



# Tutorial

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	Abstract.....	4
3	Introduction .....	5
3.1	Lesson I01: radCASE Basics.....	5
3.2	Lesson I02: The process .....	6
3.3	Lesson I03: Preparations.....	11
4	Hands-On.....	12
4.1	Lesson H01: Elements.....	12
4.1.1	Goal .....	12
4.1.2	Content .....	12
4.1.3	Conclusion .....	21
4.2	Lesson H02: Functions .....	23
4.2.1	Goal .....	23
4.2.2	Content .....	23
4.2.3	Conclusion .....	26
4.3	Lesson H03: Datatypes / Error localization .....	28
4.3.1	Goal .....	28
4.3.2	Content .....	28
4.3.3	Conclusion .....	38
4.4	Lesson H04: Environment Simulation .....	39
4.4.1	Goal .....	39
4.4.2	Content .....	39
4.4.3	Conclusion .....	42
4.5	Lesson H05: State Machine.....	43
4.5.1	Goal .....	43
4.5.2	Content .....	43
4.5.3	Conclusion .....	51
4.6	Lesson H06: First Module / Reusability .....	53
4.6.1	Goal .....	53
4.6.2	Content .....	53
4.6.3	Conclusion .....	60
4.7	Lesson H07: Libraries.....	62
4.7.1	Goal .....	62
4.7.2	Content .....	62
4.7.3	Conclusion .....	70
4.8	Lesson H08: Target HMI.....	72
4.8.1	Goal .....	72
4.8.2	Content .....	72

4.8.3	Conclusion .....	84
4.9	Lesson H09: Descriptions / Placeholders.....	87
4.9.1	Goal .....	87
4.9.2	Content .....	87
4.9.3	Conclusion .....	95
4.10	Lesson H10: Visualization Surfaces .....	96
4.10.1	Goal .....	96
4.10.2	Content .....	96
4.10.3	Conclusion .....	104
4.11	Lesson H11: Visualization Surface Navigation .....	106
4.11.1	Goal .....	106
4.11.2	Content .....	106
4.11.3	Conclusion .....	113
4.12	Lesson H12: Entity Tab .....	115
4.12.1	Goal .....	115
4.12.2	Content .....	115
4.12.3	Conclusion .....	121
4.13	Lesson H13: Inheritance.....	122
4.13.1	Goal .....	122
4.13.2	Content .....	122
4.13.3	Conclusion .....	136

## 2 Abstract

In this tutorial you will learn the basic features of radCASE. First you will learn how to operate radCASE and will get to know the example project we want to program in this tutorial. After that we will introduce the features step by step and build the example project from scratch. For this we will first build a really simple version of the process and will refine that process step by step using the different features.

## 3 Introduction

### 3.1 Lesson I01: radCASE Basics

radCASE is a tool which allows model based development of software for controllers. radCASE consists of different components.

radEDIT is the first component you will see, after starting radCASE. This is the component which allows editing the model of your process and is the software you will mainly develop in.

radGEN can be started out of radEDIT and will translate the model into C-code that will later on run on the controller / the target hardware.

radMON is a monitoring software, which you can connect to the hardware and use to monitor the process. radMON has an integrated simulation, which uses the same C-code that is used on the controller. It is strongly recommended to use the simulation for development in radCASE, because the simulation makes debugging the process much easier and will save you time when creating software for a controller. The simulation is not a detour on the way to having the software run on any hardware but it is the shortest way.

Finally there is the HAL (hardware abstraction layer) this is the part of software that has to be written for any controller and is the interface between the generated code of radCASE and the hardware. This allows programming in radCASE mostly hardware independent and makes it easy to switch a project to another hardware, if the need arises.

The simulation is basically just another HAL. The usage of the simulation makes the development of your project faster for a few reasons:

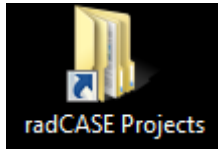
- You can already begin programming your software and testing your process before the HAL is developed. You can even do so, before even the controller is developed, giving you a head start in your project.
- Creating and starting the simulation is normally faster than creating the code for a controller and flashing the software onto it. So when repeatedly testing minor changes, this speeds up the debugging.
- The simulation allows you to also simulate the environment the controller works in, allowing you to test the process without the need to manually stimulate hardware IOs and without the danger of damaging hardware connected to the IOs, if something goes wrong in the process.
- If the need arises to debug within the generated source code the simulation lets you debug using Microsoft Visual Studio on the PC. This gives you a powerful debugging tool into your hands. In comparison debugging on the controller often needs special hardware and often is very slow and sometimes the debuggers don't even work correctly.

In almost all cases the software on the controller will not work, if the simulation does not work correctly. There are exceptions to this, but these are rare cases, where some target specific code is used. So in general it is best to use the simulation to develop the process. This is the reason this tutorial will only use the simulation to explain the usage of radCASE.

### 3.2 Lesson I02: The process

In this lesson you will learn to know the process we will implement in this tutorial. For this we will start by having a look onto the finished project.

Go to the directory the radCASE samples are installed, there should be a link to the radCASE Projects on your desktop:

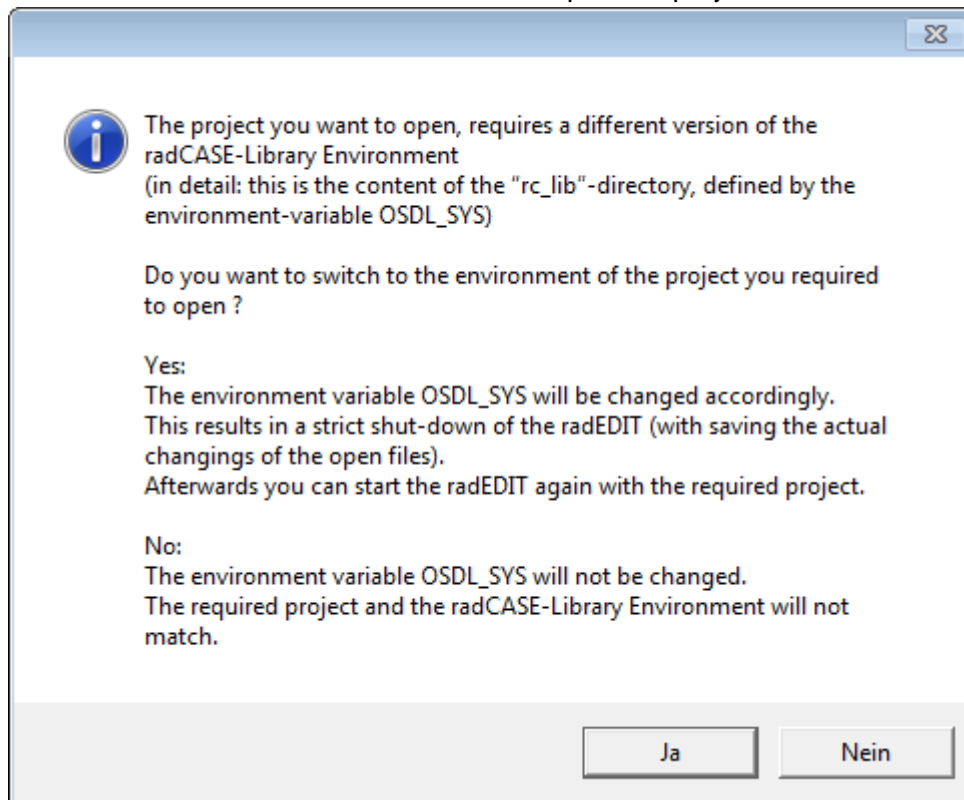


Open that link and go to the subdirectory

<radCASEProjects>\radCASESamples\tankControl\OSDL:

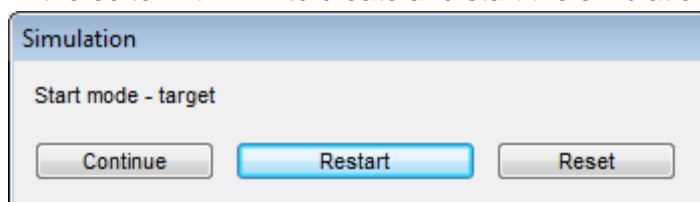
Name	Änderungsdatum	Typ	Größe
radCASEProjects			
APIs			
MyProjects			
radCASESamples			
Common			
EmBrick640			
Heat128			
Heat640			
LessonH02			
LessonH03			
LessonH04			
LessonH05			
LessonH06			
LessonH07			
LessonH08			
LessonH09			
LessonH10			
LessonH11			
LessonH12			
LessonH13			
Library			
Reference640			
TankControl			
CTR			
DEVELOP			
OSDL			
Lessons	14.02.2014 12:38	Dateiordner	
TankControl.rad	13.02.2014 16:32	radCase file	9 KB
TankLib.rad	13.02.2014 15:42	radCase file	11 KB

<Double click> on the TankControl.rad to open the project. You will most likely see this window:



Select “Yes” or the according equivalent according to the language of your operating system. After this radEDIT will close and you will have to open the project again. This time radEDIT will open with that project loaded. If a message appears with a notification about a test version, just confirm that message.

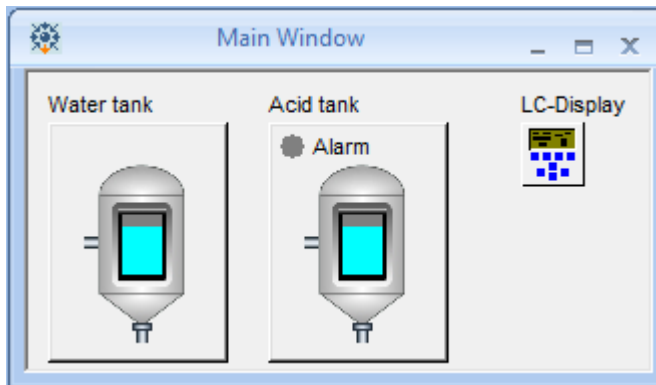
In the editor hit <F4> to create and start the simulation. The simulation will ask you for a start mode:



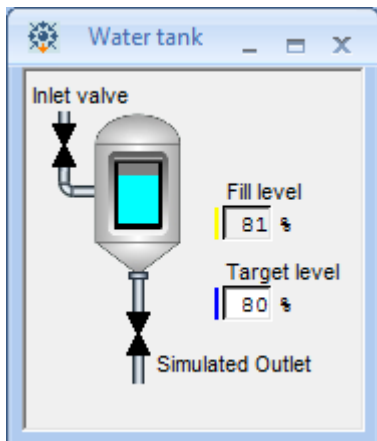
In this tutorial always use Restart or just press <Return>.



The simulation of the process will start. In the main window you will see two icons that symbolize two tanks:

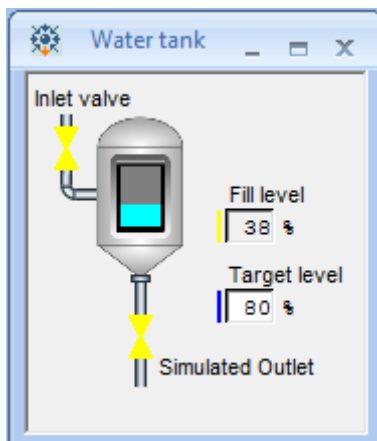


<Click> on the water tank and a second window will open:

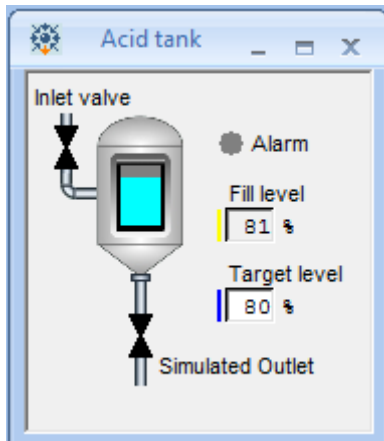


In this window you can see some details of the tank. You can see the current fill level, which is also visualized in graphical form on the tank symbol itself. You can also see a Target level, which you can edit by clicking on it. This is the level the process will try to hold the tank fill level above. To do this, the process will open the inlet valve, as soon as the fill level is below the target level for a predefined time (in this case 3 seconds).

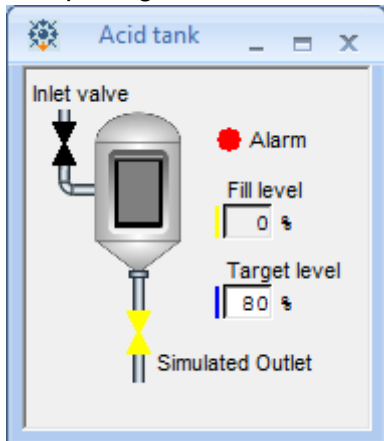
At last there is a simulated outlet. This simulates an outlet, which can be opened manually on the real tank. You can open that tank by clicking on the symbol of the simulated output. If you do this, you will notice, the Inlet valve will open 3 seconds after the fill level is below the target level. When you close the outlet by clicking on it again, the tank will fill again and stop, when reaching the target level.



Now <click> on the Acid tank in the main window:



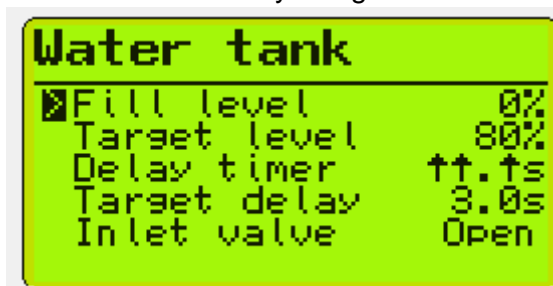
You can see this almost looks like the water tank. The process also is the same. The only difference is, there is an alarm LED which will light up, as soon as the tank is completely empty. Also the delay for opening the inlet valve is 15 seconds instead of 3 seconds.



At last <click> on the LC-Display icon in the main window:



A window will open, which simulates a keyboard and a display on a real hardware. You can use the keys on the simulated keyboard or just use the keyboard on your computer to navigate. You see there is a menu with an entry for each tank. You can select the tank using <Up>/<Down> and open the menu of a tank by using <Return>.

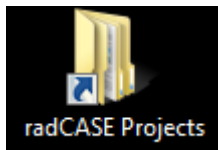





Within the menu of a tank, you can see the same values, you already saw in the windows of rad-MON, but you can also see some additional values. E.g. you can see the value Target Delay which sets the delay for opening the inlet valve. You can leave the Menu using <ESC> or by selecting the last menu item "Back".

This is the process we will implement within this tutorial. Please close the simulation and also radEDIT and continue with Lesson I03.

### 3.3 Lesson I03: Preparations

If you use the desktop symbol to go to the radCASE projects directory, you will find three directories in there:



 APIs	14.02.2014 12:38	Dateiordner
 MyProjects	14.02.2014 12:37	Dateiordner
 radCASESamples	14.02.2014 12:38	Dateiordner

The MyProjects directory is meant for you to work in. It is highly recommended to use that directory to put the projects of this tutorial into. In the directory radCASESamples you can find different samples. There you can also find different directories named LessonH<xx>. Those directories are different interim states of the tutorial. You can begin each Lesson of the tutorial by using the according project.

It is recommended to start each lesson with the according project from radCASESamples. Most of the lessons you can also use the project you created in the previous lesson, but the lessons will always use the filename of the project in radCASESamples for steps you have to know the filename or library name.

It is also recommended to copy the Lessons from radCASESamples to MyProjects and work with them in MyProjects, to have backups in the radCASESamples directory, if you want to repeat the tutorial or some of the lessons at a later point.



If you just copy the projects to the directory MyProjects, those projects will not work. You will also need the directories Common and rc\_lib, which are located in the radCASESamples directory. If you create a new project as described in lesson H01 radCASE will create all needed files in MyProjects.

If you open one of the Lesson<xx> directories, you will see some directories within:

 CTR	14.02.2014 12:38	Dateiordner
 DEVELOP	14.02.2014 12:38	Dateiordner
 OSDL	14.02.2014 12:38	Dateiordner

In the OSDL directory you will find a file named LessonH<xx>.rad. That is the project file, which you will have to open, to work with the project. If the later lessons speak of using the project LessonHxx which is delivered with radCASE, you will have to use the file:

LessonH<xx>\OSDL\LessonH<xx>.rad

When changing to another project it is best to close radEDIT and the old project at the moment. When opening different projects within radEDIT, you can end up using the wrong project and might get confused, because you can't see the expected behavior.

## 4 Hands-On

### 4.1 Lesson H01: Elements

#### 4.1.1 Goal

In this lesson you will learn how to begin a new project and how to work with elements in radCASE.

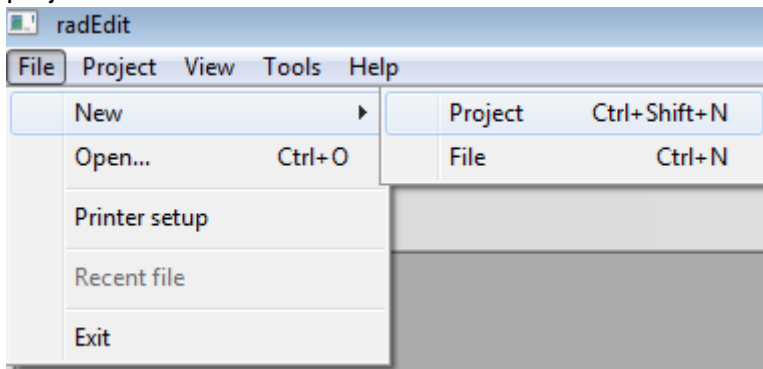
#### 4.1.2 Content

First we have to start radCASE by double clicking the desktop icon:

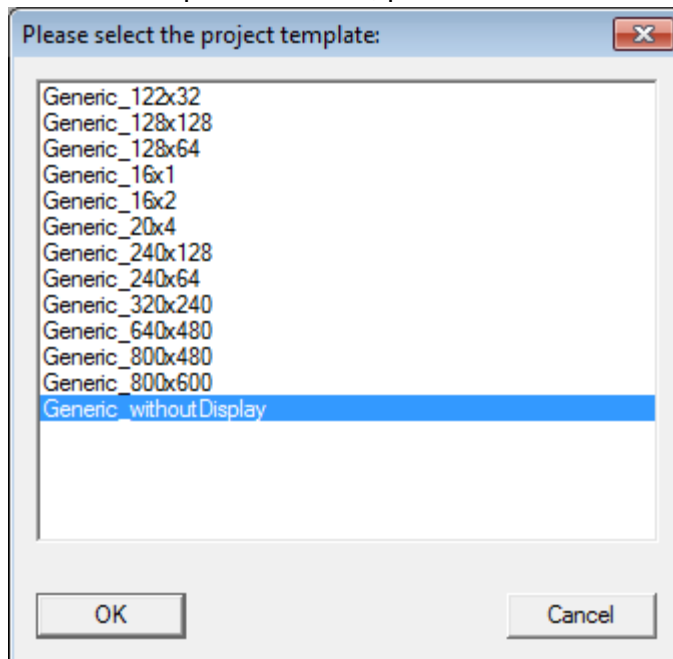


We will first start by creating a new project. Even though the endproduct will have an HMI, we will start with a project without any HMI and add the HMI at a later point.

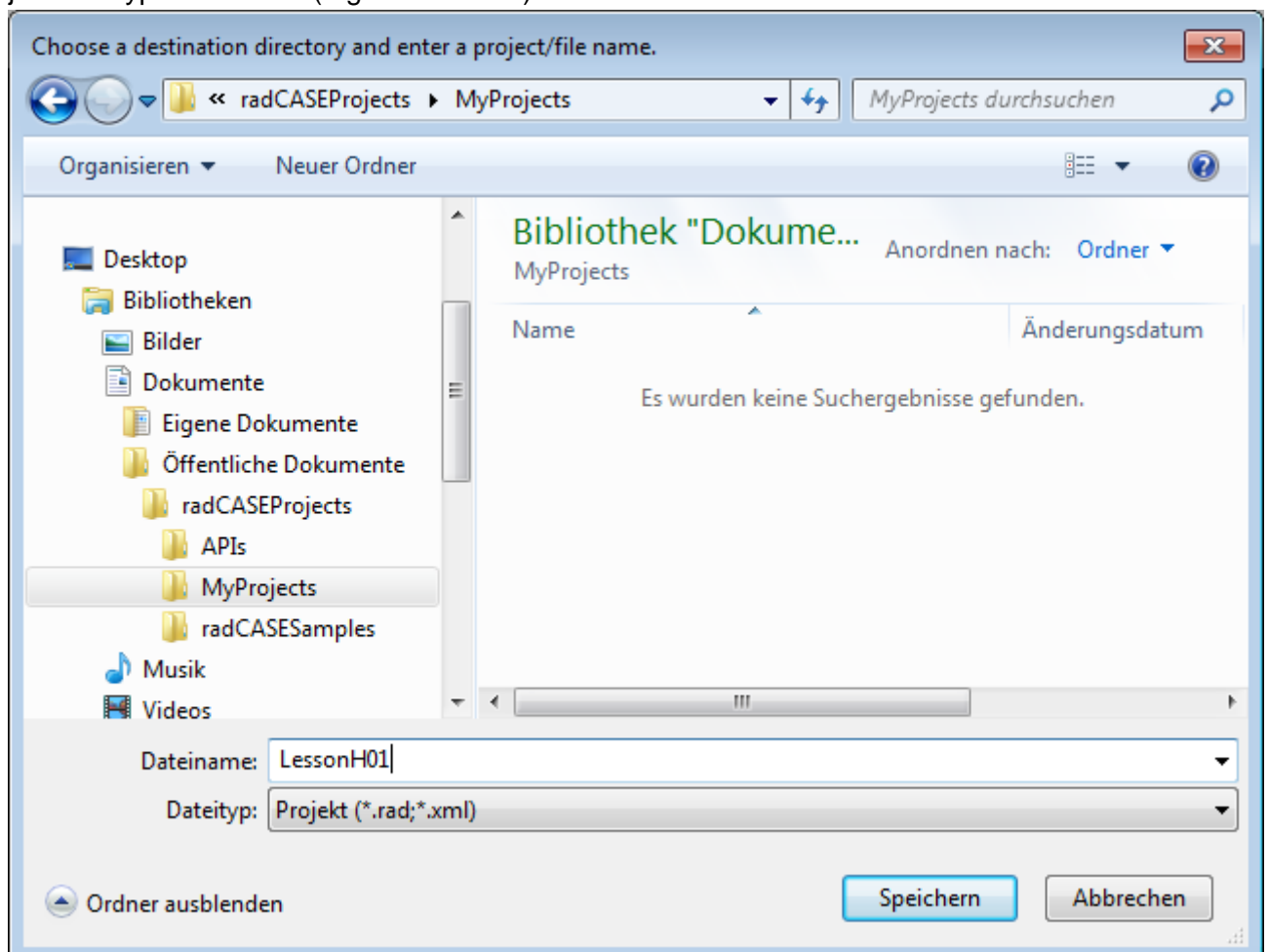
To start the new project, first start radCASE and use *<Ctrl + Shift + N>* or the menu to select a new project:



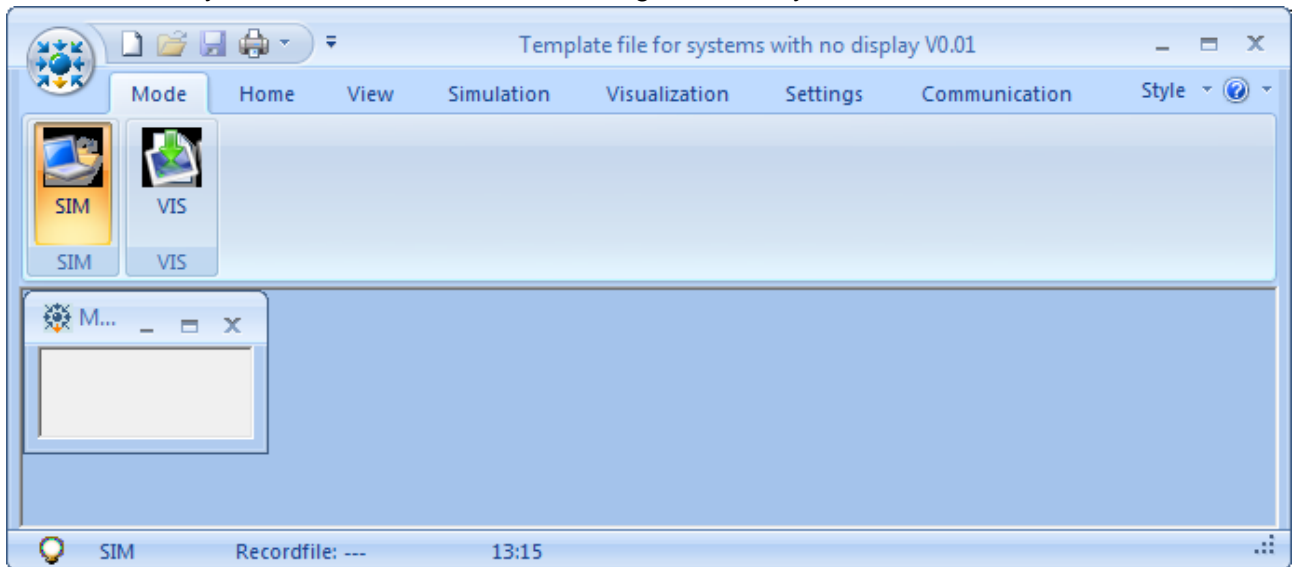
From the template selection, please select “**Generic\_withoutDisplay**”:



Select a folder (<radCASEProjects>\Myprojects recommended) where you want to create the project and type in a name (e.g. LessonH01)



The new project will be created and is already able to run. Just use <F4> to create and start the simulation and you will see the simulation running without any content:



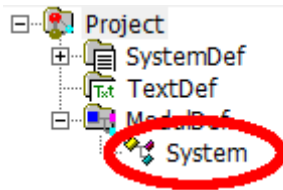
To begin modeling our process we first need some variables. In radCASE variables are called **ELEMENTS**. An **ELEMENT** in radCASE has different additional informations, which are missing in a classic variable. This includes the ranges the value is allowed to have, floating point handling and different special behavior like mapping to hardware IOs or storage in a permanent memory.

For a first raw process our project will need different **ELEMENTS**:

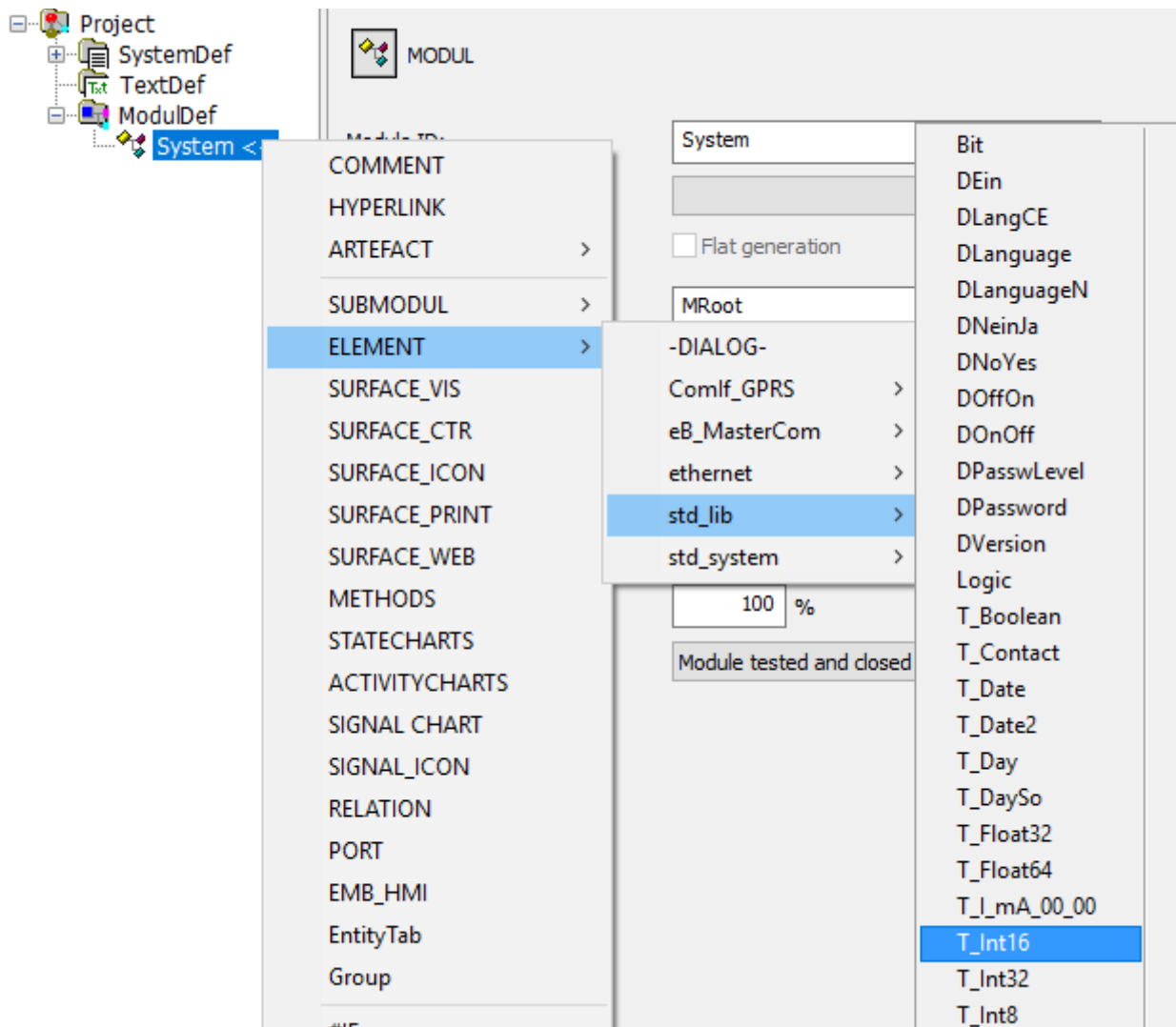
- The current fill level. This needs to be mapped to a hardware AI
- The state of the inlet. This will be mapped to a hardware DO which is connected with a valve
- A timer to get the delay for opening the valve
- The parameter which specifies which level the tank should have. This parameter should be saved permanently, meaning the saved value should persist even when the program is re-started.
- The parameter which specifies the length of the delay (also saved permanently)

We will now create three of the **ELEMENTS** in our project: the current fill level, the timer and the target fill level.

Open the ModulDef in your project by clicking on the “+” and you will see “System”:

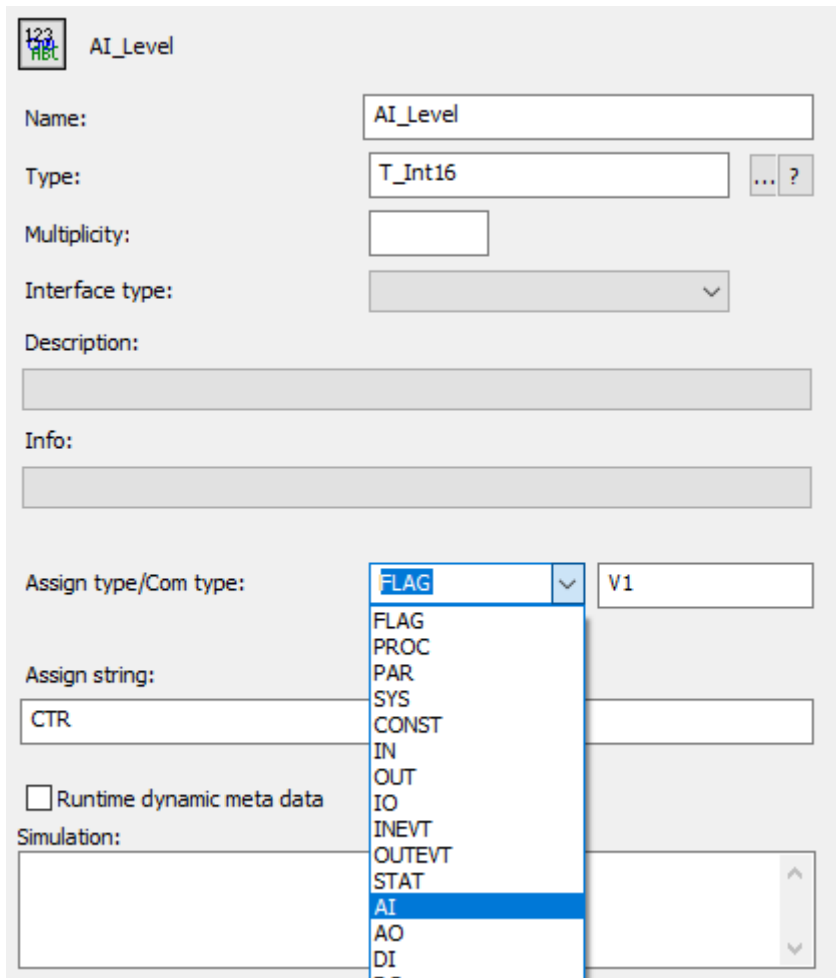


System is a module, which will be explained later in the tutorial. The module System however is a special module, because it is the entry point of a radCASE application, it is comparable to the main-function of a C-program. <Right click> on the system module and select **ELEMENT**. This again will open another context menu with the different libraries to select predefined data types. For now we will select the data type T\_Int16 from the std\_lib. We could also create our own data types but we will see that in a later lesson.



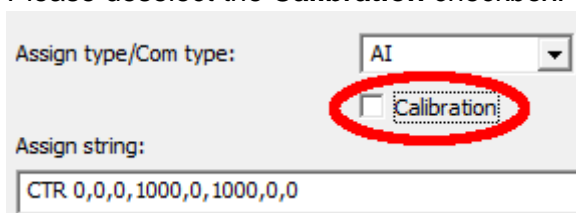
In the property editor in the middle pane of radEDIT we will set the **Name** to AI\_Level. The new **ELEMENT** is created with an **Assign type** of **FLAG**. This is the equivalent to a normal variable. But because we need it mapped to a hardware analogue input, we change the Assign type to AI.





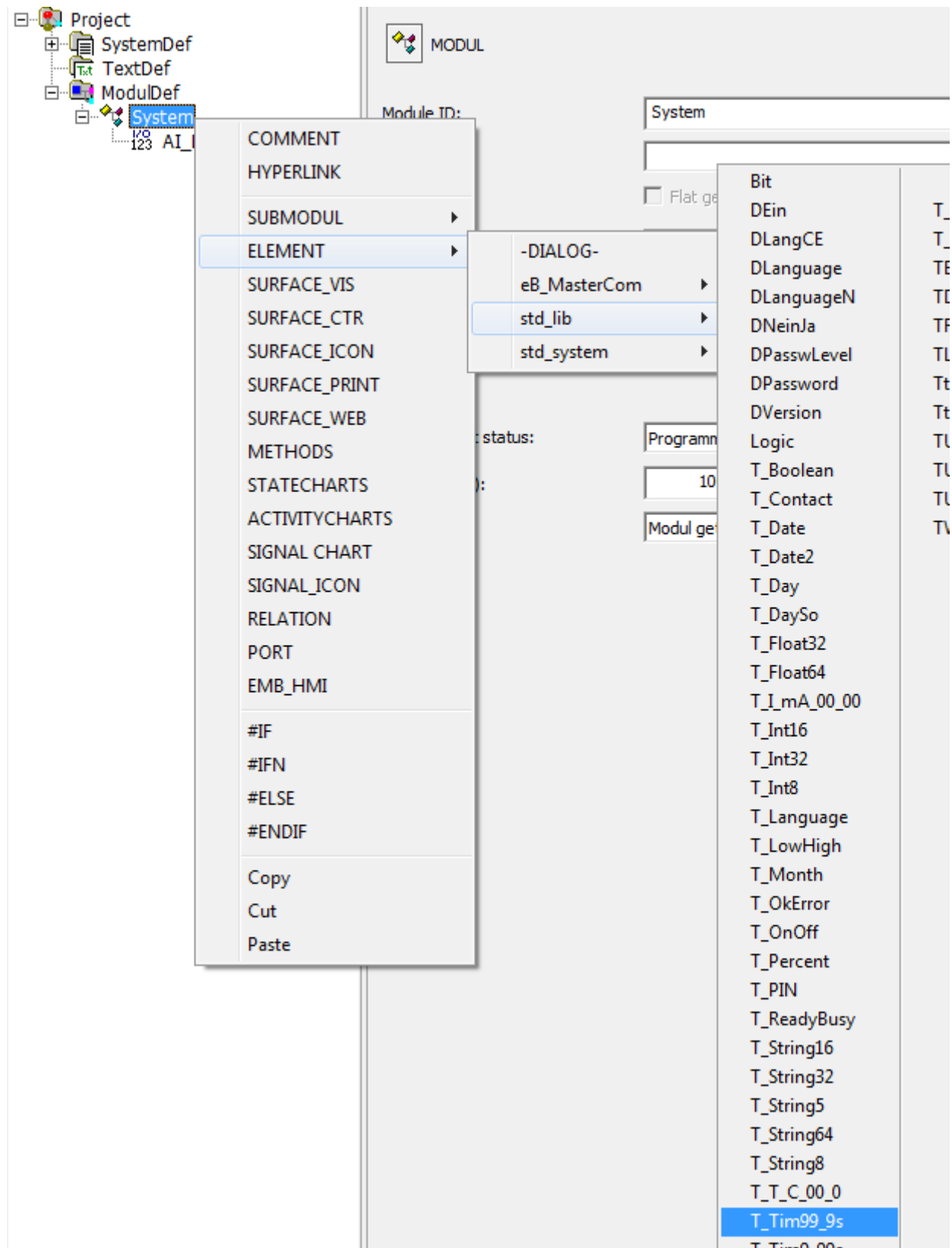
You probably will notice the **Com type** and the **Assign string** changes. These change to good default values and you don't need to bother with those values. The assign type **AI** will do everything necessary to map the element values to the HAL, so the HAL can manage all steps necessary to really read out the AI on the hardware.

Please deselect the **Calibration** checkbox:

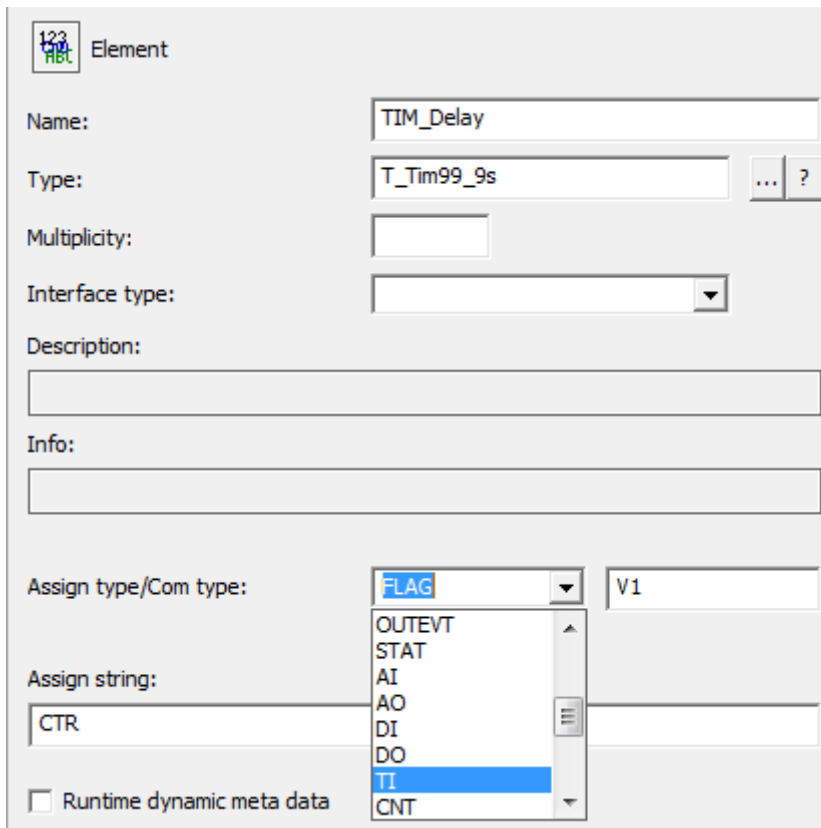


This deactivates a feature which is not supported by our project at the moment and because of that would cause an error. The rest of the properties of this element can be left on the default values.

With another <right click> on the System module, we now create another **ELEMENT** for the timer. This time we will select the type T\_Tim99\_9s from std\_lib:

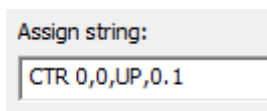


We will give the **ELEMENT** the **Name** TIM\_Delay and an **Assign type** of **TI**. This assign type automatically adds timer functionality to the **ELEMENT**, so the **ELEMENT** will already start to measure time. The details are again managed by the HAL.



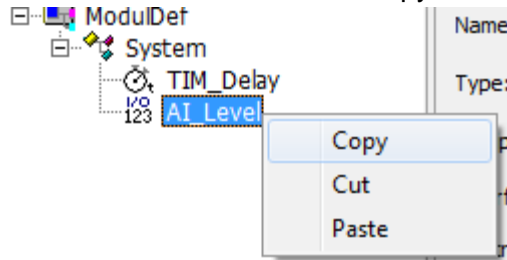
The screenshot shows the HAL configuration window for an Element. The 'Name' field is set to 'TIM\_Delay'. The 'Type' field is set to 'T\_Tim99\_9s'. The 'Multiplicity' field is empty. The 'Interface type' field is a dropdown menu. The 'Description' field is empty. The 'Info' field is empty. The 'Assign type/Com type' dropdown menu is open, showing options: FLAG, OUTEVT, STAT, AI, AO, DI, DO, TI (selected), and CNT. The 'Assign string' field is set to 'CTR'. The 'Runtime dynamic meta data' checkbox is unchecked.

Again the **Com Type** and **Assign string** change to default values. This time we want to change the last value in the **Assign string** from 1 to 0.1 to count in 0.1 second steps.

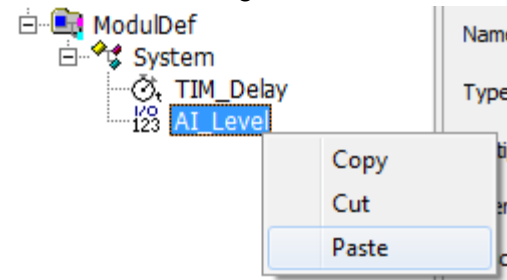


The screenshot shows the 'Assign string' field with the text 'CTR 0,0,UP,0.1'.

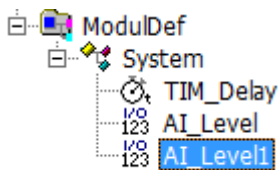
As a last element we will create the target fill level. Because this element will have the same data type like the element AI\_Level, we will make a copy of AI\_Level and modify it. <Right click> on the element AI\_Level and select Copy:



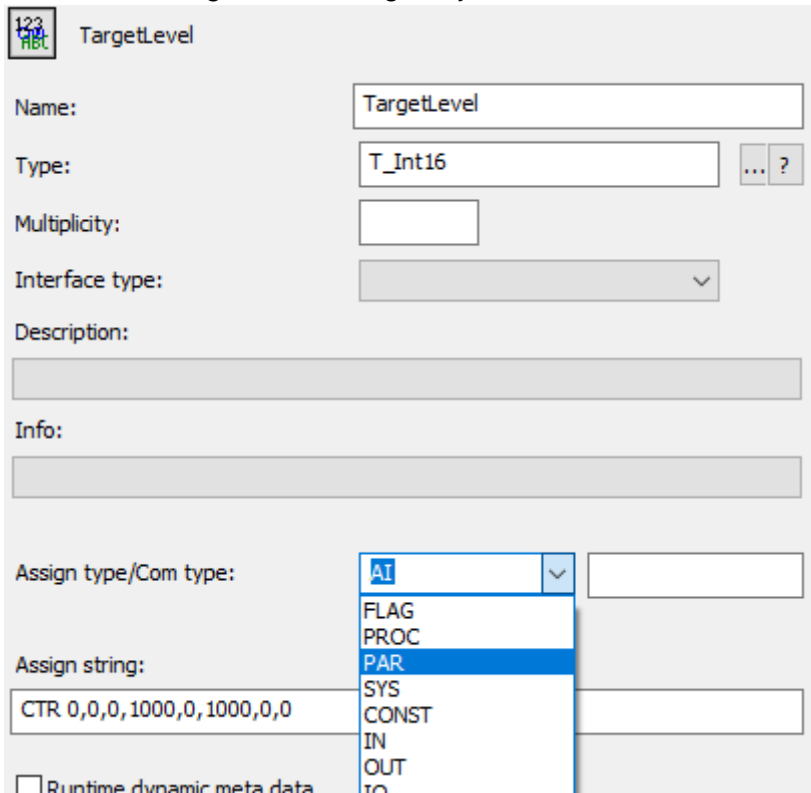
You could also select the element and use <Ctrl + C> to copy the element. Now <right click> on the element AI\_Level again and select Paste (or select it and use <Ctrl + V>):



You will see there is a new element AI\_Level1 in the project tree. Select this new element:



Now we will modify the properties of the element. We will change the name of the element from AI\_Level1 to TargetLevel. We will also change the Assign type from **AI** to **PAR**. This will turn the element to a permanently stored element. The data handling for storing the value e.g. in an EEPROM will again be managed by the HAL:



TargetLevel

Name: TargetLevel

Type: T\_Int16

Multiplicity:

Interface type:

Description:

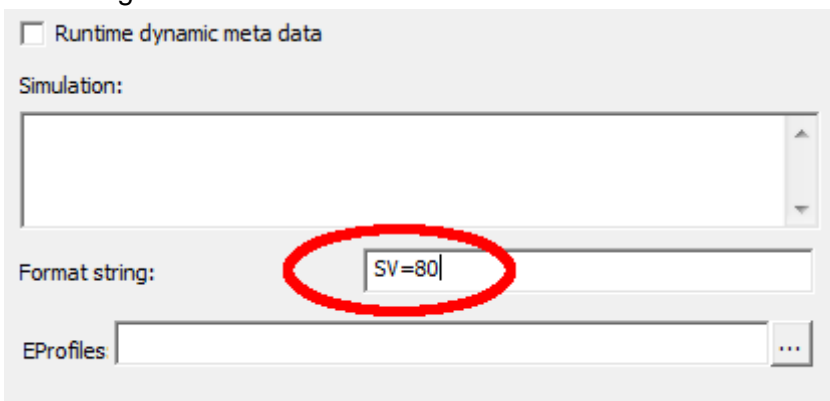
Info:

Assign type/Com type: AI

Assign string: CTR 0,0,0,1000,0,1000,0,0

☐ Runtime dynamic meta data

Again the Assign string will change to a good default value. Now we will enter SV=80 into the Format string:



☐ Runtime dynamic meta data

Simulation:

Format string: SV=80

EProfiles

This will set the standard value of the element to 80. Currently the data type is a generic type. This will later be changed to a data type more suited for the purpose of the **ELEMENT**.

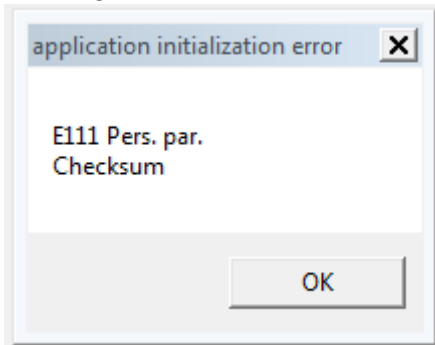
The rest of the needed **ELEMENT**s will be added in the delivered project for Lesson H02, because this is nearly the same step for every **ELEMENT** again.

The DO for handling the Valve will get an assign type of DO, which again lets the DO handling be done by the HAL. The data type will be T\_OnOff from std\_lib.

The last parameter for the TargetDelay, will be set to a standard value of 3.0 seconds.

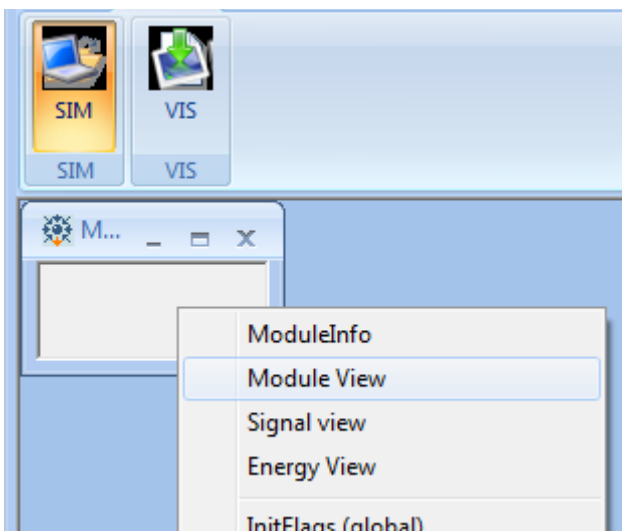
### 4.1.3 Conclusion

Hit <F4> in your project to create and start the simulation. You will probably get the following error message:

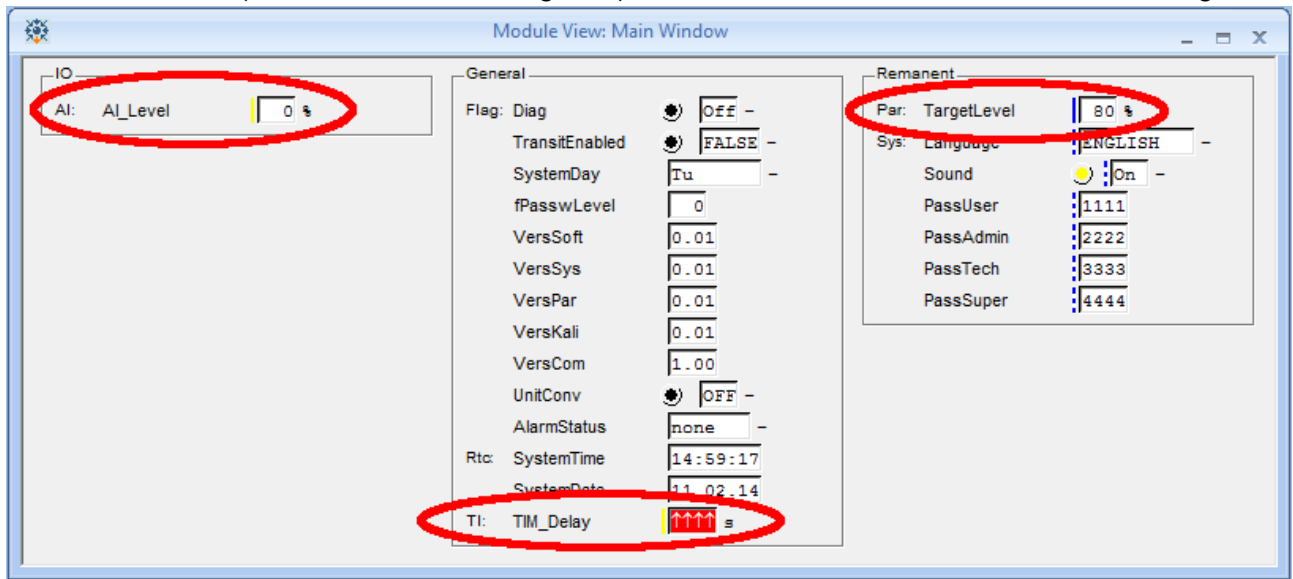


This error message occurs, because we have added an element with assign type PAR and so the old state of the permanently stored variables did not match the current element structure anymore.

On first look you will not see any difference to the empty project. <Right click> on the empty main window and select Module View:



This will open a window which is automatically created by radCASE, where all **ELEMENTS** of a module are listed (sorted in different categories) and where the value can be seen and changed.



You will notice there are multiple elements in the System module already predefined by radCASE. You will learn where those elements come from in a later lesson. However you can also find your three elements. The Timer will already count without any user interaction. If the timer reaches a value outside a predefined range the value will look like in the picture above. If this happens, you can *<click>* on the value of the timer and enter 0 into the edit dialog and after that you can see the timer count again.

