in this example).

13.2.1.6 Step 6: Add file access routines

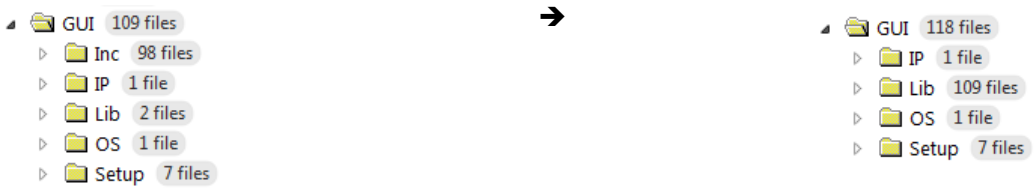
The next step is to add the file access routines to the project folder. Depending on if you want to use a file system, you have to copy one of the files located in `\Sample` into the data directory. The directory contains two files:

`APPW_X_NoFS.c`	to be used without a file system.
`APPW_X_emFile.c`	to be used with a file system, in this case emFile.

Because the hardware does not have an SD-card slot, we have to use `APPW_X_NoFS.c`. For this example, it has to be copied into the folder `\GUI\Setup\STM32F429_ST_STM32F429I_Discovery`.

13.2.1.7 Step 7: Add library to project

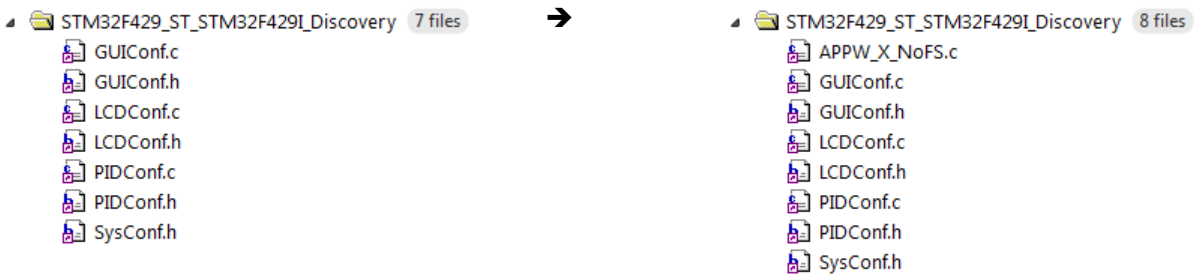
Now, open the project with SEGGER Embedded Studio. The project file is located under `BSP\ST\STM32F429_STM32F429I_Discovery`. Replace the content of the `Lib` folder with the new library and the new header files and remove the `Inc` folder.



> **Note**
>
> To remove the files, select them and press `DEL`. To add the new files, drag them from your file explorer into the correct folder in Embedded Studio.

13.2.1.8 Step 8: Add file access routines to the project

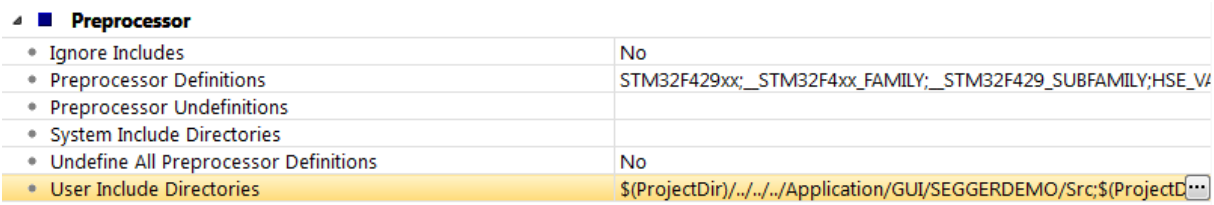
For the next step, add the file access routines to the project. The file is located under `GUI\Setup\STM32F429_ST_STM32F429I_Discovery`.



