



# Reference Manual

IMACS GmbH  
Mittelfeldstrasse 25  
D – 70806 Kornwestheim  
[www.radcase.com](http://www.radcase.com)

[support@radcase.com](mailto:support@radcase.com)  
Tel.: +49 (0) 7154 80 83 - 0

IMACS GmbH reserves the right to make changes without further notice to any products herein. IMACS GmbH makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does IMACS GmbH assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters which may be provided in IMACS GmbH data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals” must be validated for each customer application by customer’s technical experts. IMACS GmbH does not convey any license under its patent rights nor the rights of others.

Copyright © IMACS GmbH 2021. All rights reserved.  
Reproduction, in part or whole, without the prior written consent of IMACS GmbH is prohibited.

## 1 Contents

1	Contents.....	2
2	3rd party libraries.....	9
2.1	Qt.....	9
2.1.1	Written offer for LGPL source code.....	9
2.2	Qt/MFC Migration Framework.....	9
3	About This Document.....	10
3.1	How To Read This Document.....	10
3.2	Conventions.....	10
3.3	Acronyms.....	11
3.4	Glossary.....	12
4	Best Practices.....	13
4.1	Naming Conventions.....	13
4.2	Recommended Project File Structure.....	13
4.3	Recommended Project MODUL structure.....	13
4.4	File And Path Access.....	14
4.5	Calling radCASE Functions From External C Files.....	14
4.5.1	Global Wrapper Function.....	14
4.5.2	Callback Function.....	15
4.6	Simulating The Process.....	15
4.7	Being Aware Of Task-Scheduling.....	15
4.8	Using Global Code.....	16
4.9	Handling Timers.....	16
4.10	Realizing data structures in radCASE.....	17
5	Language Reference.....	18
5.1	Overview.....	18
5.1.1	General RC-Items.....	18
5.1.2	Language Tree.....	18
5.1.3	Surface Items.....	19
5.1.4	Actions.....	20
5.1.5	Sequence Diagram Items.....	21
5.1.6	State Machine Items.....	21
5.1.7	Activity Chart Items.....	22
5.1.8	Signal Chart Items.....	22
5.1.9	Usecase-Diagram Items.....	23
5.2	COMMENT.....	23
5.3	HYPERLINK.....	23
5.4	#IF, #IFN, #ELSE, #ENDIF.....	23
5.5	Project.....	23
5.6	TEXTDEF.....	24
5.7	SYSTEMDEF.....	24

5.8	SETTINGS .....	25
5.8.1	Systemcode .....	26
5.8.2	Timing .....	26
5.8.3	Ctr.....	28
5.8.4	Desktop.....	32
5.9	EMB_HMI .....	35
5.10	RADMON .....	37
5.11	Permissions.....	37
5.12	Passwords.....	37
5.13	DOCTITLE .....	37
5.14	VERSION .....	37
5.15	DEFINE .....	37
5.16	REPLACE .....	38
5.17	TYPEDEF.....	38
5.18	EBIN.....	38
5.19	EB_ENTRY .....	39
5.20	ENUM .....	39
5.21	ESTR .....	40
5.22	EDAT .....	40
5.23	ETIM .....	40
5.24	MODULDEF .....	41
5.25	MODUL .....	41
5.26	SUBMODUL.....	42
5.27	ELEMENT .....	43
5.27.1	Assign type .....	44
5.27.2	Com Type .....	45
5.27.3	Assign string .....	46
5.27.4	Format string.....	50
5.27.5	XCOM-type .....	51
5.28	SURFACE_XXX .....	51
5.29	DOCTAB .....	52
5.30	DT_ENTRY .....	52
5.31	FULL .....	52
5.32	ICON.....	53
5.33	TEXT.....	53
5.34	IPIC.....	54
5.35	MENU .....	54
5.36	AMENU .....	55
5.37	EDIT.....	55
5.38	COLUMNBREAK.....	57
5.39	ELEM .....	57
5.40	MElem.....	59
5.40.1	MElem Formatstring.....	59
5.41	DISPLAY .....	60
5.42	AICON.....	60
5.43	SET .....	61

5.44	RES.....	61
5.45	INV .....	61
5.46	PUT.....	61
5.47	INC.....	62
5.48	DEC .....	62
5.49	AEDIT .....	62
5.50	COPY.....	62
5.51	GOTOSURFACE.....	63
5.52	CALLSURFACE .....	63
5.53	RETURN (Sur) .....	63
5.54	OPEN.....	63
5.55	CLOSE.....	63
5.56	CFUNC .....	63
5.56.1	C function.....	64
5.57	USERFUNC .....	65
5.58	PROCEDURE .....	66
5.59	PASSWORD .....	66
5.60	PSWD_CLR .....	66
5.61	ASKOK.....	66
5.62	AKEY .....	66
5.63	AKEYGLOBAL .....	67
5.64	AENTER .....	67
5.65	APERM .....	67
5.66	AEXIT.....	67
5.67	ActionCond.....	67
5.68	TouchKey.....	68
5.69	IF, ENDIF, ELSE .....	69
5.70	SETPOS .....	69
5.71	RECT .....	70
5.72	CIRC .....	70
5.73	LINE.....	70
5.74	FOLDER .....	71
5.75	CARD.....	71
5.76	FILL.....	71
5.77	METHODS .....	71
5.78	PROC.....	72
5.79	SEQUENCE DIAGRAM.....	72
5.80	MESSAGE .....	73
5.81	IF, ELSE IF, ELSE, END IF .....	73
5.82	CODE (Seq).....	74
5.83	LOOP, END LOOP .....	74
5.84	RETURN (Seq) .....	74
5.85	REFERENCE .....	74
5.86	STATECHARTS.....	74
5.87	MACHINE.....	75
5.88	STATE .....	75
5.89	ENTER.....	76

5.90	EXIT .....	76
5.91	UNITSTATE .....	76
5.92	TRANSITION .....	77
5.93	DURING .....	78
5.94	CHOICE .....	78
5.95	ENDSTATE .....	78
5.96	SUBMACHINE .....	78
5.97	ENTRY POINT .....	79
5.98	EXIT POINT .....	79
5.99	ACTIVITYCHARTS .....	79
5.100	ACTIVITY .....	79
5.101	START .....	80
5.102	END .....	80
5.103	ACTION .....	80
5.104	BRANCH .....	81
5.105	LABEL .....	81
5.106	CONTROLFLOW .....	81
5.107	CODE (Act) .....	82
5.108	SIGNAL_CHART .....	82
5.109	SignalIcon .....	82
5.110	Connection .....	83
5.111	SIGNAL_ICON .....	83
5.112	PORT .....	85
5.113	ARTEFACT .....	85
5.114	RELATION .....	85
5.115	PICTDEF .....	86
5.116	PICT .....	86
5.117	ARC .....	86
5.118	CBITMAP .....	87
5.119	WBITMAP .....	87
5.120	VISUALDEF .....	87
5.121	VISBINICON .....	88
5.122	VIS_LINE .....	88
5.123	VIS_ROT .....	88
5.124	VIS_ROT_XY .....	89
5.125	EntityTab .....	89
5.126	ARTEFACTDEFS .....	89
5.127	ARTEFACTDEF .....	89
5.128	Requirement .....	90
5.129	Usecase-Diagram .....	90
5.130	Actor .....	91
5.131	Association .....	91
5.132	Usecase .....	91
5.133	Scenario .....	92
5.134	ScenarioReference .....	92
5.135	Attribute .....	92
5.136	AttributeReference .....	93
5.137	Sub-Usecase-Diagram .....	93

5.138	EProfiles.....	93
5.139	EProfile .....	93
5.140	EGroups.....	93
5.141	EGroup.....	94
6	Feature Details .....	95
6.1	Project Analysis .....	95
6.1.1	Keeping Track Of Development Status .....	95
6.1.2	Usecase Diagrams.....	95
6.2	Project Hierarchy .....	95
6.2.1	The System MODUL .....	96
6.2.2	Using The EntityTab.....	96
6.2.3	Inheritance .....	97
6.2.4	Dynamic instantiation .....	97
6.2.5	Distributed Systems .....	99
6.2.6	MODUL access.....	104
6.3	Data Modeling .....	105
6.3.1	Data Types.....	106
6.3.2	Changing Of Metadata At Runtime.....	116
6.3.3	Element Usage And Allocation .....	117
6.3.4	Element Arrays .....	121
6.3.5	Data Storage.....	121
6.3.6	radCASE Index (RDI) .....	124
6.3.7	NoValues .....	125
6.3.8	Daylight Saving Time .....	126
6.4	Behavior Modeling.....	126
6.4.1	Signal Diagrams.....	126
6.4.2	Finite State Machines.....	128
6.4.3	Element Access .....	130
6.4.4	Behavior Access .....	135
6.4.5	Text Access .....	138
6.5	HMI.....	139
6.5.1	Surface item support .....	140
6.5.2	Positioning .....	142
6.5.3	Surface Navigation.....	143
6.5.4	Font Handling.....	145
6.5.5	Target HMI .....	153
6.5.6	Text Handling.....	165
6.5.7	Graphic Handling .....	167
6.5.8	Element Visualization.....	169
6.5.9	Action Handling.....	175
6.5.10	Conditional Drawing.....	178
6.6	Text Support.....	178
6.6.1	Text ID .....	179

6.6.2	Text Table .....	179
6.6.3	Short Texts.....	180
6.6.4	Long Texts .....	181
6.6.5	Language switching on the target and in the Project Monitor .....	181
6.6.6	Supported Languages.....	182
6.6.7	Unicode.....	183
6.7	Sound reproduction .....	184
6.8	Code Size Optimization .....	184
6.8.1	Using Defines For Feature Activation/Deactivation.....	184
6.8.2	Optimizing File Size Of Osdl.txt.....	186
6.9	Scheduling.....	188
6.9.1	Tasks .....	188
6.9.2	Processing types.....	189
6.10	Target Compiler Specific Features .....	189
6.10.1	Limiting Code Size Of Generated Files .....	189
6.10.2	Harvard Architecture .....	191
6.11	Project Monitor .....	191
6.11.1	Standalone.....	192
6.11.2	Stimulation Equations .....	192
6.11.3	Virtual Keyboard Support .....	193
6.11.4	Communication .....	194
6.11.5	Permission modell.....	199
6.11.6	Output window .....	199
6.11.7	Sequences.....	200
6.11.8	Dynamic Version Switching.....	200
6.11.9	Resource Consumption Analysis .....	206
6.11.10	System Event Handling.....	207
6.11.11	Simulated Display Functions.....	208
6.11.12	Customizing The Simulation Project.....	209
6.11.13	Help .....	210
6.12	Documentation Generation.....	211
6.12.1	Defining Target HMI For Documentation .....	211
6.12.2	Structure Of Generated Documentation .....	212
6.12.3	Artefact documentation .....	216
6.12.4	Comment Syntax .....	218
6.13	Web visualization .....	220
6.14	Remote HMI on Web (browser) .....	220
6.14.1	How it works .....	220
6.14.2	How to integrate 'remote HMI on web' into a project .....	223
7	Troubleshooting.....	224
7.1	Licensing Issues.....	224
7.1.1	No license file available. The program will be aborted. Please contact your dealer. ....	224
7.1.2	License not sufficient. The program will be aborted. ....	224



7.1.3	License file corrupted .....	225
7.2	Runtime errors.....	225
7.2.1	Runtime Error Messages.....	225
7.2.2	Buffer too small – Enlarge RD_DISPCMD_BUFSIZE .....	230
7.3	Visualizing global C-Variables .....	230
8	Appendix .....	232
8.1	Key Values .....	232
8.1.1	Most Commonly Used Keys .....	232
8.1.2	Other Keys .....	232
8.2	Element Attributes .....	235
8.2.1	ENUM Attributes .....	235
8.2.2	EBIN Attributes .....	236
8.2.3	ESTR Attributes .....	237
8.2.4	ETIM Attributes .....	237
8.2.5	EDAT Attributes .....	237
8.3	Formats Of Text Visualizers.....	238
8.3.1	Formats Of Text Visualizers On SURFACE_VIS .....	238
8.3.2	Formats Of Text Visualizers on SURFACE_CTR .....	239
8.4	HAL drawing wrapper functions .....	240

## 2 3rd party libraries

### 2.1 Qt

**radCASE** uses the Qt libraries in version 5.3.2 (refer to <https://www.qt.io>) The Qt Toolkit is Copyright (C) 2014 Digia Plc and/or its subsidiary(-ies). These libraries are licensed under the GNU LGPL Version 2.1. This license can be found in `<radCASE-Common-Directory>\3rdPartyLicense\Qt\LICENSE.LGPL`. Please note Qt will allow some exceptions to the LGPL. These exceptions can be found in `<radCASE-Common-Directory>\3rdPartyLicense\Qt\LGPL_EXCEPTION.txt`.

According to LGPL license the user has to be able to relink the **radCASE** to modified versions of Qt. Because **radCASE** links to those libraries dynamically, this can be easily done by replacing the DLLs.

#### 2.1.1 Written offer for LGPL source code

On request IMACS GmbH will provide the LGPL source code files via download or CD-ROM for a nominal cost to cover shipping and media charges as allowed under LGPL. Please direct requests to [support@radcase.com](mailto:support@radcase.com). Alternatively you can download the sources directly from Qt: <http://download.qt.io/archive/qt/5.3/5.3.2/single/>

### 2.2 Qt/MFC Migration Framework

**radCASE** uses the Qt/MFC Migration Framework. Copyright (C) 2013 Digia Plc and/or its subsidiary(-ies).

The Qt/MFC Migration Framework is licensed under a BSD license. The license text can be found in `3rdPartyLicense\Qt MFC Migration Framework\BSD-License.txt`

The Qt/MFC Migration Framework is not modified and can be downloaded at <https://github.com/qtproject/qt-solutions/tree/master/qtwindowsmigrate>

### 3 About This Document

This manual is for all application developers and is a complete reference guide to radCASE modeling.

This manual assumes that the user is familiar with basic concepts of Software Engineering, in particular:

- programming in C/C++
- OO programming
- basic skill in UML
- HW/SW embedded architectures

This manual also assumes some familiarity with basic concepts of radCASE which are described in the Guidelines.

#### 3.1 How To Read This Document

This document has the following structure:

In section [Best Practices](#) there are some general instructions and tips on how to build reusable models.

The section [Language Reference](#) is an overview of the radCASE language. Within that section the section [Overview](#) gives a short overview about the hierarchical structure of the language with links to each of the RC-Items. After the overview each of the RC-Items is briefly described and all attributes are listed with a short explanation and if necessary with its syntax. In the short explanations links will guide to in depth explanations of different features. Each RC-Item section will also contain a list of all corresponding child RC-Items, if any.

The section [Feature Details](#) contains the in depth explanations of all the different features.

[Troubleshooting](#) deals with different problems that may arise while working with radCASE.

#### 3.2 Conventions

This document uses the following conventions

Example	Usage
Syntax is: <code>SV=123</code>	radCASE syntax
<code>doSomething() // Comment</code>	C-Code examples
<b>SURFACE_XXX</b>	radCASE language element
refer to <a href="#">Conventions</a>	cross reference in document
<i>module.c</i>	filename or path

Table 1, Typographical conventions




Symbol	Usage
	Used for providing hints and suggestions
	Used for calling attention to specific issues and supply important information
	Used to alert for particular issues to avoid a hard time

Table 2, Symbol conventions

### 3.3 Acronyms

	Refers to
radCASE	Rapid Application Development Computer Aided Software Engineering
HMI	Human Machine Interface
UML	Unified Modeling Language
IO	Input/Output
RTC	Real time clock
EEPROM	Electrically erasable programmable read only memory
PNG	Portable network graphic
RAM	Random Access Memory
CAN	Controller area network (see also <a href="#">Glossary</a> )
HAL	Hardware abstraction layer (see also <a href="#">Glossary</a> )
RDI	radCASE Index (refer to <a href="#">radCASE Index (RDI)</a> )
<#>	Placeholder for a numerical value
<0x#>	Placeholder for a hexadecimal numerical value
<0/1>	Placeholder for a Boolean value (normally meaning 0 disabled/1 enabled)
<\$>	Placeholder for a string

Table 3, Acronyms

### 3.4 Glossary

Refers to	
RC-Item	A radCASE language element
Model	The content of a radCASE project and all connected libraries
HAL	The implementation of the interface between the radCASE library functions and the code for a specific controller hardware
CAN	An asynchronous serial bus system
CANOpen	CANOpen is a communication protocol based on CAN

**Table 4, Glossary**

## 4 Best Practices

### 4.1 Naming Conventions

It is recommended to use the following naming conventions:

- PICT IDs of [PICTs](#) have the format  $P\langle x \rangle \langle Name \rangle \_ \langle Status \rangle$ . The  $\langle x \rangle$  is a “w” for pictures for the **Project Monitor** and “c” for pictures used on the Controller. The Status is optional and can be used to identify for which status a picture is used within a [VISBINICON](#).
- Custom visualizers (refer to [Element Visualization](#)) have the format  $V\langle x \rangle \langle Name \rangle$  where again  $\langle x \rangle$  is “w” for visualizers used in the **Project Monitor** and “c” for visualizers used on the controller.
- [TYPEDEFs](#) start with a “T”:  $T\langle Name \rangle$
- [ELEMENTs](#) containing [Hardware Specific Data](#) begin with the [Assign type](#):  $\langle Assign type \rangle \_ \langle Name \rangle$ .
- [MODULs](#) start with a “M”:  $M\langle Name \rangle$

### 4.2 Recommended Project File Structure

To get a project which is as far as possible hardware independent the project should be separated from the main project file. The main project file should only contain the Settings contained in [SYSTEMDEF](#), an empty [TEXTDEF](#), an [EntityTab](#) and [The System MODUL](#).

The **TEXTDEF** should only have all required languages defined, but contain no texts. The System **MODUL** should also be nearly empty. The main functionality should be in a library file and be included in the main project file. The System then should inherit from a **MODUL** containing all the functionality or contain a [SUBMODUL](#) of that **MODUL**.

Using this structure the functionality can easily be adapted to other target hardware, by simply writing another main project file with settings for that target hardware and the whole functionality does not have to be copied or changed. It is recommend starting all the main project file names with Gen\_ and after that have an identifier to know which hardware is created by that file. E.g. *Gen\_F345.rad*, *Gen\_N17M3.rad*, *Gen\_EA-Board.rad*, *Gen\_Terminal.rad*, etc.

Further it is recommended to use a library directory to store library files only containing cross project functionality. In this library it is also recommended to use a single file containing the cross project business line specific texts and a single file containing all the cross project pictures. For creating the library directory also refer to [File And Path Access](#).

### 4.3 Recommended Project MODUL structure

It is recommended to use [PORTs](#) within a [MODUL](#) to enhance the maintainability. By using **PORTs** the **MODUL** can be divided into functions and **ELEMENTs** used internally and those that are used from other code.

To build reusable **MODULs** it is not recommended to use backward references (refer to [MODUL access](#)). When using backward references the **MODUL** can only be used within a special **MODUL**

structure, making the **MODUL** hardly reusable. Also such references are more complicated to understand decreasing the maintainability.

The access to neighbor **MODULs** (refer to [MODUL access](#)) is strongly advised against and will therefore result in a warning of the model compiler. Additionally to the arguments against backward references there is a chance of unwanted behavior. Because radCASE generates size optimized code the **SURFACE\_CTRs** are exported only once for multiple instances of a **MODUL**. Now if the neighbor **MODUL** accessed is in another instance of such a multiple instance, radCASE will not be able to identify correctly which instance is accessed and most probably will affect the wrong instance.

#### 4.4 File And Path Access

In a radCASE model it is possible to use absolute and relative paths for including Library files or images. It is also possible to use environment variables for specifying the path. It is strongly recommended to only use absolute paths in connection with environment variables.

It is recommended to use relative paths for all files which are only usable for the current project and to use environment variables for cross-project libraries.



It has to be ensured the environment variables are known to radCASE. On creating or changing an environment variable this most likely means to restart the editor and possibly also programs used to launch the editor (like e.g. Total Commander).

#### 4.5 Calling radCASE Functions From External C Files

Sometimes it is necessary to call a function of a radCASE model from an external C-File e.g. for special library functions that are called from the HAL. Even though it is possible to directly call the generated functions this is not the recommended way, because the generated name also contains the module hierarchy. So if calling the generated function directly, the external C-File would not be reusable because it requires a specific module hierarchy. There are two recommended ways for calling a radCASE function from an external C-File:

##### 4.5.1 Global Wrapper Function

Define a global function in the global **Code** of a [METHODS](#) container and put a declaration into the **Global definitions**. In the global function just call the radCASE-function using **\$-Access** (refer to [Behavior Access](#)).

In the external C-File you have to include the *include\_all.h* located in the *APPL* directory of the project. Now you can just call your global function from the C-Code.



The global function definition will not be altered by the model compiler and so the name will be the same independent of the module hierarchy. The model compiler will take care of the module hierarchy for the **\$-notation** so the module is reusable even for other module hierarchies. However using the global function the module containing the called radCASE function may only be instantiated once.

#### 4.5.2 Callback Function

For this mechanism the external C-Code may not directly call a function, but has to call a function using a function pointer. The external C-Code also has to provide a function to set this function pointer.

In the radCASE model the function for setting the function pointer has to be declared as external function and can then be called to pass the function pointer of the radCASE-function to the external C-Code. To pass the function pointer the \$\*-notation (refer to [Function Pointer Access \(\\$\\*\)](#)) can be used.



Because radCASE expands the function arguments for multiple instances, this method also only works for function in single instantiated **MODULs**.

#### 4.6 Simulating The Process

It is recommended to use **Stimulation Equations** (refer to [Stimulation Equations](#)) for every hardware input in the project, to be able to simulate the process without the need of the actual hardware. For the simulation of the process there should also be some [ELEMENTs](#) of [Assign type](#) **LOCAL** which are used to simulate some error states to also check the correct behavior in case of hardware failures.

For example there could be a **DO** *DO\_fHeater* turning on a heating element. The temperature sensor *AI\_Temperature* then should have a **Stimulation Equation** raising the temperature if the **DO** is turned on and slowly decreasing the temperature when turned off. Now there could also be a **LOCAL** *fHeatError* which can be turned on, to cause the **Stimulation Equation** to not raising the temperature. The **Stimulation Equation** of *AI\_Temperature* could then look like:

$$AI\_Temperature[1] - 0.1 + ((DO\_fHeater \& !fHeatError) * 0.3) \min 15.0 \max 70.0$$

Using this **Stimulation Equation** the **LOCAL** *fHeatError* could be used to test if the application has a correct error handling for a defect heating element, e.g. a timeout which causes an emergency stop.

#### 4.7 Being Aware Of Task-Scheduling

It is important to have a basic understanding of the scheduling of a radCASE application (refer to [Scheduling](#)) to prevent some frequently made errors.

Because the **Perm-task** periodically calls all the **PERM**-functions it is important to avoid using loops especially loops which will consume much time until finishing in those **PERM**-functions. Using loops may otherwise result in a blocked **Perm-task** which will not continue to call other functions as long as the loop runs. Also depending on the task switching mechanism and priorities of the tasks on a target, starving of other tasks may happen. This also applies to user defined **Processing types** which are called periodically from the **Perm-task** or in a user defined task.

Because the process normally runs in the **Perm-task** and the HMI is controlled by the **Surface-Interpreter-Task**, triggering actions on the surface is not easily possible from the process. It is best to use target specific interprocess communication functionalities such as Semaphores, when it



is necessary to trigger the surface from the process. Otherwise it is recommended to only visualize the process and strictly separate the process from the surface functionality.

In the other direction when controlling the process using the surface, you should also keep in mind that a task switch could happen at any time, so you should not use variables which are not written in one step, so [ESTRs](#) should not be used for controlling the process. For other data types you have to keep your processor architecture in mind. A 32-Bit processor will have no problems writing a long in one step, for a 16-Bit processor you should not use a long for controlling the process either and on an 8-Bit processor even the use of shorts is dangerous.

[ELEMENTs](#) of [Assign type](#) **PAR** or **SYS** should only be changed if the process is stopped. This is important due to the writing process of those **ELEMENTs** takes some time, where task switches are very likely. Also because they are normally used for settings of the process it will normally result in logical problems of the process when these values change while in the according parts of the program.

The visualization object [APERM](#) contains a series of actions (refer to [Action Handling](#)) which are executed permanently while the surface is active. For **SURFACE\_CTR** the repetition rate is depending on the performance of the Surface-Interpreter-Task. For **SURFACE\_VIS** the repetition rate depends on the setting **DR=<#>** (refer to [Timing](#)).

## 4.8 Using Global Code

It is recommended to avoid using **Global code** within [METHODS](#) in a [MODUL](#) and instead use radCASE structures. Even though it is possible to reference radCASE [ELEMENTs](#) and functionality from the **Global code**, this only works for single instantiated **MODULs**, because for multiple instances radCASE expands the function arguments and also uses this internal structure for **ELEMENT** access.

Nevertheless there are cases where it makes sense to use **Global code** (e.g. for data structures; refer to [Realizing data structures in radCASE](#)). In these cases it is best to only access the global functions, variables or structures from the **MODUL** the global code is located. If access is required from another **MODUL** or external C-Code a declaration should be put into the **Global declarations**. If accessing the **Global code** from another **MODUL** and not making those declarations the code may not be found, especially when using [Ctr-Setting](#) **SMC=<#>**. So the declaration should be made for the best possible reusability.

## 4.9 Handling Timers

When working with [ELEMENTs](#) with an [Assign type](#) of **TI**, it is especially important to be aware of the task scheduling (refer also to [Being Aware Of Task-Scheduling](#)). Most often Timers are used in the Surface-Interpreter-Task or in the PERM-Task to check if a certain amount of time has passed. Depending on the rates of the tasks and the resolution of the Timer it can happen, that the timer is increased multiple times until a check is reached again. Because of this such checks should never use a comparison using **==** but instead should use **>=** to check if a certain amount of time has passed.

#### 4.10 Realizing data structures in radCASE

In every object oriented programming language a data structure can be realized as a class. Because radCASE is object oriented this also applies to radCASE. In radCASE a data structure can be created by creating a [MODUL](#) (the radCASE equivalent to a class) containing the data as [ELEMENTs](#).

Because radCASE **MODULs** and **ELEMENTs** contain some additional metadata, there is some overhead which could be undesired especially for small controllers. In this case simple data structures can also be implemented in global C-Code (refer to [Using Global Code](#)) as a *struct*.

## 5 Language Reference

### 5.1 Overview

#### 5.1.1 General RC-Items

These RC-Items can be used everywhere in the model tree

- [COMMENT](#) For commenting the model
- [HYPERLINK](#) For linking to a local file
- [#IF, #IFN, #ELSE, #ENDIF](#) For precompiler conditions

#### 5.1.2 Language Tree

These RC-Items can only be used at the according places shown in the tree below

- [Project](#) Adding libraries
  - [TEXTDEF](#) Multilingual Text Definition
  - [SYSTEMDEF](#)
    - [SETTINGS](#) System settings
    - [EMB\\_HMI](#) Embedded HMI settings
    - [RADMON](#) Settings for **Project Monitor**
      - [Permissions](#) Permission settings for functionality
      - [Passwords](#) Passwords for permissions
    - [DOCTITLE](#) Name of the model
    - [VERSION](#) Version number of project
    - [DEFINE](#) C Macro Definition
    - [REPLACE](#) Replacement for **Assign types**
  - [TYPEDEF](#)
    - [EBIN](#) Enumerative data type Definition
      - [EB\\_ENTRY](#) Selection status definition
    - [ENUM](#) Numerical data type definition
    - [ESTR](#) String data type definition
    - [EDAT](#) Date data type definition
    - [ETIM](#) Time data type definition
  - [MODULDEF](#)
    - [MODUL](#) Definition of a module
      - [SUBMODUL](#) Instantiation of a **MODUL**
      - [ELEMENT](#) Instantiation of a **TYPEDEF**
      - [SURFACE\\_XXX](#) Different HMIs
        - see list of possible [Surface Items](#)
      - [METHODS](#)
        - [PROC](#) C-function
        - [SEQUENCE DIAGRAM](#)
          - Definition of functionality with sequence diagrams
            - see list of possible [Sequence Diagram Items](#)
      - [STATECHARTS](#) Definition of functionality with state machines

- see list of possible [State Machine Items](#)
- [ACTIVITYCHARTS](#) Definition of functionality with activity charts
  - see list of possible [Activity Chart Items](#)
- [SIGNAL\\_CHART](#) Definition of functionality with signal charts
  - see list of possible [Signal Chart Items](#)
- [SIGNAL\\_ICON](#) Icon for references to **SIGNAL\_CHARTs**
  - see list of possible [Surface Items](#)
- [RELATION](#) Relation between two **MODULs**
- [PORT](#) definition of interface of a **MODUL**
  - [ELEMENT](#) Instantiation of a **TYPEDDEF**
  - [PROC](#) C-function
- [ARTEFACT](#)
- [RELATION](#) Relation between two **MODULs**
- [PICTDEF](#)
  - [PICT](#) Picture definition
    - [SETPOS](#) Position for relative positioning of following RC-Items
    - [RECT](#) Rectangle
    - [CIRC](#) Circle
    - [ARC](#)
    - [LINE](#)
    - [CBITMAP](#) Bitmap for target hardware
    - [WBITMAP](#) Bitmap for **Project Monitor**
    - [TEXT](#) Static text
    - [IPIC](#) Reference to **PICT**
    - [FILL](#) Fill an area on a surface in **Project Monitor**
- [VISUALDEF](#)
  - [VISBINICON](#) State-Visualizer for **ELEMENTs** of **EBIN** data type (e.g. LEDs)
  - [VIS\\_LINE](#) Moving visualizer for **ELEMENTs** (e.g. slide control)
  - [VIS\\_ROT](#) Rotating visualizer for **ELEMENTs** (e.g. control dial)
  - [VIS\\_ROT\\_XY](#) Moving and rotating visualizer for multiple **ELEMENTs**
- [EntityTab](#) Edit **ELEMENT** and **SUBMODUL** instances
- [ARTEFACTDEFS](#)
  - [ARTEFACTDEF](#) Definition for **ARTEFACT** types.
- [Requirement](#)
  - [Usecase-Diagram](#)
    - see list of possible [Usecase-Diagram Items](#)
- [EProfiles](#)
  - [EProfile](#) Resource consumption analysis energy profile
- [EGroups](#)
  - [EGroup](#) Resource consumption analysis energy group

### 5.1.3 Surface Items

RC-Items that can be used in different surfaces. Not all items are supported in all Surfaces. Refer to [Surface item support](#) for more information.

- [DOCTAB](#) Settings for documentation generation
  - [DT\\_ENTRY](#) Entry for **DOCTAB** settings

- [FULL](#) Display another **SURFACE\_XXX** within current one
- [ICON](#) Display icon of another **SURFACE\_XXX**
- [TEXT](#) Static text
- [IPIC](#) Reference to **PICT**
- [MENU](#)
  - [AMENU](#) menu entry that triggers an action
    - see list of possible [Actions](#)
  - [EDIT](#) menu entry to display/edit an **ELEMENT**
  - [TEXT](#) Static text
  - [LINE](#)
  - [IPIC](#) Reference to **PICT**
  - [RECT](#) Rectangle
  - [AICON](#) Icon that reacts on keyboard and touch to trigger actions
    - see list of possible [Actions](#)
  - [COLUMNBREAK](#) Separator between two columns
  - [IF, ENDIF, ELSE](#) display RC-Items conditionally
- [ELEM](#) Display/edit value of **ELEMENT**
- [MElem](#) Display multiple **ELEMENT**s
- [DISPLAY](#) Show content of embedded HMI
- [AICON](#) Icon that reacts on keyboard and touch to trigger actions
  - see list of possible [Actions](#)
- [AKEY](#) Execute actions on specific keyboard key
  - see list of possible [Actions](#)
- [AKEYGLOBAL](#) Trigger actions globally on every surface on specific keyboard key
  - see list of possible [Actions](#)
- [AENTER](#) Trigger actions on entering surface
  - see list of possible [Actions](#)
- [APERM](#) Trigger actions permanently while on surface
  - see list of possible [Actions](#)
- [AEXIT](#) Trigger actions on exiting surface
  - see list of possible [Actions](#)
- [ActionCond](#) Trigger actions on specific condition
  - see list of possible [Actions](#)
- [TouchKey](#) Generate keyboard key on touch
- [IF, ENDIF, ELSE](#) display RC-Items conditionally
- [SETPOS](#) Position for relative positioning of following RC-Items
- [RECT](#) Rectangle
- [CIRC](#) Circle
- [LINE](#)
- [FOLDER](#) not yet supported
  - [CARD](#) not yet supported
- [FILL](#) Fill an area on a surface in **Project Monitor**

#### 5.1.4 Actions

These are actions that can be used in different action triggers

- [SET](#) Sets **ELEMENT** to 1
- [RES](#) Resets **ELEMENT** to 0
- [INV](#) Inverts **ELEMENT**
- [PUT](#) Puts specific value into **ELEMENT**
- [INC](#) Increases value of **ELEMENT**
- [DEC](#) Decreases value of **ELEMENT**
- [AEDIT](#) Opens edit dialog of **ELEMENT**
- [COPY](#) Copies value of one **ELEMENT** to another
- [GOTOSURFACE](#) Changes current surface
- [CALLSURFACE](#) Changes current surface (with stack)
- [RETURN \(Sur\)](#) Returns to old surface from stack
- [OPEN](#) Open surface in another **MODUL** (with stack)
- [CLOSE](#) Return to old surface from stack
- [CFUNC](#) Call of predefined system functions
- [USERFUNC](#) Deprecated do not use anymore
- [PROCEDURE](#) Call a **PROC** in the model
- [PASSWORD](#) Require password to execute actions
- [PSWD CLR](#) Reset password level
- [ASKOK](#) Ask user to execute actions

### 5.1.5 Sequence Diagram Items

RC-Items that can be used in a **SEQUENCE DIAGRAM**

- [MESSAGE](#) Call of functionality
- [IF, ELSE IF, ELSE, END IF](#) Conditional section
- [CODE \(Seq\)](#) C code to execute
- [LOOP, END LOOP](#) Loop section
- [RETURN \(Seq\)](#) Exit sequence and return value
- [REFERENCE](#) Incorporate functionality

### 5.1.6 State Machine Items

RC-Items that can be used in **STATECHARTS**

- [MACHINE](#) Definition of a state machine
  - [STATE](#)
    - [ENTER](#) Code to execute on entering **STATE**
    - [DURING](#) Code to execute permanently while in **STATE**
    - [TRANSITION](#) Transition to other **STATES**
    - [EXIT](#) Code to execute on exiting **STATE**
  - [UNITSTATE STATE](#) containing **STATES**
    - [ENTER](#) Code to execute on entering **STATE**
    - [DURING](#) Code to execute permanently while in **STATE**
    - [EXIT](#) Code to execute on exiting **STATE**
    - [STATE](#)
      - see RC-Item **STATE**

- [UNITSTATE](#)
  - see RC-Item **UNITSTATE**
- [TRANSITION](#) Transition to other **STATES**
- [SUBMACHINE](#) Incorporate another **MACHINE**
  - see RC-Item **SUBMACHINE**
- [TRANSITION](#) Transition to other **STATES**
- [DURING](#) Code to execute permanently while in **STATE**
- [CHOICE](#) Choose between different target **STATES**
  - [TRANSITION](#) Transition to other **STATES**
- [ENDSTATE](#) Final **STATE**
- [SUBMACHINE](#) Incorporate another **MACHINE**
  - [TRANSITION](#) Transition to other **STATES**
  - [ENTRY POINT](#)
    - [TRANSITION](#) Transition to other **STATES**
  - [EXIT POINT](#)
    - [TRANSITION](#) Transition to other **STATES**
- [ENTRY POINT](#) of **SUBMACHINE**
  - [TRANSITION](#) Transition to other **STATES**
- [EXIT POINT](#) of **SUBMACHINE**
  - [TRANSITION](#) Transition to other **STATES**

### 5.1.7 Activity Chart Items

RC-Items that can be used in **ACTIVITYCHARTS**

- [ACTIVITY](#)
  - [START](#) Starting point of **ACTIVITY**
    - [CONTROLFLOW](#) Connect RC-Items of an **ACTIVITY**
  - [END](#) End point of **ACTIVITY**
  - [ACTION](#) Action to perform
    - [CODE \(Act\)](#) C code to execute
    - [CONTROLFLOW](#) Connect RC-Items of an **ACTIVITY**
  - [BRANCH](#) Conditional execution
    - [CONTROLFLOW](#) Connect RC-Items of an **ACTIVITY**
  - [LABEL](#) not yet supported

### 5.1.8 Signal Chart Items

RC-Items that can be used in a **SIGNAL\_CHART**

- [TEXT](#) Static text
- [IPIC](#) Reference to **PICT**
- [SETPOS](#) Position for relative positioning of following RC-Items
- [RECT](#) Rectangle
- [CIRC](#) Circle
- [LINE](#)
- [FILL](#) Fill an area on a surface in **Project Monitor**
- [ELEM](#) Display value of **ELEMENT**

- [SignalIcon](#) Refer to a **SIGNAL\_ICON**
- [Connection](#) Connect **SignalIcons** and **ELEM**s

### 5.1.9 Usecase-Diagram Items

RC-Items that can be used in a **Usecase-Diagram**

- [Actor](#)
  - [Association](#) Connection between **Actors** and **Usecases**
- [Usecase](#)
  - [Scenario](#) Links to the functionality of the **Usecase**
    - [ScenarioReference](#) not supported yet
  - [Attribute](#) Links to the elements of the **Usecase**
    - [AttributeReference](#) not supported yet
  - [Association](#) Connection between **Actors** and **Usecases**
- [Sub-Usecase-Diagram](#) Links to another **Usecase-Diagram**

## 5.2 COMMENT

A **COMMENT** is used to comment the model and will not affect the generated code.

The following attributes are available:

**Comment** The comment

## 5.3 HYPERLINK

A link to a local file with additional information which does not affect the generated code.

The following attributes are available:

**Hyperlink path** Path to local file

## 5.4 #IF, #IFN, #ELSE, #ENDIF

Precompiler conditions to enable/disable specific parts of a project in dependency on [DEFINE](#)s.

The following attributes are available:

**Define** Name of Define to check for

**Value** Value to compare with

## 5.5 Project

The source node of every model file, which specifies libraries and code generation.

The following attributes are available:



<b>Library</b>	Library files included in the current model file
<b>Code (only in model file with System MODUL)</b>	Selects simulation project configuration (Release/Debug) and affects creation of signal diagrams (refer to <a href="#">Virtual MODUL</a> ) Creation of a <a href="#">Standalone</a> version is only possible in Release mode

The following child RC-Items are available:

<a href="#">TEXTDEF</a>	Multilingual text definitions
<a href="#">SYSTEMDEF</a>	System settings and Defines
<a href="#">TYPEDEF</a>	Data type definitions
<a href="#">MODULDEF</a>	<b>MODUL</b> definitions
<a href="#">PICTDEF</a>	Picture definitions
<a href="#">VISUALDEF</a>	Visualizer definitions
<a href="#">EntityTab</a>	Edit <b>ELEMENT</b> and <b>SUBMODUL</b> instances
<a href="#">Requirement</a>	Usecase-Diagram definitions
<a href="#">EProfiles</a>	Resource consumption analysis energy profile definitions
<a href="#">EGroups</a>	Resource consumption analysis energy group definitions

## 5.6 TEXTDEF

Text definitions for multilingual texts. Refer to [Text Support](#) for more details.

## 5.7 SYSTEMDEF

**SYSTEMDEF** contains the different settings of a project.



**SYSTEMDEF** is only allowed in the main project file containing [The System MODUL](#).

The following child RC-Items are available:


<a href="#">SETTINGS</a>	System settings
<a href="#">EMB_HMI</a>	Embedded HMI settings
<a href="#">DOCTITLE</a>	Name of the model
<a href="#">VERSION</a>	Version number of project

## DEFINE C Macro definitions

### 5.8 SETTINGS

System settings of the project

The following attributes are available:

<b>Display edit default value (0/1)</b>	Affects the edit dialog of elements in the <b>Project Monitor</b> . Enables a button to set the element to their default value: 0: disabled. 1: enabled.
<b>Window title hierarchy (0/1)</b>	Switches between showing only the name of the current module and the whole module hierarchy in the title of a <b>Project Monitor</b> window: 0: name of the current module 1: whole module hierarchy
<b>Initial graph X-zoom level (0-20)</b>	<p>Zoom level of X-axis at the start of <b>Project Monitor</b>. (Standard value 5)</p> <p>The following zoom levels are supported (The time in parenthesis is the whole time interval shown):            0 (1s), 1 (2s), 3 (12s), 4 (24s), 5 (1min), 6 (2min), 7 (6min), 8 (12min),            9 (24min), 10 (1h), 11 (2h), 12 (6h), 13 (12h), 14 (1d), 15 (2d), 16 (5d),            17 (10d), 18 (20d), 19 (50d), 20 (100d).</p> <div>  <p>On startup the <b>Project Monitor</b> load the last user settings, so the initial zoom level only affects the first startup when there are no stored user settings. The initial zoom level is also restored whenever a new recording is created.</p> </div>
<b>Systemcode</b>	These settings define the global system properties of the target system. For a list of possible settings refer to <a href="#">Systemcode</a> .
<b>Timing</b>	The Timing settings affect the timing of target tasks and the timing in communication and data recording of the <b>Project Monitor</b> . For a list of possible settings refer to <a href="#">Timing</a> .
<b>Ctr</b>	These settings affect the generated code of the target system. For a list of possible settings refer to <a href="#">Ctr</a> .
<b>Desktop</b>	These settings affect project monitoring and code and documentation generation. For a list of possible settings refer to <a href="#">Desktop</a> .
<b>Foreground/background color</b>	Color settings for the <b>Project Monitor</b> .
<b>Development status</b>	For details refer to <a href="#">Keeping Track Of Development Status</a> .

<b>Test status</b>	For details refer to <a href="#">Keeping Track Of Development Status</a> .
<b>Target directory</b>	Directory of HAL. It is recommended to use Gen-Files with different Target directories for different hardware, refer to <a href="#">Recommended Project File Structure</a> .

### 5.8.1 Systemcode

In the Systemcode the following settings are available (it is possible to use multiple settings at once separated by space). The settings will generate Defines with the same name:

<b>DIAG_ON</b>	Activates diagnosis routines for diagnosis of IOs
<b>DISP_GRA</b>	Activate functions for graphical displays (also refer to <a href="#">Graphic Handling</a> )
<b>DISP_TXT</b>	Activate functions for text displays
<b>EEPROM_ON</b>	Activate EEPROM handling functions
<b>FLASH_ON</b>	Activates flash handling functions (e.g. for external data flash)
<b>KALI_ON</b>	Activate calibration routines for calibration of IOs
<b>RTC_ON</b>	Activates routines for handling system time using RTC
<b>TOUCH_ON</b>	Activates functionality for touch screen support (refer to <a href="#">Touch Support</a> for more details)
<b>WHEEL_ON</b>	Activates functionality for handling incremental encoders (refer to <a href="#">Keyboard Handling</a> for more details)
<b>ETHERNET_ON</b>	Activates functionality for Ethernet support (refer to <a href="#">Communication</a> )

### 5.8.2 Timing

In Timing the following settings are available (it is possible to use multiple settings at once separated by comma).

<b>ACR=&lt;#&gt;</b>	<b>Asynchronous Communication Event Rate</b> (default = 1) Sets the rate of sending asynchronous communication events. One event is send every n-th communication cycle. Refer to <a href="#">Communication Sequence</a> for an explanation of the overall impact of this setting.
<b>BG=&lt;#&gt;</b>	<b>Burst Graphic</b> (default = 20 ms) Time grid distance (in ms) between two burst samples in graphical presentation. Also refer to <a href="#">Burst</a> .
<b>BM=&lt;0/1&gt;</b>	<b>Burst Mode</b> (default = 0) If set to 1 this setting activates a special display mode for burst data in a histogram (refer to <a href="#">Histogram Visualizers</a> ). In this case the current burst data recording will always be in the middle of the histogram. Also refer to <a href="#">Burst</a>

- CD=<\$>**     **Communication DLL:** (default = cclConnection.dll)  
Defines the standard communication DLL to use if no DLL was selected on command line and no previous configuration was found (refer to [Communication](#)).
- CG=<#>**     **Compressed Graphic** (default = 1000 ms)  
Time grid distance (in ms) between two communication samples for time compressed graphical presentation (refer to [Histogram Visualizers](#)).
- DCR=<#>**     **Data Communication Rate** (default = 1)  
Sets the rate of receiving data from target. One packet of data is received every n-th communication cycle. Refer to [Communication Sequence](#) for an explanation of the overall impact of this setting.
- DMR=<#>**     **Debug Message Rate** (default = 0)  
Sets the rate of receiving debug messages from target. One packet of data is received every n-th communication cycle. If set to 0, all messages will be received every communication cycle. Refer to [Communication Sequence](#) for an explanation of the overall impact of this setting.
- DR=<#>**     **Display Rate** (default = 500 ms)  
Update rate of data from simulation in **Project Monitor** (in ms).  
Refer to [Communication Sequence](#) for an explanation of the overall impact of this setting.
- ET=<#>**     **Event Tolerance** (default = 5000 ms)  
Maximum time between a record sample and an event for the event to be associated to the record sample.
- HCR=<#>**     **HMI Communication Rate** (default = 1)  
Sets the rate of receiving display data from target. One packet of data is received every n-th communication cycle. Refer to [Communication Sequence](#) for an explanation of the overall impact of this setting.
- IBD=<#>**     **Inter Block Delay** (default = 10 ms)  
Time in ms between two blocks in the communication between Project Monitor and target hardware. Refer to [Communication Sequence](#) for an explanation of the overall impact of this setting.
- IR=<#>**     **Intertask Rate** (default = 10 ms)  
Call rate of the Inter task (in ms). This setting only generates the define **INTER\_TASK\_RATE**, that can be used i.e. by the HAL or the virtual simulation HAL.  
For more information on tasks refer to [Scheduling](#).
- PR=<#>**     **Permtask Rate** (default = 10 ms)  
Call rate of the Perm Task (in ms). This setting only generates the define **PERM\_TASK\_RATE**, that can be used i.e. by the HAL or the virtual simulation HAL.  
For more information on tasks refer to [Scheduling](#).
- SR=<#>**     **Storage Rate** (default = 1000 ms)  
Time (in ms) between two record samples in **Project Monitor**. For more information on recordings refer to [Data Recording](#).
- ST=<#>**     **Stimulation Rate** (default = 100 ms)  
Call rate of [Stimulation Equations](#) and update rate of **LOCALs** (in ms). Refer to [Communication Sequence](#) for an explanation of the overall impact of this setting.

- TCR=<#>** **Touch Communication Rate** (default = 1)  
Sets the rate of sending touch events to target. One packet of touch data is send every n-th communication cycle. Refer to [Communication Sequence](#) for an explanation of the overall impact of this setting.
- TG=<#>** **Time Graphic** (default = 1000 ms)  
Maximum time difference (in ms) between two record samples in a histogram. If the tolerance is exceeded, the two samples will not be connected and the graph will be interrupted.

### 5.8.3 Ctr

In Ctr the following settings are available (it is possible to use multiple settings at once separated by comma).

- AIS=<0/1>** **Array index substitution** (default = 0)  
If set to 1 enables the array index substitution. For more details refer to [Event triggered communication](#).
- ATP=<0/1>** **Assign type prefix** (default = 0)  
If set to 1 all generated [ELEMENT](#)s will be generated with a prefix. The Prefix `_P_` will be generated for every element, which is in a [PORT](#). In addition for each element a Prefix `_xx_` will be generated where xx are the first two characters of the [Assign type](#). Only exceptions are **INEVT** where the Prefix is `_IE_` and **OUT-EVT** where it is `_OE_`.
- BA=<#>** **Binary Alignment** (default = 2)  
This setting determines the alignment of binary exported data (*OsdL.ini*, *OsdL.txt* and *OsdL.bmp*). Note that it is affected by setting **AL=<#>** of [Desktop](#) (Minimum value of **BA** is value of **AL**).  
Since data is managed within the binary file using a 16 bit Index, this also determines the maximum length of the binary data:  
2: <128k.  
4: <256k.  
8: <512k.  
The *OsdL.txt* has other file length restrictions. Refer to [Optimizing File Size Of OsdL.txt](#) for more information.
- BF=<#>** **Blink Frequency** (default = 5  $\hat{=}$  500ms)  
Blink frequency (in 100 ms) of [Blinking Elements](#)
- BL=<#>** **Block Length** (default = 0)  
Defines the block length of the IO-block in case of multiple module instances. This setting will add an offset to the index of those IOs.
- BMA=<0x#>** **Bit Mask** (default = deactivated)  
On some processors (e.g. ARM processors) trying to set a bitmask of 0xF0000000 on a long-pointer causes errors, because the RAM starts in that area. With BMA the bit-mask is set to a lower value, preventing this error.  
**BMA** sets the highest 4 bit of the bitmask, e.g. **BMA=0x7** sets the bitmask to 0x70000000.
- BT=<#>** **Blink Type** (default = 0)  
Blinking type of [Blinking Elements](#):

0: delete (change between normal display and no display).  
 1: inverted (change between normal display and inverted display).

- CE=<#>**      **Complete Elements** (default = 0)  
 Defines whether the [ELEMENT](#) information for HMI is always exported completely for all **ELEMENTs**:  
 0: normal export  
 1: all **ELEMENTs** are exported with structure and texts (this can be enforced for single **ELEMENTs** by setting format string **EX=1** in the **ELEMENT**).  
 3: all **ELEMENTs** are exported with structure and texts. Additionally the names of the **ELEMENTs** are exported.
- CEA=<0/1>**      **CElement Access** (default = 0)  
 If set to 1 **ELEMENT** pointers and structures are also usable on targets without HMI (refer to setting **EH=<0/1>** below). This means some internal CEle-ment-functions (e.g. *CElement\_getAct()*) can be used to access **ELEMENTs**.
- COT=<#>**      **CANOpen driver** (default = 0)  
 Switches between CANOpen driver types:  
 0: IXXAT  
 1: PORT
- EAI=<0/1>**      **Export All IOs** (default=0)  
 If set to 1 all IOs are exported into the according tabs in *projekt.tio*, even those of subnodes.
- EH=<0/1>**      **Embedded HMI** (default = 1)  
 If set to 0 defines that the target system does not use an embedded HMI. This means no HMI code is generated.
- EM=<0/1>**      **EEPROM Message** (default = 1)  
 If set to 0 EEPROM initialization errors (checksum error, length error, version error) are handled by reformatting the EEPROM to default values, without an error message to confirm.
- EMA=<0/1>**      **Easy metadata access** (default = 0)  
 If set to 1, access to metadata is generated without using macros for support of external flash. Refer to [Access To Elements Metadata \(\\$#\)](#).
- ESM=<0/1>**      **Export Surfaces Multiple times** (default = 0)  
 If set to 1, **SURFACE\_CTRs** are exported for every **MODUL** instance. Normally to have optimized code size those are only exported one time for all instances. However this can cause problems with procedure calls (refer to [PROCEDURE](#)) from this surfaces.
- ETA=<0/1>**      **Export tables as arrays** (default = 0)  
 If set to 1 the element and module tables in *source.c* are exported as arrays instead of switch statements.
- FCL=<0x#>**      **Font character limit** (default = 0x800)  
 Limit of characters put into *CharsOverLimit.bin* for usage in font compression. Refer to integration manual for more details.
- FM=<0/1>**      **Force Mode** (default = 1)  
 If set to 0 **DOs**, **AOs**, **DIs**, **AI**s and **CNTs** can't be set from **Project Monitor**. (refer

to [Force Mode](#))

- FP=<0/1>**      **Use Float** (default = 0)  
If set to 1 activates real float support. It is recommended to use [Virtual Floating Point](#) instead.
- GX=<0/1>**      **Global XML** (default = 0)  
If set to 1 the file *instances.xml* is created during model compilation containing information on the instantiation of the model.
- HT=<#>**        **Help Text** (default = 1)  
If set to 0 no help texts are generated into the target system.
- LC=<\$+...+\$>**   **Language Controller** (required)  
Selection of languages available on the target system. Language codes are case sensitive and should be listed without spaces, using “+” e.g. *LC=GR+UK*  
For a list of all available language codes refer to [Supported Languages](#).
- LV=<\$+...+\$>**   **Language Visualization** (required)  
Selection of languages available in the [Project Monitor](#). Language codes are case sensitive and should be listed without spaces, using “+” e.g. *LV=GR+UK*  
For a list of all available language codes refer to [Supported Languages](#).
- MCO=<0/1>**      **Multiple CANOpen Elements** (default = 0)  
If set to 1, there will be no error message if a CANOpen index is used for multiple elements.
- MF=<#>**        **Memory Format** (default = 0)  
Defines, whether the target system (processor) works in “Motorola mode” (big-endian) or Intel mode (little-endian):  
0: Intel.  
1: Motorola.
- MK=<0/1>**      **Map Katakana** (default = 0)  
If set to 1 Katakana characters are mapped to the area 0x7A0-0x7FF (refer to [Unicode](#))
- NEP=<0/1>**      **No element pointer** (default = 0)  
If set to 1 there will be no element pointers in the generated **MODUL** structures. This is a feature to reduce the size of the application, but it will cause errors when there are elements referenced using the **\_pMod** structure. The **\_pMod** structure will be used, if an element is accessed out of a module with multiple instances. In this case module instances which inherit from that module will count as instances, too.
- NL=<#>**        **Name Length** (default = 5)  
Maximal character lengths of the **MODUL** instance names, which are taken for generating the C variable names.
- NS=<#>**        **Number of Subtexts** (default = 0)  
This setting determines the maximum number of possible subtexts (refer to [Subtexts](#))
- OPS=<#>**        **OsdI.txt Pointer Size** (default = 2)  
This setting determines how many bytes are used for pointers that address texts



in *OsdI.txt*. Refer to [Optimizing File Size Of OsdI.txt](#) for more information on how to use this feature.

- RP=<#>**      **RDI Pointer Skip** (no default)  
 Skips the export of all pointers with a lower index than this setting into *RDI\_PntTab.c*. It is also possible to select a range to skip, e.g. *RP=3-7*. For more information on this setting refer to [radCASE Index \(RDI\)](#).
- SL=<\$>**      **Standard Language** (no default)  
 Language code of standard language. In case of incomplete multilingual text definitions, standard language texts are used for undefined texts, i.e. missing translations are replaced with words in the standard language during model compilation.  
 For a list of all available language codes refer to [Supported Languages](#).
- SM=<0/1>**      **Save Menu** (default = 1)  
 If set to 0 every time a [MENU](#) is opened the first menu entry is selected. By default the **MENU** will save which menu entry was selected the last time the **MENU** was opened.
- SMC=<#>**      **Split module.c** (default = 0)  
 Maximal number of lines in one part of a splitted *module.c*. If set to 0 the feature is deactivated. For more information refer to [Limiting Code Size Of Generated Files](#) and [Limiting Size Of Module.c](#).
- SOI=<0/1>**      **Split OsdI.ini** (default = 0)  
 If set to 1 the *OsdI.ini* can be splitted. For more information refer to [Limiting Code Size Of Generated Files](#) and [Limiting Size Of OsdI.ini](#).
- SOT=<0/1>**      **Split OsdI.txt** (default = 0)  
 If set to 1 the *OsdI.txt* can be splitted. For more information refer to [Limiting Code Size Of Generated Files](#) and [Limiting Size Of OsdI.txt](#).
- SSB=<0/1>**      **Separate Source Blocks** (default = 0)  
 If set to 1 the *source.c* will be splitted into multiple files. For more information refer to [Limiting Code Size Of Generated Files](#) and [Limiting Size Of Source.c](#).
- ST=<0/1>**      **System Text** (default = 1)  
 If set to 0 radCASE system texts are not generated for the target HMI.
- TB=<0x#>**      **Transparent Background color** (no default)  
 Defines the transparent color in hexadecimal RGB format. Every part of a [CBITMAP](#) painted in this color will be displayed transparent on the target and the target simulation. The Define **RDRGBCOLTRANS** will be exported containing this value.
- UC=<#>**      **Unicode Export** (default = 0)  
 Defines whether Unicode texts (refer to [Unicode](#)) are generated for the target HMI.  
 0: target HMI texts as ASCII  
 1: target HMI texts as Unicode  
 2: target HMI texts as Unicode (UTF8)
- UKE=<0/1>**      **Sub Node Complete Export** (default = 0)  
 If set to 1 exports all elements of subnodes for PC simulation (and only there).



Doesn't change anything in access of the elements, but exports the elements for the simulation, so the simulation can simulate a whole distributed system (refer to [Distributed Systems](#)). In this way also the simulation between different targets can be simulated and debugged.

- UN=<0/1>**      **Unit conversion** (default = 0)  
If set to 1 enables [Automatic Unit Conversion](#)
- UU=<0/1>**      **Use Unsigned Long** (default = 0)  
If set to 1 switches to internal use of unsigned long pointers instead of unsigned short pointers.
- VM=<0/1>**      **Virtual Module** (default = 1)  
If set to 0 virtual **MODULs** are generated as normal signal **MODULs** (refer to [Virtual MODUL](#))
- WL=<#>**      **Warning level** (default = 1)  
Warning Levels are as follows:  
0: no warnings.  
1: important warnings.  
2: warning level 1 + warning for missing standard values.  
3: warning level 2 + warning if data type is not explicitly set and IO data types do not match generated data types  
4: warning level 3 + report of unknown XML Tags in modul.xml.

#### 5.8.4 Desktop

In Desktop the following settings are available (it is possible to use multiple settings at once separated by comma).

- AED=<0/1>**      **Old Element Visualization** (default = 0)  
If set to 1 radCASE uses old element visualization in which the background of editable elements is white colored and the background of non-editable elements is grey colored.
- AL=<#>**      **Alignment** (default = 2)  
Memory alignment of the target system.  
Possible settings: 1, 2, 4, 8.
- AN=<\$>**      **About Box Name** (default = standard radCASE About box)  
Used for customizing the About box of the **Project monitor**. The setting defines the **Name** of the **SURFACE\_VIS** of [The System MODUL](#) which contains the customized About box. That **SURFACE\_VIS** is used as content of the About box instead.
- APU=<0/1>**      **Automatic parameter update** (default = 1)  
Start value of automatic parameter update in **Project monitor**. The feature can still be switched at runtime (refer to Monitoring manual). Parameter data will be automatically transferred from controller as soon as changes are detected.
- BB=<0/1>**      **Burst Button** (default = 1)  
If set to 0 the Burst button (refer to [Burst](#)) is not shown in the **Record Panel**.
- CF=<\$>**      **Css-File** (no default)  
Custom CSS-File which will be included in the generated documentation. The CSS file is only included and has to be copied to the correct destination manually after

creation, e.g. by using *docgen\_post.bat* (refer to [Documentation Generation](#)).

- CM=<0/1>** **Compressed Mode** (default = 0)  
If set to 1 the distance between two record samples in a histogram (refer to [Histogram Visualizers](#)) isn't determined from the real time distance but they are drawn equidistantly with a fixed time schedule (refer to Setting **CG=<#>** in [Timing](#))
- DD=<\$>** **Documentation directory** (default = doc)  
Name of output directory for documentation generation (refer to [Documentation Generation](#)).
- DDT=<#>** **Developer documentation threshold** (default = 5)  
Documentation level in the range from 0-9 needed for documentation to be exported as developer documentation instead of being exported as user documentation.  
Developer documentation contains more information (refer to [Structure Of Generated Documentation](#)). Also comments may contain more information (refer to [Comment Syntax](#)).
- DE=<0/1>** **Direct Edit** (default = 0)  
If set to 1 instead of opening a dialog, editing of values will be directly in the **SURFACE\_VIS** and values are committed as soon as pressing <Enter> or <TAB>.
- DEX=<0/1>** **Documentation Export** (default = 1)  
If set to 0 there will be no binary export of texts for the documentation. Because of this, don't set this parameter to 0 if you want to generate documentation.
- DME=<0/1>** **Disable Message Not Editable** (default = 0)  
If set to 1 the message "This element is not editable" will not be shown. This may be useful if using a non-editable **ELEMENT** as icon to open another surface.
- DN=<\$>** **Dynamic Version Switching MessageBox** (default = standard radCASE error message)  
If using Dynamic Version Switching (refer to [Dynamic Version Switching](#)) and there are no configuration files for the version of a detected target, an error message is displayed. The setting defines the **Name** of the **SURFACE\_VIS** in [The System MODUL](#) that is shown instead of the standard error message.
- DP=<#>** **Data Preservation** (default = 0)  
Contains a bitmask to export special structures, to support a **Data Preservation** of **PAR**, **SYS** and **PROC ELEMENT**s on the target, even if the structure of those data blocks changes. The bitmask defines which element types are exported:  
1 = SYS  
2 = PAR  
4 = PROC  
For every **ELEMENT** of the activated types an **RDI** (refer to [radCASE Index \(RDI\)](#)) will be generated. Refer to [Data preservation](#) for further information.
- DTD=<0/1>** **Debug target display** (default = 0)  
If set to 1 the debugging of target display in the simulation is easier, because the effect of a drawing routine is directly shown on the simulated target display. However this feature should only be used, for debugging graphical routines, because it drastically slows down the performance of drawing routines.

