

AppWizard

Wizard for creating ready-to-use emWin applications

User Guide & Reference Manual

Document: UM03003
Software Version: 1.02
Revision: 0
Date: March 18, 2020



A product of SEGGER Microcontroller GmbH

www.segger.com

Disclaimer

Specifications written in this document are believed to be accurate, but are not guaranteed to be entirely free of error. The information in this manual is subject to change for functional or performance improvements without notice. Please make sure your manual is the latest edition. While the information herein is assumed to be accurate, SEGGER Microcontroller GmbH (SEGGER) assumes no responsibility for any errors or omissions. SEGGER makes and you receive no warranties or conditions, express, implied, statutory or in any communication with you. SEGGER specifically disclaims any implied warranty of merchantability or fitness for a particular purpose.

Copyright notice

You may not extract portions of this manual or modify the PDF file in any way without the prior written permission of SEGGER. The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such a license.

© 2020 SEGGER Microcontroller GmbH, Monheim am Rhein / Germany

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: March 18, 2020

Software	Revision	Date	By	Description
1.02	1	200318	FO	Chapter <i>User interface</i> updated. <ul style="list-style-type: none"> • Screenshots updated. • Added information about Thai support.
1.02	0	200313	FO	Chapter <i>User Code</i> added. <ul style="list-style-type: none"> • Sub-chapter <i>Slot routines</i> merged. • Sub-chapter <i>Screen callback routines</i> added. • Sub-chapter <i>Fonts</i> added. • Sub-chapter <i>Variables</i> merged.
1.00	1	200306	FO	Chapter <i>Interactions</i> updated. <ul style="list-style-type: none"> • 'Slot routines' section updated. • 'Custom user code' section added. Chapter <i>Board support packages (BSPs)</i> updated. <ul style="list-style-type: none"> • Examples updated.
1.00	0	200226	JE FO	Initial release. Chapter <i>Board support packages (BSPs)</i> updated. <ul style="list-style-type: none"> • 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 <i>Getting Started</i> updated. <ul style="list-style-type: none"> • Added note about the AppWizard Quick Start Guide. Chapter <i>Interactions</i> updated. <ul style="list-style-type: none"> • '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 *The C Programming Language* by Kernighan and Richie (ISBN 0--13--1103628), which describes the standard in C programming and, in newer editions, also covers the ANSI C standard.

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 or other documents.
Emphasis	Very important sections.

Table of contents

1	Introduction	11
1.1	What is the AppWizard?	12
1.2	Features	13
1.3	Requirements	15
1.3.1	Host system	15
1.3.2	Target system	15
1.3.3	Development environment	15
1.3.4	Additional software libraries	15
2	Installation	16
2.1	Windows	17
3	Getting started	18
3.1	Starting the tool	19
3.2	Creating a new project	19
3.3	Opening existing projects	21
4	User interface	22
4.1	Menu bar	23
4.2	Editor window	25
4.3	Property window	27
4.3.1	Id, position and size	27
4.3.2	Positioning logic	27
4.3.3	Positioning details	29
4.3.4	Object dependent details	30
4.4	Hierarchic tree view	31
4.5	Play window	32
4.6	Interaction window	33
4.7	Quick access buttons	34
4.7.1	Text resource window	34
4.7.2	Font resource window	35
4.7.3	Image resource window	36
4.7.4	Variable resource window	37
5	Resource management	38
5.1	Stock resources	39
5.2	Text management	40
5.3	Font management	41
5.3.1	Font creation options	42

5.3.2	Definition of code point ranges	42
5.4	Image management	43
5.5	Variable management	44
6	Objects	45
6.1	Introduction	46
6.2	Object properties	48
6.2.1	Alignment	49
6.2.2	Auto repeat	50
6.2.3	Bitmap	51
6.2.4	Blend colors	52
6.2.5	Blink period	53
6.2.6	Border size	54
6.2.7	Color and background color	55
6.2.8	Cursor inversion	56
6.2.9	Decimal mode	57
6.2.10	Fade mode	58
6.2.11	Font	59
6.2.12	Frame radius	60
6.2.13	Frame size	61
6.2.14	Gradient	62
6.2.15	Motion	63
6.2.16	ID	64
6.2.17	Initial value	65
6.2.18	Invert direction	66
6.2.19	JPEG/GIF/BMP	67
6.2.20	Maximum length	68
6.2.21	Offset	69
6.2.22	Period	70
6.2.23	Persistent mode	71
6.2.24	Position and size	72
6.2.25	Radius	73
6.2.26	Range	74
6.2.27	Rotate marker	75
6.2.28	Snap position	76
6.2.29	Span of values	77
6.2.30	Text	78
6.2.31	Text color	79
6.2.32	Tiling	80
6.2.33	Vertical mode	81
6.3	Box	82
6.4	Button	83
6.5	Edit	84
6.6	Image	85
6.7	Rotary	86
6.8	Screen	87
6.9	Slider	88
6.10	Switch	89
6.11	Text	90
6.12	Window	91
7	Interactions	92
7.1	Introduction	93
7.2	List of signals	95
7.2.1	APPW_NOTIFICATION_ANIMCOORD	96
7.2.2	APPW_NOTIFICATION_ANIMEND	97
7.2.3	APPW_NOTIFICATION_ANIMSTART	98
7.2.4	APPW_NOTIFICATION_CREATE	99
7.2.5	APPW_NOTIFICATION_DELETE	100

7.2.6	APPW_NOTIFICATION_INITDIALOG	101
7.2.7	APPW_NOTIFICATION_MOTION	102
7.2.8	WM_NOTIFICATION_CLICKED	103
7.2.9	WM_NOTIFICATION_GOTFOCUS	104
7.2.10	WM_NOTIFICATION_LOSTFOCUS	105
7.2.11	WM_NOTIFICATION_MOTIONSTOPPED	106
7.2.12	WM_NOTIFICATION_RELEASED	107
7.2.13	WM_NOTIFICATION_VALUECHANGED	108
7.3	List of jobs	109
7.3.1	APPW_JOB_ADDVALUE	110
7.3.2	APPW_JOB_ANIMCOORD	111
7.3.2.1	Dispose indexes	112
7.3.3	APPW_JOB_ANIMVALUE	113
7.3.4	APPW_JOB_ANIMRANGE	114
7.3.5	APPW_JOB_CASCADECOORD	115
7.3.6	APPW_JOB_CLEAR	116
7.3.7	APPW_JOB_SET	117
7.3.8	APPW_JOB_SETBKCOLOR	118
7.3.9	APPW_JOB_SETCOLOR	119
7.3.10	APPW_JOB_SETCOORD	120
7.3.11	APPW_JOB_SETENABLE	121
7.3.12	APPW_JOB_SETLANG	122
7.3.13	APPW_JOB_SETSIZE	123
7.3.14	APPW_JOB_SETTEXT	124
7.3.15	APPW_JOB_SETVALUE	125
7.3.16	APPW_JOB_SETVIS	126
7.3.17	APPW_JOB_SHIFTSCREEN	127
7.3.18	APPW_JOB_SHOWSCREEN	128
7.3.19	APPW_JOB_SWAPSCREEN	129
7.3.20	APPW_JOB_TOGGLE	130
7.3.21	NULL	131
8	User Code	132
8.1	Slot routines	133
8.1.1	APPW_ACTION_ITEM	134
8.1.2	Custom user code	135
8.2	Screen callback routines	136
8.3	Fonts	137
8.3.1	How to use fonts	137
8.3.2	Font API	138
8.3.2.1	APPW_GetFont()	139
8.4	Variables	140
8.4.1	How to use variables	140
8.4.2	Variables API	140
8.4.2.1	APPW_GetVarData()	141
8.4.2.2	APPW_SetVarData()	142
9	Board support packages (BSPs)	143
9.1	Preconfigured BSPs included in the shipment	144
9.1.1	Example	144
9.1.1.1	Step 1: Select BSP	144
9.1.1.2	Step 2: Generate code	144
9.1.1.3	Step 3: Run SEGGER Embedded Studio Project	144
9.1.1.4	Step 4: Compile and run on target	145
9.2	Creating custom BSPs	146
9.2.1	Example	146
9.2.1.1	Step 1: Create a project with AppWizard	146
9.2.1.2	Step 2: Create some elements	146
9.2.1.3	Step 3: Export & Save	146

9.2.1.4	Step 4: Copy evaluation software package into project folder	146
9.2.1.5	Step 5: Exchange libraries	147
9.2.1.6	Step 6: Add file access routines	148
9.2.1.7	Step 7: Add library to project	148
9.2.1.8	Step 8: Add file access routines to the project	148
9.2.1.9	Step 9: Adjust include files	148
9.2.1.10	Step 10: Add application to project	149
9.2.1.11	Step 11: Compile and run on target	150
9.3	Importing a custom BSP	151
9.3.1	Step 1: Create BSP folder	151
9.3.2	Step 2: Copy project into BSP folder	151
9.3.3	Step 3: Add an image	151
9.3.4	Step 4: Add information file	151
9.3.5	Step 5: Import the BSP into AppWizard	151
10	Glossary	153

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.

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.

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.

Precompiled BSPs

The AppWizard comes with a set of BSPs which include also precompiled libraries of emFile and embOS. In case of using a BSP the possibility of changing 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.

Variables

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.

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

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

Note

*1 128 KBytes of flash memory are required for emWin. Additional flash memory or external storage is required for resources like fonts, images and text.

*2 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

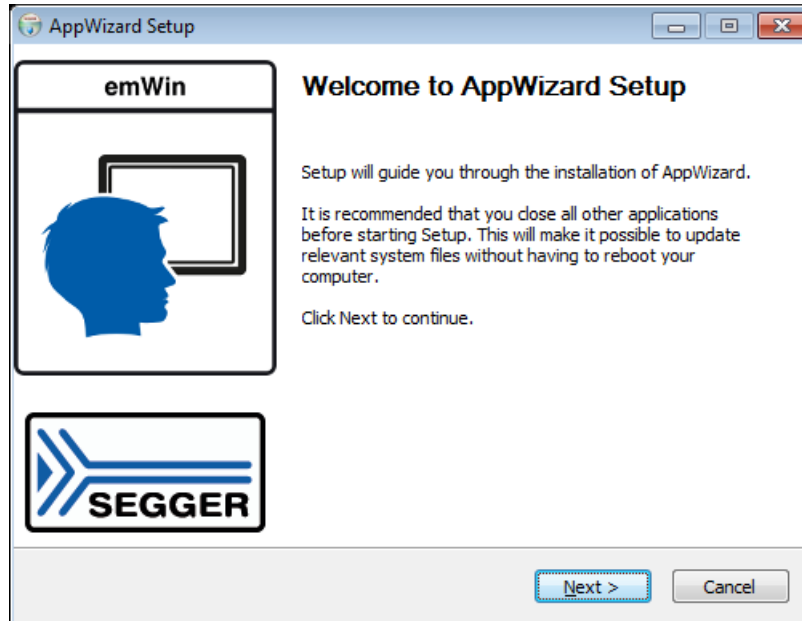
Installation

The following chapter describes how to install the AppWizard.

2.1 Windows

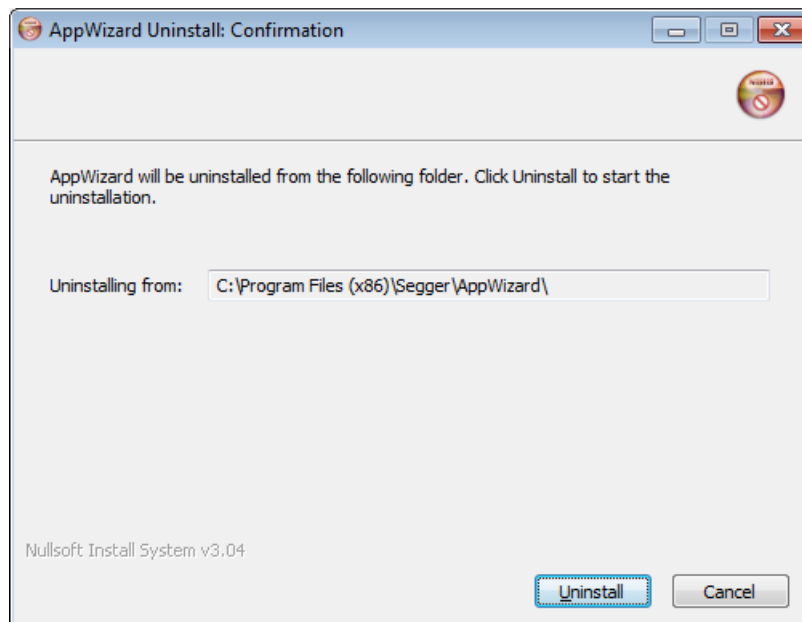
Installing the AppWizard

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 `docs` 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.

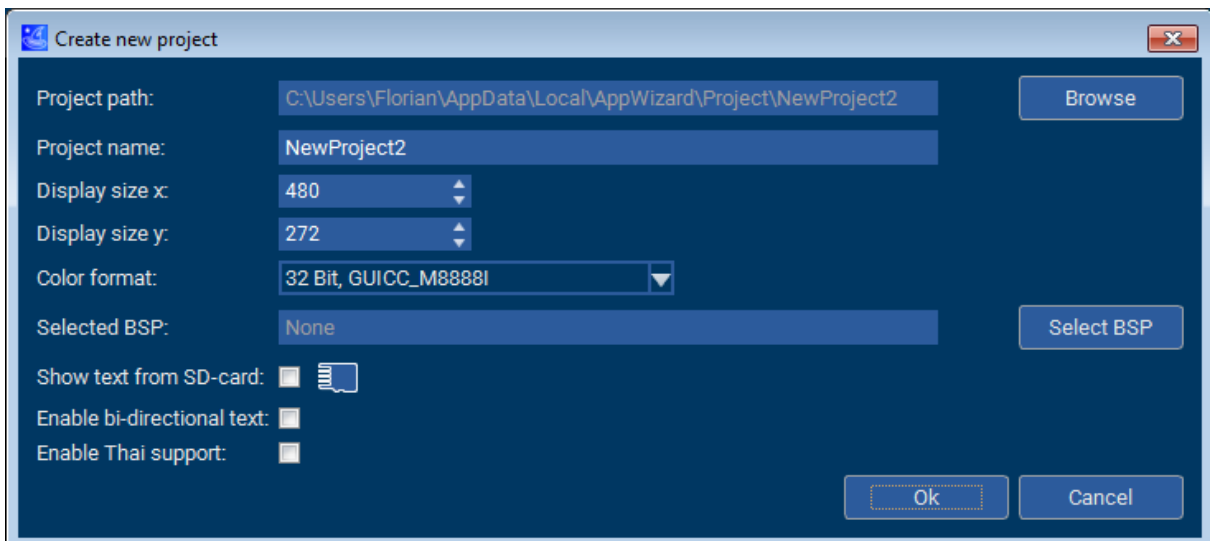
3.2 Creating a new project

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

1 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.

2 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 22. To learn more about objects, see the chapter *Objects* on page 45.

3 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 32.

④ **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.

⑤ **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.