13.2.1.9 Step 9: Adjust include files

Select the project in the Embedded Studio Project Explorer and press **ALT + ENTER** to open the project settings dialog and choose common options:

Go to the preprocessor options…

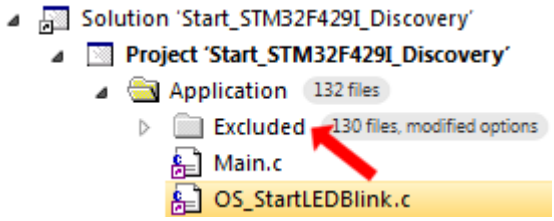


…and open the include directory dialog. Change `GUI\Inc` to `GUI\Lib`:

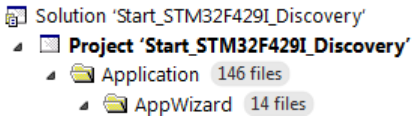
`$(ProjectDir)/../../../GUI/Inc` ➔ `$(ProjectDir)/../../../GUI/Lib`

13.2.1.10 Step 10: Add application to project

Now, you should add your application to the project. If not already done, the currently selected program should be moved into the "Excluded" folder.



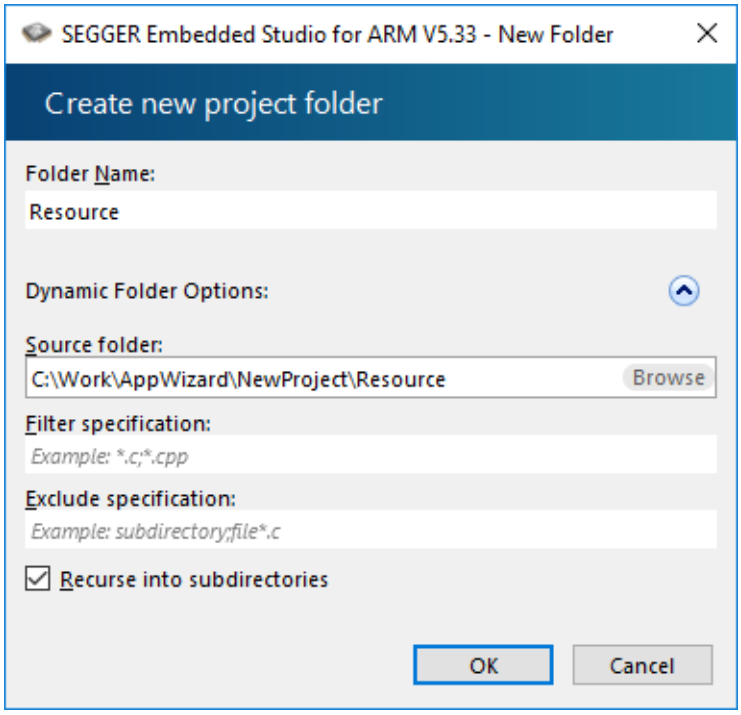
Right-click on the "Application" folder and select "New Folder", name the new folder *App-Wizard*.



Add a resource and source directory

Adding a folder for the resource and source files is done the same way.

1. Right-click on the newly created folder *AppWizard* and select "New Folder".
2. Name the folder *Resource* or *Source*, respectively and click "Dynamic Folder Options".
3. Tick "Recurse into subdirectories".
4. Click on "Browse" to select a source folder.
5. Navigate to your AppWizard project, select the "Resource" or "Source" folder, respectively and click "Select Folder".
6. Click OK.



13.2.1.11 Step 11: Compile and run on target

Now, your application should successfully compile and run on your target hardware. Don't forget to call **Build ➔ Clean Solution** after compiling the project, so the custom BSP won't include the generated object files.

> **Note**
>
> In order to flash the target using SEGGER Embedded Studio, make sure that the on-board ST-Link debugger has been upgraded to a J-Link, otherwise Embedded Studio will not be able to download the application onto the target. Click here to learn how this can be done.