## 4.2 Lesson H02: Functions

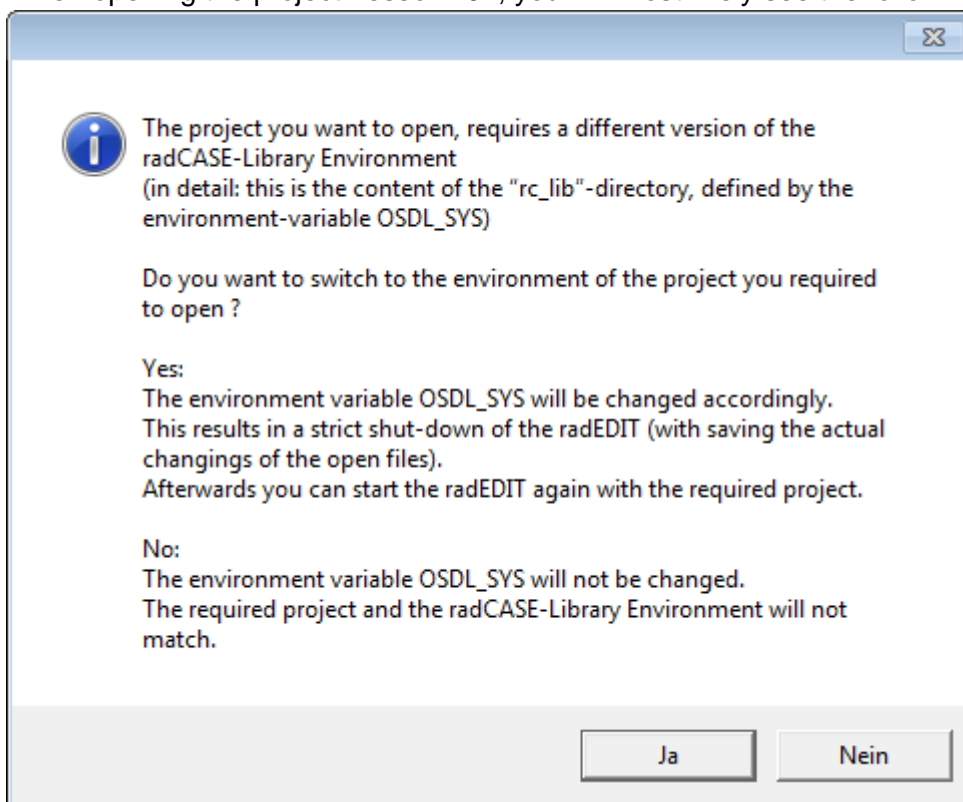
### 4.2.1 Goal

In this lesson you will learn your first way to add behavior to your project, by using a C-function.

### 4.2.2 Content

You should start this lesson with the project LessonH02 which is delivered with your radCASE copy. In this project the missing **ELEMENT**s from last lesson are already added and can be used. The project comment was also modified for that project.

When opening the project LessonH02, you will most likely see the following window:

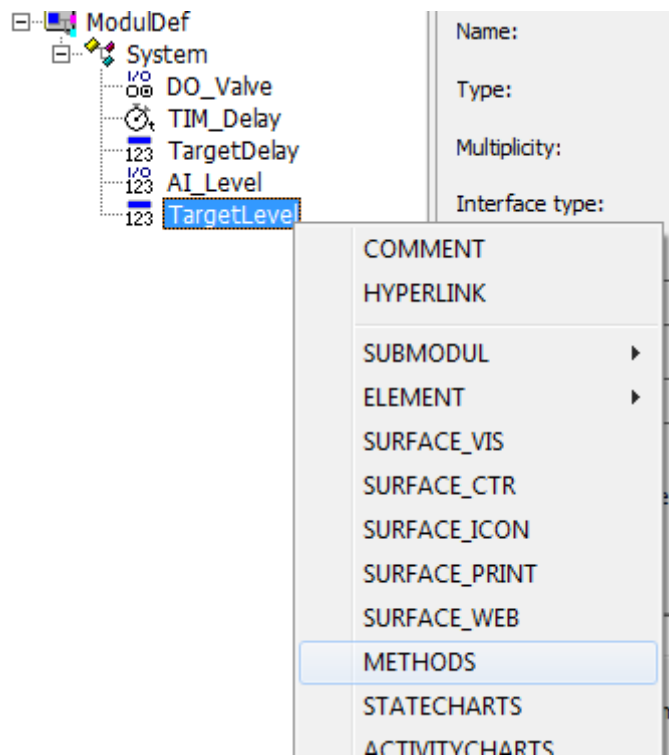


Select “Yes” or the according equivalent according to the language of your operating system. After this radEDIT will close and you will have to open the project again.

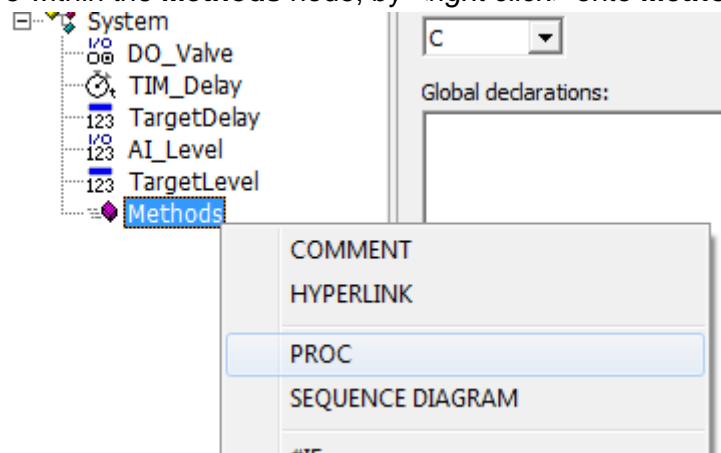
Now we want to define the behavior of our tank. In the first step we want to achieve the Valve opening after the specified delay, if the tank level is below the desired tank level. The Valve should close again, if the tank is at least at the desired tank level. We will do this by writing a small C-function.

Use <Shift + right click> on the last **ELEMENT** in the System module and select **METHODS**. The <Shift> lets you insert items below the currently selected radCASE-Item. Without the <Shift> the item is inserted into the selected item.

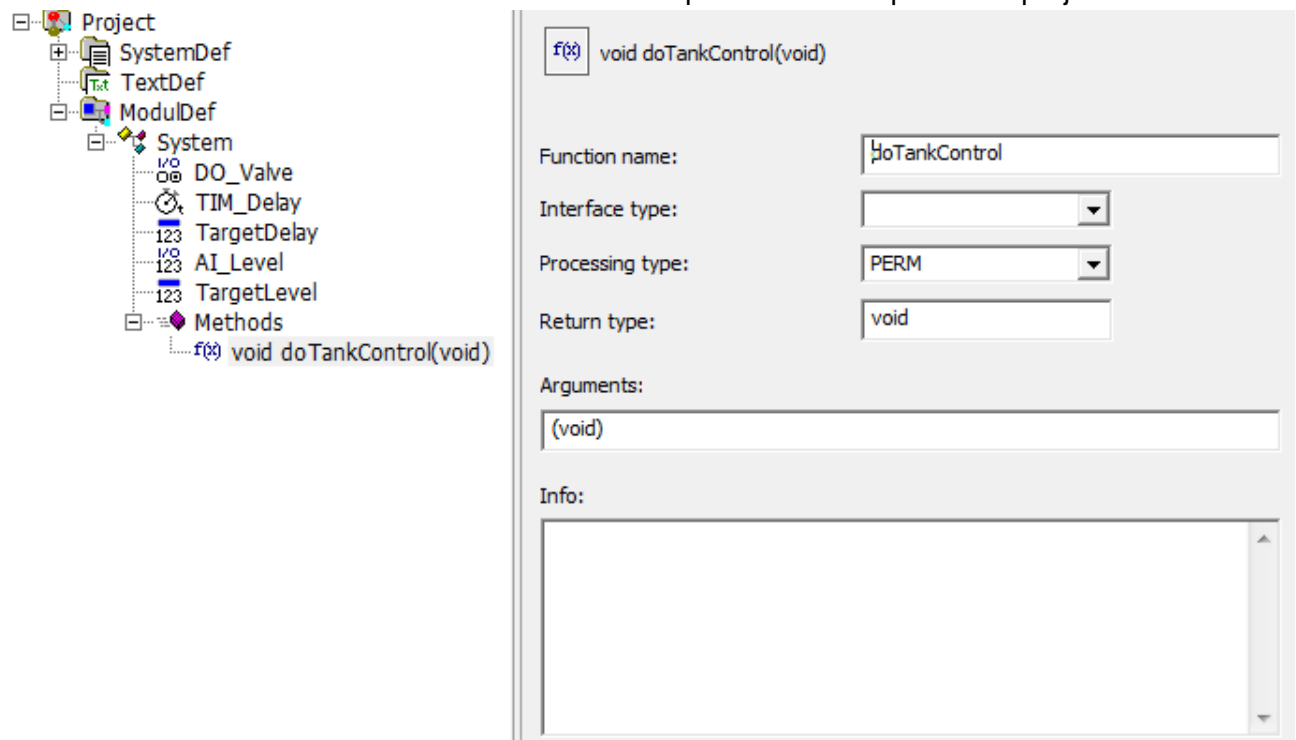




The **Methods** node is created with already correct default settings. Now we want to add a procedure within the **Methods** node, by *<right click>* onto **Methods** and selecting **PROC**.



We set the **Function name** to “doTankControl” and press <F5> to update the project tree:



The **return type** and the **arguments** are already on a good default value for our function. The **Processing type** specifies the function to be called permanently. This means the function is called in regular intervals.

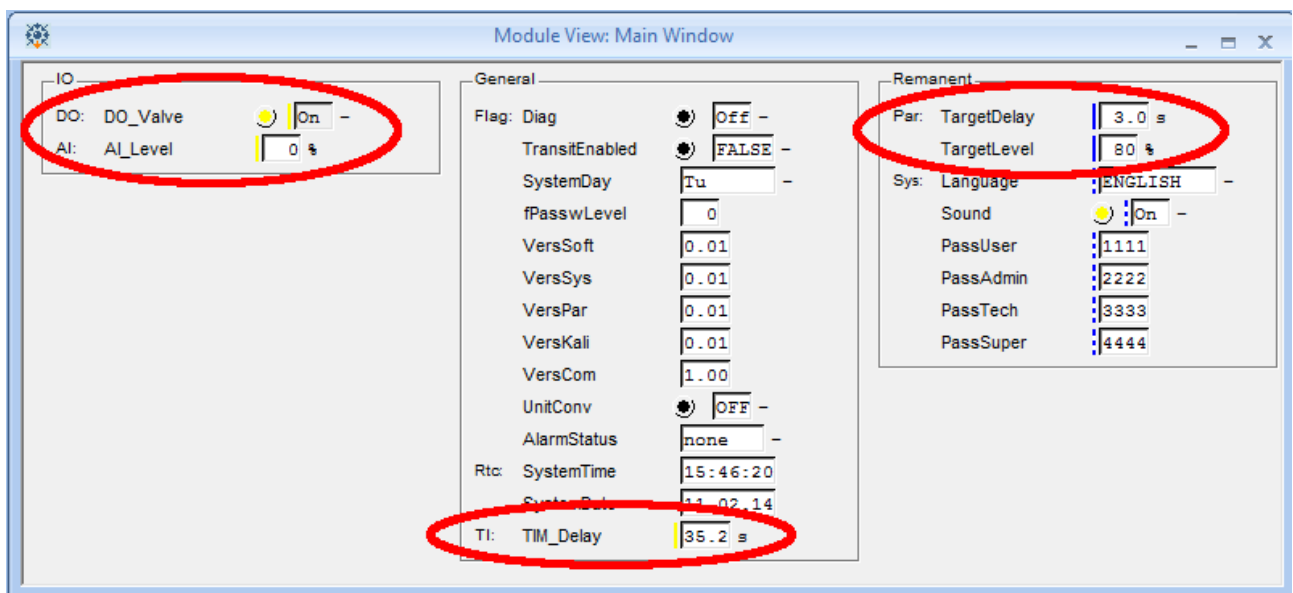
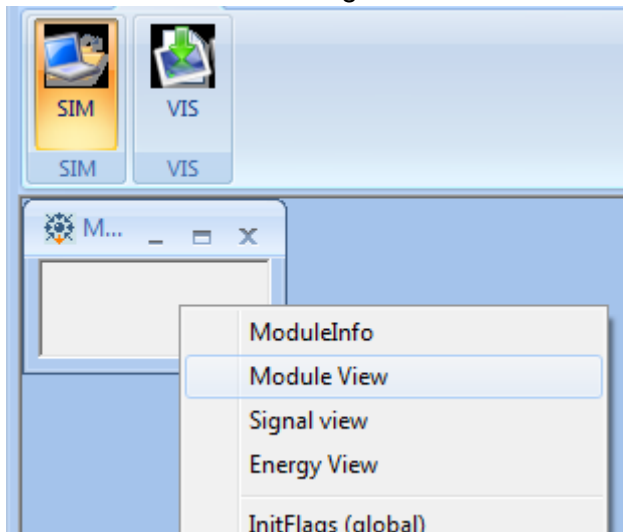
Now we can add the function in the editor in the right part of radEDIT. You can access the **ELEMENTS** in your module simply by adding a \$ at the start of the **Element name** in your C-code. You can also just use <F9> to select an **ELEMENT** and it is automatically added into your C-Code. DO\_Valve is a binary value with predefined values. Those values can be selected with <Shift + F9> when the cursor is behind the **ELEMENT**. So you can just enter \$DO\_Valve = <Shift + F9> and can select a value. The value will start with a \$ and can also be entered manually when the valid values are known.

```
{
    if ($AI_Level < $TargetLevel)
    {
        if ($TIM_Delay >= $TargetDelay)
        {
            $DO_Valve = $On;
        }
    }
    else
    {
        $TIM_Delay = 0;
        $DO_Valve = $Off;
    }
}
```

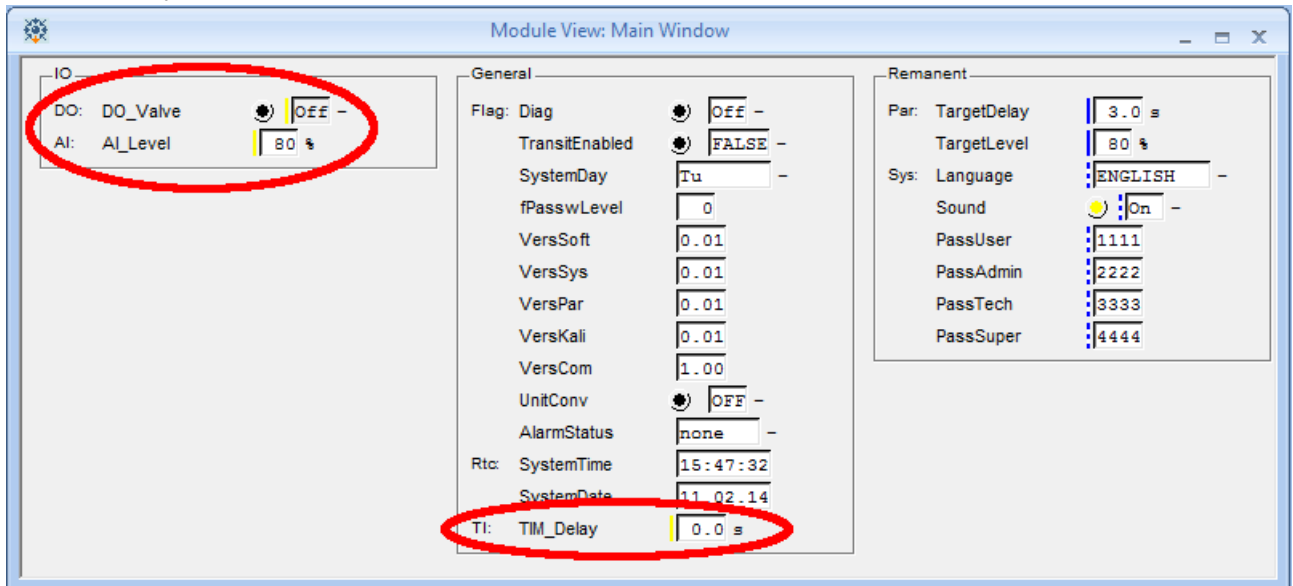
The \$ and § signs specify the values to be defined within radCASE. If those signs are missing radCASE will assume the value to be a global C-Variable or function.

### 4.2.3 Conclusion

Hit <F4> to create and start the simulation. Again open the module view using a <right click> on the main window and selecting **Module View**:



Now if you set AI\_Level to a value of 80% or higher you will note, the DO\_Valve will be set to “Off” automatically and the Timer will be set to 0:



If you now set AI\_Level below the TargetLevel the timer will start counting up and as soon as it reaches the TargetDelay, the Valve will open again.

As you can see our process already is working at this point. But you have to set everything manually in the **module view**.

## 4.3 Lesson H03: Datatypes / Error localization

### 4.3.1 Goal

In this lesson you will learn how to create your own data types and how to use the error localization to find error sources.

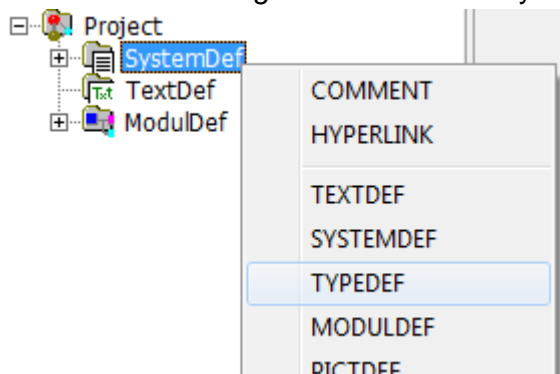
### 4.3.2 Content

You can start this lesson by using the project you created in lesson H02 or by opening the project LessonH03 that is delivered with your radCASE copy.

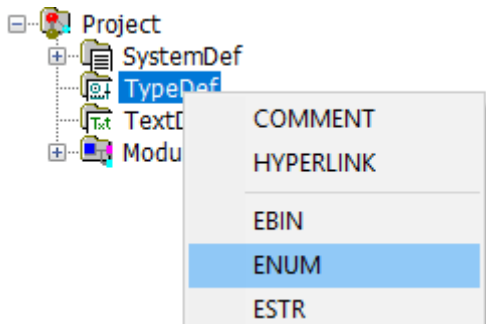
At the moment we use a generic type for our tank levels. Instead we want to use a percentage value and limit the range accordingly. Also we close our valve by assigning them the values Off and On. Because this does not correctly describe the behavior of a valve, we want to use the values Open and Closed instead.

To achieve this, we need to create new data types.

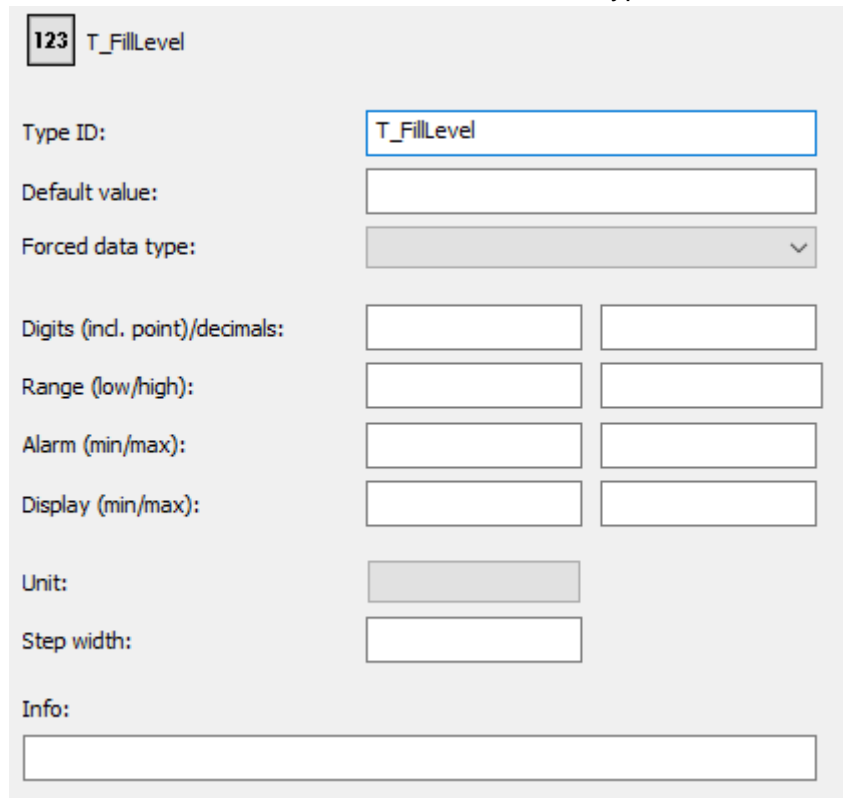
First use **<Shift + right click>** onto the SystemDef node in your project and select **TYPEDEF**:



This creates a section where data types can be defined. Now **<right click>** on the new TypeDef node and select **ENUM**:



This creates a numerical data type. In the property editor give the new ENUM a Type ID of T\_FillLevel. This is the name of our new data type.



123 T\_FillLevel

Type ID:

Default value:

Forced data type:

Digits (incl. point)/decimals:

Range (low/high):

Alarm (min/max):

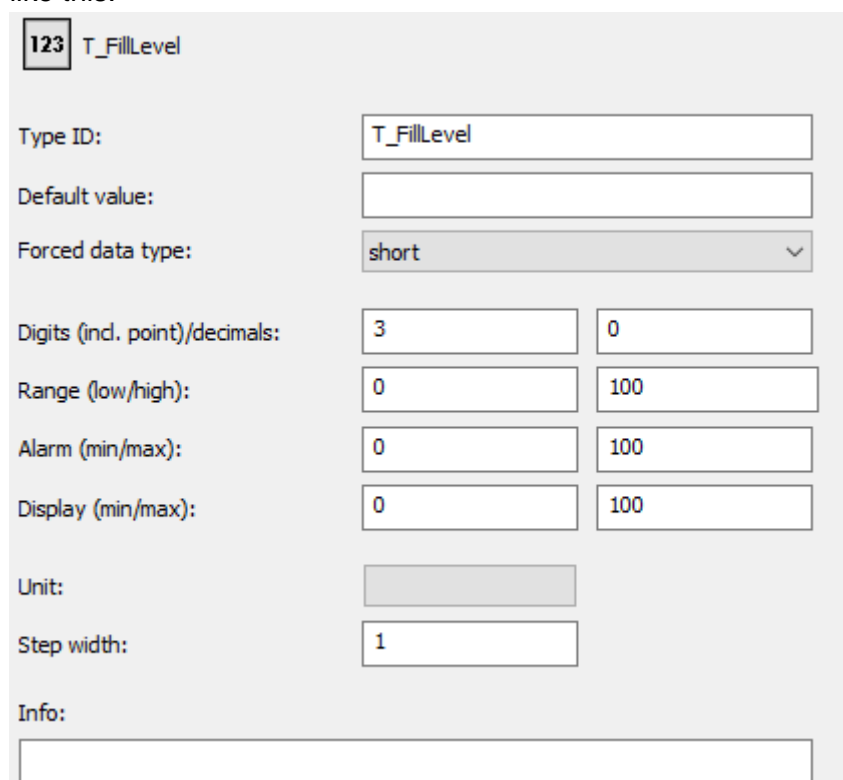
Display (min/max):

Unit:

Step width:

Info:

Set the numerical value to 3 digits and 0 decimals. Also set the lower and upper limits of Range, Alarm and Display to 0/100. And set the step width to 1. Also select a Forced data type of short (this is required, because analogue inputs internally need this data type: The data type should now look like this:



123 T\_FillLevel

Type ID:

Default value:

Forced data type:

Digits (incl. point)/decimals:

Range (low/high):

Alarm (min/max):

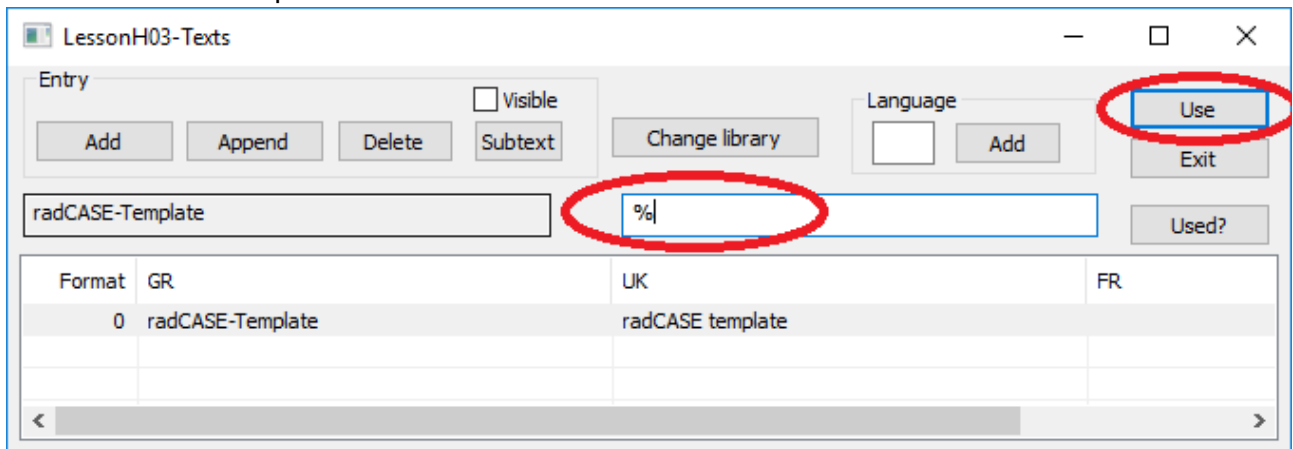
Display (min/max):

Unit:

Step width:

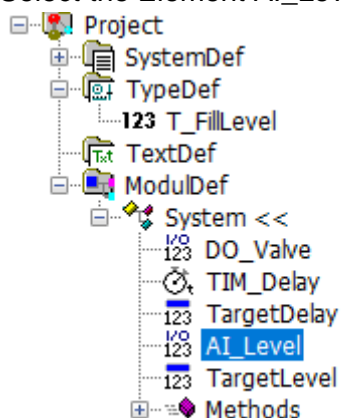
Info:

We also want a unit for our numerical value. So we will now add a single lingual unit (this will be explained in detail in a later lesson) to this ENUM. <Click> onto the rectangle next to Unit and enter “%” as unit and accept with Use.

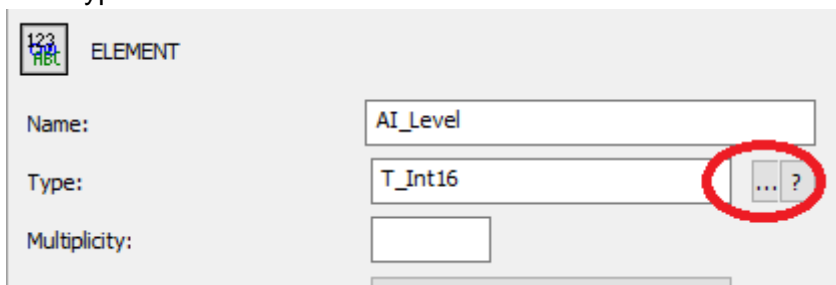


We have created our new data type. Will will change the ELEMENTs AI\_Level and TargetLevel to use the new data type:

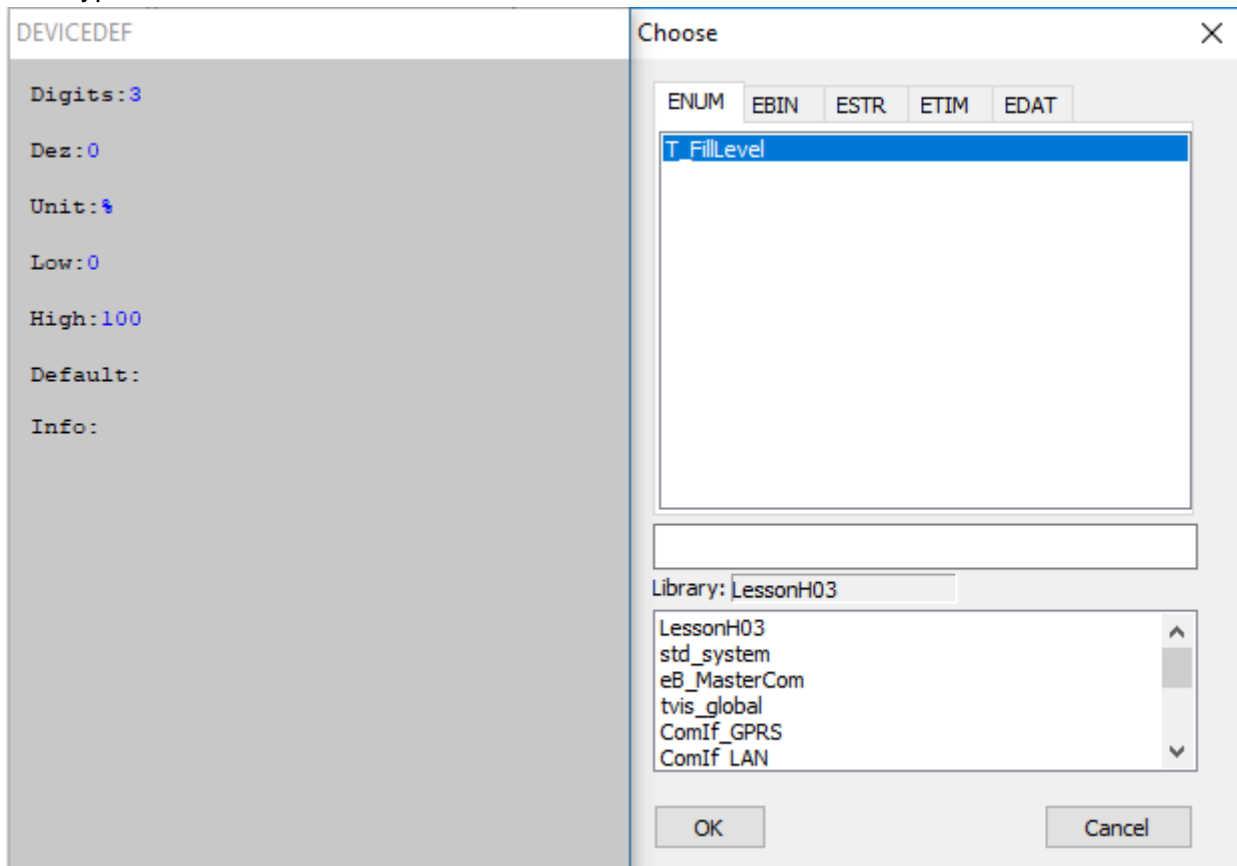
Select the Element AI\_Level:



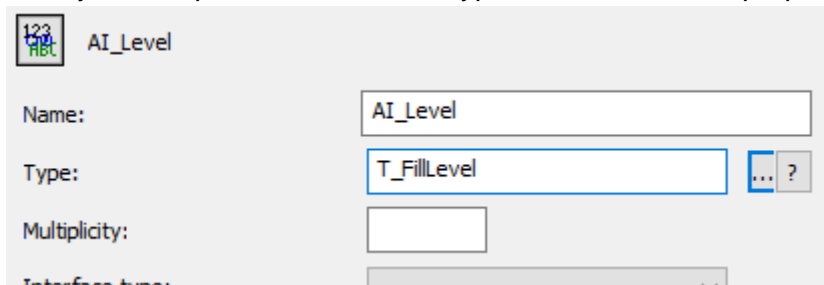
We want to change our data type. In the property editor <click> on the “...”-Button behind the current Type:



In the now opened window the Tab ENUM is already selected. The list will show our new data type T\_FillLevel. Select the data type and the second window will show information about the selected data type:

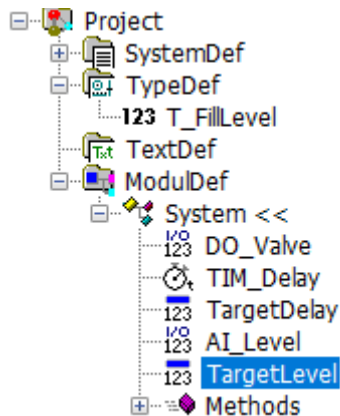


After you accept with OK the data type is selected in the property editor.

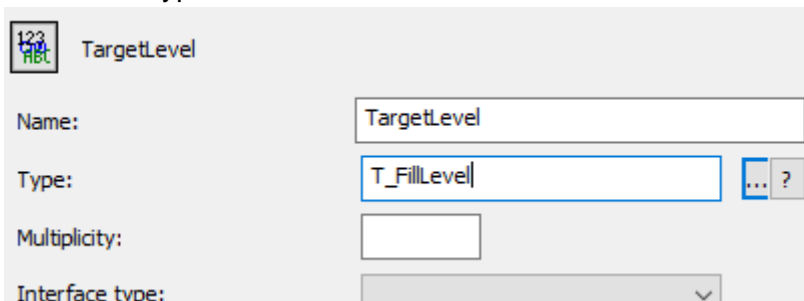




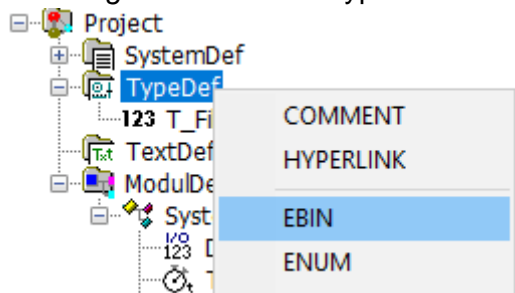
You can also change the data type by just typing the correct data type into the input field. Let us do this for the ELEMENT TargetLevel. Select the ELEMENT TargetLevel:



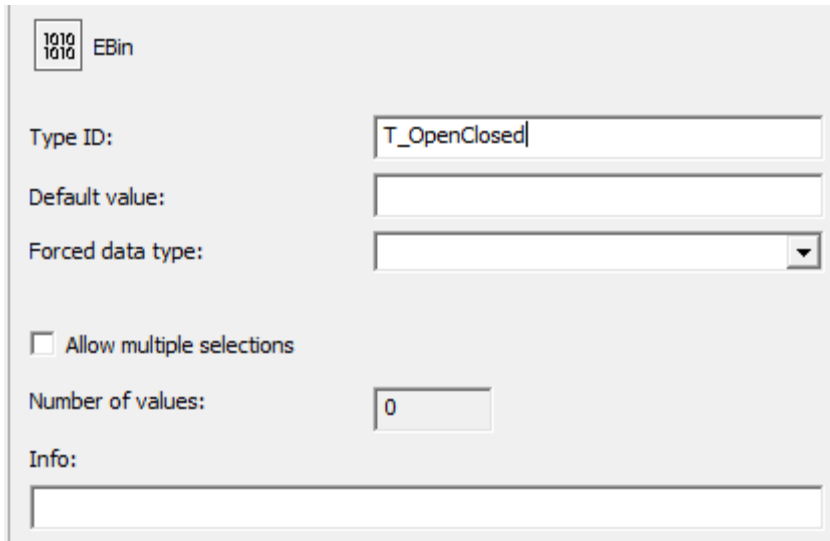
Select the Type of the ELEMENT and delete the content and replace it with the text T\_FillLevel:



Now <right click> on the TypeDef node again and select EBIN:

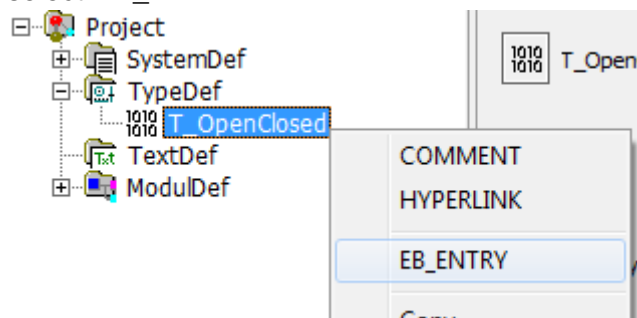


This creates an enumerative data type. In the property editor give the new EBIN a Type ID of T\_OpenClosed. This is the name of our new data type.

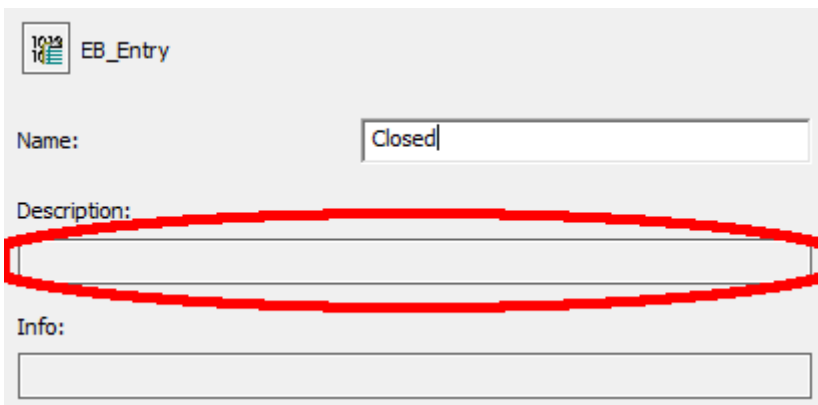


The screenshot shows the 'EBin' property editor. The 'Type ID' field contains 'T\_OpenClosed'. The 'Default value' field is empty. The 'Forced data type' field is a dropdown menu. The 'Allow multiple selections' checkbox is unchecked. The 'Number of values' field contains '0'. The 'Info' field is empty.

Press <F5> to update the project tree. Now <right click> on the new data type T\_OpenClosed and select **EB\_ENTRY**:

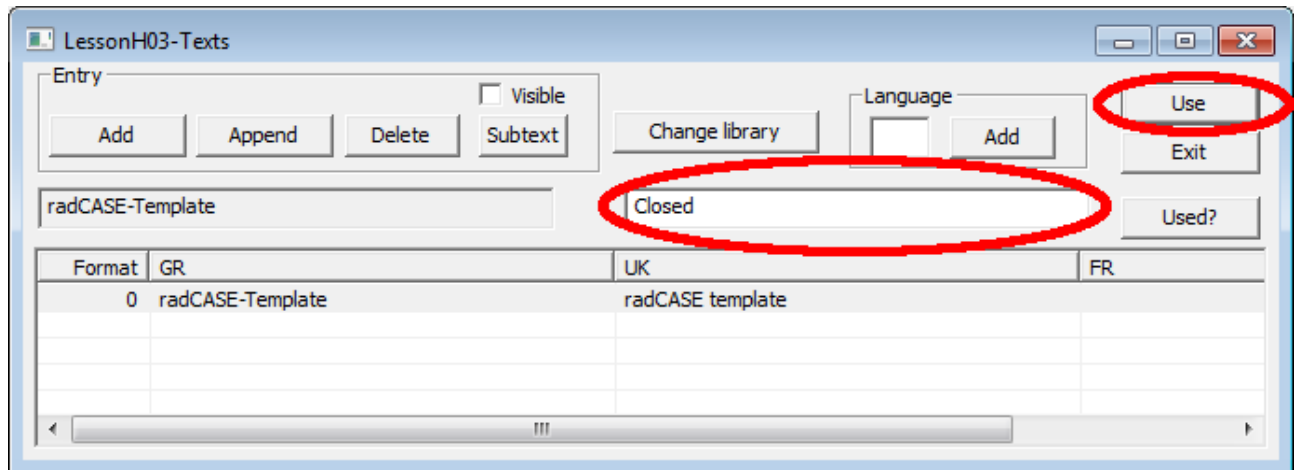


This creates a new value for our enumerative data type. In the property editor enter the name Closed.

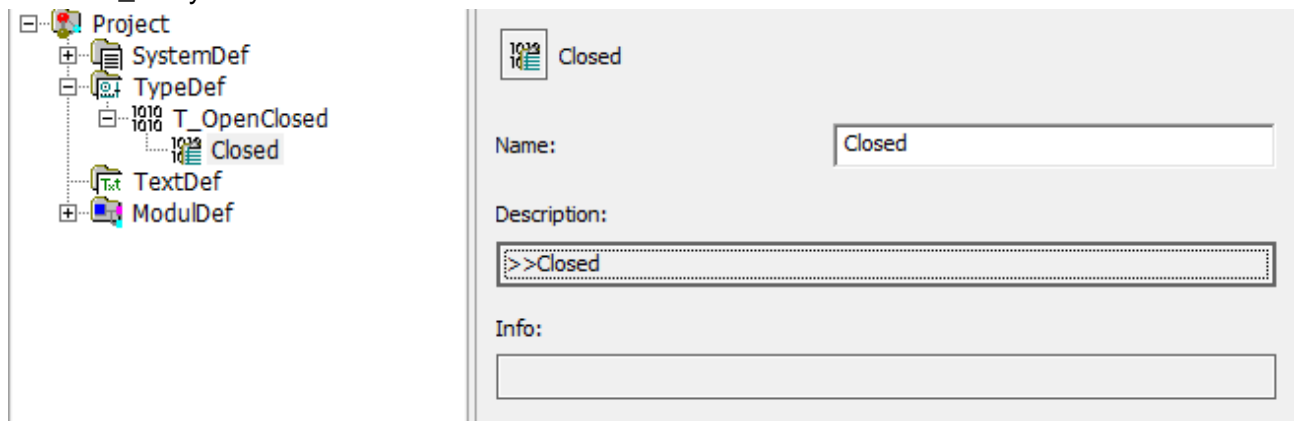


The screenshot shows the 'EB\_Entry' property editor. The 'Name' field contains 'Closed'. The 'Description' field is empty and is highlighted with a red oval. The 'Info' field is empty.

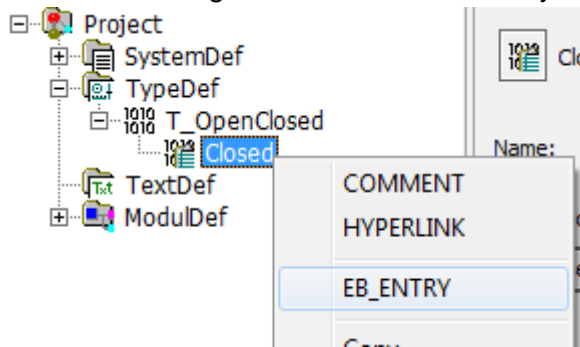
It is important to have a description for every EB\_Entry. So we will now add a single lingual description (this will be explained in detail in a later lesson) to this EB\_Entry. <Click> onto the rectangle below Description and enter "Closed" as Description and accept with Use.



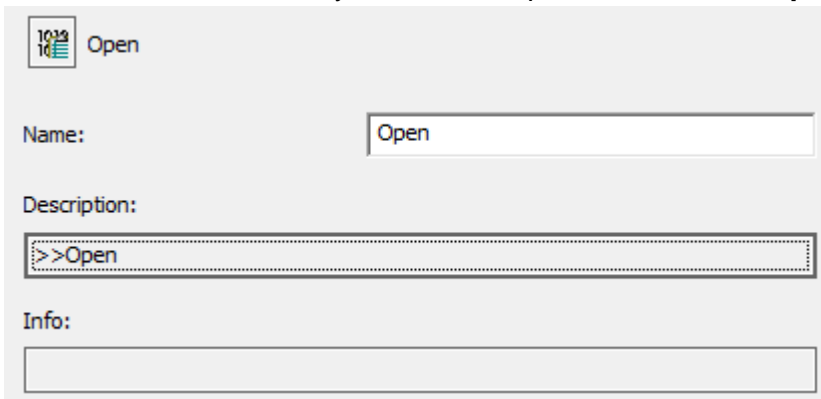
Your EB\_Entry now should look like this.



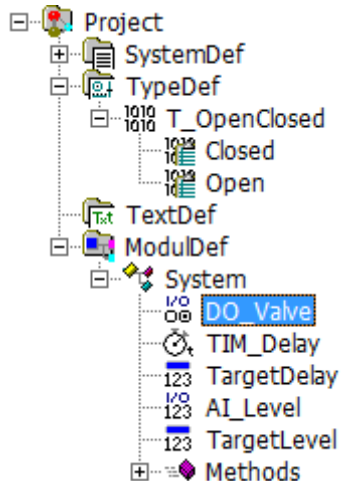
Now <Shift + Right click> on the EB\_Entry Closed and again select EB\_ENTRY:



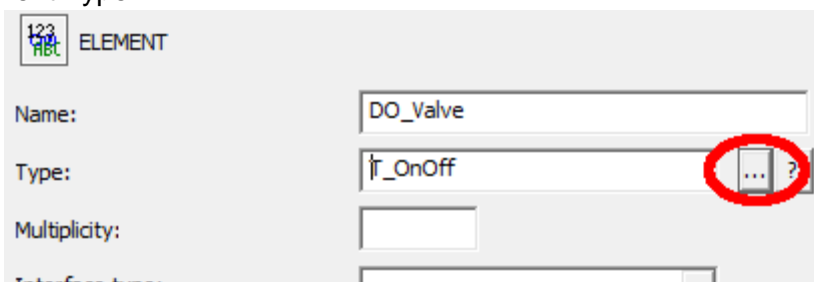
Give the second EB\_Entry the **Name** "Open" and the **Description** "Open":



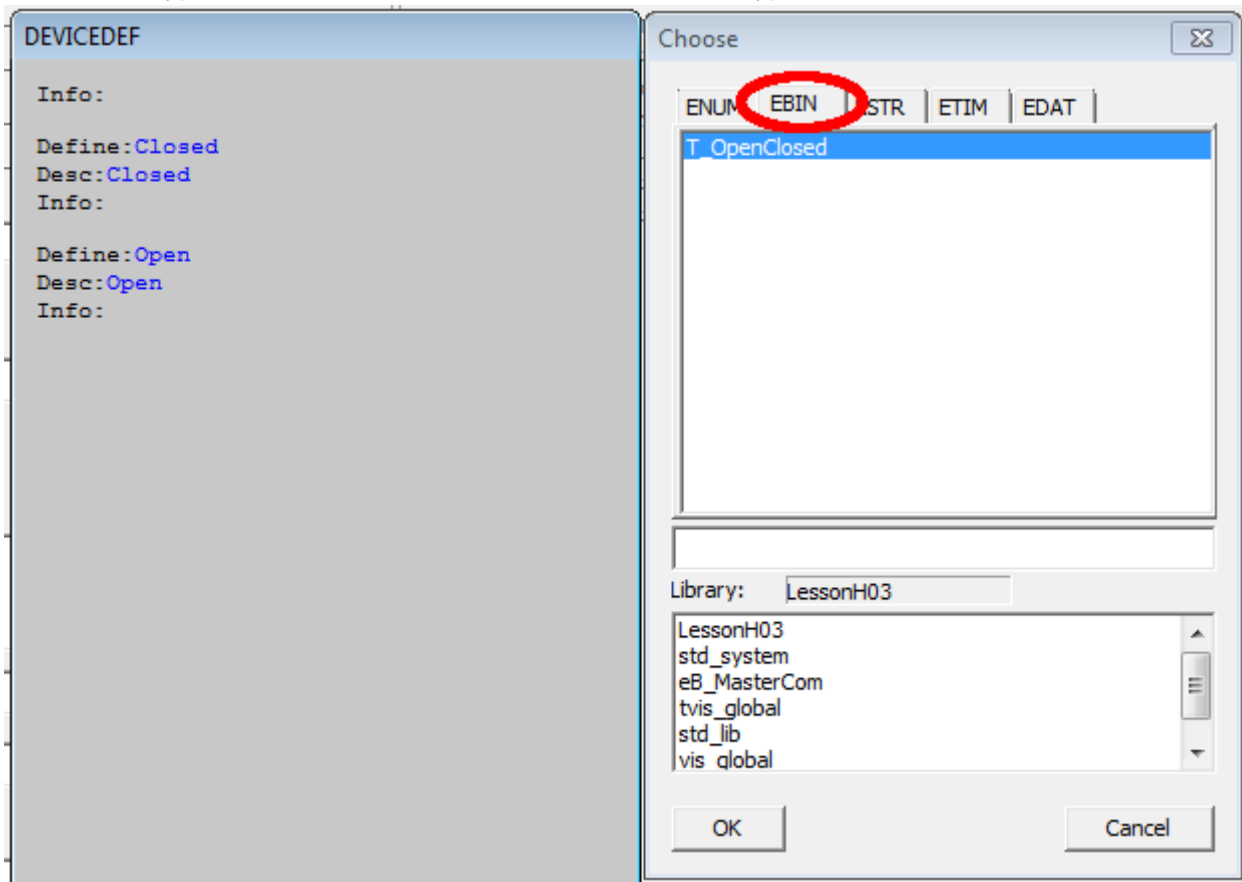
We have created our new data type. Now select the Element DO\_Valve:



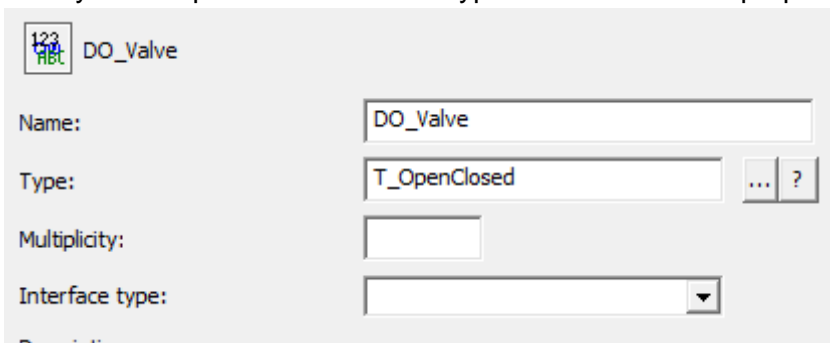
We want to change our data type. In the property editor <click> on the "..."-Button behind the current Type:



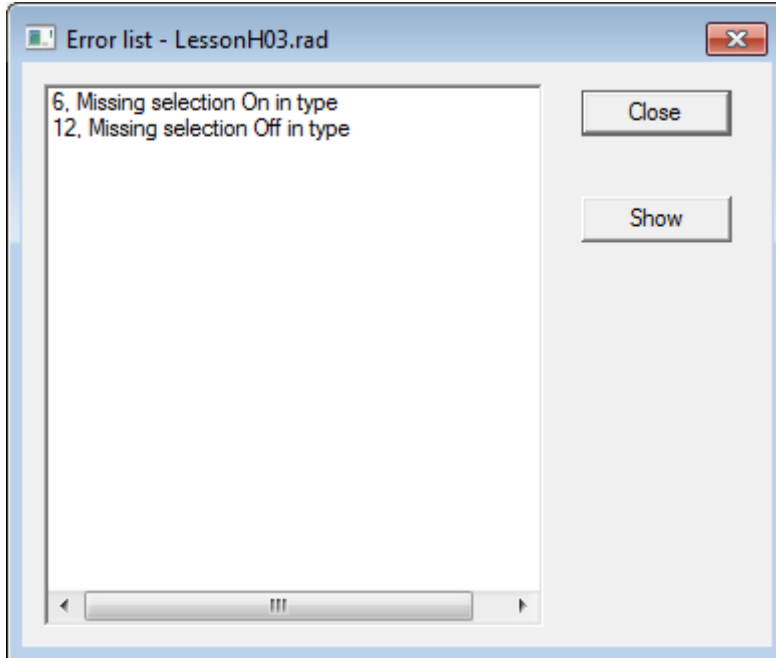
In the now opened window select the Tab EBIN. After this the list will show our new data type T\_OpenClosed. Select the data type and the second window will show information about the selected data type, in this case the different states the data type has:



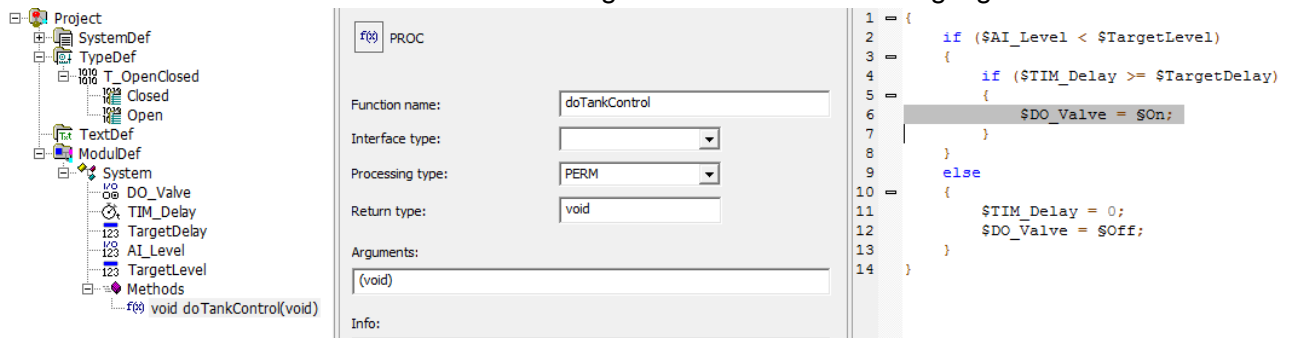
After you accept with OK the data type is selected in the property editor.