⑥ **Compile and run on target**

The chapter *Board support packages (BSPs)* on page 143 explains how to run a project on a hardware target.

3.3 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

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:

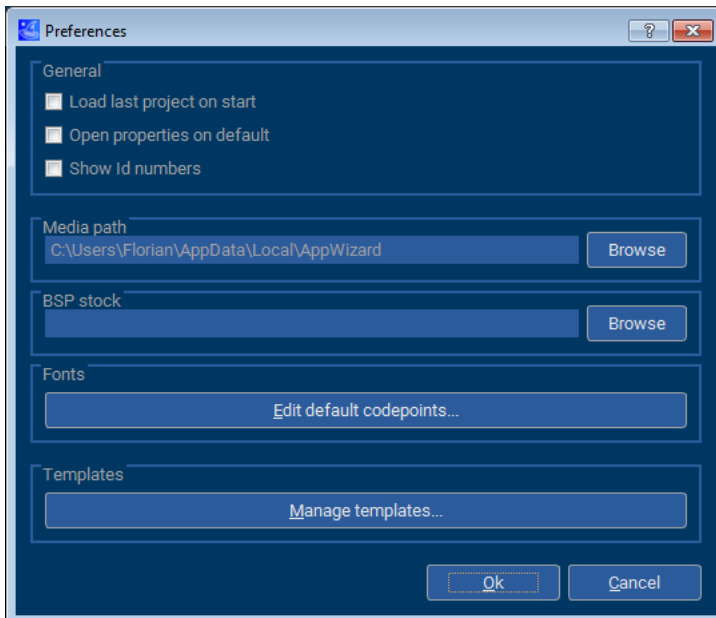
- 'Editor' window (center/top)
- 'Interactions' window (center/bottom)
- 'Add objects' window (left/top)
- 'Hierarchic tree view' window (left/bottom)
- 'Properties' window (right)
- Quick access buttons for text, fonts, images and variables at the lower left edge

4.1 Menu bar

It consists of the following items:

- File (New project, Close project, Save, Open, Save as, Export & Save, Exit, Recent files)
- Edit (Undo, Redo, Cut, Copy, Paste, Delete, Select all, Preferences)
- Project (Edit options)
- Resource (Edit Text, Edit Fonts, Edit Images, Edit Variables)

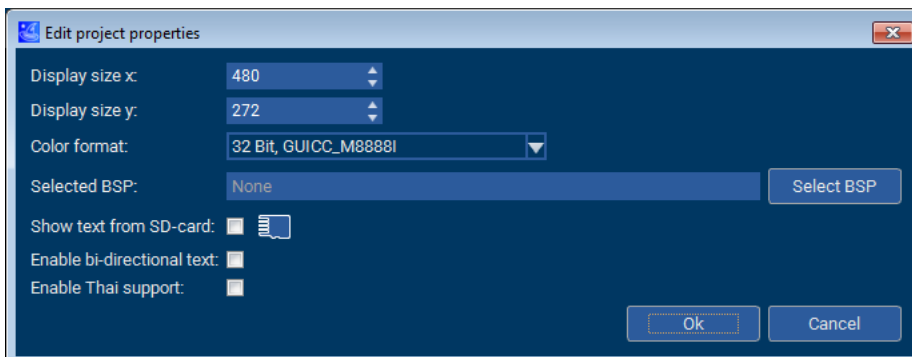
Edit / Preferences



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:** Shows the Window Manager Id next to the AppWizard Id.
- **Media path:** Path to external media.
- **BSP stock:** Sets a path, where custom BSPs are located.
- **Edit default codepoints...:** Editing the default range of code points to be used for new fonts.

Project / Edit options



The project options dialog has the following options:

- **Display size x:** Horizontal display size.
- **Display size y:** Vertical display size.
- **Color format:** Desired color format.
- **Selected BSP:** Desired BSP for target hardware.
- **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.

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

4.2 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 27.

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 32.

4.3 Property window

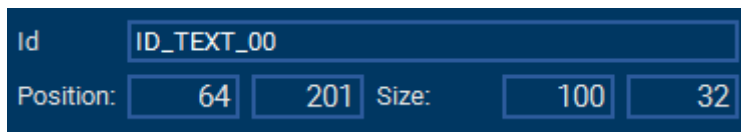
The window on the right shows the object specific properties. It consists of four areas (top to bottom):

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

4.3.1 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.



4.3.2 Positioning logic

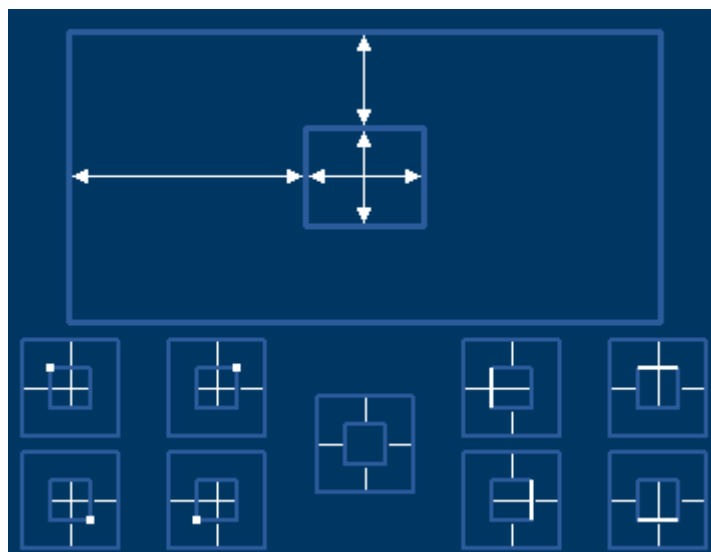
The rectangle of a simple emWin window is defined by its upper left position and its X- and Y-size. To be more flexible with this, the AppWizard supports more options.

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
	<p>Top and left coordinate relative to parent. Width and height defined by given value.</p>
	<p>Top and right coordinate relative to parent. Width and height defined by given value.</p>
	<p>Bottom and right coordinate relative to parent. Width and height defined by given value.</p>
	<p>Bottom and left coordinate relative to parent. Width and height defined by given value.</p>
	<p>Top, left, bottom and right coordinate relative to parent.</p>
	<p>Top, left and bottom coordinate relative to parent. Width defined by given value.</p>
	<p>Top, left and right coordinate relative to parent. Height defined by given value.</p>
	<p>Top, right and bottom coordinate relative to parent. Width defined by given value.</p>
	<p>Left, bottom and right coordinate relative to parent. Height defined by given value.</p>

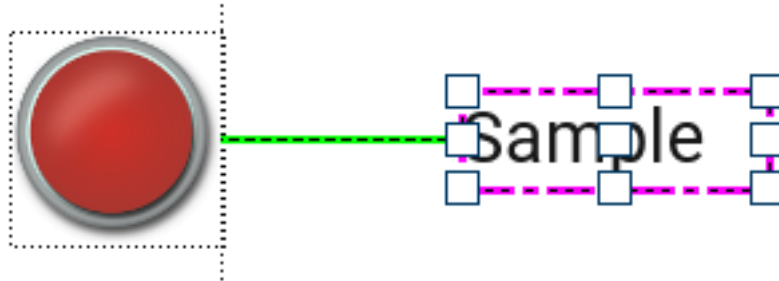
Note

The positioning logic can be changed at anytime.

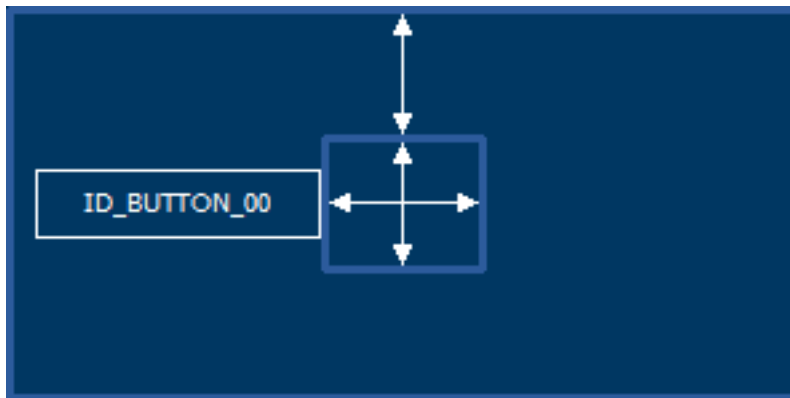
Example

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.



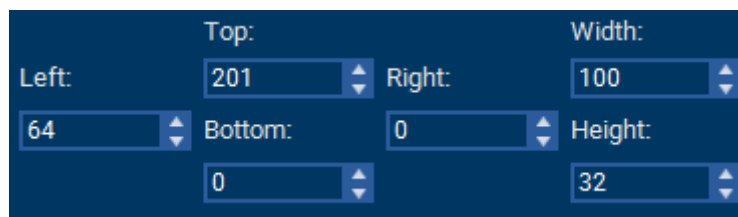
When selecting the Text object, the positioning logic property shows that its X-axis is dependent on the Button object.



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

4.3.3 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.



4.3.4 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 48.

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.



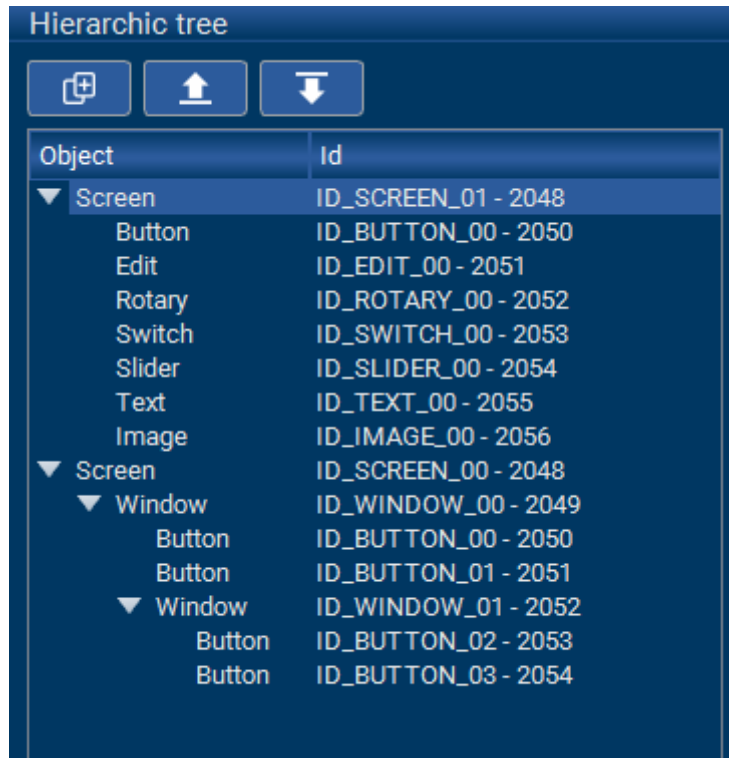
To delete an existing property, the **X** button on the right side has to be clicked.

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.

4.4 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.

4.5 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.

4.6 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.



+/-	Edit	Emitter	Signal	Job	Receiver
X		ID_BUTTON_00	WM_NOTIFICATION_CLICKED	APPW_JOB_ADDVALUE	ID_ROTARY_00
X		ID_EDIT_00	WM_NOTIFICATION_GOT_FOCUS	APPW_JOB_SETTEXT	ID_TEXT_00
+					

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.

4.7 Quick access buttons

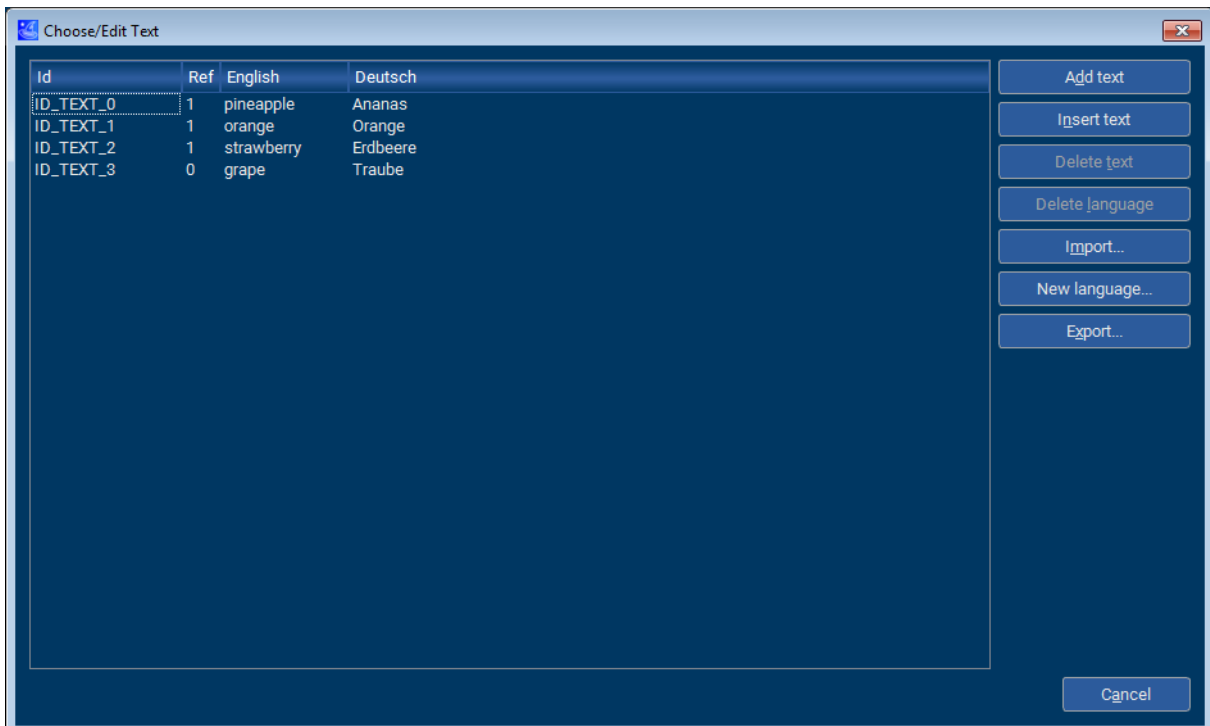
In the lower left corner there are four buttons for managing resources. Clicking on one of the four buttons (text, fonts, images and variables) will open the corresponding resource window.

For more information about managing resources, see the chapter *Resource management* on page 38.



4.7.1 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.