- EI=<#>** **Element Indices** (default = -1)  
 Processing of element indices for manually determined addressing of elements in case of distributed embedded systems (refer to [Distributed Systems](#)):  
 -1: do not generate.  
 0: list with complete path.  
 1: omit first path level.  
 ...  
 n: omit nth path level.
- EPM=<0/1>** **Export parameters into measure DB** (default = 0)  
 If set to 1 parameters will be saved in the measure DB. This means not only the last valid value of the parameter is saved, but every value over the time of the recording. This results in bigger database files.
- FS=<0/1>** **Full Screen** (default = 0)  
 If set to 1 the **Project Monitor** will always be started in full screen mode.
- LD=<0/1>** **Language Specific Decimal Point** (default = 0)  
 If set to 1 the language specific comma will be used instead of a decimal point for language selections GR and IT. The setting affects all numerical outputs of the target system as well as the **Project Monitor**.
- MD=<\$>** **Menuitem Disable** (no default)  
 Used to remove menu items of the **Project Monitor**:  
 ?\_ABOUT: Removes menu item for About dialog  
 ?\_HELP: Removes menu item for Help
- MR=<#>** **Max Records** (default = 10 000)  
 Defines how many data records can be recorded. If this value is reached the recording will be stopped. This value should not be higher, than 10 000 because this can cause problems and may result in radCASE becoming very slow in processing the fetched data.  
 If you want to record more data, you can use the data export settings (refer to Project Monitor Manual)
- MT=<\$>** **Main Title** (no default)  
 Title of main window of **Project Monitor**.
- NE=<0x#>** **Node Export** (default = 0x1)  
 Definition of the node to be exported as a Hex number.  
 E.g. 0x04 is set as bit 2 (counted from 0 onwards) → node 3 is exported.  
 Default value 0x1 means: root is the first node and this is exported.
- NS=<0/1>** **Don't Save Graph** (default = 0)  
 If set to 1 there will appear no dialog asking if you want to save the graph, when you're about to lose unsaved data.
- PCL=<#>** **PNG Compressing Level** (default = 6)  
 Sets the compression level of PNGs that are created by radCASE when generating documentation. Valid values are: 0 to 9.
- PN=<0/1>** **Password Notify** (default = 0)  
 If set to 1 there will be no notify of which password level was recognized.
- PRE=<0/1>** **Proc Export** (default = 0)

If set to 1 the **ELEMENTs** with **Assign type PROC** are exported in XML and ASCII export.

- RI=<#>**      **Root ID** (no default)  
The ID of the root node can be set here in case of distributed systems (refer to [Distributed Systems](#)).
- SD=<0/1>**      **Space Disable** (default = 0)  
If set to 1 the use of space to execute an object which is selected with a focus rectangle is disabled.
- SE=<0/1>**      **SysParam Export** (default = 0)  
If set to 1 the **ELEMENTs** with **Assign type SYS** are exported in XML and ASCII export. This also affects the loading of exported XML files.
- SLR=<0/1>**      **Select Last Record** (default = 0)  
If set to 1 when changing to playback mode the last record is selected automatically.
- SNS=<0/1>**      **Show NoValue Selection** (default = 1)  
If set to 0 you can't select **NoValues** in the edit dialogs of **SUFACE\_VIS**
- TB=<0x#>**      **Transparent background color** (no default)  
Defines the transparent color in hexadecimal RGB format. Every part of a [WBIT-MAP](#) painted in this color will be displayed transparent in the **Project Monitor**.
- TE=<0/1>**      **Transition Enable** (default = 0)  
If set to 1 the pointer `FTransEnabled` will be exported. This pointer can point to a variable of type unsigned char. When that variable is set to 0 all transitions in state machines are stopped.
- TT=<#>**      **ToolTips** (default = 0)  
Determines the type of tooltips displayed in **SURFACE\_VIS** in the **Project Monitor**:  
0: without tooltips.  
1: tooltips display the info regarding module or elements.  
2: tooltips show the info regarding module or elements and also show the programming name of the element for **ELEM**s.
- VA=<0/1>**      **Visualizer out of display** (default = 0)  
If set to 1 the simulation doesn't show the message "Visualizer out of display" if elements don't fit into the simulated target display.
- ZA=<0/1>**      **Zoom Automatically** (default = 0)  
When set to 1 zooming an element in a histogram results in automatically zooming all other elements of the same type to the same settings.

## 5.9 EMB\_HMI

Properties of embedded displays

The following attributes are available:

- Display size X/Y**      Width/height of the display in pixels. For character displays a virtual number of pixels should be defined, i.e. number of characters fitting on

	the display multiplied with size of a character in pixels.
<b>Character size X/Y</b>	Width/height of a character in pixels.
<b>Number of cursor keys</b>	<p>Describes the number of keys available not counting &lt;ENTER&gt; and &lt;ESC&gt; (refer to <a href="#">Keyboard Handling</a> for more details).</p> <p>The following settings are available</p> <p>2: &lt;UP&gt;, &lt;DOWN&gt; are available.</p> <p>4: &lt;UP&gt;, &lt;DOWN&gt;, &lt;LEFT&gt;, &lt;RIGHT&gt; are available.</p> <p>&gt;=10: like 4, but an additional numerical keypad is available.</p> <p>&gt;=100: like 4, but an additional alphanumeric keypad is available.</p>
<b>Number of function keys</b>	<p>Describes the number of function keys available not counting &lt;ENTER&gt; and &lt;ESC&gt; (refer to <a href="#">Keyboard Handling</a> for more details).</p> <p>The following settings are available:</p> <p>0: no function key is available</p> <p>1: &lt;F1&gt; is available.</p> <p>2: &lt;F1&gt;, &lt;F2&gt; are available.</p> <p>3: &lt;F1&gt;, &lt;F2&gt;, &lt;F3&gt; are available.</p>
<b>Bits per pixel</b>	<p>Color depth of display (refer to <a href="#">Color Details</a> for more information)</p> <p>The following settings are available:</p> <p>1 bit (b/w)</p> <p>2 bits (4 colors)</p> <p>4 bits (16 colors)</p> <p>8 bits (256 colors)</p> <p>16 bits (65536 colors)</p> <p>24 bits (16 million colors)</p> <p>32 bits (16 million colors)</p>
<b>Export bitmap</b>	If checked bitmaps should be exported.
<b>Export bitmaps with horizontal byte setup</b>	If checked bitmaps are exported horizontally else vertically, the HAL has to support the selected mode (refer to Integration manual).
<b>Foreground color</b>	Global foreground color of <b>SURFACE_CTR</b> . The Define <b>RDRGBCOL</b> will be exported containing the value of this setting.
<b>Background color</b>	Global background color of <b>SURFACE_CTR</b> . The Define <b>RDRGBCOLBK</b> will be exported containing the value of this setting.
<b>Font assignment</b>	Assign string for custom fonts (refer to <a href="#">Custom Fonts</a> ).
<b>Frame bitmap</b>	Definition of the display background image to be used in the documentation. One can display an image (e.g. control panel or the complete device). The path is relative from the Develop directory of the project. The display will be centered horizontally. The offset used for the X-Axis will also be used for the Y-Axis, regardless of the bitmap height.
<b>Documentation Formatting</b>	This feature isn't supported yet.

## 5.10 RADMON

**RADMON** contains different settings regarding the **Project Monitor** (refer to [Project Monitor](#)).

## 5.11 Permissions

Permission definitions for different **project monitor** features. Refer to [Feature permissions](#) for more details.

## 5.12 Passwords

Passwords for different permission levels. Refer to [Password management](#) for more information.

The following attributes are available:

**Password for Level X** Standard password for permission level X

**Encryption Password** Password for encryption of password file

## 5.13 DOCTITLE

The name of the software project, contained in the generated documentation (refer to [Documentation Generation](#)) and as title of the **Project Monitor** (refer to [Project Monitor](#)).

The following attributes are available:

**Title** The name of the software project

## 5.14 VERSION

The version number of the project, contained in the generated documentation (refer to [Documentation Generation](#)) and as title of the **Project Monitor** (refer to [Project Monitor](#)).

The following attributes are available:

**Version** Version number multiplied by 100. This means entering 1 equals Version 0.01 and entering 100 equals Version 1.00

## 5.15 DEFINE

The **DEFINE** is used to create C macro definitions. A **DEFINE** is converted by the radCASE code generator into a `#define` preprocessor directive. The preprocessor name and its value can be defined by the user. The **DEFINE** can also be used for precompiler conditions (refer to [#IF, #IFN, #ELSE, #ENDIF](#)) and for code optimization (refer to [Using Defines For Feature Activation/Deactivation](#)).

The following attributes are available:

**Name** The name of the pre-processor macro

**Value** The value of the macro

**Info** Additional information describing the macro

## 5.16 REPLACE

Used for project specific replacement of [Assign types](#). For more information refer to [Element Usage And Allocation](#).

The following attributes are available:

**Name** Name of the **Assign type** placeholder

**Value** **Assign type** to use for the placeholder

**Info** Multilingual additional information

## 5.17 TYPEDEF

Used for defining data types. For detailed information how to use this refer to [Data Modeling](#).

The following child RC-Items are available:

[EBIN](#) Enumerative data type definition

[ENUM](#) Numerical data type definition

[ESTR](#) String data type definition

[EDAT](#) Date data type definition

[ETIM](#) Time data type definition

## 5.18 EBIN

**EBIN** deals with an enumerative data type (also known as selective or binary). For more information refer to [Enumerative Elements](#).

The following attributes are available:

**Type ID** Name of data type used to reference to this data type

**Default value** The default value of the data type. For allowed syntax refer to [Enumerative Elements](#)

**Forced data type** Refer to [Forced Data Type](#)

**Allow multiple selec-** If activated the **EBIN** is a **multiselective EBIN**.



tions

**Info** Further explanatory text (for Online help and documentation)

The following child RC-Items are available:

**EB\_ENTRY** Selection status definition

## 5.19 EB\_ENTRY

An **EB\_ENTRY** is an individual selection status of an [EBIN](#). For more details refer to [Enumerative Elements](#).

The following attributes are available:

**Name** The name of one of the selections. The value will be assigned automatically from top to bottom counting up from zero. For multiselective EBINs the values will be the value of the bits starting with 1.

**Description** Multilingual selection text for the HMI.

**Info** Multilingual further explanatory text (for Online help and documentation)

## 5.20 ENUM

**ENUM** deals with numerical data type. For more information refer to [Numerical Elements](#).

The following attributes are available:

**Type ID** Name of data type used to reference to this data type

**Default value** The default value of the data type. For allowed syntax refer to [Numerical Elements](#)

**Forced data type** Refer to [Forced Data Type](#)

**Digits** Number of characters (with point, if there are decimal places and an extra place for the decimal sign if the number is signed)

**Decimals** Number of decimal places

**Range (low/high)** Lowest/highest permissible value (area limits). If the value exceeds these alarm limits the value will be displayed with upward arrows for exceeding and downward arrows in case the values fall short. Also the radCASE functionality will make sure, the limits will not be exceeded.

**Alarm (min/max)** In **Project Monitor** the values are highlighted in terms of color if these values are exceeded.




<b>Display (min/max)</b>	In <b>Project Monitor</b> the display range for graphical visualizers (refer to <a href="#">Element Visualization</a> ) is set to this value instead of <b>Range (low/high)</b> .
<b>Unit</b>	Multilingual text defining the Unit
<b>Step width</b>	Setting the step width of <b>ELEMENT</b> value changes.
<b>Info</b>	Further explanatory text (for Online help and documentation)

## 5.21 ESTR

**ESTR** deals with string data type. For more information refer to [String Elements](#).

The following attributes are available:

<b>Type ID</b>	Name of data type used to reference to this data type
<b>Default value</b>	The default value of the data type. For allowed syntax refer to <a href="#">String Elements</a>
<b>Maximum number of characters</b>	Maximum number of characters allowed in the string. For UTF-8 ( <b>UC=2</b> refer to <a href="#">Ctr</a> ) this setting will set the maximum number of allowed bytes in the string, because of the special coding of UTF-8.
<b>Longtext</b>	<p>The number of characters will not be used to determine the buffer size of internal buffers used e.g. for edit dialogs.</p> <div style="display: flex; align-items: center;">  <div style="margin-left: 10px;"> <p>When using this feature there is the danger of buffer overflows when using the different built-in functions.</p> </div> </div>
<b>Info</b>	Further explanatory text (for Online help and documentation)

## 5.22 EDAT

**EDAT** deals with date data type. For more information refer to [Date Elements](#).

The following attributes are available:

<b>Type ID</b>	Name of data type used to reference to this data type
<b>Default value</b>	The default value of the data type. For allowed syntax refer to <a href="#">Date Elements</a>
<b>Format</b>	<p>Format of display:</p> <p>MM/DD/YY (Month/Day/Year 2 digits)</p> <p>MM/DD/YYYY (Month/Day/Year 4 digits)</p>
<b>Info</b>	Further explanatory text (for Online help and documentation)

## 5.23 ETIM

**ETIM** deals with time data type. For more information refer to [Time Elements](#).

The following attributes are available:

<b>Type ID</b>	Name of data type used to reference to this data type
<b>Default value</b>	The default value of the data type. For allowed syntax refer to <a href="#">Time Elements</a>
<b>Format</b>	Format of display: hh:mm (Hours:Minutes) hh:mm:ss (Hours:Minutes:Seconds) hh:mm:ss.ms (Hours:Minutes:Seconds:Centiseconds)
<b>Info</b>	Further explanatory text (for Online help and documentation)

## 5.24 MODULDEF

The **MODULDEF** contains the **MODUL** hierarchy. Refer to [Project Hierarchy](#) for more details

The following child RC-Items are available:

<a href="#"><u>MODUL</u></a>	Definition of a module
<a href="#"><u>RELATION</u></a>	Relation between two <b>MODULs</b>

## 5.25 MODUL

A **MODUL** is the correspondence of a class in object oriented programming. Refer to [Project Hierarchy](#) for more details.

The following attributes are available:

<b>ModuleID</b>	Definition name by which this module can be instanced.
<b>Stereotype</b>	Type of <b>MODUL</b> . The following types are part of the radCASE language: <b>Normal:</b> Normal <b>MODUL</b> <b>Interface:</b> Will be treated as normal <b>MODUL</b> during model compilation. Only for classification in the model (refer also to the according editor manual of the editor you are using) <b>Abstract:</b> Will be treated as normal <b>MODUL</b> during model compilation. Only for classification in the model (refer also to the according editor manual of the editor you are using) <b>Signal:</b> refer to <a href="#">Signal Diagrams</a> .
<b>Flat generation (only if Stereotype is Signal)</b>	This option turns a Signal <b>MODUL</b> into a <b>Virtual MODUL</b> (refer to <a href="#">Virtual MODUL</a> )
<b>Derived from</b>	Specifies a base <b>MODUL</b> to inherit from (refer to <a href="#">Inheritance</a> ).
<b>Info</b>	Multilingual additional details and information

<b>Force pMod Export</b>	For internal use. Forces the <b>MODUL</b> export with <b>_pMod</b> structure.
<b>Development status</b>	For details refer to <a href="#">Keeping Track Of Development Status</a> .
<b>Progress (%)</b>	For details refer to <a href="#">Keeping Track Of Development Status</a> .
<b>Test status</b>	For details refer to <a href="#">Keeping Track Of Development Status</a> .
<b>Comment</b>	Contains further Comments to the <b>MODUL</b> . Will be used in <a href="#">Documentation Generation</a> . For mor information refer to <a href="#">Comment Syntax</a> .

The following child RC-Items are available:

<a href="#"><u><b>SUBMODUL</b></u></a>	Instantiation of a <b>MODUL</b>
<a href="#"><u><b>ELEMENT</b></u></a>	Instantiation of a <b>TYPEDDEF</b>
<a href="#"><u><b>SURFACE XXX</b></u></a>	Different HMLs
<a href="#"><u><b>METHODS</b></u></a>	Functionality
<a href="#"><u><b>STATECHARTS</b></u></a>	Definition of functionality with state machines
<a href="#"><u><b>ACTIVITYCHARTS</b></u></a>	Definition of functionality with activity charts
<a href="#"><u><b>SIGNAL CHART</b></u></a>	Definition of functionality with signal charts
<a href="#"><u><b>SIGNAL ICON</b></u></a>	Icon for references to <b>SIGNAL_CHART</b> s
<a href="#"><u><b>RELATION</b></u></a>	Relation between two <b>MODUL</b> s
<a href="#"><u><b>PORT</b></u></a>	definition of interface of a <b>MODUL</b>
<a href="#"><u><b>ARTEFACT</b></u></a>	

## 5.26 SUBMODUL

A **SUBMODUL** is an instance of a [MODUL](#). Refer to [Project Hierarchy](#) for more information.

The following attributes are available:

<b>Name</b>	Instance name to reference the <b>SUBMODUL</b> and its contents.
<b>ID</b>	The <b>ModuleID</b> of the <b>MODUL</b> to instantiate.
<b>Submodule type</b>	The following <b>SUBMODUL</b> types are supported: <b>Submodule</b> : The standard value for a normal instantiation <b>Subnode</b> : Used for distributed systems (refer to <a href="#">Distributed Systems</a> ) <b>New dependent node</b> : Deprecated - Use <b>Subnodes</b> instead

	<b>Datanode:</b> Not supported yet <b>Pointer:</b> Used for dynamic instantiation (refer to <a href="#">Dynamic instantiation</a> )
<b>Multiplicity</b>	Turns the submodule into an array of modules (works for all types of submodules). The value entered in the field array determinates the size of the array.
<b>Controller-ID</b>	The ID that is used to identify each of the subnodes in distributed systems (refer to <a href="#">Distributed Systems</a> ).
<b>Description</b>	Multilingual description of the <b>SUBMODUL</b> .
<b>Info</b>	Multilingual additional information
<b>Doc. Level</b>	Minimum documentation level in the range from 0-9 to add submodule to documentation (refer to <a href="#">Structure Of Generated Documentation</a> ).

## 5.27 ELEMENT

An **ELEMENT** is a radCASE variable, for more details refer to [Data Modeling](#)

The following attributes are available:

<b>Name</b>	Instance name of the <b>ELEMENT</b> used for referencing
<b>Type</b>	The data type created in <a href="#">TYPEDEF</a>
<b>Multiplicity</b>	Define array size of <b>ELEMENT</b> for limited <b>ELEMENT</b> array support (refer to <a href="#">Element Arrays</a> )
<b>Interface type</b>	Used for structural UML diagrams (refer to the according editor manual of the editor you are using)
<b>Description</b>	Multilingual description
<b>Info</b>	Multilingual additional information
<b>Assign type</b>	Refer to <a href="#">Element Usage And Allocation</a> . For a complete list of supported <b>Assign types</b> refer to <a href="#">Assign type</a>
<b>Com type</b>	Communication settings for communication within Distributed Systems (refer to <a href="#">Distributed Systems</a> ), with <b>Project Manager</b> or with data-base/protocol systems. For a complete list of supported settings refer to <a href="#">Com Type</a> .
<b>Calibration (only IOs)</b>	If set the IOs can be calibrated using a two point calibration, if not the calibration data is static.
<b>Assign string</b>	Element allocation and <b>assign type</b> specific settings. For syntax details for different <b>assign types</b> refer to <a href="#">Assign string</a>
<b>Runtime dynamic meta data</b>	If set the metadata can be changed at runtime (refer to <a href="#">Changing Of Metadata At Runtime</a> )

<b>Simulation</b>	Equation for Offline Stimulation (only relevant for Hardware Input Elements). Refer to <a href="#">Stimulation Equations</a>
<b>Format string</b>	Element attributes. For a complete list of attributes refer to <a href="#">Format string</a> .
<b>EProfiles</b>	Refer to <a href="#">Resource Consumption Analysis</a>
<b>Documentation</b>	Text to document the <b>ELEMENT</b> , will be used in generated documentation (refer to <a href="#">Documentation Generation</a> )
<b>Doc. Level</b>	Minimum documentation level in the range from 0-9 to add <b>ELEMENT</b> to documentation (refer to <a href="#">Structure Of Generated Documentation</a> ).
<b>XCOM-type</b>	Refer to <a href="#">XCOM-type</a>

### 5.27.1 Assign type

The following **Assign types** are supported:

<b>FLAG</b>	The standard working variable (refer to <a href="#">General Data</a> )
<b>PROC</b>	Process data which stores data over a power down (refer to <a href="#">Non-Transient Data</a> )
<b>PAR</b>	Used to permanently store data seldom changed (refer to <a href="#">Non-Transient Data</a> )
<b>SYS</b>	Used to permanently store data normally only saved once at system setup (refer to <a href="#">Non-Transient Data</a> )
<b>CONST</b>	Constant value (refer to <a href="#">General Data</a> )
<b>IN</b>	Data which is an input of a <b>MODUL</b> (refer to <a href="#">General Data</a> )
<b>OUT</b>	Data which is an output of a <b>MODUL</b> (refer to <a href="#">General Data</a> )
<b>IO</b>	Data which is an input and an output of a <b>MODUL</b> (refer to <a href="#">General Data</a> )
<b>INEVT</b>	Event triggered input of data into a <b>MODUL</b> (refer to <a href="#">General Data</a> )
<b>OUTEVT</b>	Event triggered output of data from a <b>MODUL</b> (refer to <a href="#">General Data</a> )
<b>STAT</b>	Deprecated: Do not use anymore
<b>AI</b>	Analog input (refer to <a href="#">Hardware Specific Data</a> )
<b>AO</b>	Analog output (refer to <a href="#">Hardware Specific Data</a> )
<b>DI</b>	Digital input (refer to <a href="#">Hardware Specific Data</a> )
<b>DO</b>	Digital output (refer to <a href="#">Hardware Specific Data</a> )

<b>TI</b>	Timer (refer to <a href="#">Hardware Specific Data</a> )
<b>CNT</b>	Counter (refer to <a href="#">Hardware Specific Data</a> )
<b>KEY</b>	Virtual key (refer to <a href="#">Project Monitor Specific Data</a> )
<b>LOCAL</b>	Local help variable for use in <b>Project Monitor</b> (refer to <a href="#">Project Monitor Specific Data</a> ). Can only be set manually using edit dialogs.
<b>EVA</b>	Local help variable for use in <b>Project Monitor</b> (refer to <a href="#">Project Monitor Specific Data</a> ). Can only be set using <b>Stimulation Equations</b> .
<b>EVENT</b>	Not supported yet
<b>MAILBOX</b>	Not supported yet
<b>SEMA</b>	Not supported yet
<b>MSG</b>	Not supported yet
<b>RTC</b>	Data coming from the RTC of the embedded system (refer to <a href="#">Hardware Specific Data</a> )
<b>NATIVEvar</b>	Variable data without metadata (refer to <a href="#">General Data</a> )
<b>NATIVEconst</b>	Constant data without metadata (refer to <a href="#">General Data</a> )

### 5.27.2 Com Type

The following **Com Type** settings are supported:

- V<#>** Visualization (only supported for [Assign types](#) **FLAG**, **IN**, **OUT**, **IO** and **RTC**)  
V0 or just 0 deactivates communication with Project Monitor  
V1-V9 or 1 activates the communication. V1-V9 can be used for grouping the communication **ELEMENT**s for reduced communication traffic. Refer to [Communication](#) for details.
- C<#>** Cyclical communication  
C1-C9 are used for a cyclical communication in [Distributed Systems](#) (refer to [Cyclical communication](#)). Also assigns a RDI to the **ELEMENT** (refer to [radCASE Index \(RDI\)](#)).
- E<#>** Event triggered communication  
E1-E8 are used for event triggered communication in [Distributed Systems](#) (refer to [Event triggered communication](#)). Also assigns a RDI to the **ELEMENT** (refer to [radCASE Index \(RDI\)](#)).
- D<#>** Database  
D1-D9 assigns the **ELEMENT** to one of 9 internal databases (refer to [Database Storage](#)). Also assigns a RDI to the **ELEMENT** (refer to [radCASE Index \(RDI\)](#)).
- B1** Burst  
B1 marks the **ELEMENT** for burst communication (refer to [Burst](#))

- P<#>** Protocol  
P1-P9 marks the **ELEMENT** for internal target protocoling (refer to [Protocol Storage](#))
- X<#>** ASCII data export  
X1-X9 marks the **ELEMENT** for ASCII data export (refer to [ASCII Export](#))
- XE** E-Mail Element  
Marks **ELEMENT** to be included in System Event Handling (refer to [System Event Handling](#))
- S<#>** Sequence Export  
S1-S9 groups **ELEMENT** in one sequence group for selective sequence export (refer to [Sequences](#))
- A** Assign RDI  
Assigns an RDI to the **ELEMENT** (refer to [radCASE Index \(RDI\)](#))
- L<#>** List Export  
L1-L9 will create a table ElemPntTabL<#> in the file *elemPntList.c* containing all **ELEMENTS** of the according com type. The pointers are CElement\* pointing to the metadata of the **ELEMENT** (refer to [Special Element Access Operators](#)). The export of the metadata is enforced by the comtype. Also assigns a RDI to the **ELEMENT** (refer to [radCASE Index \(RDI\)](#)).

### 5.27.3 Assign string

In most of the cases the **Assign string** will simply be *CTR*. For more information on that **Assign string**, the exceptions and additional possibilities refer to [Element Usage And Allocation](#), [Assign string of Hardware specific data types](#) and [Assign string of Project Monitoring specific data types](#).

#### 5.27.3.1 Assign string of Hardware specific data types

In most cases the **Assign strings** of hardware specific data types will be the ones of the following list. Exception to this are **Assign strings** where radCASE does not make an automatic positioning in the communication buffer (refer to [Element Usage And Allocation](#)).

In the following list parameters in parenthesis like (<parameter1>,<parameter2>) are optional. Nevertheless if you want to use one of those optional parameters you have to enter all previous parameters. So if you want to change *parameter2* you have to also enter a value for *parameter1*. Parameters in square brackets like [<param>] are optional, too, but you don't have to enter previous parameters in parenthesis.

Assign type	Assign string syntax
<b>DI</b> (Digital input)	<p><i>CTR</i> &lt;Port&gt;, &lt;Port-Bit&gt;, &lt;Logic&gt;, &lt;XX&gt;, &lt;YY&gt; (,&lt;Status&gt;, &lt;HardwareID&gt;)</p> <p><b>Port:</b> Port number/name (has to correspond to the HAL)</p> <p><b>Port-Bit:</b> Bit number of the port</p> <p><b>Logic:</b> Behavior between hardware status and logical status</p> <p><i>NINV:</i> Hardware = 1 ⇨ logic = 1</p> <p><i>INV:</i> Hardware = 1 ⇨ logic = 0 (or vice versa)</p>



Assign type	Assign string syntax
	<p><b>XX, YY:</b> Position in the data block, usually "0" (provided automatically)</p> <p><b>Status:</b> Currently not used (still available due to backwards compatibility)</p> <p><b>HardwareID:</b> Used in combination with <a href="#">Distributed Systems</a> or with simple IO-Boards. Identifies the subnode or IO-Board containing the input.</p>
<b>DO</b> (Digital output)	<p><i>CTR &lt;Port&gt;, &lt;Mirror&gt;, &lt;Port-Bit&gt;, &lt;Logic&gt;, &lt;XX&gt;, &lt;YY&gt; (,&lt;HardwareID&gt;)</i></p> <p><b>Port:</b> Port number/name (has to correspond to the HAL)</p> <p><b>Mirror:</b> Currently not used (still available due to backwards compatibility)</p> <p><b>Port-Bit:</b> Bit number of the port</p> <p><b>Logic:</b> Behavior between hardware status and logical status</p> <p><i>NINV:</i> Hardware = 1 <math>\Rightarrow</math> logic = 1</p> <p><i>INV:</i> Hardware = 1 <math>\Rightarrow</math> logic = 0 (or vice versa)</p> <p><b>XX, YY:</b> Position in the data block, usually "0" (provided automatically)</p> <p><b>HardwareID:</b> Used in combination with <a href="#">Distributed Systems</a> or with simple IO-Boards. Identifies the subnode or IO-Board containing the input.</p>
<b>AI</b> (Analog input)	<p><i>CTR &lt;Index&gt;, &lt;XX&gt;, &lt;KPL&gt;, &lt;KPH&gt;, &lt;KLL&gt;, &lt;KLH&gt; (,&lt;Mode&gt;, &lt;Filter&gt;, &lt;HardwareID&gt;)</i></p> <p><b>Index:</b> Hardware index</p> <p><b>XX:</b> Position in the data block, usually "0" (provided automatically)</p> <p><b>KPL:</b> lower physical standard calibration value</p> <p><b>KPH:</b> upper physical standard calibration value</p> <p><b>KLL:</b> lower (belonging to <b>KPL</b>) logical standard calibration value; The value is in the virtual floating point format (refer to <a href="#">Virtual Floating Point</a>)</p> <p><b>KLH:</b> upper (belonging to <b>KPH</b>) logical standard calibration value; The value is in the virtual floating point format (refer to <a href="#">Virtual Floating Point</a>)</p> <p><b>Mode:</b> Convert mode (ADC specific usage), usually "0"</p> <p><b>Filter:</b> Filter level for averaging, with the following behavior:</p> $((\text{<previous element value>} * \text{<F>}) + \text{<current input value>}) / \text{<F+1>}$ <p>Filter = 0 deactivates the averaging.</p> <p><b>HardwareID:</b> Used in combination with <a href="#">Distributed Systems</a> or with simple IO-Boards. Identifies the subnode or IO-Board containing the input.</p>
<b>AO</b> (Analog output)	<p><i>CTR &lt;Index&gt;, &lt;XX&gt;, &lt;KPL&gt;, &lt;KPH&gt;, &lt;KLL&gt;, &lt;KLH&gt; (, &lt;HardwareID&gt;)</i></p> <p><b>Index:</b> Hardware index</p> <p><b>XX:</b> Position in the data block, usually "0" (provided automatically)</p> <p><b>KPL:</b> lower physical standard calibration value</p> <p><b>KPH:</b> upper physical standard calibration value</p> <p><b>KLL:</b> lower (belonging to <b>KPL</b>) logical standard calibration value; The value is in the virtual floating point format (refer to <a href="#">Virtual Floating Point</a>)</p> <p><b>KLH:</b> upper (belonging to <b>KPH</b>) logical standard calibration value; The value is in the virtual floating point format (refer to <a href="#">Virtual Floating Point</a>)</p> <p><b>HardwareID:</b> Used in combination with <a href="#">Distributed Systems</a> or with simple IO-Boards. Identifies the subnode or IO-Board containing the input.</p>
<b>CNT</b> (Impulse counter)	<p><i>CTR &lt;Index&gt;, &lt;XX&gt;, &lt;KPL&gt;, &lt;KPH&gt;, &lt;KLL&gt;, &lt;KLH&gt; (,&lt;HardwareID&gt;) [,&lt;Rem&gt;]</i></p> <p><b>Index:</b> Hardware index</p>



Assign type	Assign string syntax
	<b>XX</b> : Position in the data block, usually "0" (provided automatically) <b>KPL</b> : lower physical standard calibration value <b>KPH</b> : upper physical standard calibration value <b>KLL</b> : lower (belonging to <b>KPL</b> ) logical standard calibration value; The value is in the virtual floating point format (refer to <a href="#">Virtual Floating Point</a> ) <b>KLH</b> : upper (belonging to <b>KPH</b> ) logical standard calibration value; The value is in the virtual floating point format (refer to <a href="#">Virtual Floating Point</a> ) <b>HardwareID</b> : Used in combination with <a href="#">Distributed Systems</a> or with simple IO-Boards. Identifies the subnode or IO-Board containing the input. <b>Rem</b> : Enter an "R" here to mark the <b>ELEMENT</b> as remanent (refer to <a href="#">Remanent Data</a> ).
<b>TI</b> (Timer)	<i>CTR</i> <Index>, <XX>, <Direction>, <Resolution> [, <Rem>]  <b>Index</b> : Timer Index, usually "0" (provided automatically) <b>XX</b> : Position in the data block, usually "0" (provided automatically) <b>Direction</b> : Counting direction ("UP" or "DOWN" for increment or decrement) <b>Resolution</b> : Timer resolution in seconds (e.g. 0.01 = 10 ms-Timer). <b>Rem</b> : Enter an "R" here to mark the <b>ELEMENT</b> as remanent (refer to <a href="#">Remanent Data</a> ).

### 5.27.3.2 Assign string of Middleware data types

Beside the direct access to the internal I/O-hardware it is more and more common to use a middleware to support internal I/Os but also I/Os in distributed systems. radCASE supports an interface to the middleware "Gamma" from "RST" in combination with the element types **AIs**, **AOs**, **DIs**, **DOs** and **CNTs**.

To do this, the **HAL** has to support Gamma and the usage of Gamma has to be enabled in the project settings (refer to the Integration-Manual). To specify a Gamma-element the keyword **CTR** has to be replaced by the keyword **GAMMA**. The **Port** or **Index** has to be replaced by the "alias" used by Gamma to identify the element.

The following lines are a part of a Gamma configuration file "System.XML", that defines the single In- and Out-Elements in Gamma by using an "Alias-Name". The other parts of a Gamma Configuration file depends on the type and structure of the used hardware. For more refer to the Gamma documentation.

```

.....
<ioMap>
  .....
  <input action="Action" channel="<Chanal-Name>" io="<API-Name>" alias="<Alias-Name>"/>
  .....
  <output action="Action" channel="<Chanal-Name>" io="<API-Name>" alias="<Alias-Name>"/>
  .....
</ioMap>

```

The following example (using an emBRICK API with emBRICK I/O-Modules) lists the Gamma config lines and the corresponding radCASE assign string (remind: not all attributes are used and set to 0; also mind the "Space" between "GAMMA" and ":").

For DI:

```
<input action="Action" channel="2Rel4Di2Ai_0.DIn0" io="EmbrickEasyAPI"
      alias="Embrick.Memory.Group.Module_2Rel4Di2Ai_0.DI[0]"/>
```

```
GAMMA_:Embrick.Memory.Group.Module_2Rel4Di2Ai_0.DI[0],0,NINV,0,0
```

For DO:

```
<output action="Action" channel="2Rel4Di2Ai_0.Out0" io="EmbrickEasyAPI"
      alias="Embrick.Memory.Group.Module_2Rel4Di2Ai_0.DO[0]"/>
```

```
GAMMA_:Embrick.Memory.Group.Module_2Rel4Di2Ai_0.DO[0],0,0,NINV,0,0
```

For AI / CNT:

```
<input action="Action" channel="2Rel4Di2Ai_0.DIn0" io="EmbrickEasyAPI"
      alias="Embrick.Memory.Group.Module_2Rel4Di2Ai_0.AI[0]"/>
```

```
GAMMA_:Embrick.Memory.Group.Module_2Rel4Di2Ai_0.AI[0],<XX>, <KPL>, <KPH>,
      <KLL>, <KLH> [,<Mode>, <Filter>, <HardwareID>]
```

For AO:

```
<output action="Action" channel="2Rel4Di2Ai_0.Out0" io="EmbrickEasyAPI"
      alias="Embrick.Memory.Group.Module_2Rel4Di2Ai_0.AO[0]"/>
```

```
GAMMA_:Embrick.Memory.Group.Module_2Rel4Di2Ai_0.AO[0],<XX>, <KPL>, <KPH>,
      <KLL>, <KLH> [,<HardwareID>]
```

### 5.27.3.3 Assign string of Project Monitoring specific data types

For **Project Monitoring** specific data types the **Assign string** has the following special syntax:

Assign type	Assign string syntax
KEY	Key code of virtual key (refer to <a href="#">Virtual Keyboard Support</a> )
LOCAL	No Assign string required
EVA	Mathematical equation. The <b>Stimulation Equation</b> (refer to <a href="#">Stimulation Equations</a> ) for <b>EVA ELEMENT</b> s should be in the <b>Assign String</b> and not the <b>Simulation</b> string.

### 5.27.4 Format string

The **Format string** defines different **ELEMENT** attributes. Multiple attributes can be entered separated by a space. An attribute has the syntax: `<attribute>=<value>` where string values are to be put in quotation marks.

The following attributes are supported:

- PL** Password Level:  
Level required for editing the **ELEMENT** (0 ... 9)
  
- EI** Extended Information (only for **Assign type PAR** or **SYS**):  
The extended information will be sent as an ID to the target system after changing the value of the **ELEMENT** in the Project Monitor. The HAL has to support the extended information and can call functionality which has to be run after changing the **ELEMENT**. It is possible to provide the same ID to multiple **ELEMENTS**.
  
- EV** Enable Value (only for [Enumerative Elements](#)):  
Activates the enabling/disabling of selections (refer to [Enabling/Disabling Selections](#)).
  
- EX** Export with element structure  
Activates the export of an element with element structure like [Ctr](#)-Setting **CE=1** but only for the current **ELEMENT**.
  
- LV** Low Value (only for [Numerical Elements](#)):  
Overwrites the low value from the **TypeDef** with a constant value. For syntax of constant low values refer to the syntax explanations for the regarding data type (refer to [Data Modeling](#)). On **SURFACE\_CTR** it is also possible to use a reference to another element, by using `$(<SubmodulPath>.)<ElementName>` (refer to [Element Access](#)) or the global access operator (refer to [Global access \(\\$~\)](#)).
  
- HV** High Value (only for [Numerical Elements](#)):  
Overwrites the high value from the **TypeDef** with a constant value. For syntax of constant high values refer to the syntax explanations for the regarding data type (refer to [Data Modeling](#)). On **SURFACE\_CTR** it is also possible to use a reference to another element, by using `$(<SubmodulPath>.)<ElementName>` (refer to [Element Access](#)) or the global access operator (refer to [Global access \(\\$~\)](#)).
  
- LA** Low Alarm (only for [Numerical Elements](#)):  
Overwrites the low alarm value from the **TypeDef** with a constant value. For syntax of constant low alarm values refer to the syntax explanations for the regarding data type (refer to [Data Modeling](#)).
  
- HA** High Alarm (only for [Numerical Elements](#)):  
Overwrites the high alarm value from the **TypeDef** with a constant value. For syntax of constant high alarm values refer to the syntax explanations for the regarding data type (refer to [Data Modeling](#)).
  
- SV** Standard Value:  
Overwrites the standard value from the **TypeDef** with a constant value. For syntax of constant standard values refer to the syntax explanations for the regarding data type (refer to [Data Modeling](#)).  
For [Numerical Elements](#) on **SURFACE\_CTR** it is also possible to use a reference to another element, by using `$(<SubmodulPath>.)<ElementName>` (refer to [Element Access](#)) or the

global access operator (refer to [Global access \(\\$~\)](#)).

### 5.27.5 XCOM-type

The XCom Typ is intended to use with the Webspace modul and the Com Type L<#>. Therefore, the following option are possible:

Basic setup: WEB, 0, 0, 0

Description: WEB, 1, 2, 3

- Web-Communication activ
- From userlevel 1 readable (0...15 - 0=Everyone can read; 15=No one can read)
- From userlevel 2 writable (0...15 - 0=Everyone can write; 15=No one can write)
- Threshold is 3 (0 -> no threshold)

In context with the Com Typ L<#>, it is possible to create element lists, which will send to the Cloud.

### 5.28 SURFACE\_XXX

Surfaces define different HMI's for target system, **web visualization** and **Project Monitor**. Refer to [HMI](#) for detailed information. Also note, not all surface items are supported on all surface types. Refer to [Surface item support](#) for more information.

The following attributes are available:

<b>Surface name/number</b>	The identifier of the surface. For <b>SURFACE_CTR</b> s only numbers are allowed.
<b>Description</b>	Multilingual description of surface
<b>Position X/Y</b>	Position of the displayed window. Only <b>SURFACE_VIS</b> and <b>SURFACE_PRINT</b> .
<b>Size X/Y</b>	Size of the displayed window on opening. Only for <b>SURFACE_VIS</b> and <b>SURFACE_PRINT</b> .
<b>Foreground color</b>	Global foreground color of surface. Not supported on <b>SURFACE_CTR</b> and <b>SURFACE_PRINT</b> .
<b>Background color</b>	Global background color of surface. Not supported on <b>SURFACE_CTR</b> and <b>SURFACE_PRINT</b> . On <b>SURFACE_ICON</b> only RC-Items with color <b>GLOBALBK</b> are affected.
<b>Topmost window</b>	If this option is enabled the window of the surface stays on top, even if another window is selected. Only for <b>SURFACE_VIS</b> .
<b>Not resizable</b>	If this option is enabled, the window size of the surface cannot be changed. Only for <b>SURFACE_VIS</b> .
<b>Automatically open during start</b>	If this option is enabled the window will be automatically opened during start of <b>Project Monitor</b> . Only for <b>SURFACE_VIS</b> .

**Doc. Level** Minimum documentation level in the range from 0-9 to add surface to documentation (refer to [Structure Of Generated Documentation](#)). Only for **SURFACE\_VIS** for **SURFACE\_CTR** refer to [DT\\_ENTRY](#).

The following child RC-Items are available:

Refer to [Surface Items](#)

## 5.29 DOCTAB

Enables and defines the appearance of a **SURFACE\_CTR** in the generated documentation. Refer to [Defining Target HMI For Documentation](#) for details.

The following attributes are available:

**Elements** List of **ELEMENT**s defined by this **DOCTAB**, the values are set in the **DT\_ENTRY**s

The following child RC-Items are available:

[DT\\_ENTRY](#) Entry for **DOCTAB** settings

## 5.30 DT\_ENTRY

Defines appearance of a **SURFACE\_CTR** in the generated documentation. Has to be used in conjunction with [DOCTAB](#). Refer to [Defining Target HMI For Documentation](#) for details.

The following attributes are available:

**Heading** Multilingual heading used as page title and heading in the generated HTML page

**HTML number** Number which will be at the end of the generated filename for this entry

**Page number** Currently not supported. Should be set to 0

**Line** Values of the **ELEMENT**s defined in the **DOCTAB**.

**Doc. Level** Minimum documentation level in the range from 0-9 to add surface to documentation.

## 5.31 FULL

A **FULL** integrates another surface in the current one. Refer to [Surface Navigation](#) for more details.

The following attributes are available:

**Module** **MODUL** path to **MODUL** containing the surface to integrate. Can be left empty for surfaces in the same **MODUL**. Also a global search using ~ is possible.

<b>Surface number</b>	Surface name/number of surface to integrate
<b>Position X/Y</b>	Position to insert integrated surface
<b>Size X/Y</b>	Not supported yet
<b>Position (name) X/Y</b>	Not supported on <b>SURFACE_CTR</b> . Position to insert the surface description relative to insertion position.
<b>Format string</b>	Not supported yet

### 5.32 ICON

An **ICON** integrates a **SURFACE\_ICON** into the current **SURFACE\_VIS** for opening another **SURFACE\_VIS**. Refer to [Surface Navigation](#) for more details.

The following attributes are available:

<b>Module</b>	<b>MODUL</b> path to <b>MODUL</b> containing the <b>SURFACE_ICON</b> to integrate. Can be left empty for surfaces in the same <b>MODUL</b> .
<b>Surface name</b>	Surface name of surface to open
<b>Position X/Y</b>	Position to insert <b>ICON</b>
<b>Position (name) X/Y</b>	Position to insert the surface description relative to insertion position.
<b>Format string</b>	Used for password protection of <b>ICON</b> . Use <b>PL=&lt;password level&gt;</b> to set required password level to open surface using this <b>ICON</b> .

### 5.33 TEXT

Display of a multilingual static text. Refer to [Text Handling](#) for more details.

The following attributes are available:

<b>Text</b>	Multilingual text to display
<b>Position X/Y</b>	Position of text (depending on horizontal alignment)
<b>Rotation</b>	Not supported on <b>SURFACE_CTR</b> . Rotation of the text in degrees. Only 0, 90, 180, 270 allowed.
<b>Font Size</b>	Font size. Refer to <a href="#">Font Handling</a>
<b>Foreground/background color</b>	Foreground and background color of text
<b>Horizontal alignment</b>	Alignment of text:

Left: Insertion point is upper left corner  
 Center: Insertion point is upper middle  
 Right: Insertion point is upper right corner

**Font specification** Font specification. Refer to [Font Handling](#)

### 5.34 IPIC

Reference to a Picture. Refer to [Graphic Handling](#) for details.

The following attributes are available:

**Pict ID** Reference to a [PICT](#)

**Position X/Y** Position to insert 0,0 point of **PICT**

### 5.35 MENU

A **MENU** is a line oriented user menu. Refer to [Menus](#) for details.

The following attributes are available:

**Top left corner X/Y** Position of top left corner of menu

**Right bottom corner X/Y** Position of bottom right corner of menu

**Cursor type** Definition of menu properties (hex entry possible). Refer to [Menus](#)

**Font size/spezification** Font size/specification. Refer to [Font Handling](#)

**Foreground/background color** Foreground and background color of menu

**Selected Item Fore-ground/background color** Foreground and background color of the highlighted menu line. Refer to [Menus](#) for more details.

The following child RC-Items are available:

[AMENU](#) menu entry that triggers an action

[EDIT](#) menu entry to display/edit an **ELEMENT**

[TEXT](#) Static text

[LINE](#) Line

[IPIC](#) Reference to **PICT**

[RECT](#) Rectangle

**AICON** Icon that reacts on keyboard and touch to trigger actions

**COLUMNBREAK** Separator between two columns

**IF, ENDIF, ELSE** display RC-Items conditionally

### 5.36 AMENU

An **AMENU** is a line in a **MENU** which will react on selecting it by performing actions (refer to [Action Handling](#)).

The following attributes are available:

<b>Title</b>	Multilingual text to display for the MENU line
<b>Position X/Y</b>	Position of text (depending on horizontal alignment)
<b>Foreground/background color</b>	Foreground and background color of text
<b>Font specification</b>	Font specification. Refer to <a href="#">Font Handling</a>
<b>Font Size</b>	Font size. Refer to <a href="#">Font Handling</a>
<b>Horizontal alignment</b>	Alignment of text: Left: Insertion point is upper left corner Center: Insertion point is upper middle Right: Insertion point is upper right corner

The following child RC-Items are available:

Refer to [Actions](#)

### 5.37 EDIT

The RC-Item **EDIT** is used for visualizing single **ELEMENT**s within a **MENU**. Many of the attributes are only available on specific visualizers; refer to [Element Visualization](#) for more details.

The following attributes are available:

<b>Name</b>	Name of the element to be displayed (refer to <a href="#">Element Access</a> ). The global access operator (refer to <a href="#">Global access (\$~)</a> ) is also available. Additionally the default value or for <b>ENUM</b> s the minimum and maximum range can be displayed, by appending <code>-&gt;setup</code> , <code>-&gt;minRange</code> or <code>-&gt;maxRange</code> at the end of the element name (e.g. <code>elem-&gt;minRange</code> )
<b>Display type</b>	Visualizer to be used (refer to <a href="#">Visualizers For Single Elements</a> )
<b>Editor type</b>	Settings for editing the values: <b>Without:</b> Editing and selecting not possible <b>Dialog:</b> Opening a dialog box



	<p><b>Toggle:</b> Direct change of value on clicking (spin box control or increasing element)</p> <p><b>Permanent:</b> Not supported in <b>MENUs</b></p> <p><b>Focus:</b> No editing possible, but item is still selectable</p>
<b>Position X/Y</b>	Display position. Normally used to specify the position of the element value
<b>Size</b>	Visualizer specific size in format X, Y. Refer to according visualizer description selected of selected <b>Display type</b> .
<b>Position description</b>	Position of <b>ELEMENT</b> description relative to <b>Position X/Y</b> in format X, Y
<b>Position Value</b>	Visualizer specific position of value in format X, Y. Refer to according visualizer description of selected <b>Display type</b> .
<b>Position Scaling</b>	Visualizer specific position of scaling in format X, Y. Refer to according visualizer description of selected <b>Display type</b> .
<b>Number of lines/columns</b>	Visualizer specific number of lines/columns. Refer to according visualizer description of selected <b>Display type</b> .
<b>Horizontal alignment</b>	Visualizer specific horizontal alignment. Refer to according visualizer description of selected <b>Display type</b> .
<b>Foreground/background color</b>	Foreground and background color of visualizer
<b>Text direction</b>	Visualizer specific text direction. Refer to according visualizer description of selected <b>Display type</b> .
<b>Mask Bin</b>	Only available on <b>SURFACE_VIS</b> for <b>EBINs</b> . A mask for selecting which Selections of the <b>EBIN</b> are available during edit. For explanations on how to do this on a <b>SURFACE_CTR</b> refer to <a href="#">Enabling/Disabling Selections</a>
<b>Subtext selection</b>	<p>Defines which Subtext will be used on a <b>SURFACE_CTR</b>. Subtexts can be used for description of the <b>ELEMENT</b>, value of the <b>ELEMENT</b> (<b>EBINs</b>) and unit of the <b>ELEMENT</b> by entering a key and the number of the chosen subtext. Multiple keys can be separated by colon.</p> <p>Keys are:</p> <p><i>D</i> = Description</p> <p><i>V</i> = Value</p> <p><i>U</i> = Unit</p> <p>For more information refer to <a href="#">Subtexts</a></p>
<b>Password level</b>	Password level needed for editing the <b>ELEMENT</b>
<b>Show value</b>	If set the Value of the <b>ELEMENT</b> is displayed
<b>Show description</b>	If set the description of the <b>ELEMENT</b> is displayed

<b>Show unit</b>	Visualizer specific activating of unit display. Refer to according visualizer description of selected <b>Display type</b> .
<b>No space between value and unit</b>	If set the unit will be directly attached to the value and there is no space between them. Value and Unit has to be displayed for this
<b>Show leading zeros</b>	If set for <b>ENUMs</b> always the full number of digits is displayed
<b>Show scale</b>	Visualizer specific activating of scale. Refer to according visualizer description of selected <b>Display type</b> .
<b>Hexdecimal values</b>	If set values are displayed hexadecimal

### 5.38 COLUMNBREAK

The RC-Item **COLUMNBREAK** is used for separating columns within a [MENU](#)

### 5.39 ELEM

The RC-Item **ELEM** is used for visualizing single **ELEMENTs**. Many of the attributes are only available on specific visualizers; refer to [Element Visualization](#) for more details.

The following attributes are available:

<b>Name</b>	Name of the element to be displayed (refer to <a href="#">Element Access</a> ). The global access operator (refer to <a href="#">Global access (\$~)</a> ) is also available. Additionally the default value or for <b>ENUMs</b> the minimum and maximum range can be displayed, by appending <i>-&gt;setup</i> , <i>-&gt;minRange</i> or <i>-&gt;maxRange</i> at the end of the element name (e.g. <i>elem-&gt;minRange</i> )
<b>Display type</b>	Visualizer to be used (refer to <a href="#">Visualizers For Single Elements</a> )
<b>Editor type</b>	Settings for editing the values: <b>Without:</b> Editing and selecting not possible <b>Dialog:</b> Opening a dialog box <b>Toggle:</b> Direct change of value on clicking (spin box control or increasing element) <b>Permanent:</b> Sets <b>ELEMENT</b> to 1 while clicking <b>Focus:</b> No editing possible, but item is still selectable
<b>Position X/Y</b>	Display position. Normally used to specify the position of the element value
<b>Size</b>	Visualizer specific size in format X, Y. Refer to according visualizer description selected of selected <b>Display type</b> .
<b>Position description</b>	Position of <b>ELEMENT</b> description relative to <b>Position X/Y</b> in format X,Y
<b>Position Value</b>	Visualizer specific position of value in format X, Y. Refer to according visualizer description of selected <b>Display type</b> .

<b>Position Scaling</b>	Visualizer specific position of scaling in format X, Y. Refer to according visualizer description of selected <b>Display type</b> .
<b>Number of lines/columns</b>	Visualizer specific number of lines/columns. Refer to according visualizer description of selected <b>Display type</b> .
<b>Horizontal alignment</b>	Visualizer specific horizontal alignment. Refer to according visualizer description of selected <b>Display type</b> .
<b>Foreground/background color</b>	Foreground and background color of visualizer
<b>Text direction</b>	Visualizer specific text direction. Refer to according visualizer description of selected <b>Display type</b> .
<b>Mask Bin</b>	Only available on <b>SURFACE_VIS</b> for <b>EBINs</b> . A mask for selecting which Selections of the <b>EBIN</b> are available during edit. For explanations on how to do this on a <b>SURFACE_CTR</b> refer to <a href="#">Enabling/Disabling Selections</a>
<b>Subtext selection</b>	Defines which Subtext will be used on a <b>SURFACE_CTR</b> . Subtexts can be used for description of the <b>ELEMENT</b> , value of the <b>ELEMENT (EBINs)</b> and unit of the <b>ELEMENT</b> by entering a key and the number of the chosen subtext. Multiple keys can be separated by colon. Keys are: <i>D</i> = Description <i>V</i> = Value <i>U</i> = Unit For more information refer to <a href="#">Subtexts</a>
<b>Password level</b>	Password level needed for editing the <b>ELEMENT</b>
<b>Rotation</b>	Visualizer specific rotation. Refer to according visualizer description of selected <b>Display type</b> .
<b>Show value</b>	If set the Value of the <b>ELEMENT</b> is displayed
<b>Show description</b>	If set the description of the <b>ELEMENT</b> is displayed
<b>Show unit</b>	Visualizer specific activating of unit display. Refer to according visualizer description of selected <b>Display type</b> .
<b>No space between value and unit</b>	If set the unit will be directly attached to the value and there is no space between them. Value and Unit has to be displayed for this
<b>Show leading zeros</b>	If set for <b>ENUMs</b> always the full number of digits is displayed
<b>Show scale</b>	Visualizer specific activating of scale. Refer to according visualizer description of selected <b>Display type</b> .
<b>Hexdecimal values</b>	If set values are displayed hexadecimal

## 5.40 MElem

The RC-Item **MElem** is used for visualizing multiple [ELEMENT](#)s. Refer to [Element Visualization](#) for more details.

The following attributes are available:

<b>Number of elements</b>	Number of <b>ELEMENT</b> s to be displayed by this <b>MElem</b> . Maximum supported number of <b>ELEMENT</b> s is 15.
<b>List of elements</b>	Comma separated list of <b>ELEMENT</b> s to be displayed. For referencing the <b>ELEMENT</b> s refer to <a href="#">Element Access</a> . The global access operator (refer to <a href="#">Global access (\$~)</a> ) is also available.
<b>Display type</b>	Visualizer to be used (refer to <a href="#">Visualizers For Multiple Elements</a> )
<b>Position X/Y</b>	Display position
<b>Options</b>	Format string (refer to <a href="#">MElem Formatstring</a> )

### 5.40.1 MElem Formatstring

The following display properties (**Options**) are available as format string of a [MElem](#). The Options are separated by spaces and are in the format `<option>=<value>`:

<b>YS</b>	Y-Spacing	Defines the distance in pixels between the graph of <a href="#">Histogram Visualizers</a> and the scrollbar. Standard value: 30.
<b>SV</b>	Size Visualizer	Visualizer specific size of visualizer in format X, Y. Refer to according visualizer description of selected <b>Display type</b> .
<b>CT</b>	Color table	Visualizer specific colors for different element values. The values are comma separated names of the colors (e.g. <i>WHITE</i> , <i>RED</i> , <i>LGRAY</i> ...) hex values are not allowed. Refer to according visualizer description of selected <b>Display type</b> .
<b>BK</b>	Background color	Background color of the visualizer. The value is the name of the color (e.g. <i>WHITE</i> , <i>RED</i> , <i>LGRAY</i> ...) hex values are not allowed.
<b>NS</b>	Number Scalelines	Number of labels on the axes in <a href="#">Histogram Visualizers</a> in format X, Y excluding the zero point.
<b>NG</b>	Number Grids	Number of grid lines in <a href="#">Histogram Visualizers</a> in format X, Y excluding the axes.
<b>LG</b>	Legend	Activates a legend in <a href="#">Histogram Visualizers</a> . Standard value: 0.
<b>BO</b>	EBin-Offset	Defines spacing between the first <a href="#">EB ENTRY</a> of an <a href="#">EBIN</a> to the X-Axis of <a href="#">Histogram Visualizers</a> . The value is in pixel with the standard value: 0.
<b>MD</b>	Mode Display	Display setting mask of <a href="#">Histogram Visualizers</a> : 0x80: Automatic zoom in Playback (Zooming will not work and should not be displayed) 0x200: Show scrollbar and Zoom-Buttons for X-Axis

		0x400: Show scrollbar and Zoom Buttons for Y-Axis
		0x800: Time axis uses relative time to start of recording instead of absolute time values.
		0x4000: Remove time information (labels on X-Axis and whole time)
		0x8000: Remove Zoom factor and <b>ELEMENT</b> selection
<b>DI</b>	Direction	Direction of <a href="#">Multiple Bar Visualizers</a> : H for horizontal V for vertical
<b>MR</b>	Movement range	Movement range of <a href="#">VIS ROT XY Visualizers</a> in format <i>X1,Y1;X2,Y2;...</i>
<b>RR</b>	Rotation range	Rotation range of <a href="#">VIS ROT XY Visualizers</a> in format <i>Min1,Max1;Min2,Max2;...</i>

## 5.41 DISPLAY

A **DISPLAY** is a visualizer to visualize the target HMI (refer to [HMI](#)) in a **SURFACE\_VIS**. It allows to remote control a target and to simulate the target HMI (refer to [Display communication](#)).

The following attributes are available:

<b>ID</b>	Identifier of the target display
<b>Position X/Y</b>	Position of visualizer
<b>Zoom</b>	Zoom factor to enlarge small target displays on a <b>SURFACE_VIS</b> for better readability on a PC screen.

## 5.42 AICON

An **AICON** is a combination of an [AKEY](#) and a [TouchKey](#). Like an **AKEY** the **AICON** will react on a key (refer to [Keyboard Handling](#)) by performing actions (refer to [Action Handling](#)). Additional to this the **AICON** will also react on a touch event (refer to [Touch Support](#)).

The following attributes are available:

<b>Key</b>	Key value (refer to <a href="#">Keyboard Handling</a> for valid values). Not used within <a href="#">MENU</a> . Can be left empty in that case.
<b>Pict ID</b>	Foreground picture drawn over the background image. The size of this picture represents the dimension of the touch active area.
<b>Pict ID (pressed)</b>	Background picture drawn when <b>AICON</b> is activated by Touch
<b>Pict ID (released)</b>	Background picture drawn when <b>AICON</b> is not activated by Touch. The size of this picture represents the dimension of the touch active area.
<b>Position X/Y</b>	Position to draw the <b>AICON</b>

**Comment** Multilingual comment

**Text** Multilingual text to display in the foreground (on top of background pictures). The font and size will be selected automatically. If more control over the font selection is needed, the text should be placed as [TEXT](#) in the [PICT](#)s used above.



If both attributes, the **Pict ID** and the **Pict ID (released)**, are defined the touch active area is a combination of both ones and will lead into different dimensions of touch active areas (refer to picture [touch active areas](#) ).

The following child RC-Items are available:

Refer to [Actions](#)

### 5.43 SET

**SET** is an action which manipulates **ELEMENT**s (refer to [Element Manipulating Actions](#)). **SET** sets the value of the **ELEMENT** to 1.

The following attributes are available:

**Element** Reference to **ELEMENT**. Refer to [Element Access](#). The global access operator (refer to [Global access \(\\$~\)](#)) is also available.

### 5.44 RES

**RES** is an action which manipulates **ELEMENT**s (refer to [Element Manipulating Actions](#)). **RES** sets the value of the **ELEMENT** to 0.

The following attributes are available:

**Element** Reference to **ELEMENT**. Refer to [Element Access](#). The global access operator (refer to [Global access \(\\$~\)](#)) is also available.

### 5.45 INV

**INV** is an action which manipulates **ELEMENT**s (refer to [Element Manipulating Actions](#)). **INV** inverts the value of the **ELEMENT** from 0 to 1 and vice versa.

The following attributes are available:

**Element** Reference to **ELEMENT**. Refer to [Element Access](#). The global access operator (refer to [Global access \(\\$~\)](#)) is also available.

### 5.46 PUT

**PUT** is an action which manipulates **ELEMENT**s (refer to [Element Manipulating Actions](#)). **PUT** sets the value of the **ELEMENT** to a specific value.

The following attributes are available:

**Element** Reference to **ELEMENT**. Refer to [Element Access](#). The global access operator (refer to [Global access \(\\$~\)](#)) is also available.