Now DO\_Valve can use the values Open and Closed. But in our function we still assign the values Off and On. If we can't remember exactly every occurrence where those values were used, we can simply use the error localization to find those occurrences. Press <F4> to create and start the simulation. During the model compilation errors will be detected and the simulation creation is aborted and instead an error list is shown:



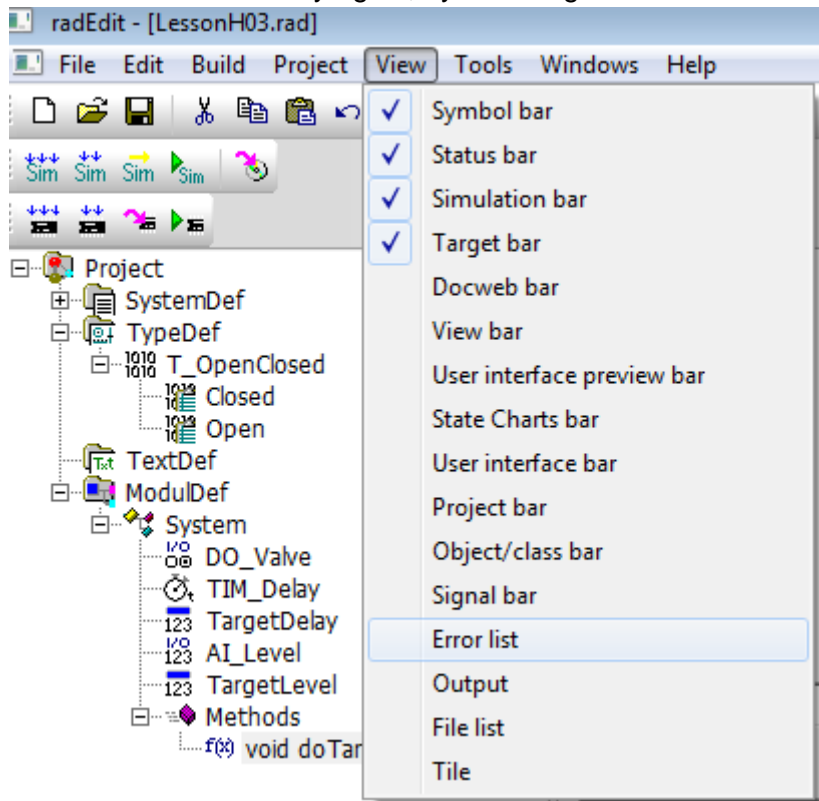
Now make a <double click> on the first line in the error list. You will notice the editor will select the cause of the error in your project. To better see it, just close the error list and you will see the method doTankControl selected and the wrong line in that function is highlighted:



Now delete \$On and use <Shift + F9> to select the state Open instead. After this the code of the entry should be:

```
$DO_Valve = $Open;
```

At this point we know, there is still another error in the project. We could use <F4> again to let the design compiler detect the other error again and show the error list. But instead we just open the error list from our last try again, by selecting **Error list** in the **View** menu of radEDIT:



You will see the same error list from before again. Now <double click> on the second error message and close the error list again. The other wrong line in our function is highlighted.

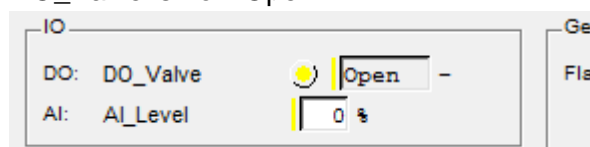
Delete \$Off in the code and use <Shift + F9> to select Closed. Your code should now look like this:

```
$DO_Valve = $Closed;
```

After this all errors should be corrected and the simulation can be created again.

### 4.3.3 Conclusion

Use <F4> to create and start the simulation. Open the module view again. You will see, the value of DO\_Valve is now Open:



## 4.4 Lesson H04: Environment Simulation

### 4.4.1 Goal

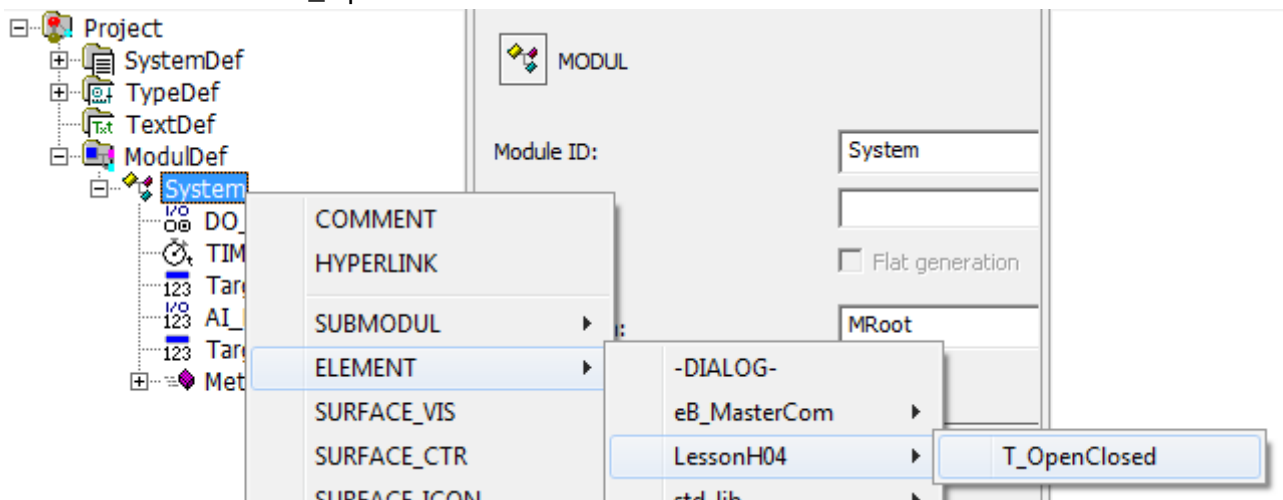
In this lesson you will learn how to simulate the environment your real hardware will be connected to, so you can easily test your process in the simulation.

### 4.4.2 Content

You can start this lesson by using the project you created in lesson H03 or by opening the project LessonH04 that is delivered with your radCASE copy.

Until now, we had to simulate the process manually setting the current level of the tank, by editing the element AI\_Level. Now we want the tank to fill automatically, if the inlet is open. Also we will create an element which will only exist in the project monitor, to simulate an outlet which is opened by hand in the tank. If that outlet is open, this should also affect the fill level of the tank.

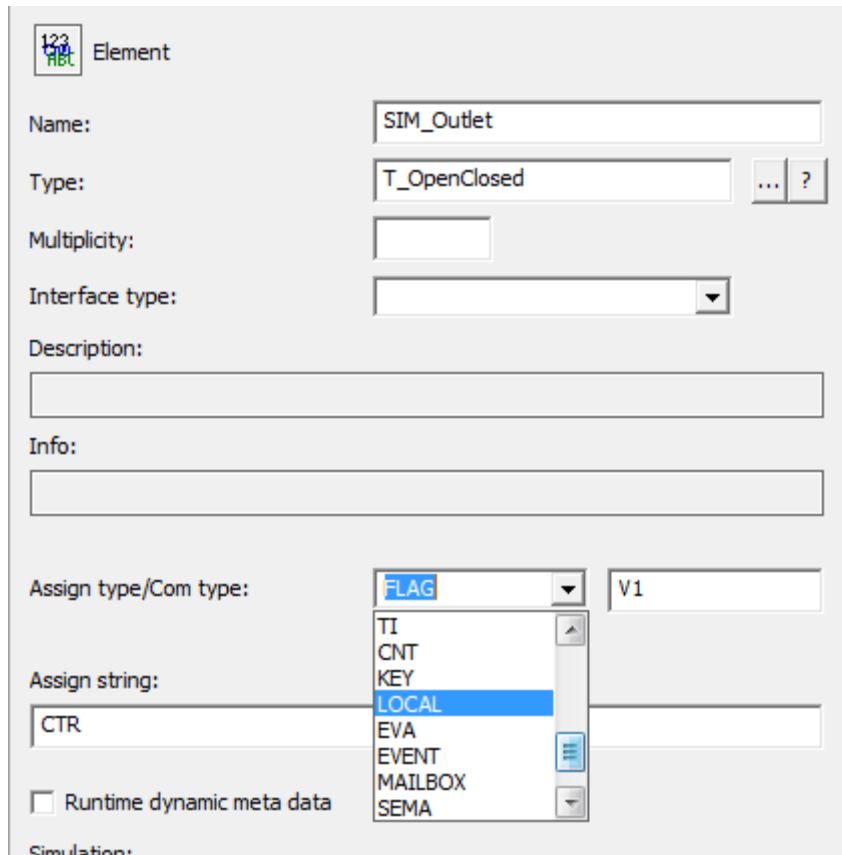
First we will create the outlet. <Right click> on the System module and select ELEMENT->LessonH04->T\_OpenClosed:



If you did not use LessonH04, replace LessonH04 by the filename you are currently working with.



In the property editor set the **Name** of the new element to “SIM\_Outlet” and the **Assign type** to **LOCAL**:



Element

Name: SIM\_Outlet

Type: T\_OpenClosed ... ?

Multiplicity:

Interface type:

Description:

Info:

Assign type/Com type: FLAG V1

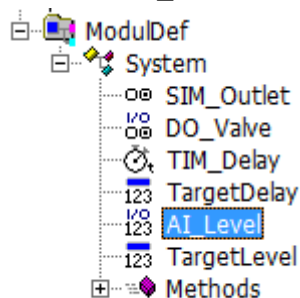
Assign string: CTR

☐ Runtime dynamic meta data

Simulation:

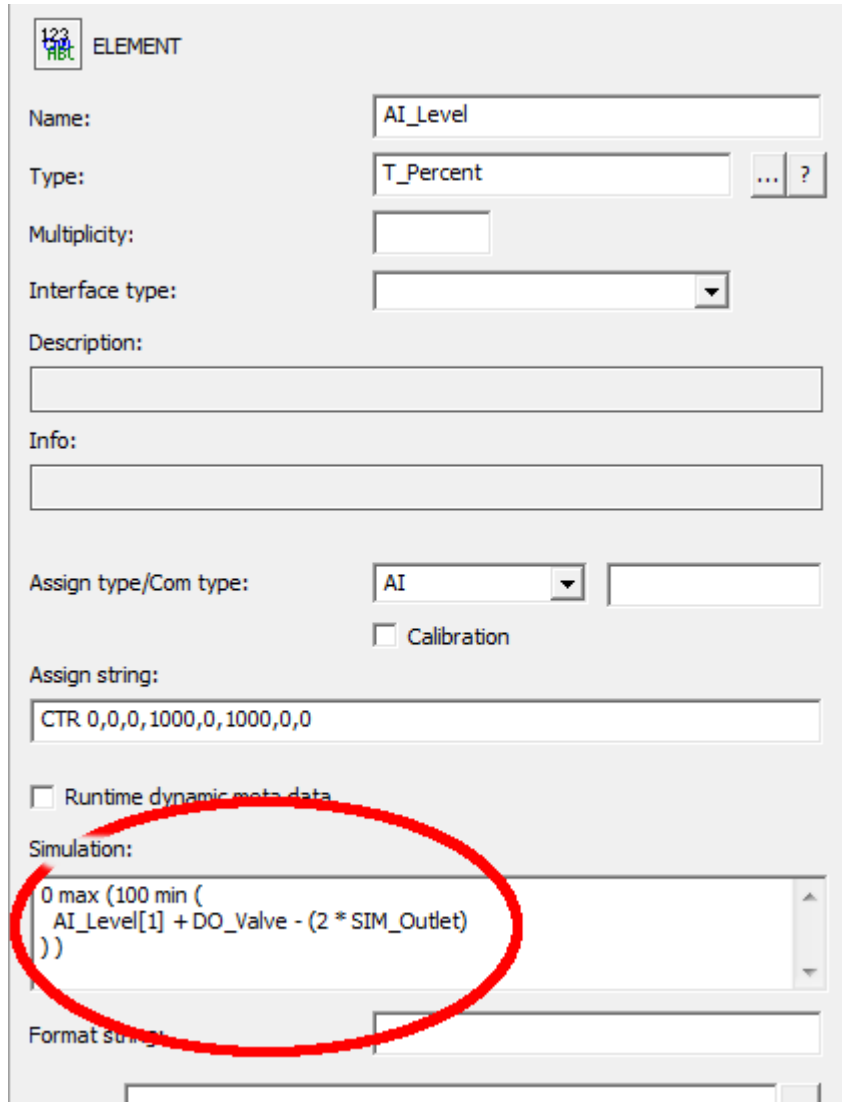
All other settings will be set to valid default values. The element you created will only exist in the project monitor and will not exist on the real hardware. So this element will not cost any memory on the hardware.

Now we want to simulate the behavior of filling the tank in dependency of the inlet DO\_Valve and the outlet SIM\_Outlet. Select the element AI\_Level in the project tree:



In the property editor enter the following formula into the field **Simulation**:

```
0 max (100 min (  
    AI_Level[1] + DO_Valve - (2 * SIM_Outlet)  
))
```



123 ELEMENT

Name: AI\_Level

Type: T\_Percent ... ?

Multiplicity:

Interface type:

Description:

Info:

Assign type/Com type: AI

☐ Calibration

Assign string: CTR 0,0,0,1000,0,1000,0,0

☐ Runtime dynamic meta data

Simulation: 0 max (100 min ( AI\_Level[1] + DO\_Valve - (2 \* SIM\_Outlet) ) )

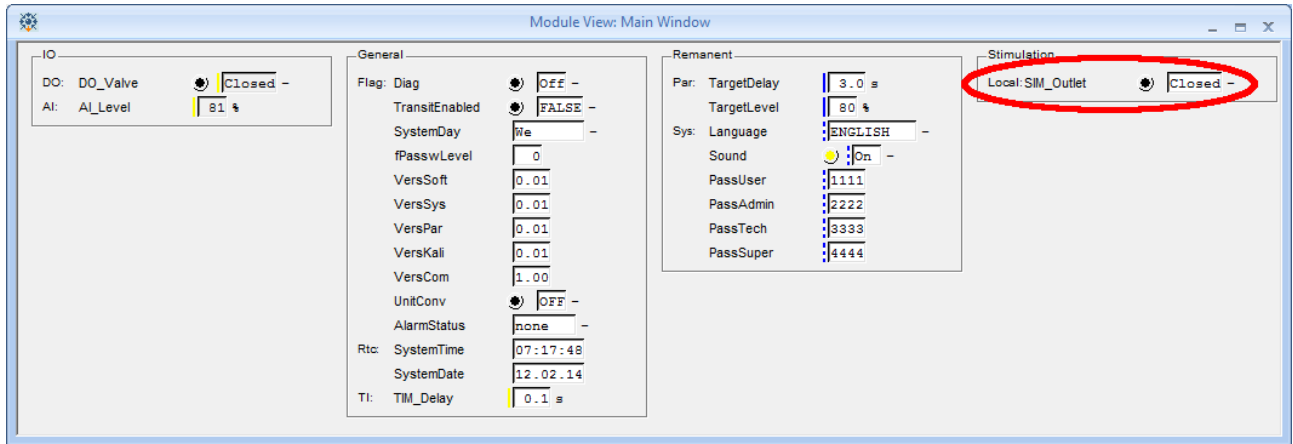
Format string:

AI\_Level[1] means to use the last value the element had. That last value is increased when the Inlet is open and decreased when the outlet is open. The outlet is multiplied by two, so the outlet will empty the tank faster, than the inlet can fill it.

The max and min functions simply ensure the tank will stay in the range from 0 to 100 percent.

### 4.4.3 Conclusion

Hit <F4> to create and start the simulation. Open the module view again:



Now you will see AI\_Level automatically is increased until DO\_Valve is closed by the process. If you <click> on SIM\_Outlet, the value of AI\_Level will decrease and after a short time DO\_Valve will open and the tank level decreases slower. If you now <click> again on SIM\_Outlet the outlet will be closed and the tank will fill again. DO\_Valve will close as soon as the TargetLevel is reached.

## 4.5 Lesson H05: State Machine

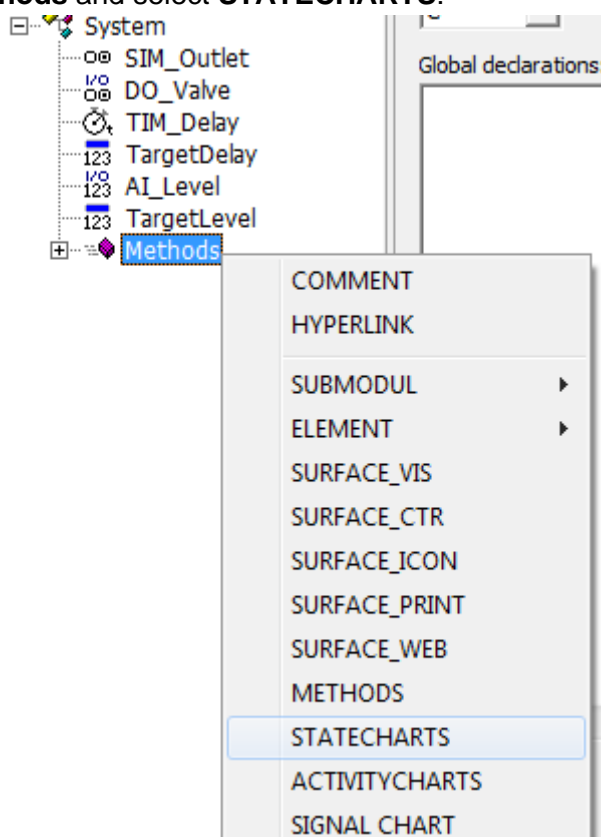
### 4.5.1 Goal

In this lesson you will learn the usage of state machines. These are a second way to adding behavior to your project.

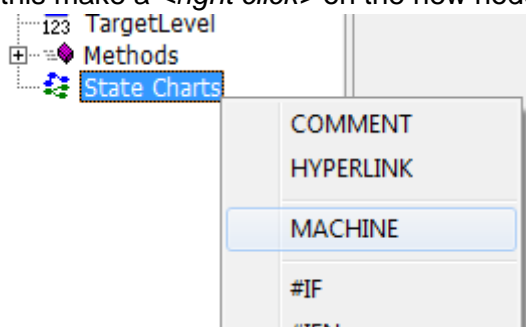
### 4.5.2 Content

You can start this lesson by using the project you created in lesson H04 or by opening the project LessonH05 that is delivered with your radCASE copy.

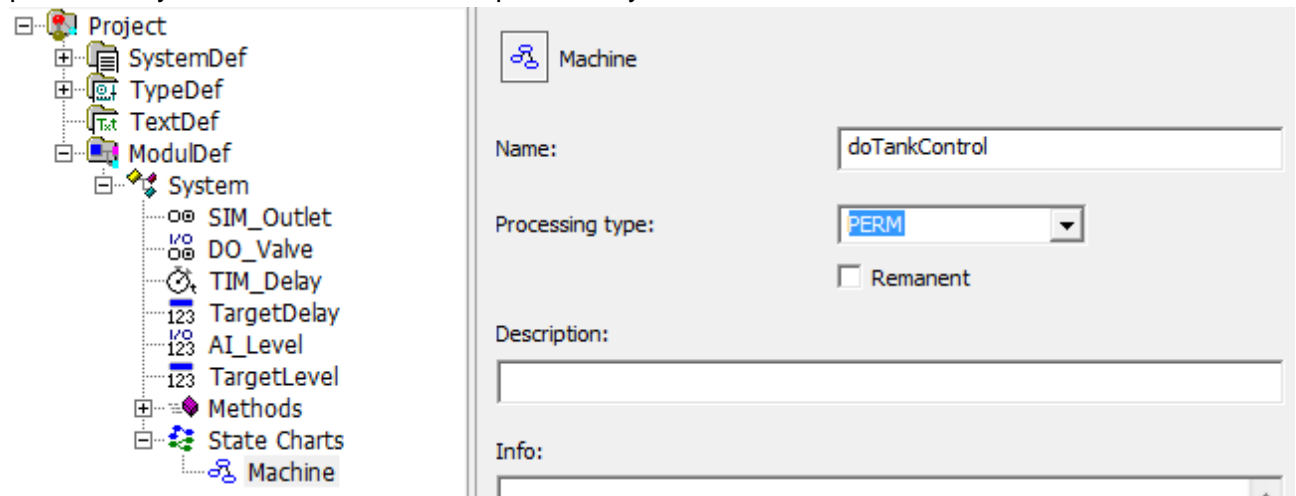
We will replace the function from Lesson H02 with a state machine. Use *<Shift + right click>* on the **Methods** and select **STATECHARTS**:



After this make a *<right click>* on the new node **State Charts** and select **MACHINE**:



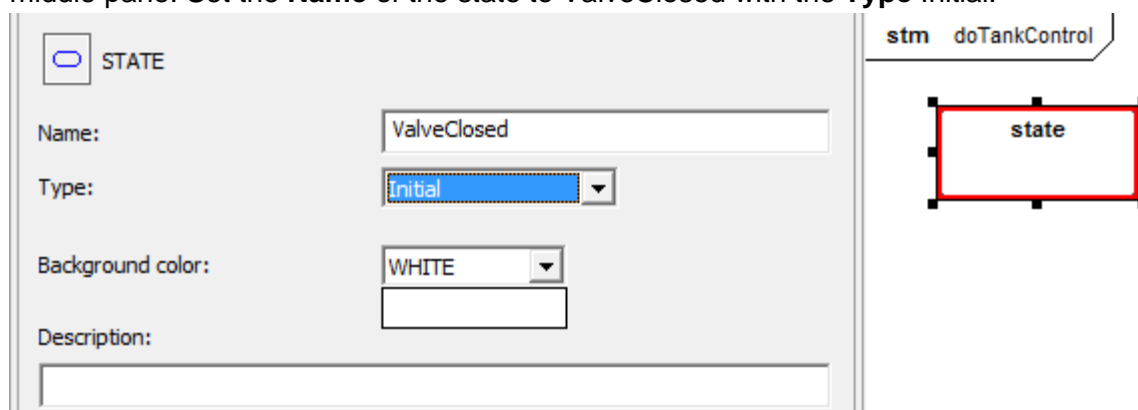
Set the **Name** of the **Machine** to doTankControl, this is the same name we used earlier for our function. Also set the **Processing type** to **PERM**. This again lets the state machine be executed permanently which means it is called periodically:



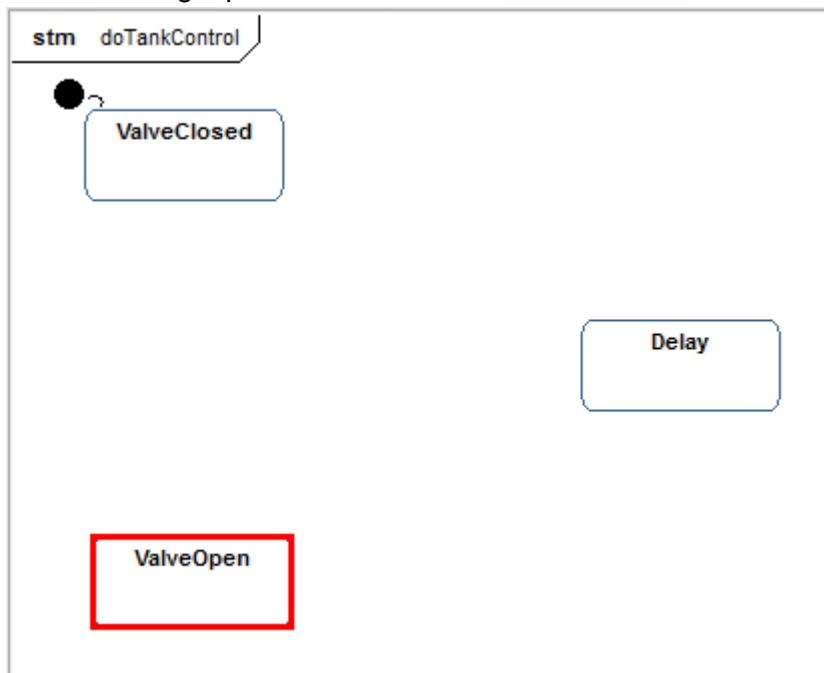
Now we will create three states the machine can be in. <Right click> in the editor in the right pane of radEDIT and select Add state.



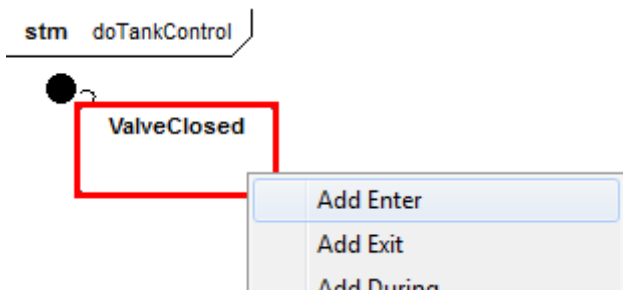
This will add a rectangle named state in the right pane. You can drag and drop that new state to position it in the right pane. Also you will see the properties of the state in the property editor in the middle pane. Set the **Name** of the state to ValveClosed with the **Type** Initial:



The type **Initial** determines the State in which the Statemachine starts. Now we will add two more states each with the Type Normal. Create the two States “Delay” and “ValveOpen” and arrange them in the right pane like this:



Now we will add Entry-Code to each of our states. This is code that will get executed once, when the state of the state machine gets active. <Right click> on the state ValveClosed and select **Add Enter**:



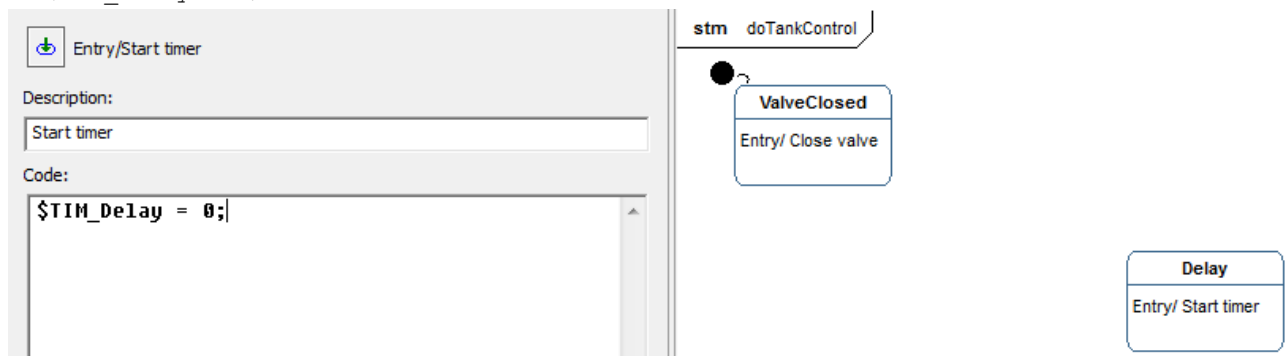
In the property editor you can enter a Description which will be displayed in the graphical editor in the right pane and code that is executed. In the code you can again use `<F9>` and `<Shift + F9>` to select variables and possible values. We want the state to close the valve on entering so add the Description: “Close valve” and add the following code:

```
$DO_Valve = $Closed;
```



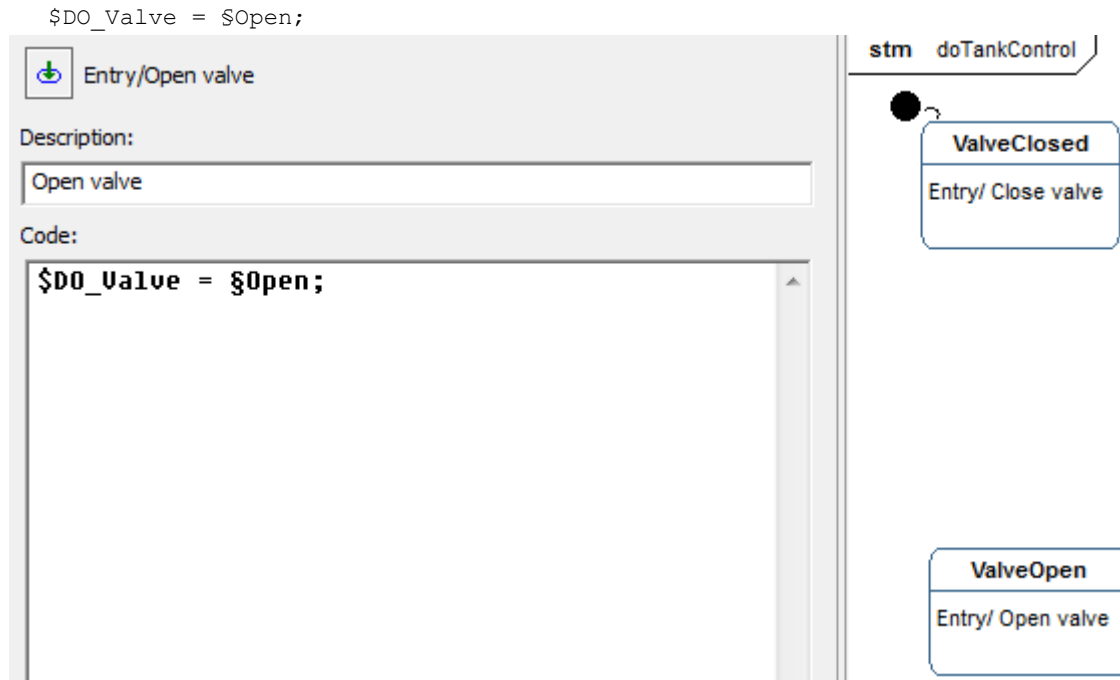
Now we want the Delay to start the timer, so add an entry there with the Description “Start timer” and the code:

```
$TIM_Delay = 0;
```



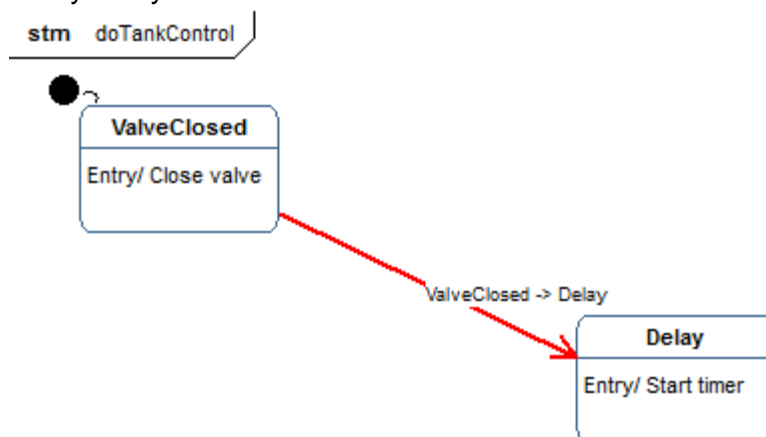
Technically the code does not start the timer, but reset it to 0. But because the timer value is only used within that state, this is all we need. There are functions to really start and stop a timer, but those functions are seldom required and will not be covered in this tutorial.

And at last we want the state ValveOpen to open the Valve so add an Entry there with the Description “Open valve” and the following code:



Now all that is left to do in the statemachine is to determine under which conditions the states can change. This also determines which changes are valid.

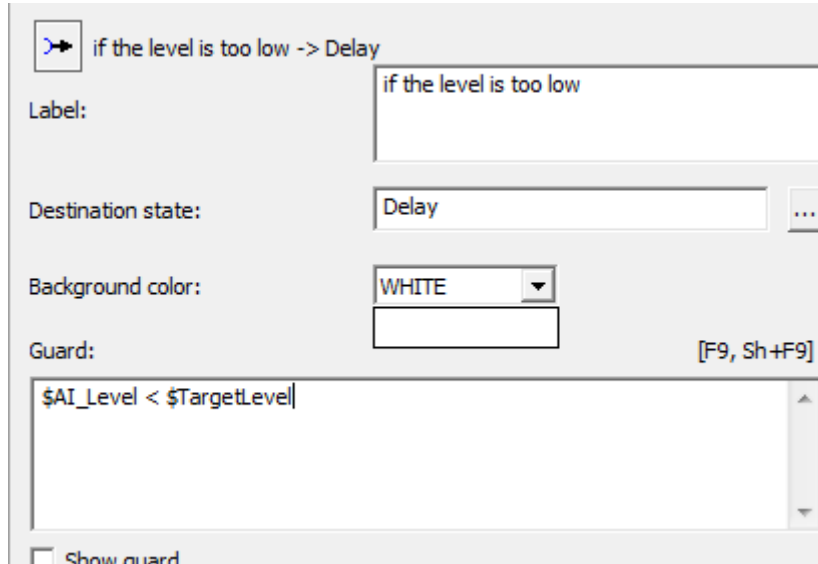
Move the mouse onto the border of the state ValveClosed until the cursor will turn into a big “+”. <Click> and you will see a line connecting the state with your mouse cursor. <Click> on the state Delay and you will have created a connection from ValveClosed to Delay:





In the property editor we will now set the condition. The **Label** defines the label which shows in the graphical editor in the right pane. The **Guard** defines a condition, which will trigger the state change. We will label our connection “if the level is too low” and set the following code into the guard:

```
$AI_Level < $TargetLevel
```

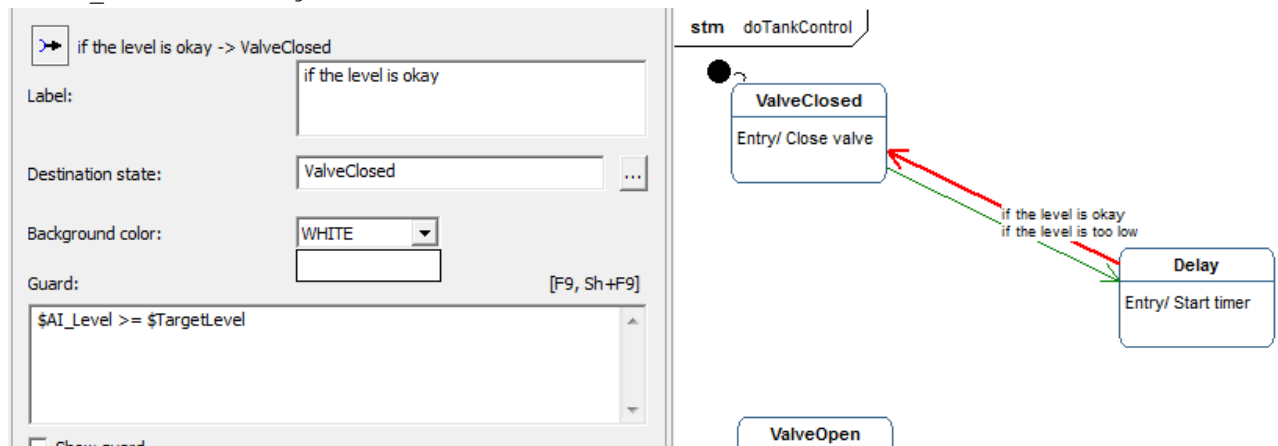


Property editor for the transition "if the level is too low -> Delay":

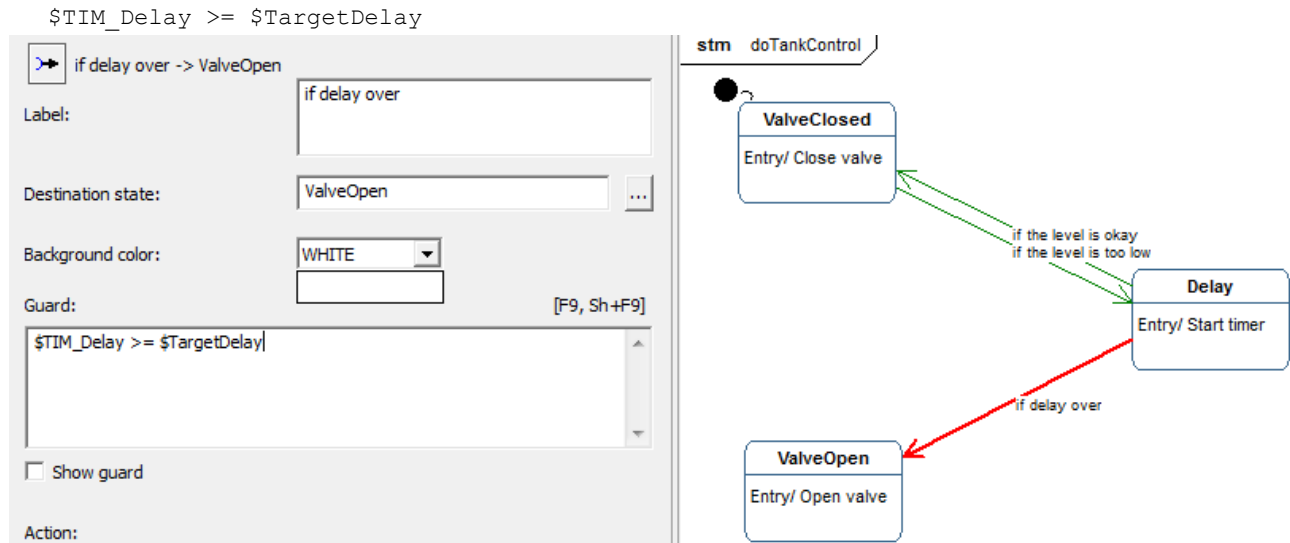
- Label:** if the level is too low
- Destination state:** Delay
- Background color:** WHITE
- Guard:** `$AI_Level < $TargetLevel` [F9, Sh+F9]
- ☐ Show guard

If the level is okay, while running the delay (e.g. the user changes the target fill level), the state machine should change back into the state ValveClosed. So add another connection starting in State Delay to the State ValveClosed. Set the **Label** to “if the level is okay” and the **Guard** to:

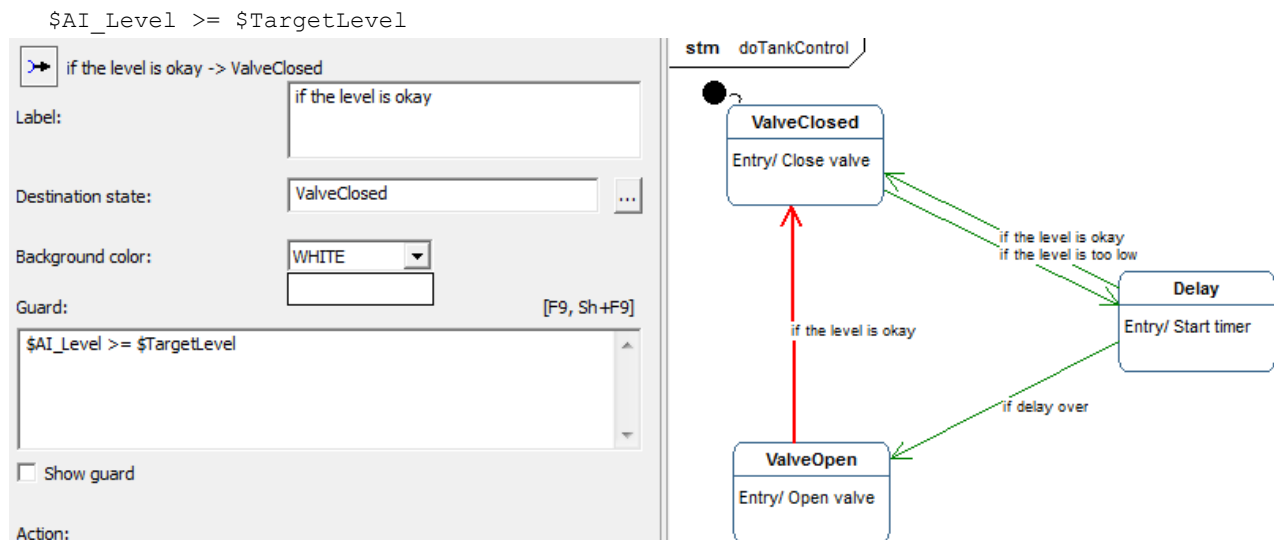
```
$AI_Level >= $TargetLevel
```



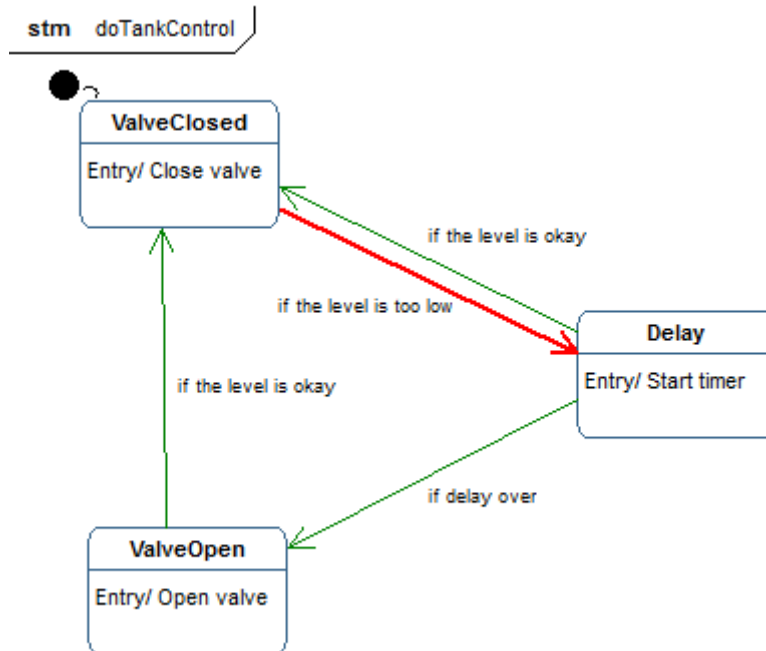
Now we want to open the valve, when the delay time is over. Add a connection from Delay to ValveOpen and set the label to “if delay over” and the Guard to:



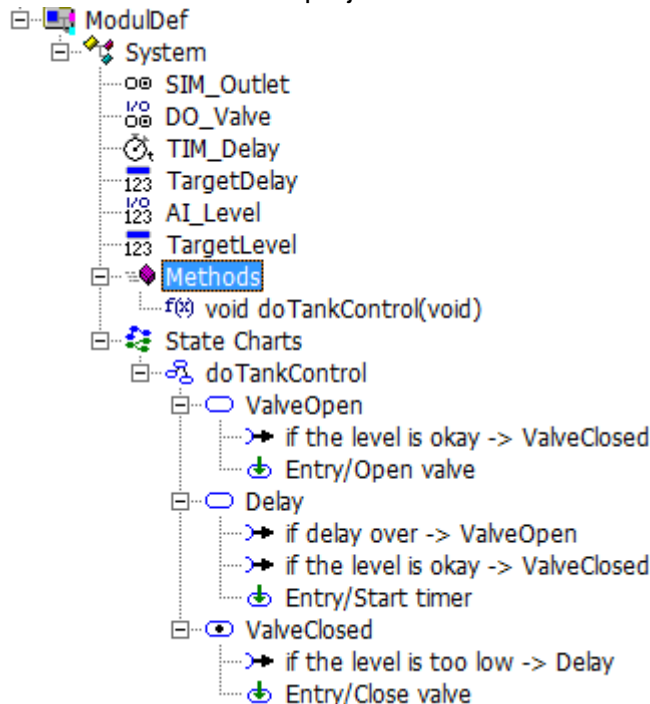
At last we want to close the valve as soon as the target fill level is reached. So add a connection from ValveOpen to ValveClosed with the Label “if the level is okay” and the Guard:



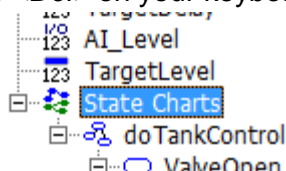
This finishes the behavior of the state machine. We will now use drag and drop to arrange the states and labels in the graphical editor in the right pane to look nicely:



Even though the state machine is finished we are not completely ready yet. We still need to delete the old C-function doTankControl, because else we will have two functions doing the same. Select the Methods node in the project tree:

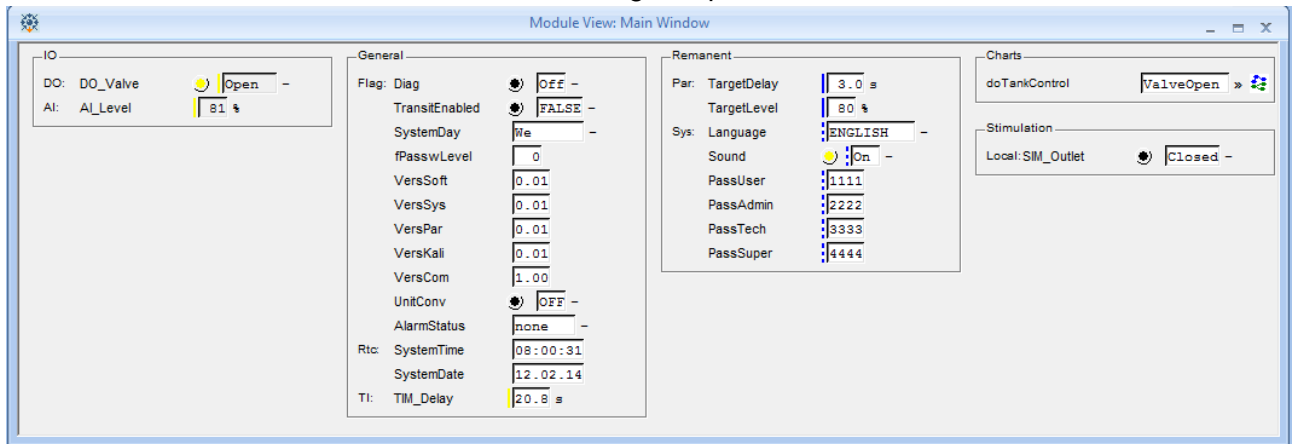


Press <Del> on your keyboard and the Methods node will disappear.

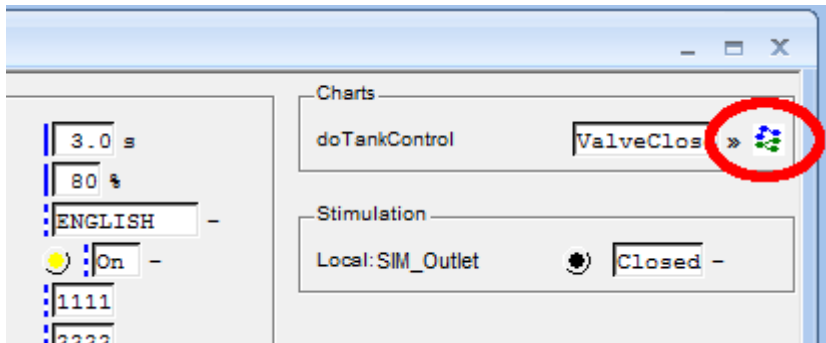


### 4.5.3 Conclusion

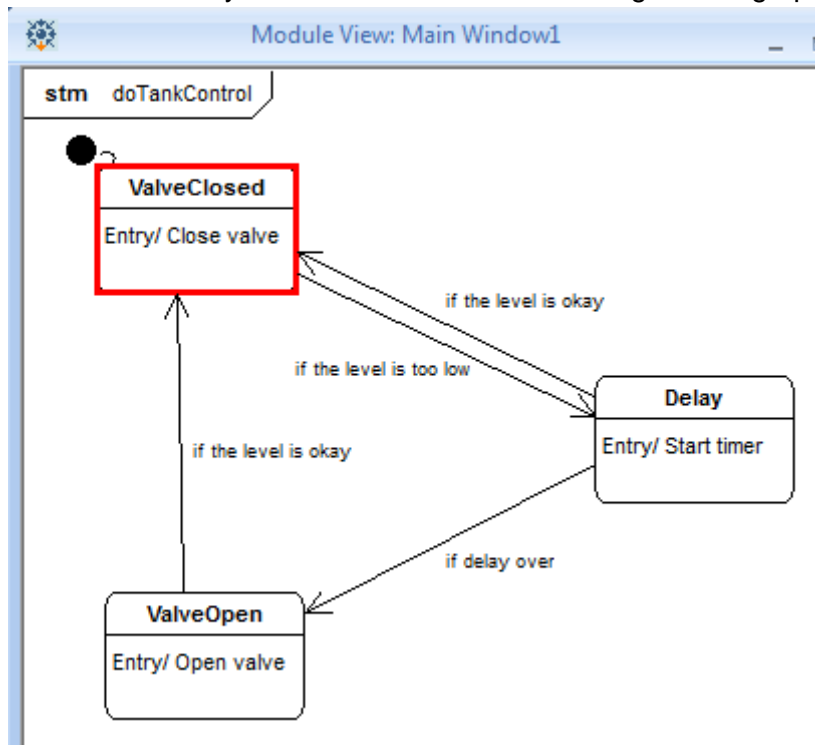
Press <F4> to create and start the simulation. Again open the module view:



You will see there is a new Section called charts in the module view. <Click> on the icon behind doTankControl:



You will see the same graph as in the graphical editor, but the current state has a red rectangle around it. You can see the behavior is the same like before, when you open and close the SIM\_Outlet. But you will also see the state change in the graph.



## 4.6 Lesson H06: First Module / Reusability

### 4.6.1 Goal

In this lesson you will learn what the radCASE items **MODUL** and **SUBMODUL** are and how to use them. You will learn how these two items help you to generate code that is reusable in your project.

### 4.6.2 Content

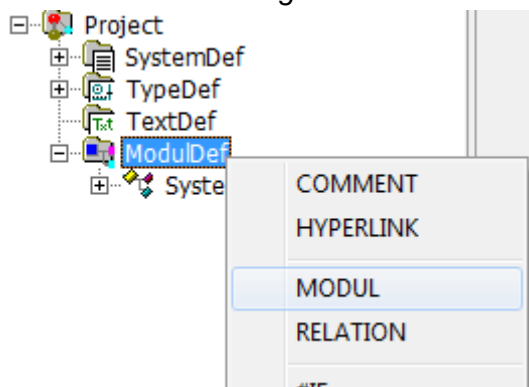
You can start this lesson by using the project you created in lesson H05 or by opening the project LessonH06 that is delivered with your radCASE copy.

Until now we have added all our elements and code to the System module, which can be compared to a main-function in C-code. But for bigger programs you would not add everything to your main function but use functions instead.

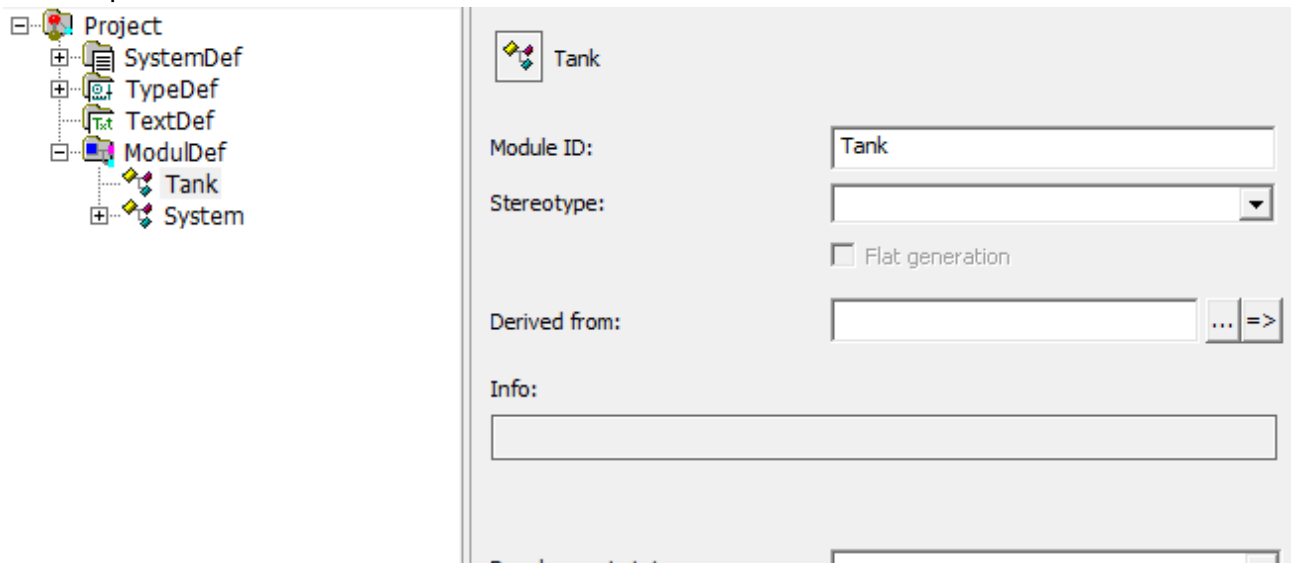
Like functions are used to make different “groups” of code with different meaning, so a module in radCASE is used to group your code in radCASE. But different to a function where you can only add code and some local variables, you can add multiple functions and variables in a module. You can use the variables added to a module in all the functions in the module. So a module helps you to group functionality and variables to logical units. In object orientation this is called a class.

But radCASE still goes one step further and you can also add different HMI in your module. So a radCASE module is a little bit more than a class in object orientation.