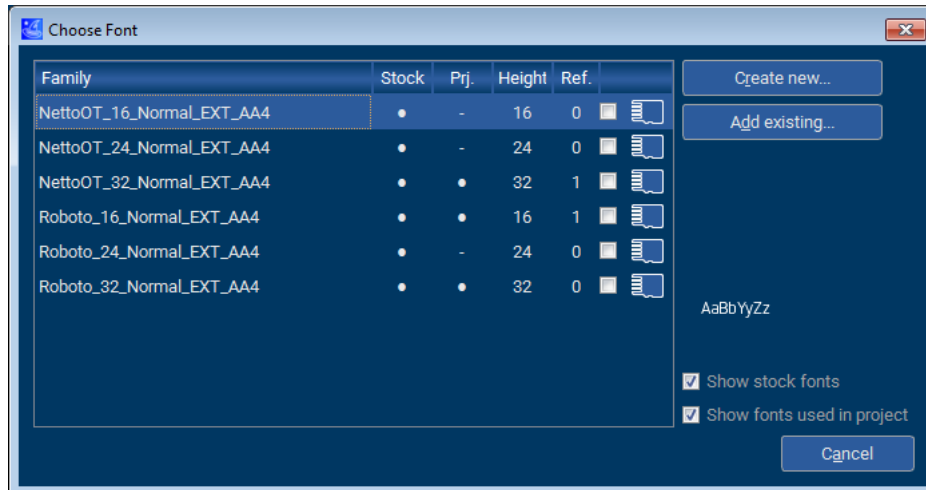
Texts can be deleted via the **Delete text** button, but only when they have no references.

Export and import texts

It is also possible to export the texts to a CSV file or import a CSV file. For importing CSV files, the rules for CSV files should be obeyed. These rules are described in the emWin User Manual under "32.2.4 Rules for CSV files" in the chapter "Language Support".

4.7.2 Font resource window

The font resource window allows the user to manage fonts.



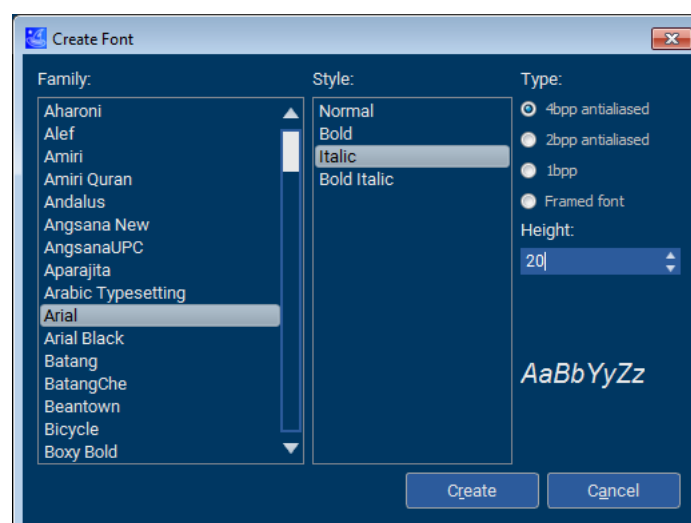
The table shows whether the font is a stock font and/or is used in the project. It also shows how often fonts are referenced in any objects and the height of the font. By ticking the checkbox next to the SD card, the font can be marked as an external resource.

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.

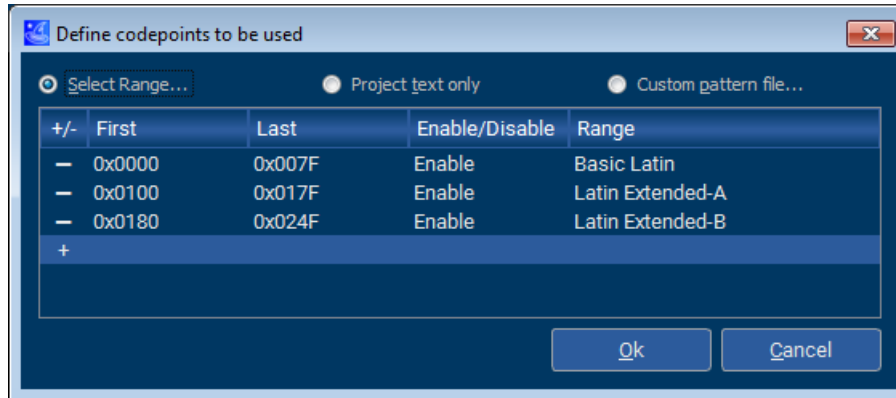


To optimize memory footprint, the user may define which characters should be present in the font. This can be done by clicking on **Codepoint range...** and either selecting a range

of characters, keeping only the characters that are used in all project's texts or by parsing a pattern file.

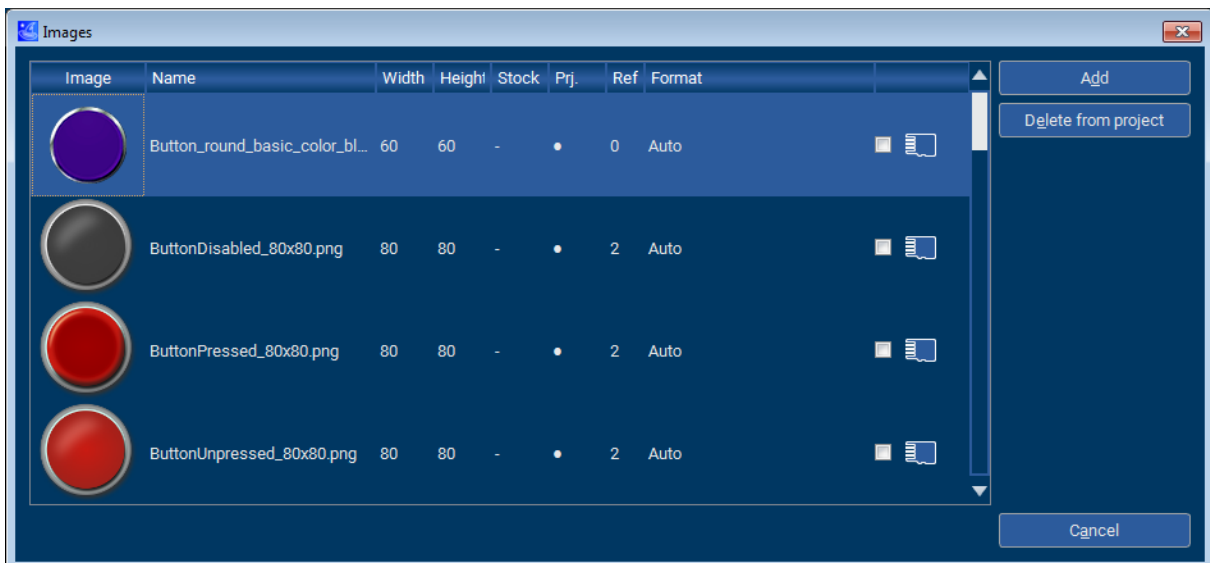
By default, a range of characters is used for creating a font. The default range of enabled characters is:

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



4.7.3 Image resource window

The image resource window allows the user to manage images.

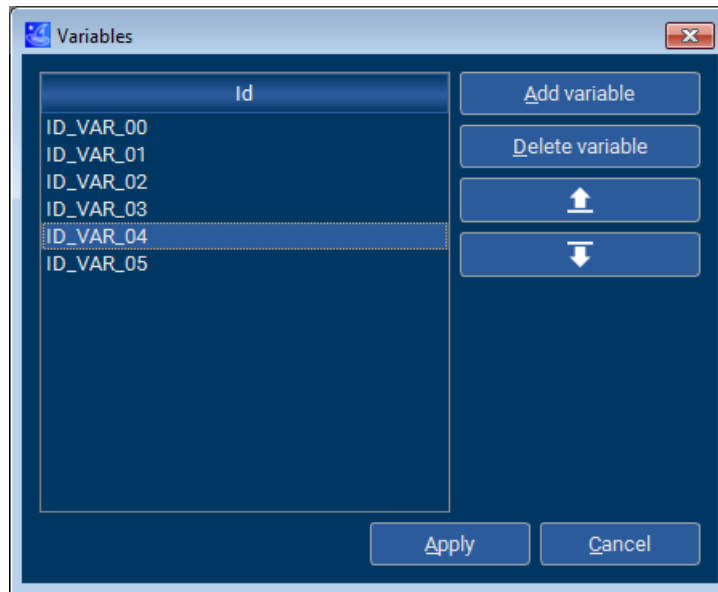


The window gives an overview of all the images used in the project, showing also their dimensions, the bitmap format applied to the image and how often it is referenced in other objects.

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.

4.7.4 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 140.



Chapter 5

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.

5.1 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!

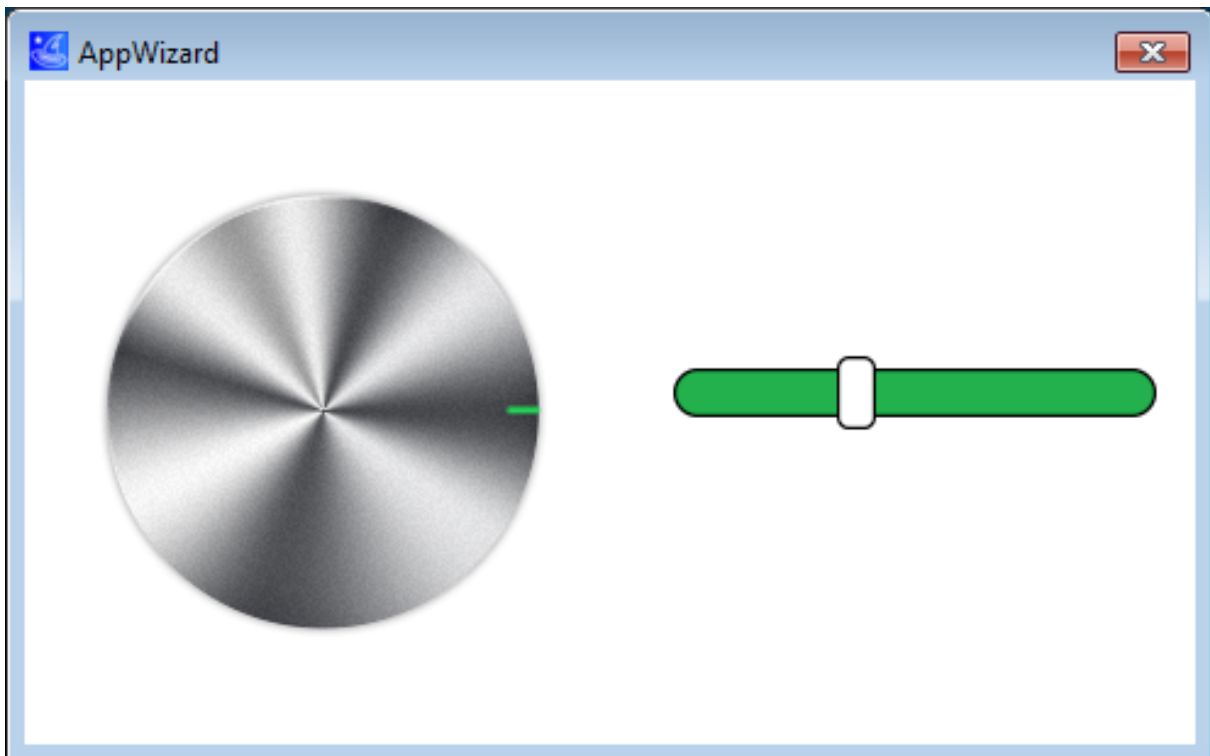
Stock fonts

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

- NettoOT_16_Normal_EXT_AA4
- NettoOT_24_Normal_EXT_AA4
- NettoOT_32_Normal_EXT_AA4
- Roboto_16_Normal_EXT_AA4
- Roboto_24_Normal_EXT_AA4
- Roboto_32_Normal_EXT_AA4

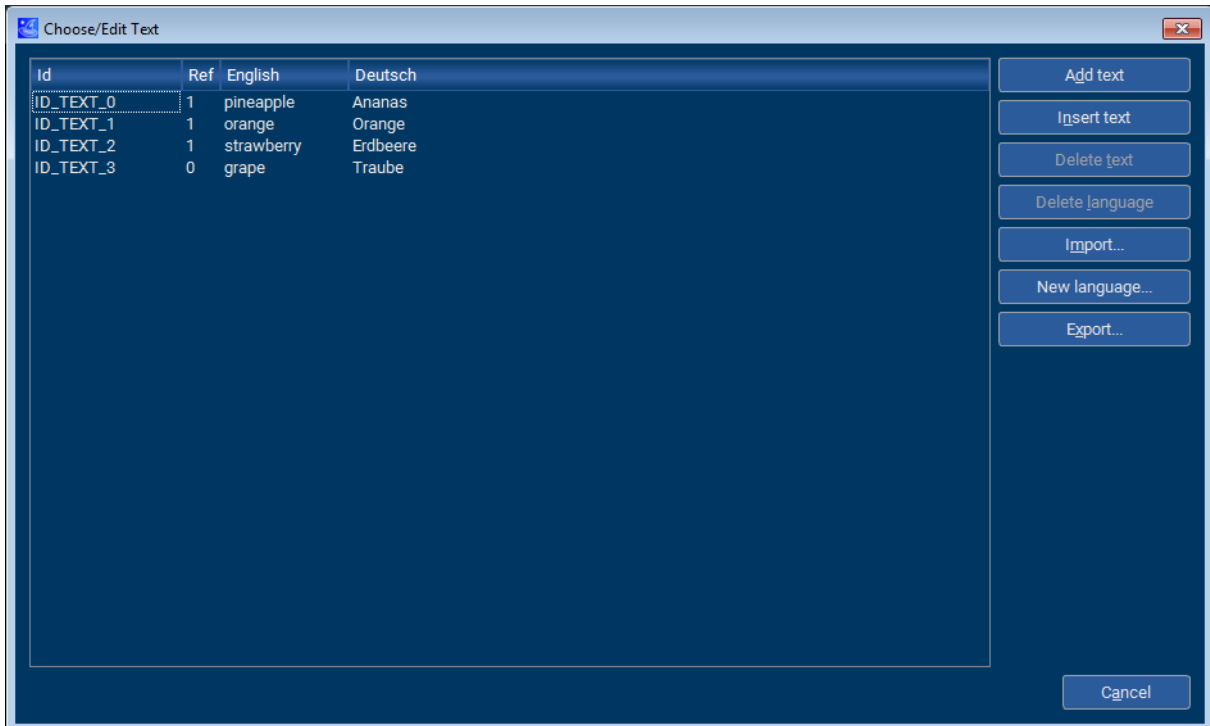
Stock images

For every object that can make use of a bitmap, there are stock images to use for supplied by the AppWizard. There are for example bitmaps for a Rotary and its marker, or a thumb and shaft bitmap for a Slider object.



5.2 Text management

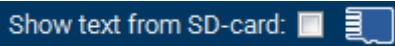
A text input dialog allows entering text in multiple languages. Text usage is based on using IDs instead of using strings directly. Text access within the application is realized by using text IDs. In combination with emWin's language module it becomes quite easy to switch between languages.