**Value** Constant value with \$-Access (refer to [Access To Element Related Constants \(\\$\)](#))

## 5.47 INC

**INC** is an action which manipulates **ELEMENT**s (refer to [Element Manipulating Actions](#)). **INC** increases the value of the **ELEMENT** by 1.

The following attributes are available:

**Element** Reference to **ELEMENT**. Refer to [Element Access](#). The global access operator (refer to [Global access \(\\$~\)](#)) is also available.

## 5.48 DEC

**DEC** is an action which manipulates **ELEMENT**s (refer to [Element Manipulating Actions](#)). **DEC** decreases the value of the **ELEMENT** by 1.

The following attributes are available:

**Element** Reference to **ELEMENT**. Refer to [Element Access](#). The global access operator (refer to [Global access \(\\$~\)](#)) is also available.

## 5.49 AEDIT

**AEDIT** is an action which manipulates **ELEMENT**s (refer to [Element Manipulating Actions](#)). **AEDIT** calls the edit dialog of the **ELEMENT**.

The following attributes are available:

**Element** Reference to **ELEMENT**. Refer to [Element Access](#). The global access operator (refer to [Global access \(\\$~\)](#)) is also available.

## 5.50 COPY

**COPY** is an action which manipulates **ELEMENT**s (refer to [Element Manipulating Actions](#)). **COPY** copies the value of an **ELEMENT** to another one.

The following attributes are available:

**Source element** Reference to source **ELEMENT**. Refer to [Element Access](#). The global access operator (refer to [Global access \(\\$~\)](#)) is also available.

**Destination element** Reference to destination **ELEMENT**. Refer to [Element Access](#). The global access operator (refer to [Global access \(\\$~\)](#)) is also available.

## 5.51 GOTOSURFACE

**GOTOSURFACE** is an action (refer to [Action Handling](#)) which is used for surface navigation (refer to [Surface Navigation](#)). **GOTOSURFACE** changes the current surface in the current **MODUL**, without storing the old surface onto the surface stack.

The following attributes are available:

**Surface number** Surface number of surface to change to

## 5.52 CALLSURFACE

**CALLSURFACE** is an action (refer to [Action Handling](#)) which is used for surface navigation (refer to [Surface Navigation](#)). **CALLSURFACE** changes the current surface in the current **MODUL** and stores the old surface onto the surface stack.

The following attributes are available:

**Page number** Surface number of surface to change to

## 5.53 RETURN (Sur)

**RETURN** is an action (refer to [Action Handling](#)) which is used for surface navigation (refer to [Surface Navigation](#)). **RETURN** restores a stored value from the surface stack.

## 5.54 OPEN

**OPEN** is an action (refer to [Action Handling](#)) which is used for surface navigation (refer to [Surface Navigation](#)). **OPEN** changes the current surface and **MODUL** and stores the old surface onto the surface stack.

The following attributes are available:

**Submodule** Reference to **MODUL** (refer to [MODUL access](#)) to open the surface in (can be left blank for opening in the same **MODUL**).

**Surface number** Surface number of surface to change to

## 5.55 CLOSE

**CLOSE** is an action (refer to [Action Handling](#)) which is used for surface navigation (refer to [Surface Navigation](#)). **CLOSE** restores a stored value from the surface stack.

## 5.56 CFUNC

**CFUNC** calls predefined system functions.

The following attributes are available:



**C function**      Predefined system function to call. Refer to [C function](#) for a list of supported C-Functions.

### 5.56.1 C function

The following predefined system functions are supported:

<b>DIAG_DI</b>	Open diagnosis dialog of digital inputs (diagnosis dialog can be customized, refer to <a href="#">Customizing Standard System Dialogs</a> )
<b>DIAG_DO</b>	Open diagnosis dialog of digital outputs (diagnosis dialog can be customized, refer to <a href="#">Customizing Standard System Dialogs</a> )
<b>DIAG_AI</b>	Open diagnosis dialog of analog inputs (diagnosis dialog can be customized, refer to <a href="#">Customizing Standard System Dialogs</a> )
<b>DIAG_AO</b>	Open diagnosis dialog of analog outputs (diagnosis dialog can be customized, refer to <a href="#">Customizing Standard System Dialogs</a> )
<b>DIAG_CNT</b>	Open diagnosis dialog of counters (diagnosis dialog can be customized, refer to <a href="#">Customizing Standard System Dialogs</a> )
<b>KALI_AI</b>	Open calibration dialog of analog inputs (calibration dialog can be customized, refer to <a href="#">Customizing Standard System Dialogs</a> )
<b>KALI_AO</b>	Open calibration dialog of analog outputs (calibration dialog can be customized, refer to <a href="#">Customizing Standard System Dialogs</a> )
<b>KALI_CNT</b>	Open calibration dialog of counters (calibration dialog can be customized, refer to <a href="#">Customizing Standard System Dialogs</a> )
<b>PROTOCOL</b>	Not supported anymore
<b>RTC_SET</b>	Setting of Real Time Clock (the dialogs used for setting RTC can be customized, refer to <a href="#">Customizing Standard System Dialogs</a> )
<b>PAR_SETUP</b>	Reset parameters to standard values
<b>PAR_COPY</b>	Copy parameters to edit copy (refer to <a href="#">Working Copy And Edit Copy</a> )
<b>PAR_RESTORE</b>	Save values of edit copy to parameters (refer to <a href="#">Working Copy And Edit Copy</a> )
<b>SYS_SETUP</b>	Reset system parameters to standard values
<b>SYS_COPY</b>	Copy system parameters to edit copy (refer to <a href="#">Working Copy And Edit Copy</a> )
<b>SYS_RESTORE</b>	Save values of edit copy to system parameters (refer to <a href="#">Working Copy And Edit Copy</a> )
<b>CALI_SETUP</b>	Reset calibration data to standard values
<b>CALI_COPY</b>	Copy calibration data to edit copy (refer to <a href="#">Working Copy And Edit Copy</a> )

<b>CALI_RESTORE</b>	Save values of edit copy to calibration data (refer to <a href="#">Working Copy And Edit Copy</a> )
<b>DB_LOAD</b>	Not supported anymore
<b>CARD_WR_SYS</b>	Chip card: Write system parameters to card
<b>CARD_RD_SYS</b>	Chip card: Read system parameters from card
<b>CARD_WR_PAR</b>	Chip card: Write parameters to card
<b>CARD_RD_PAR</b>	Chip card: Read parameters from card
<b>CARD_WR_CAL</b>	Chip card: Write calibration data to card
<b>CARD_RD_CAL</b>	Chip card: Read calibration data from card
<b>CARD_FORMAT</b>	Chip card: Format card
<b>KONF_PROT</b>	Not supported anymore
<b>CARD_DIR</b>	Chip card: Show card contents
<b>PWD_PROT</b>	Not supported anymore
<b>DI_IDENT</b>	Identification of digital inputs. Will output the digital input which is set to 1
<b>DO_EIN/AUS</b>	Toggle all digital outputs. Two modes can be toggled with keyboard (ESC to exit): Toggling all outputs at once, Toggling one output after the other.
<b>Reset Outputs</b>	Set all digital and analog outputs to 0
<b>RtcTimeSet</b>	Set system time of RTC (the dialogs used for setting RTC can be customized, refer to <a href="#">Customizing Standard System Dialogs</a> )
<b>RtcDateSet</b>	Set system date of RTC (the dialogs used for setting RTC can be customized, refer to <a href="#">Customizing Standard System Dialogs</a> )
<b>Unforce</b>	Disable force mode (refer to <a href="#">Force Mode</a> )
<b>User redraw</b>	Redraw surface

## 5.57 USERFUNC

Deprecated, do not use anymore. Use [PROCEDURE](#) instead.

## 5.58 PROCEDURE

**PROCEDURE** calls a [PROC](#) in the model. For more information on the usage and requirements of **PROCEDURE** refer to [Functionality Calling Actions](#).

The following attributes are available:

**Procedure call** Call of the PROC (refer to [Behavior Access](#))

**Description** Further descriptive information

## 5.59 PASSWORD

**PASSWORD** checks for a required password level and if necessary shows a password dialog. Refer to [Access Control Actions](#) for more information.

The following attributes are available:

**Level** Required password level to perform following actions

**Jump distance** Number of actions that should be left out, when password level is not sufficient

## 5.60 PSWD\_CLR

**PSWD\_CLR** clears the current password level. Refer to [Access Control Actions](#) for more information.

## 5.61 ASKOK

**ASKOK** opens a query dialog to ask the user to continue or abort the next actions. Refer to [User Query Actions](#) for more information.

The following attributes are available:

**Type of question** Query type:  
1: Critical Function!  
2: Execute?  
3: Save Changes? (default)  
4: Delete?  
5: Forcing active – Release ? (this type is currently only available for Displays 320\*240 or larger)

**Jump distance** Number of actions that should be left out, when Abort was selected

## 5.62 AKEY

An **AKEY** will react on a key (refer to [Keyboard Handling](#)) by performing actions (refer to [Action Handling](#)).

The following attributes are available:

**Key** Key value (refer to [Keyboard Handling](#) for valid values)

**Comment** Multilingual comment

The following child RC-Items are available:

Refer to [Actions](#)

### 5.63 AKEYGLOBAL

Like [AKEY](#) an **AKEYGLOBAL** will react on a key (refer to [Keyboard Handling](#)) by performing actions (refer to [Action Handling](#)), but **AKEYGLOBAL** will be triggered globally from all surfaces and will jump to a central surface. Refer to [Global Keys](#) for more information on **AKEYGLOBAL**.

The following attributes are available:

**Key** Key value (refer to [Keyboard Handling](#) for valid values)

**Comment** Multilingual comment

The following child RC-Items are available:

Refer to [Actions](#)

### 5.64 AENTER

An **AENTER** will execute actions (refer to [Action Handling](#)) when entering a surface.

The following child RC-Items are available:

Refer to [Actions](#)

### 5.65 APERM

An **APERM** will execute actions (refer to [Action Handling](#)) permanently while the surface is active.

The following child RC-Items are available:

Refer to [Actions](#)

### 5.66 AEXIT

An **AEXIT** will execute actions (refer to [Action Handling](#)) when leaving a surface.

The following child RC-Items are available:

Refer to [Actions](#)

### 5.67 ActionCond

An **ActionCond** will execute actions (refer to [Action Handling](#)) when a condition is true.

The following attributes are available:

<b>Variable</b>	Name of <b>ELEMENT</b> to check in condition (refer to <a href="#">Element Access</a> ). The global access operator (refer to <a href="#">Global access (\$~)</a> ) is also available.
<b>Condition</b>	Condition to check. The condition has the same syntax like described in <a href="#">Conditional Drawing</a>
<b>Update surface if condition is fulfilled</b>	If activated the surface will be completely redrawn, as soon as the condition is met.

The following child RC-Items are available:

Refer to [Actions](#)

## 5.68 TouchKey

A **TouchKey** is an icon that will react on a touch event (refer to [Touch Support](#)) by creating a keyboard event (refer to [Keyboard Handling](#))

The following attributes are available:

<b>Key</b>	Key value (refer to <a href="#">Keyboard Handling</a> for valid values)
<b>Pict ID</b>	Foreground picture drawn over the background image. The size of this picture represents the dimension of the touch active area.
<b>Pict ID (pressed)</b>	Background picture drawn when <b>TouchKey</b> is activated by Touch
<b>Pict ID (released)</b>	Background picture drawn when <b>TouchKey</b> is not activated by Touch. The size of this picture represents the dimension of the touch active area.
<b>Position X/Y</b>	Position to draw the <b>TouchKey</b>
<b>Comment</b>	Multilingual comment
<b>Text</b>	Multilingual text to display in the foreground (on top of background pictures). The font and size will be selected automatically. If more control over the font selection is needed, the text should be placed as <a href="#">TEXT</a> in the <a href="#">PICTs</a> used above.



If both attributes, the **Pict ID** and the **Pict ID (released)**, are defined the touch active area is a combination of both ones. The following picture shows seven different combinations of **Pict ID** and **Pict ID (released)** that will lead into different touch active areas, symbolized by framed red rectangles.

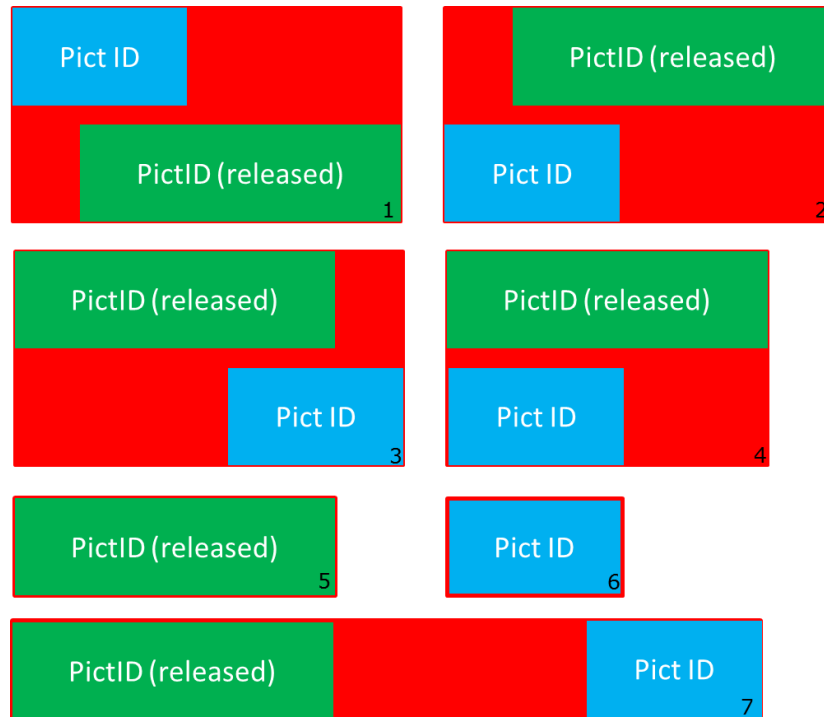


Figure 1, touch active areas

## 5.69 IF, ENDIF, ELSE

**IF**, **ELSE** and **ENDIF** can be used to create conditional surfaces (refer to [Conditional Drawing](#)), to only draw different RC-Items on the surface when specific conditions are met.

The following attributes are available:

<b>Variable</b>	Name of <b>ELEMENT</b> to check in condition (refer to <a href="#">Element Access</a> ). The global access operator (refer to <a href="#">Global access (\$~)</a> ) is also available.
<b>Condition</b>	Condition to check. The syntax is described in <a href="#">Conditional Drawing</a>
<b>Update surface if condition is fulfilled</b>	If activated the surface will be completely redrawn, as soon as the condition is met.

## 5.70 SETPOS

Used to define a new starting point for relative positioning (refer to [Positioning](#)). The position can be specified using an absolute coordinate or an **ELEMENT** for position. If mixing absolute position with an **ELEMENT** the sum of both is used.

The following attributes are available:

<b>Elem-X/Y</b>	Reference to an <b>ELEMENT</b> . If an <b>ELEMENT</b> is referenced the value of that <b>ELEMENT</b> is added to the specified position as new starting point.
-----------------	--

**PosX/Y** An absolute value which is added to the **ELEMENT** value (if any) to specify the new starting point.

### 5.71 RECT

Draws a rectangle. Also refer to [Graphic Handling](#)

The following attributes are available:

<b>Position X/Y</b>	Position of upper left corner
<b>Size X/Y</b>	Size of rectangle
<b>Pen size</b>	Border width
<b>Fill/border style</b>	Different styles of rectangle appearance
<b>Radius (only extended style)</b>	Radius of rounded corners (Valid values: 0...9)
<b>Brightness gradient X/Y</b>	Specifies brightness gradient in percent (Valid values -100...100)
<b>3D Effect</b>	If set the rectangle will get a 3D-effect
<b>Frames</b>	Select which edges of the rectangle will get a brighter border
<b>Color</b>	Color of the rectangle (border and fill if any)

### 5.72 CIRC

Draws a circle or ellipse. Also refer to [Graphic Handling](#)

The following attributes are available:

<b>Center X/Y</b>	Center of the circle/ellipse
<b>Radius X/Y</b>	Radius of circle/ellipse
<b>Pen size</b>	Border width
<b>Fill mode</b>	If set to 1 the circle/ellipse is filled
<b>Color</b>	Color of the circle

### 5.73 LINE

A LINE is a graphical object which draws a line. Refer also to [Graphic Handling](#).

The following attributes are available:

<b>Start position X/Y</b>	Start position of the line
<b>End position X/Y</b>	End position of the line
<b>Pen size</b>	Line width
<b>Line style</b>	Style of the line
<b>Color</b>	Color of the line
<b>Stard/End Design</b>	Style of the line end (only <b>SURFACE_VIS</b> )

## 5.74 FOLDER

Not yet supported.

## 5.75 CARD

Not yet supported.

## 5.76 FILL

Fills an area with a specified color.

The following attributes are available:

<b>Color</b>	Color to fill the area with
<b>Position X/Y</b>	Position of the area to fill

## 5.77 METHODS

Contains functionality of a **MODUL**. Refer to [Behavior Modeling](#)

The following attributes are available:

<b>Language</b>	Programming language to use in functionalities. Currently only C supported.
<b>Global declarations</b>	Declarations of functions and variables. Can be used to make global external functions and variables known to the code in the <b>METHODS</b> container, or to make global functions or variables in the <b>METHODS</b> container known to other code.
<b>Global code</b>	Can be used to define global functions or variables. The usage of \$-Access to access <b>ELEMENT</b> s (refer to <a href="#">Element Access</a> ) or functionality (refer to <a href="#">Behavior Access</a> ) is possible. Keep in mind there are some limitations within multiple instances (refer to <a href="#">Using Global Code</a> )

The following child RC-Items are available:



**PROC**

C-function

**SEQUENCE DIAGRAM**

Definition of functionality with sequence diagrams

**5.78 PROC**

A **PROC** is a C-function defined in a **MODUL**, which is accessible using \$-Access (refer to [Behavior Access](#)). A **PROC** is one method to realize behavior in a radCASE model (refer to [Behavior Modeling](#)).

The following attributes are available:

<b>Function name</b>	Name of the function
<b>Interface type</b>	Used for structural UML diagrams (refer to the according editor manual of the editor you are using)
<b>Processing type</b>	The <b>Processing type</b> of the function defining the scheduling of it (refer to <a href="#">Scheduling</a> )
<b>Return type</b>	C-data type returned by the function.
<b>Arguments</b>	The arguments passed to the function. The arguments must be comma separated and surrounded with parenthesis. Arguments may only be used for <b>Processing type LOCAL</b> or <b>PUBLIC</b> . If not used must be <i>(void)</i> .
<b>Info</b>	<p>Further information, will be generated as Comment of the function in generated code. The comment will be generated in Doxygen-Format and every line of the Info will be preceded with an “*”, so the following info-Text example will result in a valid Doxygen-Comment in generated code:</p> <pre> *brief Short description  *Long description; May be multiple lines  \param[in] param1 Description of Parameter 1 *return Description of return code </pre>
<b>Code</b>	Containing the code of the function, has to contain surrounding curly brackets

**5.79 SEQUENCE DIAGRAM**

A **SEQUENCE DIAGRAM** is a way to graphically realize behavior in a radCASE model using UML sequence diagrams. (Refer to [Behavior Modeling](#)). A **SEQUENCE DIAGRAM** is accessible using \$-Access (refer to [Behavior Access](#)).

The following attributes are available:

<b>Sequence</b>	Name of the sequence function
<b>Interface type</b>	Used for structural UML diagrams (refer to the according editor manual of the editor you are using)

<b>Processing type</b>	The <b>Processing type</b> of the function defining the scheduling of it (refer to <a href="#">Scheduling</a> )
<b>Return type</b>	C-data type returned by the sequence.
<b>Arguments</b>	The arguments passed to the sequence. The arguments must be comma separated and surrounded with parenthesis.
<b>Info</b>	Further information
<b>Code</b>	C-Code which is executed before the sequence functionality. E.g. for declaring local variables within the sequence.

The following child RC-Items are available:

<a href="#"><b>MESSAGE</b></a>	Call of functionality
<a href="#"><b>IF, ELSE IF, ELSE, END IF</b></a>	Conditional section
<a href="#"><b>CODE (Seq)</b></a>	C code to execute
<a href="#"><b>LOOP, END LOOP</b></a>	Loop section
<a href="#"><b>RETURN (Seq)</b></a>	Exit sequence and return value
<a href="#"><b>REFERENCE</b></a>	Incorporate functionality

## 5.80 MESSAGE

A **MESSAGE** calls functionality out of a [SEQUENCE DIAGRAM](#).

The following attributes are available:

<b>Message</b>	Name of functionality to call
<b>Type</b>	Only <b>Synchron</b> and <b>Asynchron</b> supported. Synchron calls the function directly and waits for the return. Asynchron triggers the execution of the function at a later point in time.
<b>Arguments</b>	The arguments passed to the function. The arguments must be comma separated and surrounded with parenthesis.

## 5.81 IF, ELSE IF, ELSE, END IF

**IF, ELSE IF, ELSE** and **END IF** can be used to execute parts of the sequence of a [SEQUENCE DIAGRAM](#) conditionally. For every **IF** there has to be an **END IF**, **ELSE IF** and **ELSE** are optional. **IF, ELSE IF, ELSE** and **END IF** can be nested.

The following attributes are available:

**Condition** Condition when to execute the appropriate parts of the sequence. Any C-expression can be used.

## 5.82 CODE (Seq)

**CODE** contains C-Code which is executed in the sequence of a [SEQUENCE DIAGRAM](#).

The following attributes are available:

**Info** Some informational text to the **CODE** sequence

**Action/Code** The C-Code which is executed.

## 5.83 LOOP, END LOOP

A **LOOP** can be used to loop around some code in a [SEQUENCE DIAGRAM](#) as long as a condition is met. For every **LOOP** there has to be an **END LOOP**. **LOOPS** can be nested.

The following attributes are available:

**Condition** Condition for defining how long the **LOOP** will run around the nested parts of the sequence. Any C-expression can be used.

## 5.84 RETURN (Seq)

**RETURN** can be used to exit a sequence of a [SEQUENCE DIAGRAM](#) and return a return value.

The following attributes are available:

**Return value** Return value which is returned, has to match the **Return type** of the **SEQUENCE DIAGRAM**.

## 5.85 REFERENCE

A **REFERENCE** will incorporate some functionality into a sequence of a [SEQUENCE DIAGRAM](#). The code of that functionality will be inserted inline into the sequence.

The following attributes are available:

**Reference** The name of the functionality to incorporate

## 5.86 STATECHARTS

In **STATECHARTS** there are different State Machines defined (refer to [Finite State Machines](#))

The following child RC-Items are available:

[MACHINE](#) Definition of a state machine

## 5.87 MACHINE

A **MACHINE** is the definition of a State Machine (refer to [Finite State Machines](#)).

The following attributes are available:

<b>Name</b>	Name of the State Machine function
<b>Processing type</b>	The <b>Processing type</b> of the function defining the scheduling of it (refer to <a href="#">Scheduling</a> )
<b>Remanent</b>	If activated the State Machine is remanent (refer to <a href="#">Remanent Data</a> )
<b>Arguments</b>	The arguments passed to the state machine. The arguments can be used in all C-Code within the statemachine, but may only be used in <b>Processing type LOCAL</b> or <b>PUBLIC</b> . The arguments must be comma separated and surrounded with parenthesis. If not used may be left empty or use <i>(void)</i> instead.
<b>Description</b>	Description of the State Machine
<b>Info</b>	Further information on the State Machine
<b>Code</b>	Global C-Code which can be used to define global variables or functions

The following child RC-Items are available:

<a href="#"><u>STATE</u></a>	A state of the State Machine
<a href="#"><u>UNITSTATE</u></a>	<b>STATE</b> containing <b>STATES</b>
<a href="#"><u>TRANSITION</u></a>	Transition to other <b>STATES</b>
<a href="#"><u>DURING</u></a>	Code to execute permanently while in <b>STATE</b>
<a href="#"><u>CHOICE</u></a>	Choose between different target <b>STATES</b>
<a href="#"><u>ENDSTATE</u></a>	Final <b>STATE</b>
<a href="#"><u>SUBMACHINE</u></a>	Incorporate another <b>MACHINE</b>
<a href="#"><u>ENTRY POINT</u></a>	Entry point of <b>SUBMACHINE</b>
<a href="#"><u>EXIT POINT</u></a>	Exit point of <b>SUBMACHINE</b>

## 5.88 STATE

A **STATE** of a State Machine (refer to [Finite State Machines](#)).

The following attributes are available:

<b>Name</b>	Name of the <b>STATE</b>
<b>Type</b>	Type of the <b>STATE</b> . <b>Deep History</b> not supported yet.
<b>Background color</b>	Defines the color of the <b>STATE</b> on displaying the State Machine
<b>Description</b>	Description of the <b>STATE</b>
<b>Info</b>	Further information on the <b>STATE</b>

The following child RC-Items are available:

<u><a href="#">ENTER</a></u>	Code to execute on entering <b>STATE</b>
<u><a href="#">DURING</a></u>	Code to execute permanently while in <b>STATE</b>
<u><a href="#">TRANSITION</a></u>	Transition to other <b>STATES</b>
<u><a href="#">EXIT</a></u>	Code to execute on exiting <b>STATE</b>

### 5.89 ENTER

**ENTER** defines the code executed on entering a **STATE** (refer to [Finite State Machines](#)).

The following attributes are available:

<b>Description</b>	Description of the code
<b>Code</b>	C-Code to execute when entering the <b>STATE</b>

### 5.90 EXIT

**EXIT** defines the code executed on exiting a **STATE** (refer to [Finite State Machines](#)).

The following attributes are available:

<b>Description</b>	Description of the code
<b>Code</b>	C-Code to execute when exiting the <b>STATE</b>

### 5.91 UNITSTATE

A **UNITSTATE** is a special **STATE** of a State Machine (refer to [Finite State Machines](#)). It is a **STATE** which contains **STATES** itself (refer to [Unit States](#)).

The following attributes are available:

<b>Name</b>	Name of the <b>UNITSTATE</b>
<b>Type</b>	Type of the <b>UNITSTATE</b> . <b>Deep History</b> not supported yet.
<b>Background color</b>	Defines the color of the <b>UNITSTATE</b> on displaying the State Machine
<b>Description</b>	Description of the <b>UNITSTATE</b>
<b>Info</b>	Further information on the <b>UNITSTATE</b>

The following child RC-Items are available:

<a href="#"><u>ENTER</u></a>	Code to execute on entering <b>STATE</b>
<a href="#"><u>DURING</u></a>	Code to execute permanently while in <b>STATE</b>
<a href="#"><u>EXIT</u></a>	Code to execute on exiting <b>STATE</b>
<a href="#"><u>STATE</u></a>	A state of the State Machine
<a href="#"><u>UNITSTATE</u></a>	<b>STATE</b> containing <b>STATES</b>
<a href="#"><u>TRANSITION</u></a>	Transition to other <b>STATES</b>
<a href="#"><u>SUBMACHINE</u></a>	Incorporate another <b>MACHINE</b>

## 5.92 TRANSITION

A **TRANSITION** defines the conditions when the State Machine changes from one **STATE** to another (refer to [Finite State Machines](#)).

The following attributes are available:

<b>Label</b>	Name of the <b>TRANSITION</b>
<b>Destination state</b>	<b>STATE</b> to change to
<b>Background color</b>	Defines the color of the <b>TRANSITION</b> on displaying the State Machine
<b>Guard</b>	Condition that has to be met, to change the <b>STATE</b> . Any C-expression is valid.
<b>Show guard</b>	Only for displaying purposes in editor (refer to the according editor manual of the editor you are using)
<b>Action</b>	C-Code which is executed on changing the <b>STATE</b>

### 5.93 DURING

A **DURING** defines C-Code which is executed permanently as long as the current **STATE** is active (refer to [Finite State Machines](#)).

The following attributes are available:

**Description** Description of the code

**Code** C-Code to execute while **STATE** is active

### 5.94 CHOICE

**CHOICE** is a pseudo **STATE** which realizes a conditional branch (refer to [Finite State Machines](#)).

The following attributes are available:

**Name** Name of the **CHOICE**

**Description** Description of the **CHOICE**

The following child RC-Items are available:

[TRANSITION](#) Transition to other **STATES**

### 5.95 ENDSTATE

The final **STATE** of a State Machine (refer to [Finite State Machines](#)).

### 5.96 SUBMACHINE

A **SUBMACHINE** incorporates another State Machine into a State Machine (refer to [Finite State Machines](#) and [Submachines](#)).

The following attributes are available:

**Name** Name of the **SUBMACHINE**

**Submachine** Name of the State Machine to incorporate

**Background color** Defines the color of the **SUBMACHINE** on displaying the State Machine

The following child RC-Items are available:

[TRANSITION](#) Transition to other **STATES**

[ENTRY POINT](#) Entry point of the **SUBMACHINE**

[EXIT POINT](#) Exit point of the **SUBMACHINE**

## 5.97 ENTRY POINT

**ENTRY POINT** defines an entry point of a **SUBMACHINE** (refer to [Finite State Machines](#) and [Submachines](#)).

The following attributes are available:

**Point name** Name of the **ENTRY POINT**

The following child RC-Items are available:

[TRANSITION](#) Transition to other **STATES**

## 5.98 EXIT POINT

**EXIT POINT** defines an exit point of a **SUBMACHINE** (refer to [Finite State Machines](#) and [Submachines](#)).

The following attributes are available:

**Point name** Name of the **EXIT POINT**

The following child RC-Items are available:

[TRANSITION](#) Transition to other **STATES**

## 5.99 ACTIVITYCHARTS

**ACTIVITYCHARTS** are a way to graphically realize behavior in a radCASE model using UML activity charts. (Refer to [Behavior Modeling](#)).

The following child RC-Items are available:

[ACTIVITY](#) An activity chart

## 5.100 ACTIVITY

An **ACTIVITY** is one activity chart and a way to graphically realize behavior in a radCASE model using UML activity charts. (Refer to [Behavior Modeling](#)). An **ACTIVITY** accessible using \$-Access (refer to [Behavior Access](#)).

The following attributes are available:

**Name** Name of the activity function

**Processing type** The **Processing type** of the function defining the scheduling of it (refer to [Scheduling](#))

**Description** Description of the activity chart



<b>Info</b>	Additional information on the activity chart
<b>Code</b>	Global C-Code which can be used to define global variables or functions

The following child RC-Items are available:

**START** Starting point of **ACTIVITY**

**END** End point of **ACTIVITY**

**ACTION** Action to perform

**BRANCH** Conditional execution

**LABEL** not yet supported

### 5.101 **START**

**START** defines where to start the processing of an **ACTIVITY**. If no **START** is defined the first defined RC-Item within the **ACTIVITY** is used as starting point.

The following child RC-Items are available:

**CONTROLFLOW** Connect RC-Items of an **ACTIVITY**

### 5.102 **END**

**END** defines the end of processing of an **ACTIVITY**. **END** is not mandatory, because execution stops automatically if a node is reached without a **CONTROLFLOW** which is executed.

### 5.103 **ACTION**

An **ACTION** contains C-Code which is executed when the **ACTION** is reached during processing of an **ACTIVITY**.

The following attributes are available:

<b>Name</b>	Name of the <b>ACTION</b>
<b>Type</b>	Not supported yet
<b>Background color</b>	Defines the color of the <b>ACTION</b> on displaying the Activity chart
<b>Description</b>	Description of the <b>ACTION</b>
<b>Info</b>	Further information on the <b>ACTION</b>

The following child RC-Items are available:

**CODE (Act)** C code to execute

**CONTROLFLOW** Connect RC-Items of an **ACTIVITY**

#### 5.104 BRANCH

A **BRANCH** contains different conditional **CONTROLFLOW**s to let the **ACTIVITY** branch its execution flow.

The following attributes are available:

**Name** Name of the **BRANCH**

**Description** Description of the **BRANCH**

The following child RC-Items are available:

**CONTROLFLOW** Connect RC-Items of an **ACTIVITY**

#### 5.105 LABEL

Not yet supported.

#### 5.106 CONTROLFLOW

A **CONTROLFLOW** connects different RC-Items of an **ACTIVITY** and controls the execution flow. If multiple **CONTROLFLOW**s are in an RC-Item, the first **CONTROLFLOW** with matching **Guard** will be executed.

The following attributes are available:

**Label** Name of the **CONTROLFLOW**

**Destination state** Name of the RC-Item to connect to and to execute next

**Background color** Defines the color of the **CONTROLFLOW** on displaying the Activity chart

**Guard** Condition that has to be met, to execute the connected RC-Item. Any C-expression is valid. If the **Guard** is left empty the **CONTROLFLOW** will always be executed.

**Show guard** Only for displaying purposes in editor (refer to the according editor manual of the editor you are using)

**Action** Not yet supported

### 5.107 CODE (Act)

**CODE** contains C-Code to be executed in an [ACTIVITY](#).

The following attributes are available:

<b>Description</b>	Description of the C-Code
<b>Code</b>	C-Code to execute

### 5.108 SIGNAL\_CHART

A **SIGNAL\_CHART** is a data flow oriented programming method within radCASE (refer to [Signal Diagrams](#)).

The following attributes are available:

<b>Name</b>	Name of the <b>SIGNAL_CHART</b>
<b>Processing type</b>	The <b>Processing type</b> of the function defining the scheduling of it (refer to <a href="#">Scheduling</a> )
<b>Description</b>	Multilingual description of <b>SIGNAL_CHART</b>

The following child RC-Items are available:

<a href="#"><u>TEXT</u></a>	Static text
<a href="#"><u>IPIC</u></a>	Reference to <b>PICT</b>
<a href="#"><u>SETPOS</u></a>	Position for relative positioning of following RC-Items
<a href="#"><u>RECT</u></a>	Rectangle
<a href="#"><u>CIRC</u></a>	Circle
<a href="#"><u>LINE</u></a>	Line
<a href="#"><u>FILL</u></a>	
<a href="#"><u>ELEM</u></a>	Display value of <b>ELEMENT</b>
<a href="#"><u>SignalIcon</u></a>	Refer to a <b>SIGNAL_ICON</b>
<a href="#"><u>Connection</u></a>	Connect <b>SignalIcons</b> and <b>ELEMs</b>

### 5.109 SignalIcon

A **SignalIcon** is a reference to a [SIGNAL\\_ICON](#) in a [SIGNAL\\_CHART](#) (refer to [Signal Diagrams](#)).

The following attributes are available:

<b>Module</b>	Name of the Signal <b>MODUL</b> the <b>SignalIcon</b> references to.
<b>Surface name</b>	<b>Surface name</b> of the <b>SIGNAL_ICON</b>
<b>Position X/Y</b>	Position to display the <b>SIGNAL_ICON</b>
<b>Position (name) X/Y</b>	Position to display the description of the Signal <b>MODUL</b> instance
<b>Format</b>	Currently not supported

### 5.110 Connection

A **Connection** connects two **ELEM**s in a [SIGNAL\\_CHART](#) (refer to [Signal Diagrams](#)).

The following attributes are available:

<b>Source</b>	Reference to <b>ELEMENT</b> to copy data from. Refer to <a href="#">Element Access</a>
<b>Destination</b>	Reference to <b>ELEMENT</b> to copy data to. Refer to <a href="#">Element Access</a>
<b>Line width</b>	Width of connection line for graphical display
<b>Line color</b>	Color of the line for graphical display

### 5.111 SIGNAL\_ICON

A **SIGNAL\_ICON** is a graphical icon of a Signal **MODUL** (refer to [Signal Diagrams](#)) and an interface to it.

The following attributes are available:

<b>Surface name</b>	Name of the <b>SIGNAL_ICON</b>
<b>Description</b>	Multilingual description
<b>Position X/Y</b>	Not supported yet
<b>Size X/Y</b>	Not supported yet
<b>Foreground/background color</b>	Not supported yet
<b>Topmost window</b>	Not supported yet
<b>Not resizable</b>	Not supported yet
<b>Automatically open during start</b>	Not supported yet

**Background image** Not supported yet

The following child RC-Items are available:

<a href="#"><u>DOCTAB</u></a>	Not supported in <b>SIGNAL_ICON</b>
<a href="#"><u>FULL</u></a>	Not supported in <b>SIGNAL_ICON</b>
<a href="#"><u>ICON</u></a>	Not supported in <b>SIGNAL_ICON</b>
<a href="#"><u>TEXT</u></a>	Static text
<a href="#"><u>IPIC</u></a>	Reference to <b>PICT</b>
<a href="#"><u>MENU</u></a>	Not supported in <b>SIGNAL_ICON</b>
<a href="#"><u>ELEM</u></a>	Display/edit value of <b>ELEMENT</b>
<a href="#"><u>MElem</u></a>	Display multiple <b>ELEMENT</b> s
<a href="#"><u>DISPLAY</u></a>	Not supported in <b>SIGNAL_ICON</b>
<a href="#"><u>AICON</u></a>	Not supported in <b>SIGNAL_ICON</b>
<a href="#"><u>AKEY</u></a>	Not supported in <b>SIGNAL_ICON</b>
<a href="#"><u>AKEYGLOBAL</u></a>	Not supported in <b>SIGNAL_ICON</b>
<a href="#"><u>AENTER</u></a>	Not supported in <b>SIGNAL_ICON</b>
<a href="#"><u>APERM</u></a>	Not supported in <b>SIGNAL_ICON</b>
<a href="#"><u>AEXIT</u></a>	Not supported in <b>SIGNAL_ICON</b>
<a href="#"><u>ActionCond</u></a>	Not supported in <b>SIGNAL_ICON</b>
<a href="#"><u>TouchKey</u></a>	Not supported in <b>SIGNAL_ICON</b>
<a href="#"><u>IF, ENDIF, ELSE</u></a>	display RC-Items conditionally
<a href="#"><u>SETPOS</u></a>	Position for relative positioning of following RC-Items
<a href="#"><u>RECT</u></a>	Rectangle
<a href="#"><u>CIRC</u></a>	Circle
<a href="#"><u>LINE</u></a>	Line

**FOLDER** not yet supported

**FILL** Fill an area on a surface in **Project Monitor**

### 5.112 PORT

A **PORT** is a logical interface to a [MODUL](#). **PORTs** can help to logically structure a **MODUL**.

The following attributes are available:

**Name** The name of the **PORT**

The following child RC-Items are available:

**ELEMENT** Instantiation of a **TYPEDEF**

**PROC** C-function

### 5.113 ARTEFACT

**ARTEFACT** for documentation (refer to [Artefact documentation](#))

The following attributes are available:

**Name** Name of artefact

**Doc. Level** Minimum documentation level required for **ARTEFACT** to be documented (refer to [Documentation Generation](#))

**Type** [ARTEFACTDEF](#) of the current **ARTEFACT**

... Additional attributes according to **ARTEFACTDEF**.

### 5.114 RELATION

A **RELATION** documents the relation between different [SUBMODULs](#). It is part of Structural UML diagrams used for project analysis (refer to [Project Analysis](#)).

The following attributes are available:

**Class 1 (source)** Name of the **SUBMODUL** which is source of the **RELATION**

**Class 2 (target)** Name of the **SUBMODUL** which is the target of the **RELATION**

**Type** Type of **RELATION**

**Source label** Description text of the source **SUBMODUL**

**Target label**      Description text of the target **SUBMODUL**

**Relation name**      Name of the **RELATION**

### 5.115      **PICTDEF**

Contains pictures of the model

The following child RC-Items are available:

[PICT](#)      Picture definition

### 5.116      **PICT**

A picture which can be displayed on different surfaces.

The following attributes are available:

**Pict ID**      Identifier of the picture used to reference the picture

The following child RC-Items are available:

[SETPOS](#)      Position for relative positioning of following RC-Items

[RECT](#)      A rectangle

[CIRC](#)      A circle

[ARC](#)      An arc

[LINE](#)      A line

[CBITMAP](#)      Bitmap only usable on target hardware

[WBITMAP](#)      Bitmap only usable in **Project Monitor**

[TEXT](#)      A static text

[IPIC](#)      A reference to a **PICT**

[FILL](#)      Fill an area on a surface in **Project Monitor**

### 5.117      **ARC**

Draws an arc

The following attributes are available:

<b>Center X/Y</b>	Center position (also refer to <a href="#">Positioning</a> )
<b>Radius X/Y</b>	Radius of the arc
<b>Start/end angle</b>	Angle in degree to start/end the arc. 0° is on the right side. A positive value moves the point in counter-clock direction and a negative value in clockwise direction. The arc is always drawn in counter-clock direction. So the start and end angle has to be selected accordingly.
<b>Pen size</b>	Line width of the arc.
<b>Color</b>	Color to draw with (refer to <a href="#">Color Details</a> ).

### 5.118 CBITMAP

A bitmap used on **SURFACE\_CTR**. Also refer to [Bitmap Usage](#).

The following attributes are available:

<b>Position X/Y</b>	Start position of the picture (refer to <a href="#">Positioning</a> )
<b>Size X/Y</b>	Dimensions of the bitmap (resizing of the bitmap not supported)
<b>Autosize</b>	Use the original image dimensions (disabling not supported)
<b>Filename</b>	Name and path of bitmap file (refer to <a href="#">File And Path Access</a> )

### 5.119 WBITMAP

A bitmap used in **Project Monitor**. Also refer to [Bitmap Usage](#).

The following attributes are available:

<b>Position X/Y</b>	Start position of the picture (refer to <a href="#">Positioning</a> )
<b>Size X/Y</b>	Dimensions of the bitmap (will resize the bitmap)
<b>Autosize</b>	Use the original image dimensions
<b>Filename</b>	Name and path of bitmap file (refer to <a href="#">File And Path Access</a> )

### 5.120 VISUALDEF

The **VISUALDEF** section contains customizable visualizers for **ELEMENT**s. Refer to [Element Visualization](#).

The following child RC-Items are available:



**VISBINICON** State-Visualizer for **ELEMENTs** of **EBIN** data type (e.g. LEDs)

**VIS\_LINE** Moving visualizer for **ELEMENTs** (e.g. slide control)

**VIS\_ROT** Rotating visualizer for **ELEMENTs** (e.g. control dial)

**VIS\_ROT\_XY** Moving and rotating visualizer for multiple **ELEMENTs**

### 5.121 VISBINICON

A **VISBINICON** is a graphical visualizer used to display the current status of an **EBIN** as picture. Refer to [VISBINICON Visualizers](#) for more details.

The following attributes are available:

<b>ID</b>	Visualizer ID: Used to reference the visualizer.
<b>Background image</b>	This image is displayed once on first drawing of the visualizer (for <b>SURFACE_CTR</b> it is drawn each time the value changes as Clear Image).
<b>Clear image</b>	Image used to restore area before drawing new value, drawn every time the value changes. Not supported on <b>SURFACE_CTR</b> , Background image is used instead.
<b>Image list</b>	List with images ( <b>PICTs</b> ), which visualize each status. The position of images in the list (up to down) corresponds to the status of the <b>EBIN ELEMENT</b> .

### 5.122 VIS\_LINE

A **VIS\_LINE** is a visualizer used to display the current status of an **ENUM** as slide control. For more information refer to [VIS\\_LINE Visualizers](#).

The following attributes are available:

<b>ID</b>	Visualizer ID: Used to reference the visualizer.
<b>Background image</b>	This image is displayed once on first drawing of the visualizer.
<b>Linelmng</b>	Picture of the slider that will be positioned in dependency to the value of the <b>ELEMENT</b> .

### 5.123 VIS\_ROT

A **VIS\_ROT** is a visualizer that can be used for **ENUMs** and **EBINs** (no multiselections) to display a rotating image according to the current value of the **ELEMENT**. For more information refer to [VIS\\_ROT Visualizers](#).

The following attributes are available:

<b>ID</b>	Visualizer ID: Used to reference the visualizer.
<b>Background image</b>	This image is displayed once on first drawing of the visualizer. The size of this image will be used to restore the background if the element value changes. So the size should be at least big enough to surround the whole area the rotating image will use for any rotation. At least an invisible rectangle should be used for this.
<b>RotImg</b>	Picture that will be rotated according to the value of the <b>ELEMENT</b> . In this image transparency is allowed and the image should be positioned in a manner that the rotation point of the picture is on the coordinates (0, 0).

#### 5.124 VIS\_ROT\_XY

A **VIS\_ROT\_XY** visualizer is a visualizer that will move and rotate different pictures according to [ELEMENT](#) values. It is a combination of a [VIS\\_LINE](#) and a [VIS\\_ROT](#). For more information refer to [VIS\\_ROT\\_XY Visualizers](#).

The following attributes are available:

<b>ID</b>	Visualizer ID: Used to reference the visualizer.
<b>Background image</b>	This image is displayed once on first drawing of the visualizer. The size of this image will be used to restore the background if the element value changes. So the size should be at least big enough to surround the whole area the rotating and moving image will use for any rotation and any allowed position. At least an invisible rectangle should be used for this.
<b>RotImg</b>	Picture that will be rotated and moved according to the value of the <b>ELEMENT</b> . In this image transparency is allowed and the image should be positioned in a manner that the rotation point of the picture is on the coordinates (0, 0).

#### 5.125 EntityTab

The **EntityTab** offers the possibility to edit [MODUL](#) instances (refer to [SUBMODUL](#)), their [ELEMENTs](#) and [ARTEFACTs](#) at a central location. Refer to [Using The EntityTab](#) for more information.

#### 5.126 ARTEFACTDEFS

Container for [ARTEFACTDEF](#)

The following child RC-Items are available:

[ARTEFACTDEF](#) Definition for an [ARTEFACT](#)

#### 5.127 ARTEFACTDEF

Describes a type of an [ARTEFACT](#). Refer to [Artefact documentation](#).

The following attributes are available:

<b>Type ID</b>	Type to reference from an <b>ARTEFACT</b>
<b>Type of documentation</b>	Determines how <b>ARTEFACT</b> s are documented. Currently only table supported.
<b>Sorted by</b>	Select parameter for an alphabetical sorting of the output (if parameters have the same value, the value of the next parameters will be used for alphabetical sorting)

Additionally parameters can be added, each with the following attributes:

<b>ID</b>	Identifier of the parameter
<b>Description</b>	Description of the parameter (used e.g. as table column headline).

### 5.128 Requirement

Used for collecting user requirements in form of **Usecase-Diagrams**. For how to create **Usecase-Diagrams** refer to [Usecase Diagrams](#).

The following child RC-Items are available:

<a href="#">Usecase-Diagram</a>	RC-Item for defining a <b>Usecase-Diagram</b>
---------------------------------	---

### 5.129 Usecase-Diagram

Used for creating a **Usecase-Diagram**. For in depth explanations refer to [Usecase Diagrams](#).

The following attributes are available:

<b>Name</b>	Name of the <b>Usecase diagram</b>
<b>Description</b>	Supplementary description of the <b>Usecase diagram</b>
<b>Development team</b>	Developer group processing the <b>Usecase diagram</b>
<b>Deadline</b>	Deadline on which the <b>Usecase diagram</b> should be completed
<b>Development status</b>	For details refer to <a href="#">Keeping Track Of Development Status</a> .
<b>Progress</b>	For details refer to <a href="#">Keeping Track Of Development Status</a> .
<b>Test status</b>	For details refer to <a href="#">Keeping Track Of Development Status</a> .

The following child RC-Items are available:

<a href="#">Actor</a>	User or external system interacting with the system
<a href="#">Usecase</a>	Possible interaction with the system

[Sub-Usecase-Diagram](#) Links to another **Usecase-Diagram**

### 5.130 Actor

A user or external system interacting with the system. For in depth explanations refer to [Usecase Diagrams](#).

The following attributes are available:

**Name** Name of the **Actor**

**Description** Describes the **Actor** and its role in relation with the system

The following child RC-Items are available:

[Association](#) Connection to a **Usecase**

### 5.131 Association

An **Association** defines which **Actors** and **Usecases** are interacting with each other. An **Association** can connect an **Actor** with a **Usecase**, or two **Usecases**. For in depth explanations refer to [Usecase Diagrams](#).

The following attributes are available:

**Association target** Name of **Usecase** to connect to

**Association type** Only for **Associations** between two **Usecases**. Can change the **Association type** to “**Generalize**”, “**Extend**” and “**Include**”.

### 5.132 Usecase

A **Usecase** describes a functionality that can be used in the system. For in depth explanations refer to [Usecase Diagrams](#).

The following attributes are available:

**Name** Name of the **Usecase**

**Description** Supplementary description of the **Usecase**

**Development team** Developer group processing the **Usecase**

**Deadline** Deadline on which the **Usecase** should be completed

**Development status** For details refer to [Keeping Track Of Development Status](#).

**Progress** For details refer to [Keeping Track Of Development Status](#).

**Test status** For details refer to [Keeping Track Of Development Status](#).

The following child RC-Items are available:

[Scenario](#) Links to the functionality of the **Usecase**

[Attribute](#) Links to the elements of the **Usecase**

[Association](#) Connection to another **Usecase**

### 5.133 Scenario

A **Scenario** links a **Usecase** to a **MODUL** containing the functionality of the **Usecase**. For in depth explanations refer to [Usecase Diagrams](#).

The following attributes are available:

**Name** Name of the **Scenario**

**Description** Describes the **Scenario** and its functionality

**Reference** Links to the **MODUL** covering the mentioned functionality

The following child RC-Items are available:

[ScenarioReference](#) not supported yet

### 5.134 ScenarioReference

Not yet supported.

### 5.135 Attribute

An **Attribute** links a **Usecase** to a **MODUL** containing the **Attributes** of the **Usecase**. For in depth explanations refer to [Usecase Diagrams](#).

The following attributes are available:

**Name** Name of the **Attribute**

**Description** Describes the elements of the **Attribute**

**Reference** Links to the **MODUL** containing the mentioned elements

The following child RC-Items are available:

[AttributeReference](#) not supported yet

### 5.136 [AttributeReference](#)

Not yet supported.

### 5.137 [Sub-Usecase-Diagram](#)

Links to another **Usecase-Diagram**. For in depth explanations refer to [Usecase Diagrams](#).

The following attributes are available:

**Reference** Name of **Usecase-Diagram** to link to

### 5.138 [EProfiles](#)

The **EProfiles** section contains the different energy profiles.

The following child RC-Items are available:

[EProfile](#) Resource consumption analysis energy profile

### 5.139 [EProfile](#)

An energy profile defines the current resource consumption of **ELEMENT**s with this profile. For detailed explanations of an energy profile refer to [Resource Consumption Analysis](#).

The following attributes are available:

**Name** Name of the energy profile. Used to identify the profile within an **ELEMENT**

**Desc** Multilingual description of the energy profile

**Info** Multilingual additional information on the energy profile

**Behavior** Formula for calculating the current resource consumption of an **ELEMENT**

**Group** Name of the [EGroup](#) the energy profile belongs to.

### 5.140 [EGroups](#)

The **EGroups** section contains the different energy groups.

The following child RC-Items are available:

[EGroup](#) Resource consumption analysis energy group

### 5.141 EGroup

An energy group is a group of everything that consumes a resource. For detailed explanations of an energy group refer to [Resource Consumption Analysis](#).

The following attributes are available:

<b>Name</b>	Name of the energy group. Used to identify the energy group within an <a href="#">EProfile</a> .
<b>Desc</b>	Multilingual description of energy group
<b>Info</b>	Multilingual additional information to the energy group
<b>Type</b>	Element type used for all energy consumption <b>ELEMENT</b> s within the energy group
<b>COUnit</b>	The Consumption overall unit is the unit used for the consumption overall <b>EL-ELEMENT</b> s
<b>COConversion</b>	Formula for calculating the overall consumption

## 6 Feature Details

### 6.1 Project Analysis

There are different ways to analyze a radCASE project. Most of these project analysis functionalities like Structural UML diagrams are located in the editor (refer to the according editor manual of the editor you are using) and don't have any effect on the generated code.

#### 6.1.1 Keeping Track Of Development Status

radCASE offers to keep track of the development status at different places within the model with the following settings:

<b>Development status</b>	Specifies the programming phase. The selectable values can be customized by editing the file <i>ProgState.txt</i> in the Editor directory.
<b>Progress (%)</b>	Specifies the progress in percent.
<b>Test status</b>	Specifies the testing phase. The selectable values can be customized by editing the file <i>TestState.txt</i> in the Editor directory.

#### 6.1.2 Usecase Diagrams

At the beginning of the development process requirements of the software should be collected. radCASE offers to collect those requirements in the [Requirement](#)-section of a model using UML [Usecase-Diagrams](#). For explanations on the UML specification of Usecase-Diagrams please refer to a UML Manual.

radCASE extends the UML specification by offering the possibility to keep track on the development status and testing status within the **Usecase-Diagrams** and the [Usecases](#). The **Usecases** can also be linked to the implementation using [Scenarios](#) and [Attributes](#).



Usecase-Diagrams do not affect the code generation and are only meant for the design phase of software. For UML diagrams affecting the code generation refer to [Behavior Modeling](#)

### 6.2 Project Hierarchy

Each project is divided into one main project file containing [The System MODUL](#) and any number of library files. The [MODULs](#) contained in the [MODULDEF](#) section of the main project file and the libraries contain the actual application and usually are the main content of every radCASE model.

Each **MODUL** may contain different objects:

- [SUBMODULs](#) are forming the actual project hierarchy (hierarchical instantiation of **MODULs**) with the **System MODUL** as root of that hierarchy
- [ELEMENTs](#) containing the data of the project (refer to [Data Modeling](#))



- Functionality in different forms (refer to [Behavior Modeling](#))
- User interfaces for the **Project Monitor** and the target system (refer to [HMI](#))

A **MODUL** is just the definition of all those different objects. The **MODUL** corresponds to a class in object oriented programming and no code will be created for any **MODUL**, but the **System MODUL**. A **MODUL** can inherit from another **MODUL** (refer to [Inheritance](#)) and overwrite every of the above mentioned objects. A **SUBMODUL** is an instance of a **MODUL** and corresponds to an object in object oriented programming. To initialize **SUBMODUL**s with different initial states, the **EntityTab** works as a kind of constructor for every instance in the project (refer to [Using The EntityTab](#)).

It is possible to create **SUBMODUL** arrays, which can be accessed with an index (refer to [MODUL access](#)). Also there are different **SUBMODUL** types for support of distributed systems (refer to [Distributed Systems](#)) and dynamic instantiation (refer to [Dynamic instantiation](#)).

### 6.2.1 The System MODUL

The **System MODUL** is the Root [MODUL](#) definition in the [Project Hierarchy](#). The **System MODUL** is automatically instantiated by the model compiler as **Root**. The **MODUL MRoot** contained in the *std\_system.rad* contains some basic properties and functionalities needed in every model. The **MODUL** should always be inherited by the **System MODUL** and should not be instantiated anywhere else in a radCASE project.



The **System MODUL** corresponds in a way to the main-Function of a C-Project. As thus the **System MODUL** must be exactly one time in a project. It is neither possible to have a radCASE project without a **System MODUL** nor a radCASE project with multiple **System MODUL**s.

### 6.2.2 Using The EntityTab

When using multiple instances of a [MODUL](#) or a **MODUL**-Array the [EntityTab](#) is the only way to define different standard values and descriptions for every instance of the **MODUL** itself and all [ELEMENT](#)s/[SUBMODUL](#)s and [ARTEFACT](#)s within the instance of this **MODUL**.

For example if you define a **MODUL** *MultiInst* with the **ELEMENT** *Elem*. You can enter a standard value for this **ELEMENT** using the parameter **SV=x** (refer to [Format string](#)). Now, if you create two **SUBMODUL**s of the module *MultiInst* for example *Inst1* and *Inst2* both instances are exactly the same. So *Inst1.Elem* has the same standard value like *Inst2.Elem* and if you change the standard value in the module definition of *MultiInst* this changes the standard value in all of the instances. Using the **EntityTab** you can override the standard values of each **ELEMENT** and assign a standard value for *Inst1.Elem* and a totally different for *Inst2.Elem*.

Because the **EntityTab** overrides the standard values defined in the model there are multiple use-cases for the **EntityTab**:

- Changing of **Descriptions** for usage of Placeholders (refer to [Placeholders](#))
- Changing of [Assign strings](#) to use IOs in Libraries or multiple instances
- Defining **Controller-IDs** or Communication settings in [Distributed Systems](#)
- Overriding standard values of **ELEMENT**s for each instance (refer to [Data Modeling](#))

### 6.2.3 Inheritance

radCASE supports the use of inheritance, but multiple inheritance is not supported, which means each [MODUL](#) can only inherit from one Base-**MODUL**. When inheriting from a Base-**MODUL** all contents are available in the derived **MODUL**.

Like in most object oriented languages it is also possible to overwrite the functionality of the Base-**MODUL** by creating functionality with the same name and the same function prototype including the **Processing type** (refer to [Scheduling](#)). To overwrite a functionality the type of functionality shouldn't be changed either, this means e.g. [PROC](#)s can't be overwritten with [Finite State Machines](#).

In addition to this [SUBMODUL](#)s can be overwritten, with **SUBMODUL**s of other **Submodule types**. Of course in this case the interface of both instantiated **MODUL**s should be the same. At last it is possible to overwrite the different **SURFACE\_XXX** by creating a new surface with the same **Surface number/name**.

### 6.2.4 Dynamic instantiation

[SUBMODUL](#)s with a **Submodule Type** of **Pointer** can be used to dynamically load or change [MODUL](#)s at runtime. E.g. it is possible to have 10 different **MODUL**s and to load only 5 of them depending on settings made by the user.

**SUBMODUL** pointers only work in conjunction with indistinct **MODUL**s (**SUBMODUL**s with a **Multiplicity** of 0).

Indistinct **MODUL**s are prototypes of a **MODUL** (not instances) and as those can't be accessed directly out of the model. An instance of an indistinct **MODUL** has to be generated and linked to the pointer and after that the pointer can be used to access the **SUBMODUL**.

Because the **SUBMODUL** pointer will contain instances of the indistinct **MODUL**s the **ModuleID** of the pointer has to match the ID of the indistinct **MODUL**s. This can be done, by using the same ID for pointer and indistinct **MODUL**, or by using the ID of the pointer as Base-**MODUL** (refer to [Inheritance](#)) for the indistinct **MODUL**s. In this way the functionality of different indistinct **MODUL**s can be different, but they have to have the same interface (the one of the Base-**MODUL**). Only functions contained in the Base-**MODUL** can be called externally. Functions not existing in the Base-**MODUL** can only be called from within overwritten functions or surfaces of the Base-**MODUL**.

Creating an instance of an indistinct **MODUL** and linking it to a pointer, has to be made manually within a C-Function.

First some memory has to be reserved for storing data of an indistinct **MODUL** (this is the instance of the indistinct **MODUL**), e.g. by creating a large character array for different instances.

Pointers are always created as array, so creating a pointer with a **Multiplicity** of 1 will result in an array with size 1 and the access of the pointer is always by using array syntax `$pointer[idx]` (refer to [MODUL access](#)).

Linking the instance of the indistinct **MODUL** to the **SUBMODUL** pointer can be done, by initializing the pointer\_Array structure.

```
typedef struct
{
    short                modIdx;
    short                elemtabIdx;
    void RD_MRAM         *data;
    RD_IDX_PNT_TAB RD_MROM *pnttab;
} Pnt_Array;
```

An array of this structure is created for every pointer, named *Pnt\_<path\_to\_pointer>*. Where *<path\_to\_pointer>* is the path originating from the root module (e.g. *Pnt\_Mod1\_Mod2\_PntMod*). For nested pointers the path will always go over the pointer and not over an indistinct **MODUL**. So the path would be e.g. *Pnt\_Mod1\_PntMod1\_Mod2\_PntMod2*. For nested pointers the array of this structure will have multiple dimensions (as many dimensions as nesting depth). The pointer is initialized by setting the according fields of the structure:

- modIdx** Has to be set to the index of the indistinct **MODUL** within the array *Unbestimmte\_Module[]*, to make things easier, there is a Define for every indistinct **MODUL** named: *UNBESTIMMT\_<path\_to\_indistinct>*. Where *<path\_to\_indistinct>* it again the path originating from the root module (e.g. *UNBESTIMMT\_Mod1\_Mod2\_Indistinct1*). For nested pointers this path goes over the indistinct **MODUL**s. (E.g. *UNBESTIMMT\_Mod1\_Indistinct1\_Mod2\_Indistinct2*)
- elemtabIdx** Has to be set to the index of the indistinct **MODUL** within the array *Elemente\_der\_Unbestimmten\_Module[]*.
- pnttab** Pointer to the radCASE-Index-Table (in *rdi\_pnttab.c*). To create this file, the according settings have to be set (refer to [radCASE Index \(RDI\)](#)).
- data** Pointer to the allocated memory of the instance. The size needed is the size of the generated global structure for the **ModuleID**. The global structure is named *GLOBAL\_<ModuleID>* where *<ModuleID>* is the **ModuleID** of the indistinct **MODUL** (e.g. *GLOBAL\_MIndistinct1*)

Example for initialization:

```
RD_MRAM    unsigned char    DataPool[1000]; // Has to be global!
long DataOffset = 0;

Pnt_Mod1_Mod2_PntMod[0].modIdx = UNBESTIMMT_Mod1_Mod2_Indistinct1;
Pnt_Mod1_Mod2_PntMod[0].elemtabIdx = 0;
Pnt_Mod1_Mod2_PntMod[0].pnttab = (RD_IDX_PNT_TAB *) &ElemIdxPntTab1;
Pnt_Mod1_Mod2_PntMod[0].data = (unsigned char RD_MRAM *) (DataPool + DataOffset);
DataOffset += sizeof(GLOBAL_MIndistinct1);

Pnt_Mod1_Mod2_PntMod[1].modIdx = UNBESTIMMT_Mod1_Mod2_Indistinct2;
Pnt_Mod1_Mod2_PntMod[1].elemtabIdx = 1;
Pnt_Mod1_Mod2_PntMod[1].pnttab = (RD_IDX_PNT_TAB *) &ElemIdxPntTab2;
Pnt_Mod1_Mod2_PntMod[1].data = (unsigned char RD_MRAM *) (DataPool + DataOffset);
DataOffset += sizeof(GLOBAL_MIndistinct2);
```

When using the pointer in a **SURFACE\_CTR** the placeholder **IDX** can be used as Index of the Pointer (e.g. *Pointer[IDX].elem*) in this case **IDX** will be replaced by a global variable *Index\_<path\_to\_modpointer>*. The value of that variable has to be set manually in a C-Function if this feature is to be used. For nested **SUBMODUL** pointers the path will again go over the other pointers.