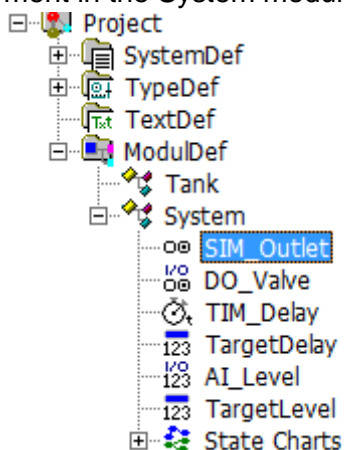
We will now make a logical unit that describes the behavior of our tank. So we will create a module we will call Tank. <Right click> on ModulDef in the project tree and select **MODUL**:



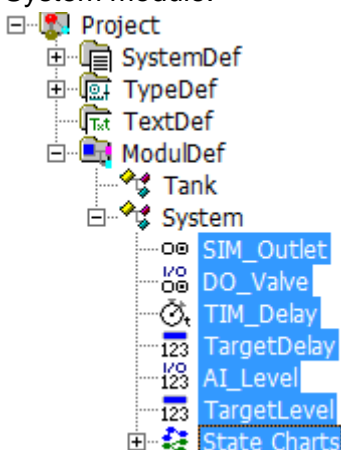
This will create a new module. Enter “Tank” as **Module ID** and leave the rest of the fields empty, because they are not needed at the moment. Press <F5> after entering the ID and the project tree will be updated and show the new module Tank:



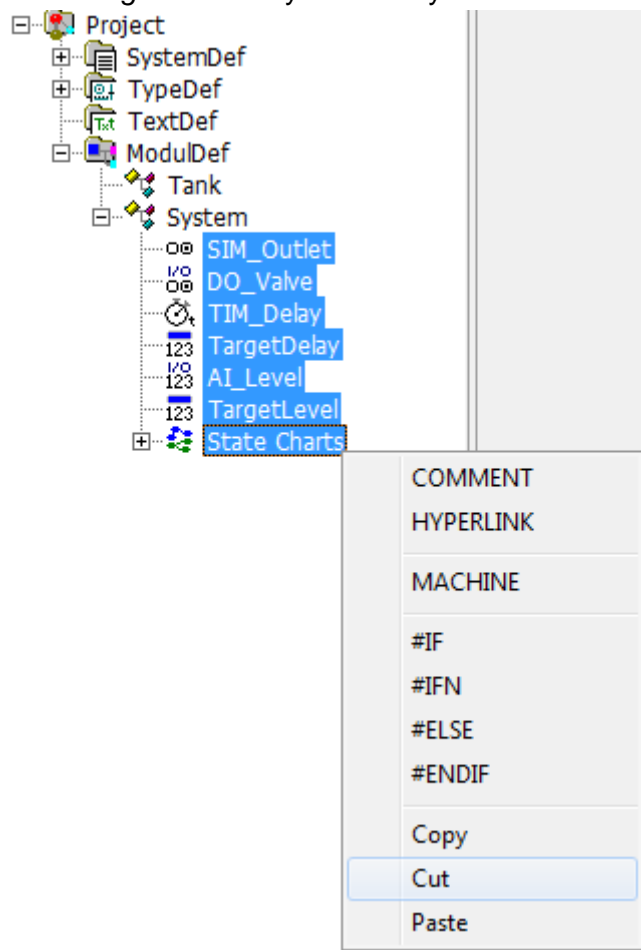
Now we want to move all the Tank functionality to the Tank module. To do this select the first element in the System module:



Now <Shift + Click> on the StateCharts node. You should now have selected everything in the System module:

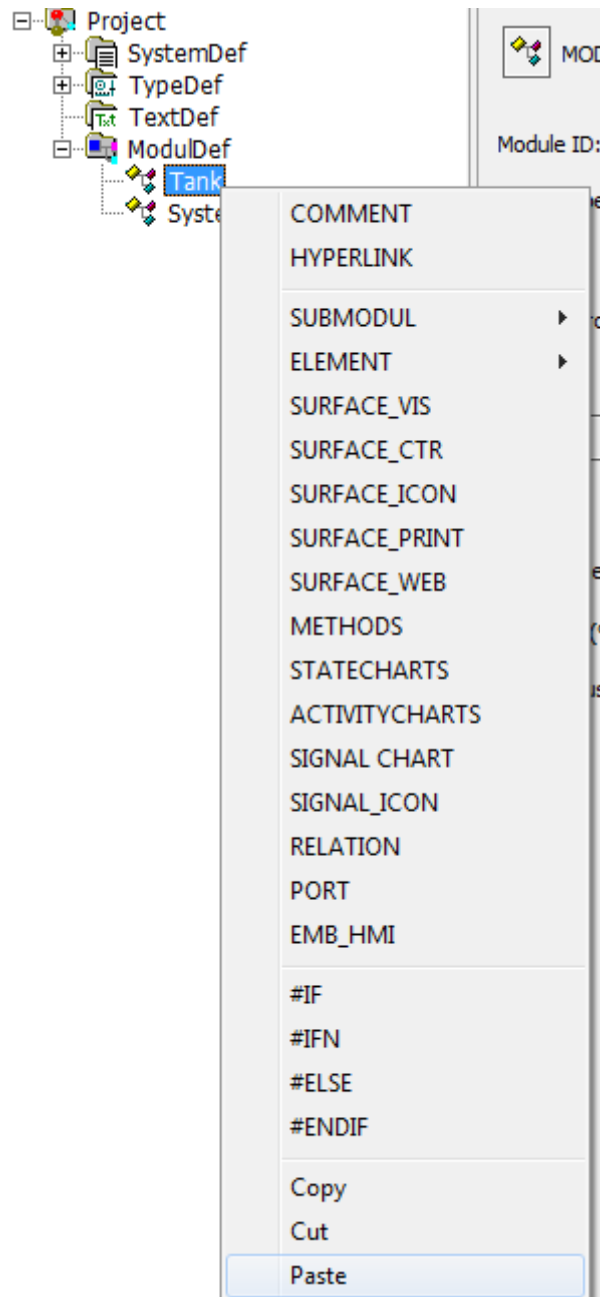


Now <Right click> anywhere on your selection and select **Cut**:

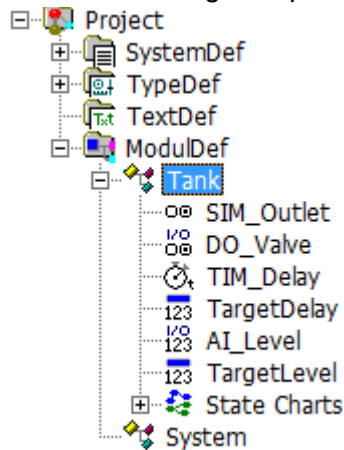




Everything will disappear from the System module. <Right click> on the Tank module and select **Paste**:



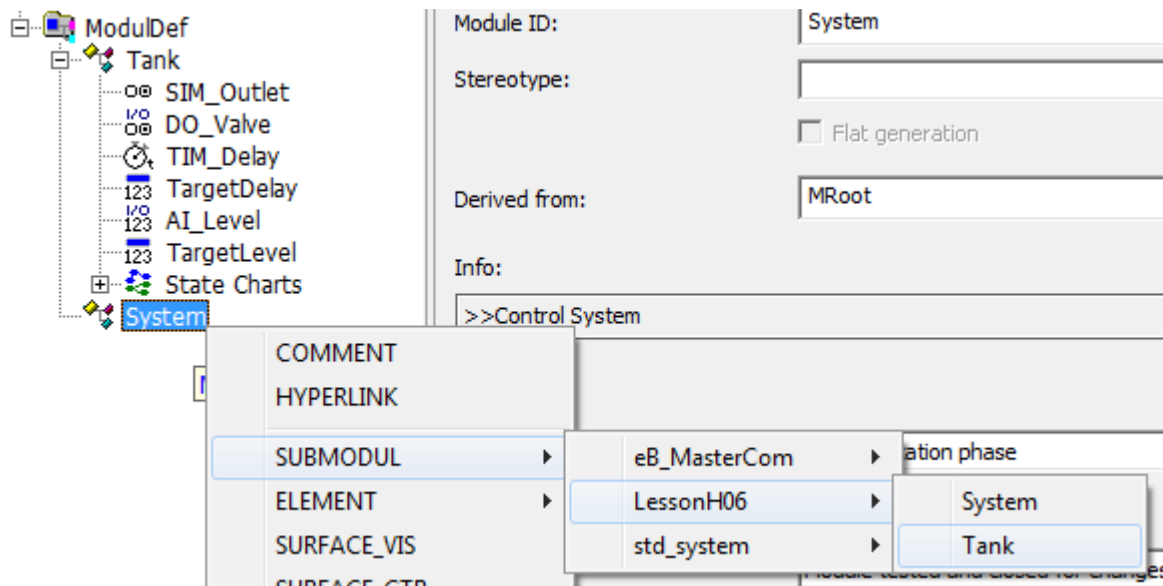
Now all the things we previously added in the System module are moved into the Tank module:



If you move code to a function the function itself will not work until you call the function from your main function. In the same way a class/module will not work when it is not “called” from the System module. But because a class/module is more than a function, we don’t call it, but we make an instance of that class.

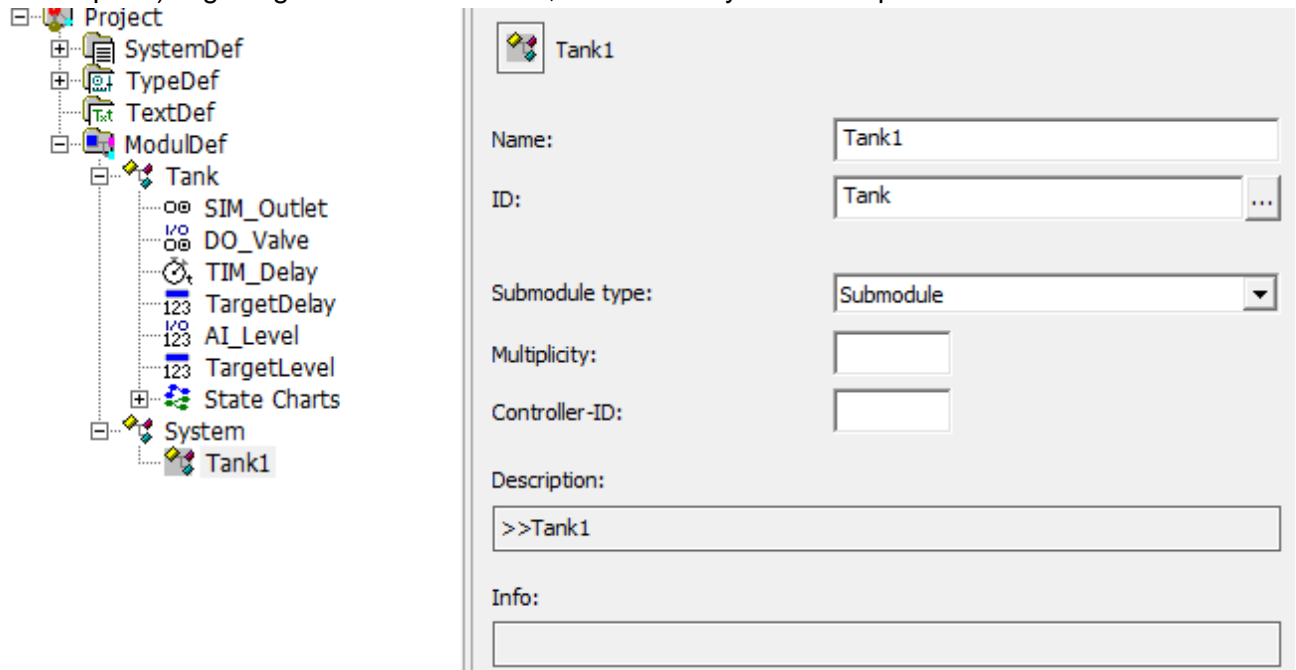
In our example the Tank module is not a tank, but more of a blueprint of a tank. So to use a tank, you have to really build a tank using that blueprint. To build a tank you have to make an instance of that tank. In object orientation this is called an object, in radCASE we call it a submodule.

Now we will create a submodule of our Tank module into the System module. <Right click> on the System module and select SUBMODUL->LessonH06->Tank:

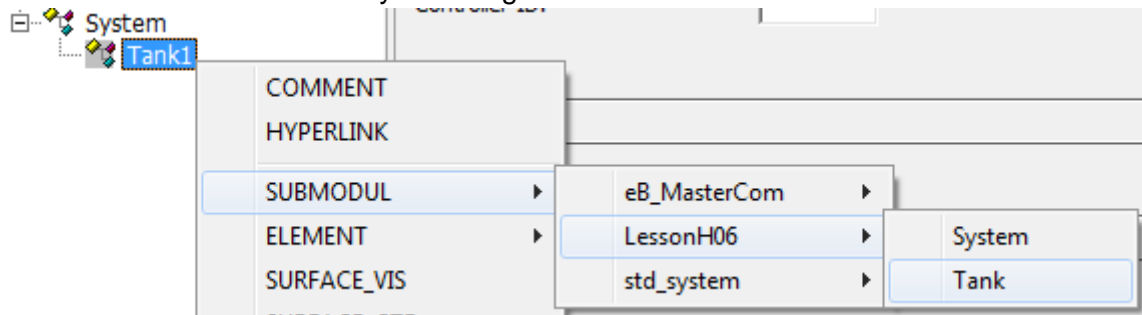


If you did not use LessonH06 please select the name of your project instead.

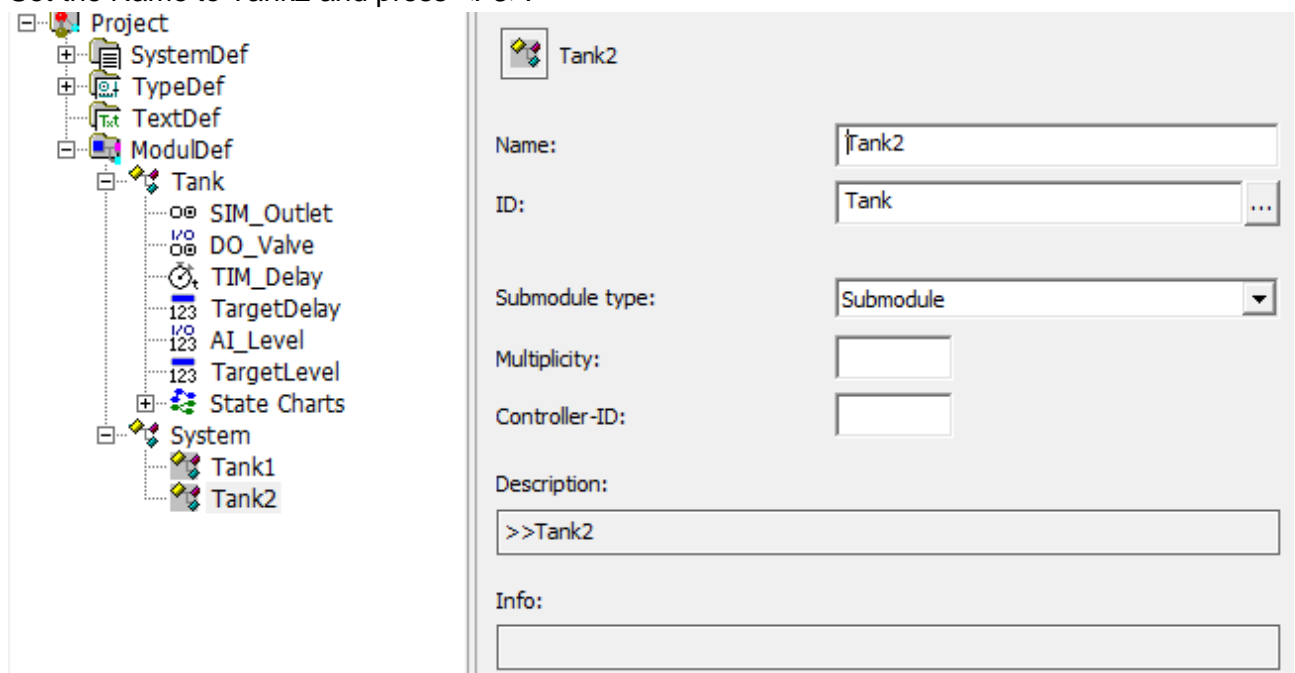
Enter the **Name** Tank1 and press <F5> to update the project tree (this also automatically will add a Description). Again ignore the other fields, because they are not important at the moment.



The grouping of functionality into a module now has the advantage, that you can reuse your module. We want to create a project which has two tanks. So because now we have the logical unit of a tank, we can simply add another tank to our project. Just <Shift + right click> on the submodule Tank1 and add another Tank by selecting SUBMODUL->LessonH06->Tank:



Set the Name to Tank2 and press <F5>:

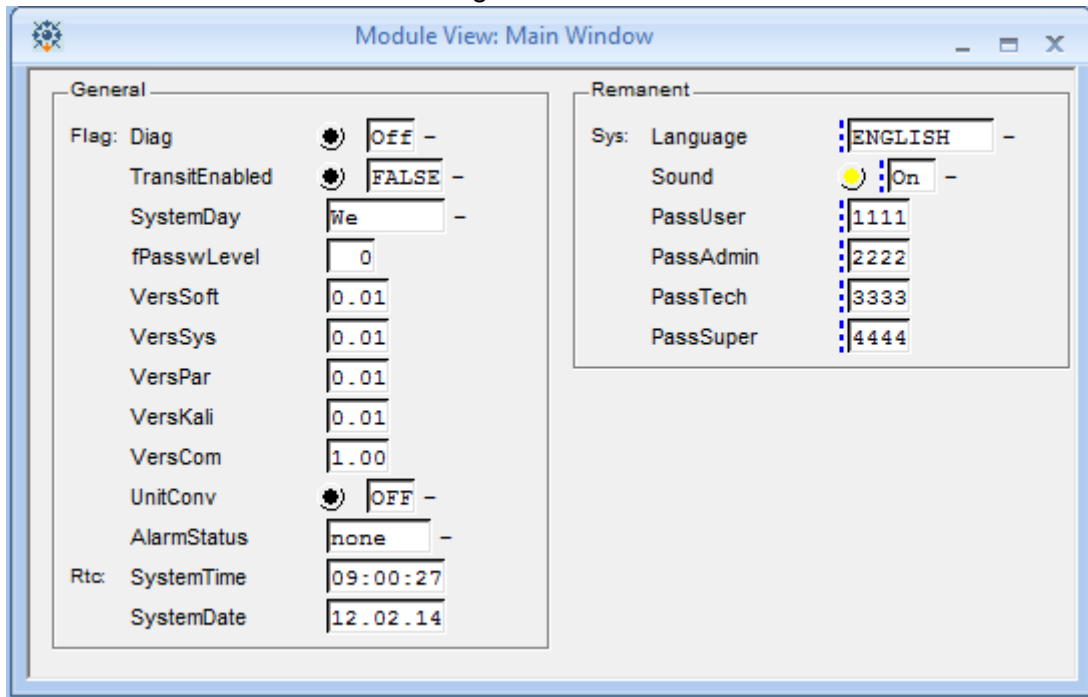


The screenshot displays the radCASE software interface. On the left, the 'Project' tree is expanded, showing a hierarchy: Project > SystemDef > TypeDef > TextDef > ModulDef > Tank. Under 'Tank', several components are listed: SIM\_Outlet, DO\_Valve, TIM\_Delay, TargetDelay, AI\_Level, TargetLevel, and State Charts. Below these, a 'System' folder contains 'Tank1' and 'Tank2'. On the right, the 'Tank2' configuration panel is open. It contains the following fields:

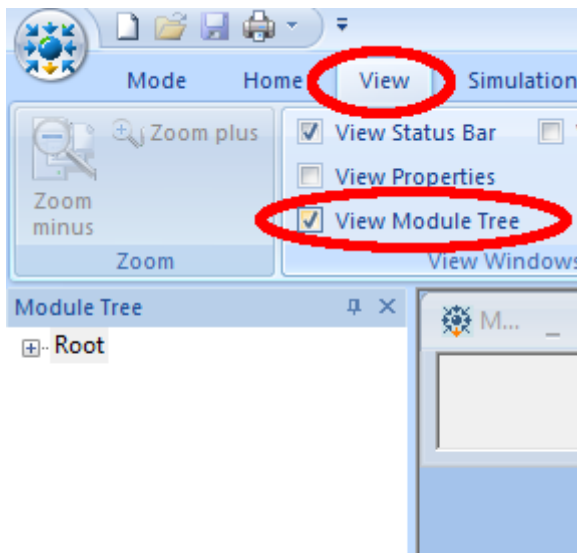
- Name:** A text box containing 'Tank2'.
- ID:** A text box containing 'Tank' with a dropdown arrow on the right.
- Submodule type:** A dropdown menu set to 'Submodule'.
- Multiplicity:** An empty text box.
- Controller-ID:** An empty text box.
- Description:** A text box containing '>>Tank2'.
- Info:** An empty text box.

### 4.6.3 Conclusion

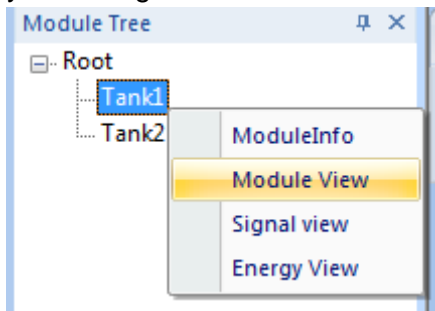
Hit <F4> in your project to create and start the simulation. When you open the Module View you will note all the elements we added are gone and there is no hint of the two submodules we created:



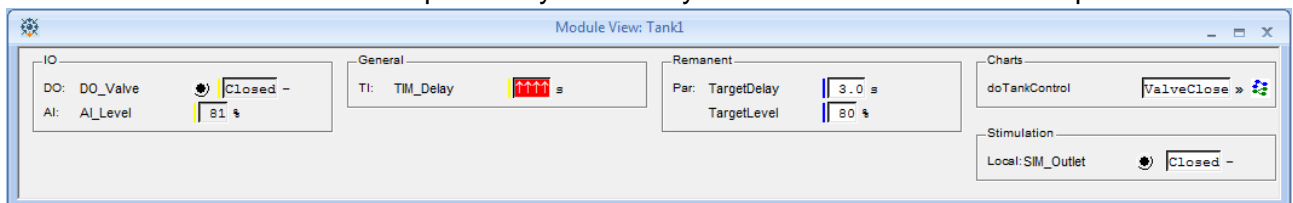
To see our submodules at the moment we have to open the module tree view, by selecting View->View Module Tree:



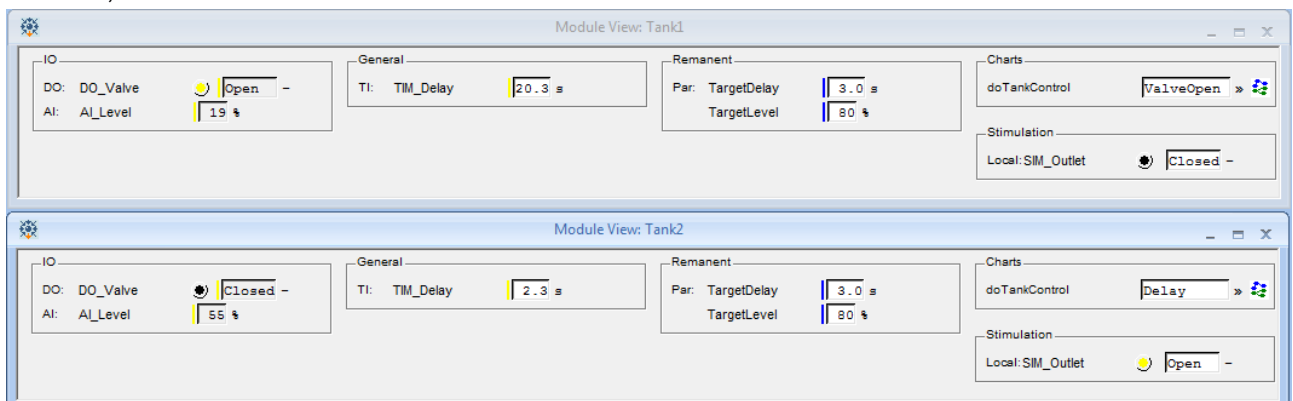
In the list, which just opened you can expand the submodule Root (this is the instance of the System module) and you will find our two submodules Tank1 and Tank2. <Right click> on Tank1 and you can again select **Module View**:



The module view of Tank1 will open and you will only see those elements that are part of our tank:



You can now open the Module View of Tank2 in the same way, and if you open/close the SIM\_Outlet on your tanks, you will notice both of your tanks are completely independent from one another, but the behavior of both tanks is the same.



## 4.7 Lesson H07: Libraries

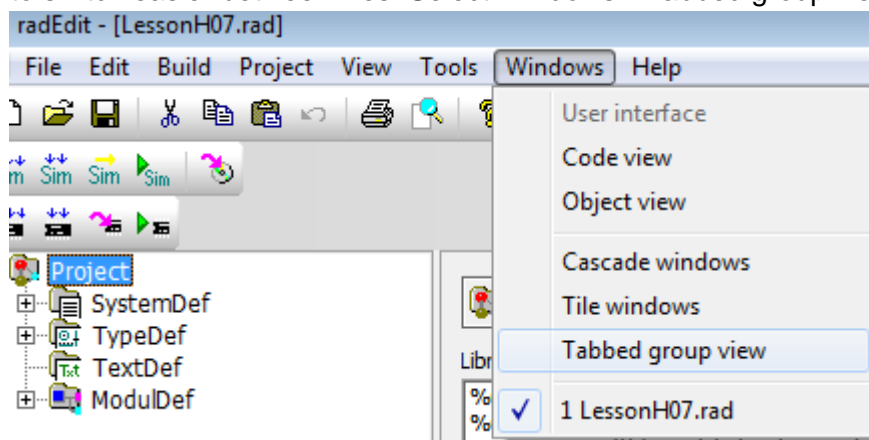
### 4.7.1 Goal

In this lesson you will learn, how to use libraries to not only reuse code within your project, but also reuse code in different projects.

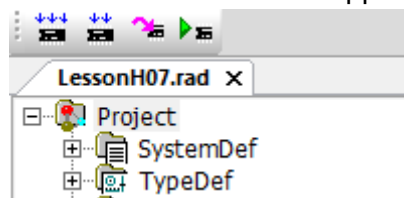
### 4.7.2 Content

You can start this lesson by using the project you created in lesson H06 or by opening the project LessonH07 that is delivered with your radCASE copy.

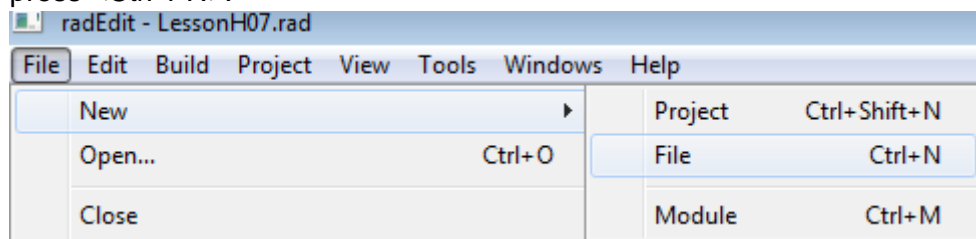
Because we will work with different files in this lesson, we should activate the Tab View in radEDIT, to switch easier between files. Select Windows->Tabbed group view:



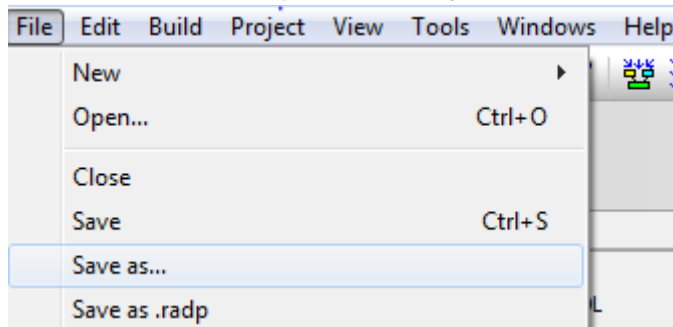
You will notice a tab bar appears above your project:



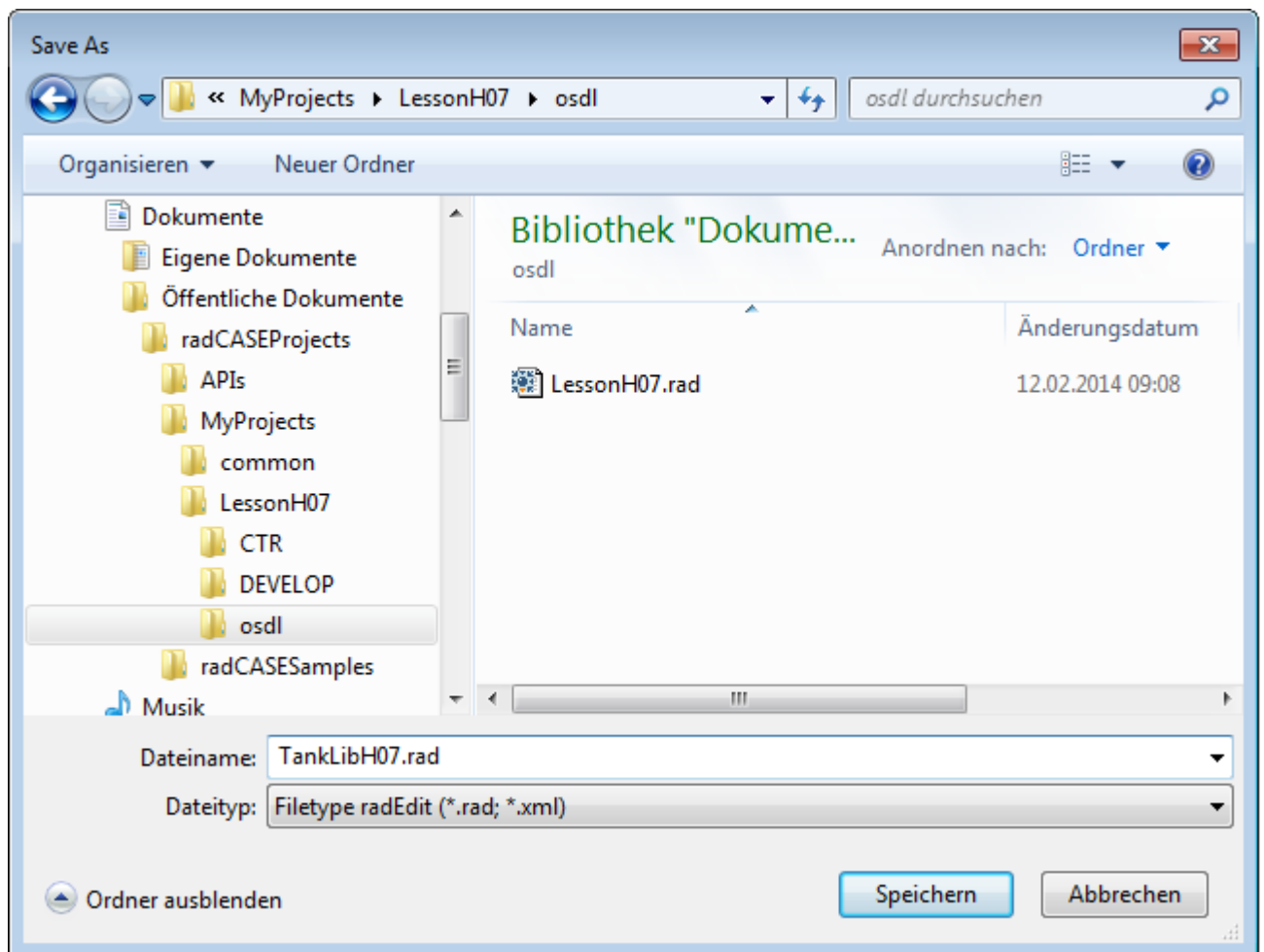
Now we will move our Tank module to a library, so we can use the Tank module in different projects. First we need to create a file for our library. Select File->New->New File from the menu or just press <Ctrl + N>:



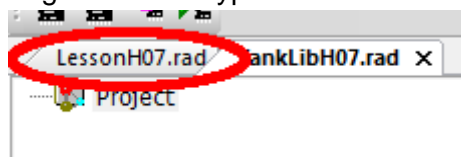
The new file will be opened directly and we want to give that file a name. Use File->Save As...



Put it in the same directory as our project and give it a name. For this lesson the name will be TankLibH07.rad:

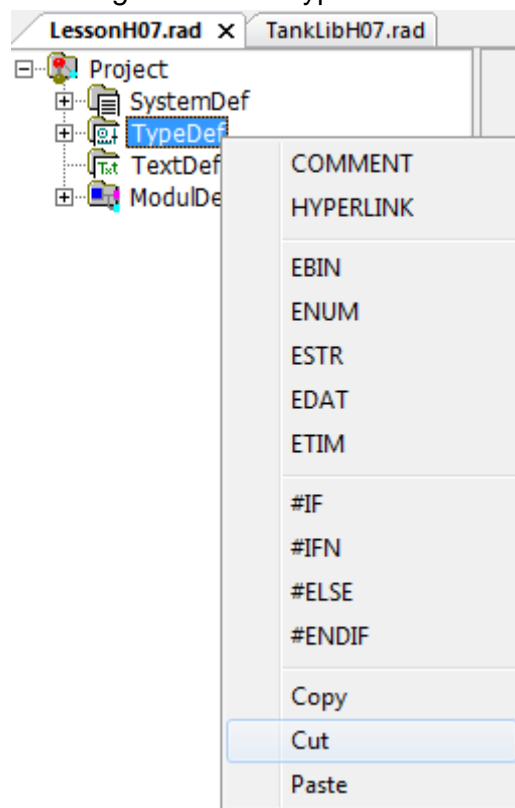


Now we want to move the Tank module and the Typedef we created to our library. First we will begin with the Typedef. Select the file LessonH07.rad:

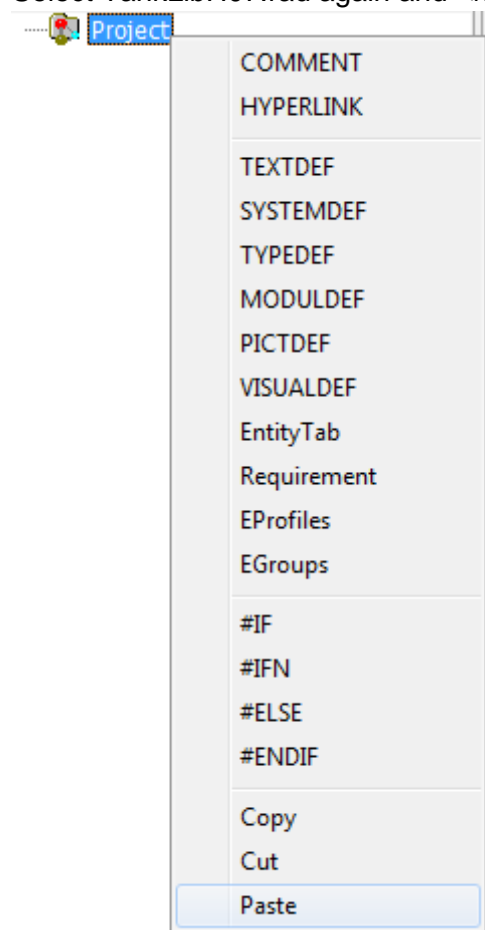




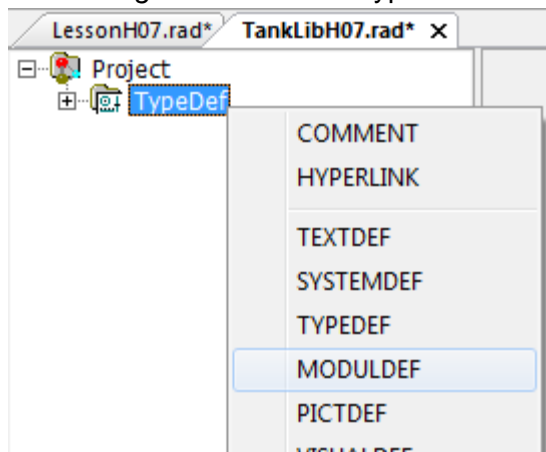
Now <right click> on Typedef and select **Cut**:



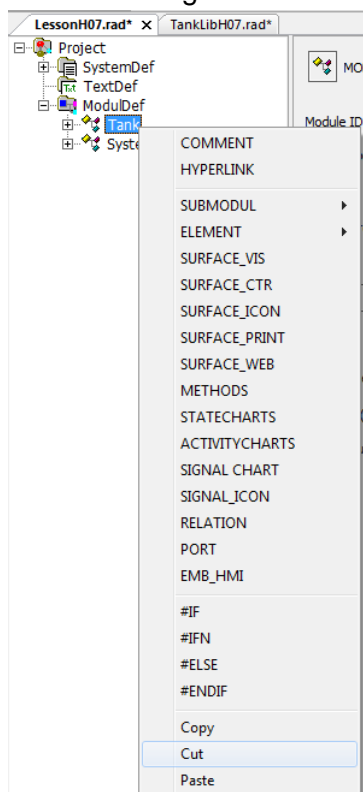
Select TankLibH07.rad again and <right click> on Project and select **Paste**:



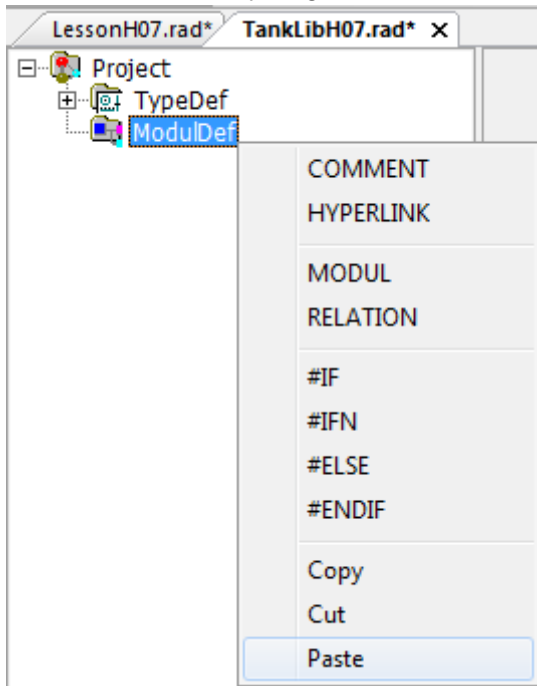
Now to move the module we have to first create the ModulDef node which contains all modules.  
<Shift + right click> on the TypeDef in the library and select **MODULDEF**:



Now switch again to LessonH07.rad and <right click> on the module tank and select **Cut**:

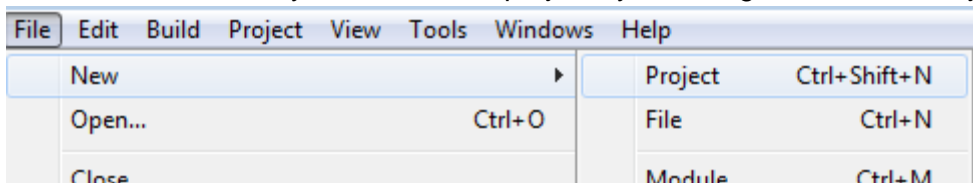


Switch to the library <right click> on the ModulDef and select **Paste**:

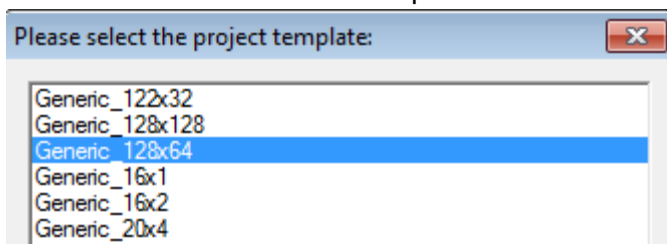


Now press <Ctrl + S> to save the current state of your library.

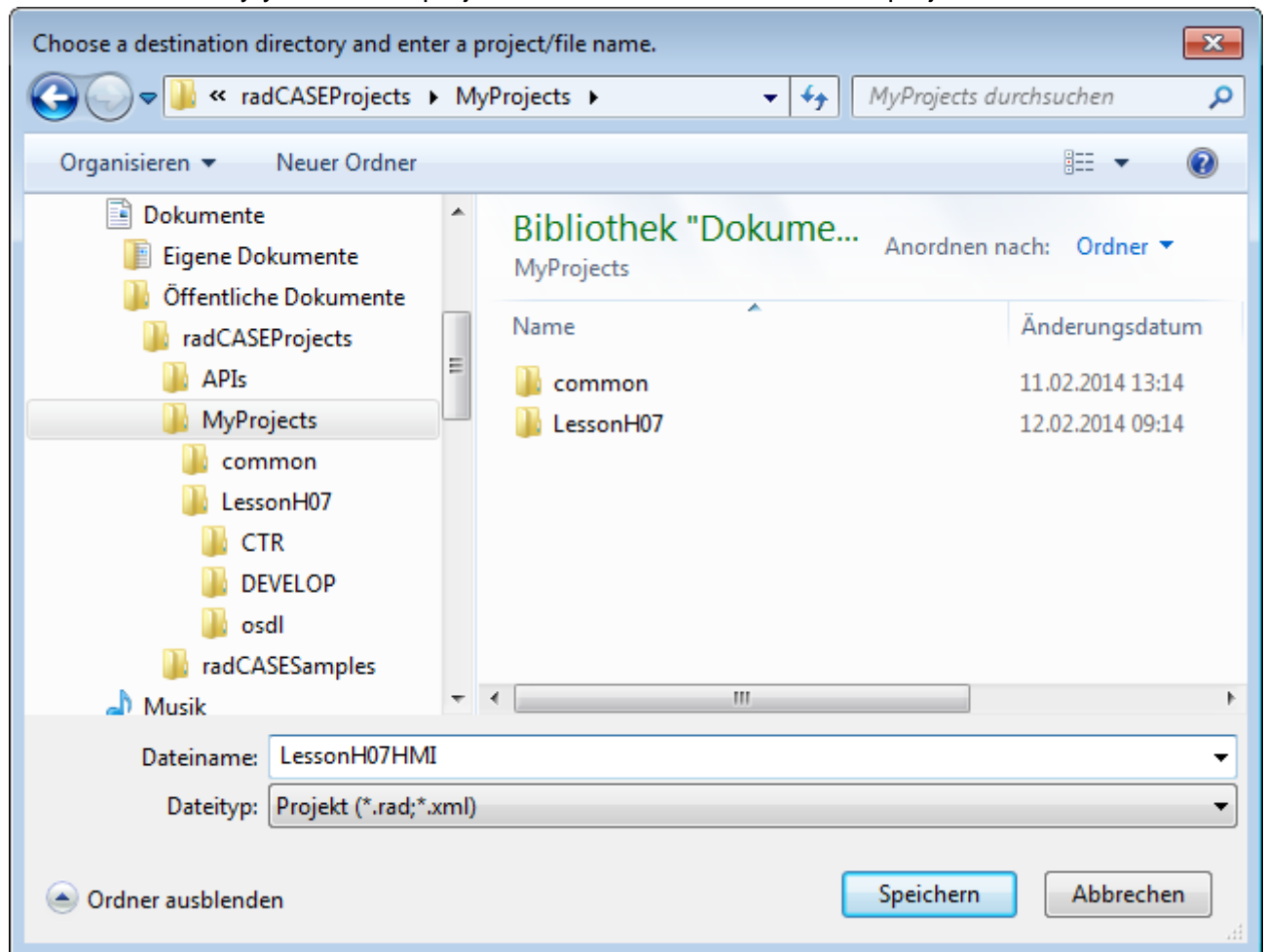
We could now integrate and use that library in our project, but instead we will create a new project, that will use that library. Create a new project by selecting File->New->Project:



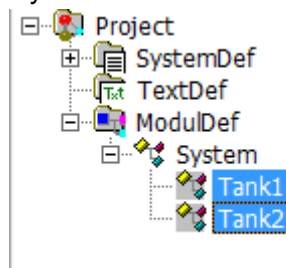
This time we will select the template 128x64:



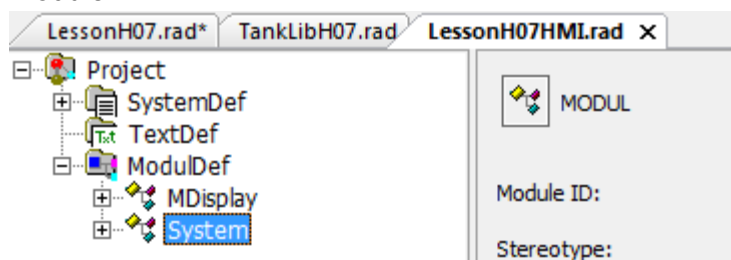
Select the directory your current project is located and name the new project LessonH07HMI:



Now select LessonH07.rad and select the submodules Tank1 and Tank2 (<Shift + Click>) in the System module:

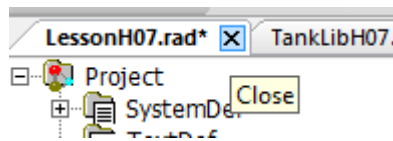


Press <Ctrl + X> to cut those tanks and select the file LessonH07HMI.rad and select the System module:



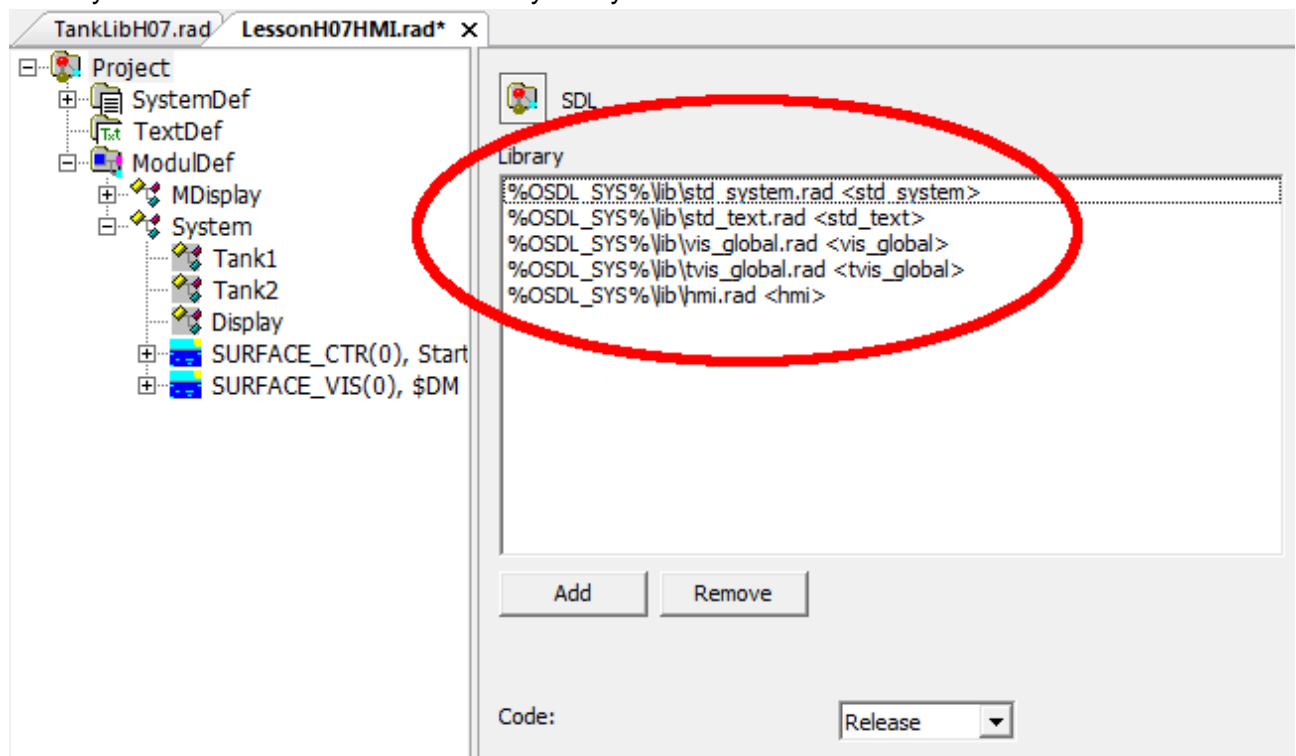
Press <Ctrl + V> to add the tanks to the system module of your new project.

Now select LessonH07.rad and close the project by selecting the “x” in the Tab:

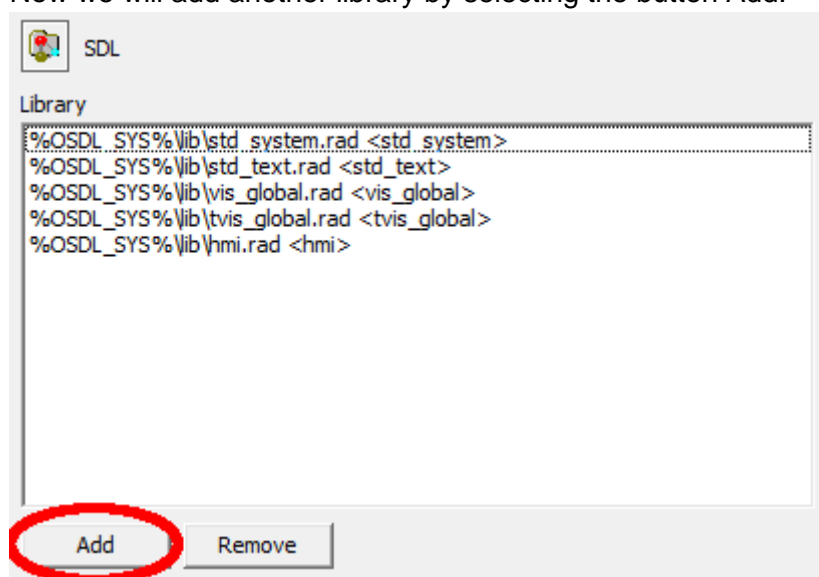


When asked if the changes should be saved you can either select Yes or No. We will not use that project anymore, so it is not important.