13.3 Importing a custom BSP

To be able to have a custom BSP available in AppWizard's BSP repository, it has to be imported. To do that, select **File ➔ Import BSP...**. But before the above created BSP can be included, we have to move it into a different folder and add some further information and an image. The following steps demonstrate how this can be achieved.

13.3.1 Step 1: Create BSP folder

Create a folder somewhere with the exact name which should be shown into the BSP selection combo box. In this example, the folder is named `STM32F429I_Disco_ES`.

13.3.2 Step 2: Copy project into BSP folder

Take the folder `SeggerEval` of the above created project and copy it as a sub folder into the BSP directory and rename it to exactly the same name as its parent directory, in this case `STM32F429I_Disco_ES\STM32F429I_Disco_ES`.

13.3.3 Step 3: Add an image

When selecting a BSP in the AppWizard, an image is shown in the dialog. In this step, such an image is added to the BSP. The filename of the image must be `<Name of BSP>.jpg`, in this case the filename is `STM32F429I_Disco_ES\STM32F429I_Disco_ES.jpg`. Since the image shown in the dialog is quite small (80x80 pixels), it is recommended using a small image with dimensions of at least 80x80 pixels.

13.3.4 Step 4: Add information file

Each BSP contains a `.BSPInfo` file containing the following information:

- Display size
- Color conversion scheme
- Board name
- IDE
- MCU
- Manufacturer

Take one of the already existing `*.BSPInfo` files and copy it into the BSP folder. The file name must be of the format `<Name of BSP>.BSPInfo`. Open the file in a text editor and add the required information:

```
<!DOCTYPE emWin_AppWizard_BSP_Info>
<BSP>
  xSizeDisplay=240
  ySizeDisplay=320
  ColorConv=GUICC_M8888I
  BoardName=STM32F429I-Discovery
  IDE=Embedded Studio
  MCU=STM32F429IIT6U
  Manufacturer=STMicroelectronics
  MultibufAvail=1
</BSP>
```

Save the file as `STM32F429I_Disco_ES\STM32F429I_Disco_ES.BSPInfo`.

Option "MultibufAvail"

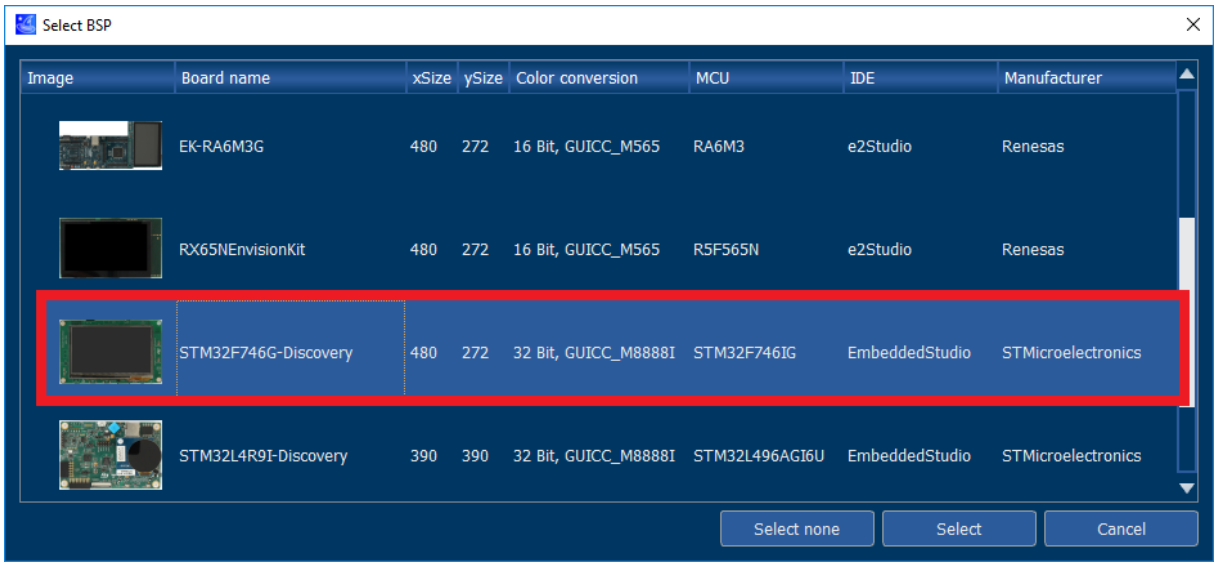
If the option `MultibufAvail` is set to 1 (or missing entirely) the AppWizard knows that the BSP supports multi buffering. Multi buffering can then be enabled and disabled through the project options.

