

# RTA Application Note - SWC Calibration Configuring a calibrateable parameters on a SWC

RTA-CAR



### Copyright

The data in this document may not be altered or amended without special notification from ETAS GmbH. ETAS GmbH undertakes no further obligation in relation to this document. The software described in it can only be used if the customer is in possession of a general license agreement or single license. Using and copying is only allowed in concurrence with the specifications stipulated in the contract. Under no circumstances may any part of this document be copied, reproduced, transmitted, stored in a retrieval system or translated into another language without the express written permission of ETAS GmbH.

©Copyright 2022 ETAS GmbH, Stuttgart.

The names and designations used in this document are trademarks or brands belonging to the respective owners.

**Document:** AN-000/EN-01-2022

Configuring a calibrateable parameters on a SWC

### Contents

1	Introduction	4	4
2	AR Calibration overview         2.1 SWC specific calibration		<b>5</b> 5
3	Sample project overview	•	7
4	Workflow description4.1 SwAddrMethod definition4.2 PerInstanceParameter creation4.3 Runnable configuration4.4 Important notes4.5 RTE generation	. 1 . 1	9 0 1
5	Initialized RAM Method           5.1 Important notes on this method	<b>1</b> 2	_
6	Single Pointered method	14	4
7	Single Pointered 2 method	1	5
8	Double Pointered method	1	6
9	Reading parameters from SWC	1	7
10	Contact, Support and Problem Reporting	19	9



Configuring a calibrateable parameters on a SWC

### 1 Introduction

This application note describes how to configure an existing ISOLAR project using ISOLAR A/B 9.2.0 to add the calibration of some parameters specific to a SWC. The document contains an explanation step by step of the workflow to follow to obtain a working project. A sample project is provided along with this document configured as shown in this application note. As starting project the RTA-SK\_VRTA\_GCC\_Standard\_9.2.0\_R1.zip is used, all the steps listed below can be performed on this project to preproduce the sample project.



Configuring a calibrateable parameters on a SWC

### 2 AR Calibration overview

AUTOSAR groups calibrations into three main categories based on the accesseability of the parameters. Calibration parameters can be:

- SWC instance specific
- SWCType specific but shared among all the SWC instances of the same type
- Global: shared among different SWC types

Thus, there are three AR artifacts corresponding to these three categories:

- SWC specific parameters are implemented in AR using a particular type of ParameterDataPrototype named perInstanceParameters; these are defined for each SWC instance
- parameters shared among SWC instances of the same SWC type are implemented with two other **ParameterDataPrototype**: **constant memories** and **shared parameters**
- global parameters are implemented using SWC R ports and a **ParameterSWC** offering all the PPorts to the SWCs

#### 2.1 SWC specific calibration

This AN focuses on the first category only. Calibration of SWC specific parameters involves two main steps:

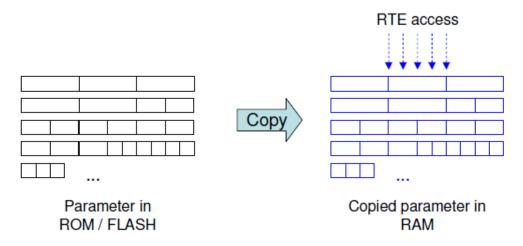
- 1. configuration of SWC description
- 2. RTE generation based on proper options

Four different methods to calibrate the SWC specific parameters will be explained. Most of the configuration is common to all of them; while the RTE generation differs for each method.

"Workflow description" chapter describes the configuration steps while the chapters after analyse the output code of each method.

The four calibration methods used are:

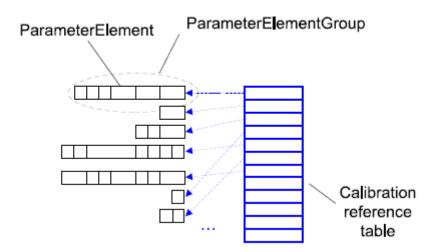
 Initialized RAM: all SWC parameters are stored in a Flash Memory Section; during the ECU startup all these parameters are copied into a user defined RAM Memory section. The latter will be used at runtime from the SWC's runnables to read the parameters values.