Managing text from SD card

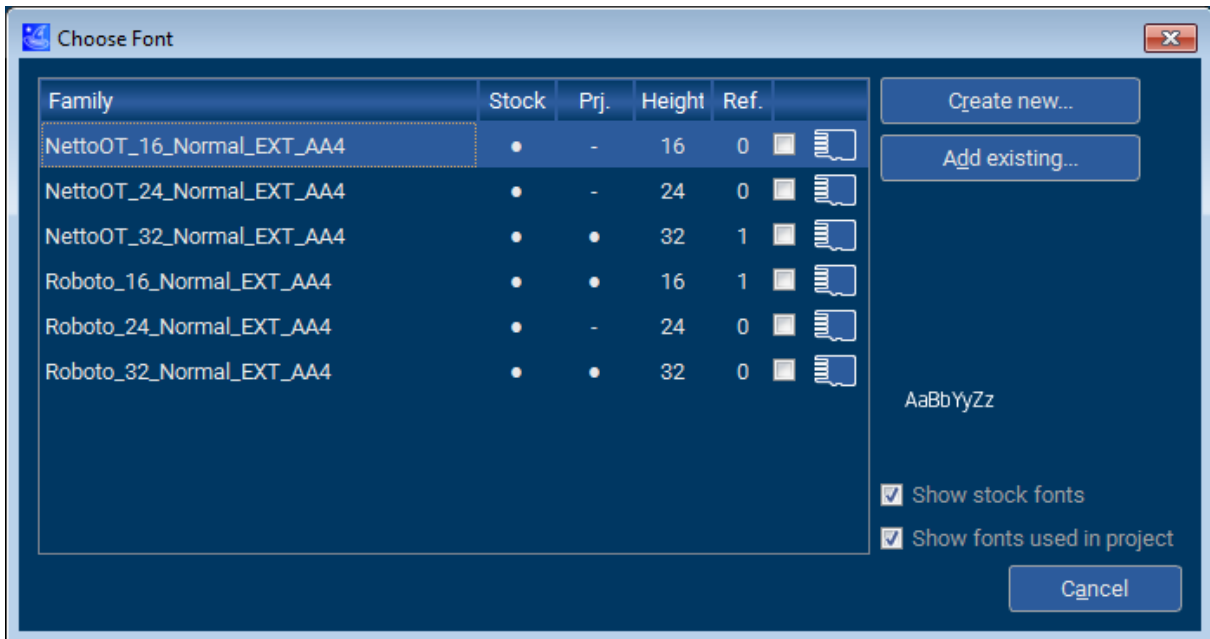
The project property dialog has the option to enable text management directly from SD card. In that case the text is not compiled and linked with the application code.

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



5.3 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 references 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 per default are compiled and linked with the application. 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.

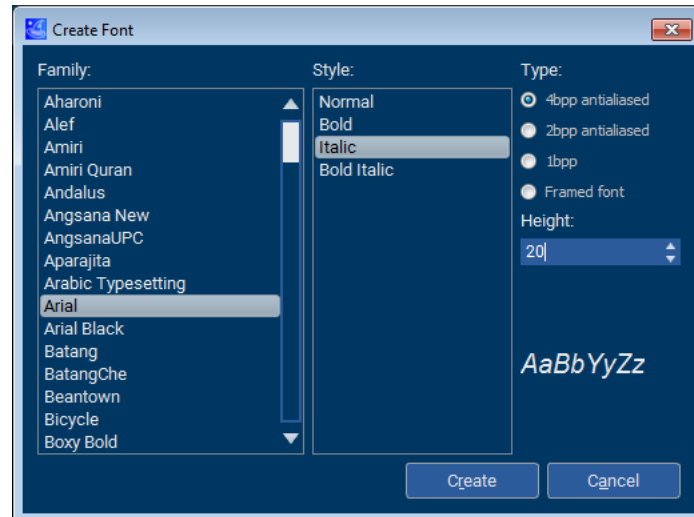
The resource folder of the project contains all fonts which are used or have been used within the project. That means the folder can contain fonts which are not currently used. Those fonts are shown with zero references and can be deleted by pressing the 'Delete' button if not planned to be used any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

5.3.1 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 fonts generated by the AppWizard contain additional information, therefore only fonts created by the AppWizard can be used.



The following types of fonts can be created:

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

5.3.2 Definition of code point ranges

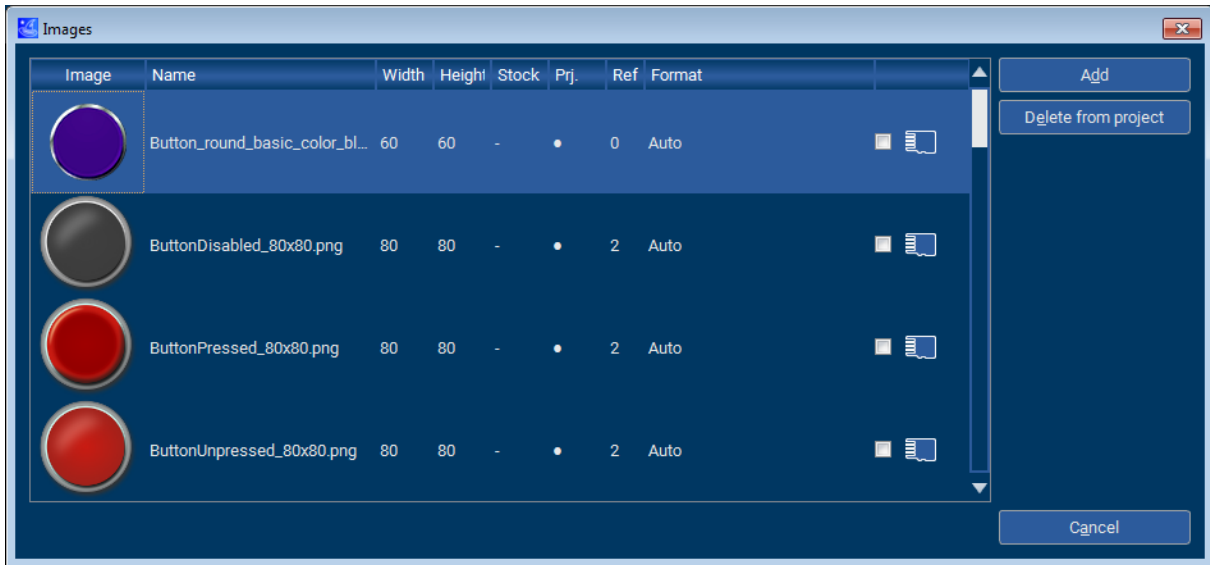
Each font can have its own range of code points. The font selection dialog has the option for specifying the desired code point range.

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

5.4 Image management

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.

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 they haven't been referenced, that means the reference count shows zero.

5.5 Variable management

The variable window allows the user to manage the variables for this project. 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.

Chapter 6

Objects









This chapter gives an overview of the objects the AppWizard supports.



6.1 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

To learn more about the object's emWin counterparts, refer to the document **UM03001 emWin User Guide & Reference Manual**.

Name	Symbol	emWin Counterpart	Description
Box		GUI_RECT	Box object that can be colored.
Button		BUTTON widget	Clickable button object.
Edit		EDIT widget	Edit field for user input.
Image		IMAGE widget	Object that displays an image.
Rotary		ROTARY widget	Circular object that can be rotated.
Screen		-	A screen serves as a parent for all other objects.
Slider		SLIDER widget	Movable slider.
Switch		SWITCH widget	Toggleable switch with two states.

Name	Symbol	emWin Counterpart	Description
Text		TEXT widget	An object displaying text.
Window		WINDOW widget	Similar to screen object, serves as a parent object for other objects.

6.2 Object properties

Every object has its own properties than can be edited. The following section will give an overview of each different property.

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

Property	Description
Alignment	Alignment of an object.
Auto repeat	Repeated clicking of a button.
Bitmap	Bitmap to be shown in an object.
Blend colors	Blending colors for Slider object.
Blink period	Cursor blinking period for Edit object.
Border size	Size of border for Edit object.
Color and background color	Foreground and background colors for object.
Cursor inversion	When disabling cursor inversion, the color for the cursor isn't the inverted background color.
Decimal mode	Makes an Edit or Text object only eligible for digits.
Fade mode	Fades the two Switch state bitmaps into each other.
Font	Font for the object.
Frame radius	Radius of frame around Edit object.
Frame size	Pixel-size of frame around Edit object.
Gradient	Horizontal and vertical gradients.
Motion	Motion settings for swiping between screens.
ID	ID for the object.
Initial value	Initial value for Rotary object.
Invert direction	Inverts direction of a Slider object.
JPEG/GIF/BMP	Image to be shown.
Maximum length	Maximum text length.
Offset	Offset for a Rotary object.
Period	Movement period.
Persistent mode	Persistent mode for screens.
Position and size	Position and size of an object.
Radius	Radius of a Rotary object.
Range	Range of position values of an object.
Rotate marker	Toggle rotate markers of a Rotary object.
Snap position	Snapping position of a Rotary object.
Span of values	Set range of values of a Rotary object.
Text	Text to be shown.
Text color	Color for text.
Tiling	Tiling mode for Image objects.
Vertical mode	Changes a slider object to be vertical.

6.2.1 Alignment

Description





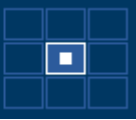




The alignment property allows to choose a combination of a horizontal alignment and a vertical alignments. This property can be set for bitmaps and texts.

Available objects

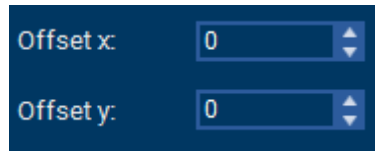
This property can be set for the following objects:

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

Usage

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

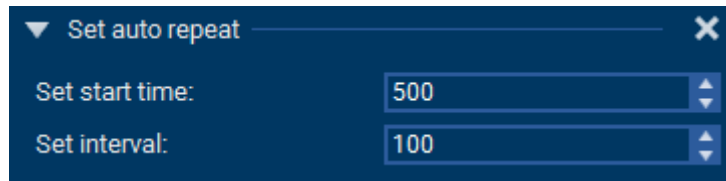
You may also add an x and y offset to the object.



6.2.2 Auto repeat

Description

Enables the 'auto repeat mode' of a button. When holding the button pressed, it begins sending clicked events after the start time period in the given interval.



Available objects

This property can be set for the following objects:

- *Button* object

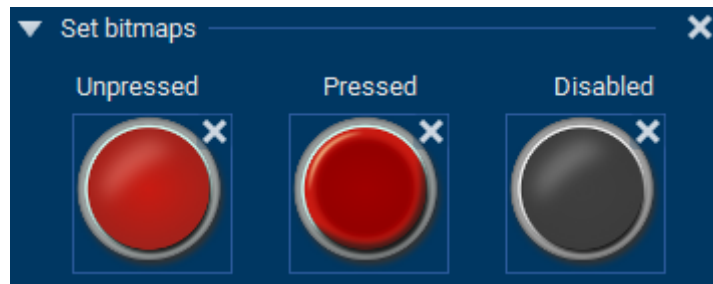
Specification

Property	Description
Start time	Period to wait until the repeat should start.
Interval	Repeating interval to be used.

6.2.3 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
- *Image* object
- *Slider* object
- *Switch* object
- *Rotary* object

6.2.4 Blend colors

Description

This property allows to define colors, that will be blended into the left and right bitmaps of a Slider object's shaft.



Available objects

This property can be set for the following objects:

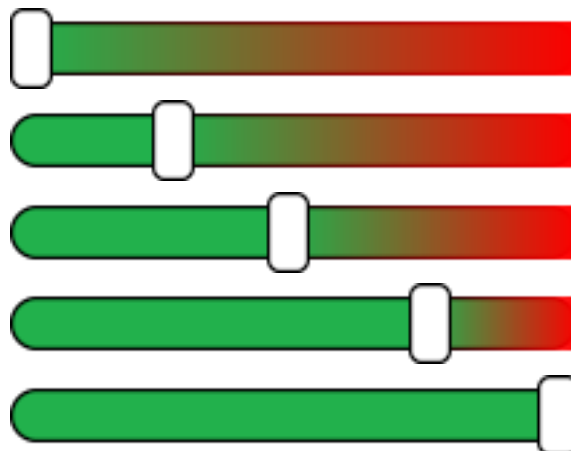
- *Slider* object

Specification

Property	Description
Shaft left	Color to be blended in on the left end of the shaft.
Shaft right	Color to be blended in on the right end of the shaft.

Example

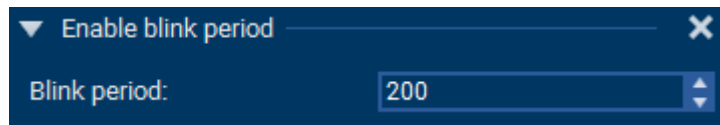
This example uses the green shaft bitmaps provided as stock images by the AppWizard and has the color red set as the blend color for the right side of the shaft.



6.2.5 Blink period

Description

This property sets the period how fast in ms the cursor of an Edit object will blink.



Available objects

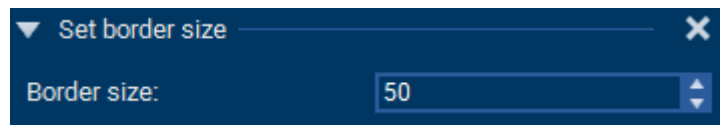
This property can be set for the following objects:

- *Edit* object

6.2.6 Border size

Description

This property sets the border size of an Edit 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

6.2.7 Color and background color

Description

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



Available objects

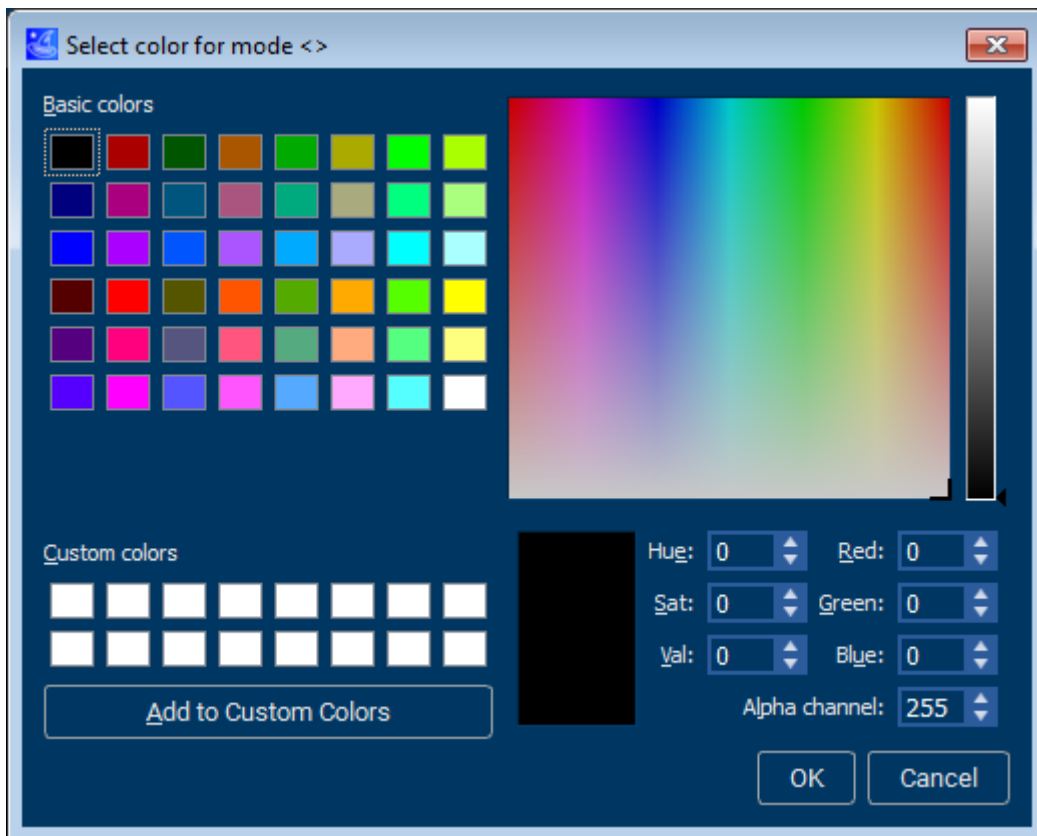
This property can be set for the following objects:

- *Box* object
- *Button* object
- *Edit* 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.