If the option is set to 0, the option to toggle multi buffering for a project will not be visible within AppWizard.

13.3.5 Step 5: Import the BSP into AppWizard

To import the BSP into AppWizard, select **File ➔ Import BSP...**. Then, select the folder `STM32F429I_Disco_ES` (the folder that contains the evaluation project, the image and the `.BSPInfo` file).

This process can take a minute or longer. After that, a new BSP should be available within AppWizard:



13.4 Using the emWin source code

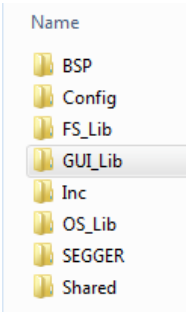
If you have purchased a emWin PRO you have access to the emWin source code. This source code can also be used within an AppWizard BSP. Either in a custom one or a BSP which comes along with the AppWizard. In general description on how to add the source code to a BSP should work for both cases.

For easiness this description is done by using the BSP for the STM32F746 Discovery which comes along with the AppWizard.

We assume that you are already familiar in using your IDE. Especially with adding new folder and setting new include paths.

13.4.1 Step 1: Remove the pre-compiled static libraries

At first delete the 'GUI_Lib' folder from the 'Target' directory within the AppWizard project.



13.4.2 Step 2: Add the source code to the project directory

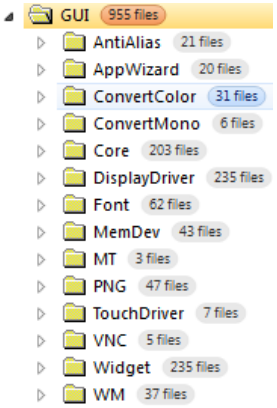
Copy the complete 'GUI' folder from your emWin shipment (found under emWin_ship) into the 'Target' directory. The 'GUI' folder contains the complete source code of emWin.

If you are using a BSP coming from SEGGER (either directly from the AppWizard or from our website) you should make sure that the following files in the 'GUI' directory are not present multiple times within the project. If they are delte those from the ''GUI' directory.

- Global.h
- SEGGER.h
- IP_FS.h

13.4.3 Step 3: Add the source code to the project

Within the Embedded Studio project for the STM32F746 Discovery add a new folder and name it 'GUI'. Now add all the subdirectories from the 'GUI' directory on your hard disk drive to the newly created 'GUI' folder in your Embedded Studio project. Although, it would be possible to use a different folder structure we strongly recommend to the structure as it is. This will make it easier if you intend to update to a newer emWin version.



13.4.4 Step 4: Set include paths

After adding the source code to the Embedded Studio project you have to set the proper include paths.

Add the include paths to the following directories in the Embedded Studio project.

- `GUI\AppWizard`
- `GUI\Core`
- `GUI\DisplayDriver`
- `GUI\Widget`
- `GUI\WM`

Chapter 14

AppWizard SPY

AppWizard SPY is an integration of the emWin SPY tool. This feature makes it possible to monitor the memory usage of the application, as well as the window properties of the application's widgets.

AppWizard SPY also offers the possibility to record any input to the application. A recording can be run and any findings can be written into a log file.

When recording, screenshots can be taken of the application and the state of variables and objects can be written to external files.

All SPY related files of a project are located in the project's Spy directory.

14.1 Requirements

Before using AppWizard SPY for the first time, a few requirements should be made sure of.