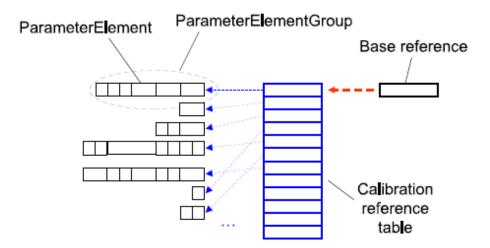
2. **Single Pointered**: a table of pointers to Flash Memory Addresses is created in a user defined RAM Memory section; at runtime the SWC makes use of the pointers to read the



parameters values.



 Double Pointered: a table of pointers to Flash Memory Addresses is created in a user defined section of Flash Memory while the RAM Memory will have a BaseAddressReference which is a pointer to the starting address of the Flash Memory Table; theBaseAddress and eventually an offset is used to access the parameters values.



4. **Single Pointered Variant**: this method is another implementation of the Single Pointered one; it is realized using an RTA-RTE extension of AR which allows to define several parameters memory sections for each SWC.



Configuring a calibrateable parameters on a SWC

### 3 Sample project overview

In the example used for this AN we used two parameters of two different BaseType:

- Param\_1 -> uint8
- Param\_2 -> uint16

These parameters will be set to an initial value in Flash memory of:

- Param\_1 = 5
- Param\_2 = 2000

## ETAS DRIVING EMBEDDED EXCELLENCE

#### **RTA Application Note - SWC Calibration** Configuring a calibrateable parameters on a SWC

### 4 Workflow description

#### 4.1 SwAddrMethod definition

Since the parameter will be placed in a defined memory section we need to create a new SwAddrMethod which will be used to link each parameters to a defined RAM memory section. In the AR explorer menu go to "Infrastructure", right click and select **New SwType -> Elements | SwAddrMethod**:

🖻 AR Explorer 🛛 🐧	📮 Filesystem Navigator 👘 🗖	][		
	E 🕏 🤀 🖶 🥙 X 🕴	8		
> 🔛 NvM	· · · · · · · · · · · · · · · · · · ·	^		
> 🔨 rba_D	liagLib			
> 🗉 Tester	-			
> 🖬 WdgN				
> 🗉 XcpS\				
> 🗟 Compos	itions			
👻 🗟 Infras	Create Component Properties	>		
> 🗟 Co	Create Unit	>		
> 🗟 Da	Create Physical Dimensions	>		
> 🗟 Un	Create Mode Declaration	>		
> 🖨 Ph	Create Measurement Calibration Support Function	>		
> 🗟 Mc	Create Constant Specification Create SwType	>	Elements   Sw. Addr Method	
		/	Elements   Sw Addr Method     Elements   Sw Axis Type	
> 🗟 Sw 🔶	Create Elements   Data Constr		Elements   Sw Record Layout	
> ₩ System ♦	Create Elements   Calibration Parameter Value Set			
& Description View	Create Elements   Interpolation Routine Mapping Set     Create Elements   Blueprint Mapping Set			
<sup>6</sup> <sup>6</sup> Description Vie ◆				
÷	Create Elements   Set			
📌 Favorites 🛛 💊	Create Elements   Port Prototype Blueprint			
🗢 ECU Configura 🔶	Create Elements   Rapid Prototyping Scenario			
VRTA Standard	(Project)	×	-	
	a production and			

Figure 4.1: New SwAddrMethod

Open the created item and set the properties as desired; for Section Type choose "CALPRM" since the data to store are calibration parameters.



Configuring a calibrateable parameters on a SWC

ARAM_SECTION [SwAddrM	lethod			
	▲ ■ I <sup>a</sup> z ◆ <u>PARAM_SECTION</u>			
ype filter text	<ul> <li>Attributes</li> </ul>			
♦ PARAM_SECTION	ShortName*	PARAM_SECTION		
	Category			80
	ShortNamePattern			80
	MemoryAllocationKeyw ordPolicy	ADDR-METHOD-SHORT-NAME	~	8
	SectionInitializationPolicy	/ INIT		8
	SectionType	CALPRM	$\checkmark$	80
	Options		•	

#### 4.2 PerInstanceParameter creation

We will add these parameters to the master SWC so that later we can test reading these parameters using CAN, however, you may want to add them to a XCP if your want to use them with INCA.

Open the existing SWC "Master\_SWC" and switch to tab **DataProtoypes**; here go to Tab **PerInstanceParameter** and then click **Add**;

🖷 AR Explorer 🛛 💊 Filesystem Navigator 📃 🔍	SWC Editor: MasterSWC ~							- 0
E % Ø # 6 % * 1	SWC Editor: MasterSWC							÷ (?)
> 11 DiagnosticSWC								
> 目 E2ESWC	Name MasterSWC			InternalBehavior		~	] <b>+ ∥ × +</b> <u>∨P.</u> <b>×</b> <u>∨P.</u>	
> 🎫 EcuM	Main 🐼 Functions 🖋 Events 🗵	IRV DataPrototypes 2)the	rs					
> III FiM	- Hand - Failedons - Erens -	and basis local best						-11
> 🗄 MasterSWC 🚺	ConstantMemories							
MemorySWC     MemorySWC	3 PerInstanceParameter	Add × Delete						
> 🛍 NVM		Sh.4.g: 0/0 ■ C >	C Inthe				🕞 🗆 RE	
> TesterSWC	StaticMemories	Showing: 0/0	Ø Filter :				UN LI KE	
> 🖬 WdgM	ArTypedPerlnceMemories	Shortname	Type	Init Value	Calibration Access	Display Format	CompuMethod E	
> T XcpSWC	Artypedrennceiviemones							
> 🤒 Compositions			V	V	7 7	7 7	7 7	
> 😣 Infrastructure								
e % 👗 🖽 🕴								
* Favorites *								
ECU Configurator *								
✓ ♥ VRTA_Standard [Project] ^								
> 🖶 Parameter Definitions								
<ul> <li>Bsw Modules</li> </ul>		د					>	
<ul> <li>Com Stack</li> </ul>								
> 🗟 Can Modules	Component DataTypes Interface Co	man Matheral Data Canatasia						_

Then begin filling the fields in as required. Here is the example steps to create our "param1" with a uint8 type and inital value of 5:



Configuring a calibrateable parameters on a SWC

📟 Add Data Prototype	- 0	I	×
Add Perinstance	e <b>Parameter</b> o create PerInstanceParameter		
Shortname*	Param1		
Туре*	uint8	~	۹
Calibration Access	READ-WRITE	~	
Display Format			
CompuMethod		~	۹
Data Constr		~	۹
Sw Addr Method	PARAM_SECTION	~	۹
?	< Back Next > 2 Finish Cancel		

Then click **Next** to configure a **NumericalInitValue** with no constants for init value and give it the value of **5**.

You should repeat this step to produce **param2** with a value of 2000 and a type of **unit16**, then you should have 2 params configured which look like this.

Shortname	Туре		Init Value	Calibration Access	Display Format	CompuMethod	Data Constr	Sw Addr Meth
	$\nabla$	7	7	$\nabla$	7	V	V	
Param2	uint16		NumericalVal	READ-WRITE				PARAM_SEC.
Param1	uint8		NumericalVal	READ-WRITE -				PARAM_SEC

#### 4.3 Runnable configuration

Each runnable defined in the SWC must be configured to have access to the parameter; to do this, open the SWC with the Component Editor, go to **Functions** Tab, select the desired runnable and on the right open the **ParameterAccessPoints** Tab to add the access point to the parameters:



Configuring a calibrateable parameters on a SWC

ime MasterSWC		rnalBehavior			✓ + <sup>ℓ</sup> × +	<u>vp</u> × <sub>vp</sub>
Main 🗛 Function Events 🗵 IRV DataPrototypes	Others					
Runnables × Delete			r Mode Points Trigger Po	ints Parameter Access	Poi Wait Points	»3
iowing: 2/2 🛛 🖲 🗢 🧭 Filter : Runnable Entity Function Name	VP Short Label	<u>s</u> × <u>Delete</u> I ● C > Ø Filte				🕞 🗆 RE
<b>V</b>	Short N		Local Parameter	Parameter Data	Accessing Por	
RE_RunTest 2 RE_RunTest RE TimeTest RE TimeTest		V		7	7	V
		Select Para Parameter Reference St	rameters(s) to be set a meters(s) to be set as refe Access(s) to be created		x	
	>	type filter te				
ponent DataTypes Interface CompuMethod DataCons			Param1 Param2			1 🗆 - 📑 -
nsoles to display at this time.		Indicates	already configured and	not possible to edit	t	

If you can't see the tab called "Parameter Access Points" in the right hand table you may need to click on the double right arrows to show more more tabs.

#### 4.4 Important notes

If a parameter is not accessed by any runnable the RTA-RTE optimization mechanism does not create any macro to read the parameter from the SWC.

#### 4.5 RTE generation

This final step is dependant on the calibration method chosen; the RTE must be generated using the additional command "–calibration-method" with the value set to the desired method. The four options supported by the RTA RTE are:

- "initializedRam" RTA-RTE generates a RAM copy of all parameters and an initializing ROM block. Generated RTE APIs access the RAM copy.
- "singlePointered" RTA-RTE generates a reference table (in RAM) that locates calibration data. The calibration values returned by individual generated RTE APIs can be modified by changing pointers in the reference table to point to alternate values in either RAM or ROM.
- "singlePointered2" RTA-RTE generates individual reference pointers (in RAM) that locate calibration data. The calibration values returned by individual generated RTE APIs can be modified by changing those reference pointers. This is a non-AUTOSARextension.
- "doublePointered" RTA-RTE generates a reference table (in ROM) and a reference base (in RAM) that points to the table. The calibration values returned by all generated RTE APIs can be modified by changing the base pointer to reference a different table.

As default the configuration for the RTE module will include a parameter

RteCalibrationSupport which as has a default value of NONE and a lower multiplicity of NONE. This means that if you use any of the above commandline options you will get a warning that is recognises 2 different global definitions for calibration method. This is expected and ok as the commandline option takes precedence over the option in RteCalibrationSupport.



Configuring a calibrateable parameters on a SWC

### 5 Initialized RAM Method

╘

Generating the RTE for this method (additional command to RTE generation: "-calibration-method=INITIALIZED\_RAM") will give as output an Rte.c file where the following items should be defined:

- RAM memory section for all the parameters
- Flash memory section with initial values of parameters

```
/* RTE GLOBALS START */
 #define RTE START SEC CALPRM 16
 #include "Rte_MemMap.h" /*lint !e537 permit multiple inclusion */
 VAR(Rte_CalprmInitRAMType, RTE_DATA) Rte_CalprmInitRAM;
 #define RTE STOP SEC CALPRM 16
 #include "Rte MemMap.h" /*lint !e537 permit multiple inclusion */
 #define RTE START SEC CONST 16
 #include "Rte_MemMap.h" /*lint !e537 permit multiple inclusion */
 CONST(Rte_CalprmInitRAMType, RTE_CONST) Rte_CalprmInitROM = {
    ł
       2000,
       5
    }
 };
 #define RTE STOP SEC CONST 16
 #include "Rte MemMap.h" /*lint !e537 permit multiple inclusion */
 #define RTE START SEC VAR CLEARED 16
 #include "Dte MemMan h" /*lint 19537 nermit multiple inclusion */

    During Rte startup phase, the copy from Flash (ROM) memory to RAM memory is

  implemented in Rte Lib.c:
FUNC(STO RETURNIYPE, KIE CODE)
                                   /* [$Satisfies $SWS 2569] */
Rte_Start(void)
{
  Std_ReturnType rtn = RTE_E_OK;
                                                /* [$Satisfies $SWS 1261] */
```

5.1 Important notes on this method

Rte Initialized = TRUE;

Rte StartHook Start();

/\* [\$Satisfies \$SWS 3915] \*/

When using calibration method "Init-RAM", RTA-RTE creates a single combined RAM block containing space for all the calibration parameters and a single combined ROM block with the initializer for every calibration parameter. Both blocks use the structure type **Rte\_CalprmInitRAMType**. The RAM block is instantiated as a variable of that type called **Rte\_CalprmInitRAM** and the ROM block is instantiated as a constant of that type called

Rte\_memcpy( &Rte\_CalprmInitRAM, &Rte\_CalprmInitROM, sizeof( Rte\_CalprmInitRAMType ) );

/\* [\$Satisfies \$SWS 1238] [\$Satisfies \$

/\* [\$Satisfies \$SWS 2535] [\$Satisfies \$



**Rte\_CalprmInitROM**. Within the blocks the parameters are grouped together by SWC and SwAddrMethod names. This scheme allows a single copy operation to perform the initialization of all parameters at once.

**NB:** this method is advisable when a low number of parameters have to be calibrated, besides startup copy from Flash to RAM time must be considered to enable fulfilling of ECU startup timing requirements.



Configuring a calibrateable parameters on a SWC

### 6 Single Pointered method

To use this method, RTE must be generated using the additional command "-calibration-method SINGLE\_POINTERED". The output code will define a specific memory section in Flash named using the SwAddrMethod associated to DatPrototypes where calibration values will be stored:

```
#define RTE_START_SEC_MyParamSection
#define RTE_MemMap.h" /*lint !e537 permit multiple inclusion */
volatile CONST(Rte_CGTi_CPT_CanCommunication_SWC_EcuA_MyParamSection, RTE_MyParamSection) Rte_CGi_CPT_CanCommunication_SWC_EcuA_MyParamSection = {
    2000,
    5
};
#define RTE_STOP_SEC_MyParamSection
#define RT
```

In RAM a reference table is created with a variable storing the Flash address of the parameters memory section start:

```
#define RTE_START_SEC_VAR_INIT_UNSPECIFIED
#include "Rte_MemMap.h" /*lint !e537 permit multiple inclusion */
VAR(RteParameterRefTabType, RTE_DATA) RteParameterRefTab = {
    (RteCalprmRefTabEntryType)&Rte_CGi_CPT_CanCommunication_SWC_EcuA_MyParamSection
};
#define RTE_STOP_SEC_VAR_INIT_UNSPECIFIED
#include "Rte_MemMap.h" /*lint !e537 permit multiple inclusion */
```

**NB:** in case more SwAddrMethods have been associated to parameters of the same SWC, more parameter groups (read Flash memory sections) will be created; in that case the RAM reference table will store the address of the start of each Flash memory section defined:

This method allows to define memory sections to create groups of parameters using one SwAddrMethod for each group.



Configuring a calibrateable parameters on a SWC

### 7 Single Pointered 2 method

This method implements the AR Single Pointered method adding the freedom to use only one SwAddrMethod for all the parameters but keeping the differentiation in terms of memory sections defining one section for each SWC instance. This method is an extension of AR supported by RTA-RTE. The output code for this example would be the following:

In flash, the memory section name define contains the name of the SWC, the ECU Instance and the name of the SwAddrMethod (if defined, otherwise a "NULLSWADDR\_16" extension is added):

	fdefine RTE_START_SEC_OGI_CPT_CANCOMMUNICATION_SMC_ECUA_MYPARAMSECTION finclude "Bte MemMan.h" /*lint !e537 permit multiple inclusion */
C	Volatile CONST (Rte_GG1_CPT_CanCommunication_SWC_EcuA_MyParamSection, RTE_GG1_CPT_CANCOMMUNICATION_SWC_ECUA_MYPARAMSECTION) Rte_GG1_CPT_CanCommunication_SWC_EcuA_MyParamSection = {
5	
	2000,
	5
	#define RTE STOP SEC CGI CPT CANCOMMUNICATION SWC ECUA MYPARAMSECTION
	Palanda What Maanaa Tum Tunaa aa

In RAM the table is replaced by a variable for each parameter group defined storing the starting Flash address of the corresponding memory section:

```
define RT_STAR_SC_FOG_CFT_CANCOMENDICATION_SNC_ECUA_MYRAPAMSECTION
finding* The Membry.h* //lint les31 permit multiple inclusion //
Marker_Kott_Grad_Communication_SNC_EcuA_MyRaranSection = Gre_Cot_CFT_CanCommunication_SNC_EcuA_MyRaranSection;
Marker_Kott_Grad_Communication_SNC_EcuA_MyRaranSection;
Medine KT_STOR_SC_ECUA_COT_CANCOMENTICATION_SNC_ECUA_MyRaranSection;
Medine KT_STOR_SC_ECUA_COT_CANCOMENTICATION_SNC_ECUA_MyRaranSection;
Medine KT_STOR_SC_ECUA_COT_CANCOMENTICATION_SNC_ECUA_MyRaranSection;
Medine KT_STOR_SC_ECU_COT_CANCOMENTICATION_SNC_ECUA_MYRARAMSECTION
And Location
Minicide* The Membry.h* //lint les31 permit multiple inclusion */
```

Below an example of two parameters groups defined:

define ET\_STAT\_SEC\_DOI\_CF\_CANCEMENTATION\_SEC\_ECUA\_MINARMENTIC Honolum The Remember / Hist (EST) Fermit multiple inclusion // VARIEE\_FOIL\_CF\_CanCommunication\_SEC\_ECUA\_MINARMENTION Honolum The Remember / Hist (EST) Fermit multiple inclusion // Honolum The Remember / Hist (EST) Fermit multiple inclusion // Honolum The Remember / Hist (EST) Fermit multiple inclusion // Honolum The Remember // Hist (EST) Fermit multiple inclusion // VARIEE\_FOIL\_CF\_CanCommunication\_SEC\_ECUA\_MINARMENTION Honolum The Remember // Hist (EST) Fermit multiple inclusion // VARIEE\_FOIL\_CF\_CanCommunication\_SEC\_ECUA\_MINARMENTION Honolum The Remember // Hist (EST) Fermit multiple inclusion // VARIEE\_FOIL\_CF\_CanCommunication\_SEC\_ECUA\_MINARMENTION Honolum The Remember // Hist (EST) Fermit multiple inclusion // VARIEE\_FOIL\_CF\_CanCommunication\_SEC\_ECUA\_MINARMENTION Honolum The Remember // Hist (EST) Fermit multiple inclusion // Honolum The Remember // Hist (EST) Fermit multiple inclusion // Honolum The REMEMBERCH // Hist (EST) Fermit multiple inclusion // Honolum The REMEMBERCH // Hist (EST) Fermit multiple inclusion // Honolum The Remember // Hist (EST) Fermit multiple inclusion // Honolum The Remember // Hist (EST) Fermit multiple inclusion // Honolum The Remember // Hist (EST) Fermit multiple inclusion // Honolum The REMEMBERCH // Hist (EST) Fermit multiple inclusion // Honolum The REMEMBERCH // Hist (EST) Fermit multiple inclusion // Honolum The REMEMBERCH // Hist (EST) Fermit multiple inclusion // Honolum The REMEMBERCH // Hist (EST) Fermit multiple inclusion // Honolum The REMEMBERCH // Hist (EST) Fermit multiple inclusion // Honolum The REMEMBERCH // Hist (EST) Fermit multiple inclusion // Honolum The REMEMBERCH // Hist (EST) Fermit multiple inclusion // Honolum The REMEMBERCH // Hist (EST) Fermit multiple inclusion // Honolum The REMEMBERCH // Hist (EST) Fermit multiple inclusion // Honolum The REMEMBERCH // Hist (EST) Fermit multiple inclusion // Honolum The REMEMBERCH // Hist (EST) Fermit multiple inclusion // Honolum The REMEMERCH /



Configuring a calibrateable parameters on a SWC

### 8 Double Pointered method

The additional command to use this method is "DOUBLE\_POINTERED". The output code is a memory section for each SwAddrMethod in Flash storing the calibration values:

```
#define RIE_START_SEC_MyParamSection
#include "Rte_MemMap.h" /*lint !e537 permit multiple inclusion */
Volatile CONST(Rte_CGTi_CPT_CanCommunication_SWC_EcuA_MyParamSection, RIE_MyParamSection) Rte_CGi_CPT_CanCommunication_SWC_EcuA_MyParamSection = {
    2000,
    5
};
#define RIE_STOP_SEC_MyParamSection
#include "Rte_MemMap.h" /*lint !e537 permit multiple inclusion */
```

Besides these, another Flash memory section defines a reference table to the previous sections starting addresses:

```
#define RTE_START_SEC_CONST_UNSPECIFIED
#include "Rte_MemMap.h" /*lint !e537 permit multiple inclusion */
CONST(RteParameterRefTabType, RTE_CONST) RteParameterRefTab = {
    (RteCalprmRefTabEntryType)&Rte_CGi_CPT_CanCommunication_SWC_EcuA_MyParamSection
};
#define RTE_STOP_SEC_CONST_UNSPECIFIED
#include "Rte_MemMap.h" /*lint !e537 permit multiple inclusion */
```

In RAM only one variable is used to store the address of the reference table in Flash.

```
#define RTE_START_SEC_VAR_INIT_UNSPECIFIED
#include "Rte_MemMap.h" /*lint !e537 permit multiple inclusion */
P2CONST(RteParameterRefTabType, AUTOMATIC, RTE_DATA) RteParameterBase = &RteParameterRefTab;
#define RTE_STOP_SEC_VAR_INIT_UNSPECIFIED
#include "Rte_MemMap.h" /*lint !e537 permit multiple inclusion */
```

**NB:** this method's pro is a very low use of RAM (only one address).



DRIVING EMBEDDED EXCELLENCE

Configuring a calibrateable parameters on a SWC

### 9 **Reading parameters from SWC**

Parameters can be accessed by SWC using the RTE defined macros:

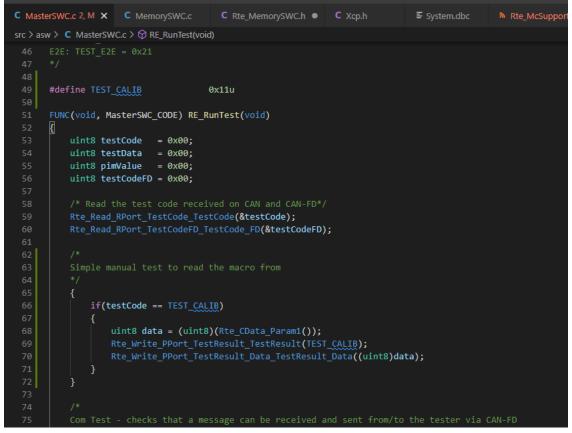
```
/* Inline calprm optimization; Rte_CData_Param1 to direct access */
#define Rte CData Param1() ( (volatile CONST(uint8, CALPRM)) Rte CalprmInitRAM.CGi CPT Master
```

Implementation of these macros depends on the calibration method used; for example for single pointered method a macro would be as following:

#define Rte\_CData\_Param\_1( self ) ( (volatile CONST(uint8, RTE\_MyParamSection)) (((volatile P2CONST(Rte\_CGTi\_CPT\_CanCommunication\_SWC\_EcuA\_MyParamSection, AUTOMATIC, RTE\_MyParamSection))RteParameterRefTab[((VAR(uint16, AUTOMATIC))RTE\_CGI\_CPT\_CANCOMMUNICATION\_SWC\_ECUA\_MYPARAMSECTION)])->Param\_1) )

In this case the macro takes for the reference table in RAM the address of Flash memory and use it to access to parameter value.

For example in the ASW I have modified the MasterSWC function RE\_RunTest to read the value of the parameter and send it over a CAN signal.



And here in Busmaster you can see it returning the value of 5 from Param1 successfully.



Configuring a calibrateable parameters on a SWC

	Tx/Rx	Channel	Msg	ID	Message	DLC	Data By	te(s)		
13:6207	Rx	1	s	A00x0	0xA	8	11 05 F	F FF FF FE	FFFFF	
00:1807	Tx	1	s	0x014	0x14	8	11 00 0	0 00 00 00	00 00	
Tx Frame Li		sages - CAN	Frame Id	Channel	Data Length	Messag	e Type RT	R Repetition	(ms) Key	
0x14			0x14	1	8	St	rd 📃	10	) 🗖 a	
	Message]		0x14	1	8	St	td 🗖	10	) 🗖 a	
	Message]		0x14	1	8		end Message	Delete	Delete A	
[Add	Message]	0 01	0x14		8	S				



Configuring a calibrateable parameters on a SWC

### **10** Contact, Support and Problem Reporting

For details of your local sales office as well as your local technical support team and product hotlines, take a look at the ETAS website:

ETAS subsidiarieswww.etas.com/en/contact.phpETAS technical supportwww.etas.com/en/hotlines.php