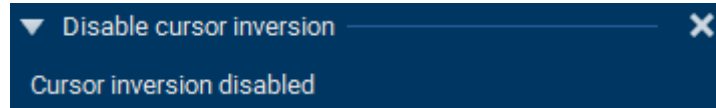
Related topics

- Text color

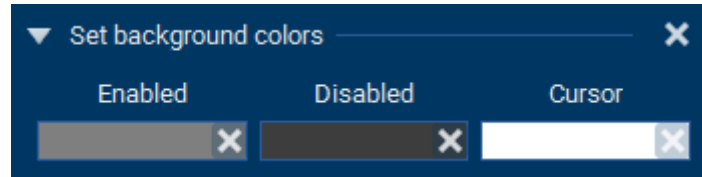
6.2.8 Cursor inversion

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

6.2.9 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

Example

As an example, the mask 00000 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

6.2.10 Fade mode

Description

When activating this setting, while moving the thumb of a Switch object or when the switch animation is performed, the bitmap of the old state will be faded into the new state's bitmap. The default setting is 'disclose mode'. A more detailed explanation about the difference between these two modes is provided in the chapter *Switch* on page 89.



Available objects

This property can be set for the following objects:

- *Switch* object

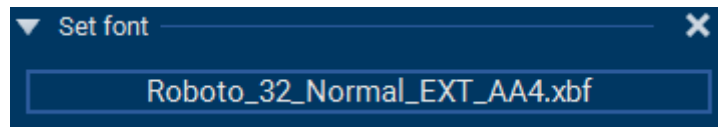
Comparison

Disclose mode	Fade mode
	

6.2.11 Font

Description

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



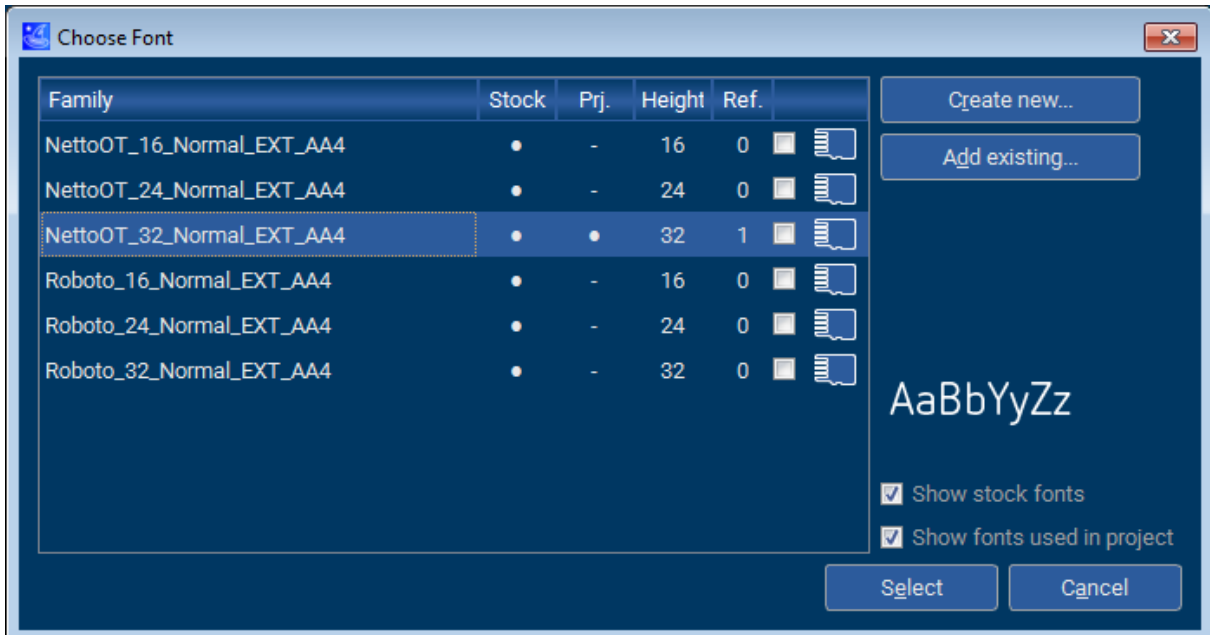
Available objects

This property can be set for the following objects:

- *Button* object
- *Edit* object
- *Switch* object
- *Text* object

Usage

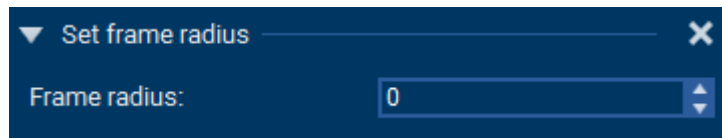
Choosing a font will open a dialog showing all fonts available in the project.



6.2.12 Frame radius

Description

This property sets the radius of the rounded corners of an Edit object in pixels.



Available objects

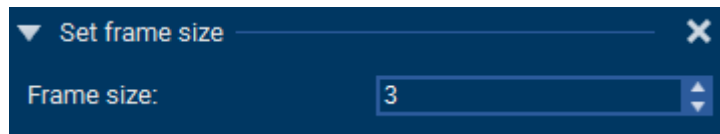
This property can be set for the following objects:

- *Edit* object

6.2.13 Frame size

Description

This property sets the size of the Edit object's frame in pixels.



Available objects

This property can be set for the following objects:

- *Edit* object

6.2.14 Gradient

Description

You can define horizontal and vertical gradients using two colors or more.



Available objects

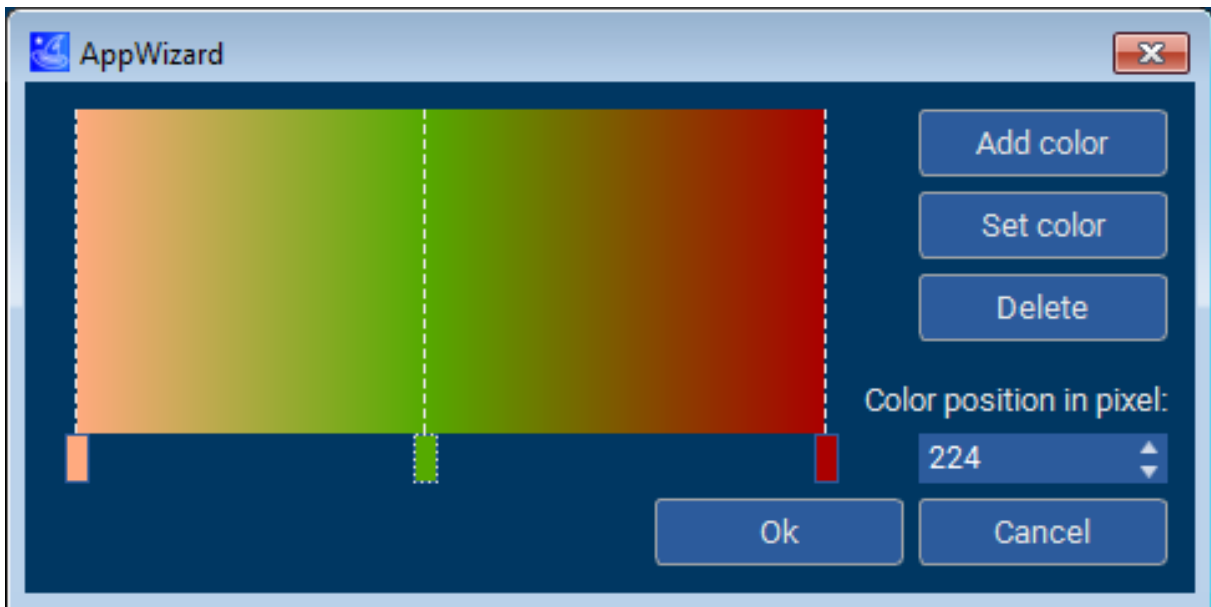
This property can be set for the following objects:

- *Box* object

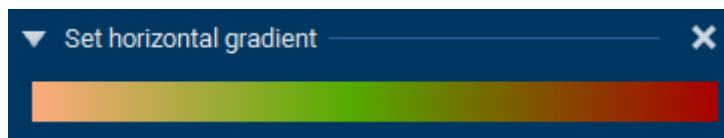
Usage

The user may add colors 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.



Result



6.2.15 Motion

Description

Horizontal and vertical motion allow swiping between different objects.

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.

6.2.16 ID

Description

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



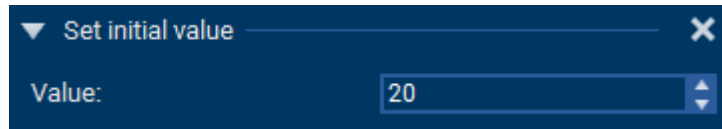
Available objects

This property can be set for all objects.

6.2.17 Initial value

Description

This property sets the initial value of a Rotary object.



Available objects

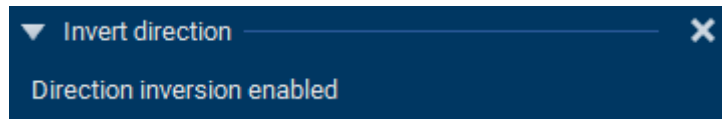
This property can be set for the following objects:

- *Rotary* object

6.2.18 Invert direction

Description

This property inverts the direction of a Slider object, meaning its lowest value and initial position of the thumb will be on the right instead of the left.



Available objects

This property can be set for the following objects:

- *Slider* object

6.2.19 JPEG/GIF/BMP

Description

This property allows to add a JPEG, a GIF or a BMP image to an Image object. Animated GIFs are supported as well.



Available objects

This property can be set for the following objects:

- *Image* object

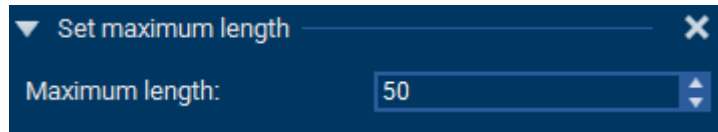
Difference between bitmaps and images

To learn about the difference between bitmaps and images, refer to *Image* on page 85.

6.2.20 Maximum length

Description

This property sets the maximum text length in bytes of an Edit object.



Available objects

This property can be set for the following objects:

- *Edit* object

6.2.21 Offset

Description

This property sets an offset angle for a Rotary object. This will make the object appear rotated by that angle from the beginning. The offset is measured in 10th of degrees ($3600 = 360^\circ$).



Available objects

This property can be set for the following objects:

- *Rotary* object

6.2.22 Period

Description

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



Available objects

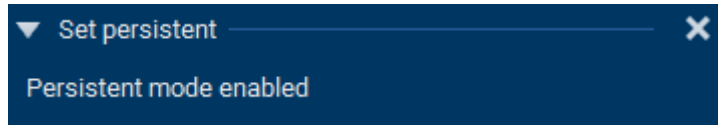
This property can be set for the following objects:

- *Rotary* object
- *Switch* object

6.2.23 Persistent mode

Description

The persistent mode property allows a screen to be persistent, so it does not get deleted during runtime when it is not visible anymore.



Available objects

This property can be set for the following objects:

- *Screen* object

Usage

It is explained when to use this mode in the chapter *Screen* on page 87.

6.2.24 Position and size

Description

Every object has its position and its size.



Available objects

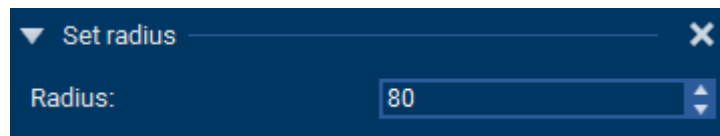
These properties can be set for every object:

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

6.2.25 Radius

Description

This property sets the radius of a 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:

- *Rotary* object

6.2.26 Range

Description

This property allows to define a range for an object.

Edit object

The range property is available for the Edit object when it is set to Decimal mode.

Property	Description
Min	Minimum value eligible to enter.
Max	Maximum value eligible to enter.

Slider object

Property	Description
Min	Minimum value on the slider.
Max	Maximum value on the slider.

Rotary object

Property	Description
Positive	Angle where the rotation stops in counterclockwise direction (10th of degrees).
Negative	Angle where the rotation stops in clockwise direction (10th of degrees).



Available objects

This property can be set for the following objects:

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

6.2.27 Rotate marker

Description

This property, when activated, rotates the bitmap set for the marker.

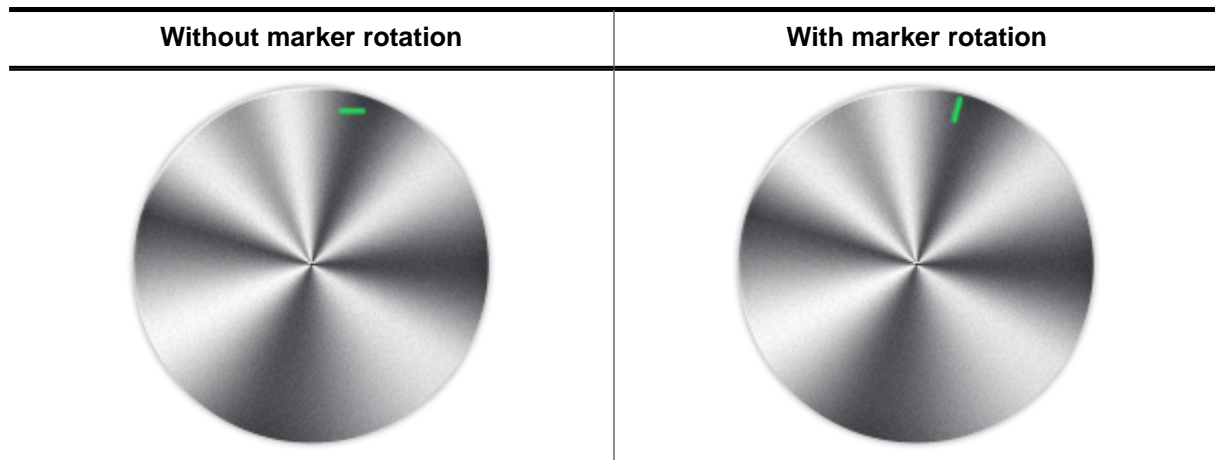


Available objects

This property can be set for the following objects:

- *Rotary* object

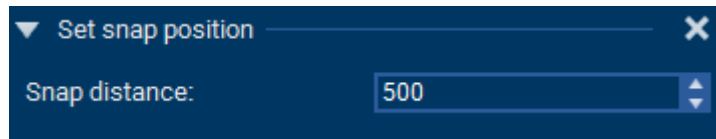
Comparison



6.2.28 Snap position

Description

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°).