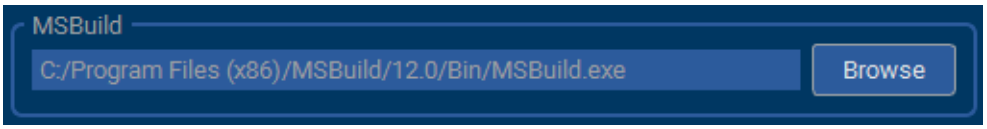
- The usage of AppWizard SPY requires **Microsoft Visual Studio 2013 or later** to be installed.
- The Visual Studio version must be selected in the Preferences dialog.
- The path to MSBuild.exe must be set.

Select Visual Studio version

Open the Preferences dialog by clicking “Edit ➔ Preferences”. Then, select the version that fits to your installation.

Path to MSBuild.exe

The path to MSBuild.exe must also be set in the Preferences dialog. This is required for AppWizard to build projects and use SPY.

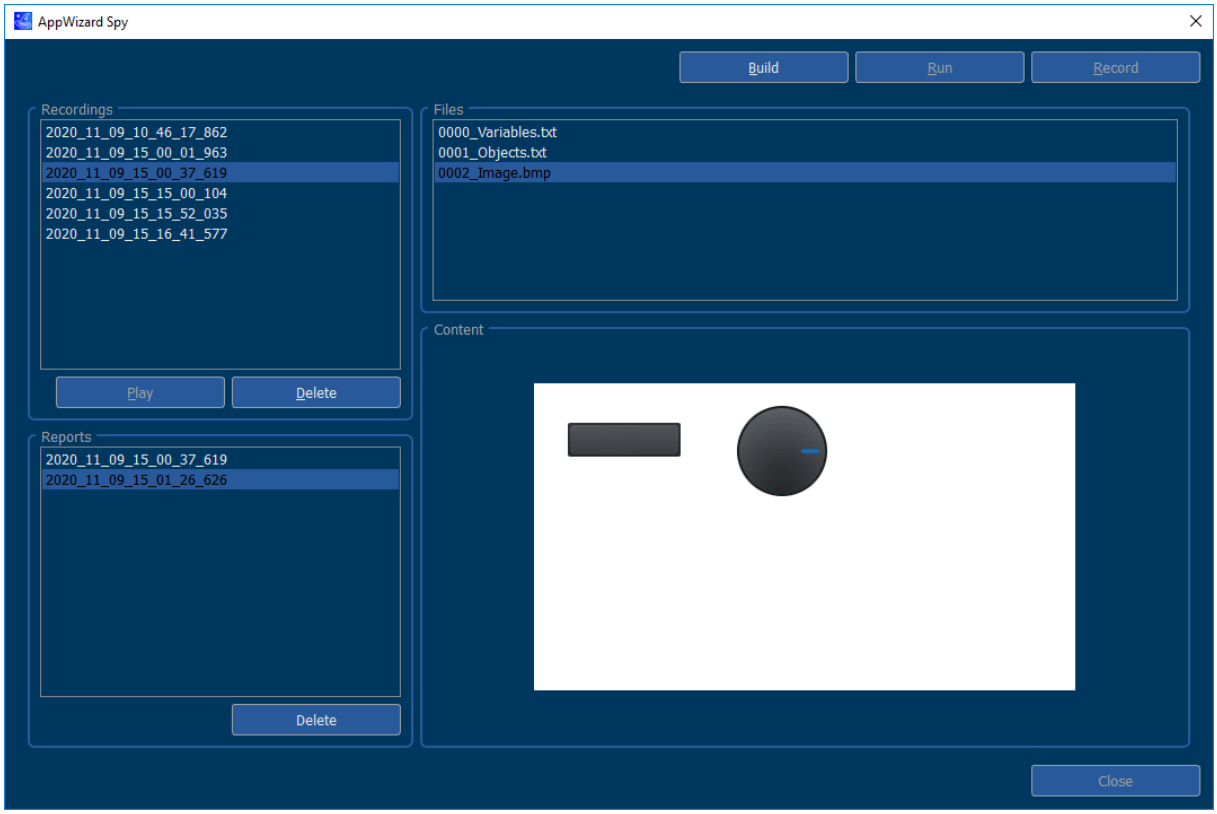


14.2 Opening the SPY dialog

The AppWizard SPY dialog can be opened by clicking `Project` ➔ `Start Spy`. Alternatively, the `F7` key can be pressed to open the dialog.