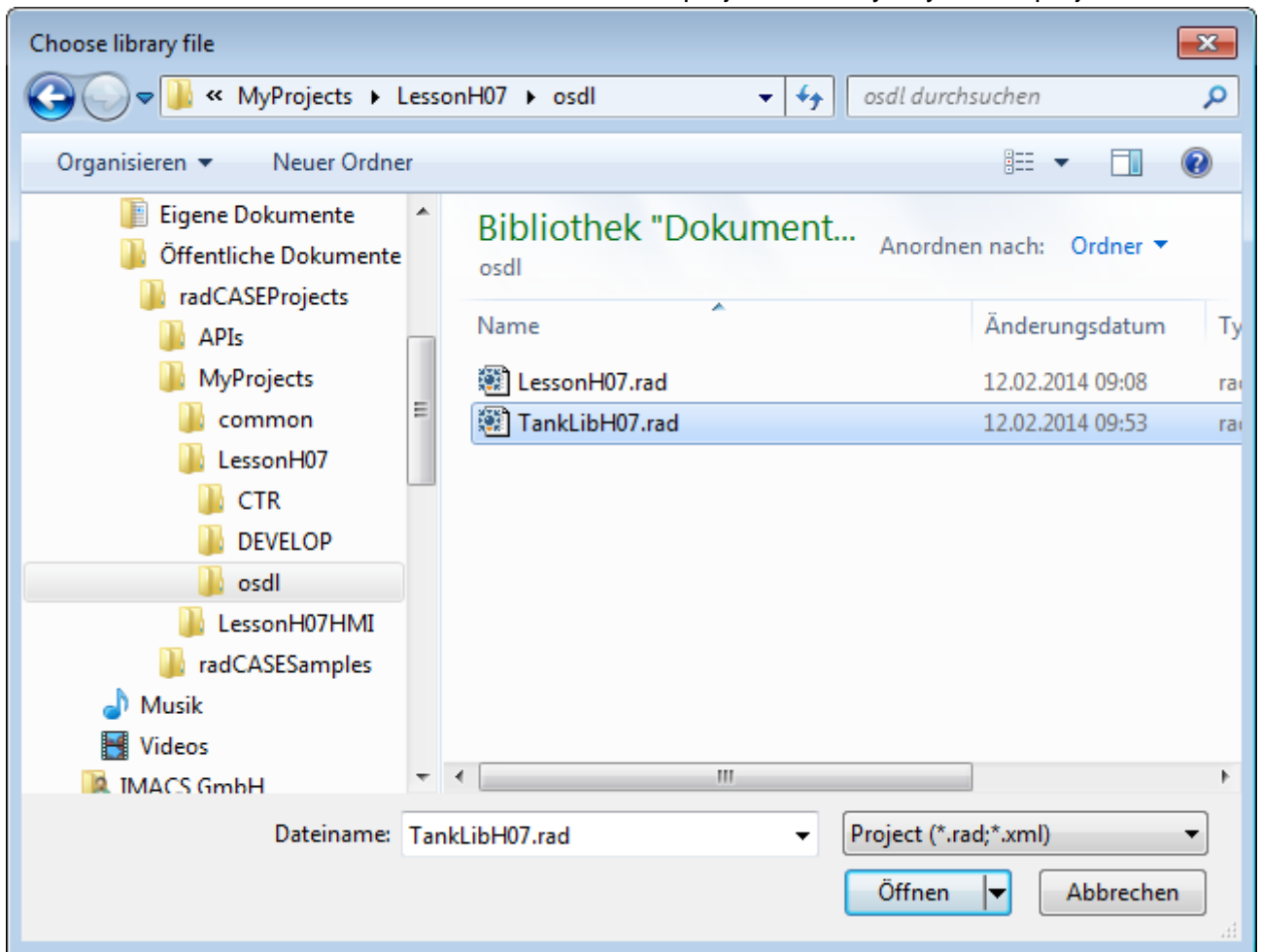
Now we will add our library to the project. This is fairly simple. Select the Project node and you will already see a list of included libraries in your system:



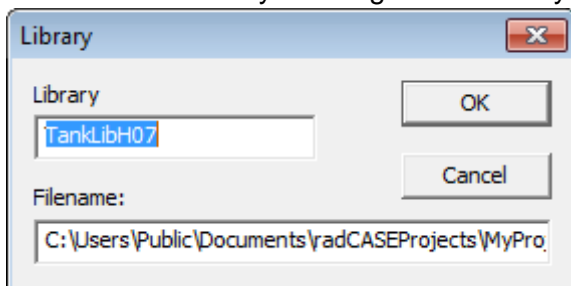
Now we will add another library by selecting the button Add:



Select the file TankLibH07.rad which is located in the project directory of your old project:



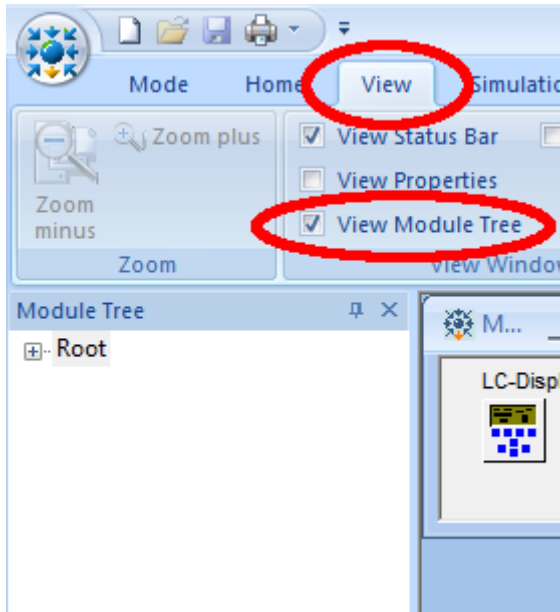
In the next window you can give that library another name, but in this case just press OK:



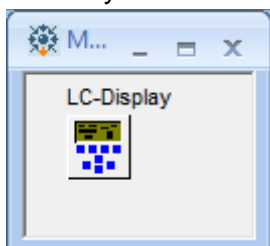
Confirm the next warning and you have successfully added your library to the project.

### 4.7.3 Conclusion

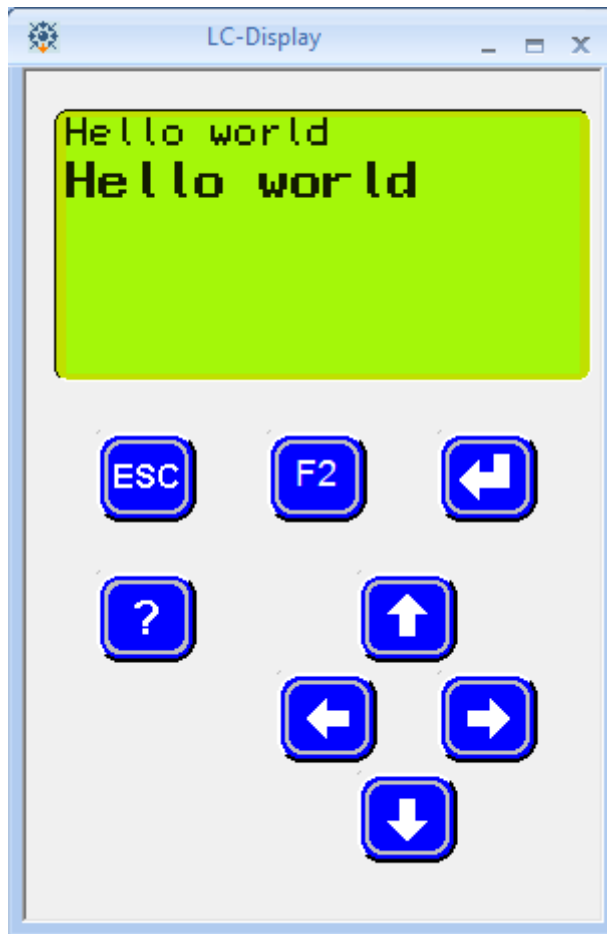
Hit <F4> to create and start the simulation. Again open the module tree view:



When you expand Root, you can see both your Tanks are there again. You can open the module view of your tanks and play with SIM\_Outlet, to see both tanks are working correctly. But as you can see on your main window, now you have another project with an HMI:



You can <click> on the Icon displayed in that main window and another window containing the display of a simulated HMI of the real hardware and some buttons with which keystrokes on the real hardware can be simulated is shown:





## 4.8 Lesson H08: Target HMI

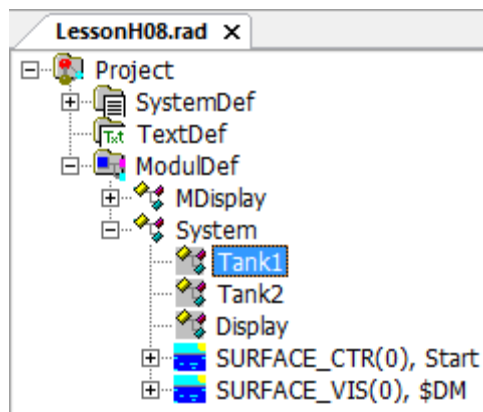
### 4.8.1 Goal

In this lesson you will learn the basics for creating HMI for displays on your hardware.

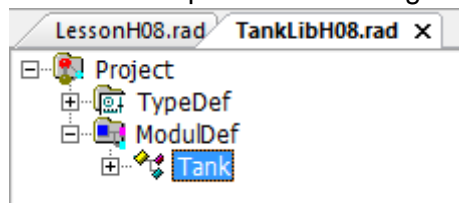
### 4.8.2 Content

You should start this lesson with the project LessonH08 which is delivered with your radCASE copy. In this project the project comment is added again, the tank library is moved into the project directory and a library containing some visual items was added to the tank library.

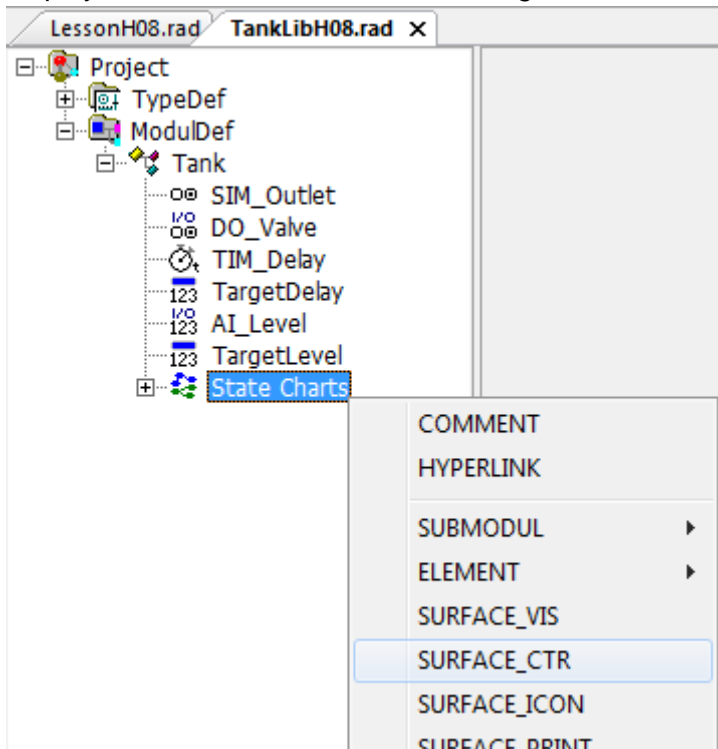
We want to add a menu for each of our tanks, which will allow to set the target fill level and see the current fill level. Because we can add this surface to the Tank module, each tank will already have such a menu. So we start by adding the menu, to the Tank module. Select one of your tank submodules:



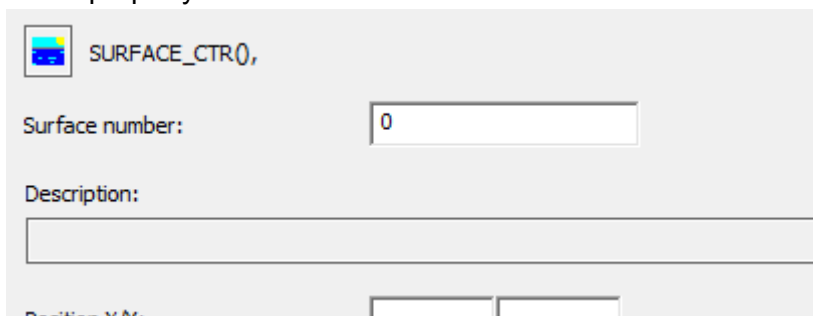
Now make a <double click> on one of the two tank submodules. This will jump you to the Tank module and open the according library, if the module is defined in a library:



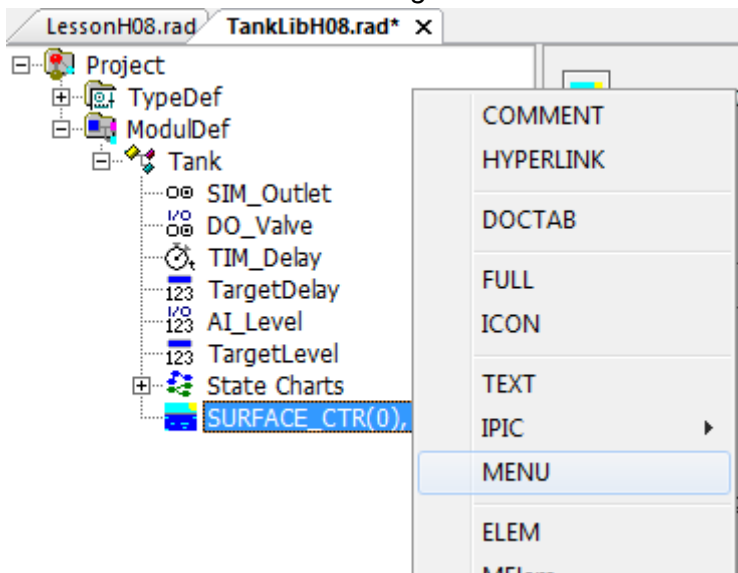
Now we add a Surface\_CTR, this is a surface which targets the controller and can be shown on the display on the real hardware. <Shift + right click> on the State Charts and select **SURFACE\_CTR**:



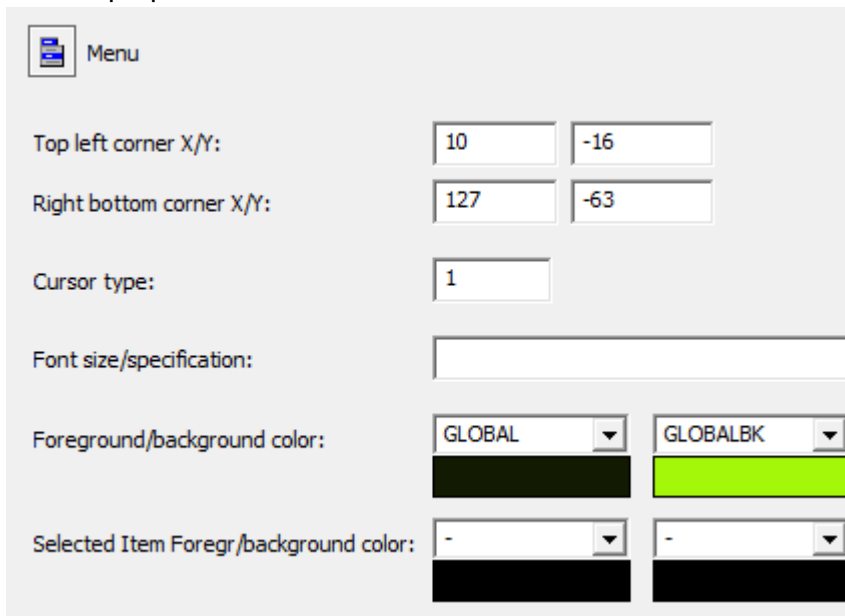
In the property editor set the Surface number to "0":



We will now add the menu. <Right click> on the Surface and select **MENU**:



Set the properties of the menu as shown below:

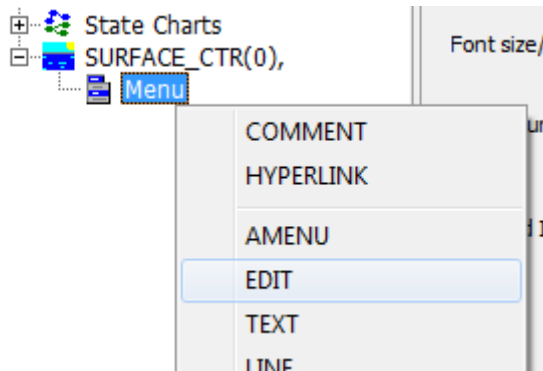


The image shows a 'Menu' properties dialog box. It contains the following fields and controls:

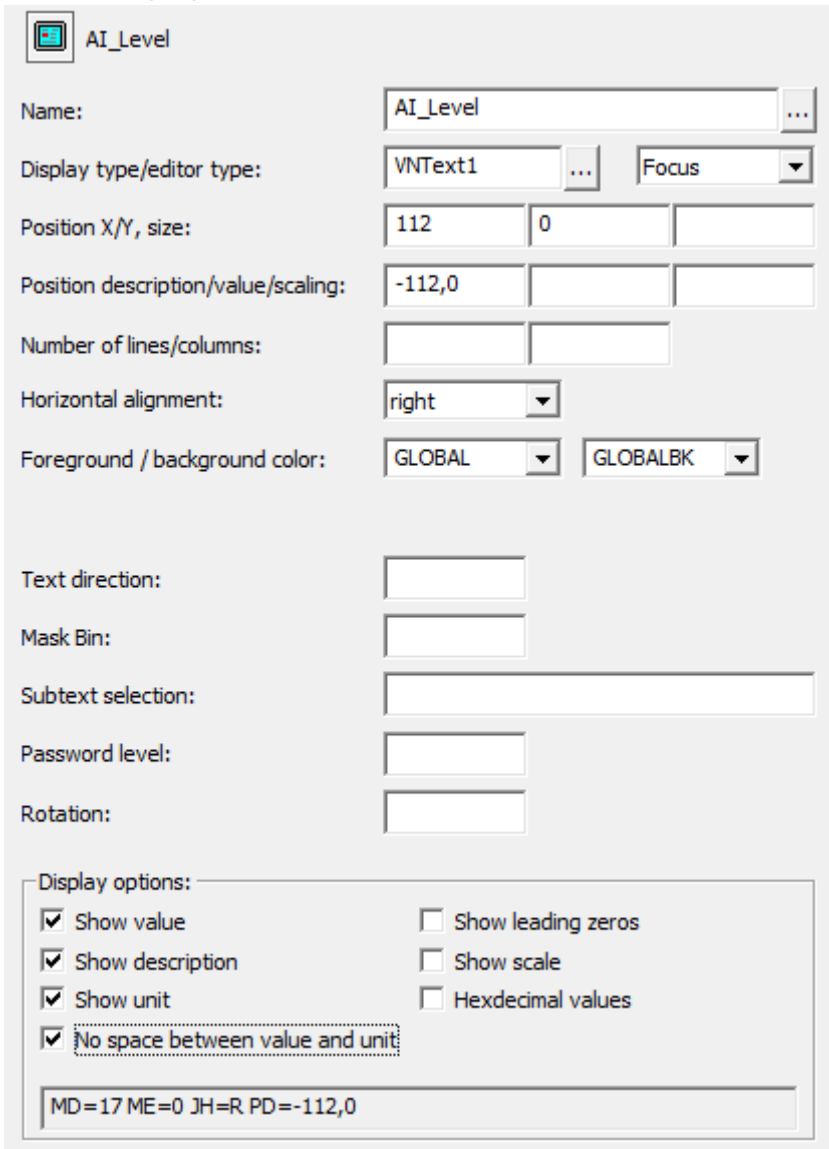
- Top left corner X/Y:** Two input boxes with values '10' and '-16'.
- Right bottom corner X/Y:** Two input boxes with values '127' and '-63'.
- Cursor type:** A single input box with the value '1'.
- Font size/specification:** A large empty text box.
- Foreground/background color:** Two dropdown menus. The first is set to 'GLOBAL' and shows a black color swatch. The second is set to 'GLOBALBK' and shows a bright green color swatch.
- Selected Item Foregr/background color:** Two dropdown menus, both set to '-' and showing black color swatches.

The corner positions determine the position and size of the menu. The menu takes nearly the whole space of the target display. We only reserved some space at the top for a Headline, which will be added later. Also there is some space reserved left of the menu for the cursor. Cursor type 1 sets a cursor which will appear left of the menu.

We will now add two elements as menu items. <Right click> on the menu and select **EDIT**:



Fill out the properties of the Edit as follows:



**AI\_Level**

Name:

Display type/editor type:

Position X/Y, size:

Position description/value/scaling:

Number of lines/columns:

Horizontal alignment:

Foreground / background color:

Text direction:

Mask Bin:

Subtext selection:

Password level:

Rotation:

Display options:

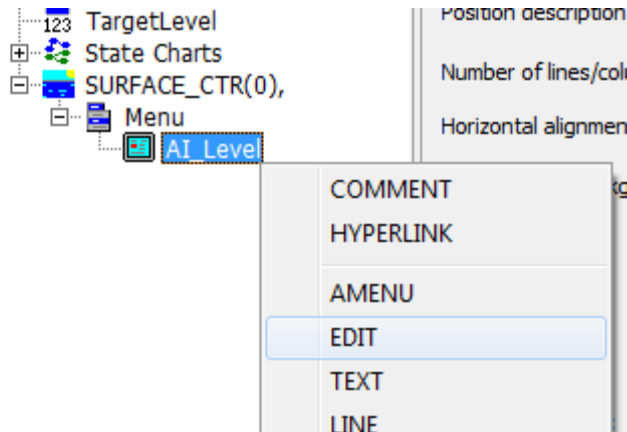
- ☒ Show value
- ☒ Show description
- ☒ Show unit
- ☒ No space between value and unit
- ☐ Show leading zeros
- ☐ Show scale
- ☐ Hexdecimal values

MD=17 ME=0 JH=R PD=-112,0

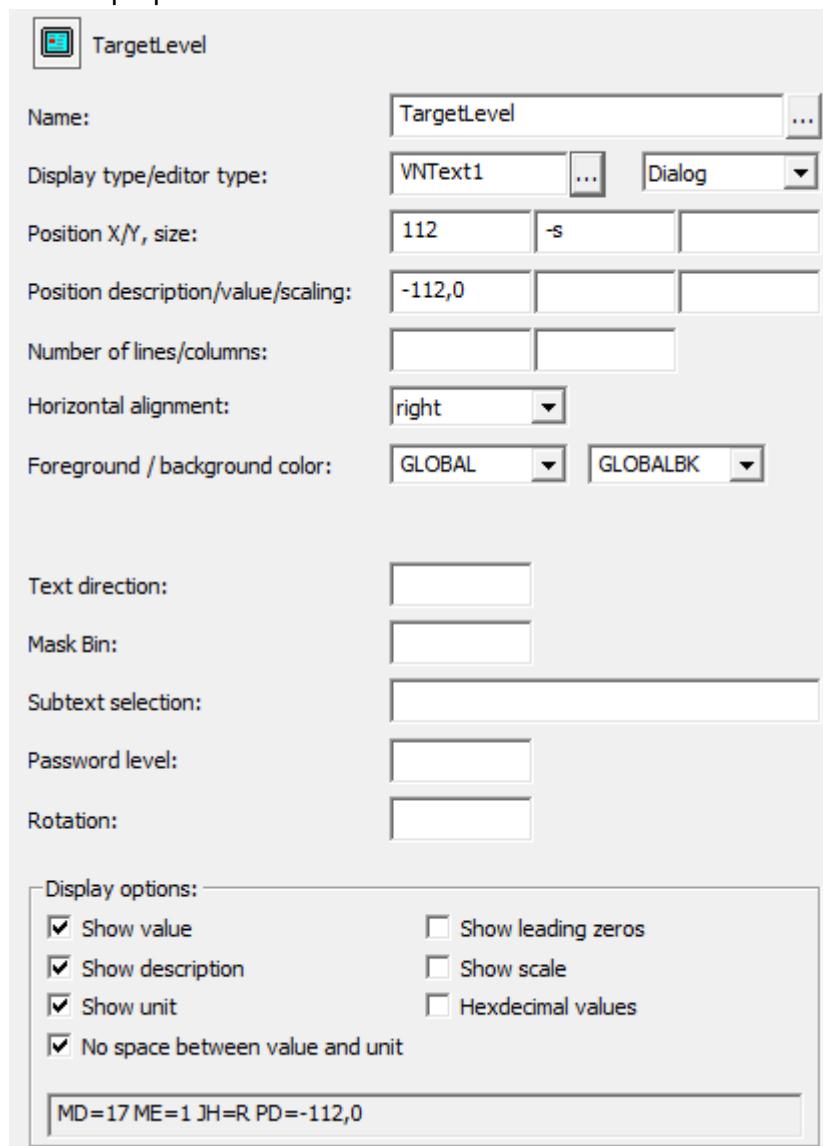
The **Name** determines the Element to show. The **Display options** at the bottom determine that the element should be displayed with value, description and unit and that there should not be space between the value and unit.

The **Display type** determines the element to be shown as plain text. The **editor type** determines that it is possible to select the menu line with that element, but it is not possible to edit the value. The position positions the value of the element relative within the menu. We want it to be displayed at the right edge of the screen. The description is positioned relative to the value. We want that to be left of our value at the left edge of the menu.

Add a second Edit using <Shift + right click> on the edit of AI\_Level we just created and selecting **EDIT**:



Set the properties as follows:



**TargetLevel**

Name: TargetLevel

Display type/editor type: VNTText1 Dialog

Position X/Y, size: 112 -s

Position description/value/scaling: -112,0

Number of lines/columns:

Horizontal alignment: right

Foreground / background color: GLOBAL GLOBALBK

Text direction:

Mask Bin:

Subtext selection:

Password level:

Rotation:

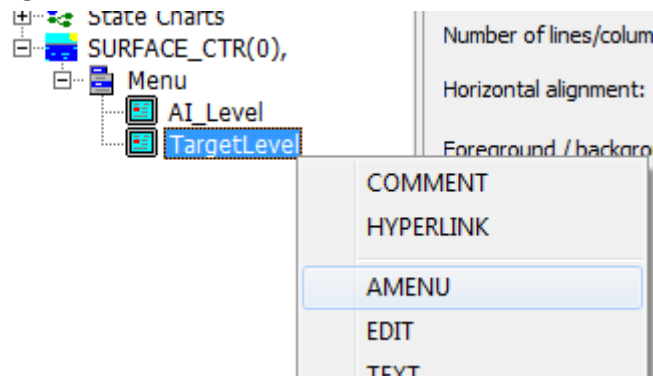
Display options:

- ☒ Show value ☐ Show leading zeros
- ☒ Show description ☐ Show scale
- ☒ Show unit ☐ Hexdecimal values
- ☒ No space between value and unit

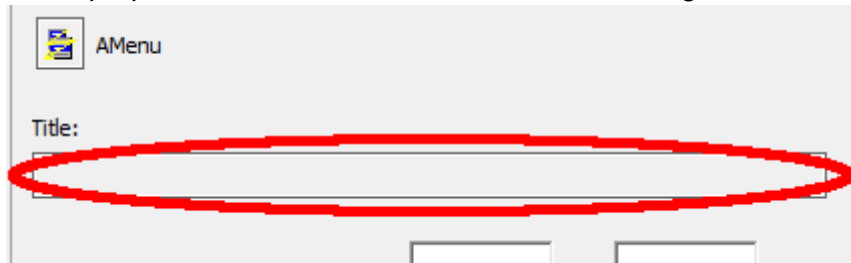
MD=17 ME=1 JH=R PD=-112,0

This is mostly the same as for AI\_Level. We have another **editor type**, which means to enable editing of the element using a dialog. Also the positioning uses -s. This is a relative positioning that can only be done in menus. This positions the menu line below the last line, using the font size of the previous entry.

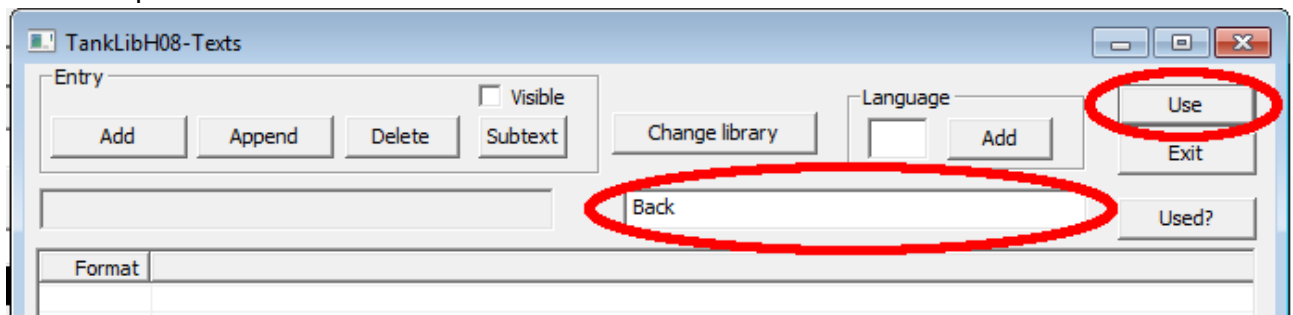
Now we want to be able to exit the surface, if we are in that surface. So we will add a menu entry, that will trigger Actions. *<Shift + right click>* on the edit for the element TargetLevel and select **AMENU**:



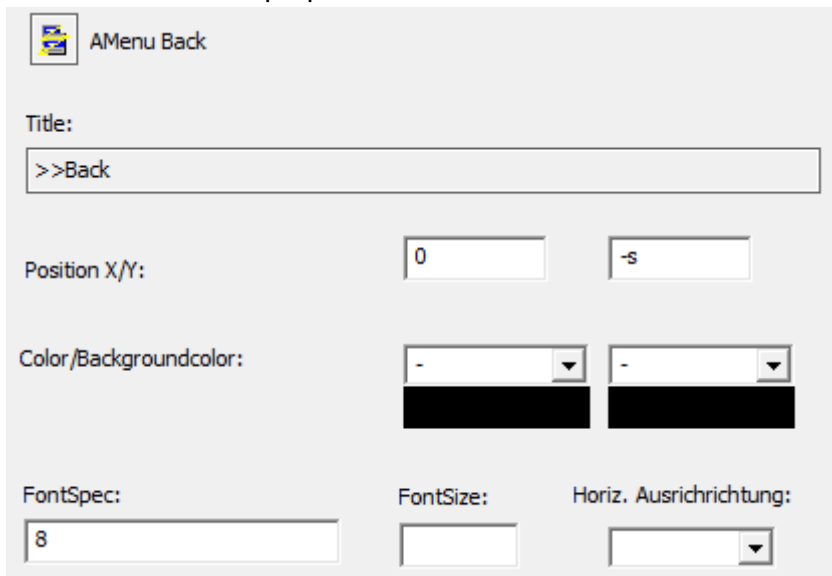
In the properties of the AMenu *<click>* on the rectangle below Title:



This will open a window. Enter the text "Back" like below and *<click>* on Use:



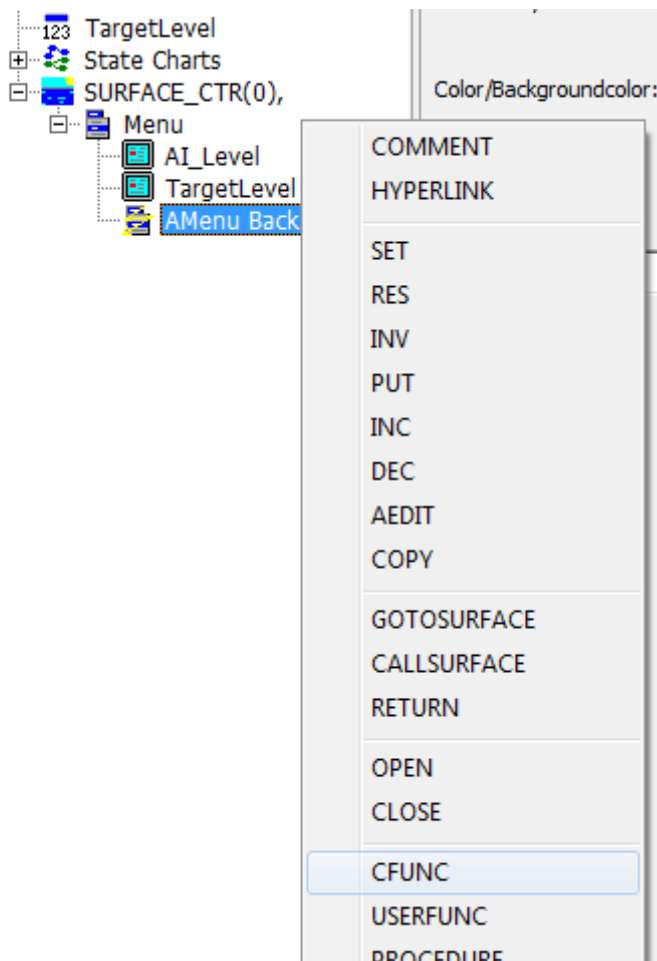
After this fill out the properties as shown below:



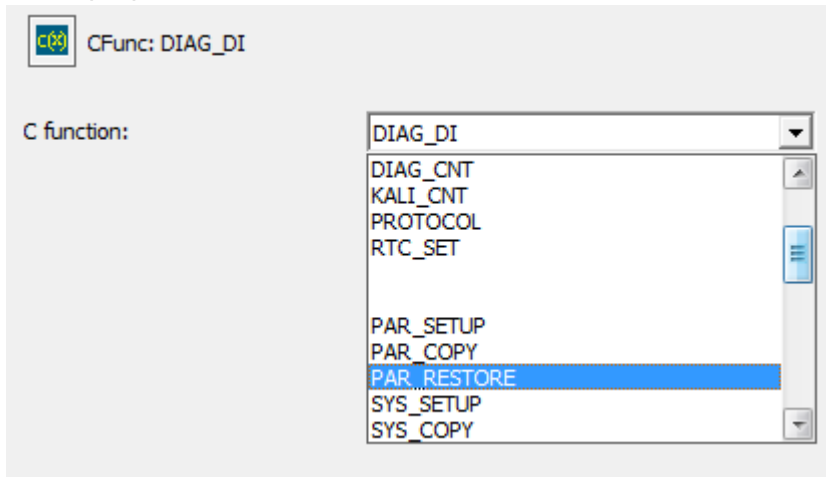
The screenshot shows the 'AMenu Back' properties dialog box. It has a 'Title' field containing '>>Back'. The 'Position X/Y' fields are '0' and '-s'. The 'Color/Backgroundcolor' fields are both set to a black color swatch. The 'FontSpec' field is '8'. The 'FontSize' field is empty. The 'Horiz. Ausrichtung' field is a dropdown menu.

The title we entered is the text shown for the menu entry. The rest will position the entry and specify the font size.

Now we want to add the actions that will be triggered when selecting that menu entry. *<Right click>* on the AMenu and select **CFUNC**:



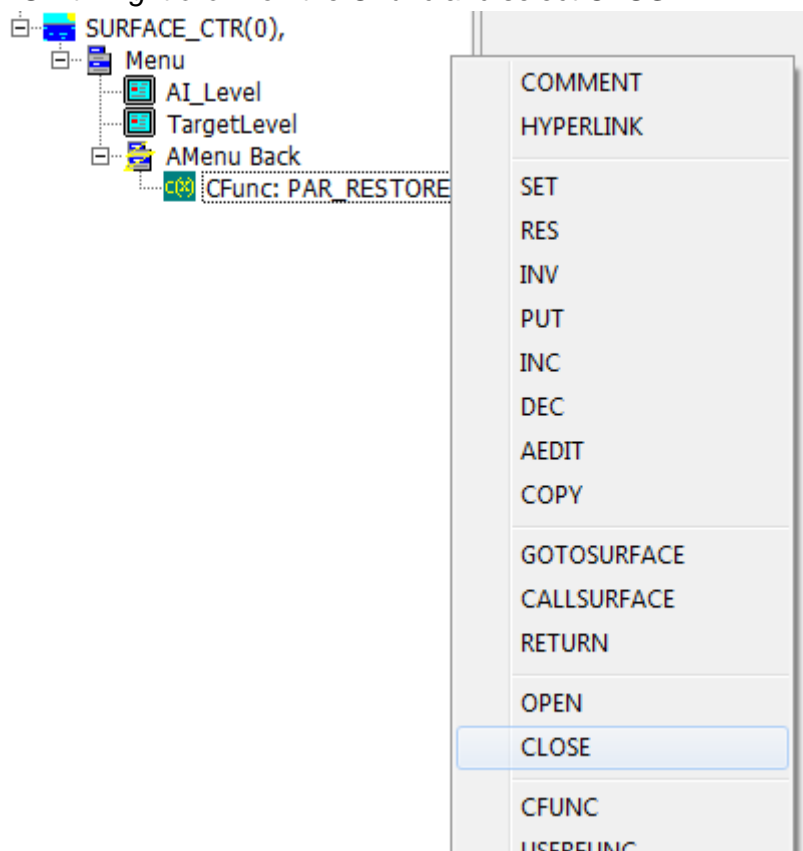
In the properties of the **CFUNC** select **PAR\_RESTORE**:



This is a predefined function of radCASE. Because the TargetLevel is saved in a variable which will be saved in a permanent memory, the value is not edited directly but within an edit copy.

PAR\_RESTORE submits the value of that edit copy to the permanent memory, which is used by the process. So as long as the value is not submitted by PAR\_RESTORE the process will still work with the old value.

Now <Shift + right click> on the CFunc and select **CLOSE**:

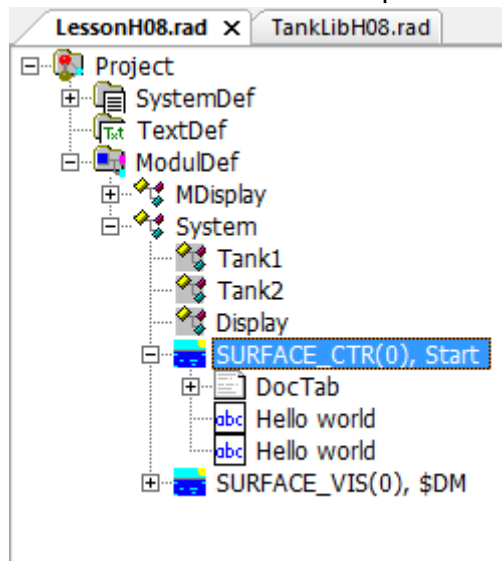


This will leave the surface and go back to whichever surface was active before the current surface was opened.

Press <Ctrl + S> to save the library. We have finished the surface within our module.

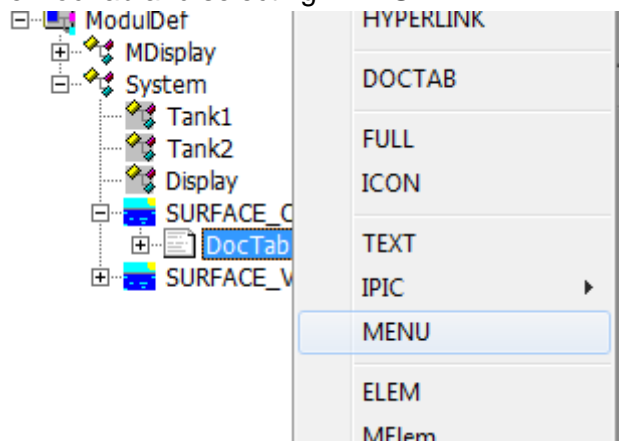


We still need to open the surface we have just created within our project. Because the System module is the entry point of our project, the entry surface is also defined within the System module. The SURFACE\_CTR(0) of the System module will always be shown at the start of our application. Switch to LessonH08 and expand SURFACE\_CTR(0) of the System module:

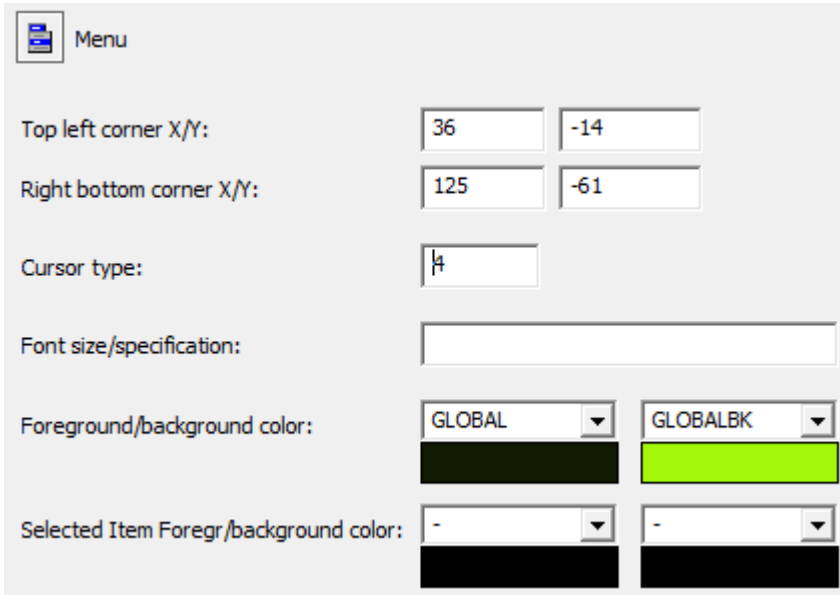


You can see, there is already some content in that surface. There is a DocTab, this is an item used for documentation generation which is not covered in this tutorial. The other two entries both display the text Hello world. This is where the text you saw on the target display in the Conclusion of the last lesson came from.

Select the both texts and delete them using **<DEL>**. After that add a menu by **<Shift + right click>** on the DocTab and selecting **MENU**:

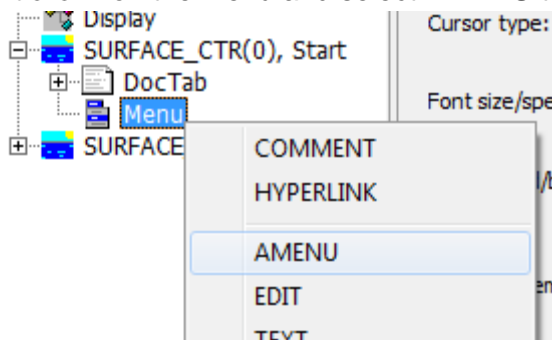


Set the properties of the Menu to the following values:

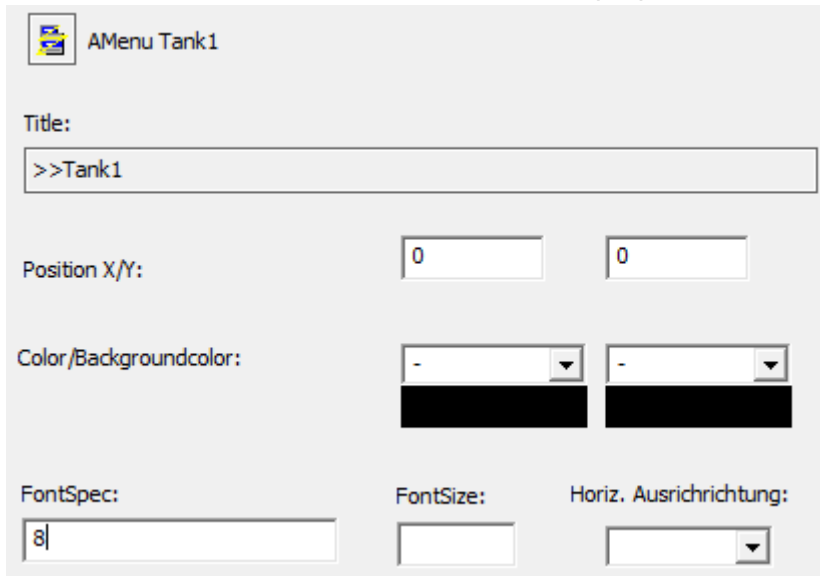


Again we use most of the space of the target display and reserve some space for a heading. This time we reserve some more space on the left for a picture, which will be added later. We don't need space for a cursor this time, because the cursor type 4 marks the currently selected line by inverting it.

<Right click> on the menu and select **AMENU** to add a menu entry, which will trigger some actions:



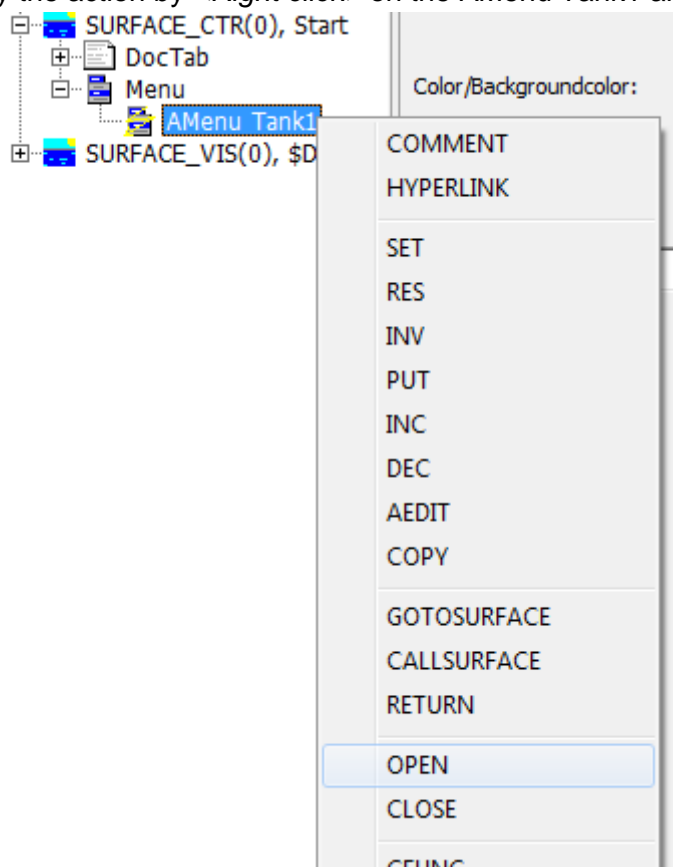
The title can be set as shown before. Set the properties of the AMenu as follows:



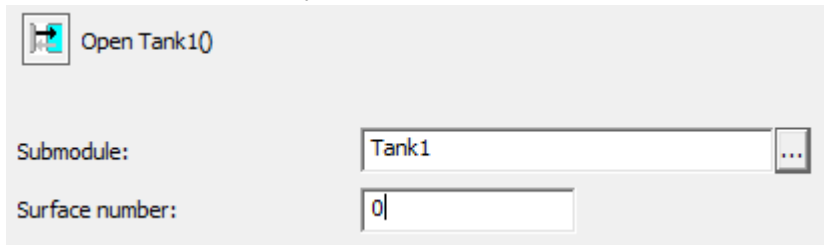
The screenshot shows the 'AMenu Tank1' properties dialog box. It contains the following fields and controls:

- Title:** A text box containing '>>Tank1'.
- Position X/Y:** Two numeric input boxes, both containing '0'.
- Color/Backgroundcolor:** Two dropdown menus, both showing a black color swatch.
- FontSpec:** A text box containing '8'.
- FontSize:** An empty numeric input box.
- Horiz. Ausrichtung:** A dropdown menu with a downward arrow.

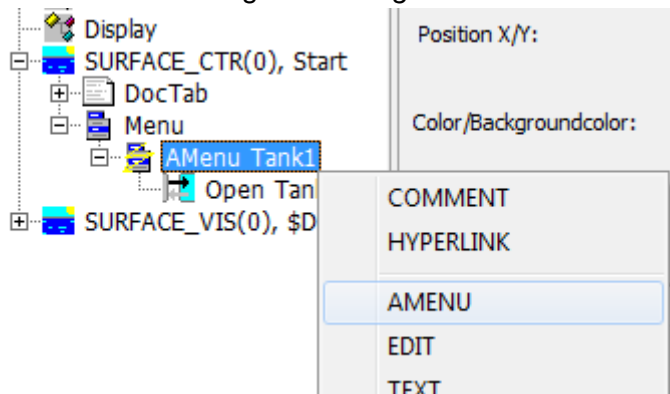
Specify the action by <Right click> on the AMenu Tank1 and select **OPEN**:



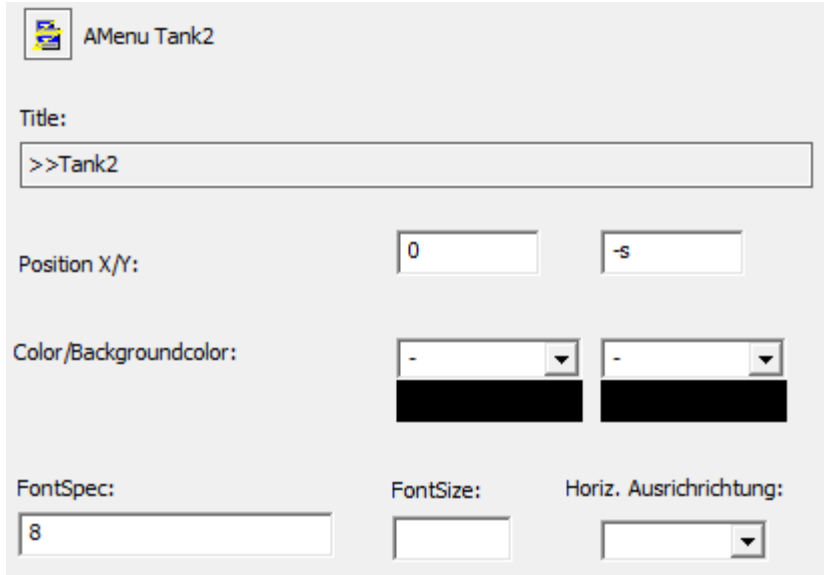
You can select a submodule by clicking on the button “...” or by simply entering the name of the submodule in the text field. Select submodule Tank1 and Surface number 0, to select the surface we created in the library.



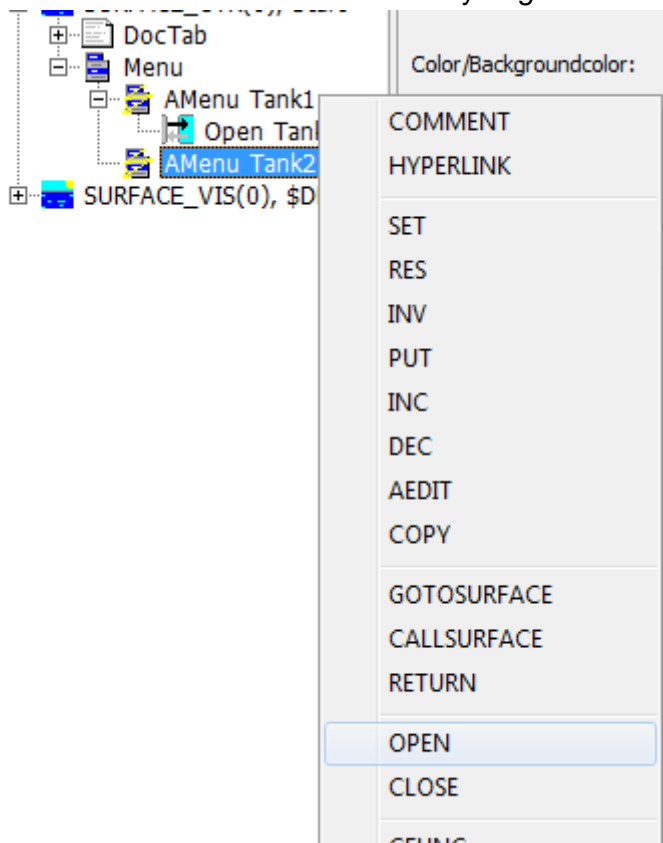
Add another AMenu using <Shift + right click> on the AMenu Tank1 and selecting AMenu:



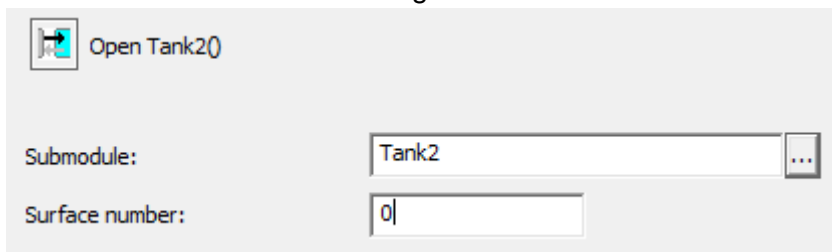
Set the properties of the AMenu as shown below:



Now add the action OPEN to the AMenu by <right click> on the AMenu and selecting **OPEN**:

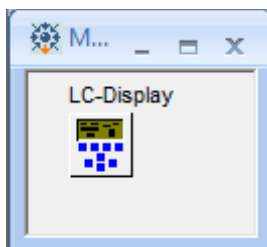


Select submodule Tank2 and again surface number 0:

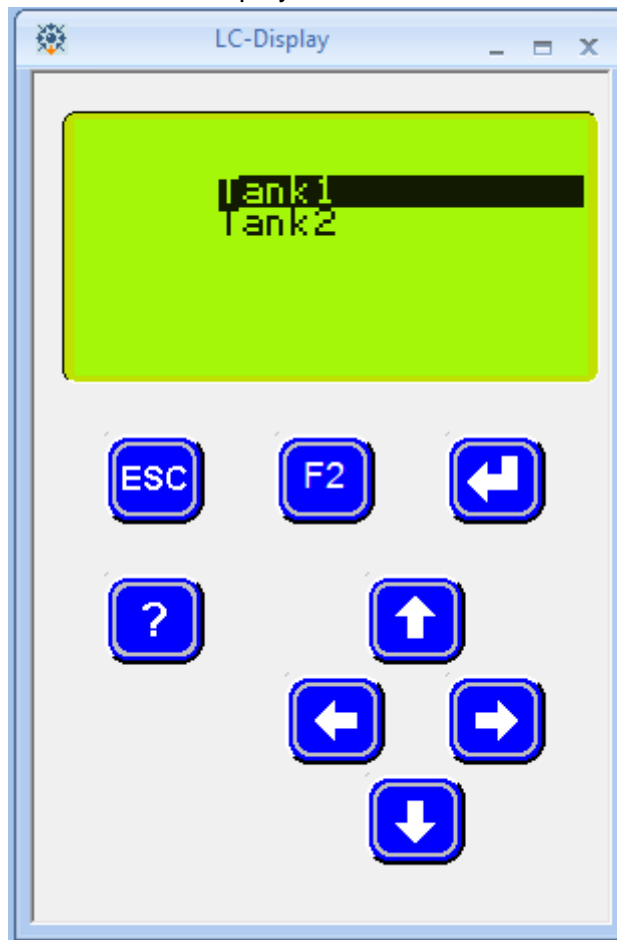


### 4.8.3 Conclusion

Hit <F4> to create and start the simulation. <Click> on the Icon in the main window:



The simulated display of the hardware will be opened and show the menu, we just created:



You can now use your keyboard and navigate using *<Up>*, *<Down>* and *<Return>* or by using the keys in the window shown above.

When you press <Return> on one of the tanks you will see the menu, we created for the Tank module:



You can't edit AI\_level, but you can set the TargetLevel to a different value. Also you can use the SIM\_Outlet in the module view of the tanks and you will see, the values on the controller display will also update to the correct values.