Available objects

This property can be set for the following objects:

- *Rotary* object

6.2.29 Span of values

Description

This property defines the range of numbers a Rotary object should return.



▼ Set span of values ×

Min: 220

Max: 50

Available objects

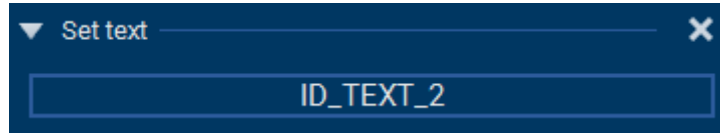
This property can be set for the following objects:

- *Rotary* object

6.2.30 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 34.



Available objects

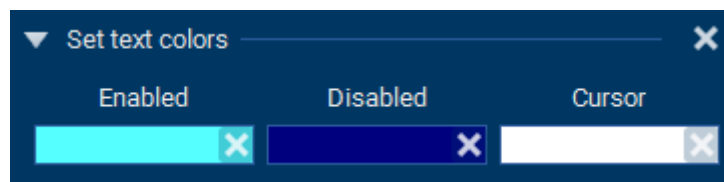
This property can be set for the following objects:

- *Button* object
- *Edit* object
- *Switch* object
- *Text* object

6.2.31 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
- *Edit* object
- *Switch* object
- *Text* object

Related topics

- Color and background color

6.2.32 Tiling

Description

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

Available objects

This property can be set for the following objects:

- *Image* object

Example

See chapter *Image* on page 85 for an example.

6.2.33 Vertical mode

Description

The vertical mode property changes a slider to be vertical. By default it is horizontal.



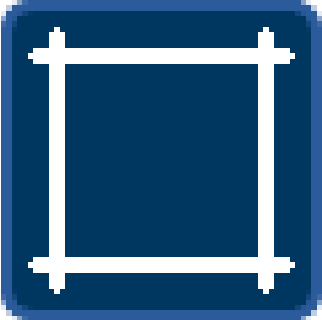
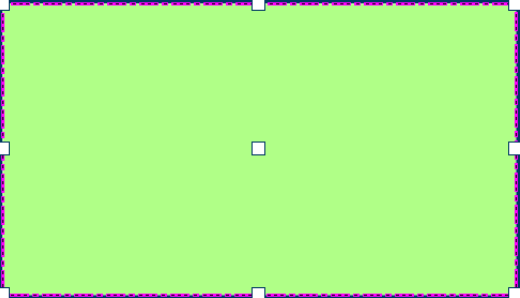
Available objects

This property can be set for the following objects:

- *Slider* object

6.3 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.

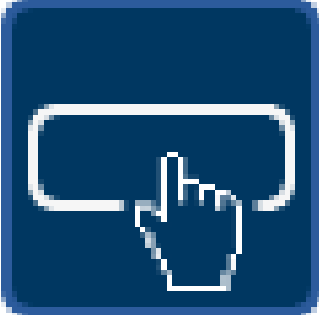

Properties

The following properties are object-specific.

- Color
- Gradients

6.4 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
	

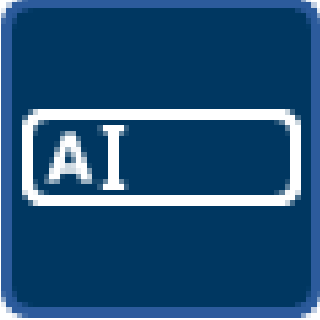

Properties

The following properties are object-specific.

- Text
- Text color
- Background color
- Bitmaps
- Bitmap and text alignment
- Auto repeat
- Font

6.5 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

The following properties are object-specific.

- Text
- Text colors
- Background colors
- Frame color
- Cursor inversion
- Blink period
- Font
- Text alignment
- Frame radius
- Frame size
- Border size
- Maximum length

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 57.

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 74.



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.

6.6 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

The following properties are object-specific.

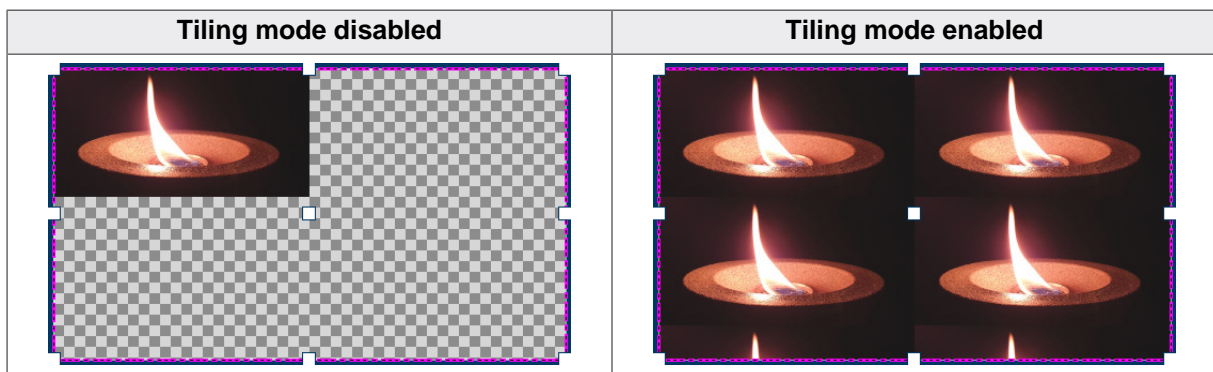
- Bitmap
- JPEG/GIF/BMP
- Tiling

Difference between bitmaps and images

In contrast to bitmaps, JPEG, GIF and BMP images are always displayed native. 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.

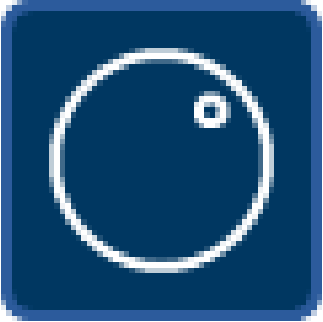
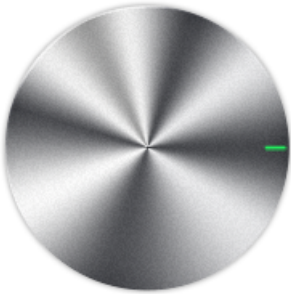


GIF support

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

6.7 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
	


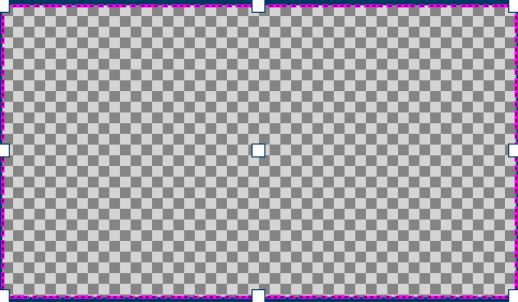
Properties

The following properties are object-specific.

- Bitmaps
- Initial value
- Range
- Span of values
- Offset
- Radius
- Rotate marker
- Period
- Snap position

6.8 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

The following properties are object-specific.



- Motion
- Persistent mode

Persistent mode

It makes sense to use this mode, when the widgets in a screen are showing values which should not get deleted.

6.9 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

The following properties are object-specific.

- Vertical mode
- Invert direction
- Bitmaps
- Blend colors
- Range


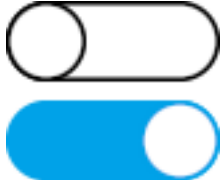
Blend colors

The blend colors setting makes it possible to choose a color for the left and/or right side of the shaft to be blended into the corresponding bitmap. A deeper explanation can be found in *Blend colors* on page 52.



6.10 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

The following properties are object-specific.

- Bitmaps
- Left text
- Right text
- Text colors
- Font
- Period
- Fade mode

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.



6.11 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
	Sample

Properties

The following properties are object-specific.

- Text
- Text color
- Background color
- Decimal mode
- Text alignment
- Font

Decimal mode


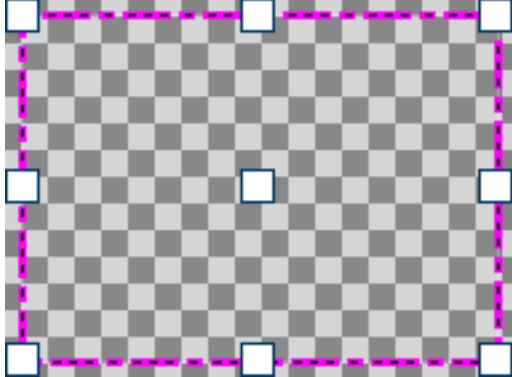
Just as decimal mode for the Edit object, with this setting the Text object is only eligible of holding digits instead of characters. For this mode, a mask of zeros has to be specified which determines how many digits are shown by the object. More details about the usage of the mask is explained under *Decimal mode* on page 57.

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 74.

00123

6.12 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

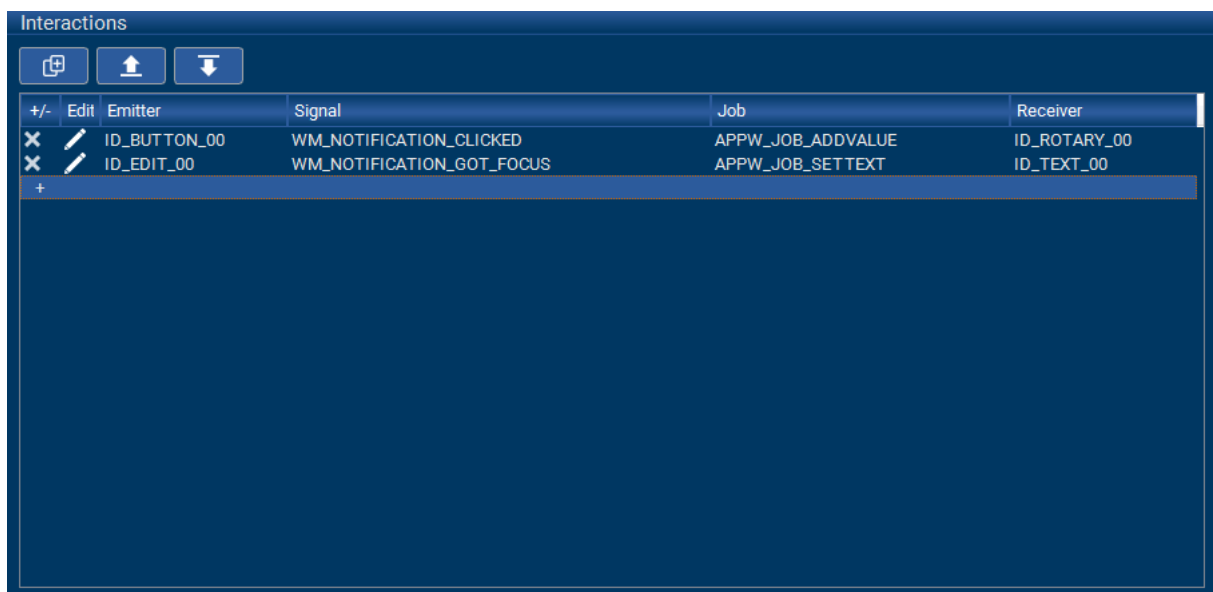
The following properties are object-specific.

- Motion

Chapter 7

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.



The screenshot shows the 'Interactions' window in AppWizard. It features a table with columns for '+/-', 'Edit', 'Emitter', 'Signal', 'Job', and 'Receiver'. There are two rows of interaction rules defined. The first row shows a button click event (ID_BUTTON_00) triggering the APPW_JOB_ADDVALUE job on the ID_ROTARY_00 receiver. The second row shows an edit event (ID_EDIT_00) triggering the APPW_JOB_SET TEXT job on the ID_TEXT_00 receiver. The table has a '+' sign at the bottom left, indicating more interactions can be added.

+/-	Edit	Emitter	Signal	Job	Receiver
X	/	ID_BUTTON_00	WM_NOTIFICATION_CLICKED	APPW_JOB_ADDVALUE	ID_ROTARY_00
X	/	ID_EDIT_00	WM_NOTIFICATION_GOT_FOCUS	APPW_JOB_SET TEXT	ID_TEXT_00

7.1 Introduction

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

1 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.

2 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 95.

3 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 APPW_JOB_SETTEXT to set the text of an Edit object.

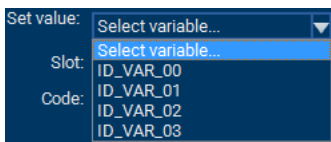
For a list of all available jobs, see the chapter *List of jobs* on page 109.

4 Select the receiver

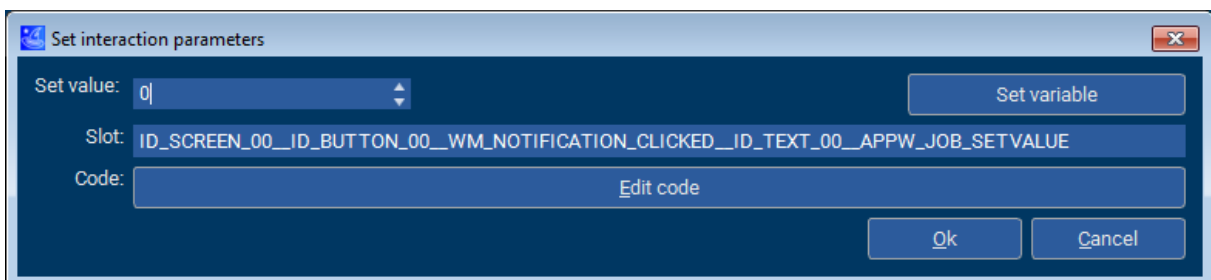
The last step is to select is a receiver for the interaction. The **receiver** specifies the ID of the widget or variable the job will be executed for. For example, if the job is APPW_JOB_SETTEXT, the receiver has to be an Edit object, whose text will then be set.

5 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 APPW_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.

⑥ 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 133.

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 135.

7.2 List of signals

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

Signal	Description
AppWizard related	
APPW_NOTIFICATION_ANIMCOORD	Emitted by an object that has been animated.
APPW_NOTIFICATION_ANIMEND	Emitted when an animation has ended.
APPW_NOTIFICATION_ANIMSTART	Emitted when an animation has started.
APPW_NOTIFICATION_CREATE	Emitted when an object was created.
APPW_NOTIFICATION_DELETE	Emitted when an object was deleted.
APPW_NOTIFICATION_INITDIALOG	Emitted right after the application has started.
APPW_NOTIFICATION_MOTION	Emitted when a user moves a screen by dragging it.
Window Manager related	
WM_NOTIFICATION_CLICKED	When the user clicks on an object.
WM_NOTIFICATION_GOTFOCUS	When an object gets the focus.
WM_NOTIFICATION_LOSTFOCUS	When an object lost its focus.
WM_NOTIFICATION_MOTIONSTOPPED	When the motion of a Rotary object has stopped.
WM_NOTIFICATION_RELEASED	Once a click on an object has been released.
WM_NOTIFICATION_VALUECHANGED	If the value of an object has changed.