When calling a method which is inside of an indistinct **MODUL** (via the pointer) the global variable `G_Procpnt[]` has to be set to point to the `Pnt_Array` structure of the currently active module:

E.g.: `G_Procpnt[0] = &Pnt_Mod1_Mod2_PntMod[1];`

The array `G_Procpnt[]` will again be as large as the nesting depth. For nested pointers the structure has to be initialized for all above layers, e.g.:

```
G_Procpnt[0] = &Pnt_Mod1_Pnt1[3];
G_Procpnt[1] = &Pnt_Mod1_Pnt1_Mod2_Pnt2[3][2];
```

Also the global variable `G_CurPntDepth` has to be set. Where the depth will start with 0, so `G_CurPntDepth = 0` for the first layer of pointers and `G_CurPntDepth = 1` for the second layer of pointers.

Both `G_Procpnt` and `G_CurPntDepth` have to be set manually.

There are some limitations when using **SUBMODUL** pointers:

- It is not possible to go through multiple layers of **SUBMODUL** pointers. So you can't call `Pnt1[0].Pnt2[0].func()` from the main **MODUL**, but you can only call a function in `Pnt1` and only from there you can call functions of the next layer
- Because functions are exported differently for normal **SUBMODULs** and indistinct **MODULs** a **MODUL** definition can only be used for one.
- The usage of **IO-ELEMENTs** is not allowed in indistinct **MODULs** and pointers. It is allowed to define IOs, but they will be ignored and can't be used.
- It is not possible to visualize **SUBMODUL** pointers using `Surface_Vis`, because the **Project Monitor** does not recognize which **MODUL** is instantiated.

### 6.2.5 Distributed Systems

Distributed systems are systems where a task is split upon two or more controllers. How to implement a distributed system in radCASE depends on the architecture of that system.

If there is a controller for the whole system and one or more controllers purely used for controlling IOs the system should not be implemented as a distributed system in radCASE. In this architecture the IOs of the IO-Boards should be implemented as if they were on the main controller. The communication should be purely handled by the HAL which will get all necessary information to identify the IO and the board it is located on using the [Assign string](#) of the IO.

If the architecture is a strict hierarchical structure the system should be implemented using [Sub-nodes](#). In this structure there is one main controller controlling the whole system and one or more subtasks which are controlled by slave controllers (nodes). E.g. within a heating system a slave could manage the controlling of a combustion chamber while the master manages the rest of the heating system. When using **Subnodes** usage of an HMI is limited. The HMI must be on the master controller and is only allowed to change and visualize elements of the slave. It is not allowed to trigger a function using a [PROCEDURE](#) or require additional [ELEMENTs](#) which are not available on the slave.

For all other architectures the system must be implemented using [Precompiler conditions](#). E.g. if having a more entangled architecture where the controllers have more or less the same architecture but different tasks within that architecture. This could be a controller which controls the system and

a second controller with a complex HMI for the system. In this case complex HMI means the HMI will need some procedures and/or elements to calculate the look of the Surfaces. Thus it could be necessary to calculate the state of the system from different sensor values and switch the visualization accordingly.

### 6.2.5.1 Subnodes

To manage a proper communication between the hardware nodes radCASE needs to know the structure of those other hardware nodes. To provide this information you can insert the [MODULs](#) holding the functionality of the other hardware nodes as [SUBMODULs](#) with a **Submodule type** of **Subnode** into your project. When finding a **Subnode** radCASE knows the functionality is on some other hardware node and doesn't need to be generated for the current controller, so the functionality of the **Subnodes** is ignored by radCASE. But radCASE identifies all [ELEMENTs](#) that are used for communication and exports the corresponding communication structure.

Both master and slave nodes are executing their own code and are using the **ELEMENTs** in their own memory (see [Figure 2](#)). The master additionally has a copy of all **ELEMENTs** which are on other nodes and need to be communicated in its own memory. While executing the code and visualizing **ELEMENTs** on the HMI (if existing) every node only accesses its own memory (blue and green lines).

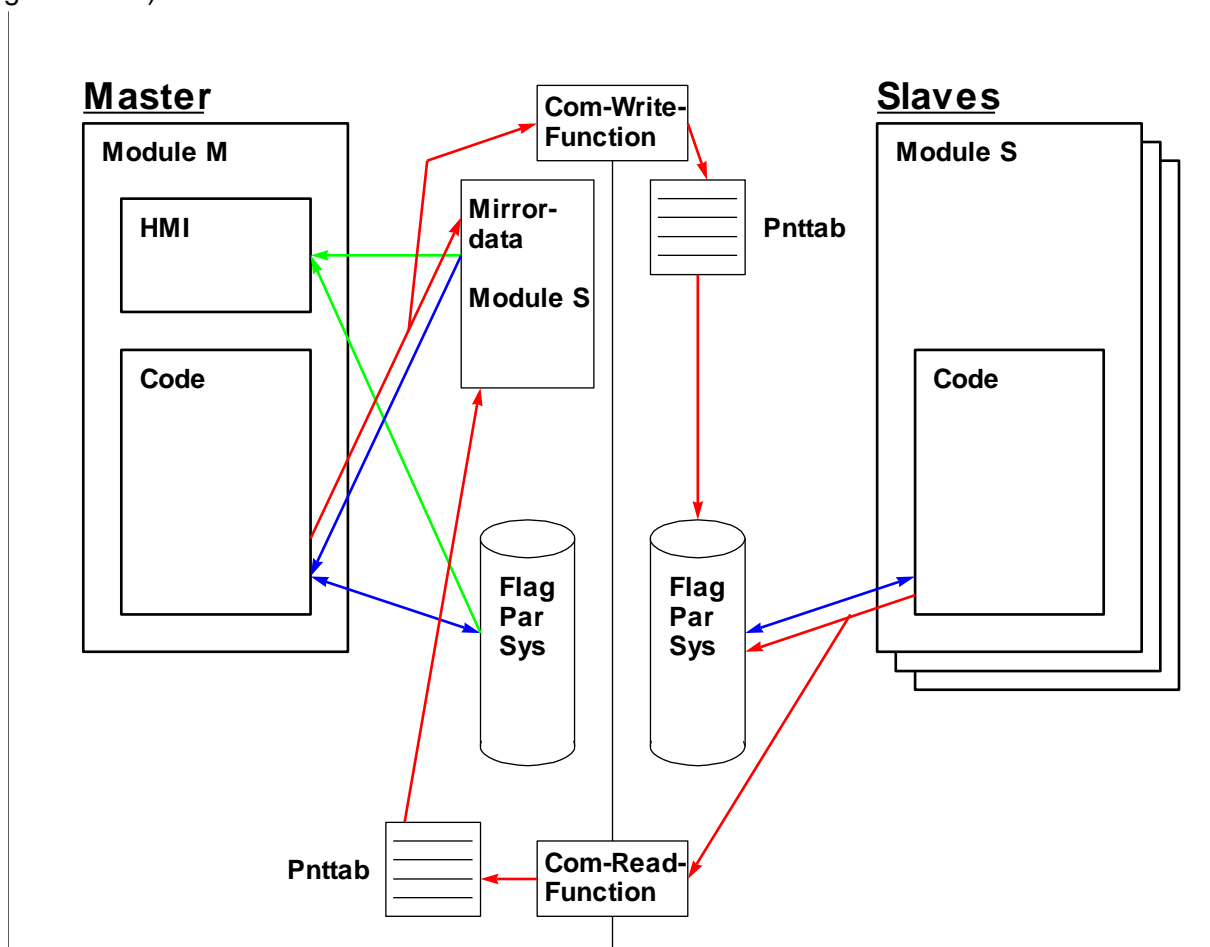


Figure 2, Element Access in distributed systems

If the master writes an **ELEMENT** in the Mirror data memory, there is also a special function called, which depends on the type of communication element (refer to [Element communication](#)). This function sends the data and an **ELEMENT** index (RDI refer to [radCASE Index \(RDI\)](#)) to the according node, which can be identified by the **Controller-ID** (refer to [SUBMODUL](#)). When the data arrives at the Slave the RDI is used to identify the **ELEMENT** sent using the Pnttab. The Pnttab is a simple list in form of an array, where the RDI is resolved to the according Pointer to the **ELEMENT**. With this information the data is written into the memory of the according slave node.

Accordingly when the Slave writes data into an **ELEMENT**, which needs to be communicated it sends the data to the master, which receives it using a special Read-Function and resolves the according data pointer using the Pnttab. After this the new value is written into the Mirror data memory.

Because in a distributed system the functions of all nodes are required, a simulation of just one node normally isn't what is required. Because of this there is a way provided, to simulate the target with all **Subnodes** simultaneously on the PC. To do this the Ctr parameter **UKE=1** needs to be set (refer to [Ctr](#)). In this case **Subnodes** are treated almost like [SUBMODUL](#)s with a **Submodule type** of **Submodule** and all the functionality is exported. The main difference between a **Submodule type** of **Submodule** and the simulation of a **Subnode** when **UKE=1** is activated is the communication. In a simulation of a **Subnode** the communication structures are still exported and are also simulated. This can be very useful to debug the communication process. Because the communication functions need to be implemented manually for the simulation it is easily possible to simulate different connection problems.



Because when activating **UKE=1** the code of all nodes is generated, do not use this setting for generating the target code. To separate between target code and a simulation of the whole distributed system you should use Gen-Files (refer to [Recommended Project File Structure](#))

#### 6.2.5.2 Precompiler conditions

For implementing a distributed system using precompiler conditions a different [DEFINE](#) has to be specified in the Gen-File (refer to [Recommended Project File Structure](#)) of each controller. All content of a [MODUL](#) that is not needed on one of the controllers will be enclosed by [#IF](#), [#IFN](#), [#ELSE](#), [#ENDIF](#) checking for the respective **DEFINE**. All [ELEMENT](#)s which should be communicated between the different controllers may not be enclosed in any of the precompiler conditions and should be marked for [Element communication](#).

There are three different ways to simulate such a distributed system, each having different advantages and disadvantages over the others.

Simulation Method	Advantages	Disadvantages
1. <a href="#">Simulation in one big system</a>	<ul style="list-style-type: none"> <li>• Easy to implement and maintain</li> <li>• Only one project to compile and debug</li> </ul>	<ul style="list-style-type: none"> <li>• Communication can't be debugged</li> <li>• Communication errors can't be simulated</li> <li>• Time difference between starting different controllers can't be</li> </ul>



Simulation Method	Advantages	Disadvantages
		simulated <ul style="list-style-type: none"> <li>Multiple controllers of the same type can't be simulated</li> </ul>
2. <a href="#"><u>Simulating each controller individually</u></a>	<ul style="list-style-type: none"> <li>Easy to implement and maintain</li> <li>Communication can be debugged</li> <li>Communication errors can be simulated</li> <li>Time difference between starting different controllers can be simulated</li> <li>Multiple controllers of the same type can be simulated</li> </ul>	<ul style="list-style-type: none"> <li>Project of each controller has to be compiled</li> <li>Multiple debugger instances needed to debug.</li> <li>While debugging other controllers can run into timeouts</li> </ul>
3. <a href="#"><u>Simulation using Processing type list</u></a>	<ul style="list-style-type: none"> <li>Only one project to compile and debug</li> <li>Communication can be debugged</li> <li>Communication errors can be simulated</li> <li>Multiple controllers of the same type can be simulated</li> </ul>	<ul style="list-style-type: none"> <li>Extra effort needed for implementation and maintenance</li> <li>Time difference between starting different controllers can't be simulated</li> </ul>

#### 6.2.5.2.1 Simulation in one big system

The simulation in one big system can simply be done, by using an additional Gen-File which has all the **DEFINES** for all controllers set. This turns the distributed system into one big system for simulation. In this case the functions for element communication are called (and can be logged) but only the local copy of all the elements is used, so that missing communication of elements will not be caught.

#### 6.2.5.2.2 Simulating each controller individually

To use this type of simulation, simply create a simulation for every controller. For this to work, the communication functions need to be implemented for the simulation to do a real communication. As long as the simulations are run from different directories all controllers can be run simultaneously on the same PC, but dependent on the implementation of the communication functions can also be run on different PCs.

This enables to also simulate turning on and off different controllers at different points in time. Each of the controllers must be debugged individually, which means when debugging the simulation the breakpoints have to be set at according places to get a synchronous debugging, without running into timeouts on one end of the communication.

#### 6.2.5.2.3 Simulation using Processing type list

The idea of this type of simulation is to have an instance for each controller in one Gen-File, but to disable all functionality except the functionality of that controller within the instance. This can be

achieved by providing a list of processing types (refer to [Processing types](#)) to run in the specific instance.

For this to work in addition to the precompiler conditions all functionality which does not run on all controllers must have a user defined processing type to be able to select which functionality to run. The execute-Function of the according processing type may be called from the processing type the function would normally have. But it is best to put that call into the System-module within the Gen-File to prevent multiple calls to the execute-Function within one run of e.g. the PERM-function. As a next step a Gen-File for the simulation should be created with an instance for every controller involved and all the **DEFINES** for all controllers set.

At last for every instance a processing type list has to be provided to select the functions to be called for that instance. To create a processing type list the string PTL= has to be added in the **Controller-ID** of the according [SUBMODUL](#), separated by a space from the Controller-ID. After the PTL= all processing types that should be run in that instance should be listed separated by comma (without spaces). The processing type "PERM" and "INIT" will always be called regardless of the arguments of the PTL. If a processing type is not listed this means the functions with that processing type will not be called within the execute-function of the processing type for that instance. So the execute-function of that processing type can still be called to execute functions of other instances.

The PTL will have effect on the submodule it is defined in and all nested submodules, that do not have an own PTL. By setting a PTL= without adding any processing types all functions will be executed again.



The PTL can also be set using the [EntityTab](#), but this will override any PTL that is defined in a nested submodule, unless the PTL of the nested submodule is also set using the entity tab.



If using processing type lists, special care must be taken when maintaining or expanding the functionality of the controllers. If a new processing type is added for a function (even in a library) the functionality will not be called, unless the processing type is enabled in all of the processing type lists. This can result in bugs which can be hard to find.

### 6.2.5.3 Element communication

There are currently three types of communication **ELEMENTs**:

**Cyclical sent ELEMENTs** Refer to [Cyclical communication](#)

**Event-triggered ELEMENTs** Refer to [Event triggered communication](#)

**CANopen ELEMENTs** Refer to [CANopen communication](#)

To use the different types of element communication you have to set the Desktop-parameters **RI=<#>** and **EI=<#>** (refer to [Desktop](#)). **RI=<#>** sets the **Controller-ID** of [The System MODUL](#). **EI=<#>** affects the generation of the Pnttab (refer to [radCASE Index \(RDI\)](#)).

#### 6.2.5.3.1 Cyclical communication

In the cyclical communication all [ELEMENTs](#) with a [Com Type C<#>](#) are sent periodically between the different nodes. Because the **ELEMENTs** are not directly sent when the value changes, it is not possible to detect which of the nodes has modified the data, so the direction the data needs to be sent is not clear. To specify the direction the **ELEMENTs** have to be defined with an [Assign type](#) of **IN** or **OUT** and are only communicated in one direction.



For more details on how to implement the cyclical communication refer to the Integration manual.

#### 6.2.5.3.2 Event triggered communication

In the event triggered communication all [ELEMENT](#)s with a [Com Type](#) **E<#>** are sent as soon as the value of the **ELEMENT** changes.

The values are sent using special HAL-functions described in detail in the Integration manual.



When enabling the Ctr-Setting **AIS=<#>** (refer to [Ctr](#)) one of the parameters sent to the HAL-function changes, so it contains the Array-Index of any [SUBMODUL](#)-Array the **ELEMENT** is in and also the **Controller-ID**.



Event triggered communication can only be used for **ELEMENT**s with an [Assign type](#) of **FLAG, IN, OUT, IO, PAR, SYS, PROC**.

#### 6.2.5.3.3 CANopen communication

In the CANopen communication all CANopen-[ELEMENT](#)s are sent using the CANopen-protocol. All PDO-**ELEMENT**s are sent periodically and all SDO-**ELEMENT**s are sent if triggered. A CANopen **ELEMENT** is identified by a special [Assign string](#):

*CANOPEN <index>, <subindex>, <attr>, <page>(<fnct-ptr>)*

If no <fnct-ptr> is provided it defaults to NULL

Additional to this the [Com Type](#) specifies if the CANopen-**ELEMENT** is an SDO or PDO. By default all **ELEMENT**s are SDOs, to define a PDO the **Com Type** must have the following format:

*C<index><r/w>.<pos>*

*<index>*: index of the PDO relative to the base index

*<r/w>*: either “r” for Rx PDOs or “w” for Tx PDOs

*<pos>*: (0...7) position of the element within the PDO

To trigger sending of SDO-Elements there is a special access operator (refer to [Access To SDO-Elements \(\\$@\)](#)).

For more information on the implementation of CANopen in the HAL refer to the Integration manual.



CANopen communication can only be used for **ELEMENT**s with an [Assign type](#) of **FLAG, IN, OUT, IO, PAR, SYS, PROC**.

### 6.2.6 MODUL access

It is possible to access surfaces, functions or **ELEMENT**s of another [SUBMODUL](#) in the [Project Hierarchy](#). This is done by simply providing the **SUBMODUL** path using the instance names separated by a point. E.g.

Syntax	Description
<code>&lt;SubmoduleA&gt;</code>	Access to <code>&lt;SubmoduleA&gt;</code>
<code>&lt;SubmoduleA&gt;.&lt;SubSubmoduleB&gt;</code>	Access to <code>&lt;SubSubmoduleB&gt;</code> which is a <b>SUBMODUL</b> of <code>&lt;SubmoduleA&gt;</code>

If one of the **SUBMODUL**s is an array, the **SUBMODUL** can be accessed with an index after the instance name:

Syntax	Description
<code>&lt;Submodularray&gt;[&lt;index&gt;].&lt;SubSubmoduleX&gt;</code>	Access to <code>&lt;SubSubmoduleX&gt;</code> which is a <b>SUBMODUL</b> of the <b>SUBMODUL</b> array <code>&lt;Submodularray&gt;</code>

It is also possible but not recommended (refer to [Recommended Project MODUL structure](#)) to access upper **SUBMODUL**s, i.e. a **SUBMODUL** which is one or more layers up in the hierarchy:

Syntax	Description
<code>_ </code>	Access to <b>SUBMODUL</b> containing the current <b>SUBMODUL</b>
<code>_  </code>	Access to <b>SUBMODUL</b> two layers up

Finally it is possible but strongly advised against (refer to [Recommended Project MODUL structure](#)) to access neighboring **SUBMODUL**s, i.e. **SUBMODUL**s which are **SUBMODUL**s one or more layers up in the hierarchy.

Syntax	Description
<code>_ &lt;Neighbormodule&gt;</code>	Access to <code>&lt;Neighbormodule&gt;</code> which is a <b>SUBMODUL</b> of the <b>SUBMODUL</b> containing the current <b>SUBMODUL</b>



Multiple modules and ELEMENT or function names are separated by dots. However when using a backward reference there is no dot between the backward reference and the next item.

### 6.3 Data Modeling

The [TYPEDEF](#) section allows defining data types. In addition to the actual type, these data types contain various additional information (meta data) such as data type, scope, areas, editing restrictions, etc., and are used in the module for simple instantiating of [ELEMENT](#)s (radCASE variables). This is done by assigning data types to each **ELEMENT**.

The **ELEMENT**s can have one of the following basic data types:

- Numerical (refer to [Numerical Elements](#))
- Enumerative/Selective/Binary (refer to [Enumerative Elements](#))
- Alphanumerical (refer to [String Elements](#))

- Time (refer to [Time Elements](#))
- Date (refer to [Date Elements](#))

To take the different **ELEMENT** usage and **ELEMENT** allocation into account, radCASE offers to use different [Assign types](#) and [Assign strings](#) to define this behavior (refer to [Element Usage And Allocation](#)) without having to take much care of the differences between those elements within the model. There is also some limited support for arrays of **ELEMENT**s (refer to [Element Arrays](#)).

The default value of an **ELEMENT** can be defined in different ways:

1. If no default value is defined radCASE chooses a standard default value for each data type
2. By defining a **Default value** in the **TYPEDEF** the radCASE default is overwritten
3. By defining the option **SV=** in the [Format string](#) of an **ELEMENT** definition, the default of the data type is overwritten.
4. By defining the option **SV=** in the **Format string** of an instance in the [EntityTab](#), the default of the **ELEMENT** definition is overwritten.

### 6.3.1 Data Types

#### 6.3.1.1 Numerical Elements

Numerical [ELEMENT](#)s can be created by assigning them a data type of [ENUM](#). The **ENUM** data type uses a virtual floating point format (refer to [Virtual Floating Point](#)) to avoid the need to use real floating point support (refer to [Ctr-Setting FP=<0/1>](#)).

The radCASE standard default value (refer also to [Data Modeling](#)) of an **ENUM** is 0. The default value can be specified as numerical value in different formats:

- Decimal** By just providing the according number. E.g. 11
- Hexadecimal** By prepending a "0x" before the hexadecimal value. E.g. 0xB
- Binary** By prepending a "0b" before the binary value. E.g. 0b1011

In the same way like overwriting the standard value of the data type (refer to [Data Modeling](#)), it is also possible to overwrite the **Range** of the **ELEMENT** using **Format string LV=** and **HV=** in the [Format string](#) or the [EntityTab](#) and to overwrite the Alarm by using the **Format string LA=** and **HA=**.



When overwriting ranges or standard value with a reference to another **ELEMENT**, the data types have to be very similar to another, because of the internal format of the [Virtual Floating Point](#) support. The decimal places will be unregarded and the internal value will be copied.

Further it is possible to dynamically change the **Unit** of an **ENUM** (refer to [Automatic Unit Conversion](#)).

#### 6.3.1.1.1 Virtual Floating Point

radCASE supports a virtual floating point format, meaning within the radCASE model it is possible to use floating point variables, which are internally handled as integers. The internal format depends on the number of **Decimals** specified in the [ENUM](#), and is calculated by multiplying the floating point value with 10 to the power of the number of **Decimals**.

So for example the following table shows the internal format of the value 1.23:

Number of Decimals	Internal value
0	1
1	12
2	123
3	1230

The virtual floating point format is used with §-Access (refer to [Access To Element Related Constants \(§\)](#))

#### 6.3.1.1.2 Automatic Unit Conversion

On the target HMI it is possible to change between different measuring systems at runtime (e.g. meter/inch or K/°C). The feature allows changing the physical unit and recalculating the numeric value of an [ENUM-ELEMENT](#) correspondingly. To enable this automatic unit conversion which converts every **ELEMENT** with a specific unit to display its value in another unit the [Ctr-Setting UN=<0/1>](#) has to be activated.

If activated there are three function pointers which can point to self-defined conversion functions:

Pointer	Function Prototype	Description
<b>CElement_P_conv_unit</b>	<i>long func(CElement RD_MROM *elem, long actval, RD_char RD_MRAM *unit, short *format);</i>	Function to convert from the original unit of the <b>ELEMENT</b> to a new one
<b>CElement_P_conv_unit_reverse</b>	<i>long func(CElement RD_MROM *elem, long actval);</i>	Function to convert back from the new unit of the <b>ELEMENT</b> to the original one
<b>CElement_P_conv_stepwidth</b>	<i>short func (CElement RD_MROM *elem, short actstepwidth);</i>	Optional function to adjust the <b>Step width</b> of the <b>ELEMENT</b> to the new unit

To initialize those pointers it is recommended, to define [PROC](#)s of the according prototypes and use \$\* (refer to [Function Pointer Access \(\\$\\*\)](#)) to assign function pointers of those **PROC**s to the pointers within a **PROC** with a **Processing type** of **INIT** (refer to [Processing types](#)). E.g.:

```
CElement_P_conv_unit = $*myConv;
CElement_P_conv_unit_reverse = $*myRevConv;
CElement_P_conv_stepwidth = $*myStepWidthConv;
```

The following example code will transform an **ELEMENT** from the format xx.xx°C to xx.xF. The step width will be changed from 0.1°C to 0.5F

First we will define the function for converting the original value to F:

**Function name:** *myConv*

**Return type:** *long*

**Arguments:** (*CElement RD\_MROM \*elem, long actual, RD\_char RD\_MRAM \*unit, short \*format*)

```

{
    // Only convert if enabled
    if ($EnableConversion)
    {
        short dec;

#ifdef RD_USE_UTF8
        char *putf;
        unsigned short *puni;
        char RD_MRAM utfcode[20];
        unsigned short RD_MRAM unicode[20];

        // Copy unit to ANSI-string
        strcpy(utfcode, "°C");
        // Convert to Unicode-string
        putf = utfcode;
        puni = unicode;
        while (*putf != 0)
        {
            *puni = (unsigned short)(unsigned char)(*putf);
            puni++;
            putf++;
        }
        *puni = 0;
        // Convert to UTF8-String
        struni2utf(unicode, utfcode, 20);
        // Compare with unit
        if (strcmp(unit, utfcode) == 0)
#else // RD_USE_UTF8
        if (RD_strcmp(unit, STR2UNI("°C")) == 0)
#endif // RD_USE_UTF8
        {
            // Get format and check digits
            // Don't use format here, because format can be NULL
            dec = (CENum_getFormat(elem) & 0xf);

            switch(dec)
            {
                // Conversion if no digit
                case 0:
                    actual = ((actual * 9) / 5) + 32;
                    if (format)
                    {
                        // Add one digit
                        *format = (*format & 0xFF0F) + ((*format & 0xF0) + (1 << 4));
                    }
                    break;

                // Conversion if one digit
                case 1:
                    actual = ((actual * 9) / 5) + 320;
                    // Remove the decimal this means two digits more before the colon,
                    // one for the decimal and one for the colon. Because we only need one
                    // more digit, we can remove one
                    if (format)
                    {
                        // Delete one decimal
                        *format = (*format & 0xFFF0) + ((*format & 0xF) - 1);
                        // Delete one digit
                        *format = (*format & 0xFF0F) + ((*format & 0xF0) - (1 << 4));
                    }
                }
            }
        }
    }
}

```

```

    }
    // Convert actual value to new element format
    actual /= 10;
    break;

    // Conversion if two digits
case 2:
    actual = ((actual * 9) / 5) + 3200;
    // Remove one decimal this means one digit more before the colon
    // this is what we want so no change in digits.
    if (format)
    {
        // Delete one decimal
        *format = (*format & 0xFFF0) + ((*format & 0xF) - 1);
    }
    // Convert actual value to new element format
    actual /= 10;
    break;
}
// Change the unit
#ifdef RD_USE_UTF8
// Copy unit to ANSI-string
strcpy(utfcode, "F");
// Convert to Unicode-string
putf = utfcode;
puni = unicode;
while (*putf != 0)
{
    *puni = (unsigned short)(unsigned char)(*putf);
    puni++;
    putf++;
}
*puni = 0;
// Convert to UTF8-String
struni2utf(unicode, utfcode, 20);
// Copy to unit
strcpy(unit, utfcode);
#else // RD_USE_UTF8
RD_strcpy(unit, STR2UNI("F"));
#endif // RD_USE_UTF8
}
}

return actual;
}

```



The format passed to this function is the attribute **format** of an **ENUM** (refer to [ENUM Attributes](#)). When changing the number of decimals, this also changes the internal data format (refer to [Virtual Floating Point](#)) and the value has to be adjusted manually to match the new internal data format.



The format is only passed to that function if needed, so there is a possibility of the variable being a NULL-Pointer.



When changing the format, keep in mind the number of digits/characters also contains the decimal point and algebraic sign. So when adding a decimal to an **ELEMENT** without deci-

mals the number of digits has to be increased by 2. Also when changing the range of the **ELEMENT** from unsigned to signed the number of digits has to be increased by 1 for the algebraic sign.

After adding the above function, the value will be shown correctly in the desired format and with the correct new unit. But to be able to also edit the **ELEMENT** radCASE needs to know how to convert the value back, so it can be stored internally in the original format.

Because of this we now define the function for converting the value back from F to °C:

**Function name:** *myRevConv*

**Return type:** *long*

**Arguments:** (*CElement RD\_MROM \*elem, long actual*)



```

{
    // Only convert if enabled
    if ($EnableConversion)
    {
        RD_char RD_MRAM testunit[20];
        short dec;

#ifdef RD_USE_UTF8
        char *putf;
        unsigned short *puni;
        char RD_MRAM utfcode[20];
        unsigned short RD_MRAM unicode[20];

        // Copy unit to ANSI-string
        strcpy(utfcode, "F");
        // Convert to Unicode-string
        putf = utfcode;
        puni = unicode;
        while (*putf != 0)
        {
            *puni = (unsigned short)(unsigned char)(*putf);
            puni++;
            putf++;
        }
        *puni = 0;
        // Convert to UTF8-String
        struni2utf(unicode, utfcode, 20);
#endif // RD_USE_UTF8

        CElement_getUnit(elem, DISP_UNIT_SPACELESS, testunit SUBTEXT_NORM);

#ifdef RD_USE_UTF8
        // Compare with unit
        if (strcmp(testunit, utfcode) == 0)
#else // RD_USE_UTF8
        if (RD_strcmp(testunit, STR2UNI("F")) == 0)
#endif // RD_USE_UTF8
        {
            // get format and check digits
            dec = (CENum_getFormat(elem) & 0xf);
            switch(dec)
            {
                // Conversion if no digit
                case 0:
                    actual = ((actual - 32) * 5) / 9;
                    break;
                // Conversion if one digit
                case 1:
                    // Convert actual value back to old element format
                    actual *= 10;
                    actual = ((actual - 320) * 5) / 9;
                    break;
                // Conversion if two digits
                case 2:
                    // Convert actual value back to old element format
                    actual *= 10;
                    actual = ((actual - 3200) * 5) / 9;
                    break;
            }
        }
    }
}

```

```

    }

    return actual;
}

```

After adding the function above, editing of the value is working correctly. However because of the **Step width** of the **ELEMENT** of 0.1°C the converted value has a Step width of 1.0F. To change this to another value, we have to define the following function:

**Function name:** *myStepWidthConv*

**Return type:** *short*

**Arguments:** (*CElement RD\_MROM \*elem, short actstepwidth*)

```

{
    // Only convert if enabled
    if ($EnableConversion)
    {
        RD_char RD_MRAM testunit[20];

#ifdef RD_USE_UTF8
        char *putf;
        unsigned short *puni;
        char RD_MRAM utfcode[20];
        unsigned short RD_MRAM unicode[20];

        // Copy unit to ANSI-string
        strcpy(utfcode, "F");
        // Convert to Unicode-string
        putf = utfcode;
        puni = unicode;
        while (*putf != 0)
        {
            *puni = (unsigned short)(unsigned char)(*putf);
            puni++;
            putf++;
        }
        *puni = 0;
        // Convert to UTF8-String
        struni2utf(unicode, utfcode, 20);
#endif // RD_USE_UTF8

        CElement_getUnit(elem, DISP_UNIT_SPACELESS, testunit SUBTEXT_NORM);

#ifdef RD_USE_UTF8
        // Compare with unit
        if (strcmp(testunit, utfcode) == 0)
#else // RD_USE_UTF8
        if (RD_strcmp(testunit, STR2UNI("F")) == 0)
#endif // RD_USE_UTF8
        {
            // Change stepwidth to in case of X.XF to 0.5F
            actstepwidth = 5;
        }
    }
    return actstepwidth;
}

```



When using the automatic unit conversion the converted value might not fit into the automatic detected C-data type. To circumvent this problem a forced data type should be used (refer to [Forced Data Type](#)).



Because radCASE extends the function arguments for multiple instanced **MODULs** the above functions only work in a single instantiated **MODUL**. It is recommended to put such global functionality into a central place at a high level in the hierarchy.

### 6.3.1.2 Enumerative Elements

Enumerative [ELEMENTs](#) can be created by assigning them a data type of [EBIN](#). An **EBIN** deals with an enumerative data type (also known as selective or binary), where the individual selection statuses are defined by [EB\\_ENTRYs](#). An element defined as **EBIN** is limited in radCASE to a maximum of 32767 **EB\_ENTRYs** for a normal **EBIN** and 32 **EB\_ENTRYs** for a **multiselective EBIN**. As **multiselective EBIN** the data type contains independent On/Off statuses, where multiple selections can be active at the same time.

The radCASE standard default value (refer also to [Data Modeling](#)) of an **EBIN** is 0.

The default value can be specified as a numerical value of the corresponding **EB\_ENTRY** or by its **Name**.

For multiselective EBINs different selections can be ored, e.g. *Sel1 | Sel2*

It is also possible to disable different selections (refer to [Enabling/Disabling Selections](#))

#### 6.3.1.2.1 Enabling/Disabling Selections

While dealing with languages or operating types, it may happen that selections have to be enabled or disabled. To turn off selections of enumerative **ELEMENTs** on a **SURFACE\_VIS** refer to setting **Mask Bin** in the element visualization (refer to [ELEM/EDIT](#)).

On a **SURFACE\_CTR** this feature must first be enabled for the **ELEMENT** by enabling the [Format string](#) setting **EV=1**. By enabling this feature a bitmask of type long (up to 32 selections) or an array of long will be created where each bit is equivalent to one selection of the **EBIN**. Please pay attention on those bitmask when customizing dialogs (refer to [Customizing Standard System Dialogs](#)).

The enabling and disabling of selections can be done from the code with the following C-functions:

```
/* Enables selection index of element elem. Indices are 0-based */
void CEBin_set_enablebit(CEBin *elem, short index);

/* Disables selection index of element elem. Indices are 0-based */
void CEBin_reset_enablebit(CEBin *elem, short index);

/* Sets the selection mask for element elem. The type of mask depends on the maximum number
of selections. Mask is of type long if 32 or less selections are allowed, otherwise it is
an array of longs. */
void CEBin_put_enablemask(CEBin *elem, mask);
```

Examples:

The following code disables the fourth selection of element options:

```
CEBin_reset_enablebit( (CEBin *) $*options, $valueToDisable);
```

The following code enables the fourth selection of options and disables all other selections:

```
CEBin_put_enablemask( (CEBin *) $*options, 1<<3);
```



If the functions are used without activating the feature with **Format string EV=1** the error code 0x100B will be thrown (refer to [Runtime Error Messages](#)).

On a **SURFACE\_VIS** disabling selections is limited to 32 selections.

### 6.3.1.3 Date Elements

Date [ELEMENTs](#) can be created by assigning them a data type of [EDAT](#). The **EDAT** data type is converted internally into a long with the following format:

```
date = (year << 16) + ((month-1) << 8) + (day-1);
```

The radCASE standard default value (refer also to [Data Modeling](#)) of an **EDAT** is 01.01.2000. The default value can be specified in the following formats:

**DD.MM.YY** Day.Month.Year with two digits (e.g. 26.06.12)

**DD.MM.YYYY** Day.Month.Year with 4 digits (e.g. 26.06.2012)

When working with two digit values for the year radCASE will automatically assume it is in the year 20xx.

### 6.3.1.4 Time Elements

Time [ELEMENTs](#) can be created by assigning them a data type of [ETIM](#). The **ETIM** data type is converted internally into a long with the following format:

```
time = (hours << 24) + (minutes << 16) + (seconds << 8) + centiseconds;
```

The radCASE standard default value (refer also to [Data Modeling](#)) of an **ETIM** is 00:00:00.00. The default value can be specified in the following formats:

**hh:mm** Hours and Minutes. E.g. 12:34

**hh:mm:ss** Hours, Minutes and Seconds. E.g. 12:34:45

**hh:mm:ss.ms** Hours, Minutes, Seconds and Centiseconds. E.g. 12:34:45.78

For the above formats, when entering a default value all not specified variables are set to 0. E.g. when specifying the standard value as 12:34 the default value will be 12:34:00.00.

### 6.3.1.5 String Elements

String [ELEMENTs](#) can be created by assigning them a data type of [ESTR](#). The **ESTR** data type is converted internally into a **RD\_char** which is defined differently for different Unicode settings (refer to [Unicode](#))

The radCASE standard default value (refer also to [Data Modeling](#)) of an **ESTR** is an empty string "" The default value can be specified by surrounding it with double quotes e.g. "new Default".



Multiline texts within **ESTRs** are only supported for activated Proportional font support (refer to [Proportional radCASE System Fonts](#))

#### 6.3.1.6 Forced Data Type

radCASE will always try to assign the most optimized C data type to variables of **TypeDef** [ENUM](#) and [EBIN](#). However in some situations the developer might want to assign another data type to those variables because he needs to change the ranges specified above e.g. when using automatic unit conversion (refer to [Automatic Unit Conversion](#)). When selecting **Forced data type**, radCASE will use this data type instead of the optimal one by referring to the range of values.

### 6.3.2 Changing Of Metadata At Runtime

It is possible to change the metadata of an [ELEMENT](#) dynamically at runtime. Normally the metadata of an **ELEMENT** is located in the *OsdI.ini* and therefore in the ROM. The metadata is referenced as an offset in the **ELEMENT** structure which is a constant variable.

To dynamically change the metadata of an **ELEMENT**, first the **ELEMENT** structure has to be defined to be not constant. This is done by activating the option **Runtime dynamic meta data** of the **ELEMENT**. Now the reference to the metadata can be changed. This option also generates a slightly different structure, which enables to save a pointer to the new metadata.

The next step to dynamically change the metadata of an **ELEMENT** is to create a copy of the metadata in the RAM. To do this first a global variable which can contain the metadata has to be created. The C-data type of this variable has to be:

- DEnum for an [ENUM](#)
- DEBin for an [EBIN](#)
- DEStr for an [ESTR](#)
- DEDat for an [EDAT](#)
- DETim for an [ETIM](#)

For example to create the global variable for an **ENUM** the following code could be inserted into the **Global code** of a [METHODS](#)-RC-Item:

```
// Global variable for containing changeable metadata of element
DEnum varMetadata;
```

After this the original metadata of the **ELEMENT** should be copied to the new global variable. The metadata pointer should point to the new global variable and the reference to the original metadata should be changed to mark it as invalid. This can be done, by inserting the following code into a [PROC](#) which is called only once; e.g. a function with a **Processing type** of **INIT** (refer to [Processing types](#)):

```
// Declare a local variable for containing binary data
RD_DECL_OSDL(DENum, def);
// Read out binary data into local variable
RD_GET_OSDL(DENum, $*someEnumElem->elem.ele1.def, def);
// initialize the changeable metadata once
memcpy(&varMetadata, &(RD_OSDL(def)), sizeof(DENum));
// Save pointer to changeable metadata in Element
$*someEnumElem->def = &varMetadata;
// Mark offset to original metadata as invalid (triggers using of the pointer)
$*someEnumElem->elem.ele1.def = RD_NOBINDATA;
```

After this, the metadata can be changed using `$#` (refer to [Access To Elements Metadata \(\\$#\)](#)), which will always access the pointer for runtime dynamic **ELEMENTS**.

### 6.3.3 Element Usage And Allocation

The **ELEMENT** usage is defined by the [Assign type](#) of the **ELEMENT**. radCASE supports the following different classes of **Assign types**:

- Normal transient data (refer to [General Data](#))
- Data connected with hardware on the embedded system (refer to [Hardware Specific Data](#))
- Non-transient data (refer to [Non-Transient Data](#))
- Data only available in Project Monitor (refer to [Project Monitor Specific Data](#))

Additional to these **Assign types** it is possible to select a unique text as a placeholder which has to be replaced by an **Assign type** using [REPLACE](#). This mechanism is useful when using the same module in different projects and having to change the **Assign type** for each of the projects.

The **ELEMENT** allocation and some additional attributes for some of the **Assign types** are defined by the [Assign string](#). For most of the **ELEMENTS** you will normally choose the **Assign string CTR**. This signals radCASE to automatically assign a position in the communication data block. For the **Hardware Specific Data** types some additional attributes are required (refer to [Assign string of Hardware specific data types](#)), for the **Project Monitor Specific Data** types the **Assign string** will have a different syntax (refer to [Assign string of Project Monitoring specific data types](#)).

#### 6.3.3.1 General Data

The general data is data corresponding to normal C-variables. The data will be set to defaults on every power down of the embedded system. The following data types are supported:

<b>FLAG</b>	This is the standard working variable.
<b>CONST</b>	Constant value. Will be exported as Define, can't be used for multiple <b>MODUL</b> instances.
<b>IN, OUT, IO</b>	Technically these are <b>FLAG</b> elements. They are meant for usage as interfaces between <b>MODULS</b> or in <a href="#">Signal Diagrams</a> . They define the direction of the data flow (In, Out, or In and Out).
<b>INEVT, OUT-EVT</b>	<b>ELEMENTS</b> with this <b>Assign type</b> provide event triggered methods for setting and fetching data. Like for <b>IN</b> and <b>OUT</b> , the only difference between those two is the data direction. For more information refer to <a href="#">Access To Event Elements</a> .

### NATIVEvar, NATIVEconst

**ELEMENT**s with this **Assign type** will be exported without any metadata (refer to [Element Attributes](#)). This reduces overhead, but severely limits functionality. **ELEMENT**s without metadata can't be visualized or otherwise used within a **SURFACE\_CTR**. The values of the **ELEMENT** also can't be communicated, neither to the [Project Monitor](#) nor within [Distributed Systems](#). As **Assign string** only CTR is supported. Only **SV** is a supported **Format string**. There is no support for **Com type**, **EProfiles** or **XCOM-Type**.

### 6.3.3.2 Hardware Specific Data

The hardware specific data is data connected to hardware on the embedded system. The following data types are supported:

- AI, AO** Analog or continuous value hardware input or output (16 bit). The only allowed data type is **ENUM**, and it should have a **Forced data type** of short.
- DI, DO** Digital input or output (1 bit). The only allowed data type is **EBIN**, and it should have a **Forced data type** of unsigned char.
- TI** Timer with selectable resolution (32 Bit). The only allowed data type is **ENUM**, and it should have a **Forced data type** of long. The value is the time passed in seconds. Also refer to [Handling Timers](#).
- CNT** Impulse counter (32 bit). The only allowed data type is **ENUM**, and it should have a **Forced data type** of long.
- RTC** Variable connected to the RTC. Technically the variable is a **FLAG** where the values are permanently copied to from the RTC (or in playback mode of **Project Monitor** from the [Data Recording](#)). This results in the RTC-**ELEMENT**s being read only. To write to the RTC, the according functions (refer to [CFUNC](#)) have to be used. There is only one [ETIM](#) and one [EDAT](#) **ELEMENT** allowed having this **Assign type**.

#### 6.3.3.2.1 Formatting of continuous values

Analog and continuous values as well as counter values need to be formatted from the raw value as obtained from the hardware into a value used by the process.

The function to handle this formatting is pointed to by the pointer named fp\_calibConvert.

The typedef of the function is

```
typedef unsigned short (*t_calibConvertFP)(struct s_kaliio RD_MIO *io, void *unconverted, void* converted);
```

\*unconverted and \*converted are of type short in case of AI, AO and of type long in case of CNT.

By default, the formatting is a 2 point linear transformation. For hardware where this is not suitable the conversion function can be replaced by an individual function performing the required formatting.

The formatting function is called:

- On reading the hardware port. In this case, both unconverted and converted are != NULL and refer to the appropriate memory location.
- At the end of a calibration sequence. In this case, converted is NULL. This is helpful if the unconverted value obtained from the calibration needs to be modified before it is stored permanently.



### 6.3.3.3 Non-Transient Data

The Non-transient data is data that lasts over a power down of the embedded system. Because the data is also stored after the software is replaced, special care has to be taken, to delete data not usable by a new software (refer to [Managing Non-Volatile Memory](#)) or to associate values to [ELEMENTs](#) across different software versions (refer to [Data preservation](#)). The following data types are supported for permanent storage:

- PROC** Process variable. The handling of this variable is the same as for a normal working variable, but the values are maintained over a power down. The physical storage usually takes place in a battery buffered RAM, an EEPROM area or similar storage media (hardware and HAL dependent). If the variables are in an EEPROM the lifetime of EEPROM has to be ensured by the HAL.
- PAR** Parameters which are stored permanently (usually in an EEPROM). Because this data should be rarely changed ( $t > 1$  day) and to ensure a long lifetime of the EEPROM the handling of saving has to be done manually (refer to [Working Copy And Edit Copy](#)).
- SYS** System parameters which are stored permanently (usually in an EEPROM). This data should change very seldom (normally only at system setup). Like for assign type PAR because of the normal saving in EEPROM the handling of saving has to be done manually (refer to [Working Copy And Edit Copy](#)). The data is stored separated from the PAR, so the data is safe even when setting back parameters to factory defaults.
- AI, AO, CNT** The calibration data of the different IOs will be stored permanently (usually in EEPROM). The handling of saving will be done automatically.

#### 6.3.3.3.1 Working Copy And Edit Copy

[ELEMENTs](#) of [Assign type](#) **PAR** and **SYS** as well as the calibration data of [Hardware Specific Data](#) are stored in the EEPROM. For each data in the EEPROM there are two copies in the RAM, the working copy and the edit copy.

For each of the data types there is a copy and a restore function, which can be called from a [CFUNC](#) or directly from C-Code. The following functions exist:

	<b>PAR</b>	<b>SYS</b>	Calibration data
CFUNC Copy function	PAR_COPY	SYS_COPY	KALI_COPY
C-Code Copy function	cFuncParCopy()	cFuncSysCopy()	cFuncKaliCopy()
CFUNC Restore function	PAR_RESTORE	SYS_RESTORE	KALI_RESTORE
C-Code Restore function	cFuncParRestore()	cFuncSysRestore()	cFuncKaliRestore()

Each of the copy functions will copy the working copy data to the edit copy. The restore functions will copy the edit copy to the EEPROM and the working copy. If edit copy's are disabled by setting a [DEFINE](#) **RD\_NO\_EDIT\_COPY**, only the restore function is available and will save the data to the EEPROM.

The working copy is used for all behavior (refer to [Behavior Modeling](#)) which changes any of the data. To store that data permanently, first the copy function and after that the restore function has to be called.



The edit copy is used for all changes from the target HMI (refer to [HMI](#)). It is best to call the copy function before doing any editing of the according data to prevent the edit copy from being out of date. After editing the changes can be saved by calling the restore function or can be reverted by calling the copy function again. This is often done by using the [ASKOK](#) dialog with **Type of Question** 3. In this case the restore function and copy function are directly behind each other and canceling the saving results in a jump over the restore function directly to the copy function.



The calibration data is normally handled in the internal calibration functions, so normally only **PAR** and **SYS** need to be handled manually.

### 6.3.3.3.2 Managing Non-Volatile Memory

There is several data of a project which is stored in non-volatile memory:

- [ELEMENTs](#) with an [Assign type](#) of **PAR**
- **ELEMENTs** with an **Assign type** of **SYS**
- **ELEMENTs** with an **Assign type** of **PROC**
- Calibration data of **ELEMENTs** with an **Assign type** of **AI**, **AO** and **CNT**.

Because this data is stored in non-volatile memory, the data is not deleted if the software is replaced. This would cause problems when replacing software with complete different software or even when updating software with a newer version where the structure of that data has changed.

radCASE tries to deal with this problem automatically by using checksums and determining the length of the memory areas, however this mechanism can't always detect a change in the structure, so it is recommended to always trigger erasing of the according memory area, when changing the structure. This can be done by different [DEFINEs](#) containing version numbers. If the version number changes the according memory area is deleted automatically.

The following **DEFINEs** are used for managing the non-volatile memory:

<b>DEVICE_ID</b>	If this version changes the whole non-volatile memory is erased.
<b>VERS_PARAM</b>	If this version changes all <b>ELEMENTs</b> with an <b>Assign type</b> of <b>PAR</b> are reset to their default values.
<b>VERS_SYSTEM</b>	If this version changes all <b>ELEMENTs</b> with an <b>Assign type</b> of <b>SYS</b> are reset to their default values.
<b>VERS_PROC</b>	If this version changes all <b>ELEMENTs</b> with an <b>Assign type</b> of <b>PROC</b> are reset to their default values.
<b>VERS_KALI</b>	If this version changes the whole memory area containing the calibration data of IOs is deleted.

### 6.3.3.3.3 Data preservation

The goal of the data preservation is to keep permanently stored data over software updates of the target controller, even when the data structure of the permanent data has changed.

In the model to support the feature, all [ELEMENTs](#) which are permanently stored must have an RDI (refer to [radCASE Index \(RDI\)](#)), which can be accomplished by setting the [Desktop](#)-Setting

**DP=<#>**. Depending on this setting the three files *preserveSys.c*, *preservePar.c* and *preserveProc.c* are generated. The file *preserveTab.c* is generated whenever **Data Preservation** is active.

Additionally, the Storage Offsets **PARPRESRV\_OFF**, **SYSPRESRV\_OFF**, **PROCPRESRV\_OFF** in the persistent data storage device (usually EEPROM) must be defined in *MEMORY.DEF*, see Integration Manual or template *MEMORY.DEF* for further details.

Note that setting **DP=7** without defining **PARPRESRV\_OFF**, **SYSPRESRV\_OFF**, **PROCPRESRV\_OFF** is required if the Parameter Import / Export to a SD-Card shall be used. See documentation in file *SD-Card.rad* for further information.

#### 6.3.3.4 Project Monitor Specific Data

The **Project Monitor** specific data is data which is only accessible within the [Project Monitor](#). The following data types are supported:

- KEY** Virtual key on target system. Refer to [Virtual Keyboard Support](#).
- LOCAL** Help variable which can be manually manipulated (only using edit dialog in **SURFACE\_VIS**) in the **Project Monitor**. For example this can be used to simulate hardware errors.
- EVA** Help variable which can only be set using **Stimulation Equation** (refer to [Stimulation Equations](#)). Mainly for modeling process and environment simulations. Only [EBIN](#) and [ENUM-ELEMENTs](#) are allowed for this **Assign type**.

#### 6.3.4 Element Arrays

There is some limited support for [ELEMENT](#)-Arrays within radCASE. **ELEMENT**-Arrays can be created by declaring a Multiplicity for the **ELEMENT**. This is the same as the array size of a C-Array and the access to **ELEMENT**-Arrays is also the same like in C by using square brackets (refer to [Element Access](#)).

At the moment **ELEMENT**-Arrays are only supported within C-Code and to copy a whole array within [Signal Diagrams](#). Among others it is currently not possible to restore the default values at runtime, to visualize **ELEMENT**-Arrays in any way or access **ELEMENT**-Arrays using Actions.

#### 6.3.5 Data Storage

##### 6.3.5.1 Database Storage

[ELEMENTs](#) can be stored in a database, where the database can be anything from a SQL-database down to a special memory area where the values are stored. Because of this wide range of support the database access handling has to be done manually.

To define **ELEMENTs** for storage in the database, the [Com Type D<#>](#) should be used. After this it is possible to search specifically for those **ELEMENTs** and read out the values to store them in the database or write values from the database into the **ELEMENTs** in a loop.

To search for the database **ELEMENTs** the following array is generated:

```
RD_MROM CEmb_Database DatenbankElemente[];
```

*CEmb\_Database* has the following structure:

```
typedef struct
{
    CElement RD_MROM *element; /* Pointer to ELEMENT */
    short comTypeD; /* number of selected database (Com Type) */
    short zugriffsnr; /* normally not used */
} CEmb_Database;
```

The *element*-Pointer of the last entry in the array *DatenbankElemente* is *NULL*, so this can be used as abort condition of a loop over the array. To access the values of the **ELEMENT**s the *element*-Pointer can be casted to a different structure containing the current values of the **ELEMENT**.

The *CElement* structure looks as follows:

```
struct S_Element
{
    unsigned char elType; /* ELEMENT type */
    RD_OSDL_REF(void) def; /* Pointer to binary metadata */
    struct S_Element RD_MROM *next; /* Pointer to next ELEMENT */
#ifdef USE_RDI_INDEX
    short rdi_index; /* radCASE Index */
#endif
};
typedef struct S_Element CElement;
```

In this structure *def* is a pointer to the binary metadata (refer to [Access To Elements Metadata \(\\$#\)](#)). *elType* can be used to identify to which structure the *element*-Pointer can be casted:

**ETYPE\_NUM** Can be casted to *CENum RD\_MROM \**

**ETYPE\_BIN** Can be casted to *CEBin RD\_MROM \**

**ETYPE\_STR** Can be casted to *CEStr RD\_MROM \**

**ETYPE\_TIM** Can be casted to *CETim RD\_MROM \**

**ETYPE\_DAT** Can be casted to *CEDat RD\_MROM \**

The structures the *element*-Pointer can be casted to, look as follows:

```
struct S_ENum
{
    CElement elEl; /* CElement structure same for all ELEMENTs */
    void RD_MRAM *act; /* Pointer to current value of ELEMENT */
    char actvalsize; /* size of current value in bytes (negative value means
                    unsigned variable */
};
typedef struct S_ENum CENum;
```

```

struct S_EBin
{
    CElement ele1; /* CElement structure same for all ELEMENTs */
    void RD_MRAM *act; /* Pointer to current value of ELEMENT */
    char actvalsize; /* size of current value in bytes (negative value means
                     unsigned variable */
    #if (RD_SELMAXSIZE > 1)
        unsigned long RD_MRAM *enable_sel[RD_SELMAXSIZE]; /* Selection mask */
    #else
        unsigned long RD_MRAM *enable_sel; /* Selection mask */
    #endif
};
typedef struct S_EBin CEBin;

struct S_EStr
{
    CElement ele1; /* CElement structure same for all ELEMENTs */
    RD_char RD_MRAM *act; /* Pointer to current value of ELEMENT */
};
typedef struct S_EStr CESTr;

struct S_ETim
{
    CElement ele1; /* CElement structure same for all ELEMENTs */
    void RD_MRAM *act; /* Pointer to current value of ELEMENT */
};
typedef struct S_ETim CETim;

struct S_EDat
{
    CElement ele1; /* CElement structure same for all ELEMENTs */
    void RD_MRAM *act; /* Pointer to current value of ELEMENT */
};
typedef struct S_EDat CEDat;

```

### 6.3.5.2 Protocol Storage

To activate Protocol storage for an [ELEMENT](#) the [Com Type P<#>](#) has to be set. If activated the values of the **ELEMENT** are recorded over the time on the embedded system. The recorded protocol can be communicated to the [Project Monitor](#) converting the protocol to a [Data Recording](#).

In addition to the **Com Type** the library file *protocol7.rad* has to be included in the project. On how to correctly include that library file into the project is documented in the library file. The functions for storing in the according memory type must be supported by the HAL (refer to the Integration manual).

### 6.3.5.3 Remanent Data

Remanent data is data which is stored when turning off the controller and which is restored on start of the controller.

State machines (refer to [Finite State Machines](#)), counters ([ELEMENTs](#) with [Assign type CNT](#)) and timers (**ELEMENTs** with **Assign type TI**) can be marked as remanent.

State machines are marked as remanent by enabling **Remanent** in the [MACHINE](#), counters and timers by appending an “R” to the **Assign string** (refer to [Assign string of Hardware specific data types](#)).

The handling of remanent data has to be supported by the HAL (refer to the Integration manual for further information).

### 6.3.6 radCASE Index (RDI)

The radCASE Index is an index which is assigned to an [ELEMENT](#) across multiple projects or project versions. The most common use of this feature is to identify the **ELEMENT**s within [Distributed Systems](#). There are two ways to activate the generation of RDIs:

1. By setting the [Desktop](#)-Setting **EI=<#>** to a value  $\geq 0$ . This setting is used to be able to tell which **ELEMENT** is the same for different projects. For this relation radCASE compares the [SUBMODUL](#) path to the **ELEMENT**. Because this **SUBMODUL** path in most cases is different in the different projects, the Setting **EI=<#>** has to be set correctly. Using **EI=<#>** different levels of the **SUBMODUL** path are ignored.

For example, the **ELEMENT** *Element1* is used in two projects and should have the same index. In the first project, *Element1* is located at *Root.Submodul.Subnode.Element1*. In the second project *Element1* is located at *Root.Subnode.Element1*. Now to get the same index in both projects, **EI=<#>** has to be set to *EI=2* in the first project and *EI=1* in the second project. In this way, the path for *Element1* is *Subnode.Element1* in both projects.

This shorter path has to be used for the assignment over *RDI\_GLOB.IDX* (refer below) and the index will be used in the element tables in the *rdi\_pnttab.c*.

For **ELEMENT**s to appear in the *rdi\_pnttab.c* the **ELEMENT** have to have an **RDI** assigned to it, by giving the **ELEMENT** an according [Com Type](#).

2. By setting the **Desktop**-Setting **DP=<#>** to a value  $> 0$ . In this case the full path has to be used in the *RDI\_GLOB.IDX* and the index will be used in the element table in the *preserveTab.c*. Refer to [Data preservation](#) for more information.



RDIs are only generated for **ELEMENT**s for instances with a **Controller-ID**. This means the **ELEMENT** is within a **SUBMODUL** with a set **Controller-ID**. This does not have to be directly, it can also be a **SUBMODUL** of a **SUBMODUL** with the **Controller-ID**. The easiest way to achieve this is by setting the **Controller-ID** of the **System MODUL** with the **Desktop**-Setting **RI=<#>**.

For the correct assignment of the index to the **ELEMENT**s the file *RDI\_GLOB.IDX* has to be created manually. This file contains all the **ELEMENT** paths and the according indices. The file has to be located in the same directory the *Common*-directory is located. It is possible to assign two different **ELEMENT** paths the same index, in case it is the same **ELEMENT**. The *RDI\_GLOB.IDX* ensures the index does not change between different projects and different software versions.

To create the *RDI\_GLOB.IDX* the easiest way is to do a model compilation, without this file. In case of a model compilation every **ELEMENT** which has a radCASE index and is not found in the *RDI\_GLOB.IDX* will get an index assigned automatically. All of the **ELEMENT**s not found and their

assigned indices are written into the file *RDI\_DIFF.IDX* in the *Develop* directory of the project. For the first creation of the *RDI\_GLOB.IDX* this file can simply be moved and renamed.

All further modification of the *RDI\_GLOB.IDX* has to be done manually, by inserting new **ELEMENTs** from *RDI\_DIFF.IDX* or by assigning radCASE indices manually to sort the **ELEMENTs**, e.g. to use [Ctr-Setting](#) **RP=<#>**.

Comments can be added in the *RDI\_GLOB.IDX* by starting the comment with a semicolon. The comment will end at the end of the line.

The type in the file *rdi\_pnttab.c* contains the following information:

- Bit 0: Floating point ([Forced Data Type](#) of float or double)
- Bit 1..2: Assign type:
  - 1: **PAR**
  - 2: **SYS**
  - 3: **PROC**
  - 0: Other

### 6.3.7 NoValues

A **NoValue** indicates an invalid value. The **NoValue** is defined for all data types (refer to list of internal defines below) and has to be considered in the [TYPEDEF](#)-Definition of radCASE. The **NoValue** should not be used as a regular value.

If code references **NoValues** in a radCASE function, this can be done using `$NOVALUE`. This is automatically replaced by the model compiler with the correct internal value. If a **NoValue** is set within a C function outside of the radCASE model (e.g. in *usercode.c*), the C defines (refer to the list below) have to be used for each **ELEMENT** type.