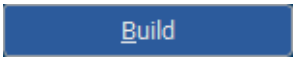
The left side of the dialog shows previously made recordings of the project. When a recording is selected, the associated log files are shown below under **Reports**.

In case a selected report has associated files, such as screenshots, they are shown under **Files** and can be viewed under **Content**.

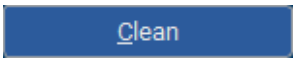


14.3 Building a project

Before a project can be run with AppWizard SPY, it has to be built. To do this, click the **Build** button.

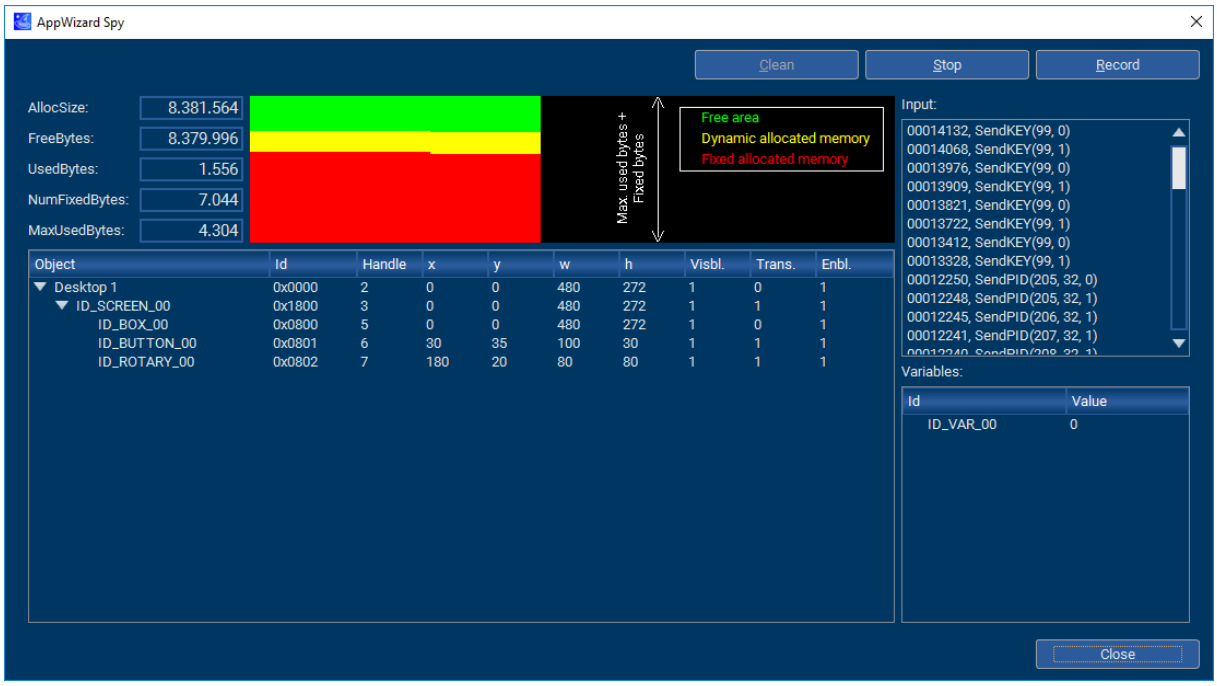


If the project should be rebuilt, the **Clean** has to be pressed first. Then, the project can be built again, as described above.