7.2.1 APPW_NOTIFICATION_ANIMCOORD

Description

This signal gets emitted by an object that the job `APPW_JOB_ANIMCOORD` was executed on. See `APPW_JOB_CASCADECOORD` on page 115 for a detailed example.

Emitting objects

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

7.2.2 APPW_NOTIFICATION_ANIMEND

Description

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

Emitting objects

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

7.2.3 APPW_NOTIFICATION_ANIMSTART

Description

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

Emitting objects

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

7.2.4 APPW_NOTIFICATION_CREATE

Description

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

Emitting objects

- *Screen* object

7.2.5 APPW_NOTIFICATION_DELETE

Description

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

Emitting objects

- *Screen* object

7.2.6 APPW_NOTIFICATION_INITDIALOG

Description

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

Emitting objects

- *Screen* object

7.2.7 APPW_NOTIFICATION_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

7.2.8 WM_NOTIFICATION_CLICKED

Description

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

Emitting objects

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

7.2.9 WM_NOTIFICATION_GOTFOCUS

Description

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

Emitting objects

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

7.2.10 WM_NOTIFICATION_LOSTFOCUS

Description

This signal gets emitted when an object lost its focus.

Emitting objects

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

7.2.11 WM_NOTIFICATION_MOTIONSTOPPED

Description

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

Emitting objects

- *Rotary* object

7.2.12 WM_NOTIFICATION_RELEASED

Description

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

Emitting objects

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

7.2.13 WM_NOTIFICATION_VALUECHANGED

Description

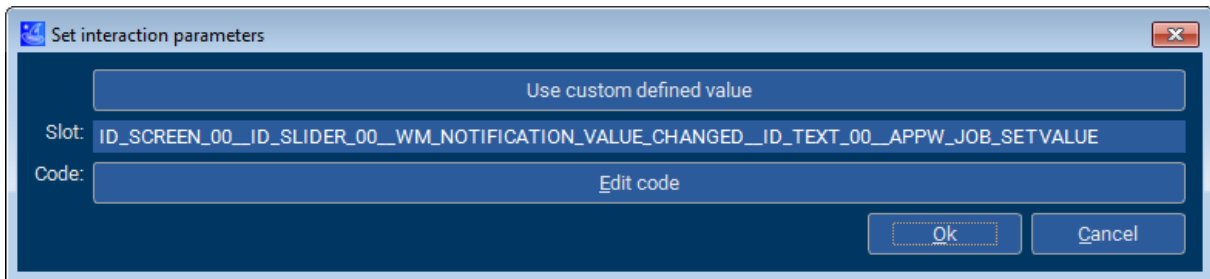
This signal gets emitted when the value of an object has changed.

Emitting objects

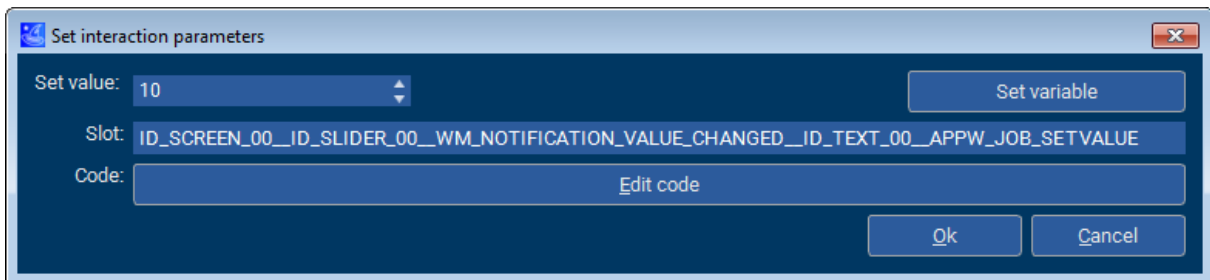
- *Button* object
- *Edit* object
- *Rotary* object
- *Slider* object
- *Switch* object
- *Text* object
- *Variables*

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 `APPW_JOB_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.



7.3 List of jobs

Job	Description
APPW_JOB_ADDVALUE	Adds a given increment to the given object.
APPW_JOB_ANIMCOORD	Animates a coordinate from the current value to the given value.
APPW_JOB_ANIMVALUE	Animates a value from the current value to the given value.
APPW_JOB_ANIMRANGE	Animates a value using the given range.
APPW_JOB_CASCADECOORD	Adds a coordinate animation to an existing animation.
APPW_JOB_CLEAR	Clears the state of the given object.
APPW_JOB_SET	Sets the state of the given object.
APPW_JOB_SETBKCOLOR	Sets the background color of the given object.
APPW_JOB_SETCOLOR	Sets the color of the given object.
APPW_JOB_SETCOORD	Sets a coordinate.
APPW_JOB_SETENABLE	Enables the given object.
APPW_JOB_SETLANG	Sets the language index of an object.
APPW_JOB_SETSIZE	Sets the size of the given object.
APPW_JOB_SETTEXT	Sets the text of the given object.
APPW_JOB_SETVALUE	Sets a value.
APPW_JOB_SETVIS	Makes the given object visible.
APPW_JOB_SHIFTSCREEN	Shifts into the given screen using the given method.
APPW_JOB_SHOWSCREEN	Makes the given screen visible.
APPW_JOB_SWAPSCREEN	Swaps the screen to the given screen.
APPW_JOB_TOGGLE	Toggles the state of the given object.
NULL	Used for only executing custom user code.

7.3.1 APPW_JOB_ADDVALUE

Description

Adds a given increment to the given object.

Receiving objects

- *Text* object
- *Rotary* object

Interaction parameters of dialog

Parameter	Description
<code>Value</code>	Value to be added.

Job-specific parameters passed to slot-routine

Parameter	Description
<code>aPara[0].v</code>	Value to be added.

7.3.2 APPW_JOB_ANIMCOORD

Description

Animates an object from its current position to the given position.

Receiving objects

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

Interaction parameters of dialog

Parameter	Description
Value	Coordinate the object should be moved to.
Coordinate	Axis to the coordinate the object 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.

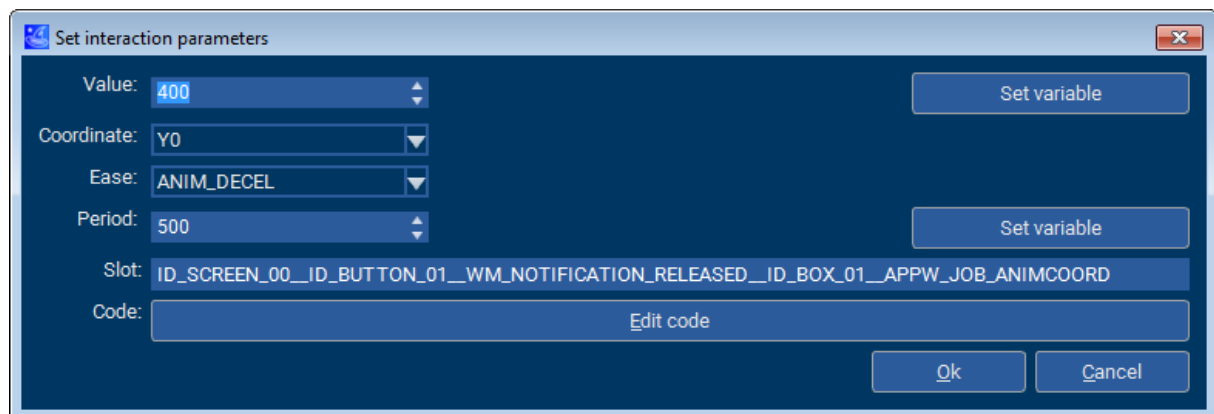
Job-specific parameters passed to slot-routine

Parameter	Description
aPara[0].v	End value.
aPara[1].v	Index of coordinate. See <i>Dispose indexes</i> on page 112 for a list of legal values.
aPara[2].pFunc	Pointer to ease function.
aPara[3].v	Animation period.

Additional information

Similar to emWin animations, additional animation items can be added to that animation. See *APPW_JOB_CASCADECOORD* on page 115 to learn how to do that.

Example



7.3.2.1 Dispose indexes

Description

These indexes are used by jobs that animate coordinates. Its value describes which axis is used for the animation. Jobs that make use of these indexes are *APPW_JOB_ANIMCOORD*, *APPW_JOB_CASCADECOORD* and *APPW_JOB_SETCOORD*.

Definition

```
#define DISPOSE_INDEX_X0    0
#define DISPOSE_INDEX_Y0    1
#define DISPOSE_INDEX_X1    2
#define DISPOSE_INDEX_Y1    3
```

Symbols

Definition	Description
DISPOSE_INDEX_X0	Animate towards negative X-axis.
DISPOSE_INDEX_Y0	Animate towards negative Y-axis.
DISPOSE_INDEX_X1	Animate towards positive X-axis.
DISPOSE_INDEX_Y1	Animate towards positive Y-axis.

7.3.3 APPW_JOB_ANIMVALUE

Description

Animates a value from the current value to the given value.

Receiving objects

- *Text* object
- *Rotary* object

Interaction parameters of dialog

Parameter	Description
<code>Value</code>	Value the object should have when the animation is finished.
<code>Ease</code>	Animation style to be used. See the chapter 'Animations' in the emWin manual for reference.
<code>Period</code>	Period in ms how long the animation will last.

Job-specific parameters passed to slot-routine

Parameter	Description
<code>aPara[0].v</code>	End value.
<code>aPara[1].pFunc</code>	Pointer to ease function.
<code>aPara[2].v</code>	Animation period.

7.3.4 APPW_JOB_ANIMRANGE

Description

Animates a value using the given range.

Receiving objects

- *Edit* object
- *Rotary* object
- *Text* object

Interaction parameters of dialog

Parameter	Description
Start	Start value.
End	End value.
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.

Job-specific parameters passed to slot-routine

Parameter	Description
aPara[0].v	Start value.
aPara[1].v	End value.
aPara[2].pFunc	Pointer to ease function.
aPara[3].v	Animation period.

7.3.5 APPW_JOB_CASCADECOORD

Description

Adds a coordinate animation to an existing animation, similar how animation items can be added to an existing animation in emWin.

This job is paired to the signal `APPW_NOTIFICATION_ANIMCOORD`, that means it can only be executed when reacting on `APPW_NOTIFICATION_ANIMCOORD`. Therefore a previously added animation is required. This is done by setting up an interaction job `APPW_JOB_ANIMCOORD`.

The emitter specified for this interaction is the object that receives the `APPW_JOB_ANIMCOORD` job. In the example below, `ID_BOX_01` is the receiver of the job `APPW_JOB_ANIMCOORD`, therefore the same box object is emitter of the signal `APPW_NOTIFICATION_ANIMCOORD`.

+/-	Edit	Emitter	Signal	Job	Receiver
X	/	ID_BUTTON_01	WM_NOTIFICATION_RELEASED	APPW_JOB_ANIMCOORD	ID_BOX_01
X	/	ID_BOX_01	APPW_NOTIFICATION_ANIMCOORD	APPW_JOB_CASCADECOORD	ID_BOX_02

Receiving objects

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

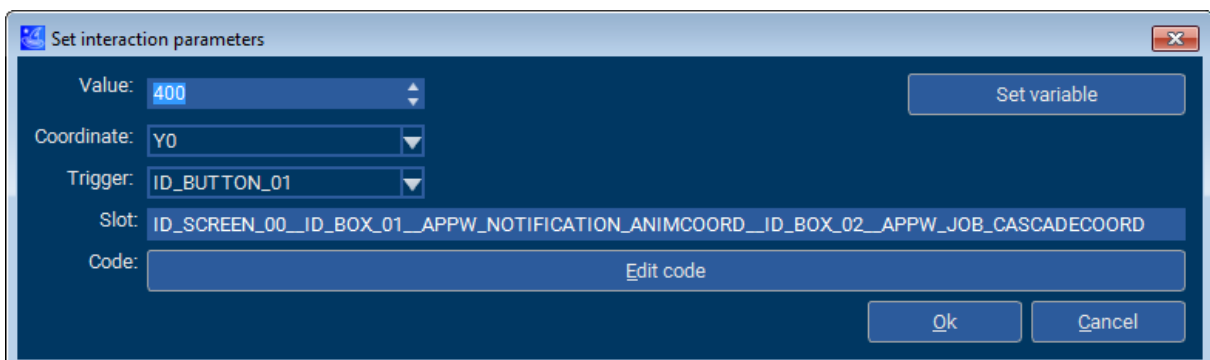
Interaction parameters of dialog

Parameter	Description
<code>Value</code>	Coordinate the object should be moved to.
<code>Coordinate</code>	Axis to the coordinate the object should be moved to.
<code>Trigger</code>	ID of the object that starts the animation, or executes the job <code>APPW_JOB_ANIMCOORD</code> , respectively.

Job-specific parameters passed to slot-routine

Parameter	Description
<code>aPara[0].v</code>	End value.
<code>aPara[1].v</code>	Index of coordinate. See <i>Dispose indexes</i> on page 112 for a list of legal values.
<code>aPara[2].v</code>	Emitter Id of trigger.

Example



7.3.6 APPW_JOB_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

7.3.7 APPW_JOB_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

7.3.8 APPW_JOB_SETBKCOLOR

Description

Sets the background color of the given object.

Receiving objects

- *Button* object

Interaction parameters of dialog

Parameter	Description
<code>Background color</code>	New background color to be used.

Job-specific parameters passed to slot-routine

Parameter	Description
<code>aPara[0].v</code>	Background color to be used.

7.3.9 APPW_JOB_SETCOLOR

Description

Sets the color of the given object.

Receiving objects

- *Box* object
- *Button* object

Interaction parameters of dialog

Parameter	Description
<code>Color</code>	New color to be used.

Job-specific parameters passed to slot-routine

Parameter	Description
<code>aPara[0].v</code>	Color to be used.

7.3.10 APPW_JOB_SETCOORD

Description

Sets a coordinate of an object.

Receiving objects

- *Box* object
- *Button* object
- *Edit* object
- *Image* 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 <i>Dispose indexes</i> on page 112 for a list of legal values.

7.3.11 APPW_JOB_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
<code>Enable state</code>	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
<code>aPara[0].v</code>	Enable state. 1 = on, 0 = off.

7.3.12 APPW_JOB_SETLANG

Description

Sets the language of the application to the given index.

Receiving objects

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

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
<code>aPara[0].v</code>	Index of language.

7.3.13 APPW_JOB_SETSIZE

Description

Sets the size of the given object.

Receiving objects

- *Box* object
- *Button* object
- *Edit* object
- *Image* 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.

7.3.14 APPW_JOB_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 <code>aPara[0].p = NULL</code> .
aPara[0].p	Handle. Only if <code>aPara[0].v < 0</code> .

7.3.15 APPW_JOB_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
- *Rotary* object
- *Slider* object
- *Switch* object
- *Text* object
- *Variables*

Interaction parameters of dialog

Parameter	Description
<code>Value</code>	New value or text to be set to the object.