Internal C-Define	Value	Comment
<b>CNOVAL</b>	(char)0x80	char NoValue
<b>UCNOVAL</b>	(unsigned char)0xFF	unsigned char NoValue
<b>BNOVAL</b>	UCNOVAL	byte NoValue
<b>SNOVAL</b>	(short)0x8000	short NoValue
<b>USNOVAL</b>	(unsigned short)0xFFFF	unsigned short NoValue
<b>LNOVAL</b>	(long)0x80000000	long NoValue
<b>ULNOVAL</b>	(unsigned long)0xFFFFFFFF	unsigned long NoValue
<b>DNOVAL</b>	(long)0	Date NoValue
<b>TNOVAL</b>	(long)0xFFFFFFFF	Time NoValue

Table 5, Internal NoValues

### 6.3.8 Daylight Saving Time

By setting the [DEFINE RD\\_AUTO\\_DLS\\_ON](#) in the model, the automatic daylight saving time mechanism is activated.

After this, if the global C-pointer **FAutoDLS** is set to the address of an [ELEMENT](#), that **ELEMENT** can be used to activate/deactivate the automatic daylight saving time.



The European daylight saving time is used. This means on the last Sunday of March the clock is put one hour forward and on the last Sunday of October the clock is put one hour back. If settings the time to a value between 2:00 and 2:59 on that last Sunday of October the value is not unambiguous. In this case the clock will be set to wintertime.

## 6.4 Behavior Modeling

radCASE supports different kinds of behavior modeling. Behavior can be modeled with C-Code using [PROC](#)s and graphically by using [SEQUENCE DIAGRAM](#)s, [Signal Diagrams](#) or [Finite State Machines](#).

Within this functionality [Element Access](#), [Behavior Access](#) and [Text Access](#) are possible.

### 6.4.1 Signal Diagrams

Signal diagrams are a dataflow-oriented way to graphically realize behavior in a radCASE model. The signal diagram is very similar to function block diagrams in IEC 61131-3 and therefore is also a way in SPS like programming.

Within a signal diagram there are mainly radCASE [ELEMENT](#)s that are connected with different [Signal Modul](#)s (function blocks). To do this, within a [SIGNAL CHART](#), the **ELEMENT**s are visualized using [ELEM](#)s and the signal **MODUL**s are visualized using [SignalIcon](#)s which reference to the according [SUBMODUL](#)s. After this the visualized **ELEMENT**s and the **ELEMENT**s of a signal **MODUL** can be connected using [Connections](#). These **Connections** also specify the direction of the data flow.

When connecting **ELEMENT**s radCASE automatically converts numerical values from smaller to bigger data sizes (e.g. short to long). A conversion from bigger to smaller data sizes should be avoided because the variables could overflow. Also a conversion between radCASE data types ([ENUM](#), [EBIN](#), [ESTR](#), [EDAT](#) and [ETIM](#)) can possibly result in hard to debug errors. To connect different data types the connection should be made through a converter ensuring the conversion does not take place accidentally. There are different predefined converters which can be found in *sig\_analog.rad*.

A signal chart can be called using \$-Access (refer to [Signal Chart Calls](#)).

#### 6.4.1.1 Signal Modul

To create a signal [MODUL](#) only a [SIGNAL ICON](#) is required in the **MODUL**, but there are different additional options available for use within signal **MODUL**s. The **SIGNAL\_ICON** itself is the interface for usage within a **SIGNAL\_CHART**. The **SIGNAL\_ICON** has to have visualizers (**ELEM**s) for all



**ELEMENTs** that need to be set within the **SIGNAL\_CHART** and all **ELEMENTs** that contain the results, because it is only possible to connect with **ELEMs** within a **SIGNAL\_ICON**.

When creating a signal **MODUL** in most cases the **MODUL** should be of **Stereotype Signal**. Because using a **SIGNAL\_CHART** normally requires to instantiate many signal **MODULs** it would be hard to distinguish them from the normal **SUBMODULs**. So using Stereotype Signal can help to improve staying on top of things.

For very simple signal **MODULs** it often makes sense to define them as **Virtual MODULs** (refer to [Virtual MODUL](#)).

Another important thing to keep in mind while creating a signal **MODUL** is the **Processing type** (refer to [Processing types](#)) of its functionality. There is a special **Processing type** for usage in signal **MODULs**. When using a **Processing type** of **SIGNAL** for functionality, this functionality is called synchronously when encountering the according **SignalIcon** within a **SIGNAL\_CHART**. All functionality with other **Processing types** will be called asynchronously, which will most likely result in slow reactions within a **SIGNAL\_CHART** on changing values.

For example if an *Input* is processed through two **SignalIcons** until reaching an *Output*, using **Processing type SIGNAL** the following steps will be executed in one execution step of the **SIGNAL\_CHART**:

1. Copy *Input* to input **ELEMENT** of signal **MODUL 1**
2. Execute functionality of signal **MODUL 1**, setting the output value of that **MODUL**
3. Copy output **ELEMENT** of signal **MODUL 1** to input **ELEMENT** of signal **MODUL 2**
4. Execute functionality of signal **MODUL 2**, setting the output value of that **MODUL**
5. Copy output **ELEMENT** of signal **MODUL 2**, to *Output*

When using **Processing type PERM**, the same execution will need three execution steps:  
Execution step 1:

1. Copy *Input* to input **ELEMENT** of signal **MODUL 1**
2. Copy old output **ELEMENT** of signal **MODUL 1** to input **ELEMENT** of signal **MODUL 2**
3. Copy old output **ELEMENT** of signal **MODUL 2**, to *Output*

Now asynchronously the functionality of both signal **MODULs** will be executed, meaning the new value of *Input* will reach the output **ELEMENT** of signal **MODUL 1**.

Execution step 2:

1. Copy *Input* to input **ELEMENT** of signal **MODUL 1**
2. Copy output **ELEMENT** of signal **MODUL 1** to input **ELEMENT** of signal **MODUL 2**
3. Copy old output **ELEMENT** of signal **MODUL 2**, to *Output*

Again asynchronously the functionality of both signal **MODULs** will be executed, which means the new value of *Input* will reach the output **ELEMENT** of signal **MODUL 2**.

Execution step 3:

1. Copy *Input* to input **ELEMENT** of signal **MODUL 1**
2. Copy output **ELEMENT** of signal **MODUL 1** to input **ELEMENT** of signal **MODUL 2**
3. Copy output **ELEMENT** of signal **MODUL 2**, to *Output*



So finally after three execution steps the changing of *Input* will result in a change of *Output*.

Many commonly used basic signal **MODUL**s can be found in the radCASE library in the *sig\_lib.rad* in the directory `%OSDL_SYS%\LIB`. This file includes the digital and analog signal **MODUL** libraries *sig\_analog.rad* and *sig\_digital.rad*.

#### 6.4.1.2 Virtual MODUL

A **Virtual MODUL** is a special kind of Signal **MODUL** (refer to [Signal Diagrams](#)) activated by the setting **Flat generation**. There will be no instance data for a **Virtual MODUL**, but it is exported Inline instead. This reduces the RAM requirement and computation time in the target system, in case of multiple calls at the cost of code length. Because there is no instance data (like **ELEMENT**s or **PROC**s) only the following items are allowed within a **Virtual MODUL**:

- Only **ELEMENT**s of **Assign type** **IN** or **OUT** (refer to [Element Usage And Allocation](#))
- Only one **PROC** as the function definition within a **METHODS** section. That **PROC** has to have a **Processing type** (refer to [Processing types](#)) of **SIGNAL**.
- **Signalcons** (refer to [Signal Diagrams](#))



Because there is no instance data exported for a **Virtual MODUL**, in the **Project Monitor** the states of the **Virtual MODUL** are not visible. For debugging purposes it can be useful to see those information, this can be done by enabling **Ctrl**-setting **VM=<0/1>** or setting the project configuration of the **Project** to **Debug**. In both cases all **Virtual MODUL**s will be exported as normal Signal **MODUL**s during model compilation.

#### 6.4.2 Finite State Machines

**STATECHARTS** are a way to graphically realize behavior in a radCASE model using UML State Machines. To define a State Machine the RC-Item **MACHINE** is used. Within a State Machine **Unit States** and **Submachines** can be used.

Every State Machine must have exactly one **STATE** or **UNITSTATE** with a **Type** of **Initial** (or **History** for **UNITSTATES**). The **Initial STATE** is the **STATE** first active when starting the system if the State Machine is not defined as **Remanent** (refer to [Remanent Data](#)).

A State Machine can be called using **\$-Access** (refer to [State Machine Calls](#)) it is also possible to get the current status of a State Machine using **\$-Access** (refer to [State Machine State Access](#)). For every call of a State Machine (regardless of being called permanently or by calling the function using **\$-Access**) one cycle of the State Machine is processed.

A cycle in the State Machine does the following steps:

1. Execute global **DURING Code** (refer below)
2. Check for global **TRANSITION Guards** (refer below)
3. Execute **ENTER Code** if **STATE** was not active in last cycle
4. Execute **DURING Code** if no global **TRANSITION** was fired
5. Check for **TRANSITION Guards** if no global **TRANSITION** was fired
6. Execute **EXIT Code** if any **TRANSITION** has fired (there are exceptions when dealing with **Unit States**)
7. Execute **TRANSITION Action** for the **TRANSITION** that has fired.

If **Guards** of multiple **TRANSITIONS** are true at the same time, the first **TRANSITION** defined in the State Machine will fire; all others will not even be checked.

By inserting a **DURING** directly into the State Machine object **MACHINE**, a global **DURING** is defined. This **DURING** will be executed in every cycle of the State Machine.

By inserting a **TRANSITION** directly into the State Machine object **MACHINE**, a global **TRANSITION** is defined. This **TRANSITION** will always be checked regardless of the currently active **STATE**. Global **TRANSITIONS** can be helpful for modeling error handling or emergency stops.

A special kind of **TRANSITION** can be made by using a [CHOICE](#). A **CHOICE** is a pseudo state which can never be active. So the **TRANSITION** to a **CHOICE** will only fire, if any outgoing **TRANSITIONS** from the **CHOICE** fire at the same time.

The [ENDSTATE](#) is handled like a normal **STATE** without any **Code** or **TRANSITION**.

#### 6.4.2.1 Unit States

A [UNITSTATE](#) is a kind of a mix between a [STATE](#) and a State Machine (refer to [Finite State Machines](#)). It has the behavior of a normal **STATE**, but like a State Machine can contain **STATES** and **UNITSTATES**. Like for a State Machine an **Initial STATE** is mandatory, however for a **UNITSTATE** there are two different kinds of **Initial STATES**.

In a **UNITSTATE** a **STATE/UNITSTATE** with a **Type** of **Initial** can be used, to start with that **STATE**, every time the **UNITSTATE** itself is the target of a [TRANSITION](#). By using a **Type** of **History** the **STATE** is only used for starting the first time the **UNITSTATE** is the target of a **TRANSITION**. After that every time the **UNITSTATE** is the target of a **TRANSITION** the last active state within the **UNITSTATE** will be active again.

The order within a cycle is mostly the same as for a State Machine without **UNITSTATES** (refer to [Finite State Machines](#)). However while in a **UNITSTATE** the following additions in the execution are made:

- After a **TRANSITION** the [ENTER Code](#) of a **UNITSTATE** is executed first and directly after this the **ENTER Code** of the now active **STATE** within the **UNITSTATE**. The following table shows which of the **ENTER Codes** are called for different kinds of **TRANSITIONS**:

Source STATE \ Destination STATE	STATE out of UNITSTATE (SOU)	UNITSTATE (SU)	STATE within UNITSTATE (SIU)
STATE out of UNITSTATE (SOU)	ENTER of SOU	ENTER of SU and SIU	ENTER of SU and SIU
UNITSTATE (SU)	ENTER of SOU	ENTER of SU and SIU	ENTER of SIU
STATE within UNITSTATE (SIU)	ENTER of SOU	ENTER of SU and SIU	ENTER of SIU

- The [DURING Code](#) and the check for a **TRANSITION Guard** of a **UNITSTATE** is executed before the **DURING Code** and check for **TRANSITION Guard** of the **STATE** within the **UNITSTATE**.

- The **DURING Code** of a **STATE** within a **UNITSTATE** will only be executed if no **TRANSITION** was fired before (global **TRANSITION** or **TRANSITION** from **UNITSTATE** itself).
- After a **TRANSITION** has fired the **EXIT Code** of a **STATE** within the **UNITSTATE** is executed before the **EXIT Code** of the **UNITSTATE**. The following table shows which of the **EXIT Codes** are called for different kinds of **TRANSITIONS**:

Source <b>STATE</b> \ Destination <b>STATE</b>	<b>STATE</b> out of <b>UNITSTATE</b> (SOU)	<b>UNITSTATE</b> (SU)	<b>STATE</b> within <b>UNITSTATE</b> (SIU)
<b>STATE</b> out of <b>UNITSTATE</b> (SOU)	<b>EXIT</b> of SOU	<b>EXIT</b> of SOU	<b>EXIT</b> of SOU
<b>UNITSTATE</b> (SU)	<b>EXIT</b> of SIU and SU	<b>EXIT</b> of SIU and SU	<b>EXIT</b> of SIU
<b>STATE</b> within <b>UNITSTATE</b> (SIU)	<b>EXIT</b> of SIU and SU	<b>EXIT</b> of SIU and SU	<b>EXIT</b> of SIU

- A **TRANSITION** to a **STATE** within a **UNITSTATE** can be made by setting the **Destination state** to *<Name of UNITSTATE>.<Name of STATE in UNITSTATE>*

#### 6.4.2.2 Submachines

A **SUBMACHINE** is a way to incorporate a State Machine into another State Machine. The code of that State Machine is incorporated, meaning the code is copied to all places a **SUBMACHINE** is used.

To use a **SUBMACHINE**, first a State Machine must be defined. To use a State Machine as a **SUBMACHINE** the State Machine has to have at least one **ENTRY POINT**. Using **UNITSTATE**s in a **SUBMACHINE** is currently not supported. If the code should return to the calling State Machine at least one **EXIT POINT** has to be defined also. The **Processing type** of a Statemachine used as **SUBMACHINE** will be ignored, but it is recommended, to use a processing type like *SUB*, to clearly identify the State Machine as **SUBMACHINE**.

After defining the State Machine, that State Machine can be used, by inserting a **SUBMACHINE** into the calling State Machine inserting the name of the State Machine to insert as **Submachine**. The **SUBMACHINE** has to have the same **ENTRY POINT**s and **EXIT POINT**s like the referenced State Machine, i.e. the same number and the same names. The **SUBMACHINE** can only be connected to the calling State Machine by using **TRANSITIONS** to the **ENTRY POINT**s and from the **EXIT POINT**s.

A **TRANSITION** to an **ENTRY POINT** is made, by setting the **Destination state** to *<Name of SUBMACHINE>.<Name of ENTRY POINT>*

#### 6.4.3 Element Access

**ELEMENT**s can be used in radCASE without paying much attention to the usage and allocation (refer to [Element Usage And Allocation](#)). radCASE will automatically generate correct access to all kind of data, so it is only required to pay attention to the data direction (not writing inputs) and data type (refer to [Data Modeling](#)). Only exception to this is the usage of Event **ELEMENT**s (refer to [Access To Event Elements](#)).

Access to the value of an **ELEMENT** is done, by using \$-Access. For this the **ELEMENT Name** is used with a Dollar: *\$<Element name>*

To access **ELEMENT**s in other **MODUL**s this can be combined with a **SUBMODUL** path (refer to [MODUL access](#)): `$<SUBMODUL path>.<Element name>`

**ELEMENT**-Arrays (refer to [Element Arrays](#)) are accessed by using square brackets around an index to specify which **ELEMENT** of the array should be accessed: `$<Element name>[<idx>]`

For all attributes of RC-Items containing direct references to **ELEMENT**s the Dollar can be dropped.

For most of the data types all assignment operators (even compound assignment operators like +=) and increment/decrement operators are available. For **ESTR ELEMENT**s only = and += are allowed. These calls are automatically converted to the according radCASE pendants of strcpy and strcat.



Even though the += operator works for strings, the + operator doesn't so don't use something like:

```
$StrElem1 = $StrElem2 + $StrElem3; // will not work
```

Instead use:

```
$StrElem1 = $StrElem2;
$StrElem1 += $StrElem3;
```



In case of string accesses, there is no automatic check/correction regarding string length.

Values of **ELEMENT**s can be specified using \$-Access (refer to [Access To Element Related Constants \(§\)](#)). There are also some special **ELEMENT** access operators described in [Special Element Access Operators](#)

#### 6.4.3.1 Access To Element Related Constants (§)

All **ELEMENT** values can be specified by using the internal data encoding, described for the different data types in [Data Modeling](#). For **ENUM**s and **EBIN**s it is also possible to specify the value using \$-Access. For every attribute that only allows an **ELEMENT** value the \$-Access is automatically used regardless of a leading §.

For an **ENUM** the values can be specified in a floating point format and using § before the value, the value is automatically converted to the according internal value (refer to [Virtual Floating Point](#)). E.g.:

```
$EnumElement = §3.4;
```

For an **EBIN** the values can be specified by using the **Name** of the according **EB ENTRY**. E.g.:

```
$MultiselectiveEbinElement = $Sel1 | $Sel2;
```

It is also possible to refer to the **NoValue** (refer to [NoValues](#)) of an **ELEMENT**. This is done by using `$NOVALUE` and works for all data types except **ESTR**, because there are no **NoValues** for **ESTR**s.



The value specified with \$-Access will be interpreted as value of the last referenced **ELEMENT** using \$-Access. So the following code will not work, as expected:

```
$EnumElem = ($EbinElem == $Sel1) ? §3.4 : §2.9; // Will not work
```

In this case radCASE will try to interpret §3.4 and §2.9 as values of the `$EbinElem`.

There is however one exception to this rule: radCASE can interpret switch-case Statements. When there is a case directly before the \$-Access radCASE will not use the last referenced **ELEMENT**, but the last referenced **ELEMENT** within a switch-Statement, so the following

code will work correct:

```
switch ($EbinElem)
{
case $Sel1:
    $EnumElem = $3.4;
    break;
case $Sel2:
    $EnumElem = $2.9;
    break;
}
```

Because the last referenced **ELEMENT** within a switch-Statement is used, nesting switch-Statements is not possible with  $\$$ -Access.

#### 6.4.3.2 Special Element Access Operators

To understand the special access methods to [ELEMENTs](#) it is advisable to understand how **ELEMENTs** are organized in radCASE first at least in the simplified explanation below.

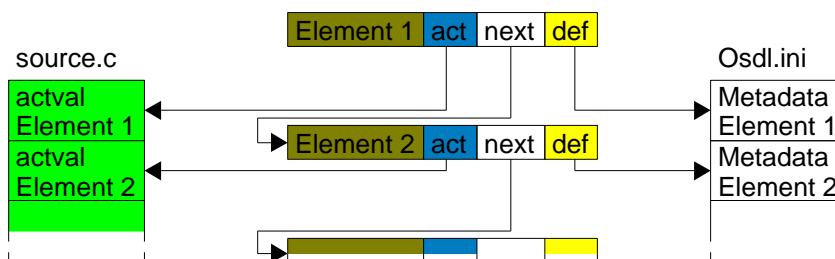


Figure 3, Element Organization

The **ELEMENTs** are stored in a linked list. So in every **ELEMENT** there is a pointer to the next **ELEMENT**. Further there is a pointer to the actual value. The  $\$$ -Access (refer to [Element Access](#)) uses this pointer to directly access the actual value of the variable (green). Finally there is the metadata of the **ELEMENT**. That metadata is stored in the binary data file *OsdL.ini*. Def (yellow) contains the offset of the metadata within the binary data.

The following special access operators are available to access that structure:

- $\$>$  (blue)** Used to get a pointer to the actual value (refer to [Access To Pointer To Actual Value \( \$\\$>\$ \)](#)) in connection with hardware specific data it accesses a special hardware structure (refer to [Access To Hardware Structure \( \$\\$>\$ \)](#))
- $\$ \#$  (yellow)** Used to get a pointer to the metadata (refer to [Access To Elements Metadata \( \$\\$ \#\$ \)](#))
- $\$*$  (dark yellow)** Used to get a pointer to the overall **ELEMENT** structure (refer to [Access To Element Pointer \( \$\\$\*\$ \)](#))

In addition to this there is the global access operator, which is used for accessing **ELEMENTs** without specifying its [SUBMODUL](#) path (refer to [Global access \( \$\\$ \sim\$ \)](#)).

##### 6.4.3.2.1 Access To Hardware Structure ( $\$>$ )

For hardware specific data (refer to [Hardware Specific Data](#)), except **RTC ELEMENTs**, there is a special hardware structure which can be accessed using  $\$>$ . There are different functions available

to call in the radCASE runtime library, which are doing special accesses and need a pointer to that hardware structure. The most commonly used are:

Functions to invert the logic of **DIs** and **DOs** at runtime:

```
di_logik($>DI_Element, <0/1>);
do_logik($>DO_Element, <0/1>);
```

Functions to activate/deactivate force mode (refer to [Force Mode](#)) and set the forced value:

```
do_force($>DO_Element, <value>);
do_unforce($>DO_Element);
ao_force($>AO_Element, <value>);
ao_unforce($>AO_Element);
```

Functions to start/stop timers (by default timers will be running, so they only have to be started, if they were stopped before):

```
ti_off($>TI_Element);
ti_on($>TI_Element);
```

#### 6.4.3.2.2 Access To Pointer To Actual Value (\$>)

Using \$> for non-hardware specific data will return a pointer to the actual value (blue in [Figure 3](#)). The pointer will be a pointer of the C-data type of the **ELEMENT**.

- [ETIM](#) and [EDAT](#) will always return a long-Pointer
- [ESTR](#) will always return a RD\_char-Pointer
- [ENUM](#) and [EBIN](#) will return different pointers, because radCASE will try to assign an optimal C-data type.



To ensure a specific C-data type for **ENUM** and **EBIN** it is best to use a **Forced data type** (refer to [Forced Data Type](#))

#### 6.4.3.2.3 Access To Elements Metadata (\$#)

Each [ELEMENT](#) possesses different attributes which are contained in its metadata. Using \$# a pointer to that metadata is returned (using the yellow offset in [Figure 3](#)). The attributes can be accessed using the following syntax: \$#<Elementname>-><Attribute>

For example:

```
$#EnumElement->minRange
```

For a list of attributes for each **ELEMENT** type refer to [Element Attributes](#)



Because the metadata is stored in the read only memory you can't change the attributes of an **ELEMENT** directly. Nevertheless you can change the attributes of an **ELEMENT** using a special mechanism (refer to [Changing Of Metadata At Runtime](#))



Because the metadata can also be stored in an external flash if not deactivated, the generated code will contain multiple macro commands. Because of this it is not possible to use \$#-notation in an expression e.g. as expression in an if-statement:



```
if ($#EnumElement->minRange > $3.0) // will not work
```

It is also not possible to define a variable and initialize it using `$#`, like:

```
long temp = $#EnumElement->minRange;
```

Both of these expressions will not work, because the design compiler generates code in the following forms:

```
{
  <Some macros to get data from external flash>
  if(<macro to get data>->minRange > $3.0)
}

{
  <Some macros to get data from external flash>
  long temp = <macro to get data>->minRange;
}
```

If access of metadata in external flash is not needed you can deactivate the export of the macros using Ctr-Setting **EMA=<0/1>** (refer to [Ctr](#)) to be able to use `$#`-notation in expressions like this.



There are no strings saved in the metadata. The values of attributes like **ELEMENT** Description or **ELEMENT** Unit consist only of the index in the text table. To get the according text you have to use the function `RD_getstr()` (refer to [Text Access](#))



radCASE tries to produce code as efficient as possible, because of this different strings are only generated into the text table, if they are used. If accessing different attributes of an **ELEMENT** in this way or indirectly, by copying the complete metadata using `memcpy` (like in the example in [Changing Of Metadata At Runtime](#)) the usage of an attribute can't be detected anymore and the text is likely to be not generated.

Even if the text is generated, e.g. by enforcing the text export with XTXT (refer to [Text Access](#)), the index in the meta data is only generated, if the usage of the text in the meta data is detected.

The export of all texts and indices for an **ELEMENT** can be enforced by using the format string **EX=1** for the **ELEMENT**, or by activating Ctr-Setting **CE=1** (will export this for all **ELEMENTS**).

Alternatively to enforce the export of a text index in the meta data, the text has to be used in a **SURFACE\_CTR**. For projects with an embedded HMI (Ctr-Setting **EH=1**) it is possible to define a dummy surface in an instantiated module, which will not be referenced in the navigation to display those texts and enforce the export of the index and the text.

#### 6.4.3.2.4 Access To Element Pointer (\$\*)

Using `$*` a pointer to the whole **ELEMENT** structure (dark yellow in [Figure 3](#)) can be accessed. These can be used for different features like [Blinking Elements](#).

#### 6.4.3.2.5 Access To SDO-Elements (\$@)

Using `$@` triggers an SDO-**ELEMENT** (refer to [CANopen communication](#) in [Distributed Systems](#)) to be sent using CANopen protocol.

E.g.:

```
$elem = $SDO-elem; // Only reading
$SDO-elem = $elem; // Writing of the local variable but no communication
$@SDO-elem = $elem; // Setting the local variable and communicating it over CANOpen
```

#### 6.4.3.2.6 Global access (\$~)

By using \$~ radCASE searches for an [ELEMENT](#) within the radCASE model. This corresponds to a great extent to the operating principles of working with global variables. This can also be combined with other special access operators (refer to [Special Element Access Operators](#)).



The global search is not unified within radCASE, meaning in some places radCASE will search the whole model starting at [The System MODUL](#) and on other places radCASE will search starting at the \$~ upwards until reaching the **System MODUL**. In all cases radCASE uses the first **ELEMENT** it finds.

Because of this, the referenced **ELEMENT** should be at a very high layer of the **MODUL** hierarchy, so it can be found going upwards from the current **MODUL** and it should have a unique name, so it is the only **ELEMENT** that can be found.



The global operator should only be used very sparsely, because it is a form of backwards reference (refer to [Recommended Project MODUL structure](#)). It should only be used for data which is centrally written at one point and is only read in the [SUBMODULs](#) (e.g. System time/date or Access Level).

#### 6.4.3.3 Access To Event Elements

An Event [ELEMENT](#) is an [EBIN](#) or [ENUM](#) of [Assign type](#) **INEVT** or **OUTEVT**. It is a container storing data as event, which can be triggered (with different values), fetched and probed. To support the full range of data an empty event **ELEMENT** will be a **NoValue** (refer to [NoValues](#)).

For accessing an event **ELEMENT** there is a special syntax, using angle brackets to add the command and any arguments to the element. The following commands are available:

- PUT** Triggering the event and putting the data into the **ELEMENT**. Previous events will be overwritten. The value is passed as argument: `$EvtElement<PUT, $12.34>`
- TEST** Probing if the event is triggered, will return the value of the **ELEMENT** or the **NoValue** if no event was triggered: `if ($EvtElement<TEST> != $NOVALUE)`
- GET** Receives the event and will return the value of the **ELEMENT** and delete the event. If no event is available will return a **NoValue**: `Value = $EvtElement<GET>`

Event **ELEMENTs** can be visualized (refer to [Element Visualization](#)) and edited with the standard edit dialogs, but they can't be used in Actions (refer to [Action Handling](#)) and [Signal Diagrams](#). The normal usage of event **ELEMENTs** is within C-Code.

#### 6.4.4 Behavior Access

Behavior can have different processing types (refer to [Processing types](#)). In case of Processing types **LOCAL** and **PUBLIC** the functionality has to be called manually. In this case it is possible to pass arguments to the function. All other functions are not allowed to receive any arguments.

To call a **LOCAL** or **PUBLIC** functionality the \$-Access is used. For this the name of the functionality is used with a Dollar: `$<functionality name>(<arguments>)`



To access functionality in other **MODULs** (only **PUBLIC**) this can be combined with a **SUBMODUL** path (refer to [MODUL access](#)): `$<SUBMODUL path>.<functionality name>(<arguments>)`

The same access is possible for [Signal Chart Calls](#) and [State Machine Calls](#), but when handling State Machines there is some additional [State Machine State Access](#).

It is also possible to handle function pointers using [Function Pointer Access \(\\$\\*\)](#).



Even though it is possible to use the same name for Methods, Signal Charts and State Machines it is strongly advised to use unique names. Using the same name will cause undefined behavior when using \$-Access.

#### 6.4.4.1 Function Pointer Access (\$\*)

The pointer access operator `$*` can be used to get a function pointer of functionality, by simply adding the functionality name after that operator: `$*<functionality name>`

To get a function pointer to a functionality in a **SUBMODUL**, this can also be combined with a **SUBMODUL** path (refer to [MODUL access](#)): `$*<SUBMODUL path>.<functionality name>`

#### 6.4.4.2 State Machine Calls

State Machines (refer to [Finite State Machines](#)) of **Processing type PUBLIC** and **LOCAL** can be called from C-Code of other functionality (refer to [Behavior Modeling](#)) using \$-Access. The **Name** of the [MACHINE](#) is the function name to call using \$-Access: `$<State Machine name>()` for example:

```
$Machine();
```

To access State Machines in other **MODULs** the **SUBMODUL** path (refer to [MODUL access](#)) can be entered in the form `$<SUBMODUL Path>.<State Machine name>()` for example:

```
$Submod.Machine();
```

Every call of a State Machine will do exactly one cycle of the State Machine (refer to [Finite State Machines](#)).

#### 6.4.4.3 State Machine State Access

There are two different ways to access the current [STATE](#) of a State Machine (refer to [Finite State Machines](#)). Both ways allow determining which the current **STATE** of the State Machine is and both of them are read only accesses.

The first way is to just getting the current **STATE**, by using \$-Access and using the [MACHINE Name](#) and appending a caret: `$<State Machine name>^` for example:

```
$Machine^
```

To get the current **STATE** of a State Machine in other **MODULs** the **SUBMODUL** path (refer to [MODUL access](#)) can be entered in the form: `$<SUBMODUL Path>.<State Machine name>^` for example:

```
$Submod.Machine^
```

It is also possible to get the current **STATE** of a [UNITSTATE](#) (refer to [Unit States](#)) by joining the State Machine name and the **UNITSTATE Name** with an underline in the form `$<State Machine Name>_<UnitState>^` for example:

```
$Machine_UnitState^
```

In the same way it is possible to access nested UnitStates e.g.

```
$Machine_UnitState_NestedUnitState^
```

All of these accesses will return the internal value of the State Machine, this value can be used in combination with `$`-Access (with the same restrictions as mentioned in [Access To Element Related Constants \(\\$\)](#)). For this a state can be used in the form `$<STATE Name in uppercase>`. So for example in a State Machine with the name *Machine* and **STATES** with the **Names** *State1*, *State2*, and *State3* this could be used in the following way:

```
switch ($Machine^)
{
    case $STATE1:
        // some code
        break;
    case $STATE2:
        // some code
        break;
    case $STATE3:
        //some code
        break;
}
```

The second way is a way to directly check if the current **STATE** is a specified **STATE**, by using the same syntax as in the first way and append the **STATE Name** directly at the caret. This way returns 1 if the **STATE** is active and 0 if not. For example:

```
if ($Machine^State)
{
    // do something
}
if ($Submodul.Machine_Unitstate^State)
{
    // do something
}
```

#### 6.4.4.4 Signal Chart Calls

Signal Charts (refer to [Signal Diagrams](#)) of **Processing type PUBLIC** and **LOCAL** can be called from C-Code of other functionality (refer to [Behavior Modeling](#)) using `$`-Access. The **Name** of the [SIGNAL CHART](#) is the function name to call using `$`-Access: `$<Signal chart name>()` for example:

```
$SignalChart();
```

To access Signal charts in other **MODULs** the **SUBMODUL path** (refer to [MODUL access](#)) can be entered in the form `$<SUBMODUL Path>.<Signal chart name>()` for example:

```
$Submod.SignalChart();
```

Every call of a Signal Chart will do exactly one execution step of the Signal Chart (refer to [Signal Modul](#)).

### 6.4.5 Text Access

It is possible to access multilingual texts of the text table (refer to [Text Table](#)) from C-Code. This can be done by using the functions listed below.

Refer to [Text Support](#) for general information on radCASE texts, and note the difference between [Short Texts](#) and [Long Texts](#).



Make sure that the desired text will actually be available on the target – refer to [Text optimization](#).

#### 6.4.5.1 RD\_getstr()

This function

- allows retrieving *a complete* text from the binary data OSDLTXT
- can only be used for short texts
- is available
  - for projects with an HMI
  - for projects without HMI (Ctr-Setting EH=0) the Ctr-Setting CEA=1 has to be set

```
RD_char RD_MRAM *RD_getstr(XPTR index, RD_char RD_MRAM *buf)
```

**index** *index* is the index in the generated text table  
Use the [Text ID name](#) of the selected text as argument for index.

**buf** *buf* is a buffer that should be big enough to contain the selected text  
The maximum size of the text is [MAX\\_CTR\\_TEXT\\_LEN](#).



Example:

To get a text with **Text ID** *XTXT\_MyOwnText* into an [ESTR](#) the following code could be used:

```
RD_getstr(XTXT_MyOwnText, $EstrElement);
```

#### 6.4.5.2 RD\_getstrPartial()

This function

- allows retrieving *a part* of a text from the binary data OSDLTXT
- can be used for long texts as well as for short texts
- is only available, if the model contains at least one long text
- For details on using this function refer to the code comment of the function in *rc\_lib\ctr\_lib\Source\ctr\_util.c*

#### 6.4.5.3 RD\_findPattern()

This function

- allows searching for a pattern in a text from the binary data OSDLTXT
- can be used for long texts as well as for short texts

- is only available, if RD\_getstrPartial() is available and the #define RD\_FIND\_PATTERN\_MAX\_LEN is non-zero – refer to the code comment of the #define in rc\_lib\Ctr\_lib\inc\defineInitializations.h
- For details on using this function refer to the code comment of the function in rc\_lib\ctr\_lib\Source\ctr\_util.c

## 6.5 HMI

radCASE offers methods to build graphical user interfaces both for the embedded system and for the **Project Monitor**. These methods include the display of texts (refer to [Text Handling](#)), graphical elements (refer to [Graphic Handling](#)) and dynamic visualization of [ELEMENT](#) values (refer to [Element Visualization](#)). Further there are methods for handling specific features of an embedded system (refer to [Target HMI](#)) and to call some functionality from the surface (refer to [Action Handling](#)).

The entire user interface methods are integrated in the [MODULs](#), which has the advantage of having reusable user interfaces which are not separated from the functionality and no separate HMI programming is required.

The interfaces are defined using surfaces (refer to [SURFACE XXX](#)), which contain the actual Visualization objects. The following different types of surfaces are supported:

<b>SURFACE_VIS</b>	Definition of a surface defining a window in the <a href="#">Project Monitor</a> .
<b>SURFACE_ICON</b>	Definition of an icon which can be used as <a href="#">ICON</a> in a <b>SURFACE_VIS</b> to open another <b>SURFACE_VIS</b> . The <b>SURFACE_ICON</b> has to be in the same <b>MODUL</b> and has to have the same <b>Surface name</b> as the <b>SURFACE_VIS</b> to open.
<b>SURFACE_CTR</b>	Definition of a surface for use as HMI on the embedded system. Only <b>Surface numbers</b> are allowed as identifier of the surface.
<b>SURFACE_PRINT</b>	Definition of a surface meant for printing. There can only be one <b>SURFACE_PRINT</b> in a <b>MODUL</b> with a <b>Surface name</b> of "0" (number). This surface is used for printing from any <b>SURFACE_VIS</b> in the <b>MODUL</b> .
<b>SURFACE_WEB</b>	Definition of a surface for usage in the <a href="#">Web visualization</a> .



Not all surface items are supported in all surfaces. For a list of supported surface items for each surface type refer to [Surface item support](#).

Multiple surfaces can be defined in the same **MODUL**; the surfaces are then identified by their **Surface name/number**. Each surface can call or integrate other surfaces using these identifiers (refer to [Surface Navigation](#)). A surface is processed cyclically by first evaluating any conditions (refer to [Conditional Drawing](#)) then drawing all static components (if required) and lastly updating all dynamic components). The order in which the visualizers in a surface are processed is always top-down, this is important for overlapping visualizers so they are drawn in the right order. For **SURFACE\_CTR** the basic processing is the same, but there are additional steps which are described in detail in the chapter [Target HMI](#).

The Target HMI can also be integrated into a **SURFACE\_VIS** using the RC-Item [DISPLAY](#), allowing simulating the Target HMI and to remote control a connected embedded system.

### 6.5.1 Surface item support

The following table contains all surface items. An X marks the item as supported on the according surface and an R marks the item supported with restrictions. See the restrictions columns for further information.

Item	VIS	ICON	CTR	PRINT	WEB	Restrictions
<a href="#">DOCTAB</a>			X			
<a href="#">DT_ENTRY</a>			X			
<a href="#">FULL</a>	X	X	X	X	X	
<a href="#">ICON</a>	X					
<a href="#">TEXT</a>	X	X	X	X	X	
<a href="#">IPIC</a>	X	X	X	X	X	
<a href="#">MENU</a>			X			
<a href="#">AMENU</a>			X			
<a href="#">EDIT</a>			X			
<a href="#">LINE</a>	X	X	X	X	X	
<a href="#">RECT</a>	X	X	X	X	X	
<a href="#">AICON</a>	X		X		X	
<a href="#">COLUMNBREAK</a>			X			
<a href="#">IF, ENDIF, ELSE</a>	X	X	X	X		
<a href="#">ELEM</a>	X	X	X	X	R	For WEB only supported with Textvisualizers, VisBinIcons and Bar-Visualizers
<a href="#">MElem</a>	X		X	X		
<a href="#">DISPLAY</a>	X					
<a href="#">AKEY</a>	X		X		X	

Item	VIS	ICON	CTR	PRINT	WEB	Restrictions
<a href="#"><u>AKEYGLOBAL</u></a>			X			
<a href="#"><u>AENTER</u></a>	X		X			
<a href="#"><u>APERM</u></a>	X		X			
<a href="#"><u>AEXIT</u></a>	X		X			
<a href="#"><u>ActionCond</u></a>			X			
<a href="#"><u>TouchKey</u></a>			X			
<a href="#"><u>SETPOS</u></a>	X	X	X	X		
<a href="#"><u>CIRC</u></a>	X	X	X	R	X	For PRINT only within a picture
<a href="#"><u>FOLDER</u></a>						
<a href="#"><u>CARD</u></a>						
<a href="#"><u>FILL</u></a>	X	X		R		For PRINT only within a picture
<a href="#"><u>ARC</u></a>	X	X	X	R	X	For PRINT only within a picture
<a href="#"><u>CBITMAP</u></a>			X			
<a href="#"><u>WBITMAP</u></a>	X	X		X	X	
<a href="#"><u>SET</u></a>	R		R			On VIS only supported for <a href="#"><u>EBIN</u></a> s and on CTR only supported for <b>EBIN</b> s and <a href="#"><u>ENUM</u></a> s.
<a href="#"><u>RES</u></a>	R		R			On VIS only supported for <b>EBIN</b> s and on CTR only supported for <b>EBIN</b> s and <b>ENUM</b> s.
<a href="#"><u>INV</u></a>	R		R			On VIS only supported for <b>EBIN</b> s and on CTR only supported for <b>EBIN</b> s and <b>ENUM</b> s.
<a href="#"><u>PUT</u></a>	R		R			Only supported for <b>EBIN</b> s and <b>ENUM</b> s.
<a href="#"><u>INC</u></a>	R		R			On VIS only supported for <b>EBIN</b> s and on CTR only supported for <b>EBIN</b> s and <b>ENUM</b> s.
<a href="#"><u>DEC</u></a>	R		R			On VIS only supported for <b>EBIN</b> s and on CTR only supported for <b>EBIN</b> s and

Item	VIS	ICON	CTR	PRINT	WEB	Restrictions
<b>ENUMs.</b>						
<a href="#"><u>AEDIT</u></a>			X			
<a href="#"><u>COPY</u></a>	R					Only supported for <b>ENUMs</b> and <b>EBINs</b>
<a href="#"><u>GOTOSURFACE</u></a>			X		X	
<a href="#"><u>CALLSURFACE</u></a>			X		X	
<a href="#"><u>RETURN (Sur)</u></a>			X		X	
<a href="#"><u>OPEN</u></a>			X		X	
<a href="#"><u>CLOSE</u></a>			X		X	
<a href="#"><u>CFUNC</u></a>			X			
<a href="#"><u>USERFUNC</u></a>						
<a href="#"><u>PROCEDURE</u></a>			X			
<a href="#"><u>PASSWORD</u></a>			X			
<a href="#"><u>PSWD_CLR</u></a>			X			
<a href="#"><u>ASKOK</u></a>			X			

### 6.5.2 Positioning

All positioning in radCASE is done in pixels or for print outputs 1/10mm. The top left corner of all surfaces is defined as the 0, 0 point. [PICTs](#) have a relative positioning system, where the top left corner of each **PICT** is defined as 0, 0.

The vertical positioning is done in a mathematically correct way, i.e. positive values will move an item up and negative down.



The mathematically correct way of positioning and the 0, 0 in the upper left corner of a surface means positioning is normally done with negative values for the Y-Coordinates.

Additionally to the absolute positioning done by entering the amount of pixels from the top left corner (e.g. 20), there are two ways of relative positioning available:

1. Relative positioning using "r". By appending the letter "r" after the amount of pixels (e.g. 15r) the position will be interpreted as a relative positioning to the previous item in the surface.

2. Relative positioning using “s”. This positioning is normally only done for the Y-Coordinate within a [MENU](#). The “s” will be interpreted as font height and will switch to the bigger size of Unicode characters (refer to [Unicode](#)). So the y-Position “-s” will be mainly used in menus to get menus where all lines are below each other regardless of the active language. This feature is only supported on **SURFACE\_CTR**.



The behavior of relative positioning in combination with conditional surfaces (refer to [IF, ENDIF, ELSE](#)) differs between **SURFACE\_CTR** and **SURFACE\_VIS**. For **SURFACE\_CTR** the positioning will refer to previous visible items, meaning the item will probably move when a condition changes. On the **SURFACE\_VIS** the positioning refers to the previous item regardless of if it is visible or not, meaning the item will never move if a condition changes.

### 6.5.3 Surface Navigation

To access surfaces from other surfaces there are different methods available.

It is possible to embed a surface in an already open surface (refer to [Embedding Surfaces into other surfaces](#)). The navigation itself is different for **SURFACE\_VIS** (refer to [Navigation in SURFACE\\_VIS](#)) and **SURFACE\_CTR/SURFACE\_WEB** (refer to [Navigation in SURFACE\\_CTR/SURFACE\\_WEB](#)).

#### 6.5.3.1 Embedding Surfaces into other surfaces

It is possible to integrate a surface into another using a [FULL](#). This however deactivates [AENTER](#) and [AEXIT](#) (refer to [Action Handling](#)) of the integrated surface. It is possible to use the global operator ~ instead of a **MODUL** reference in a **FULL**. This means the Surface to integrate is used globally in the project.



When using this, the surface should have a unique Surface name/number. The Surface should also be in a **MODUL** which is hierarchically above the **MODUL** using this syntax. It should only be used for integrating surfaces which should be used in most of the other surfaces, like a navigation bar, header, or status bar. Using ~ is a kind of backward reference, so it is recommended to not use this (refer to [Recommended Project MODUL structure](#)).

#### 6.5.3.2 Navigation in SURFACE\_VIS

Within a **SURFACE\_VIS** the method for navigating the Surfaces is the usage of [ICONs](#) which integrates a **SURFACE\_ICON** into the current **SURFACE\_VIS**. On clicking on the **ICON** the **SURFACE\_VIS** with the same Surface name like the **SURFACE\_ICON** integrated in the **SURFACE\_VIS** is opened in a new window. In the **Project Monitor** multiple **SURFACE\_VIS** can be shown at the same time. **AENTER** of a **SURFACE\_VIS** is called on opening the window of the **SURFACE\_VIS** and **AEXIT** is called on closing the window.



**AENTER** of **SURFACE\_VIS** is only called when opening the window using an **ICON**. If the window is opened at startup, because of restoration of the last active window configuration **AENTER** is not executed. This means **AENTER** of **SURFACE\_VIS(0)** in [The System MODUL](#) is never executed.



### 6.5.3.3 Navigation in SURFACE\_CTRL/SURFACE\_WEB

Within a **SURFACE\_CTRL/SURFACE\_WEB** the method for navigating the Surfaces is the usage of [GOTOSURFACE](#), [CALLSURFACE](#), [RETURN \(Sur\)](#), [OPEN](#) and [CLOSE](#). On the **SURFACE\_CTRL/SURFACE\_WEB** only the current surface is shown, and those actions will change that surface.

**GOTOSURFACE** just changes the current surface, without remembering which surface was active before.

The only difference between **OPEN** and **CALLSURFACE** is that **OPEN** can open a surface from another **MODUL** and **CALLSURFACE** can only open surface in the same **MODUL**. Both actions will save the current status to a surface stack, which enables the surface interpreter to return to the previous surface once the surface opened with **OPEN** or **CALLSURFACE** is closed again. Even though it is possible to use **OPEN** with the global operator ~ for searching a globally unique surface, it is recommended to use [AKEYGLOBAL](#) instead.

There is no difference between **CLOSE** and **RETURN**; both will restore the last status from the surface stack. Nevertheless it is recommended to differentiate between using **CLOSE** after an **OPEN** and **RETURN** after a **CALLSURFACE**.

**AEXIT** is only executed when exiting a surface. When using **CALLSURFACE** or **OPEN** the surface is still open on the surface stack, so the actions of **AEXIT** are not triggered. Analog **AENTER** is only executed when entering a surface. This means when using **RETURN** or **CLOSE** to return to a surface, the surface was still open on the stack and the actions of **AENTER** are not triggered. This leads to the following cases for navigating from *Surface 1* to *Surface 2* or returning from *Surface 2* to *Surface 1*:

Trigger on Surface 1 executed	Action for navigation	Trigger on Surface 2 executed
AEXIT	GOTOSURFACE =====>	AENTER
---	CALLSURFACE =====>	AENTER
---	RETURN <=====	AEXIT
---	OPEN =====>	AENTER
---	CLOSE <=====	AEXIT

It is also possible to return to a defined surface, when there is no key pressed for a defined time (refer to [Automatic Abort Key](#)).

The global Pointer *short RD\_MRAM \*RDpActSurNum* can be set to an **ELEMENT**. This **ELEMENT** will then always contain the current surface number.

#### 6.5.4 Font Handling

When working with fonts there are the attributes **FontSpec** and **FontSize**. Both of these attributes are combined by attaching them together to **FontSpec + FontSize**, so technically the font can be completely set using just one of those attributes. The attribute is just separated for optical reasons, so it is recommended to enter the font name if any (e.g. *Arial*) into **FontSpec** and size and attributes (e.g. *12b*) into **FontSize**.

The font handling differs between **SURFACE\_VIS** and **SURFACE\_CTR**.

The **SURFACE\_VIS** only supports one standard font in different font sizes. The font size in points is specified in the attributes **FontSpec** and **FontSize**.

The **SURFACE\_CTR** supports different fonts and attributes.



Because of limited memory space on most controllers not all fonts are supported. But it is recommended HALs should support to select which fonts are used (refer to Integration manual for further details). Even if the fonts are selectable the model designer should make a selection of only a few fonts, to reduce the generated code size.

The following types of fonts are supported on **SURFACE\_CTR**:

[Regular radCASE System Fonts](#)

[Proportional radCASE System Fonts](#)

[Proportional Operating System Fonts](#)

[Custom Fonts](#)

[True Type Fonts](#)

##### 6.5.4.1 Regular radCASE System Fonts

The regular radCASE System Fonts is a monospaced font on **SURFACE\_CTR** and works with a fixed width for every letter. The fonts are selected by entering the dimensions of the font or for most of them by just entering the height. [Table 6](#) lists all supported dimensions and for which fonts the height can be used as short form:

Dimensions (<width>x<height>)	Height (if usable as short form)
4x6	6
6x8	8
8x10	10
8x12	12
8x16	-

Dimensions (<width>x<height>)	Height (if usable as short form)
12x16	16
16x16	-
8x20	20
16x24	24
20x32	32

Table 6, Regular radCASE System Fonts

#### 6.5.4.2 Proportional radCASE System Fonts

The proportional radCASE System Fonts on **SURFACE\_CTR** have to be activated using the [DEFINE RD\\_PROPFONT](#). These fonts have different character width according to the letter (e.g. the “i” has a smaller width than the “m”). The font is selected by entering the cap height followed by the character “p”. [Table 7](#) lists all supported font heights and the according cap height inclusive the letter “p”:

Font Size	Cap height (incl letter “p”)
8	7p
12	8p
16	10p
19	12p
22	14p
24	16p
36	24p
50	36p
66	48p

Table 7, Proportional radCASE System Fonts

#### 6.5.4.3 Proportional Operating System Fonts

For more capable graphic libraries there is also support for some additional proportional fonts on **SURFACE\_CTR**. Those fonts must be enabled with the [DEFINE RD\\_PROPFONT](#). The font can be selected by entering <Font name><Font size><Attribute> e.g. *Arial12b*.

[Table 8](#) lists all supported font names, the Font size can be any number up to 255 (if supported by HAL); [Table 9](#) lists the supported attributes:

Font	Font name
Arial	<i>Arial&lt;Font size&gt;&lt;Attribute&gt;</i>
Courier New	<i>Courier&lt;Font size&gt;&lt;Attribute&gt;</i>
Terminal	<i>Terminal&lt;Font size&gt;&lt;Attribute&gt;</i>
Fixedsys	<i>Fixedsys&lt;Font size&gt;&lt;Attribute&gt;</i>
Tahoma	<i>Tahoma&lt;Font size&gt;&lt;Attribute&gt;</i>
MS Sans Serif	<i>Sansserif&lt;Font size&gt;&lt;Attribute&gt;</i>

Table 8, Proportional Operating System Fonts Names

Font style	Attribute
Regular	<i>&lt;Font name&gt;&lt;Font size&gt;</i>
Bold	<i>&lt;Font name&gt;&lt;Font size&gt;b</i>
Italic	<i>&lt;Font name&gt;&lt;Font size&gt;i</i>
Underlined	<i>&lt;Font name&gt;&lt;Font size&gt;u</i>

Table 9, Proportional Operating System Fonts Attributes

#### 6.5.4.4 Custom Fonts

radCASE supports up to six custom fonts in a project on **SURFACE\_CTR**. To use custom fonts a windows font has to be mapped to one of the custom slots (A-F). Every time that windows font is used in the model the model compiler will map it to the according custom font on the controller (**RD\_CUSTOMA** to **RD\_CUSTOMF**). To map a windows font to one of the slots, the field **Font assignment** has to be set (refer to [EMB\\_HMI](#)) by specifying *<Slot>=<Windows font>*. Multiple assignments can be comma separated, e.g. *A=Comic Sans MS, B=Verdana*.

Now the font can be used in the model like one of the [Proportional Operating System Fonts](#). You can just enter the windows font name as font name, e.g. *Comic Sans MS12b*.



The font name is not always the name used in programs like MS Word. Please refer to the font name used in the Fonts folder within the Windows Control Panel.

The windows font is used for previewing in the editor and within the simulation of the target in the **Project Monitor**. It does not have to be the same font as the one used on the target, but it is strongly recommended to use the same or at least a similar font, so the project can be properly simulated and looks like the target controller.



Fonts which contain numbers in the font name are currently not supported.

#### 6.5.4.5 True Type Fonts

radCASE supports the use of fonts derived from the Windows True Type Fonts (TTF) on **SURFACE\_CTR**. The feature also has to be supported by the according HAL by including the file *rc\_lib\ctr\_lib\source\fontdisplay.c*. Every font installed on the computer used for generating the target code can be used.

The fonts derived from true type fonts are stored in a special format. To generate the source files for these tables, the tools *sfontgen.exe* and *fontsTarget.exe* must be used.

These tables are used both in the Simulation and in the target code, therefore the simulation shows exactly the same writing as the target (Windows screen enlargement must not be used). Since the font tables are generated at compilation, the editor can only use the TTF-Font directly in the preview area of the editor. This requires the usage of the recommended font coding in the design (refer to [Font Coding In The Design](#)).

##### 6.5.4.5.1 Toolchain Requirements For True Type Fonts

The call for the functions to generate the true type font tables should be in the file *comp\_des\_post.bat*.

When using the fonts Arial16, Arial30 and Arial30 bold, this file could look like this:

```

SET FONT_FILE_C=..\CTR\APPL\BIN\fntArial16.bmp
IF NOT EXIST %FONT_FILE_C% (
"%OSDL_SYS%\TOOLS\sfontgen.exe" "Arial" 16 -- ..\CTR\APPL\BIN\fntArial16.bmp 0xFFFF
)
"%OSDL_SYS%\TOOLS\fontsTarget.exe" ..\CTR\APPL\BIN\fntArial16.bmp
..\CTR\APPL\BIN\fArial16.bmc Arial 16 2 H 2 0x20-0x7F
"%OSDL_SYS%\TOOLS\bin2c.exe" ..\CTR\APPL\BIN\fArial16.bmc ..\CTR\APPL\BIN\fArial16.c
fontArial16 1

SET FONT_FILE_C=..\CTR\APPL\BIN\fntArial30.bmp
IF NOT EXIST %FONT_FILE_C% (
"%OSDL_SYS%\TOOLS\sfontgen.exe" "Arial" 30 -- ..\CTR\APPL\BIN\fntArial30.bmp 0xFFFF
)
"%OSDL_SYS%\TOOLS\fontsTarget.exe" ..\CTR\APPL\BIN\fntArial30.bmp
..\CTR\APPL\BIN\fArial30.bmc Arial 30 2 H 2 0x20-0x7F
"%OSDL_SYS%\TOOLS\bin2c.exe" ..\CTR\APPL\BIN\fArial30.bmc ..\CTR\APPL\BIN\fArial30.c
fontArial30 1

SET FONT_FILE_C=..\CTR\APPL\BIN\fntArial30B.bmp
IF NOT EXIST %FONT_FILE_C% (
"%OSDL_SYS%\TOOLS\sfontgen.exe" "Arial" 30 B- ..\CTR\APPL\BIN\fntArial30B.bmp 0xFFFF
)
"%OSDL_SYS%\TOOLS\fontsTarget.exe" ..\CTR\APPL\BIN\fntArial30B.bmp
..\CTR\APPL\BIN\fArial30B.bmc Arial 30 2 H 2 0x20-0x7F
"%OSDL_SYS%\TOOLS\bin2c.exe" ..\CTR\APPL\BIN\fArial30B.bmc ..\CTR\APPL\BIN\fArial30B.c
fontArial30B 1

```

Since the True Type Fonts require the color table, it is recommended to create the color table in *comp\_des\_post.bat* as well. The command line could look like this :

```

REM create color table
pushd "..\CTR\APPL\BIN"

rem first delete the old one
IF EXIST coltab.c del coltab.c

"%OSDL_SYS%\TOOLS\ColTabConverter.exe" ..\..\..\DEVELOP\256coltab.bmp coltab.c ColTab 0RGB
0565 RDF16COL RDF16COLBK RDF16COLTRANS

popd

```

#### 6.5.4.5.2 Font Coding In The Design

It is recommended but not mandatory to use TTF fonts in combination with proportional operating system fonts (refer to [Proportional Operating System Fonts](#)) or custom fonts (refer to [Custom Fonts](#)). The usage of these fonts will result in the following defines used on the controller:

Fonts:

```
RD_FNTARIAL
RD_FNTCOURIER
RD_FNTTERMINAL
RD_FNTFIXEDSYS
RD_FNTTAHOMA
RD_FNTSANSSEIF
RD_CUSTOMA
RD_CUSTOMB
RD_CUSTOMC
RD_CUSTOMD
RD_CUSTOME
RD_CUSTOMF
```

These fonts will be "OR"ed with size and style settings:

```
RD_FNTBOLD
RD_FNTITALIC
RD_FNTUNDERL
```

The resulting font used on the target would be i.e. "RD\_FNTARIAL | RD\_FNTBOLD | 12".



The used font including style settings and size has to be generated by the toolchain (refer to [Toolchain Requirements For True Type Fonts](#))

#### 6.5.4.5.3 Additional Code Requirements

To use the true type fonts, the [DEFINE](#) **RD\_USE\_TTFONTS** must be defined as "1" in the [SYS-TEMDEF](#) section of the design.

The c-files generated by the *comp\_des\_post.bat* sequence described in [Toolchain Requirements For True Type Fonts](#) do not contain any #includes or other header information. They should be included into the file *usercode.c*, e.g. as follows:

```
/****** INCLUDES FONTS *****/
#include "bin\fArial30B.c"
#include "bin\fArial30.c"
#include "bin\fArial16.c"
```

Additionally, a function *void c\_dis\_SetFont(void)* must be implemented into *usercode.c* to map the font settings of the Design to the appropriate font tables. The function has to use the generated defines as described in [Font Coding In The Design](#):

```
void c_dis_SetFont(void)
{
    static unsigned short actualFont = 0 ; // currently used font
    if (actualFont != _sysFont) // new Font, set header
    {
        switch (_sysFont)
        {
            case (RD_FNTARIAL | 16) :
                pFontHeader = (rC_SFHeader *)fontArial16 ;
                break ;
            case (RD_FNTARIAL | 30) :
                pFontHeader = (rC_SFHeader *)fontArial30 ;
                break ;
            case (RD_FNTARIAL | 30 | RD_FNTBOLD) :
                pFontHeader = (rC_SFHeader *)fontArial30B ;
                break ;
            default : // not supported font
                pFontHeader = NULL ;
                break ;
        }
        actualFont = _sysFont ;
    }
}
```

When using radCASE libraries designed for regular or proportional radCASE system fonts (refer to [Regular radCASE System Fonts](#) and [Proportional radCASE System Fonts](#)), it is possible to map these radCASE fonts to the new TTF fonts in the function `c_dis_SetFont()` and thus avoid to rewrite existing libraries, e.g.:



```

void c_dis_SetFont(void)
{
    static unsigned short actualFont = 0 ; // currently used font
    if (actualFont != _sysFont) // new Font, set header
    {
        switch (_sysFont)
        {
            // these fonts are used by the system and are mapped herein to Arial
            case PROPFONT16X24 :
                _sysFont = (RD_FNTARIAL | 16) ;
                // fall thru
            case (RD_FNTARIAL | 16) :
                pFontHeader = (rC_SFHeader *)fontArial16 ;
                break ;
            case FIXFONT32X20 :
                _sysFont = (RD_FNTARIAL | 30) ;
                // fall thru
            case (RD_FNTARIAL | 30) :
                pFontHeader = (rC_SFHeader *)fontArial30 ;
                break ;
            case PROPFONT24X36:
                _sysFont = (RD_FNTARIAL | 30 | RD_FNTBOLD) ;
                // fall thru
            case (RD_FNTARIAL | 30 | RD_FNTBOLD) :
                pFontHeader = (rC_SFHeader *)fontArial30B ;
                break ;
            default : // not supported font
                pFontHeader = NULL ;
                break ;
        }
        actualFont = _sysFont ;
    }
}

```

#### 6.5.4.5.4 Font Characters Exported

The tool *fontsTarget.exe* exports patterns for all characters which are either in an area or in the file *CharsOverLimit.bin*. The file *CharsOverLimit.bin* contains all characters used in the design which are above the font character limit (refer to [Ctr-Setting FCL](#)). Areas are defined as command line parameters passed to the tool. It is strictly recommended to specify an area containing all characters up to the font character limit.



If text is used in the C-code of the design the text is disregarded for the creation of the *CharsOverLimit.bin*. If that text is used to display the text and uses characters above the font character limit, those characters must also be passed as an area to the *fontsTarget.exe*, for the according pattern to be generated.

#### 6.5.4.5.5 Pattern optimization

To further optimize the size of the font patterns created by *fontsTarget.exe*, the tool can be called with the parameter „Opti=2“. This causes a compression of the generated patterns, resulting in a reduced size of the patterns at the cost of performance.

#### 6.5.4.5.6 Transparent Mode

Writing characters with transparent background is very time consuming at runtime. Therefore to optimize drawing speed of texts, the supported transparent modes can be set using the `#define RD_SOFTFONTBACKGROUND:`

If `RD_SOFTFONTBACKGROUND` is not set or `=0`, no transparent background is supported. This setting should always be used if no writing over transparent background is required since runtime is optimal with this setting.

If `RD_SOFTFONTBACKGROUND` = 1, text over monochrome Background (transparent) is supported.

If `RD_SOFTFONTBACKGROUND` = 2, text over multi-color Background (transparent) is supported. This selection should be chosen with care since it requires a lot of performance and tends to make the drawing of texts very slow.

### 6.5.5 Target HMI

The Target HMI is modeled in different **SURFACE\_CTR**. As **Surface number** only numbers are allowed. If that number is equal or greater than 5000 some special mechanism is activated for [Global Keys](#). If the number is equal or greater than 10,000 the old content is not deleted when entering the surface, which allows drawing of dialogs or edit panels.

In the **SURFACE\_CTR** some features can be used, which are only available for the embedded system. These features contain graphical items, like [Menus](#) and [Customizing Standard System Dialogs](#). Additionally there is some hardware specific support like [Keyboard Handling](#) and [Touch Support](#).