## 4.9 Lesson H09: Descriptions / Placeholders

### 4.9.1 Goal

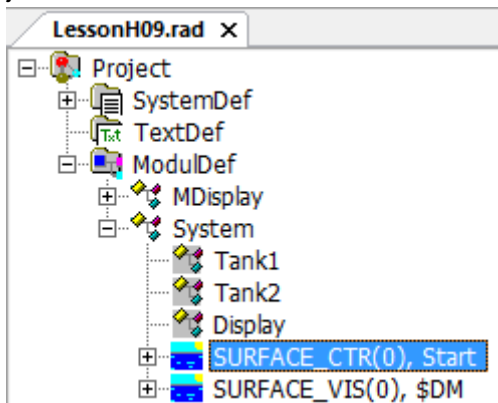
In this lesson you will learn how to set and use descriptions in your project. You will also learn the use of text placeholders.

### 4.9.2 Content

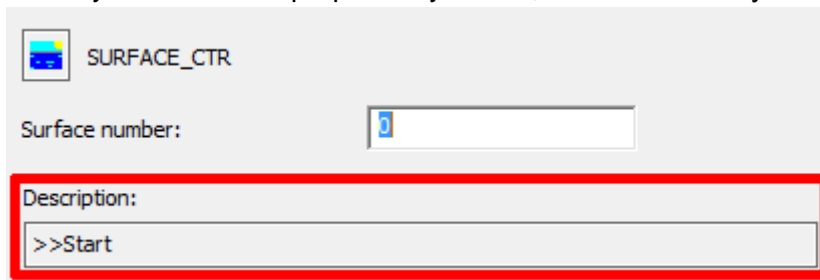
You should start this lesson with the project LessonH09 which is delivered with your radCASE copy. In this project the HMI generated in the last lesson was extended by some graphical items and by completing the elements shown in the tank library.

In this lesson we will add a description to the main surface of the project and use that description as a heading in the surface. We will also add descriptions to the two tank submodules and use that description within the menu to select a tank. At last we will also set the description of an element and show how this is already used within the tank surface we created.

To add a description to the surface, select SURFACE\_CTR(0) in the System module of your project:



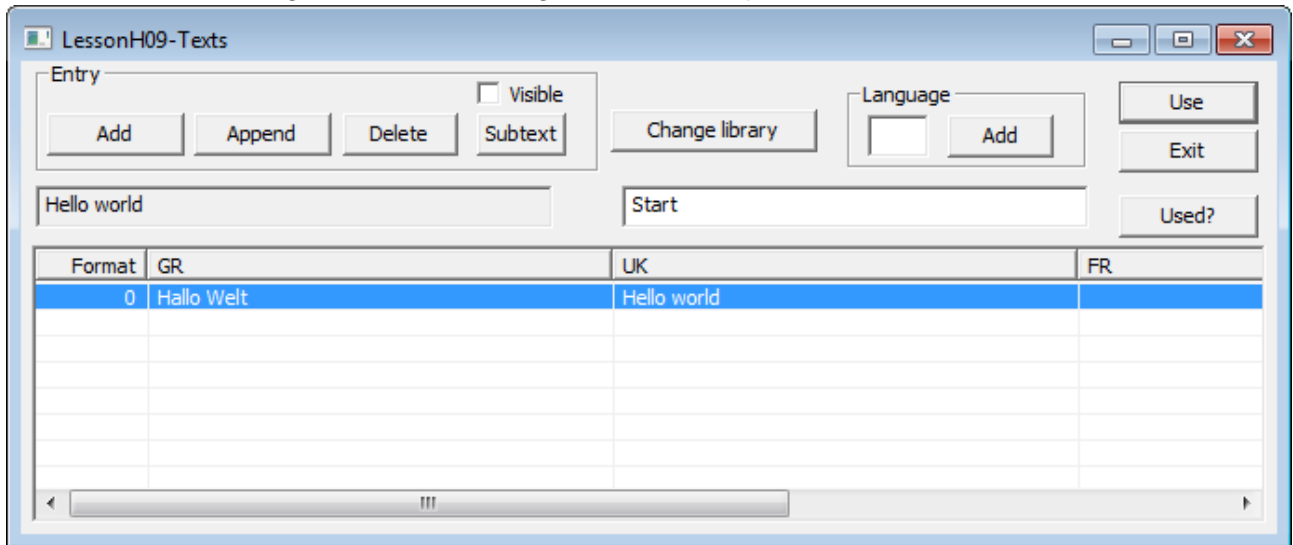
When you look at the properties you see, there is currently already the Description "Start":



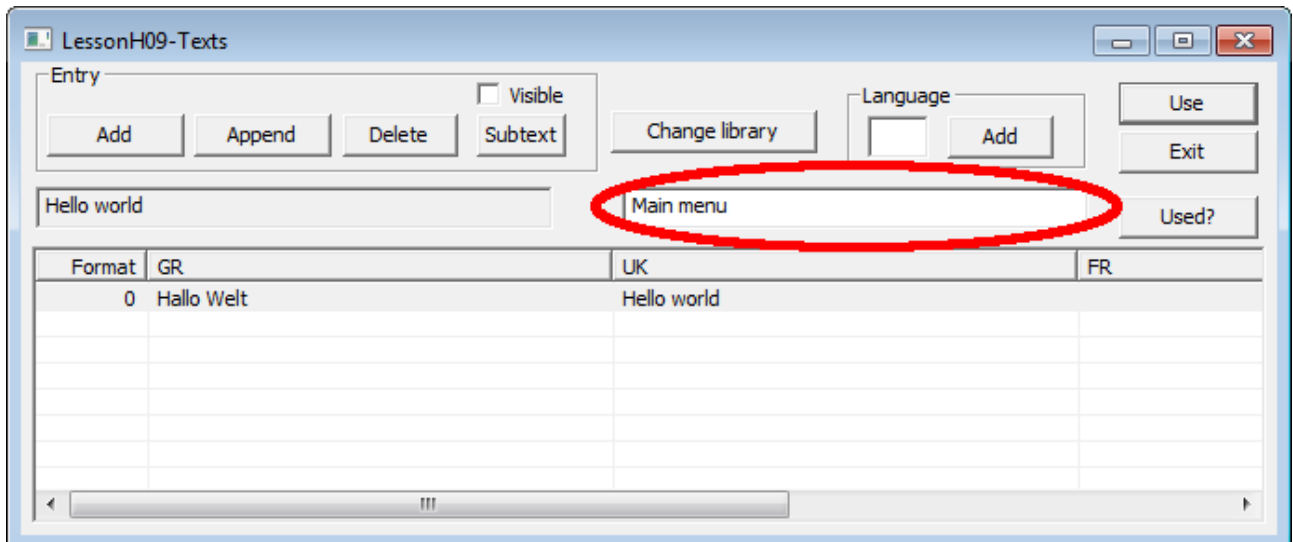
You can see there is a rectangle around the description instead of a normal text input box. This means you can use a text from the text table at this location.



<Click> on the rectangle and the following windows will open:



At the bottom you can see the text table. There you can enter texts for multiple languages. When using the text table a further advantage is, that if a text is used at multiple places and you want to change that text, you can do this in one central place, instead of changing all occurrences. In this tutorial however we will not use the text table, but single language fixed texts instead. Those fixed texts can be entered in the text input field in this window. Change the text of the description to “Main menu”:



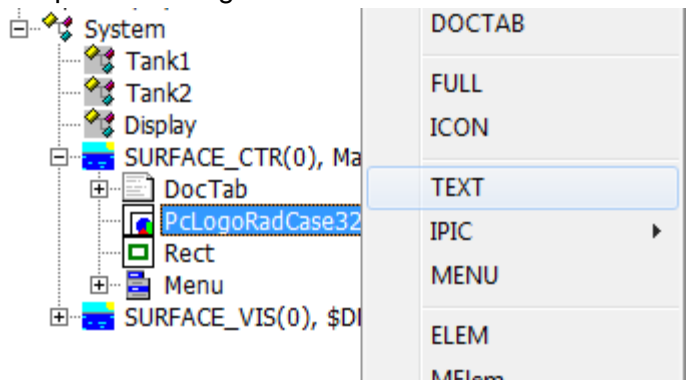
You can confirm the text by clicking Use in the upper right corner. In the rest of the lesson, we will refer to this text input method as entering a fixed text.

The text will now look like:

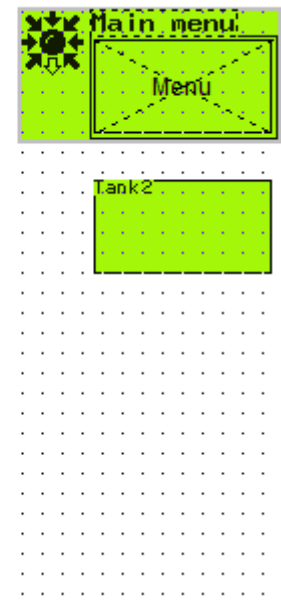
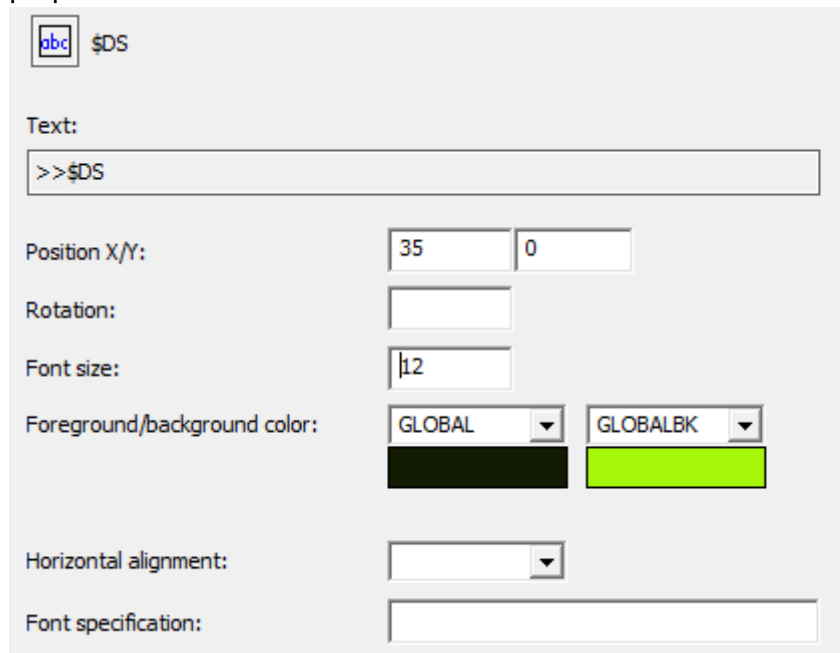


The “>>” in front of the text marks it as a fixed text.

Now we will add this description as text into the surface. Expand the surface and <Shift + right click> on the picture PcLogoRadCase32 and select **TEXT**:

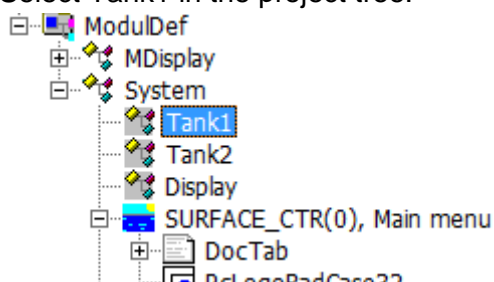


This will add the text between the picture and the rectangle. Set the fixed text to “\$DS” and fill in the properties as shown below and hit <F5>:

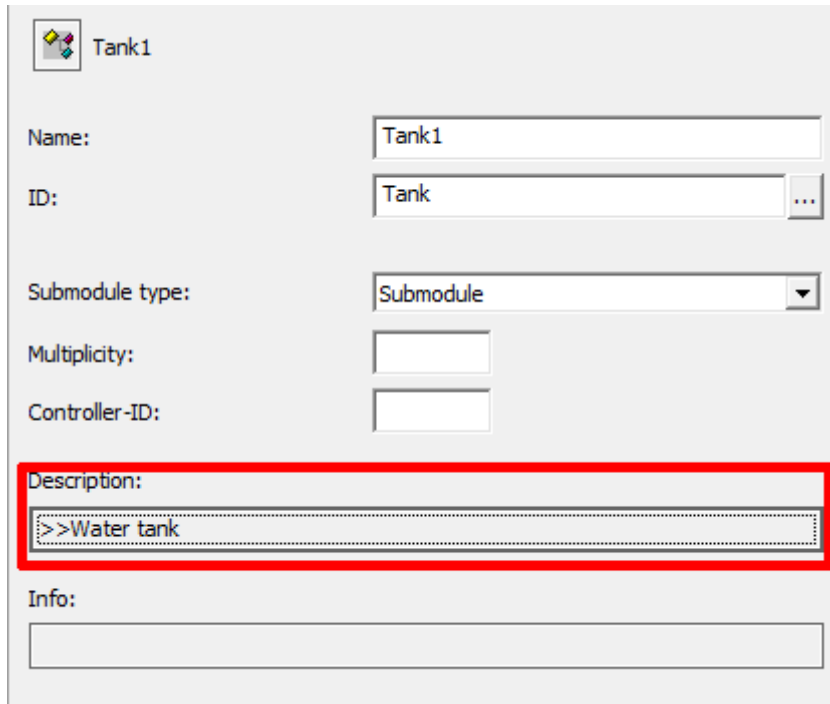


You will notice in the preview at the right the text is displayed as Main menu. The \$ marks a placeholder which will be replaced by another text. DS stands for description surface, so the \$DS will be replaced by the description of the surface. Depending on the context either a surface referred to or the current surface is used. If you use \$DS for an AMenu which opens another surface the description of that surface is used. If the \$DS is not in the context of another surface, the current surface is used.

Select Tank1 in the project tree:



Set the description to the fixed text “Water tank”:



Tank1

Name: Tank1

ID: Tank

Submodule type: Submodule

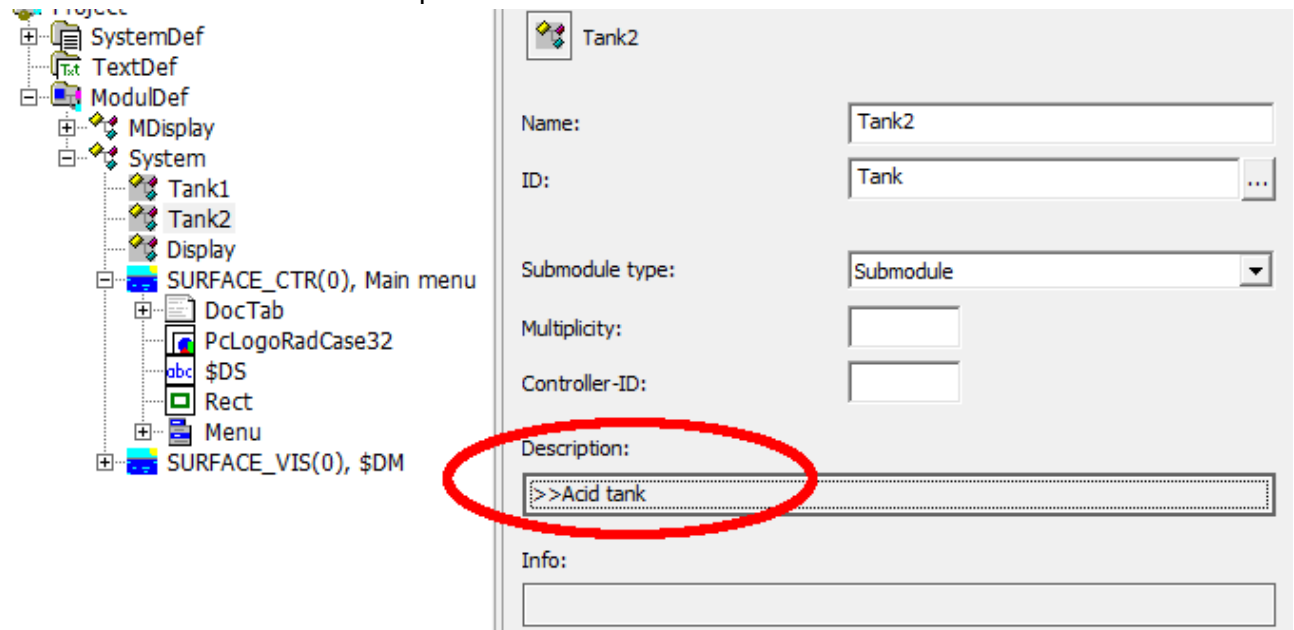
Multiplicity:

Controller-ID:

Description: >>Water tank

Info:

Select Tank2 and set the description the the fixed text “Acid tank”:



Project

- SystemDef
- TextDef
- ModulDef
  - MDisplay
  - System
    - Tank1
    - Tank2
    - Display
  - SURFACE\_CTR(0), Main menu
    - DocTab
    - PcLogoRadCase32
    - \$DS
    - Rect
    - Menu
  - SURFACE\_VIS(0), \$DM

Tank2

Name: Tank2

ID: Tank

Submodule type: Submodule

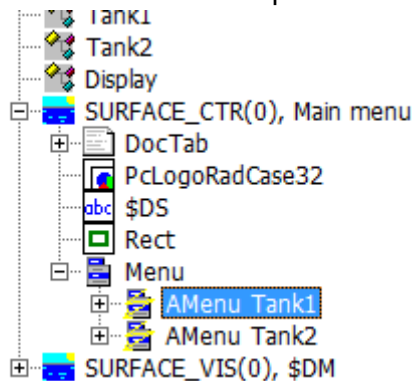
Multiplicity:

Controller-ID:

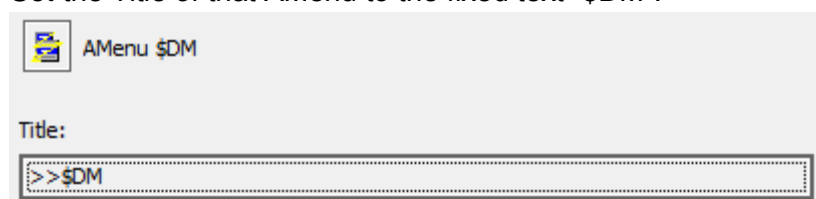
Description: >>Acid tank

Info:

Select the AMenu which opens the surface of our first tank:

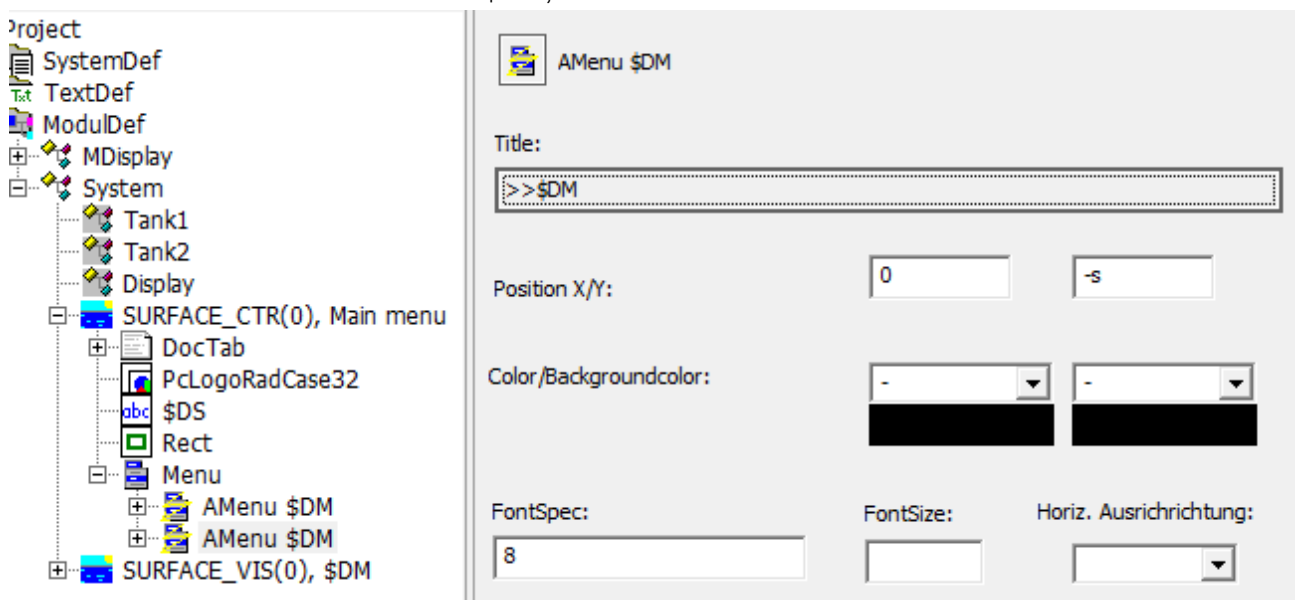


Set the Title of that AMenu to the fixed text "\$DM":

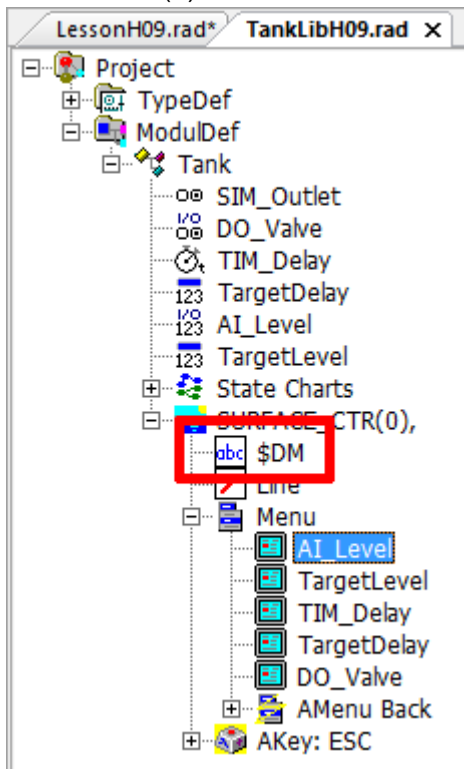


Again \$DM is a placeholder. This one stands for description module and will also use different module descriptions depending on the context. Because the AMenu has actions in it, which will open a surface in the submodule Tank1, instead of using the description of the current module (System) instead the description of the submodule Tank1 is used.

Set the title of the second AMenu to \$DM, too:

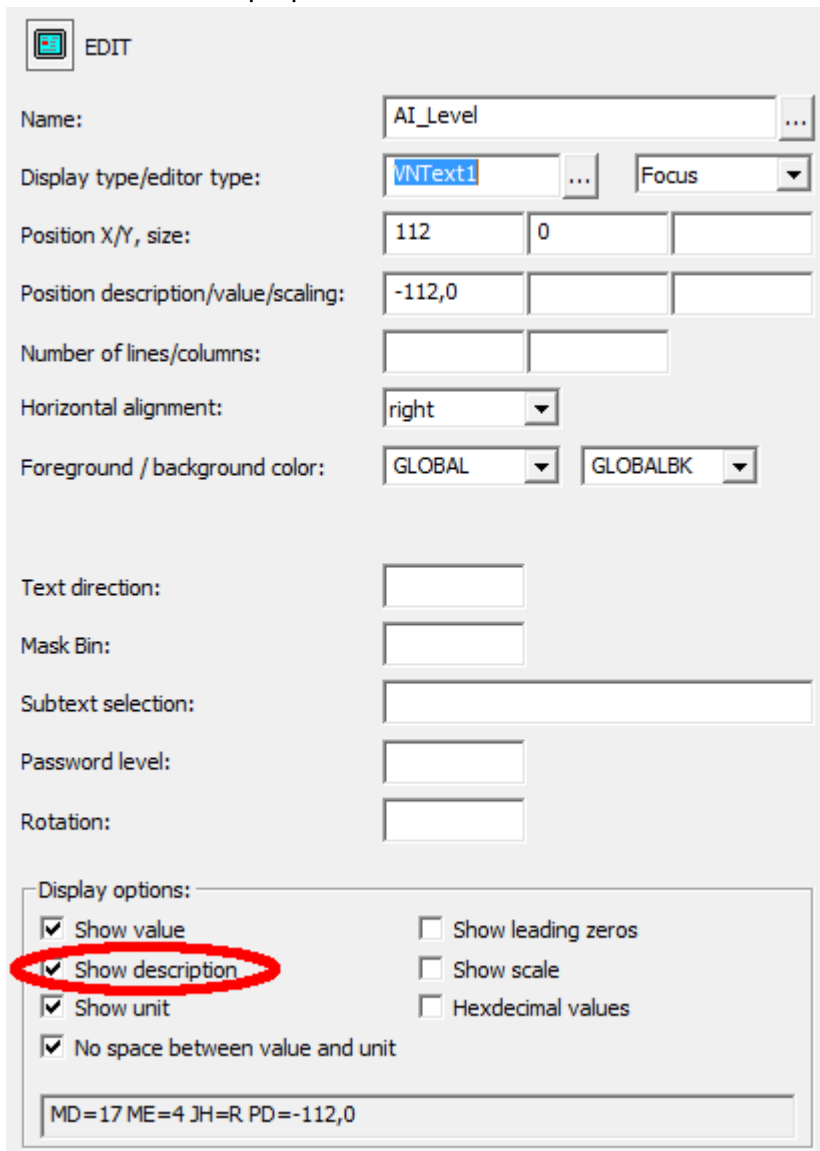


Now we will have a look at element descriptions in our library. First open the tank library by double clicking on one of the Tank submodules. Select the element AI\_Level in the menu of SURFACE\_CTR(0):



You will notice the surface has already added a heading with the description of the module. Because in this case there is no other module in the context, the description of the current module will be used. This value will change according to the selected submodule.

Take a look at the properties of AI\_Level:



**EDIT**

Name:  ...

Display type/editor type:  ... Focus ▾

Position X/Y, size:

Position description/value/scaling:

Number of lines/columns:

Horizontal alignment:  ▾

Foreground / background color:  ▾  ▾

Text direction:

Mask Bin:

Subtext selection:

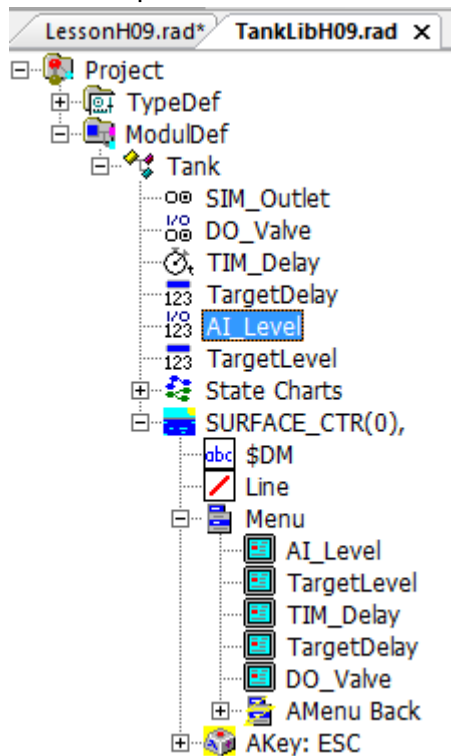
Password level:

Rotation:

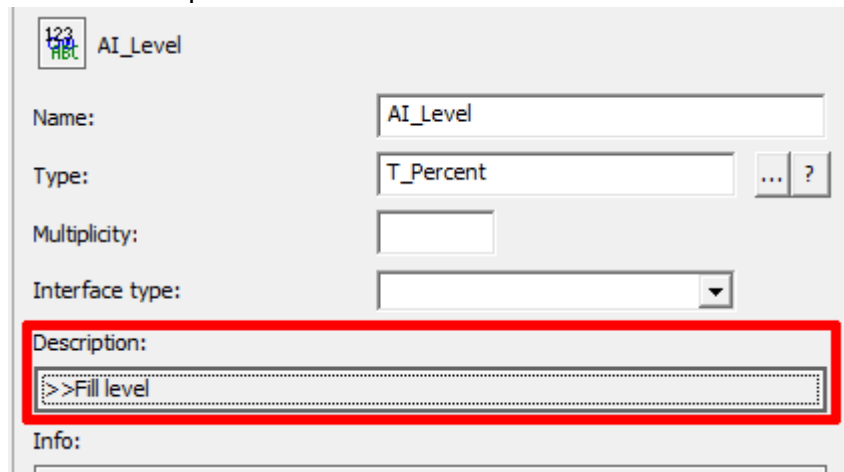
Display options:

- ☒ Show value ☐ Show leading zeros
- ☒ Show description ☐ Show scale
- ☒ Show unit ☐ Hexdecimal values
- ☒ No space between value and unit

You will note, the element visualizer already uses the description of the element AI\_Level. Because there is no description yet, radCASE uses the element name as description. We will now change the description of the element. Select the element AI\_Level:



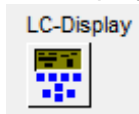
Set the description to the fixed text "Fill level":



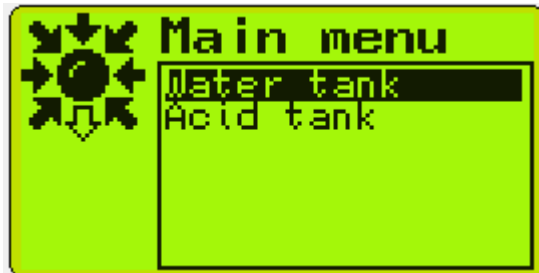
123 ABC	AI_Level
Name:	AI_Level
Type:	T_Percent ... ?
Multiplicity:	
Interface type:	
Description:	>>Fill level
Info:	

### 4.9.3 Conclusion

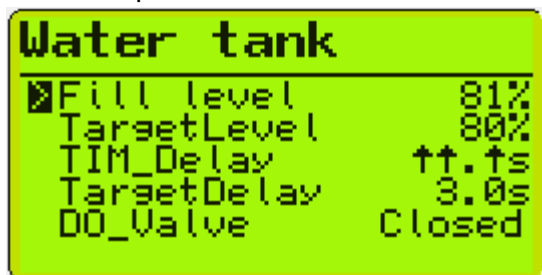
Press <F4> to create and start the simulation. Open the target display using a <click> on the LC-Display icon:



The window showing the hardware display will open:



You see, the \$DS is replaced by our surface description and both menu entries are named after the descriptions of those modules. If you hit <Return> to select the Water tank, the menu of the water tank will open:



You can see, the heading is also Water tank, like the description of the submodule Tank1. You can also see, AI\_Level is now named Fill level in the menu. Press <ESC> to go back to the main menu and select Acid tank. This will open the same surface, for the other module:



You can see, in this case \$DM is replaced by the description of the second Tank submodule



## 4.10 Lesson H10: Visualization Surfaces

### 4.10.1 Goal

In this lesson you will learn how to add HMI to the project monitor, which allows visualizing and controlling the process.

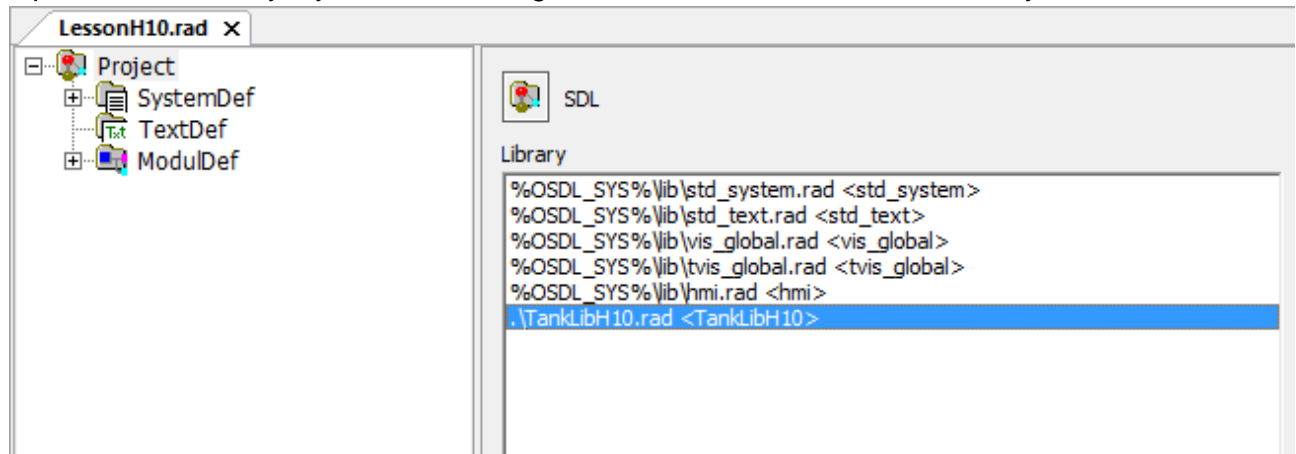
### 4.10.2 Content

You should start this lesson with the project LessonH10 which is delivered with your radCASE copy. In this project the following items were added:

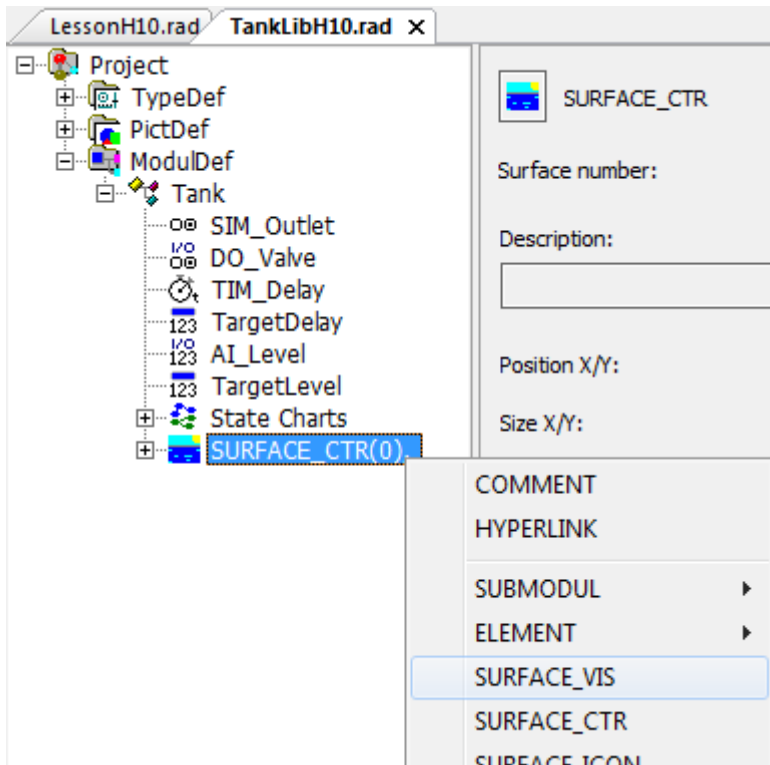
- the library vis\_water.rad is added to the tank library, to allow using visualizers out of that library
- also the library vis\_global.rad was added to the tank library
- two pictures were added to the tank library for use in the HMI. (You can see them in the PictDef-section).
- descriptions for all elements were completed

Until now, we controlled the process using the module view. Now we want to create an HMI which shows the current state of the process in a better understandable way and only allow to view and set a defined set of elements. In this case we want to only be able to set the target fill level and open the simulated outlet and view the current fill level and the state of the inlet valve.

Open the tank library, by *<double clicking>* on the TankLibH10.rad in the library list:

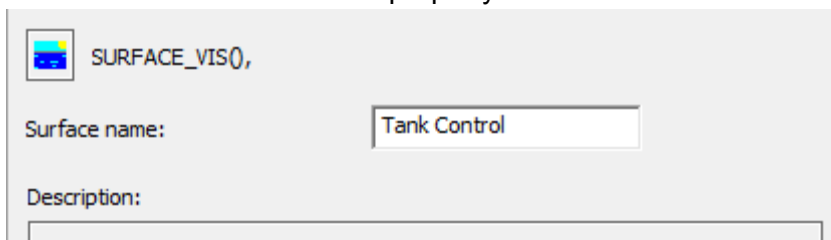


First we need to create a surface. <Shift + right click> on the SURFACE\_CTR(0) in your Tank module and select **SURFACE\_VIS**:

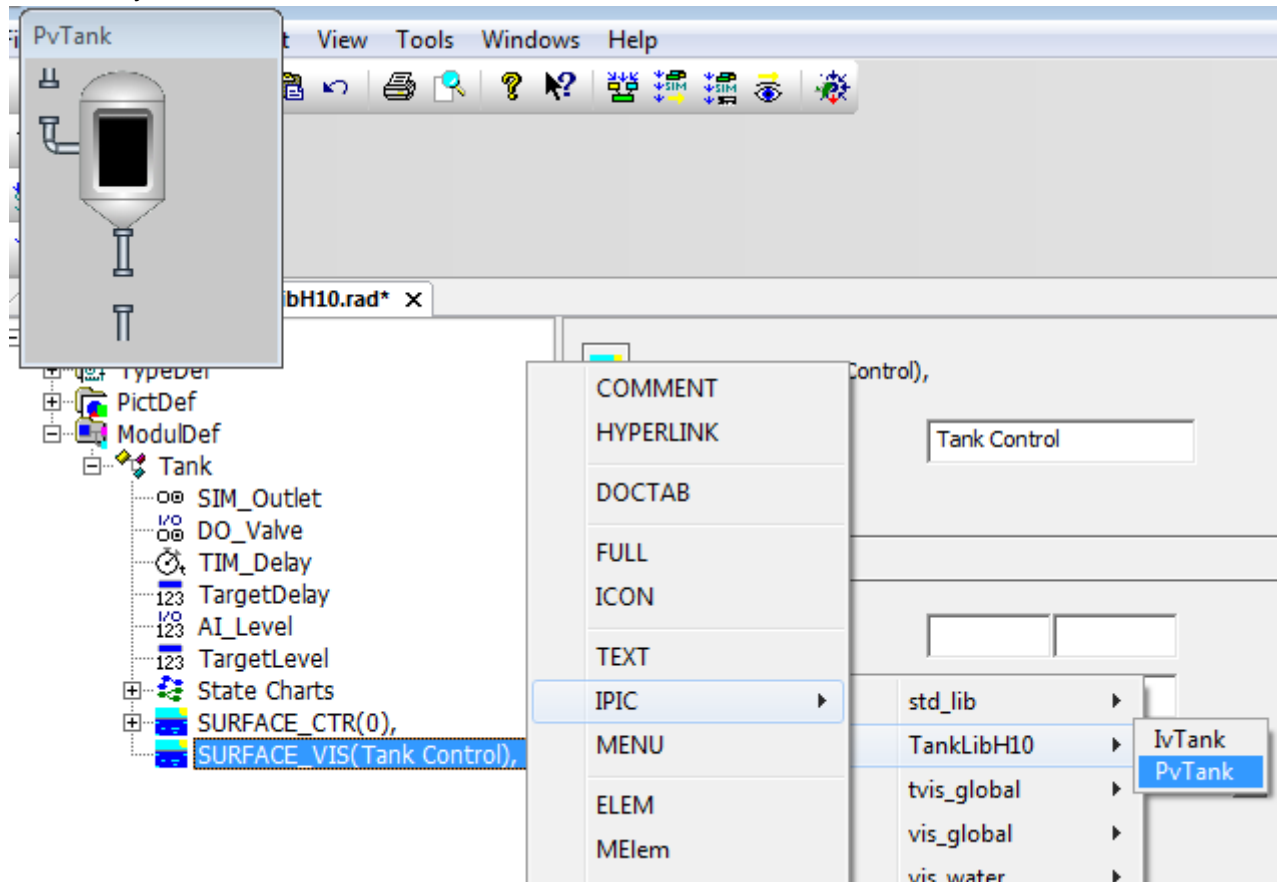


A SURFACE\_VIS is a surface which will be used in the visualization only by the project monitor. It will not use any memory on the real hardware, but can be used to visualize the process with a real hardware connected to the project monitor.

Set the **Surface name** in the property editor to “Tank Control”:

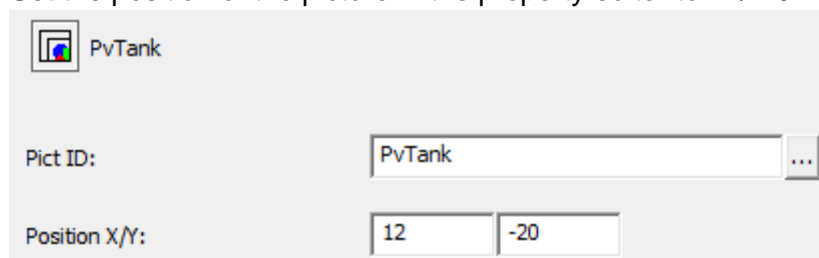


Now we add a background image, which will help understanding what the process does. <Right click> on your new **SURFACE\_VIS** and select the **IPIC** PvTank from TankLibH10:



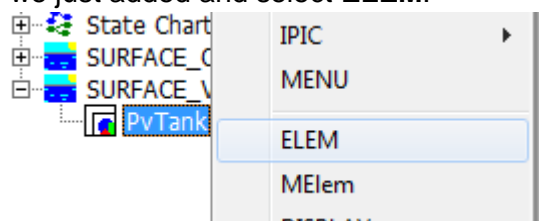
You can see a preview of the picture in another window (top left).

Set the position of the picture in the property editor to 12/-20:

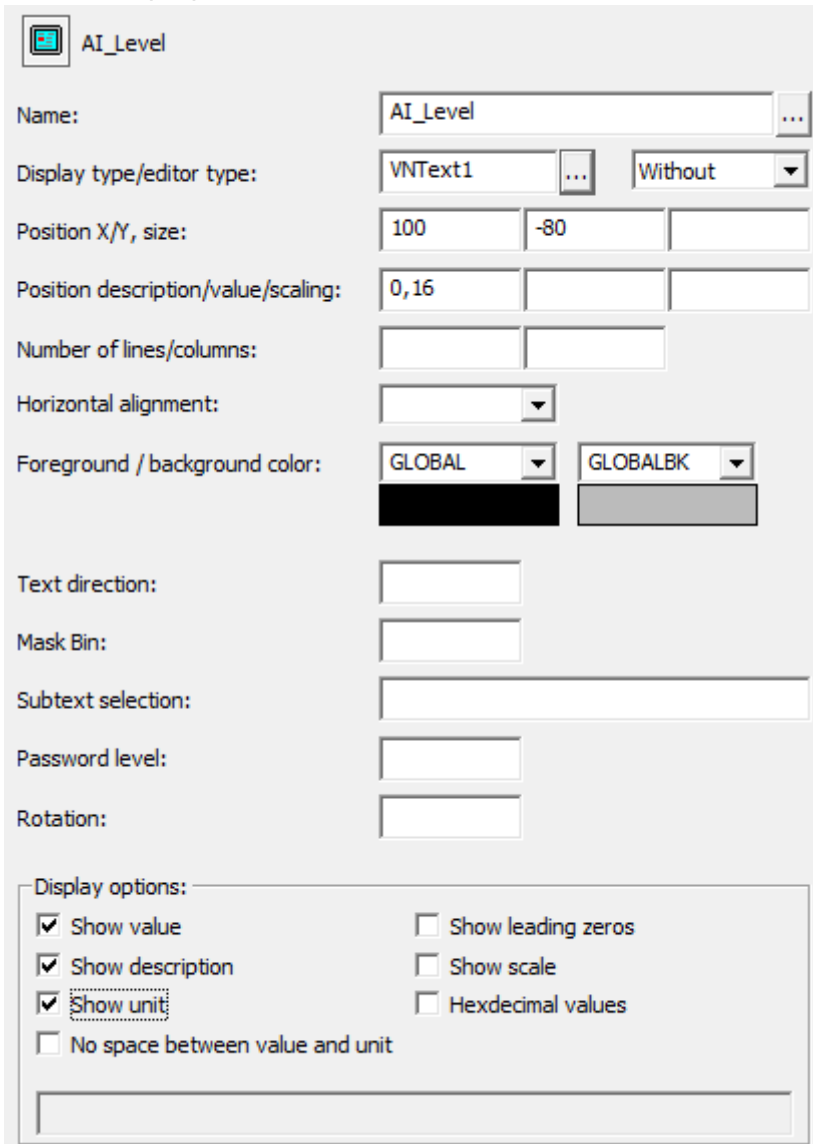


You will notice in the right pane of radEDIT you can see a preview of your surface. You can also position elements using drag&drop in that pane.

Now we will add a visualizer showing the value of AI\_Level as text. <Shift + right click> on the picture we just added and select **ELEM**:



Fill out the properties editor as shown below:

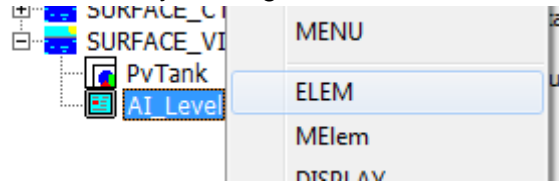


The screenshot shows a properties editor for an element named 'AI\_Level'. The interface includes various input fields and dropdown menus for configuring the element's appearance and behavior. The 'Name' field is set to 'AI\_Level'. The 'Display type/editor type' is set to 'VNTxt1' with a 'Without' dropdown. Positioning is set to X=100, Y=-80, with a description '0,16'. The 'Number of lines/columns' is set to 1 line and 1 column. Horizontal alignment is set to 'Left'. Foreground and background colors are set to 'GLOBAL' and 'GLOBALBK' respectively, with corresponding color swatches. Text direction, mask bin, subtext selection, password level, and rotation are all set to their default values. The 'Display options' section at the bottom contains several checkboxes: 'Show value', 'Show description', 'Show unit', 'Show leading zeros', 'Show scale', 'Hexdecimal values', and 'No space between value and unit'. The 'Show value', 'Show description', and 'Show unit' checkboxes are checked.

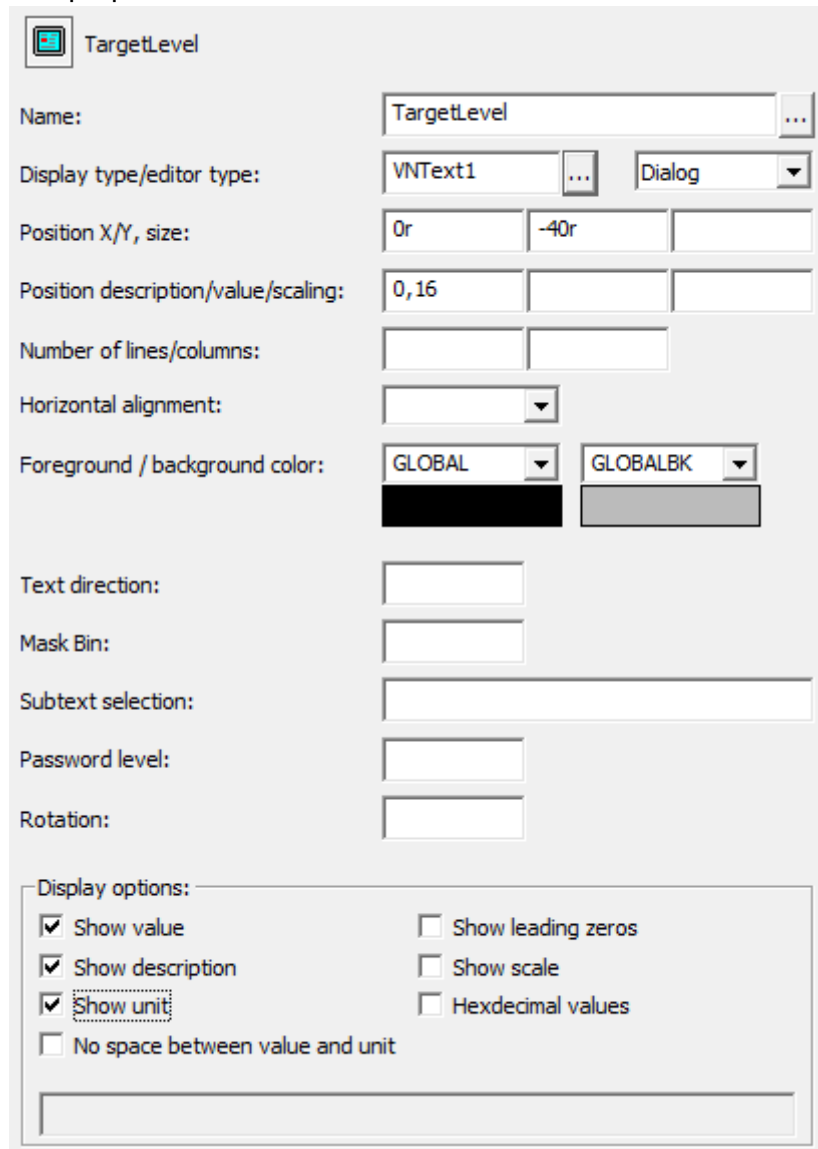
AI_Level	
Name:	AI_Level
Display type/editor type:	VNTxt1 Without
Position X/Y, size:	100 -80
Position description/value/scaling:	0,16
Number of lines/columns:	1 1
Horizontal alignment:	Left
Foreground / background color:	GLOBAL GLOBALBK
Text direction:	Left
Mask Bin:	
Subtext selection:	
Password level:	1
Rotation:	0
Display options:	
<input checked="" type="checkbox"/> Show value	<input type="checkbox"/> Show leading zeros
<input checked="" type="checkbox"/> Show description	<input type="checkbox"/> Show scale
<input checked="" type="checkbox"/> Show unit	<input type="checkbox"/> Hexdecimal values
<input type="checkbox"/> No space between value and unit	

The **Name** determines that the element AI\_Level is visualized according to its value. The **Display type** VNTxt1 determines it is displayed as text. The **editor type** Without determines, that the element can't be edited. The checkboxes at the bottom determine the element is displayed with its description, value and unit.

We will now create a second visualizer for the element TargetLevel, again by using *<Shift + right click>*, this time by clicking on AI\_level:



The properties are set as shown below:



**TargetLevel**

Name: TargetLevel

Display type/editor type: VNTxt1 Dialog

Position X/Y, size: 0r -40r

Position description/value/scaling: 0,16

Number of lines/columns:

Horizontal alignment:

Foreground / background color: GLOBAL GLOBALBK

Text direction:

Mask Bin:

Subtext selection:

Password level:

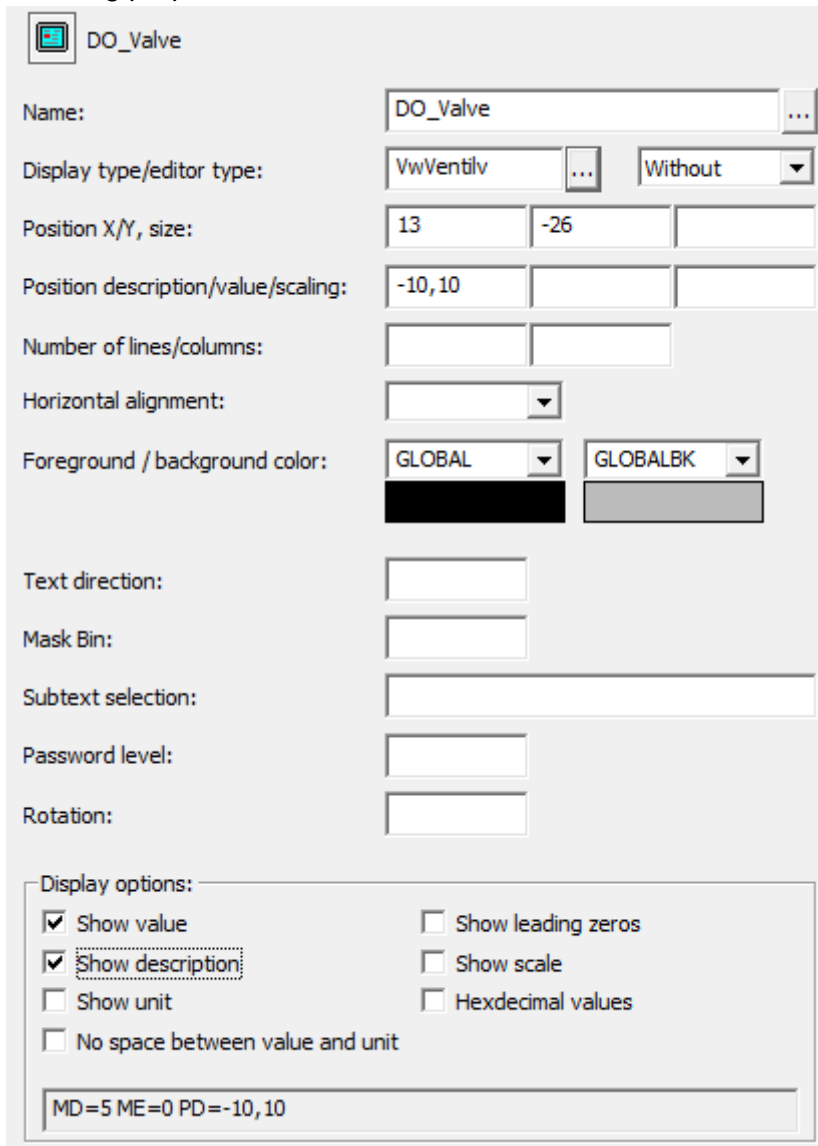
Rotation:

Display options:

- ☒ Show value
- ☒ Show description
- ☒ Show unit
- ☐ No space between value and unit
- ☐ Show leading zeros
- ☐ Show scale
- ☐ Hexadecimal values

Mostly the settings are the same as for AI\_Level. Only the **editor type** was changed to Dialog, so the element can be edited in a dialog which will open, when clicking on the element. Also the positioning is done using relative positioning. The “r” behind the values of the position determines the position is relative to the position of the previous element (AI\_Level).