14.4 Running a project

When the build process has finished, the **Run** button can be clicked to start the application. The built `Simulation.exe` of the project is opened and the according data is shown in the SPY window.



Allocated memory

In the upper section of the SPY dialog, the available and allocated memory of the application is displayed by numbers and by a graph.

Objects

Below the allocated memory, the user finds a detailed tree of objects that are present in the application. The columns of the table contain the following object information:

Column	Description
Object	Hierarchical list of each object in the application. Shown is the ID of the object.
Id	AppWizard ID of the object as a hex number.
Handle	Window Manager handle number of the object.
x	X-position of the object.
y	Y-position of the object.
w	Width of the object.
h	Height of the object.
Visbl.	1 if the object is currently made visible, 0 if not.
Trans.	1 if the object has transparency (`WM_CF_HASTRANS` flag), 0 if not.
Enbl.	1 if the object is currently enabled, 0 if not.

Input

Any form of input (PID and keys) the application receives is shown here, as well as commands. If recording is active, this input will be saved to a file, so that it can be run at a later time. A more detailed description on the input and commands can be read further on under *Recording* on page 257.

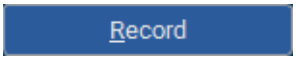
Variables

All AppWizard variables are shown in the lower right corner of the dialog and showing their current value, as the application is running.

14.5 Recording

The AppWizard SPY makes it possible to record a running AppWizard application. A recording is saved in the project directory and can be played at a later time.

To start a recording, the application has to be running. A recording can be started by pressing the **Record** button.



Logging of user input

The following input information is logged:

- PID information (touch pressed and released)
- Key information (key pressed and released)

Commands during recording

The following commands can be issued while recording an application.

Command	Hotkey	Description
Request Variables	F9	Writes the current state of all project variables into a dedicated file.
Request Objects	F11	Writes the contents of the object table into a dedicated file.
Request Screenshot	F12	Takes a screenshot of the application.

File syntax

> **Note**
>
> The syntax used for `.AppRec` files is not final and could change in future versions!

Every input the application receives or command that is issued during a recording is saved in a `.AppRec` file. The following base syntax is used:

```
<Ticks>, <Command>(<Params>)
```

The following commands exist, with their corresponding parameters:

Command	Parameters	Description
`SendPID`	`(x, y, Pressed)`	PID state at a given position.
`SendKEY`	`(Key, Pressed)`	A given key has been pressed or released.
`RequestVARIABLES`	None.	Saves the state of all project variables in a file.
`RequestOBJECTS`	None.	Saves the data of the object table in a file.
`RequestSCREEN`	None.	Takes a screenshot of the application.

Directory structure and file naming

All SPY-related files of a project are located in the `Spy` directory.

Every record file is saved in the following syntax format:

```
<YYYY>_<MM>_<DD>_<HH>_<MM>_<SS>_<MS>.AppRec
```

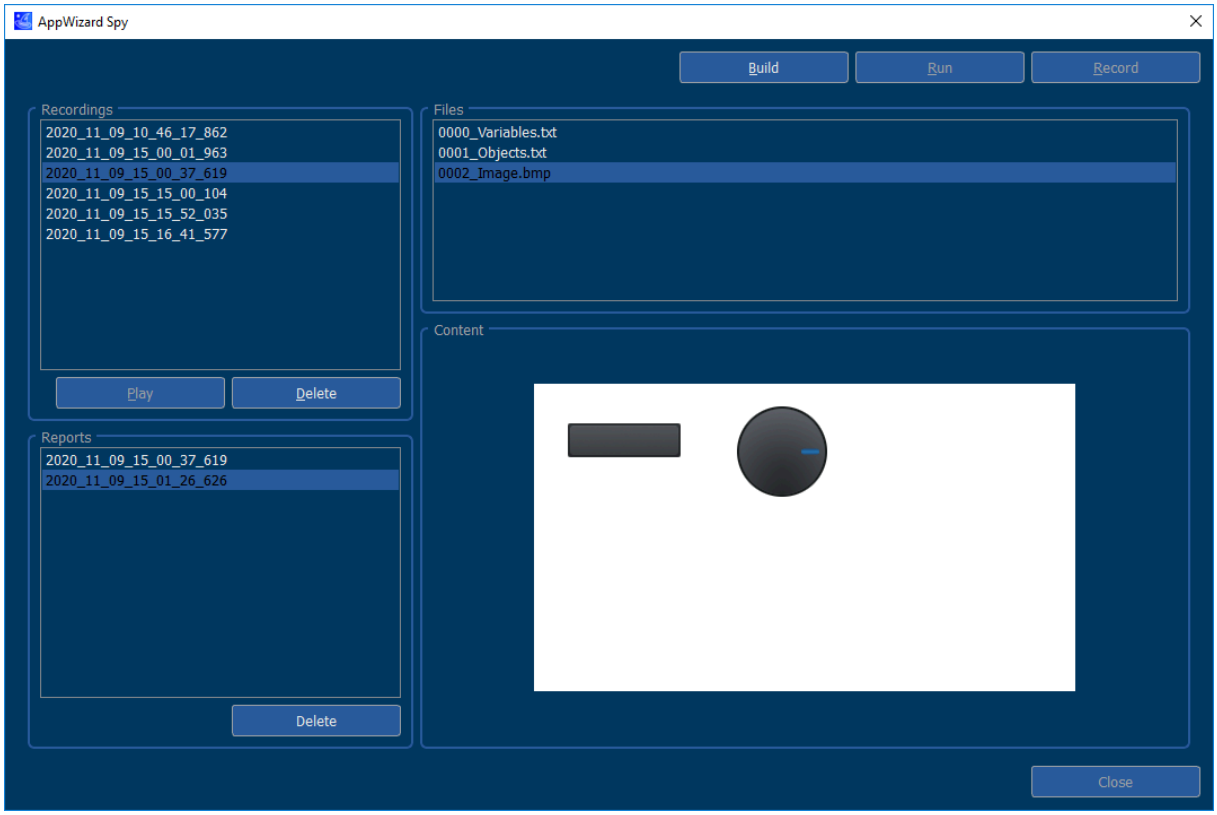
Example:

```
2020_11_09_10_46_17_862.AppRec
```


For each recording, a directory is created that contains any saved object or variable data, as well as screenshots of the application.

14.6 Playing a recording

To play an existing recording of a project, open the main AppWizard SPY dialog.



To play a recording, selected the desired recording and press the **Play** button.

The application will be executed and all the recorded data, such as PID, pressed keys, etc. will be applied to the application. Any exported data (e.g. screenshots) will be saved in a newly generated folder according to the current time in the same format as described above. This new sub-folder will be located in the main folder of the recording.

Chapter 15

Command line usage

AppWizard offers some commands that can be executed via the command prompt.

15.1 Command format

Commands are entered using the following format:

```
AppWizard.exe <ProjectName>.AppWizard <-command>
```

Example

The example below will open a given AppWizard project, export its code and then close AppWizard once the export is done.

```
AppWizard.exe ClimateControl.AppWizard -export -exit
```

15.2 Command line options

Command	Description
-export	Exports the code of a given AppWizard project.
-exit	Closes AppWizard.

Chapter 16

Glossary

BSP
Board support package.

embOS
Embedded real-time operating system.

emFile
Embedded file system.

emWin
Embedded graphics library.

Hierarchic tree
Widget on the left side of the AppWizard that displays the object hierarchy of the application.

MCU
Microcontroller unit.

Object
AppWizard equivalent of an emWin widget.

RAM
Random access memory.

ROM
Read-only memory.

SES
SEGGER Embedded Studio.

SPY
Tool for monitoring memory usage of the application. See *AppWizard SPY* on page 251.

`WM_PAINT`
Message sent to a window/widget by emWin's window manager. This message executes drawing operations of a window/widget.

WYSIWYG
What You See Is What You Get.