A **SURFACE\_CTR** like any other [HMI](#) is processed cyclically, but there are additional steps. The whole processing order is as follows:

- Execute [AENTER](#) (only in first run after surface change)
- Evaluate keyboard and touch events
- Execute [ActionConds](#) (not in first run after surface change)
- Evaluate any conditions (refer to [Conditional Drawing](#))
- Draw all static components (only if complete redraw is necessary)
- Update all dynamic components and execute [APERM](#)

Because the whole surface is already drawn when executing the **APERM**, the **APERM** can call functions which draw user defined objects to the screen.

#### 6.5.5.1 Menus

A [MENU](#) defines a line oriented user menu. Within a menu there are [AMENU](#)s, [EDIT](#)s and [AICON](#)s which can be selected and trigger actions (refer to [Action Handling](#)) or edit **ELEMENT**s. Additionally there are static texts and different graphical items which can't be selected. The color of the selected line is specified by the **Selected item Foreground/Background color** or in case those are not specified by the global foreground/background color of the menu (depending on the cursor type by either using those colors or inverting them). For **AICON**s the **PICT ID (pressed)** is used to highlight the selected item.

It is possible to specify multicolumn menus. To do so each column must be interrupted by a [COLUMNBREAK](#). The **COLUMNBREAK** does not affect the positioning of the menu items, but will specify the next column for navigation with left/right.



When using multicolumn menus the use of PGUP and PGDN is not supported.

The behavior of the menu is mostly defined by the **Cursor type** which is a bit mask with the following options:

- Bit 0...7**            Cursor mode (refer below)
  
- Bit 8 (0x100)**    0 = Scroll with up/down key  
                          1 = Scroll with left/right key
  
- Bit 9 (0x200)**    0 = No vertical jumping in menu  
                          1 = Jump back up to first line after last entry in column (or vice versa)
  
- Bit 10 (0x400)**   0 = No automatic exit from menu  
                          1 = Exit menu after last menu entry (or before first entry)
  
- Bit 11 (0x800)**   0 = No user defined scrollbar  
                          1 = User defined scrollbar (refer to [User defined scrollbar](#))
  
- Bit 12 (0x1000)**   0 = No horizontal jumping in menu  
                          1 = Jump back left to the first column after last entry in row (or vice versa)
  
- Bit 13 (0x2000)**   0 = No scrollbar  
                          1 = show scrollbar next to the menu (does only work with one column menus)
  
- Bit 14...15**       Determines which line is selected when changing column:  
                          0 = Select n-th line in new column, when coming from n-th line in old column  
                          0x4000 = Always select first line in column after column change  
                          0x8000 = Call a user defined function to determine new line (refer to [User defined column change](#))



If activated the scrollbar will be drawn at the right side of the menu, but not inside of the menu reserved space. So keep in mind, when using scrollbars the menu will be wider than the reserved menu space. For touch displays with more than 64 pixels height, the scrollbar will not use the full height of the menu, but will reserve space for arrow buttons for scrolling up and down.

The following cursor types are supported in a menu:

- 0 - No cursor**    The current selected line will not be highlighted
  
- 1 - Cursor**        The current selected line will be highlighted by drawing a cursor to the left of the menu. The cursor will be left of the specified menu area, so the menu should leave some space to the left, if using this cursor. The cursor size is also influenced by the font size of the menu
  
- 2 - Blinking line**   The current selected line will be highlighted by blinking the text of the line in the selection/inverted colors.
  
- 3 - Line select**     The current selected line will be highlighted by drawing the text of the line in the selection/inverted colors.
  
- 4 - Line select (whole)**   The current selected line will be highlighted by drawing the whole line (full menu width) in the selection/inverted colors.

line)

- 5 - Graphical line select** The current selected line will be highlighted by drawing the whole line (full menu width) in the selection/inverted colors. After drawing the line all non-selectable items after the current line will be redrawn. This can be used to draw menu entries with graphical items in the selected line.
- 6 - User defined cursor** The current selected line will be highlighted with a user defined function. Refer to [User defined cursor](#)

The cursor can be disabled and enabled by calling the function

```
void SetMenuCursorVisible(unsigned char now);
```

The scrolling in the menus can be influenced system wide with the [DEFINE RD\\_MENU\\_SCROLL\\_TYPE](#). The following define values are available:

- RD\_SINGLESTEP** For scrolling the menu one line at a time, when selection is out of display
- RD\_HALFPAGE (default)** For scrolling the menu for a half page. The active selection will be brought to the middle of the menu, when selection is out of display.
- RD\_FULLPAGE** For scrolling a whole page when selection is out of display.



Scrolling assumes the first line in the menu is selectable and positioned at the start of the menu. It is not possible to scroll upward further than the first selectable line. Also the cursor will be positioned according to the Define above for selectable items. It is possible unselectable items will not show in the menu, because the menu scrolls over them, depending on the scroll setting and the distance to any selectable lines.

The selection in a menu can be moved in the following ways:

- By using up, down, left and right keys to change lines and columns
- By selecting the menu line with the touch support (refer to [Touch Support](#))
- By entering the initial character on a keyboard (if there are multiple menu entries with the same beginning character the cursor jumps from entry to entry for each press.
- By using the function: `void setMenuCursorPosition(short column, short line)`  
The column and line are 0-based indices.

When selecting a line directly with touch or pressing <ENTER> the actions or editing of the selected line are started. Scrolling can also be done by dragging a selectable line up/down with the touch.

There are also some global C variables that can be accessed read only from C code to get status of the current menu:

Global C variable	Description
short RDnumMenu	Overall number of menu items

Global C variable	Description
short RDAcLine	Active menu item relative to top of menu
short RDAcMenuLine	Active menu item relative to top of currently visible section of the menu
short RDAcValPosX	X Position of the currently selected <b>ELEMENT</b>
short RDAcValPosY	Y Position of the currently selected <b>ELEMENT</b>
CElement RD_MROM *RDAcElement	Pointer to currently selected <b>ELEMENT</b>

#### 6.5.5.1.1 User defined scrollbar

To use a user defined scrollbar in a [MENU](#), Bit 11 (0x800) of **Cursor type** has to be set. If this bit is set in any menu, the global pointer *RDfp\_drawUserScrollBar* will be available. This pointer has to point to a user defined function with the prototype:

```
void drawUserScrollBar(M_D_SCR DVisMenuBegin RD_MBIN *thisvis, short itemCount);
```

**thisvis**      Pointer to the menu structure. Can be used to get information on location and size of the menu and the number of visible items.

**itemCount**    Total number of visible items in the menu.

The following example code shows how such a function could look like:

```

RDfp_drawUserScrollBar = &drawUserScrollBar;
void drawUserScrollBar(M_D_SCR DVisMenuBegin RD_MBIN *thisvis, short itemCount)
{
    short sb_sizeY;
    short posX = thisvis->xr + 2;
    short posY = thisvis->yo;
    short anzMenItems;

    if ((VARPOOLSHORT(thisvis->numMenuIdx)) > itemCount)
    {
        anzMenItems = min(itemCount, ((VARPOOLSHORT(thisvis->numMenuIdx)) -
            (VARPOOLSHORT(GUniquePoolOffset + thisvis->startLineIdx))));

        sb_sizeY = (((thisvis->yo - thisvis->yu) * 10) / (VARPOOLSHORT(thisvis->numMenuIdx))
            * anzMenItems) / 10;
        posY -= (((thisvis->yo - thisvis->yu) * 10) / (VARPOOLSHORT(thisvis->numMenuIdx))
            * (VARPOOLSHORT(GUniquePoolOffset + thisvis->startLineIdx)) / 10);
        c_dis_clrrect1(M_SCR_K posX+2, thisvis->yo, 16, -(thisvis->yo - thisvis->yu)
            M_S_COLBK );
        c_dis_rect1(M_SCR_K posX+8, thisvis->yo, 4, -(thisvis->yo - thisvis->yu), 1
            M_A_COL(thisvis->col), 1);
        c_dis_rectext1(M_SCR_K posX+2, posY, 16, -sb_sizeY, 6 M_A_COL(thisvis->colselbk),
            -30, -30, 0x1F);
    }
}

```

#### 6.5.5.1.2 User defined cursor

To use a user defined cursor in a [MENU](#), **Cursor type 6** has to be selected. If this cursor type is selected in any menu, the global pointers *RDfp\_updateUserCursor* and *RDfp\_belongsToUserCursor* will be available.

The pointer *RDfp\_updateUserCursor* is used to draw the cursor and has the following prototype:

```
void drawUserCursor(M_D_SCR DVisMenuBegin RD_MROM *thisvis);
```

**thisvis** Pointer to the menu structure. Can be used to get all needed information of the menu



The implementation of the user defined cursor also has to remove the selection of the previously selected line.

The following example code shows how such a function could look like:

```

extern XPTR UVisObj_nextEnabledMenuItem(XPTR nextIdx, RD_OSDL_PNT(union UVisObj, vis)
                                         M_D_HMIID_KV);
extern XPTR visMenu_getCondSafeMenuItemPos(M_D_SCR XPTR startvisIdx, RD_OSDL_PNT(union
                                         UVisObj, cursorvis), short num, short RD_MRAM *px, short RD_MRAM *py);
extern XPTR visMenu_getMenuItemPos(M_D_SCR XPTR startvisIdx, RD_OSDL_PNT(union UVisObj,
                                         cursorvis), short num, short RD_MRAM *px, short RD_MRAM *py);
extern void UVisObj_getHeight(union UVisObj RD_MBIN *vis, short RD_MRAM *height);
extern void UVisObj_drawMenu(M_D_SCR union UVisObj RD_MBIN *thisvis, short menuindex,
                             unsigned char selected, short drawmode M_D_COL M_D_COLBK);

RDfp_updateUserCursor = &drawUserCursor;
void drawUserCursor(M_D_SCR DVisMenuBegin RD_MROM* vmb)
{
    short fntHeight;
    RD_DECL_OSDL(union UVisObj, vis);
    XPTR visIdx;
    XPTR startvisIdx;
    short x = 0, y = 0;

    startvisIdx = UVisObj_nextEnabledMenuItem(vmb->next, &vis M_S_HMIID_KV);

    if (VARPOOLSHORT(vmb->oldLineIdx) >= 0) /* if a line was previously selected */
    {
        visIdx = visMenu_getCondSafeMenuItemPos(M_SCR_K startvisIdx, &vis,
                                                (VARPOOLSHORT(vmb->oldLineIdx)), &x, &y);

        if (visIdx != RD_NOBINDATA) /* In case of conditional surfaces, the old line could
                                     be hidden */
        {
            UVisObj_getHeight(&(RD_OSDL(vis)), &fntHeight);
            c_dis_clrrect1(M_SCR_K 0, y, vmb->xr-vmb->x1, -fntHeight M_A_COL(vmb->colbk));

            UVisObj_drawMenu(M_SCR_K &(RD_OSDL(vis)), (VARPOOLSHORT(vmb->oldLineIdx)), FALSE, 1
                            M_A_COL(vmb->col) M_A_COL(vmb->colbk));

            /* Set standard font of menu (e.g. for Description of VisBinIcon)
               Has to be done again, because font could be overwritten by UVisObj_drawMenu */
            setFont(vmb->fontsize UNI_DYNAMIC);
        }
    }
    visMenu_getMenuItemPos(M_SCR_K startvisIdx, &vis, RDAcLine, &x, &y);
    UVisObj_getHeight(&(RD_OSDL(vis)), &fntHeight);
    UVisObj_drawMenu(M_SCR_K &(RD_OSDL(vis)), RDAcLine, TRUE, TRUE M_A_COL(vmb->col)
                    M_A_COL(vmb->colbk));

    c_dis_rect1(M_SCR_K 0, y, vmb->xr-vmb->x1, -fntHeight, 1 M_A_COL(vmb->colselbk), 0);
}

```

The pointer *RDfp\_belongsToUserCursor* is used to determine if a menuitem belongs to the currently selected menuitem. This option can be used, for menus where one line in the menu consists of multiple visualizers to prevent redrawing of items over the current cursor. The pointer has the following prototype:

```
char belongsToUserCursor(DVisMenuBegin RD_MBIN *thisvis, short itemIdx);
```

**thisvis**      Pointer to the menu structure. Can be used to get all needed information of the menu

**itemIdx** Index of the menu line in menu to check

#### 6.5.5.1.3 User defined column change

To use a user defined column change in a [MENU](#), Bit 15 (0x8000) of **Cursor type** has to be set. If this bit is set in any menu, the global pointer *RDfp\_changeColumn* will be used. This pointer can be set to point to a user defined function with the prototype:

```
short userChangeColumn(DVisMenuBegin RD_MBIN *thisvis, short firstLineInCol);
```

**thisvis** Pointer to the menu structure. Can be used to get information on location and size of the menu and the objects within the menu.

**firstLineInCol** This is the index of the first line within the new column. The index is an overall index of all RC-Items within the menu.

The function returns the overall index of the new selected line.

#### 6.5.5.2 Keyboard Handling

On the [Target HMI](#) radCASE supports keyboard handling, to react to keyboard events of a keyboard of the embedded system.

The build in target HMI functionality like [Menus](#) or Edit dialogs require the following keys for a correct functionality:

<b>ESC</b>	E.g. to abort editing an <b>ELEMENT</b>
<b>CR or WHEEL_PRESS</b>	E.g. to execute a menu entry or confirm the editing
<b>UP/DOWN or WHEEL_CW/WHEEL_CCW</b>	E.g. for selecting menu entries or editing values

If some of those keys are not available on the controller, the functionality can be assured by using keyboard mapping (refer to Integration manual). Alternatively the build in dialogs can be operated by touch (refer to [Touch Support](#)).

To call user defined functionality, navigate through surfaces and edit **ELEMENT**s the user can define actions (refer to [Action Handling](#)) to execute when a specific keyboard event occurs. This can be done using [AKEY](#), [AKEYGLOBAL](#) and [AICON](#). For more details on AKEYGLOBAL refer to [Global Keys](#).

The keys on which the RC-Items should react can be defined in the following ways:

- By entering the decimal value from ASCII or Unicode table e.g. 27 (for <ESC>)
- By entering the character in single quotation marks e.g. 'a'
- By entering one of the predefined key values (refer to [Key Values](#)) e.g. ESC

##### 6.5.5.2.1 Global Keys

Like [AKEY](#) the [AKEYGLOBAL](#) reacts on a key, but **AKEYGLOBAL**s react globally on keys regardless of which surface is active. By triggering an **AKEYGLOBAL** a central surface will be opened and the actions (refer to [Action Handling](#)) of the **AKEYGLOBAL** will be executed.





Local key handling has priority over global key handling, so if the key is handled by an [AKEY](#) or an [AICON](#) in the active surface the **AKEYGLOBAL** will not be triggered. This can be used to actively prevent an **AKEYGLOBAL** from being executed e.g. to prevent surface stack overflows on using multiple **AKEYGLOBAL**s to open central surfaces.

There are two slightly different features that can be triggered with an **AKEYGLOBAL**:

- [Return To Central Surface](#): This feature will leave the current surfaces and go back to a specified surface. This can e.g. be used to implement a home screen to return to.
- [Opening A Central Surface](#): This feature will open a central mask, which can be closed to return to the surface which was open before triggering the **AKEYGLOBAL**. This can e.g. be used to be able to show a special status surface from everywhere.

### Prerequisites

To enable this feature the [Prerequisites](#) of the [Automatic Abort Key](#) must be met. If the functionality of the automatic abort key is not desired, the timer can be disabled as described in [Disabling Automatic Abort At Runtime](#).

Additionally the following code must be added in both of the functions `short key_pressed(short key)` and `short key_pressed_stat(M_D_SCR short key)` which are normally located in the `usercode.c`:

```
#ifdef T_KEY_ABORT
    if (testGlobalKeyAbort())
        return abortGlobalKey;
#endif /* #ifdef T_KEY_ABORT */
```

### Return To Central Surface

The **AKEYGLOBAL** should be defined in the central surface to return to. That surface should be in a central [MODUL](#). It is recommended to either put the surface into [The System MODUL](#) or into the main **MODUL** of the project if using the [Recommended Project File Structure](#).

Upon triggering the **AKEYGLOBAL** the surface interpreter will [CLOSE/RETURN \(Sur\)](#) (refer to [Surface Navigation](#)) until any surface in the **MODUL** the **AKEYGLOBAL** is defined in is found and will then use [GOTOSURFACE](#) to change to the surface the **AKEYGLOBAL** is defined in.

After this the Actions of the **AKEYGLOBAL** are executed.



Because of the repeated **CLOSE/RETURN (Sur)** it has to be ensured that any surface of the **MODUL** the **AKEYGLOBAL** is defined in can be reached with repeated **CLOSE/RETURN (Sur)** everytime the **AKEYGLOBAL** is triggered. If there are surfaces where this can't be ensured, the key code of the **AKEYGLOBAL** should be handled there locally, to disable the **AKEYGLOBAL**.



Because on triggering the **AKEYGLOBAL** the surface interpreter will only return into the first surface in the correct **MODUL**, special attention must be spent on any surface navigation within that **MODUL**, to prevent surface stack overflows. [CALLSURFACE](#) within that **MODUL** should be avoided and if necessary (e.g. also using **AKEYGLOBAL** with [Opening A Central Surface](#)) the **AKEYGLOBAL** should be disabled in the called surface and all surfaces navigated from there using local key handling of the key code.

### Opening A Central Surface

For this feature a Surface with a surface number  $\geq 5000$  has to be defined. That surface will be the surface opened on triggering the **AKEYGLOBAL**. The **AKEYGLOBAL** has to be defined in any other surface in the same **MODUL** and has to contain a [CALLSURFACE](#) to that surface.



The actions of the **AKEYGLOBAL** are executed as if triggered in the surface where the **AKEYGLOBAL** is triggered, with the only difference that the **CALLSURFACE** will open the surface in the **MODUL** the **AKEYGLOBAL** is defined in. So any access to elements within such an action will fail at runtime. So it is strongly recommended to use the **CALLSURFACE** as the only action of the **AKEYGLOBAL**.

Even though the **AKEYGLOBAL** and Surface may be anywhere in the project it is recommended to also put this into a central **MODUL** like the **AKEYGLOBAL**s used for [Return To Central Surface](#).



The surface number of the surface  $\geq 5000$  has to be unique in the whole project and the **MODUL** containing the surface may only be instantiated once.

On [CLOSE/RETURN \(Sur\)](#) the surface interpreter will return to the surface, which was active when the **AKEYGLOBAL** was triggered.



Because the surface  $\geq 5000$  will be opened using the surface stack, to prevent surface stack overflows all **AKEYGLOBAL**s must be disabled by local handling those keys within that surface. This includes the **AKEYGLOBAL** used to open that surface.

#### 6.5.5.2.2 Automatic Abort Key

If no key has been pressed for a specified time an automatic abort mechanism can be activated returning to a predefined central surface.

#### Prerequisites

To use this feature the following [DEFINE](#)s has to be set:

- |                      |  |
|----------------------|--|
| <b>T_KEY_ABORT</b>   | Duration in seconds until the abort mechanism is activated if no key was pressed |
| <b>SUR_KEY_ABORT</b> | Surface number to return to  |

If the abort mechanism is triggered the keyboard handling has to repeatedly return the key **KEYABORT** until the mechanism has reached the specified surface. This is done, by adding the following code to the function *short key\_pressed(short key)* normally located in the *usercode.c*:

```

#ifdef T_KEY_ABORT
    /* test for key abort */
    if (testAbort())
    {
        return abortKey;
    }
    /* if normal key */
    else
    {
        ResetAbortTimer();
    }
#endif /* T_KEY_ABORT */

```

The following code has to be added to the function *short key\_pressed\_stat(M\_D\_SCR short key)* also normally located in the *usercode.c*:

```

#ifdef T_KEY_ABORT
    if (testAbort())
        return abortKey;
#endif /* T_KEY_ABORT */

```

## Usage

By setting the **DEFINES** specified in the [Prerequisites](#) the feature is already activated. If the system internal timer reaches the time specified by **T\_KEY\_ABORT** the key code **KEYABORT** is sent to the surface interpreter. This causes the surface interpreter to close surfaces until it reaches the specified surface (**SUR\_KEY\_ABORT**) or until it reaches a surface which processed the key **KEYABORT** (e.g. with an [AKEY](#)).



Because the surfaces are closed until reaching the surface specified in **SUR\_KEY\_ABORT** it is mandatory that surface is reachable by repeated [RETURN \(Sur\)/CLOSE](#). If no surface with that surface number is reached the Surface Interpreter will be terminated.

## Disabling Automatic Abort At Runtime

The system timer used for the automatic abort mechanism can be reset by calling the system function *ResetAbortTimer()*. This resets the timer so this functions can be used in an [APERM](#) to prevent the abort mechanism for a specific surface.

To disable the automatic abort permanently for all surfaces the timer can be deactivated. This can be done by calling the function:

```
ti_off(&timKeyAbort);
```

The function can be called from a [PROC](#) with a **Processing Type** of **INIT**. If the automatic abort should only be deactivated for a surface and all subsurfaces, the function could be called in a **PROC** which is called using a [PROCEDURE](#) within an [AENTER](#). In this case the timer is normally activated again from an [AEXIT](#) in the same surface.

After deactivating the timer the timer can be reactivated by calling the function:

```
ti_on(&timKeyAbort);
```

### 6.5.5.3 Touch Support

On the **SURFACE\_CTR** radCASE supports touch handling. To activate this feature the **System-code** Setting **TOUCH\_ON** has to be activated (refer to [Systemcode](#)). If activated many of the system functionalities have built in touch support (e.g. **ELEMENT** visualizations (refer to [Element Visualization](#)) can be clicked on to open an edit dialog, or menus can be operated with touch (refer to [Menus](#)).

To use touch support in user defined surfaces [AICONs](#) and [TouchKeys](#) can be used. Both of these RC-Items support a hover effect, highlighting a button as long as the touch is pressed in that area. For most cases the **AICON** will be the first choice. The **TouchKey** is mainly interesting for building virtual keyboards and using an [AKEY](#) to react on **KEYOTHER** and calling functions that will get the key code from the surface stack. This usecase is described in detail in [Customizing Standard System Dialogs](#). The TouchKey can also make sense to trigger an [AKEYGLOBAL](#).

### 6.5.5.4 Customizing Standard System Dialogs

It is possible to overwrite the existing standard system dialogs in radCASE. To do this there are several function pointers which can be set to a function defining the new dialogs.

In the radCASE library there is an implementation of overwritten dialogs in the *tvis\_dialogs.rad*. In this library file most of the dialogs are already overwritten using those pointers, and the **MODULs** in the file can be used as base-**MODULs** (refer to [Inheritance](#)). To customize the look and/or the behavior those dialogs can be overwritten and edited within radCASE.

There are function pointers available if not using the predefined functionality within *tvis\_dialogs.rad* for:

- [Customizing Element Edit Dialogs](#)
- [Customizing Password Dialog](#)
- [Customizing RTC Dialogs](#)
- [Customizing ASKOK Dialogs](#)
- [Customizing Diagnosis/Calibration Dialogs](#)
- [Customizing No-Authorization Dialogs](#)

#### 6.5.5.4.1 Customizing Element Edit Dialogs

The following function pointers are available for overwriting editing of [ELEMENTs](#):

<b>CElement_P_edit_dialog</b>	Called for editing <b>ELEMENTs</b> in an <a href="#">ELEM/EDIT</a> with an <b>Editor type</b> of <b>Dialog</b> .
<b>CElement_P_edit_toggle</b>	Called for editing <b>ELEMENTs</b> in an <b>ELEM/EDIT</b> with an <b>Editor type</b> of <b>Toggle</b> .
<b>CElement_P_edit_dauer</b>	Called for editing <b>ELEMENTs</b> in an <b>ELEM/EDIT</b> with an <b>Editor type</b> of <b>Permanent</b> .
<b>CElement_P_edit_action</b>	Called for editing <b>ELEMENTs</b> by calling the Action <a href="#">AEDIT</a> .
<b>CElement_P_edit_focus</b>	Called for editing <b>ELEMENTs</b> in an <b>ELEM/EDIT</b> with an <b>Editor type</b> of <b>Focus</b> .

All of the above function pointers have the following function prototype:

```
short func(M_D_SCR CElement RD_MROM *elem, short key);
```

**elem** a pointer to the internal **ELEMENT** structure of the **ELEMENT** to edit

**key** The key code (refer to [Key Values](#)) used to trigger the editing

The function returns a short in the same format as that which is expected for functions called from an action (refer to [Functionality Calling Actions](#))

#### 6.5.5.4.2 Customizing Password Dialog

The following function pointer is available for overwriting a password check:

**RDfp\_PasswordTest** Called when using [PASSWORD](#).

The above function pointer has the following function prototype:

```
short func(M_D_SCR short level);
```

**level** The password level to check for

The function returns *TRUE* if the current password level matches the required level and otherwise *FALSE*.

#### 6.5.5.4.3 Customizing RTC Dialogs

The following function pointers are available for overwriting RTC functionalities

**RDfp\_SetRtcTime** Called when using [CFUNC](#) with **C function** of **RTC\_SET** or **RtcTimeSet**

**RDfp\_SetRtcDate** Called when using **CFUNC** with **C function** of **RTC\_SET** or **RtcDateSet**

The above function pointers have the following function prototype:

```
short func(M_D_SCR long RD_MRAM *val);
```

**val** Current value of date/time

The function returns *TRUE* if the RTC value was modified and otherwise *FALSE*.

#### 6.5.5.4.4 Customizing ASKOK Dialogs

The following function pointers are available for overwriting [ASKOK](#) dialogs:

**RDfp\_WarnAskOk** Called for **Type of question 1** (Critical Function!)

**RDfp\_AskOk1** Called for **Type of question 3** (Save Changes?)

**RDfp\_AskOk2** Called for **Type of question 2** (Execute?)

**RDfp\_AskDel** Called for **Type of question 4** (Delete?)

The above function pointers have the following function prototype:

```
short func(M_D_SCR_S);
```

The function returns *JUMPOVER* if the dialog was cancelled or otherwise 0.

#### 6.5.5.4.5 Customizing Diagnosis/Calibration Dialogs

The following function pointers are available for overwriting Diagnosis/Calibration dialogs:

<b>RDfp_DiagDi</b>	Called when using <a href="#">CFUNC</a> with a <b>C function</b> of <b>DIAG_DI</b>
<b>RDfp_DiagDo</b>	Called when using <b>CFUNC</b> with a <b>C function</b> of <b>DIAG_DO</b>
<b>RDfp_DiagAi</b>	Called when using <b>CFUNC</b> with a <b>C function</b> of <b>DIAG_AI</b>
<b>RDfp_DiagAo</b>	Called when using <b>CFUNC</b> with a <b>C function</b> of <b>DIAG_AO</b>
<b>RDfp_DiagCnt</b>	Called when using <b>CFUNC</b> with a <b>C function</b> of <b>DIAG_CNT</b>
<b>RDfp_KaliAi</b>	Called when using <b>CFUNC</b> with a <b>C function</b> of <b>KALI_AI</b>
<b>RDfp_KaliAo</b>	Called when using <b>CFUNC</b> with a <b>C function</b> of <b>KALI_AO</b>
<b>RDfp_KaliCnt</b>	Called when using <b>CFUNC</b> with a <b>C function</b> of <b>KALI_CNT</b>

The above function pointers have the following function prototype:

```
short func(M_D_SCR_S)
```

The function always returns *TRUE*.

#### 6.5.5.4.6 Customizing No-Authorization Dialogs

The following function pointer is available for overwriting No-Authorization dialogs:

<b>RDfp_NoAuthorization</b>	Called when <a href="#">PASSWORD</a> level does not fit after trying to edit an <b>ELEMENT</b> with given password level.
-----------------------------	---

The above function pointer has the following function prototype:

```
short func(M_D_SCR_S)
```

The function returns a short in the same format as that which is expected for functions called from an action (refer to [Functionality Calling Actions](#)).

### 6.5.6 Text Handling

Text handling in radCASE can be done by using static multilingual texts and by using [ESTR-ELEMENT](#)s. The texts can be formatted using special syntax (refer to [Text Formatting](#)) and there can be some [Placeholders](#) within texts, which will be evaluated to show different texts. If updating of **ESTRs** doesn't work correctly, please refer to [Custom Hash Function](#).

#### 6.5.6.1 Text Formatting

In radCASE, texts can be specially formatted using the following prefix strings:

- \n** Line break, where the start position of the new line is the same as the start position of the text output (i.e. not the left edge of the screen or display).
- ##<code>** Extended characters. Since XML cannot handle characters with character code greater than 127 directly, there is the option of defining characters by specifying the character code number. Every digit after **##** until the first non-numerical character will be evaluated as the code.  
E.g. **##97b** as well as **##0097b** will be displayed as "ab".

The following formats can only be used on embedded displays:

- \r** Line break, where the start position of the new line is equal to 0 (i.e. the left border of the screen or display).
- \\** Shows a backslash
- lc##<pos>x** Cursor controls. It sets the cursor to position <pos> of the current text line. Positions are starting with 1.  
E.g. **lc##1x** sets the cursor at position 1 (first character in the current line), while **lc##31x** sets the cursor at position 31.

#### 6.5.6.2 Placeholders

There are different placeholders that can be used as variable texts, depending on different conditions (i.e. multiple instances of a module where each instance should use different texts).

The following placeholders are available:

- \$DS** Can be used to insert the description of the current surface into a [TEXT](#) or to insert the description of a referenced surface ([GOTOSURFACE](#)/[CALLSURFACE](#)/[OPEN](#)) into the **Title** of an [AMENU](#).
- \$DM** Can be used to insert the description of the current **MODUL** into a **TEXT** or to insert the description of a referenced **MODUL** (**OPEN**) into the **Title** of an **AMENU**.  
Additionally **\$DM** can be used within the surface description to insert the description of the current **MODUL** into the surface description.  
At last in **SURFACE\_VIS** **\$DM** can be used within the element description of [Text Visualizers](#) and [VISBINICON Visualizers](#).  
In all places **\$DM** can be used except for code generated for the controller, it is also possible to reference upper modules of the referenced module, by inserting a backward reference **\_**. E.g. **\$\_DM** for the topmodule or **\$\_\_DM** for two layers up.
- \$DE** Inserts Description of **ELEMENTs** that are referenced within an action of an **AMENU**.
- \$VE** Inserts current Value of **ELEMENTs** that are referenced in an action of an **AMENU**.
- \$UL** Inserts a User-defined Label. Can only be used in a **SURFACE\_CTR** for visualization object **TEXT** and within visualization object **AMENU**.

To use a user-defined label you have to set the global pointer **RDfp\_getUserLabel** to a user-defined function, during initialization.

The user-defined function has to have the following format:

```
void getUserLabel(RD_char RD_MRAM *label, RD_char RD_MRAM *retStr);
```



The variable label provides the function with a string of everything that was entered behind the **\$UL**. So e.g. if you enter **\$ULmyLabel** the variable label will contain the string “myLabel”. This can be used to identify different labels.

The variable *retStr* expects the string that will be displayed and has to be filled within the user-defined function. The maximum length of *retStr* must not exceed **MAX\_CTR\_TXT\_LEN** (a #define, generated into the file *rdnum.h*).

Note:

If a module is instantiated more than once, you cannot distinguish the different instances by the label, since it will be the same.

### 6.5.6.3 Custom Hash Function

Hash values are used to detect changes in strings. A string is marked as changed and redisplayed, if the hash value of its current value is different from the stored hash value. The radCASE default hash function is a relatively simple function which uses shifting and sums of the characters to calculate the hash. This reduces code size and operating time at the cost of the possibility of same hash codes for different strings, resulting in strings not being updated.

In case the radCASE default function does not suffice it is possible to write a custom function to calculate the hash. To activate a custom hash function the **DEFINE RD\_CUSTOM\_HASH** has to be set. If set, the following function has to be implemented:

```
long CESTr_hash(RD_char* str);
```

**str** 0 terminated input string

The function returns the calculated hash value.

### 6.5.7 Graphic Handling

On **SURFACE\_VIS** some graphical items are supported. These graphical items are also supported on **SURFACE\_CTR** in case of a graphical display on the embedded system (refer to setting **DISP\_GRA** in [Systemcode](#)).

The graphical support covers some simple graphical items like [RECTs](#), [CIRCs](#), [LINEs](#), [FILLs](#) and [ARCs](#).

Additional to this there is support for [PICTs](#), which are complexer graphical items, which can contain all of the simple graphical items and also bitmaps (refer to [Bitmap Usage](#)).

When working with graphical items also keep the [Color Details](#) in mind.

#### 6.5.7.1 Bitmap Usage

When using bitmaps in a project it is required to separate bitmaps used in **SURFACE\_CTR** on the target hardware ([CBITMAP](#)) and those used on other surfaces in the Project Monitor ([WBITMAP](#)).



For the target hardware radCASE supports the export into an internal pixelmap format and PNG. Different source image formats (like GIF, JPG, BMP) are transformed into the internal pixelmap format.

For the internal pixelmap format different color depths are supported:

<b>1 Bit per pixel</b>	Monochrome pictures
<b>2 Bits per pixel</b>	4 Color images. Only supported for BMP. This is created by automatic conversion from 4 Bit BMPs (refer to <a href="#">2-Bit Export Of 4-Bit Bitmaps</a> )
<b>4 Bits per pixel</b>	16 Color images. Only supported for BMP.
<b>8 Bits per pixel</b>	256 Color images.
<b>24 Bits per pixel</b>	RGB images. Source can also be a 32 Bit image, which will be converted to 24 Bits. The alpha channel will be ignored.

The different image formats and color depths have to be supported by the HAL. The Model compiler will not convert the color depth apart from the exceptions in the table above.

When working with Bitmaps with less than 16-Bit also keep the color table in mind (refer to [Color Palettes](#))

#### 6.5.7.1.1 2-Bit Export Of 4-Bit Bitmaps

If the color depth of a target is set to 2 Bit (refer to **Bits Per Pixel** in [EMB\\_HMI](#)) all 4-Bit BMPs referenced in a [CBITMAP](#) are converted to 2-Bit Bitmaps. The mapping between the 4-Bit Index table and the 2-Bit index table is made as shown in [Table 10](#).

Color value of 4-Bit BMP	Exported color value (2 bit)
0, 4, 8, 12	0
1, 5, 9, 13	1
2, 6, 10, 14	2
3, 7, 11, 15	3

Table 10, Mapping Of 4-Bit BMPs To 2-Bit BMPs

#### 6.5.7.1.2 Color Palettes

Because bitmaps with less than 16-Bit are working with a color table, a color table is required for radCASE, too. The color palettes used for all pictures in the simulation are the *16coltab.bmp* and *256coltab.bmp* in the *Develop* directory of the project. It is recommended to also use the *256coltab.bmp* for the target hardware using the external tool *ColTabConverter.exe* in the *rc\_lib\tools* directory.



The internal pixelmap format of radCASE does not include color table information. Because of this all bitmaps used should use the same color table and that color table should also be used in Simulation (*256coltab.bmp*) and on the target hardware.

### 6.5.7.2 Color Details

For bitmaps different color depths are supported (refer to [Bitmap Usage](#)). For all other graphical objects radCASE supports a special 8 Bit color mode. This contains 16 basic colors, 234 enhanced colors and some special color attributes:

- 16 basic colors** These colors should not be changed:  
BLACK, BLUE, GREEN, RED, CYAN, MAGENTA, BROWN, LGRAY,  
DGRAY, LBLUE, LGREEN, LRED, LCYAN, LMAGENTA, YELLOW, WHITE
- 234 enhanced colors** Index 16...249: interpretation according to HAL and simulation color palette (refer to [Color Palettes](#))
- 6 reserved color entries** 6 color entries reserved for special radCASE features.

The color entries reserved for special radCASE features are:

- NON** Used for transparency, e.g. to draw static text without changing the background.
- GLOBAL** The global foreground color is used (refer to [EMB HMI](#))
- GLOBALBK** The global background color is used (refer to [EMB HMI](#))
- BACKGRND** The background layer is restored. For graphical color displays a layer support can be supported by the HAL (refer to Integration manual). If this is supported radCASE will draw on two layers. The background layer will be used to draw all static items; the foreground layer will be used to draw all dynamic items (dependent on **ELEMENT** values). If using color BACKGRND the items of the foreground layer will be deleted in the specified area. This can be used for a transparency of dynamic items.

### 6.5.8 Element Visualization

The values of [ELEMENTs](#) can be visualized in different textual and graphical ways. The main ways for visualizing **ELEMENTs** are the RC-Items [ELEM/EDIT](#) and [MElem](#). In all of these RC-Items there are different visualizers which can be selected using the **Display type**.

**ELEM/EDIT** can be used to visualize the current value of only one **ELEMENT**. A description of all available visualizers for single **ELEMENTs** can be found at [Visualizers For Single Elements](#).

**MElem** can be used to visualize the current value of multiple **ELEMENTs** at the same time. Depending on the selected visualizer **MElem** can even be used to visualize the stored values of multiple **ELEMENTs** in a data recording (refer to [Data Recording](#)). A description of all available visualizers for multiple **ELEMENTs** can be found at [Visualizers For Multiple Elements](#).

### 6.5.8.1 Visualizers For Single Elements

The values of single [ELEMENT](#)s can be visualized with [ELEM](#)s/[EDIT](#)s. There are different visualizers that can be selected using the **Display type**. On the one hand, there are [Text Visualizers](#) which print out the current value of the **ELEMENT** as text and on the other hand there are different graphical visualizers which can be used for different **ELEMENT** types. Text visualizers can also be displayed as [Blinking Elements](#).

The following graphical visualizers are available:

- Bar Visualizers will display the value of **ENUMs** as a bar correlated to its range
- Radio Button Visualizers will display the selections of **EBINs** as radio buttons
- VISBINICON Visualizers can display each state of an **EBIN** as a picture. This visualizer can for example be used for realizing checkboxes.
- VIS\_LINE Visualizers display the value of **ENUMs** as a slide controller
- VIS\_ROT Visualizers display the value of **ENUMs** and **EBINs** as turning knob

### 6.5.8.1.1 Text Visualizers

There are several visualizers that display the ELEMENT value in text form. The general format for all those text visualizers is, using `VNTextXX` as **Display type** of the ELEM/EDIT. They all differ in character size and partial (only in special cases on **SURFACE\_VIS**) in coloration.



Even though the text visualizers have the same **Display Types** for both **SURFACE\_CTR** and **SURFACE\_VIS**, the look is different. Refer to [Formats Of Text Visualizers](#) for information on the look of the different supported text visualizers.

All of the Text visualizers support the following visualizer specific attributes:

<b>Size X</b>	Sets the text length to be displayed, expands or shortens the text accordingly. (only <b>SURFACE_CTR</b> )
<b>Position Value</b>	Moves the value and unit relative to the display position. This is mainly used for 2-line menus, because the position of the description may not be top/left of the display position. (only <b>SURFACE_CTR</b> )
<b>Horizontal alignment</b>	Aligns the text horizontally. Refer to <a href="#">Table 11</a> for further explanations.
<b>Rotation</b>	Rotates the whole displayed text. Valid values are 0, 90, 180 and 270. (only <b>SURFACE_VIS</b> )
<b>Show unit</b>	If set the unit of the <b>ELEMENT (ENUMs)</b> is displayed

**Horizontal alignment**

left

Reading 1      0123.45\_V/m

←→

>|<

Horizontal alignment    Display	
right	<div>Reading 10123.45_V/m&gt; &lt;</div>
center	<div>Reading 10123.45_V/m&gt; &lt;</div>
String left (not supported for proportional fonts)	<div>&gt; &lt;Reading 10123.45_V/m</div>
String right (not supported for proportional fonts)	<div>Reading 10123.45_V/m&gt; &lt;</div>
String center (not supported for proportional fonts)	<div>Reading 10123.45_V/m&gt; &lt;</div>

Table 11, Alignment Of A VNTexXX Visualizer

Explanations:

- >|<      Display position (as set by **Position X**)
- ↔      Distance between value and description defined by **Position description**.
- \_      Optional space between value and unit (if **No space between value and unit** is activated)
- 0      Optional leading zeroes (if **Show leading zeros** is activated)

6.5.8.1.2 Bar Visualizers

A Bar visualizer is a visualizer which displays the current value of an [ENUM-ELEMENT](#) as a bar in correlation to a range. To select a bar visualizer *VNBar1* has to be selected as **Display type** of the [ELEM](#). On **SURFACE\_CTR** the value is drawn in correlation to the **Range (low/high)**, on **SURFACE\_VIS** it's drawn in correlation to **Display (min/max)**. If the value is out of the **Alarm (min/max)**-range, on **SURFACE\_VIS** the bar is drawn with a white background and red foreground color, else the colors of the visualizer are used.

The bar visualizer supports the following visualizer specific attributes:

<b>Size</b>	The size of the bar
<b>Position Scaling</b>	Position of a scale bar if activated (only <b>SURFACE_VIS</b> ). For vertical bars only X-position and for horizontal bars only Y-position is taken into account.
<b>Text direction</b>	Specifies direction of the bar: H = horizontal V = vertical
<b>Show scale</b>	Used to activate displaying of a scale bar (only <b>SURFACE_VIS</b> )

If a scale bar is activated in **SURFACE\_VIS** the scale bar will show lines for the **Alarm (min/max)** values.

#### 6.5.8.1.3 Radio Button Visualizers

A radio button visualizer is a visualizer which displays all selections of an [EBIN-ELEMENT](#) (no multiselections) as radio buttons, displaying the current value as selected. To select a radio button visualizer *VNRadioBut01* has to be selected as **Display type** of the [ELEM](#). The radio button visualizer is only available on **SURFACE\_VIS**.

The radio button visualizer supports the following visualizer specific attributes:

<b>Size</b>	Specifies the space between each radio button.
<b>Position Value</b>	Specifies the position relative to each radio button where the description of the output should be displayed.
<b>Number of lines/columns</b>	Specifies how many lines or columns should be displayed. If both values are entered the number of lines will be ignored and all items are displayed in as many lines as it takes to display all selections with the specified number of columns. If no value is specified all selections are displayed in one line.

#### 6.5.8.1.4 VISBINICON Visualizers

A [VISBINICON](#) is a visualizer used to display the current status of an [EBIN](#) (no multiselections). For each possible status there is a picture which will be automatically selected according to the current [ELEMENT](#) status. To select a **VISBINICON** visualizer the **ID** of the **VISBINICON** has to be selected as **Display type** of the [ELEM](#).

The **VISBINICON** visualizer supports the following visualizer specific attributes:

<b>Position Value</b>	Activates displaying of selection texts and positions them next to the picture visualizing the current <b>ELEMENT</b> status.
-----------------------	---

#### 6.5.8.1.5 VIS\_LINE Visualizers

A [VIS\\_LINE](#) is a visualizer used to display the current status of an [ENUM](#). The visualizer positions the **LineImg** depending on the actual value of the **ENUM**. So this visualizer resembles a slide control. To select a **VIS\_LINE** visualizer the **ID** of the **VIS\_LINE** has to be selected as **Display type** of the [ELEM](#).

For support on the embedded system the HAL has to provide extended graphic support (refer to Integration manual).

The **VIS\_LINE** visualizer supports the following visualizer specific attributes:

- Size** Determines the range of movement of the slider.  
 A value of x, 0 means a horizontal movement of x pixels horizontally. A value of 0, y means a vertical movement of y pixels.  
 A negative or positive size determines the direction of the movement. For a positive x-Size the lowest value of the **ELEMENT** will result in the slider being at the left and moving to the right for raising values. A negative X-Size means the slider starts at the right and moves to the left for raising values.  
 A positive Y-Size will move the slider from bottom to top and a negative Y-Size will move the slider from top to bottom for raising values of the **ELEMENT**.

#### 6.5.8.1.6 VIS\_ROT Visualizers

A [VIS\\_ROT](#) is a visualizer that can be used for [ENUMs](#) and [EBINs](#) (no multiselections). In this visualizer the **RotImg** is rotated according to the current value of the **ELEMENT**. To select a **VIS\_ROT** visualizer the **ID** of the **VIS\_ROT** has to be selected as **Display type** of the [ELEM](#). For support on the embedded system the HAL has to provide extended graphic support (refer to Integration manual).

The **VIS\_ROT** visualizer supports the following visualizer specific attributes:

- Size** Determines the range of rotation of the **RotImg**.  
 The X-value in this attribute is the rotation the image has, when the **ELEMENT** has the lowest possible value. The Y-value in this attribute determines the rotation, when the **ELEMENT** has the highest possible value.  
 The rotation has to be provided in degree, where 0 is no rotation, a positive value is a counterclockwise rotation and a negative value is a clockwise rotation.

#### 6.5.8.1.7 Blinking Elements

Within a **SURFACE\_CTR** an [ELEMENT](#) which is displayed as Text visualizer (refer to [Text Visualizers](#)) can be displayed flashing. For this to work a blink frequency and a blinking type has to be selected (refer to [Ctr-Settings](#) **BF=<#>** and **BT=<#>**).

The **ELEMENT** which should be displayed flashing is selected by setting the global pointer **RD\_Blinkelem** to this **ELEMENT**:

```
RD_Blinkelem = (CElement*) ($*Elem);
```

To deactivate the flashing the global pointer can be set to *NULL*.

### 6.5.8.2 Visualizers For Multiple Elements

The values of multiple [ELEMENTs](#) at once can be visualized with [MElem](#)s. There are different visualizers that can be selected using the **Display type**.

The following visualizers are available:

- [Histogram Visualizers](#) will display the **ELEMENTs** of a [Data Recording](#) as a graph in correlation to the time
- [Multiple Bar Visualizers](#) will draw multiple bars for each two **ELEMENTs** according to their ranges.
- [VIS\\_ROT\\_XY Visualizers](#) will draw moving and rotating images according to the values of different **ELEMENTs**.

### 6.5.8.2.1 Histogram Visualizers

A histogram visualizer is a visualizer which displays all [ELEMENT](#) values of selected [ENUM](#)s and [EBIN](#)s contained in a [Data Recording](#). It simultaneously displays the values of multiple **ELEMENT**s in correlation to time using the specified colors. To select a histogram visualizer *VMNHisto1* has to be selected as **Display type** of the [MElem](#). The histogram visualizer is only available on **SURFACE\_VIS** and **SURFACE\_PRINT**.



The size of the visualizer specifies the drawing area. The visualizer will need additional space around it for displaying legends, scrollbars, etc. The Y-value of the size has to be positive, meaning the **Position X/Y** of the **MElem** is the bottom left corner of the drawing area.



Interruptions of the graph are drawn as little rectangles, so even short interruptions can be detected. The maximum time allowed between two record samples to be displayed as continuous data recordings is set with [Timing](#)-Setting **TG=<#>**.

Two **Options** affect the display of **EBIN**s:

1. **LG**: If this **Option** is activated by setting *LG=1*, it is possible to define a legend for **EBIN**s. Instead of displaying the Selection texts the internal value is displayed, which claims less space. The values then can be translated to the selection texts, by drawing a legend using visualizers available on the according surface.
2. **BO**: If an **EBIN** has a value of 0 (first selection) the according graph of that **ELEMENT** is drawn beneath the X-Axis, so it is not visible. By using an offset using this **Option** the first value of an **EBIN** can be drawn the specified amount of pixels above the X-Axis.

A histogram visualizer supports the following **Options**:

- **YS**: Y-Spacing
- **SV**: Size visualizer
- **CT**: Color table
- **BK**: Background color
- **NS**: Number scale lines
- **NG**: Number of grid lines
- **LG**: Legend
- **BO**: EBin-Offset
- **MD**: Mode Display

### 6.5.8.2.2 Multiple Bar Visualizers

The multiple bar visualizer is similar to the [Bar Visualizers](#) for single [ELEMENT](#)s. Different to these the multiple bar visualizer displays the bar not from the origin up to the current value of an **ELEMENT**, but from the current value from one **ELEMENT** to the current value of a second **ELEMENT**. To select a multiple bar visualizer *VMNBar1* has to be selected as Display type of the [MElem](#). For support on the embedded system the HAL has to provide extended graphic support (refer to Integration manual).

The visualizer can display multiple bars in one visualizer, which means the **Number of elements** used in the **MElem** has to be a multiple of 2. Every two **ELEMENT**s will display one bar and will use



one of the colors specified with the **Option CT**. The direction for the visualizer can be horizontal and vertical.

A multiple bar visualizer supports the following **Options**:

- **SV**: Size visualizer
- **CT**: Color table
- **BK**: Background color
- **DI**: Direction

#### 6.5.8.2.3 VIS\_ROT\_XY Visualizers

A [VIS\\_ROT\\_XY](#) is a visualizer that can be used for [ENUMs](#) and [EBINs](#) (no multiselections). In this visualizer the **RotImg** is rotated and moved according to the values of different [ELEMENTs](#). To select a **VIS\_ROT\_XY** visualizer the **ID** of the **VIS\_ROT\_XY** has to be selected as **Display type** of the [MElem](#). For support on the embedded system the HAL has to provide extended graphic support (refer to Integration manual).

For each **RotImg** three **ELEMENTs** have to be specified in the **List of elements** of the **MElem**. The first **ELEMENT** is for the movement in X-direction, the second for the movement in Y-direction and the third for the rotation.

The movement range specified by the **Option MR** correlates with the **Size** of [VIS\\_LINE Visualizers](#).

Each X,Y-entry in **MR** specifies the movement range for one of the **RotImgs** in X- and Y-direction.

The rotation range specified by the **Option RR** correlates with the **Size** of [VIS\\_ROT Visualizers](#).

Each Min,Max-Entry in **RR** specifies the rotation range for one of the **RotImgs**.

The entries for **MR** and **RR** for each **RotImg** are semicolon separated.

#### 6.5.9 Action Handling

There are different RC-Items that can be used in a surface, which will trigger different actions.

To trigger actions the following RC-Items can be used: [AMENU](#), [AICON](#), [AKEY](#), [AKEYGLOBAL](#), [AENTER](#), [APERM](#), [AEXIT](#) and [ActionCond](#). Each of these RC-Items can contain a list of multiple actions which are executed from top down. When navigating to another surface using the surface stack (refer to [Surface Navigation](#)) the list will be interrupted until returning from that surface. When navigating to another surface without using surface stack ([GOTOSURFACE](#)) all further actions in the list will never be executed.



**AENTER** and **AEXIT** are called in connection with the Surface Navigation. Please refer to [Surface Navigation](#) for detailed information on when they are called.

The different actions that can be triggered in a surface can be divided into the following groups:

- Actions which will manipulate **ELEMENTs**: [SET](#), [RES](#), [INV](#), [PUT](#), [INC](#), [DEC](#), [AEDIT](#) and [COPY](#)  
Refer to [Element Manipulating Actions](#)
- Actions used for surface navigation (refer to [Surface Navigation](#)): [GOTOSURFACE](#), [CALLSURFACE](#), [RETURN \(Sur\)](#), [OPEN](#) and [CLOSE](#)



- Actions for access control: [PASSWORD](#) and [PSWD\\_CLR](#)  
Refer to [Access Control Actions](#)
- An action for user queries: [ASKOK](#)  
Refer to [User Query Actions](#)
- Actions for calling functionality: [PROCEDURE](#) and [CFUNC](#)  
Refer to [Functionality Calling Actions](#)

#### 6.5.9.1 Element Manipulating Actions

For all actions (refer to [Action Handling](#)) which manipulate **ELEMENT**s on a **SURFACE\_CTR** it is possible to use the system variable **RDActElement**. This variable will resolve to the currently selected **ELEMENT** in a menu (refer to [Menus](#)).



If using **RDActElement** with [PUT](#) the internal data format is used (refer to [Virtual Floating Point](#))

For [INC](#) and [DEC](#) it is further possible on **SURFACE\_CTR** to override the default step width of 1. To enable this feature the [DEFINE RD\\_USE\\_MULTISTEP](#) has to be set. If the feature is activated, there will be a global variable **RDMultiStep**, which will be used instead of the default step width. Also for those two RC-Items when reaching the limit of an **ELEMENT**, for an **EBIN** there will be an overflow, going to the other end of the value limits and for an **ENUM** the decreasing/increasing will stop.



The standard edit dialog opened with **AEDIT** can be customized. Refer to [Customizing Standard System Dialogs](#).

#### 6.5.9.2 Access Control Actions

For Access Control in radCASE there are different kinds of password levels:

- Level 0** No password entered (default level)
- Level 1** User password entered
- Level 2** Admin password entered
- Level 3** Technician password entered
- Level 4** Super user password entered

The current level will be saved globally and when using [PASSWORD](#) radCASE will check, if the current password level is equal or greater than the required password level. If the current password level is not sufficient a password dialog will be opened to enter a password and the current password level will be set according to the entered password. After this radCASE checks again if the password level is sufficient. If the level is not sufficient the next actions in the list will not be executed. The number of jumped over actions is set in the **PASSWORD**.



If entering a password not sufficient for the next actions the password will still be set to the password level entered in the dialog, even if the previous password level was higher. E.g. if the required password level is 3 and the current password level is 2 and the password for level 1 is entered the current password level will switch to level 1.

If entering a wrong password the current password level will not be changed. Using [PSWD CLR](#) the current password level will be reset to 0, so every password protected action will again ask for a password.



The standard password dialog and message box for wrong passwords can be customized. Refer to [Customizing Standard System Dialogs](#).

### 6.5.9.3 User Query Actions

Using [ASKOK](#) the user can be queried to confirm to continue with the selected action. There are different predefined questions (refer to [ASKOK](#)) that can be selected. The standard question is “Save changes?” which will be selected for every invalid Type of question selected (e.g. 0 or an empty **Type of question**).

For the “Save changes?”-option the dialog will only appear if the global Flag **RD\_fModified** is set. This flag will be set automatically when the value of an **ELEMENT** is changed using standard edit dialogs.

If the user selects abort the next actions in the list will not be executed. The number of jumped over actions is set in the **ASKOK**.



The standard query dialogs can be customized. Refer to [Customizing Standard System Dialogs](#).

### 6.5.9.4 Functionality Calling Actions

There are different ways to call functionality as action in the surface:

- Using [PROCEDURE](#) as action a [PROC](#) can be called.
- Using [CFUNC](#) to call predefined system functionalities.

For using **PROCEDURES** the **PROC** has to be defined as **Processing type PUBLIC** and must have a **Return type** of *short*. With the return value of the function the surface interpreter can be triggered to do some actions:

Return value	Triggered action in surface interpreter
REDRAW_NON	No action
REDRAW_ALL	Redraws whole surface
EXITMODULE	Close current surface and restore old from surface stack
NEW_SURFACE	Initializes surface variables (like conditions) and redraws surface (should be used when entering a new surface)

### 6.5.10 Conditional Drawing

It is possible to create conditional surfaces, where different RC-Items are drawn depending on different conditions. This can be done using [IF, ENDIF, ELSE](#). For each **IF** there has to exist an **ENDIF**, the **ELSE** is optional. **IF, ENDIF, ELSE** can be nested within other **IF, ENDIF, ELSE**.

With **IF** an **ELEMENT** is checked against a condition. The condition consists of the condition operator and a constant value.

The following condition operators are available

- **== <const> or = <const>**: **ELEMENT** equals <const>
- **!= <const> or ! <const>**: **ELEMENT** equals not <const>
- **< <const>**: **ELEMENT** is lesser than <const>
- **> <const>**: **ELEMENT** is greater than <const>
- **<= <const>**: **ELEMENT** is equal or less than <const>
- **>= <const>**: **ELEMENT** is equal or greater than <const>
- **& <const>**: **ELEMENT** bitwise AND <const> (used for multiselective **EBINs** to check a bit)

The constant value can be used with §-Access (refer to [Access To Element Related Constants \(§\)](#))

For SURFACE\_VIS it is possible to check against special system variables:

- |                    |   |
|--------------------|---|
| <b>SIMMODE</b>     | With the system variable <b>SIMMODE</b> the condition checks if the <b>Project Monitor</b> is in Simulation or Visualization mode. To do this the <b>Variable</b> field is left empty and the <b>Condition</b> is set to <b>=SIMMODE</b> or <b>!SIMMODE</b> |
| <b>PC_PWDLEVEL</b> | The system variable <b>PC_PWDLEVEL</b> contains the current password level of the <b>Project Monitor</b> , and can be entered in the <b>Variable</b> field to check against a value.  |

## 6.6 Text Support

Texts are also part of the radCASE model.

radCASE supports multilingual and non-multilingual texts.

For the **multilingual texts**, there is a text table (refer to [Text Table](#)) for each project file where all texts are stored and managed using a single [Text ID](#). The texts can be referenced in different places while being defined only once in the text table, which also avoids redundancies.

**Non-multilingual texts** can be useful in cases where the text will be the same for every language (e.g. unit of an **ELEMENT** like voltage).

Multilingual texts are further distinguished into [Short Texts](#) and [Long Texts](#).

During model compilation, the texts exported into the **binary data**, called OSDLTX.T.

To save memory on the target, some short texts are not exported – see [Text optimization](#).

On the target, the binary data may be placed in **internal** (program) **memory** (which is the default) or some **external memory** (e.g. a data flash) – refer to the Integration Manual, chapter “Data on external devices” for more information.

Multilingual texts can be accessed in the C code (refer to [Text Access](#)).

### 6.6.1 Text ID

Each multilingual text is associated with a *unique* Text ID which is used to identify the text. The Text ID consists of a [Text ID name](#) and a [Text ID value](#).

#### 6.6.1.1 Text ID name

The beginning of the Text ID name determines how the multilingual text is treated and if [Text optimization](#) takes place:

Beginning of the Text ID name	Treated as ...	Text optimization
RTXT (default)	short text	Yes
XTXT	short text	No
LTXT	long text	No

**Table 12, Text ID name and Handling**

If a new text is created in the editor, the editor automatically creates a unique Text ID name beginning with RTXT. If a different handling is desired, the user has to edit the Text ID name manually (refer to the editor manual for details).

If new texts are added to the model during the text import, the Text ID name has to be created externally.

If the Text ID name is modified or created by the user, he has to make sure, that the name is unique throughout the model and keep in mind, that it will be used for creating a C #define – see [Text ID value](#).

#### 6.6.1.2 Text ID value

The Text ID value is the index of a text in the text table. During the model compilation, a C #define in the following form is generated into the file `CTRVAPPL\projectdefines.h`:

```
#define <Text ID name>                <Text ID value>
E.g.  #define RTXT1793528096_17555012082008  17
```



Since the Text ID value (a number) may change on the next model compilation, always use the Text ID *name* in your code.

### 6.6.2 Text Table

The text table consists of all the texts for the different languages. Each text is identified by its [Text ID](#) and contains translations of the text for the different languages. On how to manage the text table please refer to the according editor manual of the editor you are using.

To ensure all the texts will fit on the display of the target system radCASE offers two additional features. To ensure a maximum text length it is possible to let radCASE check if no translation exceeds that maximum text length (refer to [Text Length Checking](#)). For small displays very short texts have to be used, to be able to connect those texts with longer descriptions (e.g. used in the **Project Monitor**) radCASE offers the feature of Subtexts (refer to [Subtexts](#)).

#### 6.6.2.1 Text Length Checking

By defining a **Format** for a text entry the **Model Compiler** will automatically check the text length of all used translations.

The format can be specified as a maximum text length by simply inserting the number of allowed characters. When dealing with Unicode characters (refer to [Unicode](#)) the size of a character might change, so specifying the maximum number of allowed characters will not suffice.

In this case it is also possible to specify the maximum text length in pixels by appending the letter “p” at the end of the number. So by entering e.g. “30p” the **Model Compiler** will check no text will exceed a length of 30 pixels.

To calculate the pixel length of a text radCASE will use the standard font width (**Character Size X** in [EMB\\_HMI](#)) for ASCII characters and 16 pixels for Unicode characters.

#### 6.6.2.2 Subtexts

While using smaller displays it often becomes necessary to provide short texts in addition to the normal texts to use the normal texts in the **Project Monitor** and the short texts on the target display.

In this case it is important to have a connection between the normal text and the short text so it is clear to see (e.g. for a translator) that the short text is a shorter version of the normal text. For this radCASE offers the option of creating subtexts for normal text. For explanations on how to create a subtext please refer to the editor manual of the editor you are using.

Subtexts can be used in the **ELEMENT** visualizer (refer to [ELEM/EDIT](#)) by using the Setting **Sub-text selection**.



When using a subtext for an **ELEMENT** visualizer, the Ctr-Setting **NS=<#>** (refer to [Ctr](#)) has to be set accordingly.

#### 6.6.3 Short Texts

**Short texts** are the “normal” texts that can be used everywhere in the radCASE system.

In particular, they can be used everywhere in the radCASE model for easy display on surfaces.

The handling of short texts in the model is extensively automated and transparent to the user.

However, it is also possible to access a text in the C code – refer to [Text Access](#).

For short texts, radCASE allocates a number of buffers of the size [MAX\\_CTR\\_TEXT\\_LEN](#) in RAM.