Create the next visualizer again as last element in the surface using <Shift + right click> with the following properties:



The image shows a configuration window for a visualizer named "DO\_Valve". The window has a title bar with a small icon and the name "DO\_Valve". Below the title bar, there are several fields and options for configuring the visualizer. The "Name" field is set to "DO\_Valve". The "Display type/editor type" is set to "VwVentilv" with a dropdown arrow. The "Position X/Y, size" fields are set to "13" and "-26". The "Position description/value/scaling" field is set to "-10, 10". The "Number of lines/columns" fields are empty. The "Horizontal alignment" dropdown is set to "Left". The "Foreground / background color" fields are set to "GLOBAL" and "GLOBALEBK" with corresponding color swatches. The "Text direction" field is empty. The "Mask Bin" field is empty. The "Subtext selection" field is empty. The "Password level" field is empty. The "Rotation" field is empty. At the bottom, there is a "Display options" section with several checkboxes: "Show value" (checked), "Show description" (checked), "Show unit" (unchecked), "No space between value and unit" (unchecked), "Show leading zeros" (unchecked), "Show scale" (unchecked), and "Hexdecimal values" (unchecked). Below the checkboxes, there is a text box containing the string "MD=5 ME=0 PD=-10,10".

DO\_Valve

Name: DO\_Valve

Display type/editor type: VwVentilv Without

Position X/Y, size: 13 -26

Position description/value/scaling: -10, 10

Number of lines/columns:

Horizontal alignment:

Foreground / background color: GLOBAL GLOBALEBK

Text direction:

Mask Bin:

Subtext selection:

Password level:

Rotation:

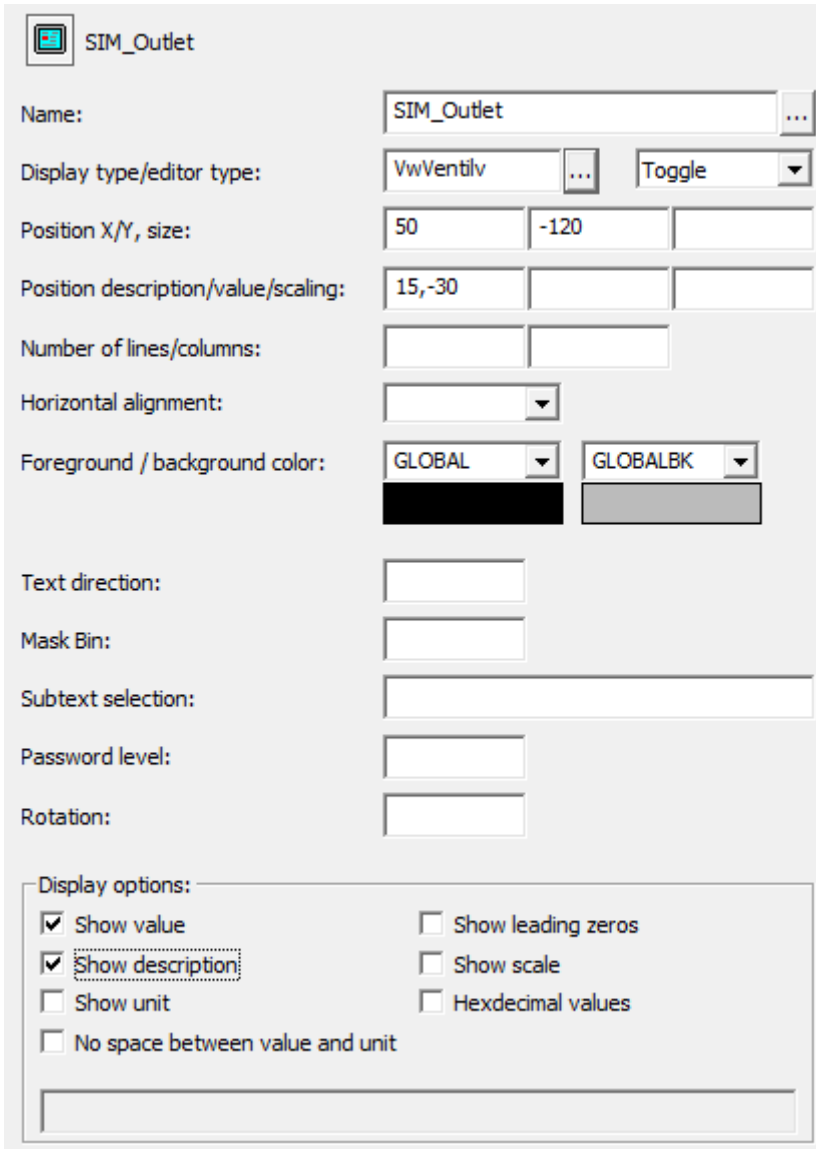
Display options:

- ☒ Show value
- ☒ Show description
- ☐ Show unit
- ☐ No space between value and unit
- ☐ Show leading zeros
- ☐ Show scale
- ☐ Hexdecimal values

MD=5 ME=0 PD=-10,10

In this case the difference is the **Display type**. This time the value is not displayed as text, but using different pictures. This is a visualizer located in a library. You can create your own visualizers like this, but this will not be covered in this tutorial. Also we will not show a unit, because the element does not have any unit. Enabling that option would make no difference.

We will now use the same visualizer for the SIM\_Outlet. Again using *<Shift + right click>* to create the visualizer as last element in the list. Set the properties to the following values:



**SIM\_Outlet**

Name: SIM\_Outlet ...

Display type/editor type: VwVentilv ... Toggle

Position X/Y, size: 50 -120

Position description/value/scaling: 15,-30

Number of lines/columns: 1 1

Horizontal alignment: Left

Foreground / background color: GLOBAL GLOBLEBK

Text direction: Horizontal

Mask Bin: 1

Subtext selection: 1

Password level: 1

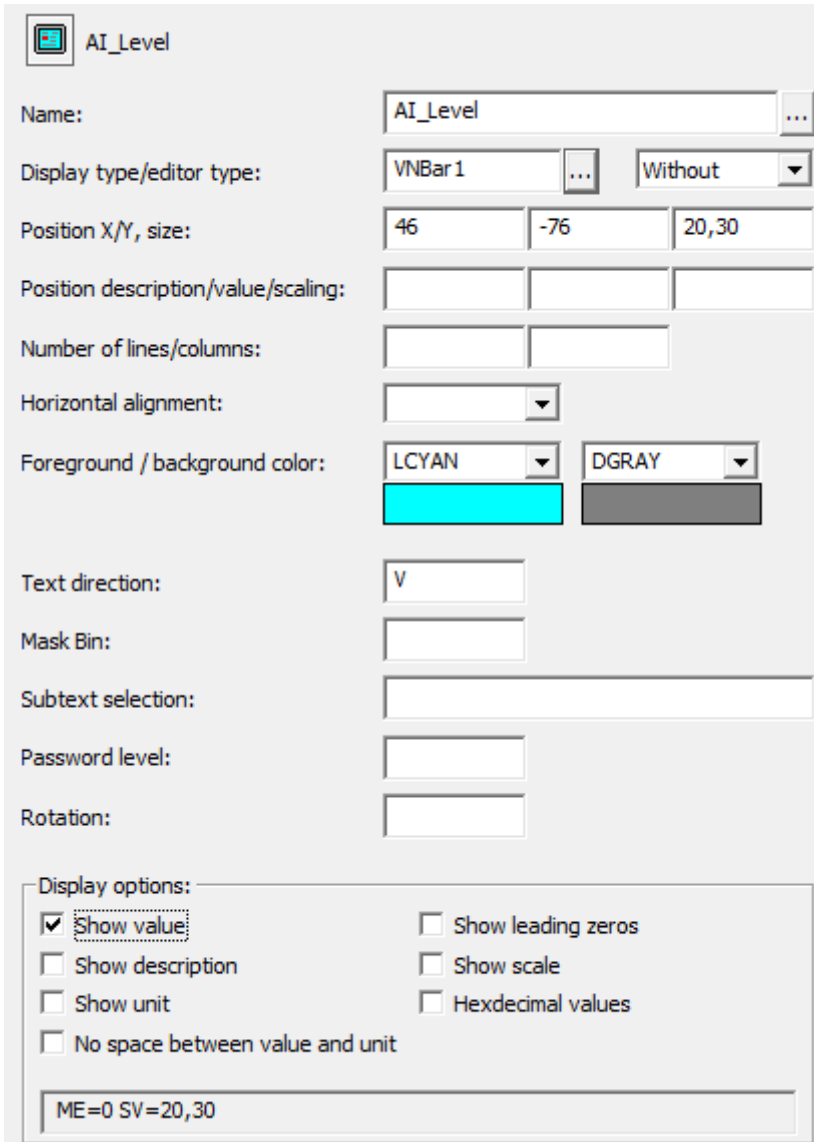
Rotation: 0

Display options:

- ☒ Show value
- ☐ Show leading zeros
- ☒ Show description
- ☐ Show scale
- ☐ Show unit
- ☐ Hexdecimal values
- ☐ No space between value and unit

This time the only new is the **editor type**. The editor type Toggle determines you can edit the element, but there will be no dialog letting you select the values, but instead the different possible values, which are Open and Closed in this case, are toggled.

We will now add the AI\_Level a second time as our last visualizer. But this time, we will use another visualizer type. Add the visualizer again with *<Shift + right click>* and set the properties as follows:



The screenshot shows the configuration window for a visualizer named "AI\_Level". The window has a title bar with a small icon and the name "AI\_Level". The configuration is organized into several sections:

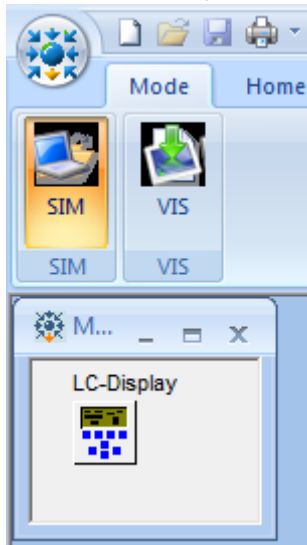
- Name:** A text field containing "AI\_Level" with a dropdown arrow on the right.
- Display type/editor type:** A dropdown menu set to "VNBar1" with a dropdown arrow on the right, and a "Without" dropdown menu.
- Position X/Y, size:** Three text fields containing "46", "-76", and "20,30".
- Position description/value/scaling:** Three empty text fields.
- Number of lines/columns:** Two empty text fields.
- Horizontal alignment:** A dropdown menu.
- Foreground / background color:** Two dropdown menus. The first is set to "LCYAN" and the second to "DGRAY". Below each dropdown is a color swatch: a cyan rectangle for the foreground and a gray rectangle for the background.
- Text direction:** A text field containing "V".
- Mask Bin:** An empty text field.
- Subtext selection:** An empty text field.
- Password level:** An empty text field.
- Rotation:** An empty text field.
- Display options:** A group box containing several checkboxes:
  - ☒ Show value
  - ☐ Show description
  - ☐ Show unit
  - ☐ No space between value and unit
  - ☐ Show leading zeros
  - ☐ Show scale
  - ☐ Hexdecimal values
- Preview:** A text field at the bottom containing "ME=0 SV=20,30".

This will display the fill level as a bar which will grow as the value gets bigger. This time we only show the value. The colors define the colors of the background of the bar area and the color of the bar. Also the size must be defined to determine the size of the area the bar is shown. At last the text direction determines that the bar is vertical.



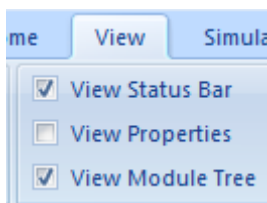
### 4.10.3 Conclusion

Hit <F4> to create and start the simulation. You will notice the simulation does not show any sign of the surface we just created:

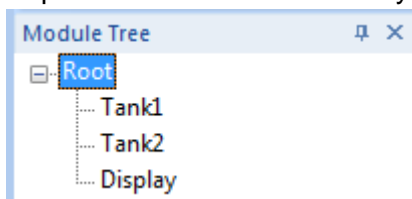


This is because like for SURFACE\_CTR the main window shown at start is always SURFACE\_VIS(0) of the System module. Because we have created our surface in the Tank module, we can't see it at the moment. How to change that and open that surface in an easy way, will be addressed in the next lesson.

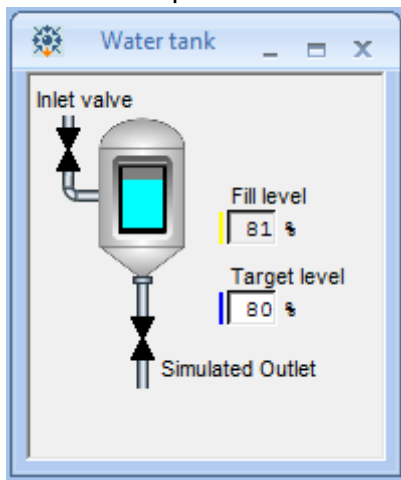
To see the surface we just created open the module tree:



Expand the Root module and you will see your two tank submodules:



We already used that, to open our module view for the tanks. But this time instead of opening the module view use a *<double click>* onto the Tank1 entry in the module tree. This will open the surface we just created. If there are multiple surfaces in a module a selection dialog of all available surfaces is opened instead.



You can now play with the process by changing the target fill level or by opening the Simulation outlet by clicking on the symbol. You can also open the second tank with a *<double click>* and play with both tanks.

## 4.11 Lesson H11: Visualization Surface Navigation

### 4.11.1 Goal

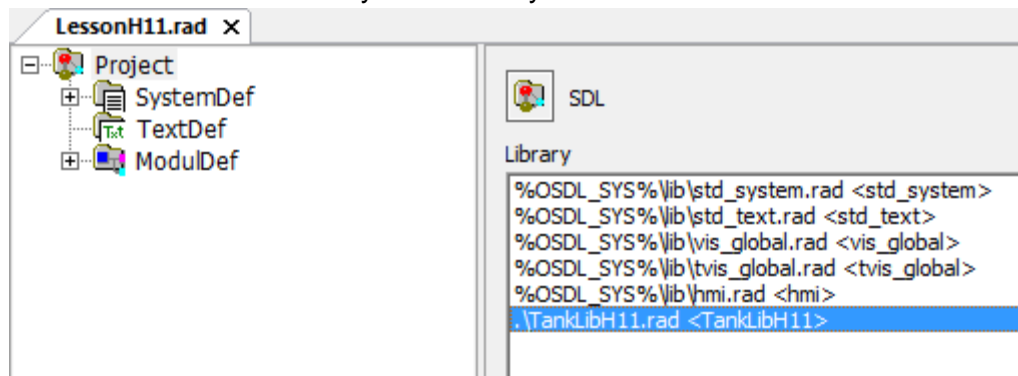
In this lesson you will learn how to make a surface navigation in the project monitor, using Icons.

### 4.11.2 Content

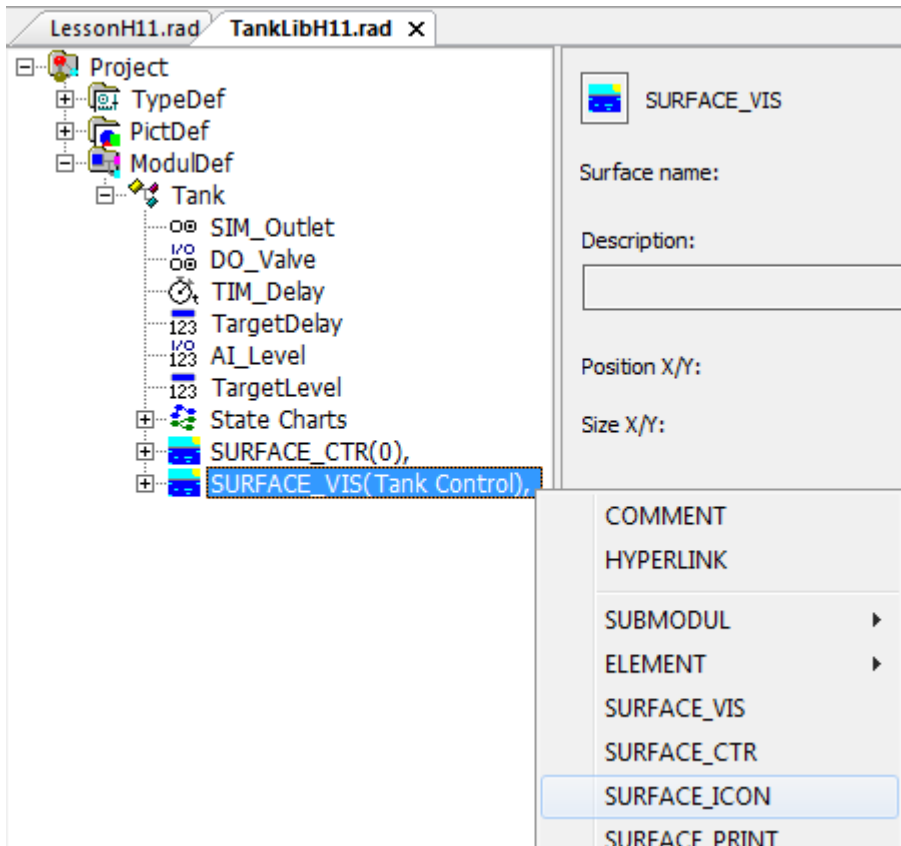
You can start this lesson by using the project you created in lesson H10 or by opening the project LessonH11 that is delivered with your radCASE copy.

In the last lesson we learned how to add a surface into the Tank module, but we could not open that surface in an easy way. Now we will add a button onto the main surface of our project. When clicking on that button the surface we created in the last lesson will be opened.

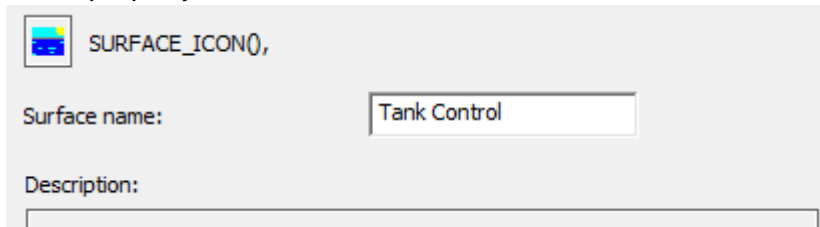
To do this, we first must define what the button should look like. Open the tank library using a *<double click>* on the library in the library list:



Use <Shift + right click> on the **SURFACE\_VIS** Tank Control, we created in the last lesson and select **SURFACE\_ICON**:

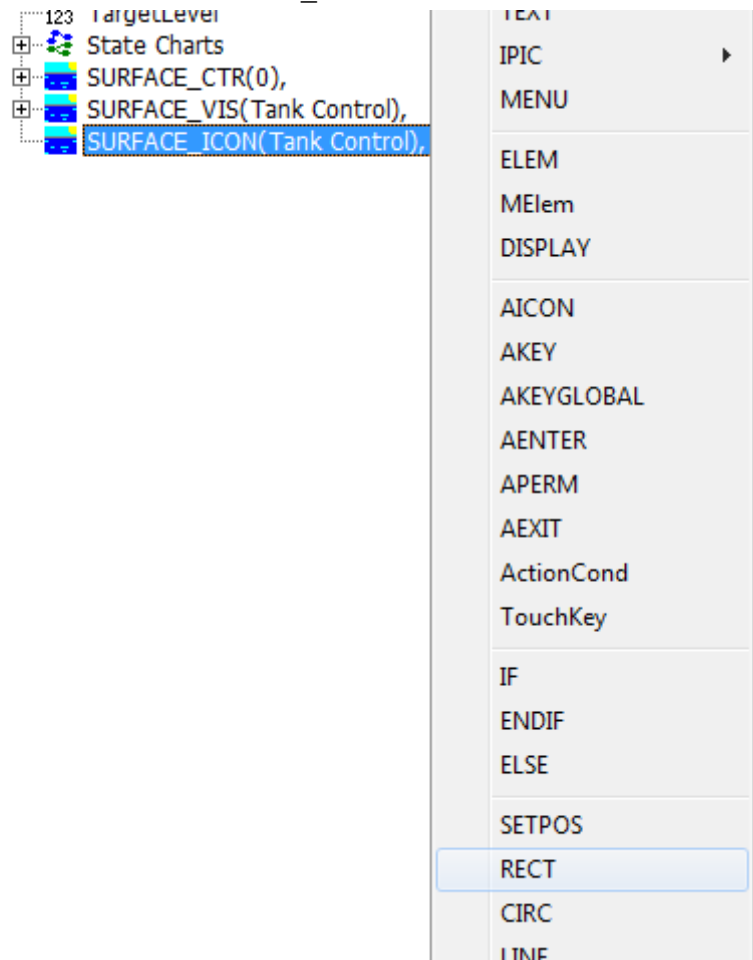


In the property editor set the **Surface name** of the **SURFACE\_ICON** to Tank Control:

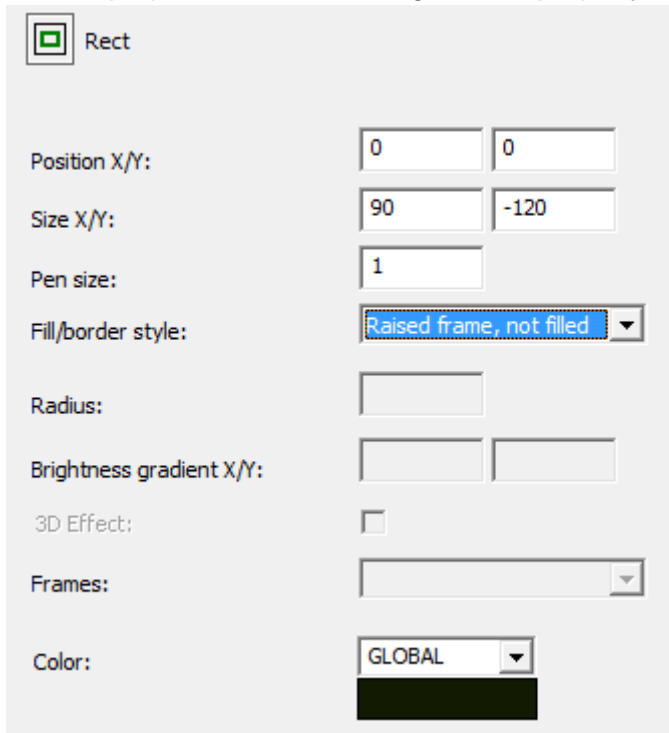


This is exactly the same name as the **SURFACE\_VIS**. This way we define that this **SURFACE\_ICON** is used to open the **SURFACE\_VIS** with the name Tank Control. The **SURFACE\_ICON** itself can be filled with the same contents as a **SURFACE\_VIS**. The difference is, you can't edit any elements in a **SURFACE\_ICON**. So now we will add how our button will look like.

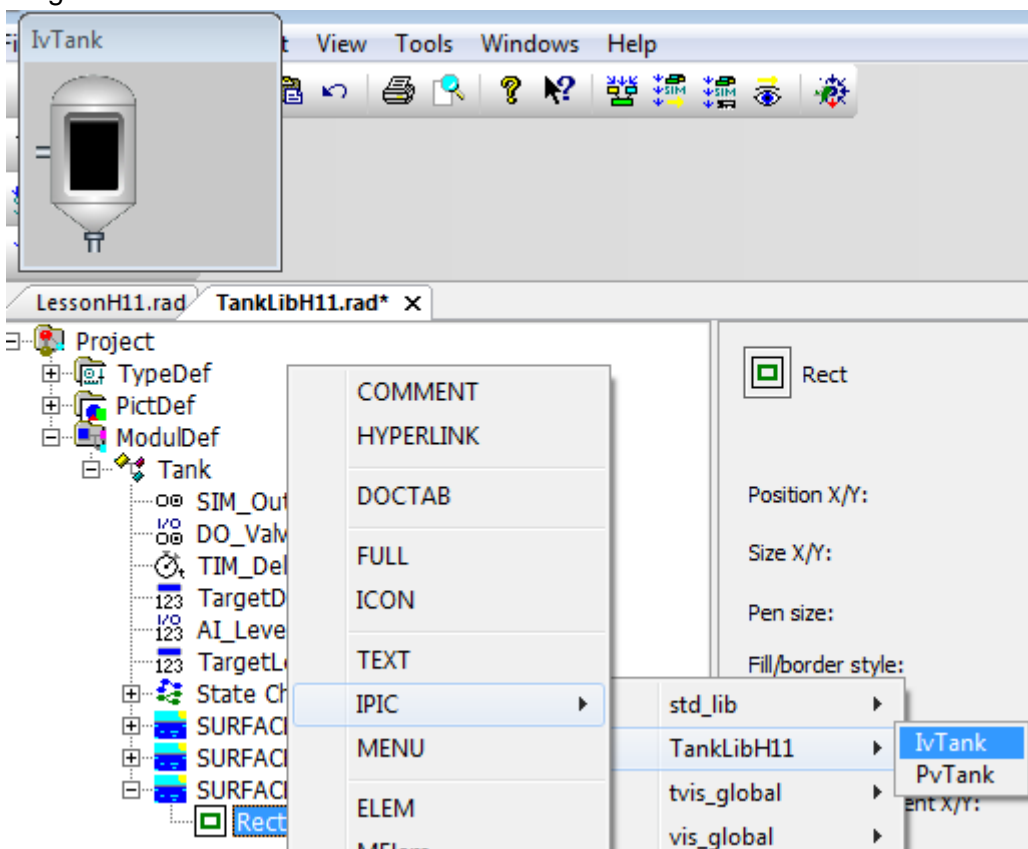
First we want to add a surrounding rectangle, so the user will see the area of the button he can click.  
<Right click> on the **SURFACE\_ICON** and select **RECT**:



Set the properties of the rectangle in the property editor as follows:

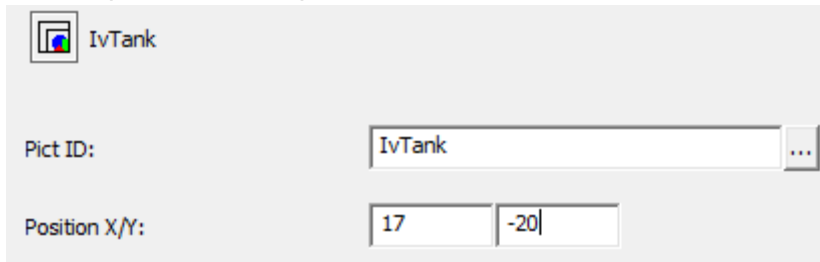


To have a good looking button, we will add a background picture. <Shift + right click> on the Rectangle and select the **IPIC** IvTank in TankLibH11:



You will see a preview window, like the one in the top left.

Set the position of the picture to 17 / -20 to have it centered in the rectangle:

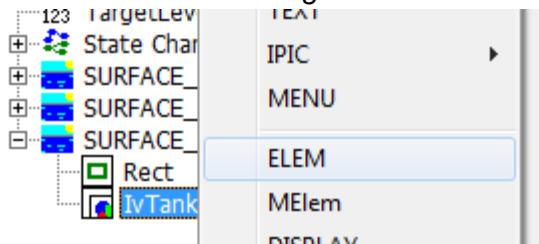


IvTank

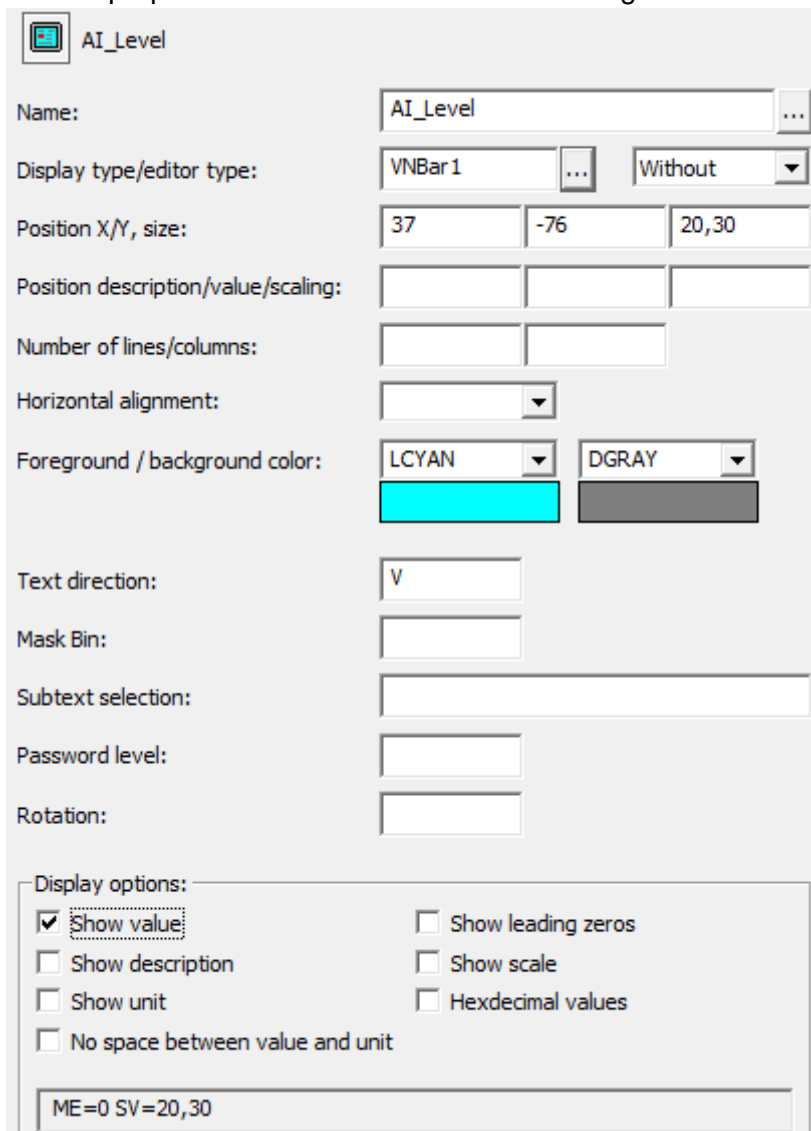
Pict ID: IvTank

Position X/Y: 17 -20

At last we will add a bar visualizer of the current fill level. This allows us to already see the tank fill level in our button. *<Shift + right click>* on the IvTank picture and select **ELEM**:



Set the properties of the element to the following values:



AI\_Level

Name: AI\_Level

Display type/editor type: VNBar 1 Without

Position X/Y, size: 37 -76 20,30

Position description/value/scaling:

Number of lines/columns:

Horizontal alignment:

Foreground / background color: LCYAN DGRAY

Text direction: V

Mask Bin:

Subtext selection:

Password level:

Rotation:

Display options:

☒ Show value ☐ Show leading zeros

☐ Show description ☐ Show scale

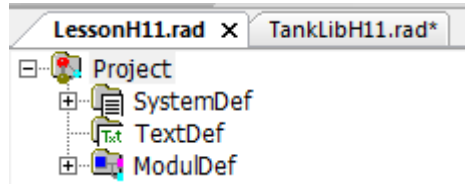
☐ Show unit ☐ Hexadecimal values

☐ No space between value and unit

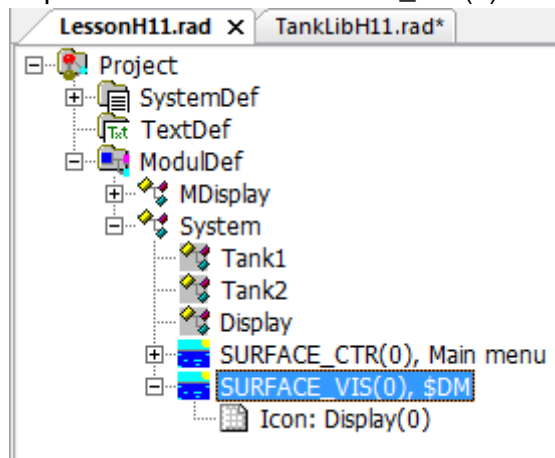
ME=0 SV=20,30

With this we have finished the look of our button. Now we will have to add this button onto the main surface, which is always opened in the project monitor. The main surface is always the **SURFACE\_VIS** with the name "0" in the System module. If there is no **SURFACE\_VIS(0)** in the System module radCASE will open an empty surface instead.

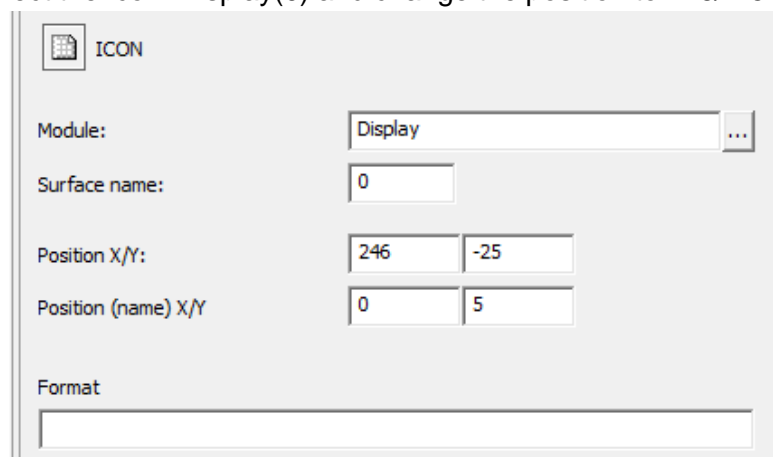
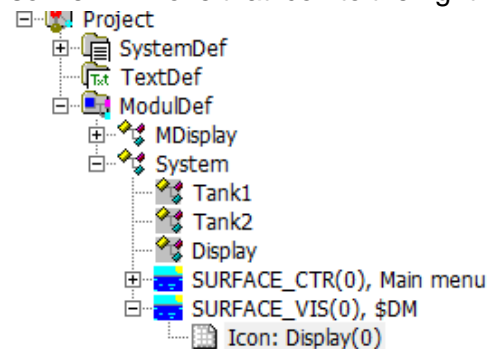
Go to the main project file LessonH11.rad, or the according name in your project:



Expand and select SURFACE\_VIS(0) in the System module:

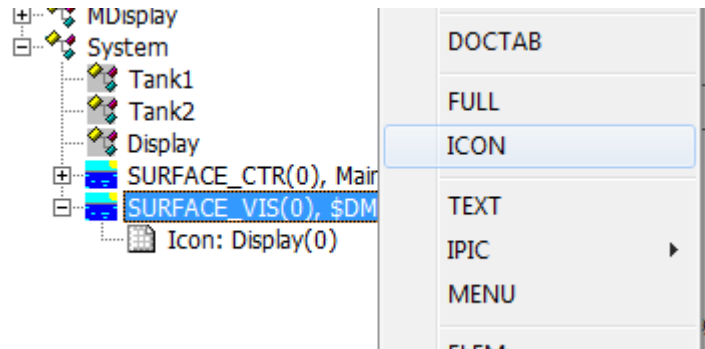


You can see, there is already an Icon in this surface. You can see that icon in the preview as the icon we used to open the simulated hardware display. We want to make room for our Tank-Icons, so we will move that icon to the right. Select the Icon: Display(0) and change the position to 246/-25:

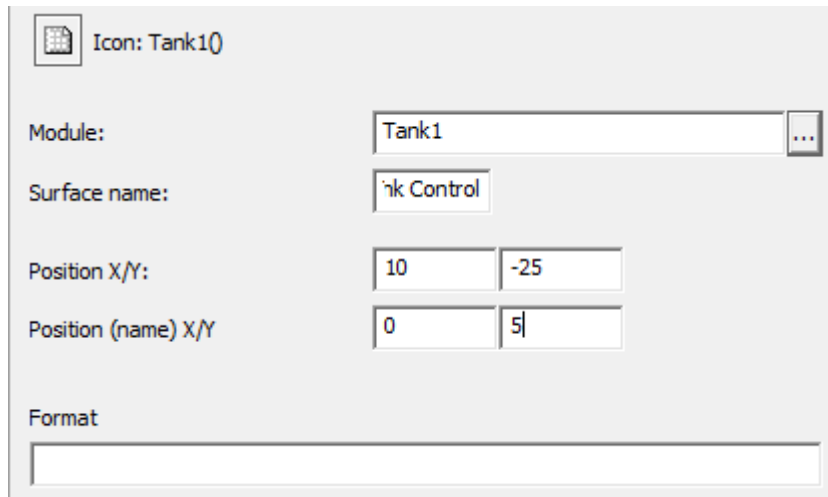




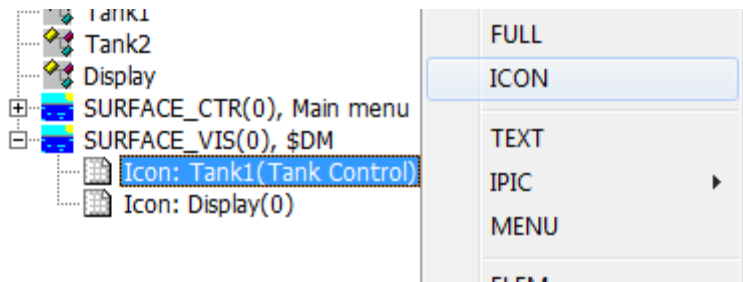
Now we just need to add our button to the main surface. <Right click> on the surface and select **ICON**:



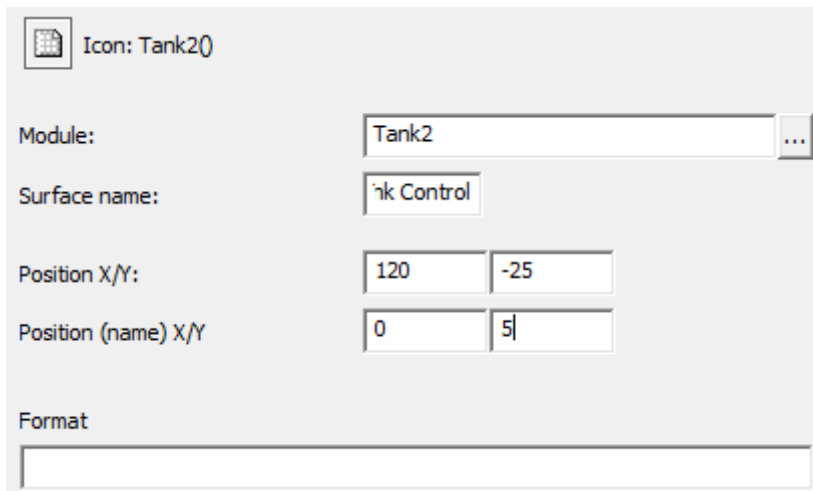
Set the properties of the **ICON** to the following values. Note: You can select the **Module** by clicking on the Button "...". The **Surface name** is again "Tank Control" to identify the Button you created in the Tank module:



We will add another ICON for our second Tank. <Shift + right click> on the Icon: Tank1(Tank Control) and select **ICON**:



Set the properties of that Icon to the following values:



Icon: Tank2()

Module: Tank2

Surface name: Tank Control

Position X/Y: 120 -25

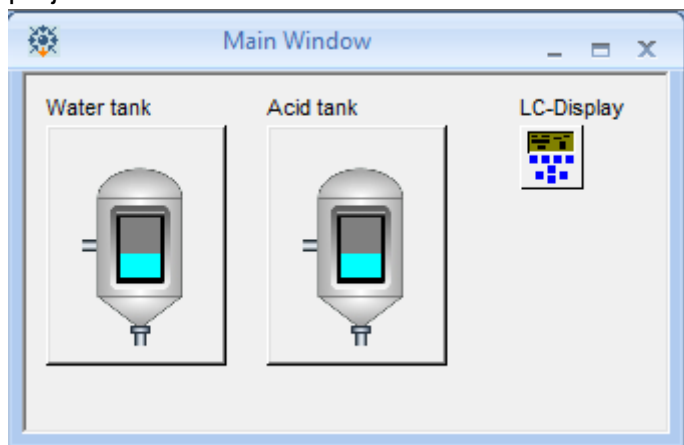
Position (name) X/Y: 0 5

Format

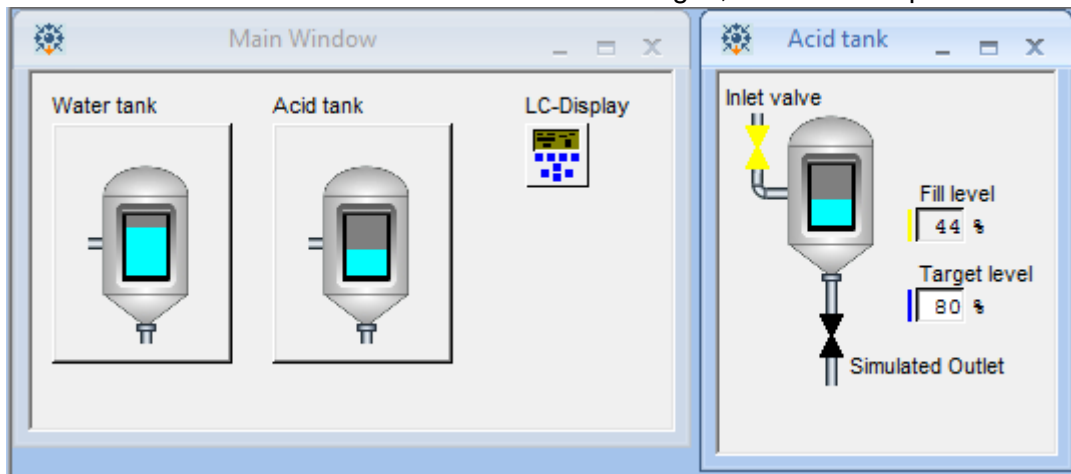
The **Surface name** is again “Tank Control”.

#### 4.11.3 Conclusion

Hit <F4> to create and start the simulation. You will instantly see the buttons we created in the project and will also see how the tanks fill at the start of simulation:



Now you can <click> on one of the buttons (e.g. the button Acid tank) in the main view and the surface we created in the last lesson will open. If this is necessary you can move that window of Tank1 out of the way and you can see, both windows are still open. When you play with the values in the surface of the acid tank and the fill level changes, the value is updated in both surfaces:



## 4.12 Lesson H12: Entity Tab

### 4.12.1 Goal

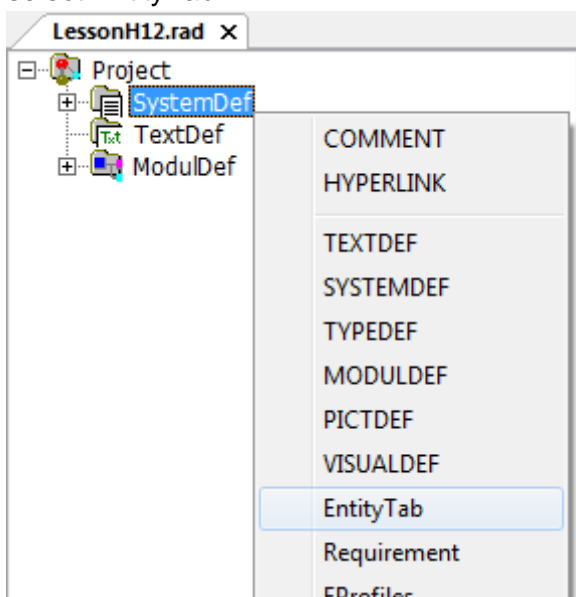
In this lesson you will learn how to create different standard values for different submodules of the same module, using the EntityTab. You will also learn how the EntityTab helps in mapping IO-elements to the IOs on a real hardware.

### 4.12.2 Content

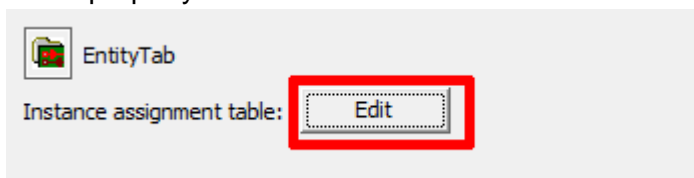
You can start this lesson by using the project you created in lesson H11 or by opening the project LessonH12 that is delivered with your radCASE copy.

At the moment the delay before opening the inlet is by default 3.0 seconds for both of our tanks. If we want to change that delay to 15.0 seconds for the acid tank, we don't need to create a new module, but we can change that standard value for one of our submodules using the EntityTab.

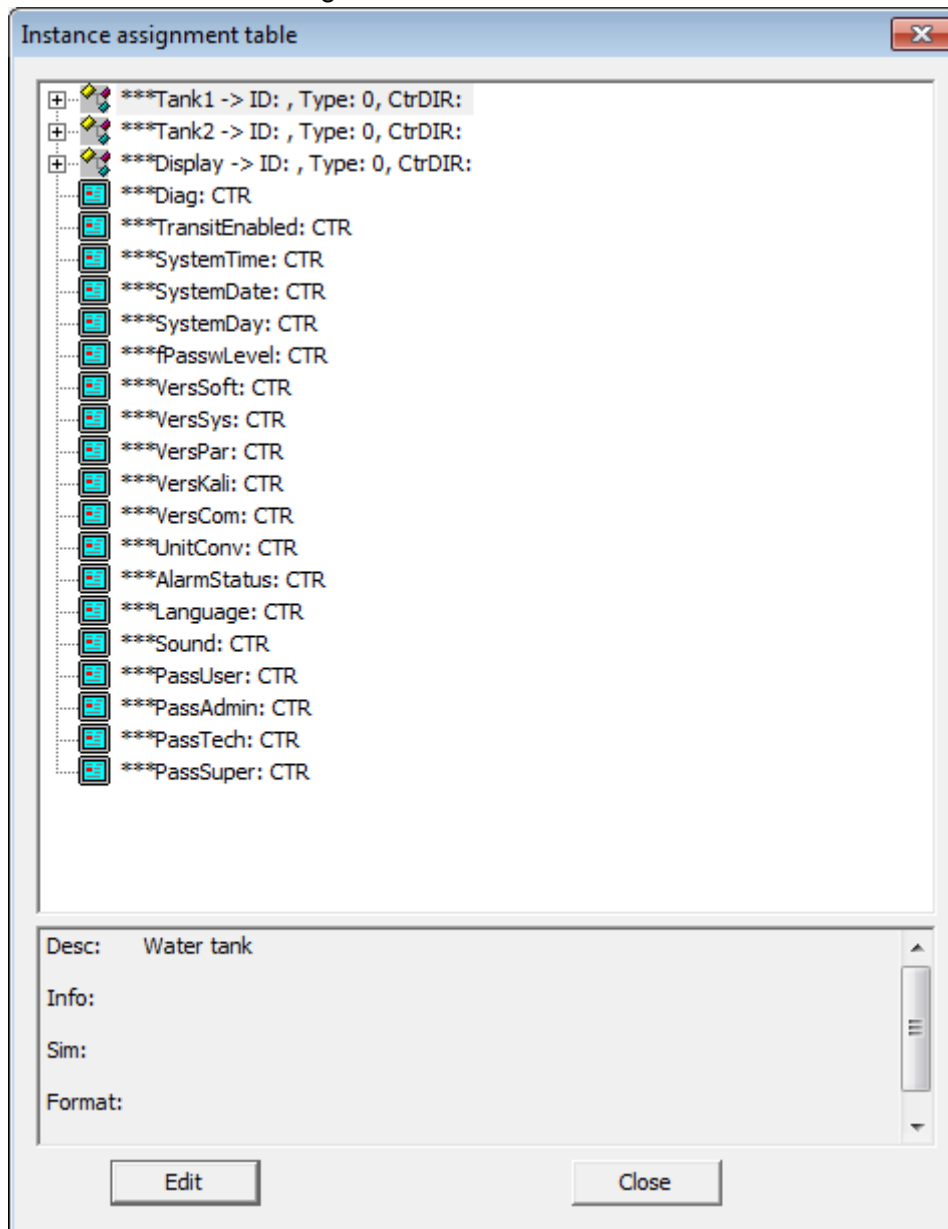
To do this, we first have to add an EntityTab to our project. *<Shift + right click>* on SystemDef and select EntityTab:



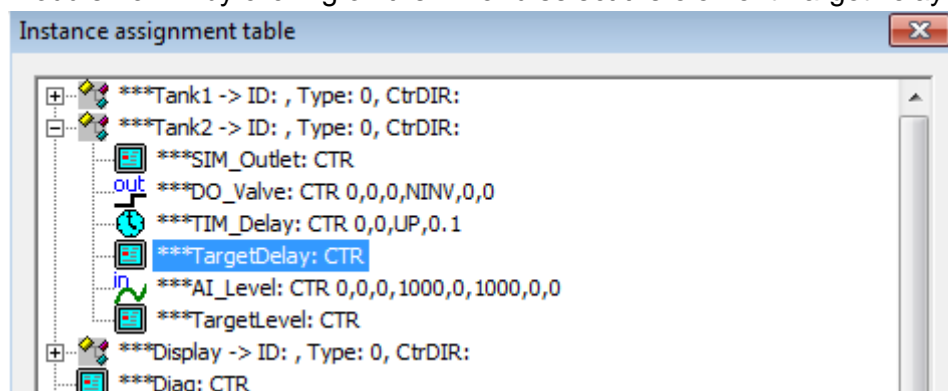
In the property window select Edit for the instance assignment table:



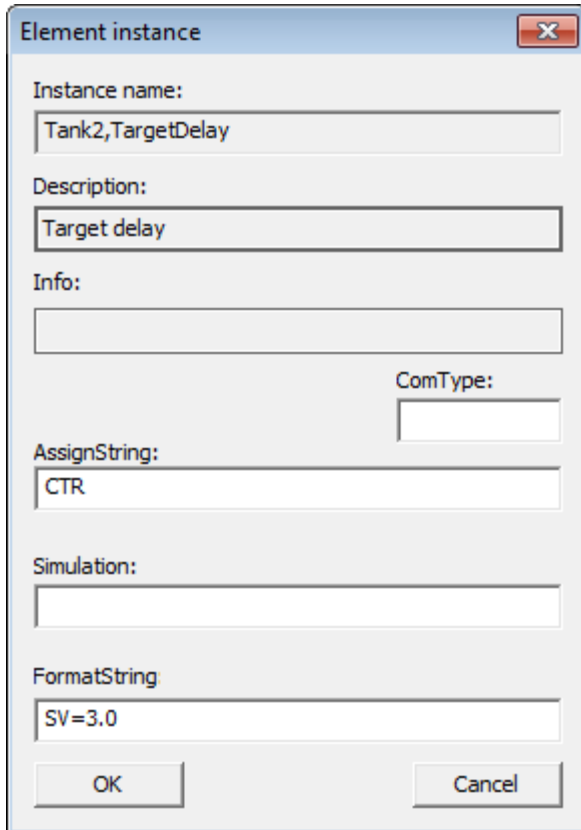
You will see the following window:



This window shows all elements in every submodule and the whole project tree. Expand the submodule Tank2 by clicking on the "+" and select the element TargetDelay:



You can edit the element by either using a *<click>* on the Edit-button at the bottom of the window or using a *<double click>* on the element. A new window will open:



Element instance

Instance name:  
Tank2,TargetDelay

Description:  
Target delay

Info:

ComType:

AssignString:  
CTR

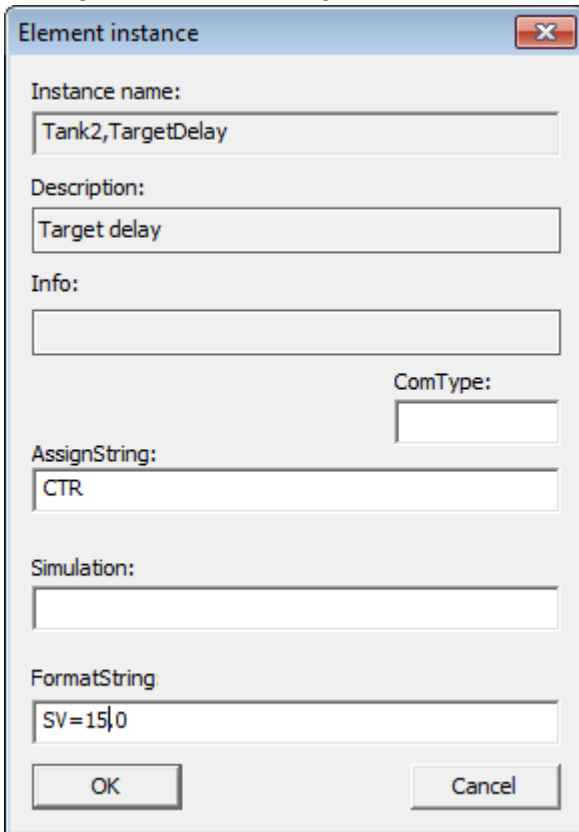
Simulation:

FormatString  
SV=3.0

OK Cancel

You can edit different settings of the element like the Description or the FormatString. Changes you made in this window, will only affect the element TargetDelay in the submodule Tank2.