Job-specific parameters passed to slot-routine

Parameter	Description
<code>aPara[0].v</code>	Value to be set.

Additional information

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

7.3.16 APPW_JOB_SETVIS

Description

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

Receiving objects

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

Interaction parameters of dialog

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

Job-specific parameters passed to slot-routine

Parameter	Description
<code>aPara[0].v</code>	Visibility flag. 1 = on, 0 = off.

7.3.17 APPW_JOB_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	<i>Not used for this job.</i>

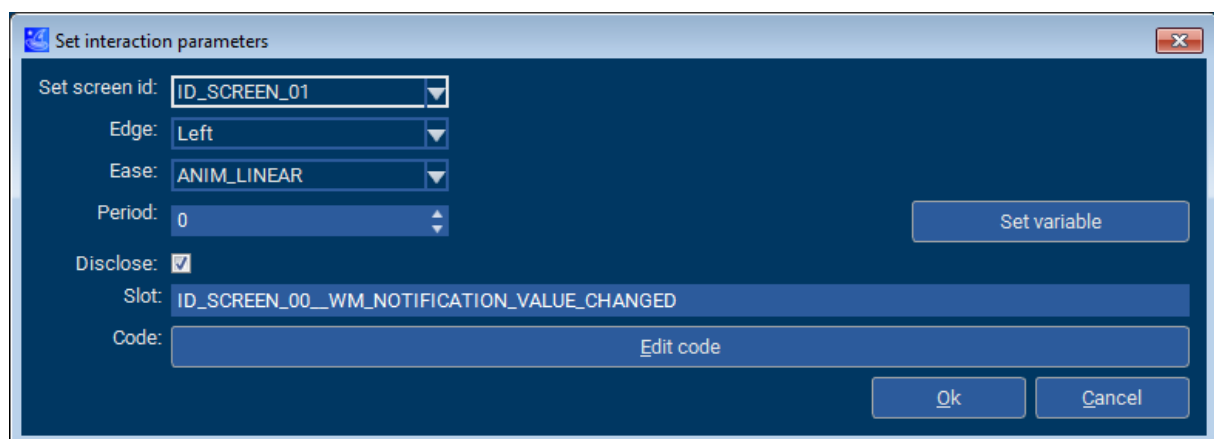
Job-specific parameters passed to slot-routine

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

Additional information

Note that screens that are not being marked as persistent (see *Persistent mode* on page 71) will be deleted after they have been faded out.

Example



7.3.18 APPW_JOB_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 71) will be deleted after they have been faded out.

7.3.19 APPW_JOB_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.

7.3.20 APPW_JOB_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

7.3.21 NULL

Description

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

Receiving objects

- *none*

Chapter 8

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.

8.1 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
<code>pAction</code>	Pointer to an <code>APPW_ACTION_ITEM</code> structure.
<code>hScreen</code>	Handle of the screen.
<code>pMsg</code>	Pointer to a <code>WM_MESSAGE</code> structure. <code>pMsg->hWin</code> is the handle to the receiver while <code>pMsg->hWinSrc</code> is the handle to the emitter.
<code>pResult</code>	Pointer to an <code>int</code> 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 109.

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.

8.1.1 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
<code>IdSrc</code>	Id of the emitter.
<code>NCode</code>	Id of the signal.
<code>IdDst</code>	Id of the receiver.
<code>IdJob</code>	Id of the job.
<code>pfSlot</code>	Function pointer to a slot routine. Prototype explained under Slot routines.
<code>aPara</code>	Optional job specific parameters. The parameter for each job is explained in the <i>List of jobs</i> on page 109.

8.1.2 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. Any non-generated routines from the user **must** be added to this file.

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

8.2 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;
    }
}

```


8.3 Fonts

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

Note

The chapter *Font management* on page 41 explains how fonts can be created using AppWizard.

8.3.1 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 133.

8.3.2 Font API

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

Routine	Description
<code>APPW_GetFont()</code>	Fills a font structure using the addressed setup structure.

8.3.2.1 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	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 137 for an example.

8.4 Variables

8.4.1 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()`.

8.4.2 Variables API

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

Routine	Description
<code>APPW_GetVarData()</code>	Returns the value of a variable.
<code>APPW_SetVarData()</code>	Sets the value of a variable.

8.4.2.1 APPW_GetVarData()

Description

Returns the value of a variable.

Prototype

```
U32 APPW_GetVarData(U16 Id,  
                   int * pError);
```

Parameters

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

Return value

Data value of the specified variable.

8.4.2.2 APPW_SetVarData()

Description

Sets the value of a variable.

Prototype

```
int APPW_SetVarData(U16 Id,  
                   U32 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 9

Board support packages (BSPs)

As already mentioned in the chapter *Requirements* on page 15 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.

9.1 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?

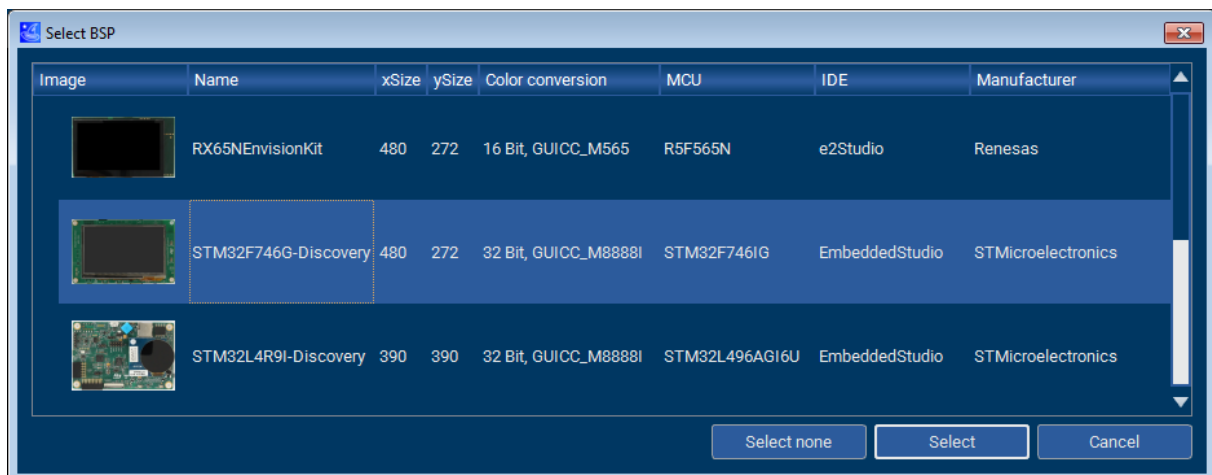
A file system is only required if resources should be outsourced to external media, for which we use the emFile system.

9.1.1 Example

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

9.1.1.1 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.

9.1.1.2 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`.

9.1.1.3 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`.

 Start_STM32F746_ST_STM32F746G_Discovery.emProject	19-12-02 10:59	EMPROJECT File	7 KB
---	----------------	----------------	------

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

9.1.1.4 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`.

9.2 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.

9.2.1 Example

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

9.2.1.1 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:

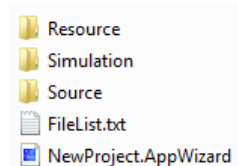
```
xSize:          240
ySize:          320
Color conversion:  GUICC_M8888I
Name:           Does not matter.
BSP:            None
```

9.2.1.2 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.

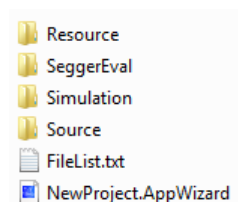
9.2.1.3 Step 3: Export & Save

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



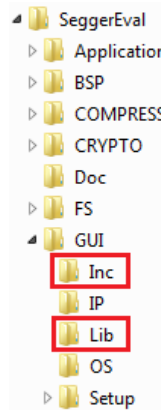
9.2.1.4 Step 4: Copy evaluation software package into project folder

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`.



9.2.1.5 Step 5: Exchange libraries

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

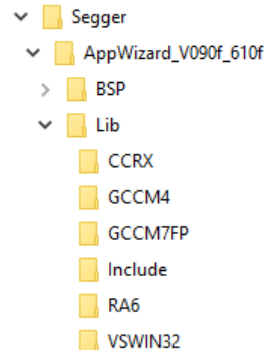


Note

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 \geq 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).

9.2.1.6 Step 6: Add file access routines

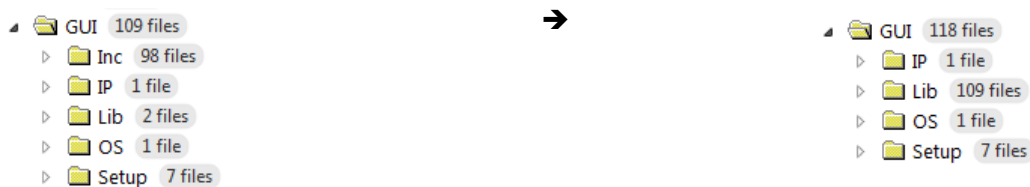
The next step is to add the file access routines to the project folder. Depending on if you want to use a file system, you have to copy one of the files located in `\Sample` into the data directory. The directory contains two files:

`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`.

9.2.1.7 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.

9.2.1.8 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`.



9.2.1.9 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...

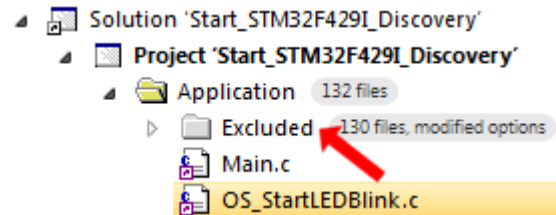
Preprocessor	
• Ignore Includes	No
• Preprocessor Definitions	STM32F429xx;_STM32F4xx_FAMILY;_STM32F429_SUBFAMILY;HSE_VA
• Preprocessor Undefinitions	
• System Include Directories	
• Undefine All Preprocessor Definitions	No
• User Include Directories	\$(ProjectDir)/../../../../Application/GUI/SEGGERDEMO/Src;\$(ProjectD...

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

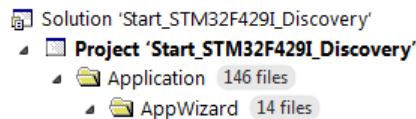
\$(ProjectDir)/../../../../GUI/Inc → \$(ProjectDir)/../../../../GUI/Lib

9.2.1.10 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.



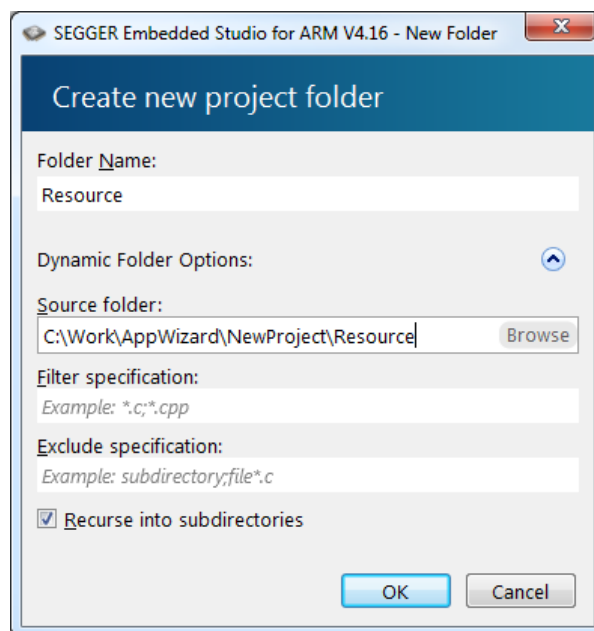
Right-click on the "Application" folder and select "New Folder", name the new folder *AppWizard*.



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.



9.2.1.11 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.

9.3 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.

9.3.1 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`.

9.3.2 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`.

9.3.3 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.

9.3.4 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

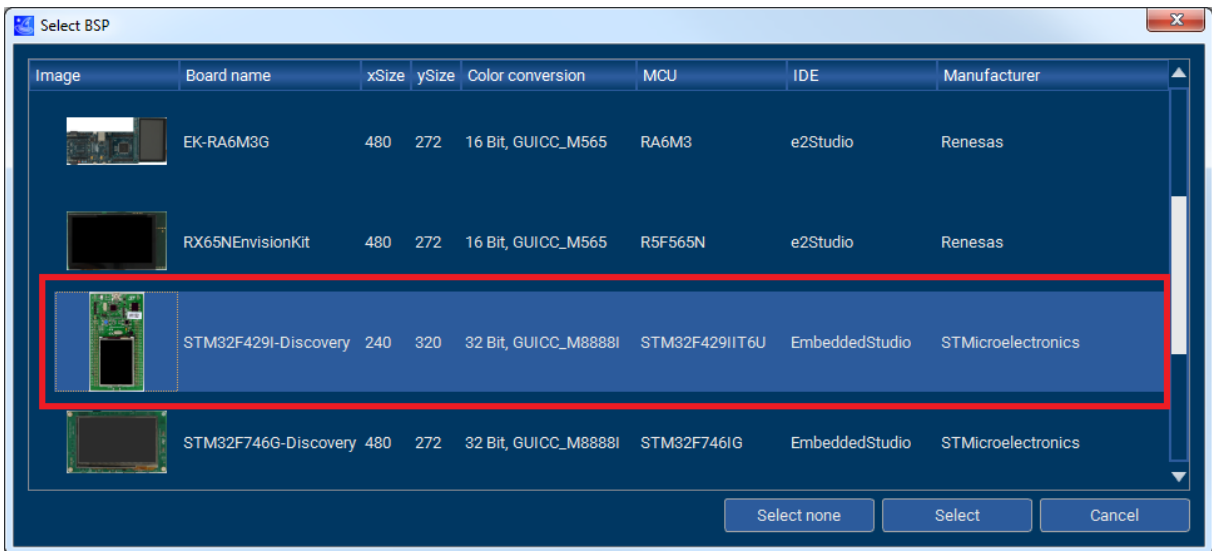
Take one of the already existing `*.BSPInfo` files and copy it into the BSP folder. The file name must be of the format `<Name of BSP>.BSPInfo`. Open the file in a text editor and add the required information:

```
<!DOCTYPE emWin_AppWizard_BSP_Info>
<BSP>
  xSizeDisplay=240
  ySizeDisplay=320
  ColorConv=GUICC_M8888I
  BoardName=STM32F429I-Discovery
  IDE=Embedded Studio
  MCU=STM32F429IIT6U
  Manufacturer=STMicroelectronics
</BSP>
```

Save the file as `STM32F429I_Disco_ES\STM32F429I_Disco_ES.BSPInfo`.

9.3.5 Step 5: Import the BSP into AppWizard

To import the BSP into AppWizard, select **File → Import BSP...** This process can take a minute or longer. After that, a new BSP should be available within AppWizard:



Chapter 10

Glossary

BSP

Board support package.

embOS

[Embedded real-time operating system.](#)

emFile

[Embedded file system.](#)

emWin

[Embedded graphics library.](#)

MCU

Microcontroller unit.

RAM

Random access memory.

ROM

Read-only memory.

SES

[SEGGER Embedded Studio.](#)

WYSIWYG

What You See Is What You Get.