### 6.6.3.1 MAX\_CTR\_TEXT\_LEN

MAX\_CTR\_TEXT\_LEN is the size of the the longest short text in the binary data. It is generated as #define in the file `CTR\APPL\rddnum.h` during the model compilation.

### 6.6.3.2 Text optimization

radCASE tries to generate size optimized code (refer to [Text ID name](#)):

Short texts with a Text ID name beginning with *RTXT* are only exported to the binary data ODSLTX, if they are used in the radCASE model. Otherwise, (if such a text would only referenced in the C-Code) radCASE can't detect this and the text will not be available.

If you want to prevent optimization of a short text and enforce the export to ODSLTX, modify the Text ID name to begin with *XTXT*.

Long texts will always be exported and be available for use in the C-Code.

### 6.6.4 Long Texts

Long texts are – as the name says – intended specifically for storing (very) long *multilingual* texts on the target in the binary data ODSLTX. In radEDIT, long texts are handled in the same way as short texts – except that the [Text ID name](#) has to begin with **LTX**. On the target, long texts can be accessed in C code – refer to [Text Access](#).

As opposed to short texts,

- Long texts are very limited in their usage – see [Limitations](#)
- radCASE does not allocate space in RAM for the long texts
- Long texts are not considered in determining [MAX\\_CTR\\_TEXT\\_LEN](#)
- All long texts of a model stored in ODSLTX, i.e. [Text optimization](#) is not applied
- The maximum length of a long text is 65535 RD\_char's

#### 6.6.4.1 Limitations

- Long texts
  - Are only available in the C code of the target via the functions explicitly allowed for long texts – refer to [Text Access](#).
  - Cannot be used in the Project Monitor
  - Cannot be used in the Documentation Generation
  - Must not be used anywhere in the model, e.g. they cannot be displayed directly on a any surface (CTR, VIS, WEB)
  - Must not be used with any function (other than those mentioned above) in the runtime library (e.g. string processing functions), since this may result in a buffer overflow
- radEDIT Build 23.360 of Apr, 11, 2017 or later has to be used

### 6.6.5 Language switching on the target and in the Project Monitor

Using Ctr-Settings **LC**=<\$+...+\$> and **LV**=<\$+...+\$> (refer to [Ctr](#)) the supported languages can be selected for the target hardware (LC) and for the **Project Monitor** (LV). To change the language on

the target hardware, the according language element has to be edited and then the language change will be initiated by **SYS\_COPY** or **SYS\_RESTORE** (refer to [Working Copy And Edit Copy](#)).

For the **Project Monitor** the language change can be made using the GUI (refer to Project Monitor manual).

### 6.6.6 Supported Languages

The following languages are supported:

Language index	Language	Language code (Code as per ISO639)
0	German	GR (DE)
1	English	UK (EN)
2	French	FR
3	Italian	IT
4	Dutch	NL
5	Spanish	SP (ES)
6	US English	US
7	Swedish	SV
8	Polish	PL
9	Portuguese	PO (PT)
10	Finnish	SU (FI)
11	Danish	DK (DA)
12	Belgian	BE
13	Norwegian	NO
14	Turkish	TU (TR)
15	Greek	GK (EL)
16	Japanese	JP (JA)
17	Chinese	CN (ZH)

Language index	Language	Language code (Code as per ISO639)
18	Russian	RU
19	Czech	CZ (CS)
20	Slovenian	SL
21	Croatian	HR
22	Serbian	SR
23	Korean	KO
24	Malay	MS
25	Indonesian	IN
26	Thai	TH
27	Japanese Katakana	JK
28	Latvian	LV
29	Lithuanian	LT
30	Estonian	ET
31	Hungarian	HU
32	Arabic	AR
33	Slovakian	SK
34	Bulgarian	BG
35	Hebrew	HE
36	Romanian	RO

Table 13, Supported languages

### 6.6.7 Unicode

radCASE supports the full Unicode character set for specified languages. Those languages are Chinese, Japanese, Korean, Malay, Indonesian and Thai. Furthermore Katakana supports a circumscribed character set of the Unicode. To use Unicode the Ctr-Setting **UC=<#>** has to be set (refer to [Ctr](#)).



The Unicode support differentiates between characters under 0x800h and over 0x800h. Characters below 0x800h are treated like in normal fonts and are displayed using the specified font for the text to display. Characters over 0x800h are displayed using a special Unicode font which has the dimensions 16x16. In strings which mix normal and Unicode characters, the normal characters are vertical bottom aligned with the Unicode characters and the width of the normal characters remains like specified. So for example a string consisting of 2 Unicode characters and 2 normal characters (using a font of 8x6) has a complete width of  $2 \times 16 + 2 \times 6$ . Positioning using “-s” for fonts with full Unicode character support is always using a height of 16.

Katakana is a special case: because Katakana is a Japanese character set which is designed for computer systems and can be displayed using smaller fonts, Katakana can be mapped from the area 0x30A0-0x30FF to the area 0x7A0-0x7FF using Ctr-Setting MK=1 (refer to [Ctr](#)). Because of this mapping Katakana is handled like a normal font, even though it is in the Unicode font area. In addition to the usual full-width display forms of characters, Katakana has a second half-width form. When originally devised, the half-width Katakana were represented by a single byte each, again in line with the capabilities of contemporary computer technology. The half width Katakana letters that can be found in the area 0xFF65-FF9F are mapped also to the area 0x7A0-0x7FF.

When using Unicode it is recommended to use font compress (refer to Integration manual) to reduce size for the target, by mapping the used Unicode characters into one coherent block.



To display Unicode characters in the **Project Monitor** Unicode support must be installed in your operating system. (Normally included in support for East Asian languages).

## 6.7 Sound reproduction

There is functionality that can be used to produce a beep on the controller and in the simulation (using the PC internal speaker).

**void longBeep(void)** This function plays a long beeping sound:  
On controller: 200ms with 1000Hz  
In Simulation: 200ms with 500Hz

**void shortBeep(void)** This function plays a short beeping sound:  
On controller: 100ms with 1000Hz  
In Simulation: 50ms with 1000Hz



There are some differences between controller and simulation for the sound reproduction. Apart from the differences in length and frequency of the functions above, the interface function `c_beep()` (refer to Integration manual) has a completely different behavior. On the controller it will play a sound until the sound is turned off using `c_beep(0)`, in the simulation it will just output a `shortBeep()`.

## 6.8 Code Size Optimization

### 6.8.1 Using Defines For Feature Activation/Deactivation

Apart from activating/deactivating features using the [SETTINGS](#), there are several [DEFINE](#)s to activate/deactivate features:

- [Defines For Feature Deactivation](#)
- [Defines For Feature Activation](#)

### 6.8.1.1 Defines For Feature Deactivation

The following [DEFINE](#)s deactivate features, to be able to reduce generated code size:

Define	Comment
<b>RD_NO_ACTION_EDIT</b>	Deactivates the functionality of <a href="#">AEDIT</a>
<b>RD_NO_ARABIC</b>	For Arabic fonts different characters have to be replaced for displaying a string. The replace functionality can be deactivated using this DEFINE when using Unicode (refer to <a href="#">Unicode</a> ) but not using Arabic characters.
<b>RD_NO_BEEP</b>	Disables functionality of <a href="#">Sound reproduction</a>
<b>RD_NO_BLINK</b>	Disables functionality of <a href="#">Blinking Elements</a> . Will automatically be set, if <b>RD_NO_COLOR</b> is set.
<b>RD_NO_COLOR</b>	Disables functionality for colors. Can only be used with text displays or monochrome displays if no inverted colors like in cursors are used. In addition to this the HAL has to support this Define. If functions are disabled <b>RD_NO_BLINK</b> will be set automatically.
<b>RD_NO_COMMUNICATION</b>	Communication between target and Project Monitor is deactivated. This Define also has to be supported by the HAL.
<b>RD_NO_CTR_REF</b>	Deactivates support of <a href="#">Placeholders</a>
<b>RD_NO_EDIT_COPY</b>	Deactivates the edit copy (refer to <a href="#">Working Copy And Edit Copy</a> )
<b>RD_NO_EVENT</b>	Deactivates functionality of <a href="#">ELEMENT</a> s with an <a href="#">Assign type</a> of <b>INEVT</b> and <b>OUTEVT</b>
<b>RD_NO_EXTVIS</b>	Deactivates most visualizers only <a href="#">Text Visualizers</a> and <a href="#">VISBIN-CON Visualizers</a> will still work
<b>RD_NO_FULL</b>	Disables the functionality of <a href="#">FULL</a> s
<b>RD_NO_HEX</b>	No support for <b>Hexadecimal values</b> in <a href="#">ELEM/EDIT</a>
<b>RD_NO_MULTIEBIN</b>	Deactivates multiple selections for <a href="#">EBIN</a> s
<b>RD_NO_MULTILINE</b>	Line breaks are not supported in <a href="#">Text Formatting</a>
<b>RD_NO_PASSFUNC</b>	Deactivates all functionality of <a href="#">Access Control Actions</a>
<b>RD_NO_RANGE</b>	Deactivates all functionality for handling the range of <a href="#">Numerical Elements</a> . Will also deactivate all <b>ELEMENT</b> visualizers which

Define	Comment
	display the value in connection with its range, like <a href="#">Bar Visualizers</a> , <a href="#">VIS LINE Visualizers</a> , <a href="#">VIS ROT Visualizers</a> , <a href="#">Multiple Bar Visualizers</a> and <a href="#">VIS ROT XY Visualizers</a>
RD_NO_SETUP	Will deactivate setting and displaying of standard values of <a href="#">ELEMENTS</a>
RD_NO_STDEDIT	Deactivates all standard edit dialogs; Only customized edit dialogs will be available (refer to <a href="#">Customizing Element Edit Dialogs</a> )
RD_NO_TEXTJUST	Disables horizontal alignment of <a href="#">TEXT</a> and <a href="#">Text Visualizers</a>
RD_NO_TIMEDATE	Disables all functionality for handling of <a href="#">Date Elements</a> and <a href="#">Time Elements</a> . Can only be used when <b>RTC_ON</b> is not active (refer to <a href="#">Systemcode</a> )

#### 6.8.1.2 Defines For Feature Activation

The following [DEFINE](#)s activate features, which should only be enabled when using them, because of the influence on the generated code size:

Define	Comment
RD_ERROR_CHECK	Enables runtime error messages (refer to <a href="#">Runtime Error Messages</a> )
RD_PROPFONT	Activates proportional fonts (refer to <a href="#">Font Handling</a> )

#### 6.8.2 Optimizing File Size Of OsdI.txt

Because of the pointer size (normally 2 Bytes) used to address texts in the *OsdI.txt*, there is a limit on how much text can be inserted into it. There are two settings which affect the maximum file size:

1. [Ctr](#)-Setting **BA=<#>** which affects the alignment within the *OsdI.txt*
2. [Ctr](#)-Setting **OPS=<#>** which affects the pointer size used within the *OsdI.txt*

Both settings can be used to enlarge the maximum file size at the cost of increasing the current file size.

Header		9	12	16	17
19	abc	abcd			
a	ab	abc			

Figure 4, Example of *OsdI.txt* with 1-Byte pointer and 1-Byte alignment

[Figure 4](#) shows a coarse structure of the *OsdI.txt*. After a short header a list of pointers for each text follows pointing to the text table after those pointers. In this example the alignment is 1 and the pointer size is also 1, limiting the maximum file size of the *OsdI.txt* to around 255 Bytes (at least the last text has to start at an offset of 255).

Header		5	7	9	10
11		abc		ab-	
-cd	a		ab	ab-	
-C					

Figure 5, Example showing the effect of alignment on file size of *OsdI.txt*

[Figure 5](#) shows the same *OsdI.txt* only with an alignment of 2. Because of the alignment all texts now start on an address which can be divided by 2. The pointer values are also divided by two and every time when accessing a text the pointer is multiplied with the alignment, which allows finding the correct text. Because now the pointers are divided by 2 this means the maximum file size is doubled to around 510 Bytes.

It should also be mentioned how the setting affects the current file size by adding gaps into the *OsdI.txt*. In the best case scenario all texts in the project have a length which can be divided by the alignment, which would mean that the file size would not change, in the worst case gaps are inserted behind every text.

Header		14	17
21	22	24	ab-
-c	abcd		a ab
abc			

Figure 6, Example showing the effect of pointer size on file size of *OsdI.txt*

[Figure 6](#) again shows the same *OsdI.txt* only this time with a pointer size of 2 and an alignment of 1. The pointers now take more space, but there is no space wasted between texts. The maximum file size is enlarged to 65535 Bytes. When increasing the pointer size the maximum file size increases exponential.

Again the effect on the file size should be mentioned. When increasing the pointer size by 1 Byte, for each text in the text table the current file size increases by 1 Byte.



The pointers are evaluated byte wise, this means the pointer size does not have to be a processor supported memory format and even odd values like 3 Bytes for the pointer size may be used. So the pointer only has to be as big as needed.

Both settings allow for bigger file sizes of the *OsdI.txt* and both settings result in a bigger current file size. Which of those two settings will result in the smallest file is very dependent on the used texts, so depending on the texts it can be better to increase the alignment or to increase the pointer size in other cases combinations of both may be the best way.

## 6.9 Scheduling

For correct working radCASE needs some kind of task support or at least a minimal kind of task switching. There are different tasks in a radCASE application (refer to [Tasks](#)). To use those different tasks or even create new ones there are different **Processing types** (refer to [Processing types](#)).

### 6.9.1 Tasks

radCASE expects at least the following three tasks:

<b>Surface-Interpreter Task</b>	A task that drives the radCASE surface interpreter. The task is mainly responsible for displaying and updating the Target HMI (refer to <a href="#">Target HMI</a> ) including the listening for keyboard events (refer to <a href="#">Keyboard Handling</a> ) and execution of actions (refer to <a href="#">Action Handling</a> ). Procedures called from an action will run also in this task (refer to <a href="#">Being Aware Of Task-Scheduling</a> ).
<b>Perm-Task</b>	The default execution loop. All functions of <b>Processing type PERM</b> (refer to <a href="#">Processing types</a> ) are executed in this task. All functionality which is not time critical should normally be executed here.
<b>Inter-Task</b>	On most targets the Inter-Task is driven by a timer interrupt. This task is highly time critical and is normally a real time task, which carries out only time critical functionality. Normally the radCASE timers (refer to <a href="#">Assign type TI</a> ) are increased and the hardware IOs are evaluated here. Because of this the timer accuracy is dependent of the Inter task rate (refer to <a href="#">Timing-Setting IR=&lt;#&gt;</a> ) if supported by HAL.

In addition to these tasks the user can define and use additional tasks, e.g. for user defined **Processing types**. Those tasks however must be supported by the HAL.

## 6.9.2 Processing types

The **Processing type** of a functionality defines when a functionality is called and in some cases in which task (refer to [Tasks](#)) it will run. The following standard radCASE **Processing types** are supported:

<b>LOCAL</b>	The function is only available in the current <b>MODUL</b> and will only be called, if the function is called using \$-Access (refer to <a href="#">Behavior Access</a> ). The function will run in the task of the calling functionality.
<b>PUBLIC</b>	The function is available also from another <b>MODUL</b> and will only be called, if the function is called using \$-Access (refer to <a href="#">Behavior Access</a> ). The function will run in the task of the calling functionality.
<b>PERM</b>	The function will be called periodically in the <b>PERM</b> -Task (refer to <a href="#">Tasks</a> )
<b>SIGNAL</b>	The function is called for every usage of the <b>MODUL</b> in a signal diagram (refer to <a href="#">Signal Diagrams</a> ). The function will run in the task of the calling signal diagram.
<b>INITpre</b>	The function is called once during system startup, before <b>INIT</b> .
<b>INIT</b>	The function is called once during system startup.
<b>INITpost</b>	The function is called once during system startup, after <b>INIT</b> .
<b>PARAMUPD</b>	Called whenever a parameter (refer to <a href="#">Assign types</a> <b>PAR</b> and <b>SYS</b> ) is changed. This function is intended to change Flags and perform other actions which are depending on the setting of a Parameter. It is the duty of the programmer to invoke <code>executePARAMUPD()</code> after a change of Parameters in the HMI.

In addition to these standard **Processing types** the user can define custom **Processing types**. For every defined processing type a function with the name *execute<Processing type name>* is created. Those functions can be called in one of the predefined tasks (e.g. in a special case within a **PERM**-function) or in a user defined task (which has to be supported by the HAL, which then has to call the according function).

## 6.10 Target Compiler Specific Features

### 6.10.1 Limiting Code Size Of Generated Files

Some compilers (e.g. Fujitsu compilers) have a limit on how big a single object file may get. If reaching that limit for the generated files of radCASE the generated files can be splitted in multiple files, with each having smaller object file size, so to be able to compile them.

#### 6.10.1.1 Limiting Size Of Module.c

If the object file size of the *module.c* is too big for the compiler, the *module.c* can be splitted with Ctr-Setting (refer to [Ctr](#)) **SMC=<#>**.

The **SMC**-Parameter sets the maximal number of lines for every part of the splitted *module.c*. The parts will be numbered so generated files are named *module1.c*, *module2.c*, etc. radCASE will

automatically delete old files, so there will be only the files which are generated new, which is important, if the code size is smaller after a change and there is one less file.

radCASE will always keep the generated code of a module together, so if the source code generated for one module is bigger, than the maximal number of lines specified by **SMC** the generated file will be larger than the specified value.



If a project will often change code size, to avoid having to modify the Makefiles each time the project size changes, it is possible to use empty placeholder files. For this just make placeholder files named after the part files *module2.c*, *module3.c*, etc. and insert as less content as possible to link those files to a project (e.g. a global char-Variable `char module2, char module3,...`). After this store those files in a subfolder and have your target creation copy those files to your source file directory for each part-file that does not exist. By this the Makefile can always insert the same amount of part-Files.

### 6.10.1.2 Limiting Size Of OsdL.ini

If the object size of the *osdl.ini* is too big for the compiler, the *osdl.ini* can be made splittable with Ctr-Setting (refer to [Ctr](#)) **SOI=<0/1>**

By activating this feature, the *osdl.ini* can be split with an external tool (e.g. *Offset.exe* in the *lrc\_lib\tools* directory) every 32768 Bytes. The first position to split the *osdl.ini* is at 0x8000, the second at 0x10000 and so on. This option makes sure, that there is a gap every 0x8000 bytes, so that no binary data stretches over two parts.

After splitting the *osdl.ini* the Macro **RD\_GET\_OSDL(type, index, var)** has to be adapted to use all parts. The macro should set the value of the variable *var*, which has the datatype *type*. The data can be found at the *index*. The index has to be multiplied with **BINARYALIGNMENT** to get the offset in the *osdl.ini*.

### 6.10.1.3 Limiting Size Of OsdL.txt

If the object size of the *osdl.txt* is too big for the compiler, the *osdl.txt* can be made splittable with Ctr-Setting (refer to [Ctr](#)) **SOT=<0/1>**

By activating this feature, the *osdl.txt* can be split with an external tool (e.g. *Offset.exe* in the *lrc\_lib\tools* directory) every 32768 Bytes. The first position to split the *osdl.txt* is at 0x8000, the second at 0x10000 and so on. This option makes sure, that there is a gap every 0x8000 bytes, so that no text will be stretched over two parts.

To use those splitted texts on the target there is an option to overwrite the standard methods `get_ptr2()` and `get_ptr3()` in *ctr\_util.c* with a self-defined function. For this you have to make a Define **GET\_TXT\_PTR** which contains the new functionality.

For example if you split the *osdl.txt* at 65Kb and you are using a Fujitsu-processor as target, the following Define could be added to *memory.def*:

```
#ifdef __CTR__
#define GET_TXT_PTR(offset, accesstype) (void RD_MBIN*)((offset > 65535) ? (osdltxt2 + offset - 65536) : (osdltxt1 + offset))
#endif
```

The *offset* is the offset within the *osdl.txt*. The *accesstype* is 0 if accessing header information and 1 if accessing a text within the *osdl.txt*.





There are two theoretically possible cases, where the mechanism will fail.

1. The header of the *osdl.txt* is bigger than 32Kb
2. One of the texts is bigger than 32Kb

In these cases there are points where the *osdl.txt* can't be splitted. To know if this is the case radCASE will log the first possible splitting position into *logfile.txt* and also if there are further places where the *osdl.txt* can't be split.

#### 6.10.1.4 Limiting Size Of Source.c

If the object file size of the *source.c* is too big for the compiler, the *source.c* can be splitted with Ctr-Setting (refer to [Ctr](#)) **SSB=<0/1>**.

By enabling this option the model compiler generates the files *source\_eldef.c*, *source\_helpvar.c*, *source\_moddef.c* and *source\_visdef.c* in addition to the *source.c*. Each of these new files contains a part of the source code of the *source.c* and they can be used instead of the *source.c*, to get smaller object files.

To use this option the Makefile has to be adjusted.

#### 6.10.2 Harvard Architecture

The normal memcpy command cannot be used for most Harvard architectures like the C167. Instead of memcpy radCASE provides the both function *RD\_MemcpyRomRam()* and *RD\_MemcpyRamRam()* which can be called depending on the memory area from where to copy the data into the RAM.

To use these functions the correct handling of *RD\_MRAM* and *RD\_MROM* is mandatory.

### 6.11 Project Monitor

The **Project Monitor** works in two different modes. In visualization mode the **Project Monitor** communicates with a connected embedded controller and shows the values of the [ELEMENT](#)s. In simulation mode the embedded controller is simulated and the **Project Monitor** communicates with that simulated controller and visualizes it.

The **Project Monitor** can be converted to a [Standalone](#) version which can run without any pre-installed radCASE and can be given to customers. As **Standalone** the **Project Monitor** can also be used in combination with different software versions of the embedded controller (refer to [Dynamic Version Switching](#)).

For simulation mode there is some additional functionality to simulate some special behavior of targets (refer to [Simulated Display Functions](#)) and simulate the environment the controller runs in (refer to [Stimulation Equations](#)). There is also some support for [Sequences](#) which can be recorded and replayed after modifications of the system. Additionally in simulation mode an [Output window](#) is supported.

For more control over the simulation there is support for [Customizing The Simulation Project](#).

There are also different start modes for the simulation mode (refer to Project Monitor manual).

Those start modes can be selected by default by setting the [DEFINE](#) SIM\_AUTO\_START. The following values are supported:



SIM_AUTO_START value	Start mode
0	Continue
1	Restart
2	Reset

For the visualization mode (and also simulation mode) there are some [Communication](#) features to get information from the target. To be able to remote control the target, there is also some [Virtual Keyboard Support](#). It is also possible to react to special system events (refer to [System Event Handling](#)) and analyze the consumption of different resources of a project (refer to [Resource Consumption Analysis](#)).

On more details on how to operate the Project Monitor please refer to the Project Monitor manual.

### 6.11.1 Standalone

The **Standalone** is a special version of the [Project Monitor](#), which runs without the need to install radCASE beforehand. It is a tool which can be used for rapid prototyping, because in combination with the simulation mode prototypes of the software can be developed very quick and the **Standalone** can then be used to deliver first prototypes to the customer.

On details on how to create the **Standalone** refer to the Editor manual of the Editor in use. If there are some project specific files, which also needs to be copied to the **Standalone** (e.g. Configuration files), the file *standalone\_prj.bat* can be created in the *DEVELOP*-directory of the project, which will be called during **Standalone** creation.

### 6.11.2 Stimulation Equations

**Stimulation Equations** can be used to simulate dynamic behavior like the system environment (e.g. a temperature rising if a heating element is turned on). This can be used to simulate complex processes dependent on different hardware inputs, completely in the development environment without the actual hardware.

**Stimulation Equations** are simple mathematical expressions combining values with different operators.

The following values are supported:

<b>Element names</b>	By providing <b>ELEMENT</b> names (refer to <a href="#">Element Access</a> ) the value of an <b>ELEMENT</b> can be used in the equation
<b>Numerical constants</b>	Numerical constants like 5 or 7.3
<b>Predefined constants</b>	The predefined constants <b>pi</b> and <b>e</b>

**History values**      The previous value of an **ELEMENT** can be accessed with its Element name (refer above) and [1]. E.g. *Element[1]*  
 The previous value of that can be accessed with the Element name and [2]. E.g. *Element[2]* to the second to last value.  
 There are no other history values available. You can't use [0] or [3] etc.

The following operators are supported:

**Arithmetic infix operators**       $\langle a \rangle + \langle b \rangle$ : Adding  $\langle a \rangle$  to  $\langle b \rangle$   
 $\langle a \rangle - \langle b \rangle$ : Subtracting  $\langle b \rangle$  from  $\langle a \rangle$   
 $\langle a \rangle * \langle b \rangle$ : Multiplying  $\langle a \rangle$  and  $\langle b \rangle$   
 $\langle a \rangle / \langle b \rangle$ : Dividing  $\langle a \rangle$  by  $\langle b \rangle$   
 $\langle a \rangle ^ \langle b \rangle$ :  $\langle a \rangle$  to the power of  $\langle b \rangle$   
 $\langle a \rangle \% \langle b \rangle$ :  $\langle a \rangle$  modulo  $\langle b \rangle$

**Comparison operators**       $\langle a \rangle < \langle b \rangle$ : 1 if  $\langle a \rangle$  is lesser than  $\langle b \rangle$   
 $\langle a \rangle > \langle b \rangle$ : 1 if  $\langle a \rangle$  is greater than  $\langle b \rangle$   
 $\langle a \rangle = \langle b \rangle$ : 1 if  $\langle a \rangle$  is equal to  $\langle b \rangle$

**Logical operators**       $\langle a \rangle \& \langle b \rangle$ : 1 if  $\langle a \rangle$  AND  $\langle b \rangle$  are not 0  
 $\langle a \rangle | \langle b \rangle$ : 1 if  $\langle a \rangle$  OR  $\langle b \rangle$  are not 0  
 $! \langle a \rangle$ : 1 if  $\langle a \rangle$  is 0

**Infix functions**       $\langle a \rangle \min \langle b \rangle$ : Minimum of  $\langle a \rangle$  and  $\langle b \rangle$   
 $\langle a \rangle \max \langle b \rangle$ : Maximum of  $\langle a \rangle$  and  $\langle b \rangle$

**Prefix functions**       $\text{sqr}(\langle a \rangle)$ : Square root of  $\langle a \rangle$   
 $\text{sin}(\langle a \rangle)$ : Sinus of  $\langle a \rangle$  (Radian measure)  
 $\text{cos}(\langle a \rangle)$ : Cosinus of  $\langle a \rangle$  (Radian measure)  
 $\text{tan}(\langle a \rangle)$ : Tangent of  $\langle a \rangle$  (Radian measure)  
 $\text{asin}(\langle a \rangle)$ : Inverse sinus of  $\langle a \rangle$  (Radian measure)  
 $\text{acos}(\langle a \rangle)$ : Inverse cosinus of  $\langle a \rangle$  (Radian measure)  
 $\text{atan}(\langle a \rangle)$ : Inverse tangent of  $\langle a \rangle$  (Radian measure)  
 $\text{exp}(\langle a \rangle)$ : e to the power of  $\langle a \rangle$   
 $\text{log}(\langle a \rangle)$ : logarithm of  $\langle a \rangle$

**Stimulation Equations** are evaluated from left to right (so no order of operation) only parenthesis can be used to change that order. E.g.  $3 + (5 * 2)$  to get the mathematical correct behavior.

The **Stimulation Equations** transforms all values to a double internally (considering the [Virtual Floating Point](#) format) and uses the double accuracy for internal calculations. The result will be converted into the target ELEMENT format (also considering the [Virtual Floating Point](#) format)



The syntax of **Stimulation Equations** is different from C syntax. Every operator is only one character in size, so no == but = is used instead and no && but & is used instead. Also there is no support for combinations of comparison operators like >=. Boolean results will not be TRUE but 1, so you can e.g. add 2 to a Boolean value to get 3.

### 6.11.3 Virtual Keyboard Support

For remote controlling an embedded controller or to operate the simulation there is some virtual keyboard support within the [Project Monitor](#). To define a virtual key, an **ELEMENT** with **Assign type KEY** has to be created and visualized on a **SURFACE\_VIS**. The **Assign string** of the key has to contain the key code (refer to [Key Values](#)) or the numerical value of the key. When clicking on that

**ELEMENT** the key is sent to the controller and is processed, as if the key was pressed on the controller itself.

It is also possible to use the PC keyboard to send keys to the controller.

#### 6.11.4 Communication

The communication between [Project Monitor](#) and embedded controller can use different communication interfaces. The communication can be password protected (refer to [Password protection](#))

To use different communication interfaces a DLL implementing the communication over the according interface is required. The DLL can be selected in different ways. Refer to Monitoring manual for more information on how to select the DLL and which DLL is used at startup. The [Timing](#)-Setting **CD=<\$>** is used as one of the ways to select the DLL.

By standard radCASE is delivered with the following two DLLs:

1. cclConnection.dll: This is the standard communication interface and uses a serial communication using the RS232 interface.
2. cclConnectionEth.dll: This DLL implements the communication interface over an Ethernet interface. The [Systemcode](#)-Setting **ETHERNET\_ON** has to be set for this to work correctly.

The communication contents between **Project Monitor** and embedded controller are mostly defined by the [Com Type V<#>](#) of the [ELEMENT](#)s. Every **ELEMENT** with **Com Type** V1-V9 will be communicated to the **Project Monitor**.



Many **ELEMENT**s with activated communication result in a high load on the according communication interface. If the bandwidth of the communication interface is reaching its capacity, the updating of **ELEMENT** values in the Project Monitor will noticeably slow down.

Because the bandwidth of the communication interface can be reached very fast for modem or GSM modem connections, it can be required to be able to have a fast communication of few **ELEMENT**s, but also be able to get the status of all **ELEMENT**s. This can be done by assigning different **Com Types** (e.g. V9) for **ELEMENT**s which need to be communicated very fast.

If using V1-V9 it is possible to define a reduced communication (refer to Project Monitor manual), which only transfers specific data. The user can then switch between a slow full communication and a fast reduced communication.

Instead of a reduced communication it is also possible to use the [Burst](#) mode to make a fast communication of measure data.

The communication is done in a special [Communication Sequence](#) where different timings should be considered. It is possible to record communicated data into a [Data Recording](#).

To set some hardware outputs overriding the values set from the application the [Force Mode](#) can be used.

#### 6.11.4.1 Password protection

It is possible to use password protection for the communication with the controller. When password protection is enabled, the communication can only be initiated when providing the correct password. The pointer *RD\_char\* pPassVisu* must point to the password used. If the pointer is *NULL* no connection to the controller is possible.

By default the pointer is connected to the [ELEMENT](#) PassVisu in the [MODUL](#) MRoot in *std\_system.rad* (refer to [The System MODUL](#)).

The password protection is disabled by setting the password to "00000000". This is the default value of the **ELEMENT**. For any other value the project monitor will prompt for a password during connection establishment (refer to Project Monitor manual).

#### 6.11.4.2 Communication Sequence

The [Communication](#) with the target runs in a request/response scheme. Every communication is initiated by the project monitor (with the exception of the [Burst-Mode](#)). This means the project monitor will send a data packet or a request to the target and will receive an answer if the target is required to send one. This will be referred to as a communication block in the further course. After every communication block the project monitor will delay the next communication block for the Inter Block Delay (**IBD=<#>** refer to [Timing](#)) specified in the project. The communication is divided into two separate phases:

1. An initialization phase where the connection is established and some basic information is exchanged.
2. The data exchange phase which runs a periodic communication cycle.

The periodic communication cycle itself is divided into subphases, each transferring one communication block:

- a. Receiving an [ELEMENT](#) data block. This is either receiving data from IOs, FLAGS and PROCs or on demand (either automatically or triggered by user) data from PARs and SYSs. If the data is too big for one communication block, the data will be sent split into multiple communication blocks. So to receive element data it may be necessary to wait multiple communication cycles. The communication of this data can be slowed down in favour of other data by specifying the **Timing-Setting DCR=<#>**. This setting will cause the communication to only receive a communication block every n-th communication cycle.
- b. Receiving a debug message data block or all debug message data blocks in a loop if **Timing-Setting DMR=<#>** is set to 0. If all data blocks are received the Inter Block delay is inserted between each block. The debug messages are only received when the feature is enabled and there are debug messages to receive. The received debug messages are put into the Output window (refer to [Output window](#) for more information). The communication of this data can be slowed down in favour of other data by specifying the **Timing-Setting DMR=<#>**. This setting will cause the communication to only receive a communication block every n-th communication cycle.
- c. Receiving an HMI data block (only if HMI communication isn't disabled). The display data is also split into multiple communication blocks according to display data size and communication block size. So to receive the display data, multiple communication blocks may be needed, also. The communication of this data can be slowed down in favour of other data by specifying the **Timing-Setting HCR=<#>**. This setting will cause the communication to only receive a communication block every n-th communication cycle.

- d. Sending an asynchronous communication event if an event is buffered. These events are communication events triggered by interaction of the user with the project monitor. This includes changing **ELEMENT**-values, sending data blocks, sending keyboard events and executing functionality on the target. Some requests like transferring data blocks may be split up into multiple communication blocks. The communication of this data can be slowed down in favour of other data by specifying the **Timing-Setting** **ACR=<#>**. This setting will cause the communication to only send a communication block every n-th communication cycle.
- e. Sending touch events. This isn't an asynchronous communication event for technical reasons due to the nature of how touch data is handled. Pressed coordinates are buffered and sent as data in this block. If there are no coordinates buffered and the user is pressing the touch, the current coordinate is sent as data. As long as data is not sent those interim values are discarded. This enables usage of hover effects on the target while keeping the needed bandwidth low as possible. The communication of this data can be slowed down in favour of other data by specifying the **Timing-Setting** **TCR=<#>**. This setting will cause the communication to only send a communication block every n-th communication cycle.



There are some communication modes which will pause the communication cycle until finished. These modes are the **Burst** mode, File transfer, Target protocol transfer and some plugin functionality.

The communication of display data from the target to the project monitor is as follows:

The display data is fetched from the connected controller and saved in a special receive buffer for display data. Depending on the data size and display communication type, there can be multiple blocks of data which are fetched individually for the complete data. The update rate of the whole display data is affected by the settings discussed above.

The communication with the simulation is more direct. There is no initialization phase required and asynchronous and touch events are sent directly into the simulation code. The HMI data is drawn from simulation code directly into the visualization display. **ELEMENT** data is fetched in a whole block (not in data chunks like from the target). The rate in which **ELEMENT** data is fetched is determined by the **Timing-Setting** **DR=<#>**.

The communication of **ELEMENTS** runs in multiple steps through the whole project monitor and there are different interim buffers where the values are stored:

1. The **ELEMENT** data is fetched from the connected controller or simulation and saved in a Receive-Buffer. Depending on the data size, there can be multiple communication blocks of data which are fetched individually for the complete data (only target). The rate at which the whole data is fetched is influenced by the settings discussed above.
2. As soon as a complete set of data is received the data is copied from the Receive-Buffer into a data recording buffer. The Storage Rate (**SR=<#>**) determines how often the values from the data recording buffer are saved into the database of the [Data Recording](#).
3. The data is also copied from the Receive-Buffer to the visualization **ELEMENTS**. The data is shown immediately in the **Surface\_VIS**.

Special cases in this scenario are **LOCALS**, **EVA**s and **IO**s with [Stimulation Equations](#) (only in simulation mode). The Stimulation rate (**ST=<#>**) determines how often the **EVA**s and **Stimulation**

**equations** are calculated. After this the values are copied into the data recording buffer and the visualization **ELEMENTs** and are directly displayed.

**ELEMENTs** of type **LOCAL** are updated each time the visualization **ELEMENTs** are updated. This means every time the stimulation rate triggers the update of **EVA**s and **Stimulation equations** or when **ELEMENT** data is received from the target or simulated target, the **LOCALs** are updated in the Surface\_VIS.



Most of the settings discussed above can be set in the project monitor at runtime. Those settings will be saved after modification and will override the settings made in the design. When changing the standard values in the design, the saved values will be reset to the design default.

#### 6.11.4.3 Data Recording

Within the [Project Monitor](#) the recording of data is available. All communicated [ELEMENTs](#) are stored in a file based database over the time and can also be displayed in a histogram (refer to [Histogram Visualizers](#)). Each dataset can be opened and the values of the **ELEMENTs** can be analyzed.

The recording can be created from a normal communication, imported from a Protocol (refer to [Protocol Storage](#)) or opened from a previously saved file.

The recording can be saved into a file, exported into an ASCII format (refer to [ASCII Export](#)) or into a sequence (refer to [Sequences](#)).

##### 6.11.4.3.1 ASCII Export

[ELEMENTs](#) can be marked for an ASCII-Export by [Com Type](#) **X<#>**. After a [Data Recording](#) all values in the recording of all **ELEMENTs** with this **Com Type** can be exported into a CSV-File. This can be useful to analyze data with external tools like Microsoft Excel.

#### 6.11.4.4 Force Mode

Normally all hardware outputs ([ELEMENTs](#) of [Assign type](#) **AO**, **DO**, **AI**, **DI** and **CNT**) are driven by the process defined in the application or by sensors connected to hardware and can't be manipulated using the [Project Monitor](#). To manipulate an output or input from the **Project Monitor** the **Force Mode** has to be activated (refer to Project Monitor manual). When the **Force Mode** is activated the outputs/inputs can be forced to use another value, even if the process or sensor sets them to a different value.

#### 6.11.4.5 Burst

In the normal [Communication](#) every data packet is requested by the [Project Monitor](#) and every packet is confirmed by sending a checksum. The burst mode is a very fast type of communication, where only the activation of the burst mode is confirmed and after this the embedded controller only sends the data to the **Project Monitor** without checking if the **Project Monitor** received the data.

[ELEMENTs](#) which should be sent when in burst mode must have the [Com Type](#) **B1**. The burst mode has to be supported by the HAL (refer to Integration manual) and to activate the [DEFINE USE\\_BURST](#) has to be set. In addition the HAL has to compile and link the file *distrib.c* for the burst mode to work. There are two ways to start and stop the burst mode. The recommended way is to let



the application decide when to start or stop bursting (refer to Integration manual for further information). The other way is to use the Burst Button in the **Project Monitor** to start/stop the burst mode (refer to Project Monitor manual). For this the Burst button has to be enabled (refer to [Desk-top-Setting BB=<0/1>](#)).

The burst mode is only meant for short and fast bursts of data and should not be used for long data recordings.

#### 6.11.4.6 Display communication

radCASE offers to display HMI contents of a remote target within the visualization. If the HAL supports it, in most cases adding a [DISPLAY](#) to a **SURFACE\_VIS** is sufficient for this feature to work. By adding the [DEFINE](#) RD\_NO\_DISPLAYCOMM to the project the display communication can be deactivated to save memory.

There are three different modes of display communication:

1. Text based: This communication type is only supported for text displays and is also the default mode for this type of display. The display content is communicated as texts displayed on the target.
2. Pixel based: This communication type is only supported for monochrome graphic displays and is also the default mode for this type of display. The display content is communicated on a pixel basis. Every pixel of the display is represented by one bit within the data buffer used in communication.
3. Command based: This communication type is supported for all displays, but is only enabled by default for color displays. The display content is communicated as a list of drawing commands (e.g. draw a line) with their attributes. This mode can save communication buffer size at the cost of not knowing exactly, how big the buffer size has to be. The needed size largely depends on the connection speed and the number of rc\_Items used in a **SURFACE\_CTR**. There are a few Defines connected with this mode:
  - RD\_DISPCMD\_COMMUNICATION: This define enables/disables this communication type. By setting it to 1 the mode is enabled; by setting it to 0 the mode is disabled. If the define is not set, the default value depends on whether it is a color display.
  - RD\_DISPCMD\_BUFSIZE: This define influences the size of the buffer used to store the display commands in bytes. The default value is 1024 Bytes. This define should be adapted to the actual project needs.
  - RD\_DISPCMD\_MAXRETRY: After a buffer overflow of the communication buffer, the protocol receiver tries to recover, by emptying the buffer and triggering a redraw. After a connection drop the buffer could have an overflow, because of a changing element being updated multiple times before the reconnect. This type of error can be resolved by the redraw. The value of the Define influences how often the protocol receiver will try to recover until reporting an error to the visualization. The default value is 3.



When using command based communication the HAL functions for drawing directly to the screen should not be used. Instead of these functions wrapper functions should be used (refer to [HAL drawing wrapper functions](#)). The wrapper functions will call the HAL functions, but also add the commands to the display command buffer used in communication. If not using these wrapper functions the display contents will not be communicated correctly to the visualization.

### 6.11.5 Permission modell

radCASE allows to password protect different features of the **project monitor**. There are different password levels for protection available. Refer to [Password management](#) for information on how to manage the standard passwords for those password levels. Each feature of the **project monitor** can have a specific password level needed to use. Refer to [Feature permissions](#) on how to set the needed permissions for a feature of the project monitor.

#### 6.11.5.1 Password management

Passwords can be changed within the **project monitor**. Refer to Monitoring Manual for further information on how to set passwords in the **project monitor**. The passwords are stored in the file *passwordlist.dat* in the *Develop*-directory of the project. This file is encrypted with the **encryption password** specified in [Passwords](#) in the project. If the file does not exist or can't be opened (e.g. because of a wrong password) the standard passwords specified in **Passwords** are used. If no passwords are specified in the project, the passwords radon1-radon9 are used for the according password levels.

By specifying an **encryption password**, only files encrypted with the correct password can be opened. This means the password list can't be easily replaced with another file with different passwords without knowing the **encryption password**. By using the same **encryption password** for different projects or different project versions, an end customer can simply create a standard file and copy it to different Standalone versions (refer to [Standalone](#)), to ensure the same passwords are used. To reset a project to standard passwords the file can simply be removed.

#### 6.11.5.2 Feature permissions

Each feature in the **project monitor** has a UID. For each UID available in the ribbon bar configuration a specific permission can be set. The ribbon bar configuration also contains the standard values for the permissions. For more details on the ribbon bar configuration and the available UIDs refer to the Monitoring Manual. For details on how to edit the permissions refer to the editor manual of the editor you are using.

Each permission consists of the UID of the feature, the needed password level and the disable mode. Most likely the editor will also show the name of the feature as specified in the configuration. The needed password level is the minimum password level (refer to [Password management](#)) needed to use the feature. Below this password level the feature is disabled as specified in the disable mode. A feature can either be disabled by greying out the item, or by hiding the item and not showing it in the ribbon bar.

If no permissions are specified in the project the ribbon bar configuration file is used as a standard. If a password level is specified in the configuration the item is by standard disabled by hiding.

### 6.11.6 Output window

It is possible to output text to the output window. There are currently two ways to do so:

1. Using the debug message functionality. This feature has to be enabled by setting the [DEFINE](#) `RD_DEBUG_MESSAGES` to 1. Also the feature has to be supported by the HAL (refer to Integration manual for more information on HAL-support). A debug message will be communicated



to the project monitor using the normal communication cycle (refer to [Communication Sequence](#) for more information). The message can be passed to the project monitor by calling the function:

```
void RdOutputDebugStr(const char* pMessage);
```



The function is not thread safe. When calling the function from different threads, the calls have to be secured by surrounding them with a mutex or similar thread synchronization objects.

When using the function it is possible to automatically add timestamps to all messages, by setting the function pointer pCreateTimestamp to a function of the prototype `const char* (*)(void)`; e.g.

```
const char* createTimestamp(void);
```

The function has to return a pointer to the created timestamp, by using a global or static variable.

2. By using the following function:

```
RDLogMessage(std::wstring windowname, std::wstring text);
```

This function is only supported in simulation mode. The specified windowname specifies the tab the text should be put into. If the tab does not exist it is created in the output window.

Log messages received will automatically be logged in a file called `Output_<tabname>.txt` in the `Develop`-directory of the project.



On receiving a message a timestamp will be added, too. This timestamp will be the time of receiving the message. When high accuracy is needed, the automatically added timestamp described above is recommended.

### 6.11.7 Sequences

Sequences contain information on different [ELEMENT](#) values at different times. Sequences can be inserted into the simulation and be stepped or run through. The simulation will run normally, but the **ELEMENT** values are set, at the specified times of the sequence. So a sequence can be used, to test if specified inputs at specified times result in a correct behavior of the process or to check if a changed process still delivers correct results.

A sequence can also be created from a [Data Recording](#). Normally the sequence will contain all **ELEMENT**s of the project and every change of the value. By specifying a [Com Type](#) `S<#>` **ELEMENT**s can be sorted into different groups. These groups can be selected for a Sequence export, to select only **ELEMENT**s, which are inputs to the process. When using a Data Recording to create a sequence, the data should be recorded in a synchronized mode, so the simulation tasks will run synchronized and every step in the simulation will be recorded.

### 6.11.8 Dynamic Version Switching

It is possible to use the **Project Monitor** for visualization or simulation of different versions of controller software, e.g. different similar controller software or different software versions of the same controller software.

This can be achieved by using the **Standalone Selection Tool** (radSEL). This tool will provide a list of available software versions to start. It can detect the controller software version of a connected controller and start the according **Project Monitor** automatically. The following chapters will explain how to configure the **Standalone Selection Tool** (refer to [Configuring radSEL](#)) and how to use it (refer to [Using radSEL](#)).

#### 6.11.8.1 Configuring radSEL

For **radSEL** to work it requires a unique identification of every version of the controller software it should support. The identification consists of the combination of software version (refer to [VERSION](#)) and the **DEVICE\_ID** (refer to [Managing Non-Volatile Memory](#)).

For every software version of the target software a standalone version of the **Project Monitor** has to be created (refer to [Standalone](#)). Because different **radCASE** versions use different communication protocol versions a standalone can only support controller software created with the same **radCASE** version. Standalones created with the same **radCASE** version can be combined into one standalone with a directory containing the version dependent data (refer to [Version dependent data](#)). Different **radCASE** versions need to use standalones in differently directories (refer to [Directory structure](#)).



**radSEL** detects the correct standalone version to use by the version dependent data. So even if there is only one version of controller software created with a specific **radCASE** version, there needs to be at least one file of version dependent data. However it is advised to use all files normally used as version dependent files to easily be able to add further standalones at a later time.

To configure different settings of **radSEL** a configuration file has to be created. For information on those settings and what is affected by those settings refer to [Configuration file](#).

For older **radCASE** versions the communication DLLs use an ini-File for storing the communication settings. To pass the settings from **radSEL** to the according standalone, **radSEL** has to create those files. To let **radSEL** know how to create those files for older **radCASE** versions, a template for creating these configuration files has to be provided (refer to [Communication configuration templates](#)).

By default **radSEL** only supports german and english language, however the application can easily be translated to other languages by adding translation files (refer to [Translations](#)).

The default icon of **radSEL** can be changed by putting another icon named *radsel.ico* into the same directory as *radSEL.exe*.

When configured, the directory containing **radSEL** can be distributed to the end customers (refer to [Distribution](#)).

##### 6.11.8.1.1 Directory structure

Starting point for the needed directory structure is the *radSEL* directory in the *Common* directory. The files in that directory are everything needed for **radSEL** and can safely be separated from the rest of the *Common* directory.

For every **radCASE** version a standalone directory is required. **radSEL** will search recursively for Standalones with the required version dependent data (refer to [Version dependent data](#)) in its own directory. So the different standalone directories can be added directly in the directory containing the *radSEL.exe* or can be put in further subdirectories. However putting standalone directories into other standalone directories is not allowed.

The version dependent data has to be in a directory called „*versionfiles*“ and this directory has to be directly in the standalone directory containing the *radMON.exe* or for older **radCASE** versions the *Common.exe*.

#### 6.11.8.1.2 Version dependent data

The version dependent data are the files that change from standalone to standalone of the same **radCASE** version. These files have to be put in the directory „*versionfiles*“ directly in the standalone directory. Files in this directory are copied directly into the standalone directory. Files in subdirectories of *versionfiles* are copied to the according subdirectories of the standalone. So to copy files into the *win* directory of the standalone these files have to be put into a *win* directory in *versionfiles*.

**radSEL** will copy every file found in *versionfiles* matching the name pattern below. So it is easy to add additional files needed for a specific standalone e.g. password file (refer to [Password management](#)) or a different configuration of the ribbon bar (refer to Monitoring manual).

The name pattern for files to copy is: *<Software version>\_<Device ID>\_<original filename>*

The files are automatically renamed to the original filename when copied, so the standalone will work with these files.

At least the following files have to be put into *versionfiles*:

- *sim.dll*
- *struct.lst*
- *win/modul.xml*
- *win/osdl.ini*
- *win/visual.lic*

When in doubt a directory comparison tool can be used to show differences between directories, to select files which should be put into *versionfiles*.

#### 6.11.8.1.3 Configuration file

The configuration file needs to be named „*versions.ini*“ and must be put at the same level as *radSEL.exe*. The ini-File makes use of arrays. An array is defines as follows:

- *<arrayname>\size=<#>*: Specifies the size of the array.
- *<arrayname>\<index>\<valuenam>=<value>*: A value is specified in the array. The index is 1-based means there is no index 0 and the last index is the value of the size. It is possible to specify multiple values for an arrayindex as long as the valuenames differ.

It is possible to also use the arrayname for an additional normal value

E.g.:

```
name=Default translation
name\size=2
name\1\langindex=0
name\1\translation=German translation
name\2\langindex=1
name\2\translation=English translation
```

Options for section [General]:

- **bigEndian=<0/1>**: If **bigEndian** is set to 1, the whole communication is expected to be in big endian format. This means every controller has the [Ctr-Setting MF=1](#) activated. You can't mix different memory formats in **radSEL**. If not specified the default value is **bigEndian=0**.

Every supported software version needs a section named [**<Software version>\_<Device ID>**]. The order of the sections will determine the order within the selection list.

Options for each software version:

- **name=<\$>**: The name under which the software is listed for selection in **radSEL**. If no translation is specified, for the current selected language this entry is used as the default fallback. If no name is specified the default is **<Software version>\_<Device ID>**.
- **name\size=<#>**: The size of the array of translations
- **name\<index>\langindex=<#>**: The language index of a translation (refer to [Translations](#))
- **name\<index>\translation=<\$>**: The translation for the language specified in **langindex**
- **supportedCommunication\size=<#>**: Size of supported communications array
- **supportedCommunication\<index>\connectionDll=<\$>**: The name of a DLL that is supported for communication with the target controller. Use *cclConnectionEth.dll* for ethernet communication, *cclConnection.dll* for serial communication and *Simulation* for simulation support. The order of DLLs in the configuration file determines the order of DLLs in the communication interface selection. If only one option is used throughout the configuration, the interface selection will not be displayed.
- **supportedCommunication\<index>\templateSource=<\$>**: The file used as a template for communication configuration (refer to [Communication configuration templates](#)). Directories are separated by "/". The filename is relative to the *radSEL.exe*. So a value of *templates/ethTemplate.ini* will use a file named *ethTemplate.ini* in the directory *templates* as a template for the communication configuration.
- **supportedCommunication\<index>\templateTarget=<\$>**: The file to be created when processing the template. Directories are separated by "/". The filename is relative to the standalone directory of the software version.



The selection of supported communication interfaces, is limited to **radSEL** and will not affect the started standalone. If no simulation or visualization should be supported the modus change has to be prevented by the ribbon bar configuration (refer to Monitoring manual). If a communication interface should not be available in the visualization the according communication interface plugin has to be removed.

#### 6.11.8.1.4 Communication configuration templates

The template file uses a regular expression to process the DLL-Configuration. The DLL-Configuration is in the format as described in the Monitoring manual for the command line parameter **/COMDLL=<filename>:<configuration>**. The first line of the template specifies the regular expression to used on the configuration.

All lines after this will be used to create the configuration file. In this part of the template `%<index>%` can be used as a placeholder to insert the matches of the regular expression. `%0%` will match the whole regular expression and `%1%`, `%2%`, ... will match the according subgroup.

For **radCASE** before Version 5.1.0 (Build 10744.11) communication configuration templates are needed for the communication DLLs of **radCASE**.

For `cclConnectionEth.dll` the templateTarget must be `ccITcpIp.ini`. The template file must contain:

```
^(.*)$
[General]
IP=%0%
```

For `cclConnection.dll` the templateTarget must be `ccISerial.ini`. The template file must contain:

```
^C(\d*) B(\d*) D(\d*) P(\d*) S(\d*)$
[General]
Baudrate=%2%
Databits=%3%
Parity=%4%
Stopbits=%5%
Port=%1%
```

#### 6.11.8.1.5 Translations

**radSEL** supports different languages through the usage of language files, which are located in the *languages* directory. The language files are simple text files with the extension `.Ing`. Within the file there are labels prepended with a colon which are used by **radSEL** to find the according text and the translation for the language is in the next line. For each supported language a language file is required.

If a text is not found in the language file **radSEL** will use a hard coded english text instead. The file names have the following pattern:

`<index>_<name>.Ing`

Index is the language index. This index will also be passed to the communication DLLs which have the same language mechanism as **radSEL**. For the correct language index refer to [Supported Languages](#).

The name is the name which will be displayed in the language selection.

#### 6.11.8.1.6 Distribution

After configuring **radSEL** as explained above, the directory containing **radSEL** can be distributed to end customers. The directory contains everything for the application to run.



**radSEL** uses the Qt library. The Qt libraries are licensed under a LGPL license (refer to [Qt](#)). When redistributing **radSEL** to the end customers, make sure to abide to the legal obligations of this license.

In short the legal obligations are:

- Deliver the source code of the LGPL software or have a written offer with instructions on how to get the source code. The offer from IMACS may be passed for this.
- The user has to be able to relink the software with modified versions of Qt. Because **radCASE** uses dynamically linked libraries, this can be done by replacing the according DLLs.
- Notify the user about his rights by providing a copy of the LGPL license (contained in

the directory *3rdPartyLicense*. Also a prominent notice about the usage of LGPL licensed software is necessary.

#### 6.11.8.2 Using radSEL

In the top left **radSEL** presents a list of supported standalone software versions for the currently selected communication interface or simulation. A software version can be started by double clicking the entry in the list, or selecting an entry and clicking “Start communication”/“Start simulation”. When started **radSEL** will minimize to tray and wait for the standalone to end, and restore the window.

“Search controller” will search for a connected controller on the selected communication interface with the current interface configuration. When a controller is found, the matching standalone software version is selected. This search is done automatically on starting the software or changing the communication interface or interface configuration.

In the top right corner the communication interface can be selected (if multiple interfaces are supported). With “Configure Communication Interface” the currently selected interface can be configured.

When enabling “Autoconnect” **radSEL** will search for connected controllers permanently. As soon as a connected controller is found, the software will not only be selected but the matching standalone is started automatically.

In the bottom right corner a button with different flags allows the selection of the language of the user interface.

With “Minimize” the application can be minimized to the system tray. “Quit” will exit the application.

Standalones that were created using **radCASE** version 5.1.0 (Build: 11084.32) or newer will communicate with **radSEL** and detect if another controller was connected (at least for the same communication protocol version) and if **radSEL** has a matching standalone version for that controller. If a matching standalone version is detected the start of the correct version is offered. If this option is selected the correct standalone will be started automatically if not the standalone have to be shut down manually. If no matching standalone is found an error message is displayed. That error message can be customized by using [Desktop](#)-setting **DN=<\$>** within the project the standalone is created from.

Standalones from older **radCASE** versions will just go offline or display an error message. For those versions the standalone will have to be shut down manually.

When the standalone was shut down manually and the Autoconnect feature is enabled, a dialog is displayed offering to search for another controller or to end **radSEL**.

If **radSEL** is started the last configuration is used, unless another configuration is provided using the command line (refer to [Command line options](#)).



### 6.11.8.2.1 Command line options

There are different command line options that can be used to start **radSEL**.

- ?, -h, --help: Will display a help dialog containing the command line options (English only).
- v, --version: Display version information.
- d, -dll <filename>: Select <filename> of communication DLL.
- c, --config <configuration>: Set <configuration> for selected communication DLL. Will only work in combination with option \"-d\".
- a, --autoconnect <on/off>: Turn <on/off> automatic connection to target.

### 6.11.9 Resource Consumption Analysis

The resource consumption analysis is a way to analyze the resource consumption of a model. It provides ways to get the current and past consumption of different resources and also an overview on which resources were already consumed.

The resource consumption analysis consists of three parts: An [ELEMENT](#) that has one or more energy profiles, the energy profile (refer to [EProfile](#)) which is part of an energy group and the energy group (refer to [EGroup](#)).

An energy group is a group of everything that consumes the same resource. E.g. energy group *Power5V* could be a group for everything that uses 5V as input. The energy group defines the **ELEMENT Type** which every energy **ELEMENT** uses. In the example *Power5V* this could be an [ENUM](#) with 2 **Digits** after the comma and the **Unit W**.

The energy group also defines how an overall consumption is calculated. The **COUnit** (Consumption overall unit) is the Unit that is displayed for an overall consumption **ELEMENT** the rest of the **Type** definition is the same as the **Type** definition for every energy **ELEMENT**. The **COConversion** defines a formula on how to sum up the current consumption to an overall consumption. In this formula the variable *\$co* can be used. This variable already contains a sum of the overall consumption, where the current consumption is summed up for every second, so in our example *\$co* would contain the overall consumption in *Ws*.

The formula can contain the whole syntax of **Stimulation Equations** (refer to [Stimulation Equations](#)) except the historical values. A formula can be specified to get the value in a better format, in our example we could want to have the value in *kWh* instead of *Ws* so the formula would be: *\$co / 1000 / 3600*

An energy profile defines the current resource consumption of an **ELEMENT**. The energy profile must be part of an energy group. An energy group can have multiple different energy profiles.

To define the behavior again a formula has to be defined, which again can use the whole syntax of the **Stimulation Equations** except the historical values. To calculate the current behavior of an element there are different variables that can be used:

**\$act**      The current value of the **ELEMENT** that has the specified energy profile

**\$arg[x]**    *x* is an integer defining which argument is meant starting with 0. The energy profile can

get different parameters to calculate dependencies from other **ELEMENTs** e.g. different consumption in dependency on the temperature.

An **ELEMENT** can be part of multiple energy profiles, e.g. a pump could be part of an energy profile defining the electricity consumed by the pump and also of an energy profile defining the consumption of the resource transported by the pump (e.g. water).

In the **ELEMENT** the different **EProfiles** can be put into the **EProfile** field. Multiple energy profiles are separated by semicolon, the arguments are passed in parenthesis and are separated by commas: e.g. `eprofile1(arg0, arg1, arg2, ...);eprofile2(arg0, arg1, arg2, ..)`

In the Project Monitor there is an “**energy view**”. In this view a rectangle is drawn for every energy group containing the values of that energy group.

The first **ELEMENT** is the overall consumption since start of the simulation of the current **MODUL** and all **SUBMODULs**. The second variable is the current consumption of the current **MODUL** and all **SUBMODULs**. After that the current consumption for every **SUBMODUL** is listed and at last the current consumption of every **ELEMENT** is listed. For the **ELEMENTs** there is also a connection to the energy profile, so if an element has two different energy profiles of the same group that **ELEMENT** is listed for both energy profiles.

It is also possible to show the energy consumption in a **MElem** (refer to [MElem](#)) and also record the consumption for later analysis. To use the consumption analysis in a **MElem** the generated name of the **ELEMENT** has to be used.



The **ELEMENTs** are in the same **MODUL** as the **ELEMENTs** that use an energy profile are defined, so you can also access energy **ELEMENTs** of a **SUBMODUL** with the normal syntax for **ELEMENT** access (refer to [Element Access](#))

The names of the resource consumption **ELEMENTs** are:

Name	Description
<code>Energy_Overall_&lt;EGroup-Name&gt;</code>	For the overall consumption of a <b>MODUL</b>
<code>Energy_Sum_&lt;EGroup-Name&gt;</code>	For the current consumption of a <b>MODUL</b>
<code>Energy_&lt;EProfile-Name&gt;_&lt;Element-Name&gt;</code>	For the current consumption of an <b>ELEMENT</b>

#### 6.11.10 System Event Handling

The project monitor can react on different system events by notifying the user with an e-mail or by calling an external program. The following system events are available:

- Start/Shutdown of **Project Monitor**
- Changing to Online/Offline
- Changing of a **SysEvent-ELEMENT**



A **SysEvent-ELEMENT** is defined by creating any [ENUM](#) or [EBIN](#) with the name **SysEvent**. There can be multiple **SysEvent-ELEMENTS** across the model, but only one per **MODUL**. On every change of that **ELEMENT** a system event is triggered and the data of the according **SysEvent-ELEMENT** is passed to the user with an e-mail or to an external program. When a **SysEvent-ELEMENT** triggers a system event it is possible to pass data of additional **ELEMENTS** to the user/external program, by adding the [Com Type XE](#).

For further information on how to enable/disable system events and how to access the data passed to an external program refer to the Project Monitor manual.

### 6.11.11 Simulated Display Functions

#### 6.11.11.1 Change Foreground/Background Color

It is possible to change the colors of the simulated display. This functionality enables you to simulate special hardware functions like a screensaver/energy saving function where the backlight is turned off after a while, or special monochrome displays where the colors of the display can be changed.

To change the colors of the display in the simulation the two functions *dll\_dis\_fgcolor()* and *dll\_dis\_backcolor()* are available. *dll\_dis\_fgcolor()* changes the global foreground color of the display and *dll\_dis\_backcolor()* changes the global background color of the display.

Both functions require the same two parameters:

```
void dll_dis_backcolor(short screenID, long bcolor)
void dll_dis_fgcolor(short screenID, long bcolor)
```

**screenID** Identifier to identify the screen. You can get this parameter using a special function (refer to example below).

**bcolor** RGB-value of the color the background/foreground color should be set to.

Example:

```
#ifndef __CTR__
    EMB_HMI *screen;
    screen = RootScreenI->getScreenData();

    // Set Backgroundcolor to blue
    dll_dis_backcolor(screen->screenID, #000000FF);
    // Set Foregroundcolor to red
    dll_dis_fgcolor(screen->screenID, #00FF0000);
#endif
```



To use these functions the file *\$OSDL\_SYS\Def\htmdec.h* has to be included.

#### 6.11.11.2 Replace Colors Of An Area

You can replace colors in an area of your simulated display using the function *dll\_dis\_replaceColors()*.

The function has the following parameters:

```
void dll_dis_replaceColors(short screenID, short x, short y, short sizex, short sizey, long*
fromCol, long* toCol, short arraySize);
```

<b>screenID</b>	An identifier to identify the screen. You can get this parameter using a special function (refer to example below).
<b>x, y</b>	Position of the area, where the colors will be replaced
<b>sizex, sizey</b>	Size of the area, where the colors will be replaced
<b>fromCol</b>	Array of RGB-Values of the colors that should be replaced
<b>toCol</b>	Array of RGB-Values of the colors that are used to replace the colors of <i>fromCol</i>
<b>arraySize</b>	size of the arrays for color replacement

Example:

```
long colFrom[3];
long colTo[3];

// Change Color 0x00FF00 to 0xCCFFCC
colFrom[0] = 0x00FF00;
colTo[0] = 0xCCFFCC;

// Change color 0xFF0000 to 0xFFCCCC
colFrom[1] = 0xFF0000;
colTo[1] = 0xFFCCCC;

// Change color 0x0000FF to 0xCCCCFF
colFrom[2] = 0x0000FF;
colTo[2] = 0xCCCCFF;

#ifdef __CTR__
    EMB_HMI *screen;
    screen = RootScreenI->getScreenData();
    dll_dis_replaceColors(screen->screenID, 12, 15, 40, 40, colFrom, colTo, 3);
#endif
```



To use this function the file `$OSDL_SYS\Def\htmdec.h` has to be included.

### 6.11.12 Customizing The Simulation Project

There are several ways to customize the simulation project:

It is possible to separate code for simulation and embedded controller by using the Define `__CTR__`. This Define is only set on the embedded controller and `#ifdef` / `#ifndef` can be used to

only execute code in the simulation or on the embedded controller. This can be used to call special functionality on the embedded controller and to simulate that special functionality in the simulation.

To simulate special functionalities sometimes it is mandatory to include some additional libraries or add some further C-Files to the Simulation. There are two ways to achieve this, both use the directory *CustomProj*. radCASE automatically replaces the Simulation project to always use an up to date project matching the highest supported Visual Studio version available on the PC. When cleaning up, radCASE will delete everything in the *Sim* directory, except the directory *CustomProj*. After creating the simulation project everything from the *CustomProj* directory is copied into the *Sim* directory, replacing any existing files in the process. So creating the directory *CustomProj* in the *Sim* directory and putting files into it, is the only way to alter the simulation project permanently.

There are three files, which can be added there, to alter the behaviour of the standard simulation project:

1. *projectSpecificFiles.txt* is used to add additional files to be compiled and linked to the project
2. *projectSpecificIncludeDirectories.txt* is used to add additional directories to the search path for include files
3. *projectSpecificLibraries.txt* is used to add additional libraries to be linked to the project

All these files share the same format. Each file or directory to add has to be in a separate line. The filenames/directory names can use a relative path to the *Sim* directory, an absolute path or a path using environment variables e.g. %OSDL\_ECOSOLUT%.

If a file added to *projectSpecificLibraries.txt* does not exist, the entry from the txt-file is passed without change to the simulation project. This allows adding system libraries like e.g. *ws2\_32.lib*.

If the changes possible in this way don't suffice the project files can be changed directly and the modified project files can be put into the *CustomProj* directory. This will result in the modified ones being used and the standard project files being overwritten. This however will limit the simulation project to the Visual Studio version of the project files used in that directory. The automatic detection of the highest supported Visual Studio version will not work anymore.



After creating a *CustomProj* directory with modified project files those files are always used, so the developer has to ensure they are always up to date (when updating radCASE). On updates of radCASE watch especially for the chapter "Custom Sim" in the radCASE-Changelog.

### 6.11.13 Help

It is possible to add project specific instructions on how to use the Project Monitor for the project by adding a document as help file.

After the first execution of the Simulation or Visualization the file *config.ini* is created in the Develop-directory or for the standalone (refer to [Standalone](#)) in the directory *lwin*. After the creation the *config.ini* needs to be edited.

If for example the help file would be named *MyHelpFile.pdf*, add the following line under the section

```
[GENERAL]:
```

```
HELPPFILE=MyHelpFile.pdf
```

The document *MyHelpFile.pdf* has to be located in same directory with the file *radMON.exe*. If the help file should be located in a subdirectory (e.g. *help*) of the standalone version, you can add the relative path (starting from the directory where the *radMON.exe* is located).

Example:

```
HELPPFILE=help\MyHelpFile.pdf
```

## 6.12 Documentation Generation

The documentation generation is a way to automatically create documentation from a radCASE model. For the Embedded HMI the extent of the documentation has to be selected (refer to [Defining Target HMI For Documentation](#)). The content is depending on the selected language and documentation level (refer to the according editor manual of the editor you are using, for information on how to select a language and documentation level for export). For more information on the contents of the generated documentation refer to [Structure Of Generated Documentation](#).

The documentation is created into the selected directory according to [Desktop](#)-setting **DD=<\$>**.



Independent from this setting, the file *doc\printall.htm* is created redirecting to the created documentation. This file only serves as central entry point for the editor.

The generated documentation can be modified by creating the batch *docgen\_post.bat* in the *Develop*-directory of the project. The batch will be executed after the creation and the path to the generated documentation will be passed as a parameter to the batch. This can be used e.g. to copy a user defined CSS-file to the documentation (refer to **Desktop**-Setting **CF=<\$>**)

### 6.12.1 Defining Target HMI For Documentation

The usage of [Element Visualization](#) and [Conditional Drawing](#) results in a number of possible different interface formats, but it is not desirable to document all of those combinations. For some **SURFACE\_CTRs** however it can be useful to document multiple different states and some of the surfaces shouldn't be documented at all. To influence the way the Target-HMI is documented radCASE offers the mechanism of [DOCTABs](#) and [DT\\_ENTRYs](#).

The existence of a **DOCTAB** in a **SURFACE\_CTR** determines if the surface should be documented. A **DOCTAB** must have at least one **DT\_ENTRY**. The **Doc. Level** in the **DT\_ENTRY** determines the minimum documentation level needed for the surface defined by the **DT\_ENTRY** to be included in the documentation.

If the documentation of the surface with the standard values of all **ELEMENTs** is sufficient the **Elements** in the **DOCTAB** and the according **Line** in **DT\_ENTRY** may be left blank.



The existence of a **DOCTAB** affects the documentation of a surface and all surfaces which are opened from that surface. This means, if a surface has no **DOCTAB** or all **DT\_ENTRYs** require a higher documentation level than currently selected, the surface will not appear in the documentation, and all surfaces which are opened from that surface are not documented either, even if they are defined with a **DOCTAB**.

Surfaces in a Subnode (refer to [Distributed Systems](#)) are only documented if subnode complete export is enabled (**UKE=<0/1>** in [Ctr](#)).



If adding a **DOKTAB** to a surface, the **DOKTAB** must always be the first RC-Item in the surface.

For influencing the **Conditional Drawing** of a surface, **ELEMENT** values can be set using the **DOKTAB**. To do so, a list of all **ELEMENT**s which should not have their standard value must be provided in the **Element** list of the **DOKTAB**. The **ELEMENT**s have to be comma separated and referencing of **ELEMENT**s in other **MODUL**s is also possible by providing the according **MODUL** path (refer to [Element Access](#)). For each of the provided **ELEMENT**s the values have to be provided in the **Line** of the according **DT\_ENTRY**. The values must also be comma separated and have to be in the same order like the **ELEMENT**s in the **DOKTAB**.



This mechanism is currently only supported for [ENUM](#)s and [EBIN](#)s. For **EBIN**s the numerical value has to be provided. It is not possible to select an **EBIN** value using its **Name**.

It is possible to define multiple **DT\_ENTRY**s for one **DOKTAB** with different **ELEMENT** values. Each of the **DT\_ENTRY**s will be exported as one entry in the HMI-Overview. Also a HTML page will be exported for each of the **DT\_ENTRY**s. The name of those files will be:

`<MODUL instance name><Surface number>_<Page number>_<HTML number>.htm`

The page number in this file name is automatically generated by radCASE to get unique filenames for each **DT\_ENTRY**. The HTML number is the one provided in the **DT\_ENTRY**.

### 6.12.2 Structure Of Generated Documentation

The generated documentation is in HTML format. The amount of documentation contained is determined by the selected documentation level. The [Desktop](#)-Setting **DDT=<#>** determines the minimum documentation level for a developer documentation. With activated developer documentation the project contains more information (see below and in sub chapters). Also there are some items which are only exported if the documentation level is greater or equal to a minimum documentation level for export specified in the design for this item (see below and in sub chapters).

The start page contains the following information:

<b>License for</b>	Contains licensing information of radCASE
<b>Title</b>	The name of the software project (refer to <a href="#">DOCTITLE</a> )
<b>Project version</b>	Version of the software project (refer to <a href="#">VERSION</a> )
<b>radCASE Compiler Version</b>	Shows the version of the radCASE compiler used to create the documentation
<b>Created</b>	Date of creation of the documentation

The generated documentation is subdivided into the following sections:

<a href="#">System Informations</a>	General system information (only in developer documentation)
-------------------------------------	--

<a href="#"><u>TypeDefs</u></a>	Information on <b>TYPEDEFS</b> (only in developer documentation)
<a href="#"><u>PictDefs</u></a>	Information on <b>PICTDEFS</b> (only in developer documentation)
<a href="#"><u>VisualDefs</u></a>	Information on <b>VISUALDEFS</b> (only in developer documentation)
<a href="#"><u>Elements</u></a>	Information on <b>ELEMENTs</b> (more element types in developer documentation)
<a href="#"><u>Communication</u></a>	Information on CANopen <b>ELEMENTs</b> (if available)
<a href="#"><u>Module tree</u></a>	Information on <b>MODULs</b> as instance tree (containing more information in developer documentation and information according to documentation level).
<a href="#"><u>Module list</u></a>	Information on <b>MODULs</b> (containing more information in developer documentation and information according to documentation level).
<a href="#"><u>PC-GUI (SURFACE VIS)</u></a>	List of all <b>SURFACE_VIS</b>
<a href="#"><u>Embedded HMI</u></a>	Detailed list of all <b>SURFACE_CTR</b>
<a href="#"><u>HMI-Overview</u></a>	Overview of all <b>SURFACE_CTR</b>
<a href="#"><u>HMI-Navigation</u></a>	Detailed information on different <b>SURFACE_CTR</b> with navigation

### 6.12.2.1 System Informations

The System Informations section is divided into [Instances](#) and [EEPROM](#) detail sections.

#### 6.12.2.1.1 Instances

The Instances section contains information on the count of different items in the system:

- Number of [ELEMENTs](#) of Type [ENUM](#)
- Number of **ELEMENTs** of Type [EBIN](#)
- Number of **ELEMENTs** of Type [ESTR](#)
- Number of **ELEMENTs** of Type [EDAT](#)
- Number of **ELEMENTs** of Type [ETIM](#)
- Number of **ELEMENTs** in total
- Number of [MODUL](#) instances
- Number of State Machines (refer to [Finite State Machines](#))
- Number of Signal Diagrams (refer to [Signal Diagrams](#))
- Number of C-Functions (refer to [PROC](#))
- Number of Activity Charts (refer to [ACTIVITY](#))
- Number of Sequence diagrams (refer to [SEQUENCE DIAGRAM](#))

#### 6.12.2.1.2 EEPROM

The EEPROM section shows the memory requirements for data stored in the EEPROM.

### 6.12.2.2 TypeDefs

The section TypeDefs contains a list of all [TYPEDEFs](#) in the project. There is a list for each data type (refer to [Data Modeling](#)) and within each data type the **TYPEDEFs** are listed for each library file. For each of the **TYPEDEFs** there is detailed information.

### 6.12.2.3 PictDefs

The section PictDefs contains a list of all [PICTDEFs](#) in the project. The [PICTs](#) are listed for every library file and a preview image is shown.

### 6.12.2.4 VisualDefs

The section VisualDefs contains a list of all [VISBINICONS](#) in the project. The VISBINICONS are listed for every library file and contain a preview image of every possible status.

### 6.12.2.5 Elements

The section Elements contains a list of all [ELEMENTs](#) with different [Assign types](#). The **ELEMENTS** are sorted by **Assign type** and contain detailed information about each **ELEMENT**.

The developer documentation contains **ELEMENTs** of **Assign type**: **PAR**, **SYS**, **PROC**, **AI**, **AO**, **CNT**, **DI**, **DO** and **CONST**

The user documentation contains **ELEMENTs** of **Assign type**: **PAR**, **AI**, **AO**, **CNT**, **DI** and **DO**.

The documentation only contains **ELEMENTs** with the required documentation level. This means **ELEMENTs** are only included if the documentation level is greater or equal to the required **Doc. Level** of the **ELEMENT**. Also **ELEMENTs** within **MODULs** without the required documentation level are not listed.

The **Documentation** comment of the **ELEMENT** will only be included in the developer documentation.

### 6.12.2.6 Communication

The section Communication contains information about CANopen [ELEMENTs](#) (refer to [CANopen communication](#)). All CANopen **ELEMENTs** are listed with their CANopen attributes.

Each **ELEMENT** is listed with its CANopen index and the following attributes:

<b>Objectcode</b>	ARRAY for <b>ELEMENT</b> -Arrays and VAR for normal <b>ELEMENTs</b>
<b>Canopen Subindex</b>	Name of <b>ELEMENT</b>
<b>Module path</b>	Instantiation path of <b>ELEMENT</b>
<b>Data type</b>	Type of <b>ELEMENT</b> ( <a href="#">TYPEDEF</a> )
<b>Attribute</b>	CANopen attribute
<b>Default</b>	Default value of <b>ELEMENT</b>



<b>Range</b>	Value range according to <b>ELEMENT</b> type
<b>PDO-Mapping</b>	Index, position and type of PDO (only listed for PDOs)
<b>Description</b>	<b>Documentation</b> info for <b>ELEMENT</b>

#### 6.12.2.7 Module tree

The section Module tree contains an instance tree of the project ([SUBMODUL](#) tree). It contains only instances with the required documentation level. This means those instances are only included if the documentation level is greater or equal to the required **Doc. Level** of the **SUBMODUL**. Each instance links to the **MODUL** documentation as described in [Module list](#).

#### 6.12.2.8 Module list

The section Modules contains a list of used [MODULs](#) in a project. Only **MODULs** are listed, which have at least one instance with the required documentation level. This means the documentation level for the documentation is greater or equal to the required **Doc. Level** of the according [SUB-MODUL](#). For each **MODUL** a detailed description of the **MODUL** is available containing the following sections:

<b>Info</b>	Contains the <b>Info</b> text of the <b>MODUL</b>
<b>Comment</b>	Contains the <b>Comment</b> text of the <b>MODUL</b> . (the comment can contain more information in developer documentation refer to <a href="#">Comment Syntax</a> ).
<b>Interface</b>	Contains detailed information on <a href="#">ELEMENTs</a> and <a href="#">PROCs</a> used as interfaces of the <b>MODUL</b>
<b>Intern</b>	Contains detailed information on internal <b>ELEMENTs</b> of the <b>MODUL</b> . Refer to <a href="#">Elements</a> for information which element types are listed for developer and user documentation.
<b>Artefacts</b>	List of <a href="#">ARTEFACTs</a> in the <b>MODUL</b> . Refer to <a href="#">Artefact documentation</a> .
<b>State machines</b>	Contains information on the State Machines (refer to <a href="#">Finite State Machines</a> ) of the <b>MODUL</b> . (only in developer documentation)
<b>Activity Diagrams</b>	Contains information on the <a href="#">ACTIVITYs</a> contained in the <b>MODUL</b> . (only in developer documentation)
<b>Submodule</b>	Contains information on the <a href="#">SUBMODULs</a> of the <b>MODUL</b> . (only in developer documentation and only if the required documentation level is met)
<b>Signal function</b>	Contains information on the Signal diagrams (refer to <a href="#">Signal Diagrams</a> ) of the <b>MODUL</b> . (only in developer documentation)
<b>Methods</b>	Contains detailed information on the internal <b>PROCs</b> of the <b>MODUL</b> . (only in developer documentation)



If an instantiated **MODUL** is not listed it is still documented. On releasing the generated documentation please make sure there are no information included in the documentation, which are not meant to be seen by the receiver of the documentation.



### 6.12.2.9 PC-GUI (SURFACE\_VIS)

The section PC-GUI contains a list of preview images for all **SURFACE\_VIS** in the project. The section only contains preview images of surfaces with the required documentation level. This means those surfaces are only included if the documentation level is greater or equal to the required **Doc. Level** of the **SURFACE\_VIS**.

### 6.12.2.10 Embedded HMI

The section Embedded HMI contains a list of all **SURFACE\_CTRs** which are marked for documentation (refer to [Defining Target HMI For Documentation](#)). For each **SURFACE\_CTR** a preview image and a description of all Actions that may be triggered and of any Menu contained are listed.

### 6.12.2.11 HMI-Overview

The section HMI-Overview contains a hierarchically structured list of preview images of all **SURFACE\_CTRs** which are marked for documentation (refer to [Defining Target HMI For Documentation](#)). The section grants an overview over all Surfaces and the navigational structure.

### 6.12.2.12 HMI-Navigation

The section HMI-Navigation is a way to navigate through the different **SURFACE\_CTRs** which are marked for documentation (refer to [Defining Target HMI For Documentation](#)). The first surface shown is **SURFACE\_CTR(0)** of [The System MODUL](#). For each **SURFACE\_CTR** a preview image and a description of all Actions that may be triggered and of any Menu contained are listed. All navigational actions are exported as links to the according surface to provide the navigational structure.

## 6.12.3 Artefact documentation

[ARTEFACTs](#) are documented in three different places. Within the [MODUL](#)-documentation **ARTEFACTs** of the MODUL can be documented using a placeholder (refer to [Comment Syntax](#)). Additionally there is an **ARTEFACT**-documentation in the files *Artefact\_<Artefact-Type>.htm* and *Artefact\_<Artefact-Type>.txt*. The .htm file contains the artefacts in a table within a HTML-file and the .txt file contains the same table in CSV format with TABs as delimiters. The **ARTEFACT**-documentation contains the documentation of each **ARTEFACT**-instance so it is possible to affect this documentation through the [EntityTab](#).

For an **ARTEFACT** to be included in the documentation two preconditions have to be satisfied:

1. The necessary documentation level has to be met. This means the current documentation level is greater or equal to the **Doc. Level** of the **ARTEFACT**.
2. The **ARTEFACTs** name, does not start with an '#' (refer to [Artefact parameter placeholders](#)).

Each **ARTEFACT** has additional parameters which can be custom defined within the [ARTEFACTDEF](#). Within the parameters different placeholders can be used (refer to [Artefact parameter placeholders](#)).

The appearance of the **ARTEFACT**-documentation is dependent on the **type of documentation** defined in the according **ARTEFACTDEF**. Currently only the type "Table" and "TableRotated" are supported.

Within a table, the **ARTEFACT**s are listed according to the parameters specified in the **ARTEFACTDEF**. The parameters are the columns in the order specified in the **ARTEFACTDEF**. The **Description** of the parameter is used as the table heading. Within the **ARTEFACT**-documentation there is a further first column containing the instantiation path of the **ARTEFACT**. This path is only included in the developer documentation (refer to [Desktop](#)-Setting **DDT=<#>**)

### 6.12.3.1 Artefact parameter placeholders

Within **ARTEFACT** parameters different placeholders can be used. While processing the placeholders the order of processing **ARTEFACT**s is in the order of instantiation in the model from top to bottom. In case of instantiation of a **SUBMODUL** the **ARTEFACT**s within the **SUBMODUL** will be processed first, before further processing the current **MODUL**.

<b>\$\$</b>	For printing a Dollar (Escaping)
<b>\$DM</b>	<p>For inserting the module description (depends on instantiation and will therefore not work in <b>MODUL</b> documentation).</p> <p>It is possible to reference the containing module or upper modules by inserting a backward reference <b>\</b>. E.g. <b>\$_\DM</b> will insert the description of the containing module and <b>\$_\_DM</b> will insert the description of the module two layers up.</p>
<b>\$&lt;Counter-Operation&gt;(&lt;#&lt;Filling&gt;)&lt;#&gt;</b>	<p>For each parameter within an <b>ARTEFACTDEF</b> there is a global counter, shared by all <b>ARTEFACT</b>s, which can be affected by different operations. An operation consists of an operator and a value.</p> <p>If the operator is in front of the value, the operation is performed first and the counter value after the operation is used to replace the placeholder. If the operator is behind the value the placeholder is replaced first and after that the operation is performed.</p> <p>The allowed operations are:</p> <ul style="list-style-type: none"> <li><b>+</b>: To add the value to the counter</li> <li><b>-</b>: To subtract the value from the counter</li> <li><b>=</b>: To set the counter to the value</li> </ul> <p>The value of the operation has to be an unsigned integer. The starting value of a counter is 0.</p> <p>Additionally an optional filling can be specified. The filling is identified by a <b>#</b> after the operation. The first character after the <b>#</b> is the character used for filling in front of the string. After the character the width of the string is specified.</p> <p>e.g.: <b>\$=5\$</b> will output "5"</p> <p><b>\$=5#03\$</b> will fill the string to a width of 3 using '0', so the resulting output will be "005". The width is a minimum width of the string, the string is not limited to that length. So e.g. <b>\$=123#02\$</b> will result in 123.</p> <p>(the feature depends on instantiation and will therefore not work in <b>MODUL</b> documentation)</p>

### **\$C<Comparison Operation><Comparing String>\$**

For each parameter within an **ARTEFACTDEF** a comparison can be enabled, which will be applied on all following **ARTEFACT**s including the current **ARTEFACT**. The whole parameter after replacing placeholders will be compared as a string to the comparing string set by this placeholder using the comparison operation set by this placeholder.

The placeholder itself will be replaced with an empty string.

Only **ARTEFACT**s without an active comparison or which meet the conditions will be displayed in the documentation.

The allowed comparison operations are:

>: Will display all **ARTEFACT**s with a parameter greater than the comparing string

>=: Will display all **ARTEFACT**s with a parameter greater than or equal to the comparing string

<: Will display all **ARTEFACT**s with a parameter lesser than the comparing string

<=: Will display all **ARTEFACT**s with a parameter lesser than or equal to the comparing string

==: Will display all **ARTEFACT**s with a parameter equal to the comparing string

!= : Will display all **ARTEFACT**s with a parameter not equal to the comparing string

An active comparison can be disabled by using no comparison operation and comparing string: **\$C\$** (the feature depends on instantiation and will therefore not work in **MODUL** documentation)

#### 6.12.4 Comment Syntax

When commenting a [MODUL](#) by standard the documentation generation will escape characters like "<" to be displayed as a "<" in the generated HTML-code (so replacing it internally with &lt;). Also line breaks will be generated as line breaks in the generated documentation. Using \$-signs special placeholders can be inserted into the generated documentation.

Because the \$ is used to determine the placeholders the dollar must be escaped to print a \$ in the generated documentation:

**\$\$** For printing a Dollar (Escaping)

The following placeholders can be used to format the generated text:

**\$b\$<text>\$/b\$** Prints the <text> bold

**\$i\$<text>\$/i\$** Prints the <text> italic

**\$u\$<text>\$/u\$** Prints the <text> underlined

**\$t:<indentation>\$<text>\$/t\$** Indents the <text> by <indentation> pixels

**\$html\$<text>\$/html\$**

The specified text is marked as HTML-code. In this text the automatic replacement of special characters and line breaks is deactivated, enabling the user to directly insert HTML-code into the generated documentation. In this HTML-code the user has to manage line breaks and HTML-escaping. However placeholders can still be used in the HTML-code and the \$ sign still needs to be escaped.

The following placeholders can be used to insert content into the generated documentation:

**\$private\$<text>\$/private\$**

<text> is a private comment and can not be copied using **\$CM:<modulename>\$** or **\$CM:\$parent\$**. See also **\$protected\$**

**\$protected\$<text>\$/protected\$**

<text> is a protected comment and can not be copied using **\$CM:<modulename>\$**. In difference to **\$private\$** the <text> will be copied for **\$CM:\$parent\$**.

**\$developer\$<text>\$/developer\$**

<text> will only appear in developer documentation (refer to [Desktop](#)-setting **DDT=<#>**).

**\$:<submodname>\$**

Prints the **Name** of [SUBMODUL](#) <submodname> and creates a link to the documentation of the referred **MODUL**.

**\$LM:<submodname>\$**

Prints the **ID** of **SUBMODUL** <submodname> and creates a link to the documentation of the referred **MODUL**.

**\$DM:<submodname>\$**

Prints the **Description** of **SUBMODUL** <submodname>. No backward references using \_\ are allowed.

**\$IM:<submodname>\$**

Prints the **Info** of **SUBMODUL** <submodname>. No backward references using \_\ are allowed.

**\$CM:<submodname>\$**

Prints the **Comment** of the **MODUL** referred to by **SUBMODUL** <submodname>

**\$CM:\$parent\$**

Prints the **Comment** of the **MODUL** the current **MODUL** is derived from (refer to [Inheritance](#))

**\$:<elemname>\$**

Prints the **Name** of [ELEMENT](#) <elemname>

**\$DE:<elemname>\$**

Prints the **Description** of **ELEMENT** <elemname>

**\$IE:<elemname>\$**

Prints the **Info** of **ELEMENT** <elemname>

**\$CE:<elemname>\$**

Prints the **Documentation** of **ELEMENT** <elemname>

**\$TextDefLink:<TextID>\$**

Will use a multilingual text according to selected documentation language.

**\$AT:<artefacttype>\$**

Will print the documentation of the [ARTEFACT](#) type <artefacttype>. The documentation will be formatted as specified in the [ARTEFACTDEF](#) and include all **ARTEFACT**s of the specified type in the current **MODUL**.

Because the **MODUL** documentation is independent of instantiation, the Artefact parameter placeholders depending on the instantiation will not work in this placeholder (refer to [Artefact parameter placeholders](#)).

## 6.13 Web visualization

The web visualization is a generated webpage which can be used to communicate with a target and display and change the current state of the target software. The web visualization needs a correctly set up webserver (refer to Integration manual).

The web visualization is created by adding **SURFACE\_WEBs** to the project (refer to [HMI](#)). The generation starts with SURFACE\_WEB(0) in the [The System MODUL](#).

## 6.14 Remote HMI on Web (browser)

Activating this feature makes it possible to visualize the HMI of a target on a browser and control it remote.

### 6.14.1 How it works

The following interfaces are involved:

#### 6.14.1.1 The Display command buffer of the controller

In the activated RDN\_CIRCULAR\_DISPLAYBUFFER of the visualized target, the elements (text, bitmap, line, rectangle, etc.) of the display are stored.

Therefore in the SystemDef-area of the design, a „RD\_DISPCMD\_COMMUNICATION 1“, „RD\_HMI\_ON\_WEB 1“ and for example „RD\_DISPCMD\_BUFSIZE 4096“ have to be inserted. At which the least define, the display buffer size, is dependend of the display size respective the number of displayed elements. Furthermore the dynamic of the HMI and the time between the requests of the display content may not lead to a buffer overflow.

#### 6.14.1.2 radCase network interface rdsockets.c/h

With this network interface API implemented on the controller, connections can be made and data exchanged between browser (PC, smartphone, ...) and controller via TCP/IP sockets.

#### 6.14.1.3 Webserver httpsrv.c/h on the target

Also running on the controller is a web server that receives connections from a browser (e.g., Chrome or FireFox) on TCP/IP network port 80.

This http server can send requested files / data to the browser, whereby the following file names are internally intercepted and operated by the web server and are not processed as files:

- „**gettype.xml**“: Character encoding (**ASCII**, **UTF-8**, **Unicode**), pixel resolution and controller/project name are separated by a colon.

To do this, a function "`const char * GetCtrType(void)`" must be implemented in the application, which returns this informations (for example: "U:640x480:ZE-XL").

- „**status.xml**“: This "file" is requested cyclically by the browser. The webserver answers with the current display buffer content.
- **All other file names**: All other requested files/data must be sent to the browser via the application-specific "SendFile"-callback function set via "`int SetSendFile-Callback(fnSendFileCallback func)`".

Reason: The general server can not manage the application-specific file accesses (SD card, Linux flash partition, ...).

Attention! In this function, the file name "**coltab.bin**" must either directly send the ColTab[256]-array or there must be a corresponding file on the controller.

Also the requested "**osdl.bmp**-file" has to be implemented and may send the bit-map-data of the project! To do so, the size of osdlbmp[]-array must be known. → The size variable osdlbmp\_size will be generated with this \TOOLS\BIN2C

```
..\CTR\APPL\BIN\osdl.bmp osdl_bmp.c osdlbmp 1 1
```

With the following functions, callbacks can be set, to pass on information to the application:

- `int SetKeyCallback(fnKeyCallback func)`: Keys pressed in the browser are passed to the specified function.
- `int SetTouchCallback(fnTouchCallback func)`: X/Y positions clicked or "tapped" in the browser are passed on to this callback function.
- `int SetOnlineCallback(fnVoidCallback func)`: The function specified here is called as soon as a browser has established a connection to the controller. Therein, e.g. redraw the display so that the display buffer contents contain the complete current surface.

#### 6.14.1.4 Application-specific callback functions (repetition)

This allows the application to respond to the requests of the browser.

- `int KeyCallback(int ch) → ch: KeyCode` see ctr\_cons.h.
- `int TouchCallback(short x, short y) → clicked X / Y position.`
- `void OnlineCallback(void) → trigger rebuilding of the surface.`  
Example: "fExtRedrawAllTrigger = TRUE;"
- `int SendFileCallback(char * pszFileName, tSocket client) → Send requested file/data to the browser.`

Example „coltab“:

```
if (strstr(pszFileName, "coltab.bin"))
{
    unsigned long o=0, tabsz=sizeof(ColTab);
    unsigned char* p = (unsigned char*)ColTab;

    while(o < tabsz)
    {
        int size = 2048;

        if ((o+size) > tabsz)
```

```

        size = tabsize - o;

        if (rdSend(client, (char*)&p[o], size, 0) <= 0)
            return -1;

        o += size;
    } //while(o < tabsize)

    return 0;
}

```

#### 6.14.1.5 Files to be Preserved on the Controller

##### →index.html

The actual "homepage" of the controller. This contains the JavaScript code, which i.a. cyclically requests the display buffer and draws/displays in the browser.

##### →ajaxpoll.js

Here are the functions for building an XMLHttpRequest to the controller, which implement the data exchange.

##### →consttables.js

The tables with the character widths of the supported proportional fonts. (See also font files below.)

##### →app\_specific.js

Here are the application-specific implementations of the homepage. This allows a touch keyboard to be implemented in the browser (for operation with a smartphone, for example):

- function appCtrType (CtrType, resolution)  
The data received from the controller is reported / forwarded (e.g., CtrType = "ZE-XL" and resolution = "640x480").
- function appDrawAfterClr()  
This function is called whenever the controller display and thus the browser display is to be redrawn.  
Here, for example, the touch keyboard should be drawn.
- function appOnMouseDown (ctx, x, y)  
Here you can evaluate, if a "touch-key" was clicked.  
A return value of 1 indicates that this position should not be reported to the controller. Otherwise, a 0 should be returned.

##### →favicon.ico

The icon which is displayed in the browser on the left corner of the title bar.

##### →bs.png, enter.png, key.png, shift.png

For example, images used for buttons of a virtual application-specific keyboard.

##### →Font files of all display fonts used in .png format:

fnt10p16.png



fnt12p19.png  
 fnt12x16.png  
 fnt14p22.png  
 fnt16p24.png  
 fnt16x24.png  
 fnt20x32.png  
 fnt24p36.png  
 fnt4x6.png  
 fnt6x8.png  
 fnt7p8.png  
 fnt8p12.png  
 fnt8x10.png  
 fnt8x12.png  
 fnt8x16.png  
 fnt8x20.png  
 uni16x16.png

#### 6.14.2 How to integrate 'remote HMI on web' into a project

1. Add a define **RD\_DISPCMD\_COMMUNICATION=1** and **RD\_HMI\_ON\_WEB=1** in the **SystemDef** of the project design.
2. The file "`\System\rdsockets.c`" must be implemented for the target.
3. Add and compile the webserver "`\System\httpsrv.c`" for the target.
4. Copy the file "`\System\http_app_template.c`" to "`\target\target_specific\http_app.c`" and make application-specific changes (– see also 6. + 7. of the list here)
5. Integrate the files of the target HMI website/homepage `\rc_lib\webremotehmi`:  
**Always:** index.htm, ajaxpoll.js, consttables.js, app\_specific.js, favicon.ico  
**Application specific:** bs.png, enter.png, key.png, shift.png  
**Used fonts:** fnt10p16.png, fnt12p19.png, fnt12x16.png, fnt14p22.png, fnt16p24.png, fnt16x24.png, fnt20x32.png, fnt24p36.png, fnt4x6.png, fnt4x6.png, fnt4x6.png, fnt4x6.png , fnt16x24.png, fnt16x24.png, fnt4x6.png, fnt8x10.png, fnt8x12.png, fnt8x16.png, fnt8x20.png, uni16x16.png
6. Call **http\_init()** in the **UserInit()**. There the web server task should be started and also the callback functions should be set for key or touch position processing.
7. `http_app.c`:  
 In this file the callback functions must/can be implemented →  

```

int KeyCallback(int ch);
int TouchCallback(short x, short y);
void OnlineCallback(void);
void http_error(char* szError);
int SendFileCallback(char* pszFileName, tSocket client);
void http_init(void);

```



## 7 Troubleshooting

### 7.1 Licensing Issues

#### 7.1.1 No license file available. The program will be aborted. Please contact your dealer.

If this message appears during model compile the model compiler could not find a license file. There are two different locations where a license file could be located:

1. In the *DEVELOP*-directory of the project, there should be a *radCASE.lic* or a *radon.lic*. Normally this file is located in the directory of the radCASE editor (e.g.: *C:\Program Files\radCASE\radEDIT*) and is automatically copied to that directory. To check if the editor has the correct license please refer to the according Editor manual of the editor you are using. You should also check the access rights of the *DEVELOP*-directory.
2. For HASP dongles it is also possible to have the license file located on that dongle. If using a HASP dongle with license file on the dongle, you should check if the dongle can be accessed from the according PC. You can also have a look into the file *DEVELOP\logfile.txt*, to see where radCASE looked for a license.

If the message appears when trying to start the **Project Monitor** the **Project Monitor** could not find a license file. The license file should be in the *DEVELOP*-directory of the project and is named *visual.lic*. The *visual.lic* is generated by the model compiler during model compilation. You should check the access rights of the *DEVELOP*-directory.

#### 7.1.2 License not sufficient. The program will be aborted.

If this message appears the license does not contain all packages required for model compiling the project.

The following table lists all packages available for radCASE and the resulting restrictions of a not activated package:

Package	Restrictions if package is not available	Comment
BAS	-	Always enabled
OOM	-	Always enabled
HMI	Target-HMI can't be used	The project may not contain any <b>SURFACE_CTR</b> (refer to <a href="#">Target HMI</a> )
VSF	Standalone can't be passed to third party	The <b>Project Monitor</b> will only work with a valid dongle
UML	No graphical modeling of functions	Usage of Statemachines (refer to <a href="#">Finite State Machines</a> ), Activity Charts (refer to <a href="#">ACTIVITYCHARTS</a> ) and Sequence diagrams (refer to <a href="#">SEQUENCE DIAGRAM</a> ) is forbidden

Package	Restrictions if package is not available	Comment
PLC	No implementation of FUP and ST according to IEC61131-3	Usage of ST-Code and signal diagrams (refer to <a href="#">Signal Diagrams</a> ) is forbidden
DIS	Can't model distributed systems	Can't use <b>Submodule type Subnode</b> or <b>Pointer</b> in a <a href="#">SUBMODUL</a> .
ERS	No support for enhanced recording and stimulation	Can't generate and use sequences or sync recording mode
ITC	Can't use interface to external tools	Can't use interface to get current data and drive sequences from an external tool
EAM	Can't use Enterprise architect for modeling a project	
RCA	No resource consumption analysis	Can't use <a href="#">EProfiles</a> and <a href="#">EGroups</a>
WEB	No web visualization	Can't start creation of web visualization
INT	Can't use international HMLs	<a href="#">Ctr</a> -Setting <b>UC=&lt;#&gt;</b> has to be 0 and only one language is allowed for <b>Ctr</b> -Setting <b>LC=&lt;\$+...+\$&gt;</b> and <b>LV=&lt;\$+...+\$&gt;</b>

### 7.1.3 License file corrupted

If this message appears the license file could be found, but is not in the format which is expected by radCASE. If the message appears when starting the **Project Monitor** this most probably is caused by a write protection of the file *DEVELOP\visual.lic*. That file is generated by the model compiler during model compilation and contains the licensing information for the **Project Monitor** and this file must match the project.

## 7.2 Runtime errors

### 7.2.1 Runtime Error Messages

To debug errors in the application the runtime library throws error messages for many errors that can be detected. To reduce code size, the error handling has to be enabled with the [DEFINE RD\\_ERROR\\_CHECK](#) on the controller. In the simulation the error handling is always activated.

There are three different formats for an error message:

1. If the application is running in the simulation a dialog box will be displayed with the title "Runtime-Kernel Error". The dialog box contains a description of the error encountered. If the error is unknown (e.g. if the application code also uses that mechanism), the error code is displayed in the following format:  
*"Unknown Error: <Error-Code>h, Attr: <Attribute>"*

In the case *<Error-Code>* is the error code as a hexadecimal value with 8 digits and *<Attribute>* is a decimal containing further information.

- If the application is running on a controller with HMI (*Ctr-Setting EH=1*) the error is displayed on the target display in the following format:  
SysErr: 0x<Error-Code>  
Attrib: <Attribute>
- If the application runs on a controller without HMI (*Ctr-Setting EH=0*) the error message will create a blinking error code, which means the HAL-function `do_error()` (refer to Integration manual) is called, which should display the error using blinking LEDs.

The following Error-Codes exist:

Error Code	Description	Attribute
0x32	Surface-Interpreter was terminated. Probably closed too many surfaces ( <a href="#">CLOSE/RETURN (Sur)/Automatic Abort Key</a> )	-
0x33	Start <a href="#">MODUL</a> not found	-
0x34	Current surface not found	-
0x35	Current surface is empty	-
0x1001	Unknown Userfunction	Number of Userfunction
0x1002	Wrong (not existent) description-placeholder. Only <b>\$DS</b> , <b>\$DM</b> , <b>\$DE</b> , <b>\$VE</b> and <b>\$UL</b> are allowed (refer to <a href="#">Placeholders</a> )	-
0x1004	Unknown action in surface	ID of unknown action
0x1007	Too many surfaces opened by <a href="#">CALLSURFACE</a> or <a href="#">OPEN</a>	-
0x1008	Too many <a href="#">FULL</a> s nested	-
0x100B	<code>CEBin_put_enablemask()</code> , <code>CEBin_set_enablebit()</code> , <code>CEBin_reset_enablebit()</code> is called for an <a href="#">EBIN-ELEMENT</a> , where <b>EV</b> is not set to 1 (refer to <a href="#">Enabling/Disabling Selections</a> )	-
0x1011	Stack size error	Location provided by function <code>RdStack-Check()</code>
0x2776	<b>MODUL</b> not found in <b>FULL</b>	-
0x2777	Surface not found in <b>FULL</b>	-
0x2778	Root-visualizer not found in <b>FULL</b>	-

Error Code	Description	Attribute
0x2779	<b>ELEMENT</b> not found for <b>Condition</b> in <a href="#">IF, ENDIF, ELSE</a> -clause of a <b>SURFACE_CTR</b>	-
0x277A	<b>ELEMENT</b> not found for <a href="#">ActionCond</a> of a <b>SURFACE_CTR</b>	-
0x277B	<b>MODUL</b> of <b>FULL</b> not found while parsing conditions	-
0x277C	Surface of <b>FULL</b> not found while parsing conditions	-
0x277D	Root-visualizer not found while parsing conditions	-
0x277F	<b>MODUL</b> table not found for <b>FULL</b>	-
0x2780	<b>MODUL</b> not found while drawing <b>FULL</b>	-
0x2781	Surface not found while drawing <b>FULL</b>	-
0x2782	Root-visualizer not found while drawing <b>FULL</b>	-
0x2783	<b>MODUL</b> not found while updating <b>FULL</b>	-
0x2784	Surface not found while updating <b>FULL</b>	-
0x2785	Root-visualizer not found while updating <b>FULL</b>	-
0x2786	<b>MODUL</b> not found while initializing <a href="#">MENU</a>	-
0x2787	Surface not found while initializing <b>MENU</b>	-
0x2788	Root-visualizer not found while initializing <b>MENU</b>	-
0x278A	<b>MODUL</b> not found while executing key	-
0x278B	Surface not found while executing key	-
0x278C	Root-visualizer not found while executing key	-
0x278D	<b>MODUL</b> not found while initializing <b>FULL</b>	-
0x278E	Surface not found while initializing <b>FULL</b>	-
0x278F	Root-visualizer not found while initializing <b>FULL</b>	-
0x2791	<b>MODUL</b> not found while entering new surface	-

Error Code	Description	Attribute
0x2793	<b>ELEMENT</b> not found while updating element visualizer	-
0x2794	<b>ELEMENT</b> not found while updating bar visualizer	-
0x2796	Condition overflow (Too much nesting of <b>IF</b> -Conditions)	Nesting depth
0x2799	<a href="#">PICT</a> not found while drawing <b>PICT</b>	-
0x279B	Unknown <a href="#">MElem</a> <b>Display type</b> .	ID of <b>Display type</b>
0x279C	Unknown object type.	ID of Object type
0x279D	Too many surfaces opened by <b>CALLSURFACE</b> or <b>OPEN</b>	Count of opened surfaces
0x27D9	Surface not found	Surface number
0x27DA	<b>ELEMENT</b> table not found.	Requested <b>ELEMENT</b> number
0x27E6	<b>ELEMENT</b> not found while getting text	-
0x27E7	<b>MODUL</b> not found while getting <b>MODUL</b> text	-
0x27E8	<b>MODUL</b> definition not found while getting <b>MODUL</b> text	-
0x283D	Calibration data not found	-
0x283E	Calibration data not found	-
0x283F	Calibration data not found	-
0x2840	Calibration data not found	-
0x2841	Calibration data not found	-
0x2842	Calibration data not found	-
0x2843	Calibration data not found	-
0x2844	Calibration data not found	-
0x2845	Analog input not found	-
0x2846	Analog input not found	-

Error Code	Description	Attribute
0x2847	Analog input not found	-
0x2848	Analog output not found	-
0x2849	Analog output not found	-
0x284A	Analog output not found	-
0x284B	Counter not found	-
0x284C	Counter not found	-
0x284D	Counter not found	-
0x284E	Calibration data not found	-
0x284F	Calibration data not found	-
0x2850	Calibration data not found	-
0x2851	Calibration data not found	-
0x2852	Analog input not found	-
0x2853	Analog output not found	-
0x28A1	<b>ELEMENT</b> not found while changing value in action	-
0x2970	Language <b>ELEMENT</b> is defined without <b>EV=1</b>	-
0x29CD	Access to <b>MODUL</b> table with wrong index	Index
0x29CE	Access to <b>ELEMENT</b> table with wrong index	Index
0x29D0	Object version does not support extended <a href="#">RECTs</a>	-

#### 7.2.1.1 Too many surfaces opened by CALLSURFACE or OPEN

When opening a surface using [OPEN](#) or [CALLSURFACE](#) the old surface is saved to a stack, to be able to return to that surface on [RETURN \(Sur\)/CLOSE](#). By standard this stack has a size of 11. Because the start surface is also opened, this means it is possible to have a nesting depth of 10 in the surface navigation.

The error message has two different causes:

1. The nesting depth is really too deep. In this case it is possible to change the maximum nesting depth for opening by setting the Define **RD\_SURSTACKSIZE** to the required nesting depth.
2. Somewhere in the design a surface is opened with **OPEN** or **CALLSURFACE**, but is not closed and the navigation goes back with a [GOTOSURFACE](#) instead. And if that navigation path is used often enough the stack will overflow.

The second cause can be a really hard to find error in larger projects. To find this error it can be useful to visualize the current depth within the stack, to try different navigation paths and watch if the depth is returning to the correct value.

The current stack depth is saved in the global variable `char G_ActSurStack`. For more information on how to visualize this variable refer to [Visualizing global C-Variables](#).

### 7.2.2 Buffer too small – Enlarge RD\_DISPCMD\_BUFSIZE

This error message will be displayed in command based [Display communication](#) on the [DISPLAY](#) in case of a buffer overflow in the display command buffer on the target. This sort of display communication works in two different phases:

1. After changing or redrawing a surface only static content (not depending on element values) is buffered in the command buffer.
2. After retrieving the data from phase 1 a partial redraw is triggered, resulting in redrawing the dynamic content and the buffer will accept any drawing commands.

The error message can occur in both of these phases and is normally triggered out of 2 reasons:

1. The buffer is too small to either hold the complete static content of a surface in phase 1 or there is a burst of drawing commands in phase 2 which the buffer is unable to hold. In this case the buffer has to be enlarged by enlarging the Define **RD\_DISPCMD\_BUFSIZE**.
2. Content on the surface is drawn faster, than the visualization retrieves the data. In most cases this is caused by a method in an [APERM](#) which draws on the surface on every run of the surface interpreter.

Another possible cause of this problem could be the connection or connection settings being not fast enough. On every cycle of display communication the visualization will get a chunk of data from the command buffer. If those chunks are smaller than the data put into the buffer between two chunks the buffer will overflow. The chunk size depends on the communication buffer size. The rate at which those chunks are communicated depends on the speed of the communication medium and a few different settings (refer to [Communication Sequence](#)).

## 7.3 Visualizing global C-Variables

Sometimes it can be useful to visualize different global C-Variables in the [Project Monitor](#) (e.g. while debugging the HAL). This can be easily done, by creating radCASE-[ELEMENT](#)s for storing the value of the global variable. It is best to force the datatype of the radCASE-**ELEMENT**, to match the C-datatype of the global variable (refer to [Forced Data Type](#)).

To copy the value of the global variable a [PROC](#) with **Processing type PERM** can be created, which copies the values from the global variable to the radCASE-**ELEMENT**:

```
$srcElement = globalVar;
```

After this the **ELEMENT** can simply be visualized on a **SURFACE\_VIS** (refer to [Element Visualization](#)).



## 8 Appendix

### 8.1 Key Values

#### 8.1.1 Most Commonly Used Keys

Key	Code	Key	Code	Key	Code	Key	Code
ESC	27	F1	315	UP	328	KEYABORT	999
CR	13	F2	316	DOWN	336	AKEY1	801
ENTER	13	F3	317	LEFT	331	AKEY2	802
BS	8	F4	318	RIGHT	333	AKEY3	803
TAB	9	F5	319	PGUP	329	AKEY4	804
SPACE	32	F6	320	PGDN	337	AKEY5	805
		F7	321	HOME	327		
WHEEL_CW	651	F8	322	END	335		
WHEEL_CCW	652	F9	323	INS	338		
WHEEL_FCW	653	F10	324	DEL	339		
WHEEL_FCCW	654	F11	389				
WHEEL_PRESS	655	F12	390				

Table 14, Most commonly used keys

#### 8.1.2 Other Keys

Key	Code	Key	Code	Key	Code
ALT_A	286	ALTC_A	512	CTRL_A	601
ALT_B	304	ALTC_B	513	CTRL_B	602
ALT_C	302	ALTC_C	514	CTRL_C	603
ALT_D	288	ALTC_D	515	CTRL_D	604

Key	Code	Key	Code	Key	Code
ALT_E	274	ALTC_E	516	CTRL_E	605
ALT_F	289	ALTC_F	517	CTRL_F	606
ALT_G	290	ALTC_G	518	CTRL_G	607
ALT_H	291	ALTC_H	519	CTRL_H	608
ALT_I	279	ALTC_I	520	CTRL_I	609
ALT_J	292	ALTC_J	521	CTRL_J	610
ALT_K	293	ALTC_K	522	CTRL_K	611
ALT_L	294	ALTC_L	523	CTRL_L	612
ALT_M	306	ALTC_M	524	CTRL_M	613
ALT_N	305	ALTC_N	525	CTRL_N	614
ALT_O	280	ALTC_O	526	CTRL_O	615
ALT_P	281	ALTC_P	527	CTRL_P	616
ALT_Q	272	ALTC_Q	528	CTRL_Q	617
ALT_R	275	ALTC_R	529	CTRL_R	618
ALT_S	287	ALTC_S	530	CTRL_S	619
ALT_T	276	ALTC_T	531	CTRL_T	620
ALT_U	278	ALTC_U	532	CTRL_U	621
ALT_V	303	ALTC_V	533	CTRL_V	622
ALT_W	373	ALTC_W	534	CTRL_W	623
ALT_X	301	ALTC_X	535	CTRL_X	624
ALT_Y	277	ALTC_Y	536	CTRL_Y	625
ALT_Z	300	ALTC_Z	537	CTRL_Z	626
SHIFT_F1	340	P_NULL	500	CTRL_P_NULL	510

Key	Code	Key	Code	Key	Code
SHIFT_F2	341	P_UP_LE	501	CTRL_P_UP_LE	511
SHIFT_F3	342	P_UP	502	CTRL_P_UP	512
SHIFT_F4	343	P_UP_RI	503	CTRL_P_UP_RI	513
SHIFT_F5	344	P_LE	504	CTRL_P_LE	514
SHIFT_F6	345	P_RI	505	CTRL_P_RI	515
SHIFT_F7	346	P_DN_LE	506	CTRL_P_DN_LE	516
SHIFT_F8	347	P_DN	507	CTRL_P_DN	517
SHIFT_F9	348	P_DN_RI	508	CTRL_P_DN_RI	518
SHIFT_F10	349	P_MIDDLE	509	CTRL_P_MIDDLE	519
CTRL_F1	631	P_LEFT	256	SH_LEFT	0x01
CTRL_F2	632	P_RIGHT	257	SH_RIGHT	0x02
CTRL_F3	633	P_MID	258	SH_CTRL	0x04
CTRL_F4	634	H_LEFT	259	SH_ALT	0x08
CTRL_F5	635	H_RIGHT	260	SH_SCROLL	0x10
CTRL_F6	636	H_MID	261	SH_NUM	0x20
CTRL_F7	637	R_LEFT	262	SH_CAPS	0x40
CTRL_F8	638	R_RIGHT	263	SH_INS	0x80
CTRL_F9	639	R_MID	264	CTRL_UP	580
CTRL_F10	640	DC_LEFT	265	CTRL_DOWN	581
CTRL_F11	641	DC_RIGHT	266	CTRL_LEFT	371
CTRL_F12	642	DC_MID	267	CTRL_RIGHT	372
STX	2			SHIFT_UP	590
ETX	3	CF_HOCH	1	SHIFT_DOWN	591

Key	Code	Key	Code	Key	Code
ENQ	5	CF_RUNTER	2	SHIFT_LEFT	592
LF	10	CF_RECHTS	3	SHIFT_RIGHT	593
FF	12	CF_LINKS	4	SHIFT_TAB	271
DLE	16	CF_ESC	5		
NAK	21	CF_TOGGLE	6	ALT	601
SUB	26	CF_SET	7	SHIFT	602
BACKSLASH	92	CF_RES	8	SHIFTLOCK	603
KEYOTHER	998	CF_HELP	9	NOKEY	-1

## 8.2 Element Attributes

For [ELEMENT](#) attributes accessible using \$# (refer to [Access To Elements Metadata \(\\$#\)](#)) refer to the according subchapter:

[ENUM](#) Refer to [ENUM Attributes](#)

[EBIN](#) Refer to [EBIN Attributes](#)

[ESTR](#) Refer to [ESTR Attributes](#)

[ETIM](#) Refer to [ETIM Attributes](#)

[EDAT](#) Refer to [EDAT Attributes](#)

### 8.2.1 ENUM Attributes

In the following table all items with a C-data type of **XSTR** are an index for getting the text (refer to [Text Access](#))

C-data type	Name Of Attribute	Comment
XSTR	name	Name (only if <a href="#">Ctr</a> -Setting <b>CE=3</b> )
XSTR	desc	Description
XSTR	info	Info text (only if <a href="#">Ctr</a> -Setting <b>HT=1</b> )

C-data type	Name Of Attribute	Comment
XSTR	unit	Physical Unit
long	setup	Standard value
long	minRange	Minimum value range
long	maxRange	Maximum value range
short	format	Format: 0x00ZD with: Z = Maximum number of characters when converted to a string (including '.' and '-') D = Number of decimals  Functions converting a value to a string will accept 0x100 ored to the format to enforce prezeros.
unsigned short	setupIndex	Index of <b>ELEMENT</b> in <code>elementtabelle</code> containing the standard value (0xFFFF if no <b>ELEMENT</b> was specified)
unsigned short	minRangeIndex	Index of <b>ELEMENT</b> in <code>elementtabelle</code> containing the minimum value range (0xFFFF if no <b>ELEMENT</b> was specified)
unsigned short	maxRangeIndex	Index of <b>ELEMENT</b> in <code>elementtabelle</code> containing the maximum value range (0xFFFF if no <b>ELEMENT</b> was specified)

### 8.2.2 EBIN Attributes

In the following table all items with a C-data type of **XSTR** are an index for getting the text (refer to [Text Access](#))

C-data type	Name Of Attribute	Comment
XSTR	name	Name (only if <a href="#">Ctr-Setting CE=3</a> )
XSTR	desc	Description
XSTR	info	Info text (only if <a href="#">Ctr-Setting HT=1</a> )
XSTR	selStr	Description of first selection. The descriptions of the following selections are on the following indices
XSTR	selDesc	Info text of first selection (only if <a href="#">Ctr-Setting HT=1</a> ). The info texts of the following selections are on the following indices
char	numSel	Number of <a href="#">EB ENTRY</a> s. Negative value for multiselective <a href="#">EBIN</a> s

C-data type	Name Of Attribute	Comment
unsigned char	setup	Standard value

### 8.2.3 ESTR Attributes

In the following table all items with a C-data type of **XSTR** are an index for getting the text (refer to [Text Access](#))

C-data type	Name Of Attribute	Comment
XSTR	name	Name (only if <a href="#">Ctr-Setting CE=3</a> )
XSTR	desc	Description
XSTR	info	Info text (only if <a href="#">Ctr-Setting HT=1</a> )
unsigned char	lenStr	String length
XSTR	setup	Standard value

### 8.2.4 ETIM Attributes

In the following table all items with a C-data type of **XSTR** are an index for getting the text (refer to [Text Access](#))

C-data type	Name Of Attribute	Comment
XSTR	name	Name (only if <a href="#">Ctr-Setting CE=3</a> )
XSTR	desc	Description
XSTR	info	Info text (only if <a href="#">Ctr-Setting HT=1</a> )
short	ttype	Format: 0: hh:mm 1: hh:mm:ss 2: hh:mm:ss.ms
XSTR	setup	Standard value

### 8.2.5 EDAT Attributes

In the following table all items with a C-data type of **XSTR** are an index for getting the text (refer to [Text Access](#))

C-data type	Name Of Attribute	Comment
XSTR	name	Name (only if <a href="#">Ctr</a> -Setting <b>CE=3</b> )
XSTR	desc	Description
XSTR	info	Info text (only if <a href="#">Ctr</a> -Setting <b>HT=1</b> )
short	dtype	Format: 0: DD.MM.YY 1: DD.MM.YYYY
XSTR	setup	Standard value

### 8.3 Formats Of Text Visualizers

The format of Text Visualizers (refer to [Text Visualizers](#)) differs between **SURFACE\_VIS** and **SURFACE\_CTR** even for [ELEM/EDIT](#)-visualizers with the same **Display type**. For the formats on **SURFACE\_VIS** refer to [Formats Of Text Visualizers On SURFACE\\_VIS](#), for the formats on **SURFACE\_CTR** refer to [Formats Of Text Visualizers on SURFACE\\_CTR](#).

#### 8.3.1 Formats Of Text Visualizers On SURFACE\_VIS

On **SURFACE\_VIS** the different [Text Visualizers](#) differ in size, and depending on [Desktop](#)-setting **AED=<0/1>** color and background color. Independent of this setting, if reaching the **Alarm (min/max)** value of an [ENUM](#) the background color of the value will turn red and the foreground color white. If **AED=<0/1>** is deactivated the colors specified in the visualizer are used. [Table 15](#) lists the different possible **Display types** for text visualizers and their format including colors for activated **AED=<0/1>**.

The font on **SURFACE\_VIS** for descriptions and unit is always the proportional windows font Arial and for Values the fixed font Courier New.

Display type	Size	Foreground color	Background color
VNText1	11	Black	White
VNText2	26	Black	White
VNText3	11	White	Black
VNText4	26	White	Black
VNText5	11	Black	White
VNText6	6	Black	White

Display type	Size	Foreground color	Background color
VNText8	11	Black	White
VNText10	13	Black	White
VNText12	15	Black	White
VNText16	19	Black	White
VNText24	22	Black	White
VNText32	32	Black	White
VNText8x16	19	Black	White
VNText8x20	20	Black	White
VNText80	26	Black	White
VNText1Y	11	Black	Yellow
VNText7p	11	Black	Yellow
VNText8p	15	Black	Yellow
VNText10p	19	Black	Yellow
VNText12p	19	Black	Yellow

Table 15, SURFACE\_VIS: Formats Of Text Visualizers

### 8.3.2 Formats Of Text Visualizers on SURFACE\_CTR

On **SURFACE\_CTR** the different [Text Visualizers](#) differ in size and font. In general every font can be used as a text visualizer on the **SURFACE\_CTR**. To do this the **Display type** of the [ELEM/EDIT](#) can be set to `VNText<fontname>` where `<fontname>` can be every font specification allowed on **SURFACE\_CTR** (refer to [Font Handling](#)).

Additional to these text visualizers there are some **Display types** used on **SURFACE\_VIS** which will map to a regular radCASE System Font (refer to [Regular radCASE System Fonts](#)). [Table 16](#) lists all of these additional **Display types** and the font they map to.

Display type	Font
VNText1	6x8
VNText2	6x8



Display type	Font
VNText3	6x8
VNText4	6x8
VNText5	6x8
VNText80	20x32

Table 16, SURFACE\_CTR: Formats Of Text Visualizers

## 8.4 HAL drawing wrapper functions

To fully support command based display communication (refer to [Display communication](#)) every custom drawing should use a wrapper function instead of the display drawing commands for HAL (refer to Integration manual). The wrapper will call the according HAL function after putting the command into the communication buffer.

The following table lists all the wrapper functions which should replace the according HAL function.

Wrapper function	HAL function
SETLAYER()	c_dis_layer()
c_cur_mode1()	c_cur_mode()
c_cur_clear1()	c_cur_clear()
c_dis_clr1()	c_dis_clr()
c_dis_char1()	c_dis_char()
c_dis_str1()	c_dis_str()
c_dis_char161()	c_dis_char16()
c_dis_str161()	c_dis_str16()
c_cur_movePos()	c_cur_move()
c_cur_setPos()	c_cur_set()
c_dis_clrrect1()	c_dis_clrrect()
c_dis_bitmap1()	c_dis_bitmap()
c_dis_rect1()	c_dis_rect()

Wrapper function	HAL function
c_dis_line1()	c_dis_line()
c_dis_circ1()	c_dis_circ()
c_dis_arc1()	c_dis_arc()
c_dis_bitmap1_rot()	c_dis_bitmap_rot()
c_dis_area_init1()	c_dis_area_init()
c_dis_save_area1()	c_dis_save_area()
c_dis_restore_area1()	c_dis_restore_area()