Change the FormatString to SV=15.0:

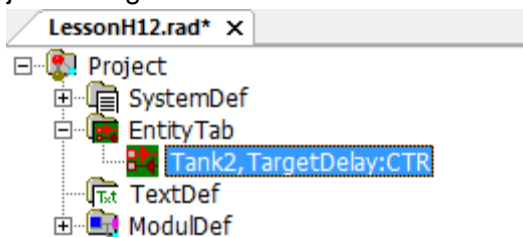


The 'Element instance' dialog box is shown with the following fields:

- Instance name: Tank2,TargetDelay
- Description: Target delay
- Info: (empty)
- ComType: (empty)
- AssignString: CTR
- Simulation: (empty)
- FormatString: SV=15.0

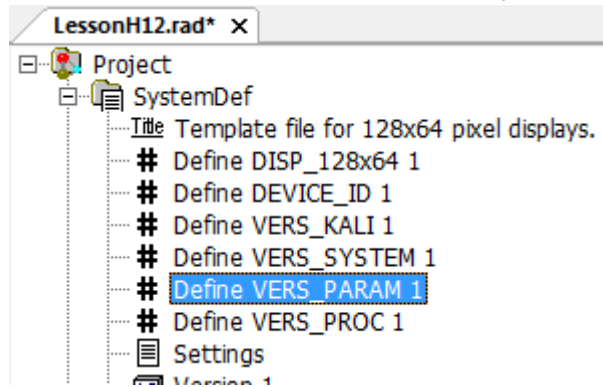
Buttons: OK, Cancel

Confirm with OK and Close the instance assignment table window. You will see the element you just changed is now also as subnode in the EntityTab-node:

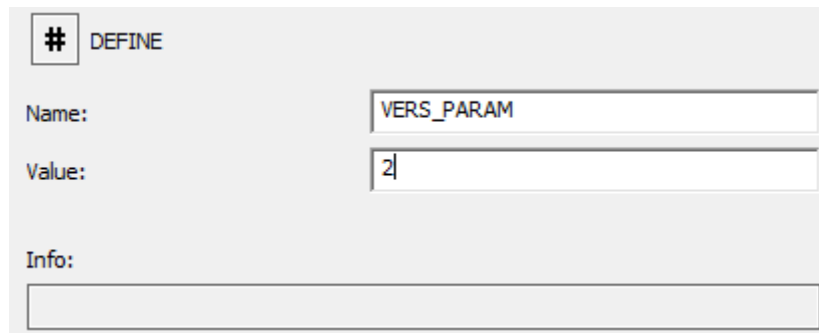


We now have changed the standard value of a parameter which is stored in permanent memory. However, if the application has run before, the old value is saved in the permanent memory and will be used, regardless of the standard value. We will have to enforce a reset of the permanent memory, to enforce using our new standard value.

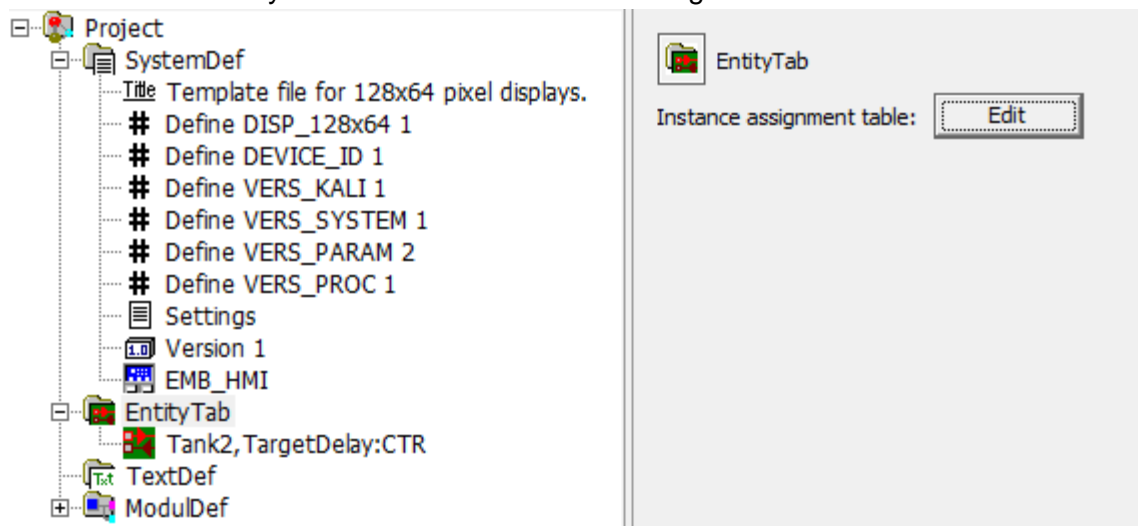
Select the Define VERS\_PARAM in SystemDef:



This Define is a version number for all parameters. This value is also saved in the permanent memory. If the value changes, the application will notice the new version on startup. The different version means something has changed in the parameter structure and the application will format the parameter area of the permanent memory and reset all parameters to default values. So change the value to 2:

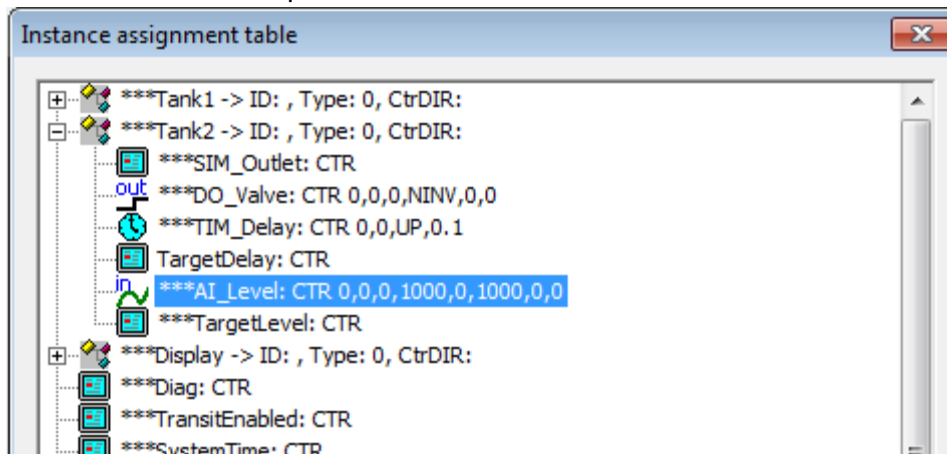


Go back to the EntityTab and select Edit for the Assignment instance table:

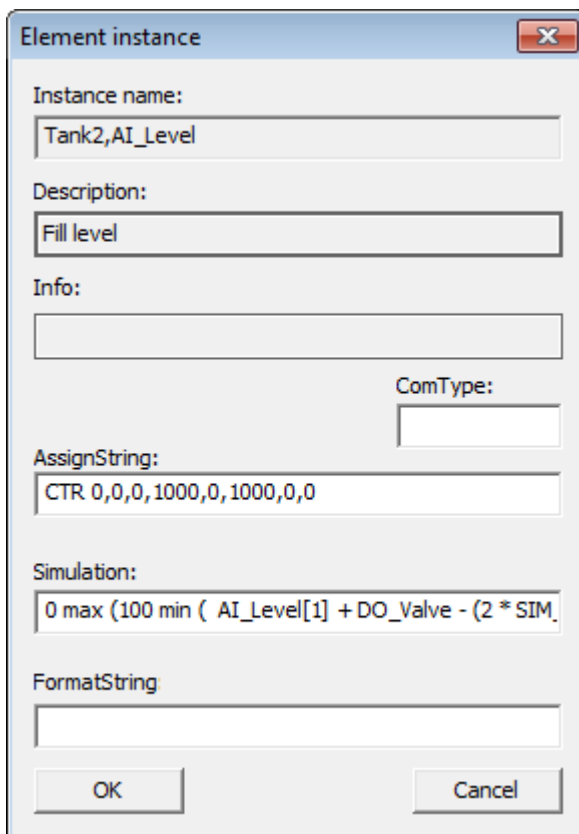




In the window now expand Tank2 and select AI\_Level:



Select Edit or <double click> on the element:



You see you can edit the AssignString of the IO-element. The AssignString contains parameters which are passed to the HAL to identify which real IO to use for an element. So the AssignString is determined by the HAL used, to connect the real hardware. Because two different submodules will most likely not use the same IO on the hardware, the AssignString is normally set in the EntityTab. So the EntityTab is an important tool to map the elements of radCASE to the real hardware.

Because we don't have any real hardware for this tutorial, we will leave the values as they are and just use Cancel and close the Instance assignment table again.

### 4.12.3 Conclusion

Hit <F4> to create and start the simulation. If you are observant you will notice the acid tank now starts filling at a later moment, in comparison with the water tank. If you miss it, you can just open the surfaces of both tanks and open the SIM\_Outlet to observe how the Acid tank opens the inlet significantly later than the Water tank.

## 4.13 Lesson H13: Inheritance

### 4.13.1 Goal

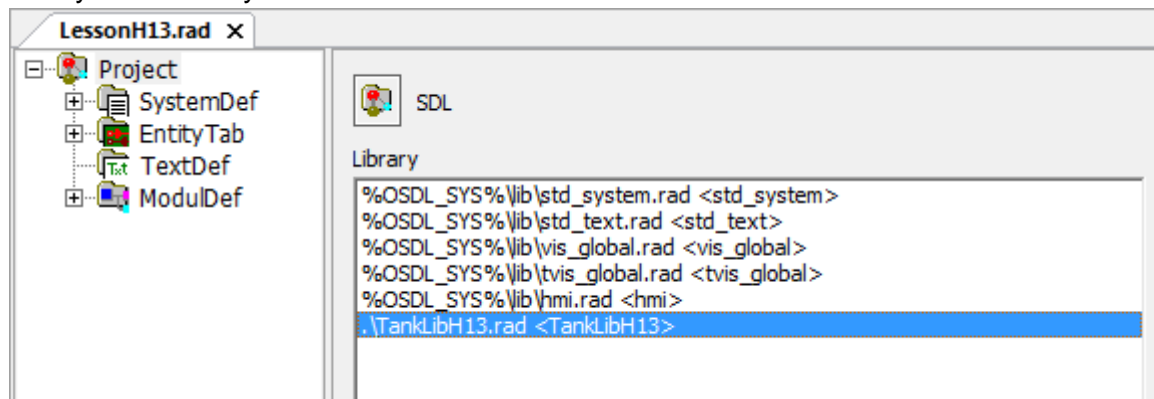
In this lesson you will learn, how to reuse code for similar functionalities using inheritance.

### 4.13.2 Content

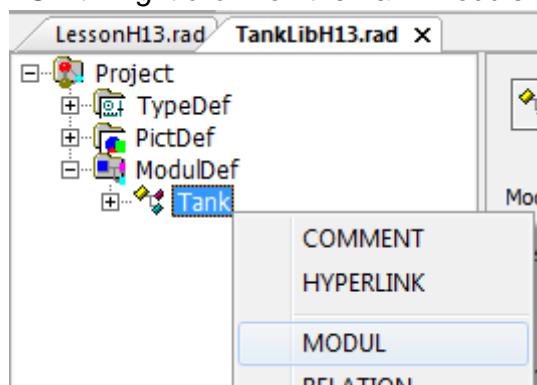
You can start this lesson by using the project you created in lesson H12 or by opening the project LessonH13 that is delivered with your radCASE copy.

In this lesson we want to change the acid tank, so it will have a LED which will light up, when the tank is empty. We can add this functionality, without having to copy the whole Tank module, using Inheritance.

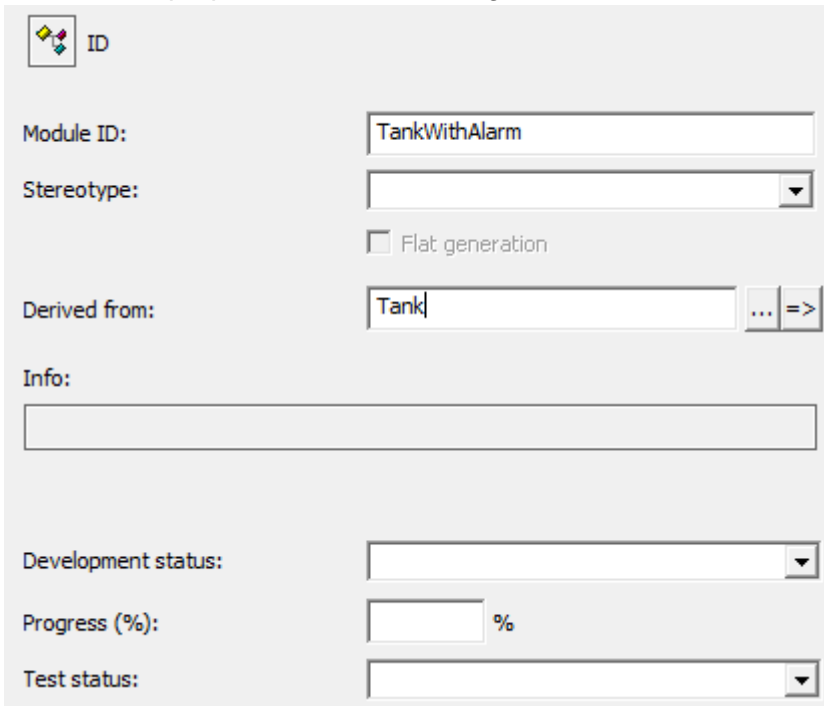
First we will create a new module in our tank library. Open the library, by *<double clicking>* the tank library in the library list:



*<Shift + right click>* on the Tank module and select **MODUL**:



Now set the properties to the following values:

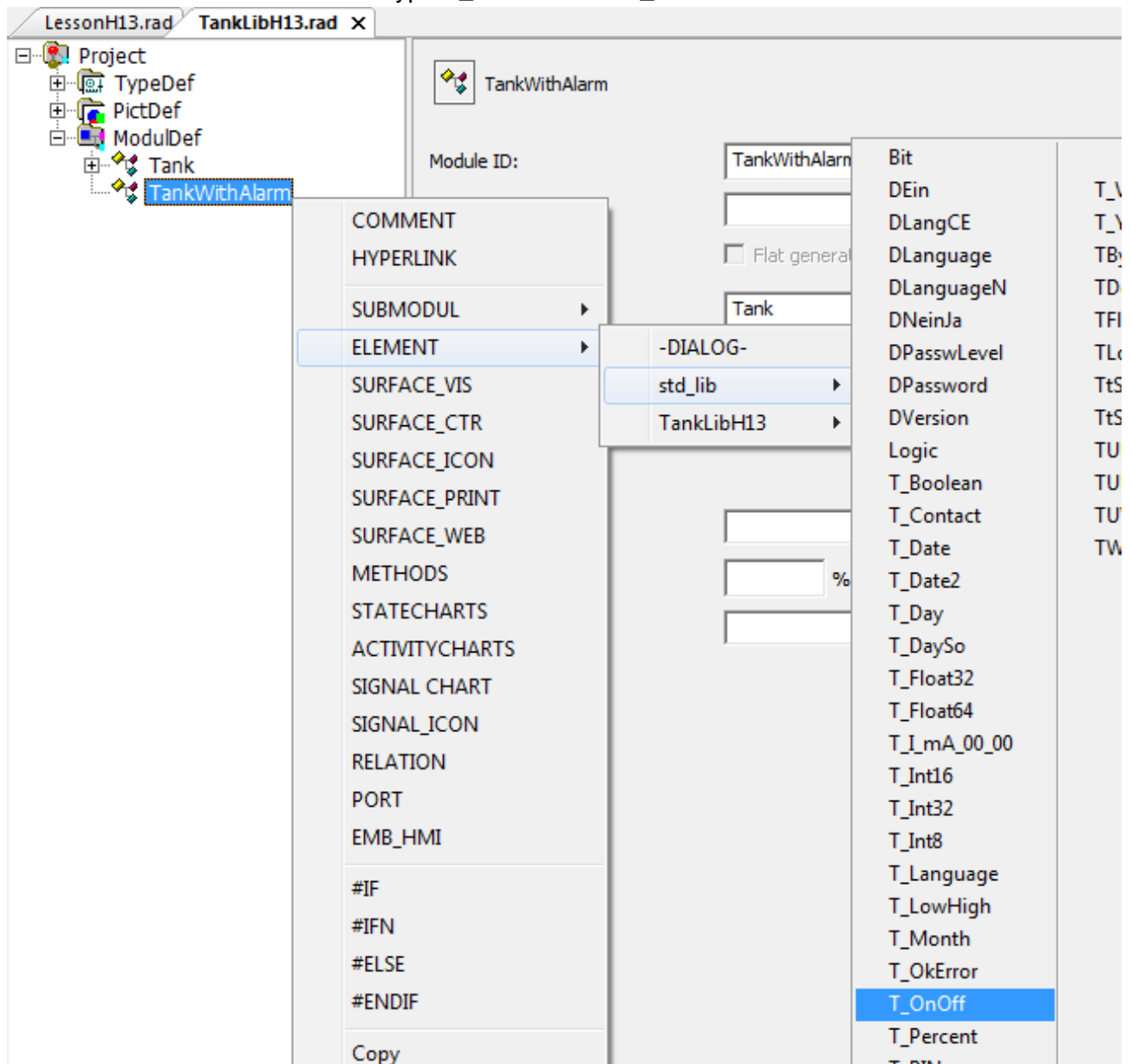


The screenshot shows the configuration window for a module in radCASE. At the top left is a small icon with four colored squares (yellow, red, green, blue) and the text 'ID'. Below this, the 'Module ID:' field contains 'TankWithAlarm'. The 'Stereotype:' field is an empty dropdown menu. Below it is a checkbox labeled 'Flat generation' which is unchecked. The 'Derived from:' field contains 'Tank' and has a button with three dots and an equals sign to its right. Below this is an 'Info:' section with a large empty text area. At the bottom, there are three more fields: 'Development status:' with an empty dropdown, 'Progress (%):' with an empty input field followed by a '%' symbol, and 'Test status:' with an empty dropdown.


This creates a new module TankWithAlarm which is derived from Tank. This means this module inherits all elements and functionality from the module Tank. At this point both modules do exactly the same and both have the same elements. TankWithAlarm is somewhat of a copy of the Tank module.

However we now can add elements, functionality, submodules and surfaces to TankWithAlarm and so expand the functionality of the original Tank module. It is also possible in some extent to change functionality, submodules and surfaces of TankWithAlarm.

We will now add the LED to the TankWithAlarm module. <Right click> the TankWithAlarm module and select an ELEMENT with the type T\_OnOff from std\_lib:



Set the properties of the element to the following values:

 DO\_Alarm

Name:

Type:  ... ?

Multiplicity:

Interface type:

Description:

Info:

Assign type/Com type:

Assign string:

☐ Runtime dynamic meta data

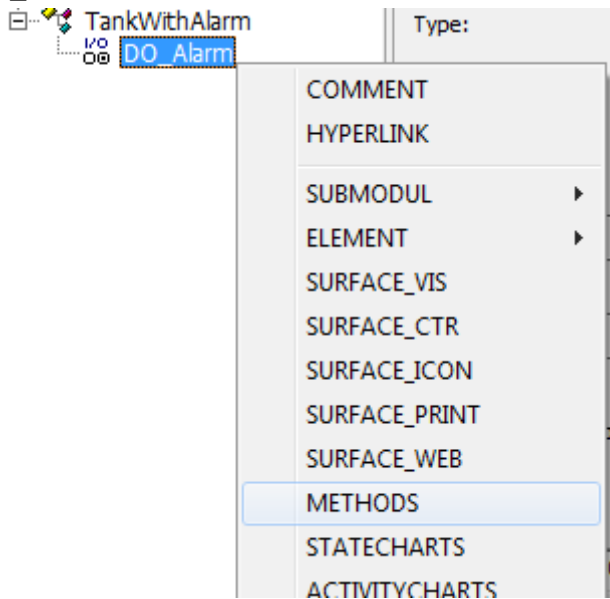
Simulation:

Format string:

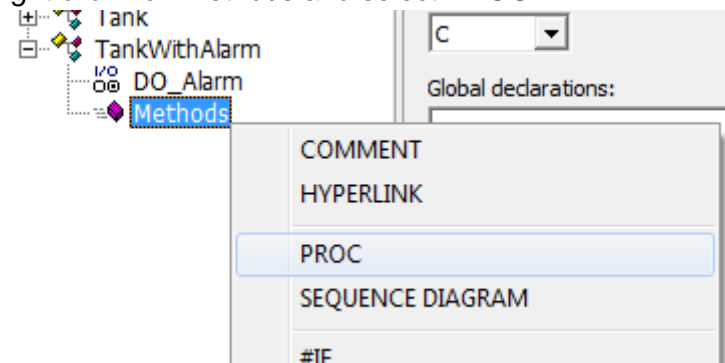
EProfiles:  ...

Documentation:

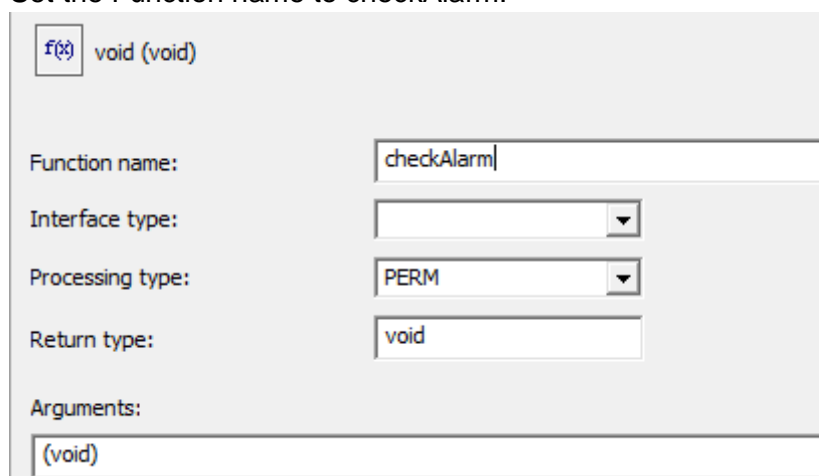
Now we want to light up that LED if the tank level is below 3%. <Shift + right click> on the element DO\_Alarm and select METHODS:



<Right click> on Methods and select PROC:



Set the Function name to checkAlarm:

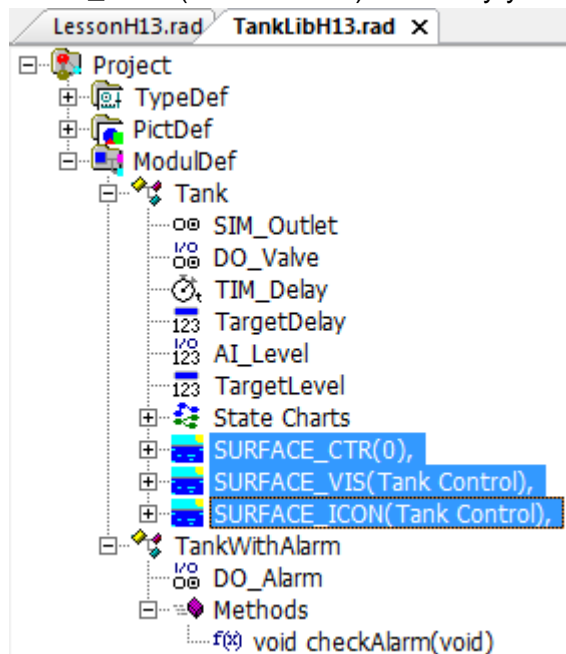


Add the following code to the function in the right pane of the editor:

```
{  
  if ($AI_Level < $3)  
    $DO_Alarm = $On;  
  else  
    $DO_Alarm = $Off;  
}
```

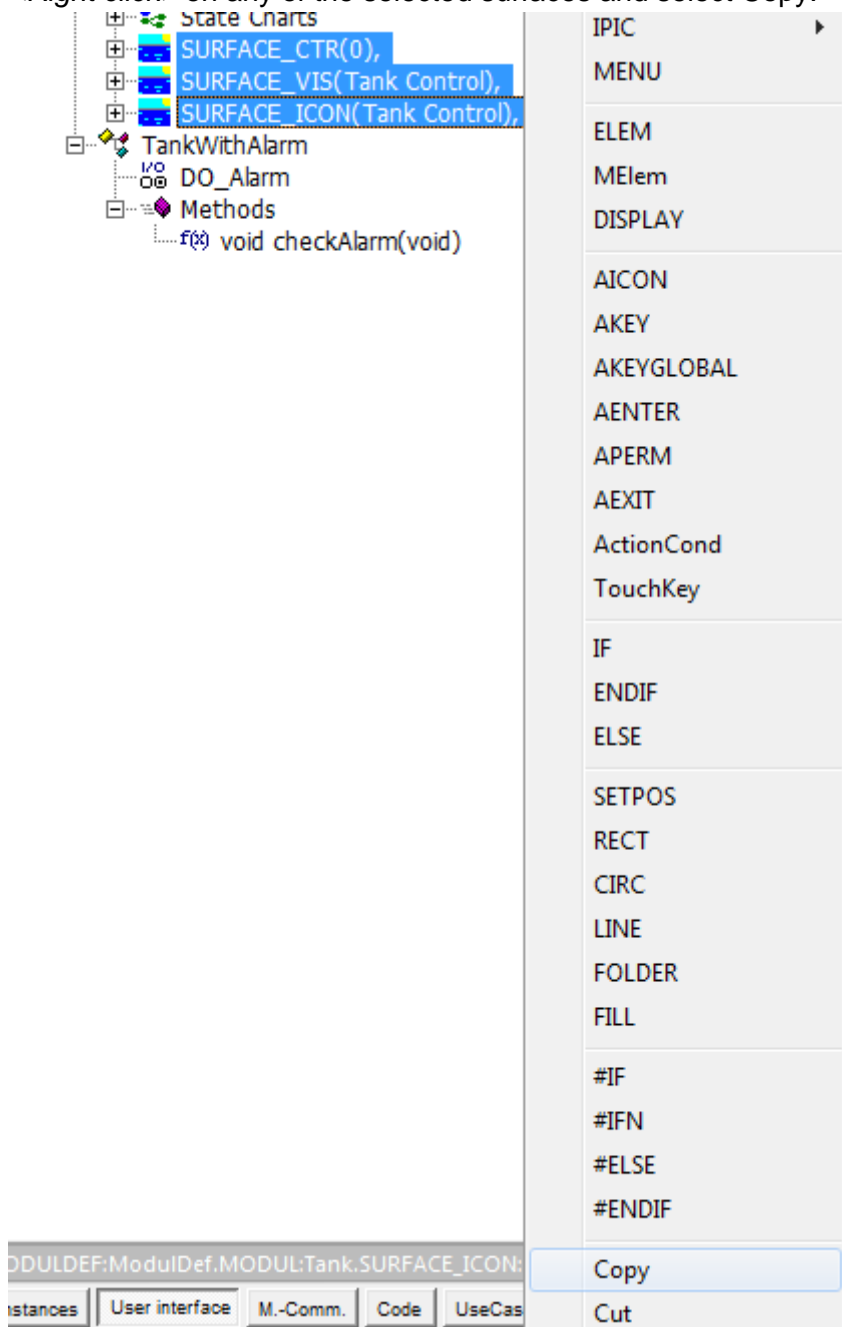
This way we added the whole functionality needed for the alarm LED. But we will only see the LED in the module view, because all of our Surfaces are defined in the Tank module, which does not know about the LED. So we will have to change all the surfaces.

Select the SURFACE\_CTR(0) of the Tank module and then <Shift + click> on the SURFACE\_ICON(Tank Control). This way you should select all three surfaces of the tank module:

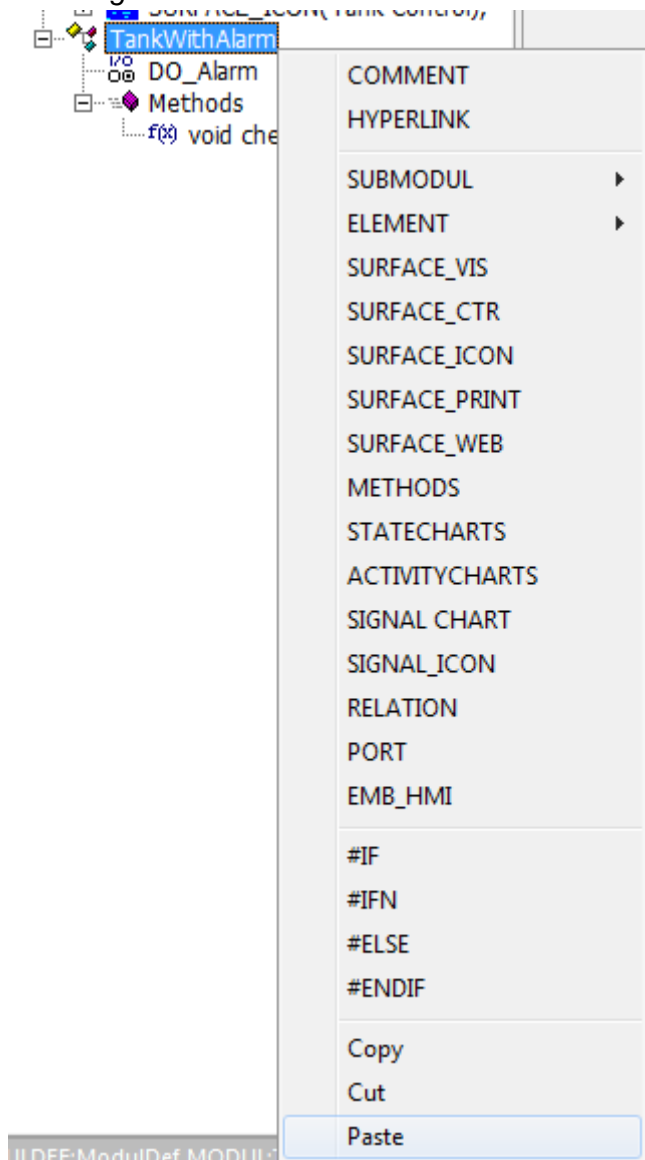




<Right click> on any of the selected surfaces and select Copy:

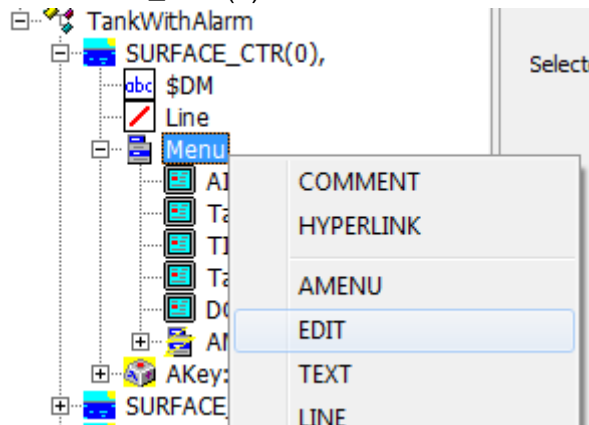


Now <right click> on the module TankWithAlarm and select Paste:

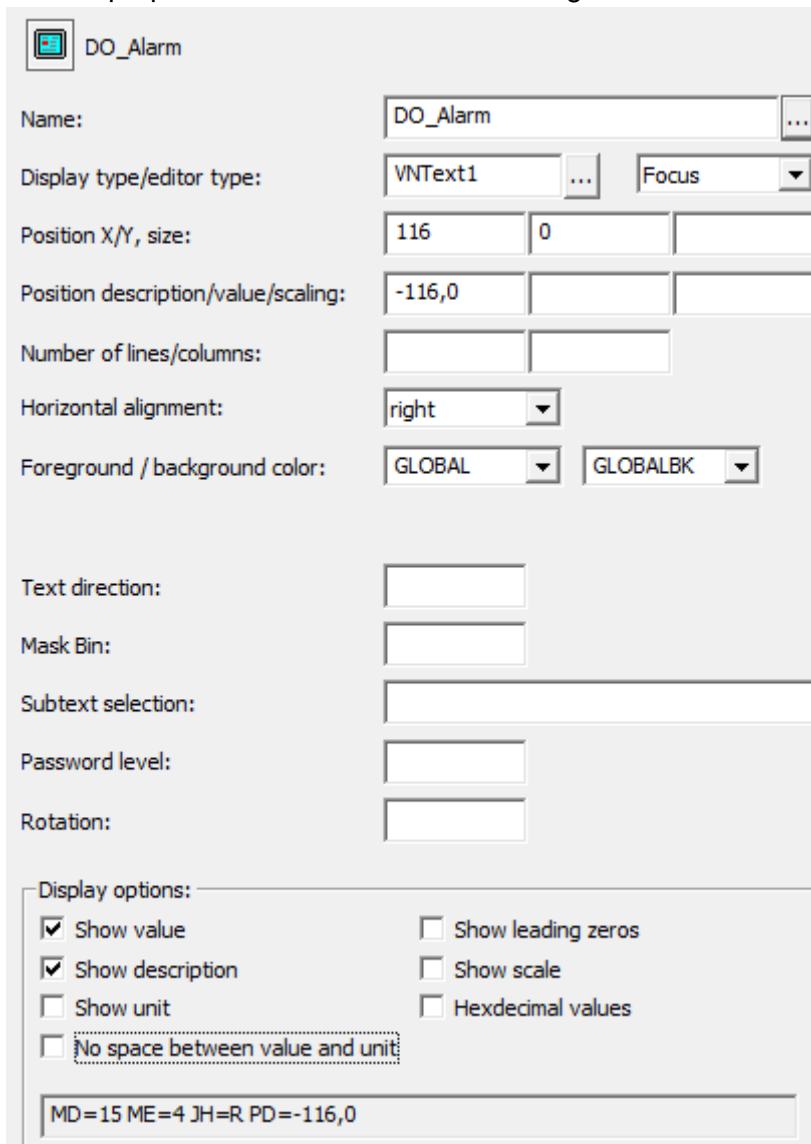


We have copied the surfaces into the TankWithAlarm module. Because the surfaces all have the same name as in the Tank module (base module) the surfaces from TankWithAlarm are used. This is called overwriting or overriding. This is the way to change behavior or surfaces in the derived module.

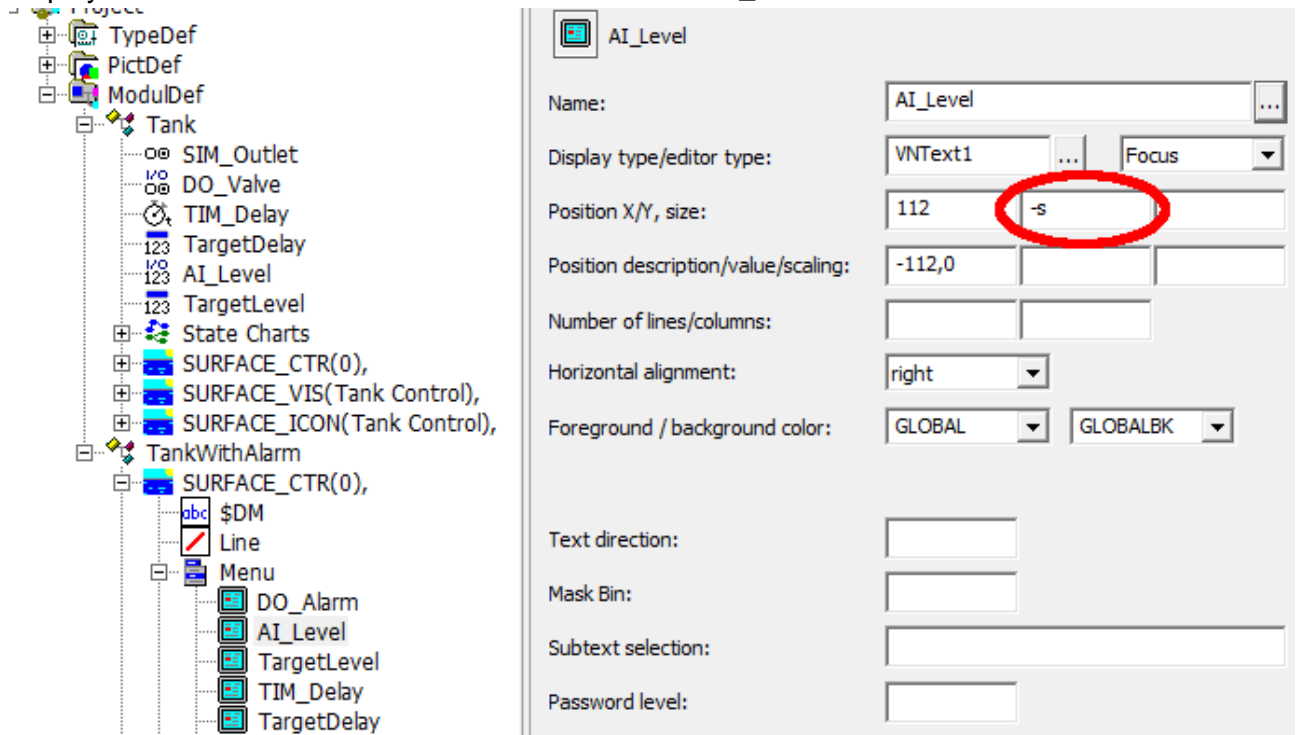
We will now add the visualization of the alarm to all of the surfaces. First *<right click>* on the Menu in SURFACE\_CTR(0) and select Edit:



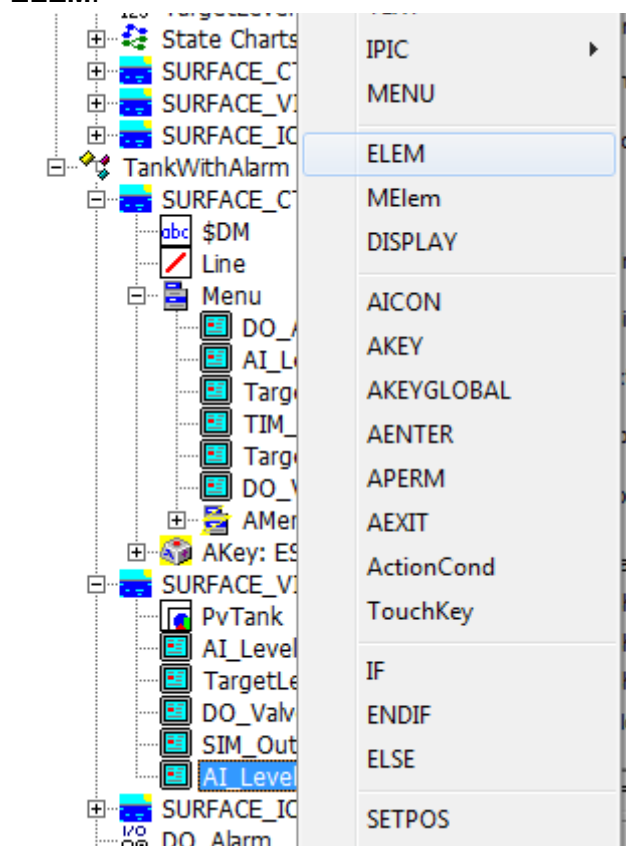
Set the properties of the Edit to the following values:

A screenshot of the 'DO\_Alarm' property dialog box. The dialog has various input fields and checkboxes. The 'Name' field is 'DO\_Alarm'. 'Display type/editor type' is 'VNTxt1' with a 'Focus' dropdown. 'Position X/Y, size' has fields for '116', '0', and an empty field. 'Position description/value/scaling' has fields for '-116,0' and two empty fields. 'Number of lines/columns' has two empty fields. 'Horizontal alignment' is 'right'. 'Foreground / background color' has 'GLOBAL' and 'GLOBALBK' dropdowns. 'Text direction' is an empty field. 'Mask Bin' is an empty field. 'Subtext selection' is an empty field. 'Password level' is an empty field. 'Rotation' is an empty field. A 'Display options' section contains checkboxes: 'Show value' (checked), 'Show leading zeros' (unchecked), 'Show description' (checked), 'Show scale' (unchecked), 'Show unit' (unchecked), 'Hexdecimal values' (unchecked), and 'No space between value and unit' (unchecked). At the bottom, a text box contains the string 'MD=15 ME=4 JH=R PD=-116,0'.


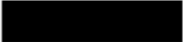

Because this is now the first item in the menu, we have to position the second line of the menu to be displayed below this new first line. Select the element `AI_Level` and set the Y-Position to `-s`:



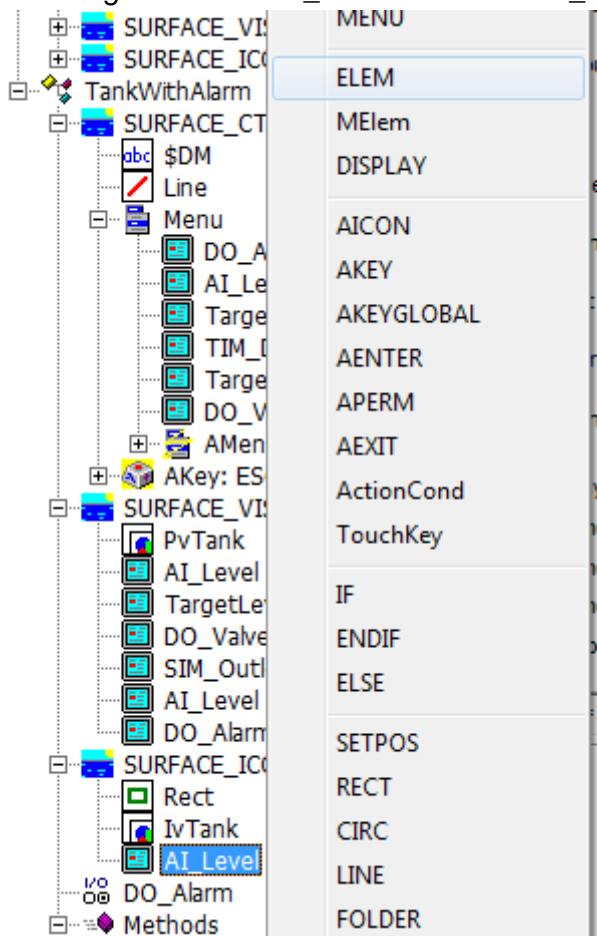
Now **<Shift + right click>** on the element `AI_Level` in `SURFACE_VIS(Tank Control)` and select **ELEM**:



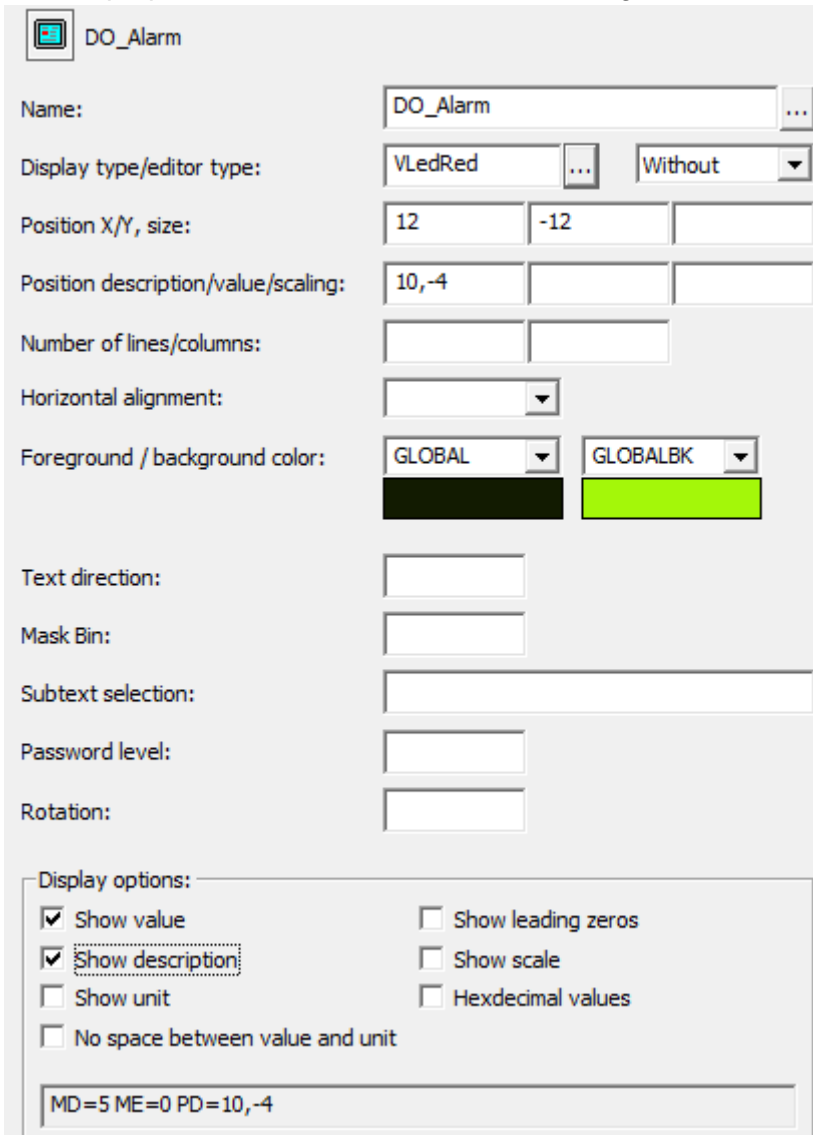
Set the properties of that element to the following values:

 ELEM  
  
Name: DO\_Alarm ...  
Display type/editor type: VLedRed ... Without ▾  
Position X/Y, size: 101 -35  
Position description/value/scaling: 10,-4  
Number of lines/columns:   
Horizontal alignment: ▾  
Foreground / background color: GLOBAL ▾ GLOBALBK ▾  
   
  
Text direction:   
Mask Bin:   
Subtext selection:   
Password level:   
Rotation:   
  
Display options:  
☒ Show value ☐ Show leading zeros  
☒ Show description ☐ Show scale  
☐ Show unit ☐ Hexdecimal values  
☐ No space between value and unit  
  
MD=5 PD=10,-4

<Shift + right click> on AI\_Level in SURFACE\_ICON(Tank Control) and select ELEM:



Set the properties of the element to the following values:



**DO\_Alarm**

Name:

Display type/editor type:

Position X/Y, size:

Position description/value/scaling:

Number of lines/columns:

Horizontal alignment:

Foreground / background color:

Text direction:

Mask Bin:

Subtext selection:

Password level:

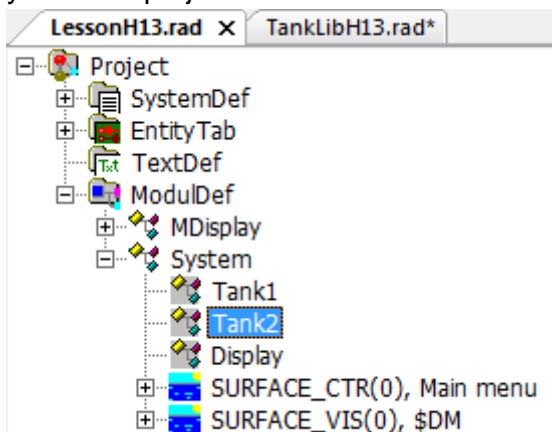
Rotation:

Display options:

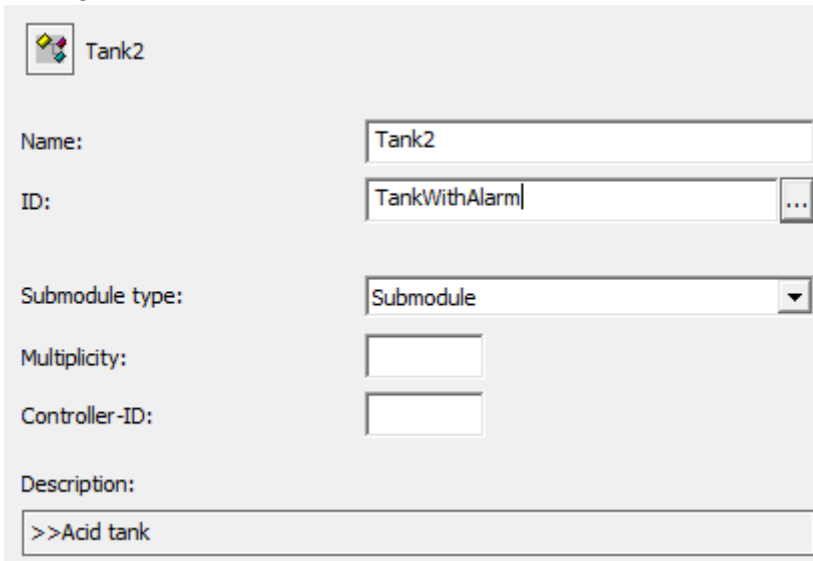
- ☒ Show value
- ☒ Show description
- ☐ Show unit
- ☐ No space between value and unit
- ☐ Show leading zeros
- ☐ Show scale
- ☐ Hexdecimal values

MD=5 ME=0 PD=10,-4

At last we have to change the Acid tank to a TankWithAlarm instead of being just a Tank. Switch to your main project file LessonH13.rad and select Tank2 in the System module:

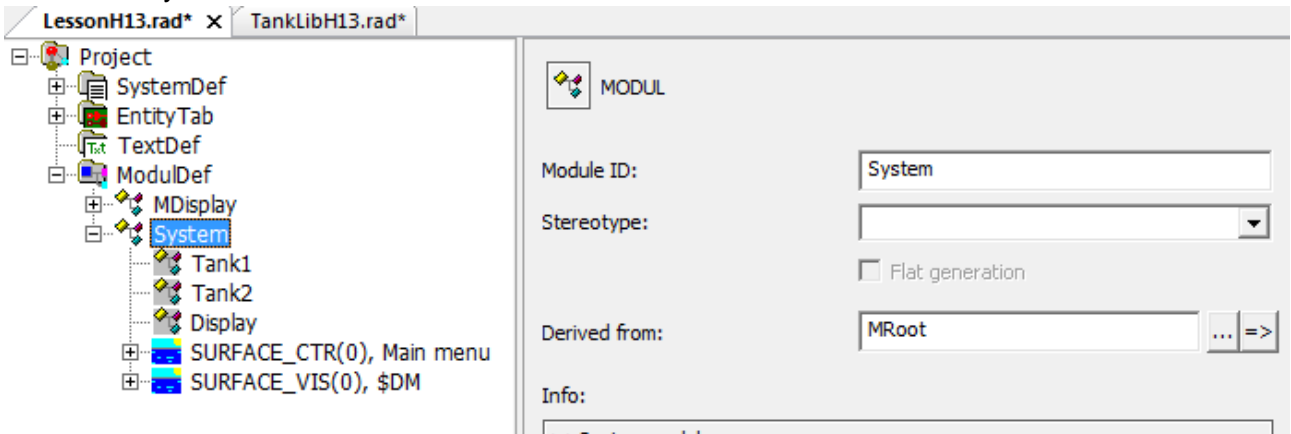


Change the ID of Tank2 to TankWithAlarm:



The screenshot shows the configuration dialog for 'Tank2'. The 'Name' field is 'Tank2'. The 'ID' field is 'TankWithAlarm'. The 'Submodule type' is 'Submodule'. The 'Multiplicity' and 'Controller-ID' fields are empty. The 'Description' field contains '>>Acid tank'.

Select the System module:



The screenshot shows the radCASE interface. On the left, the 'Project' tree is expanded, showing the 'System' module selected. On the right, the 'MODUL' configuration dialog is open. The 'Module ID' is 'System'. The 'Stereotype' is empty. The 'Flat generation' checkbox is unchecked. The 'Derived from' field is 'MRoot'. The 'Info' field is empty.

You can see the System module is also derived from another module name MRoot. There is some basic radCASE functionality in that module. This is where all those elements in the module view of the System module, you saw in Lesson H01, came from.



### 4.13.3 Conclusion

Hit <F4> to create and start the simulation. You will now see the LED lighting up in all the surfaces of the acid tank, as soon as the tank is empty:

