ETAS

RTA-FBL FCA PORT

USER MANUAL

Status: RELEASED

Copyright

Document : RTA_FBL_FCA_PORT_UserManual.docx

Contents

Introduction

This user manual introduces the RTA-FBL port for FCA. It provides an overview of the RTA-FBL architecture and software design. It also provides detailed information of the FCA port for users developing ECUs that will be reprogrammed with RTA-FBL. This includes information about how to configure RTA-FBL, as well as how to integrate the Application Software on the ECU.

1.1 Revision History

Version	Author	Date	Change (Why, What)
0.1	Francesco Ficili	29/11/2018	First version.
1.0	Andrew Borg	03/02/2019	First release.
1.1	Daniele Cloralio	29/05/2020	FCA Port version
1.2	Daniele Cloralio	15/07/2020	Minor changes
1.3	Daniele Cloralio	22/10/2020	Typo and minor changes
1.4	Francesco Sfragara	05/11/2020	Minor changes in section 4.3 and 3

1.2 Definition and Abbreviations

Term/Abbreviation	Definition
ADC	Analogue to Digital Convertor
AR	AUTOSAR
Application Software (ASW)	This is the software that executes the control logic of the ECU
AUTOSAR	AUTomotive Open System Architecture
BLSM	Bootloader State Manager
BSW	Basic Software
CAN	Controller Area Network
CAN FD	CAN Flexible Datarate
CRC	Cyclic Redundancy Code – a CRC module is provided in RTA-BSW
Dcm	Diagnostic Communication Manager
DID	Data IDentifier
DLL	Dynamic Link Library
ECU	Electronic Control Unit
FBL	Flash Bootloader
Fee	Flash EEPROM Emulation
FW	Firmware
HW	Hardware
ISR	Interrupt Service Routine
MCAL	Micro-Controller Abstraction Layer
NvM	Non-Volatile Memory
OS	Operative System

Term/Abbreviation	Definition
RTA-x	The ETAS suite of embedded SW products
S&K	Seed And Key
SW	Software
UDS	Unified Diagnostic Services

1.3 References

Ref.	Document Name	Ver.
[1]	CS.00101_ECU FLASH Reprogramming Requirements	Rev. C
[2]	CS.00102_Standardized Diag Data	Rev. D
[3]	CS.00099_Diag Reqs UDS	Rev. C
[4]	CS.00100_Diagnostic Services	Rev. C
[5]	ISO 14229-1:2013 Road vehicles -- Unified diagnostic services (UDS)	Ed.: 2

1.4 About this Document

This document provides a detailed description of ETAS' RTA-FBL Port for FCA OEM. It provides a reference for ECU developers that will allow reprogramming of their ECU using RTA-FBL.

1.5 Chapter Description

Chapter	Description
Chapter 1	This is the document introductory chapter.
Chapter 2	This chapter introduces ECU reprogramming in general and associated tooling, including RTA-FBL.
Chapter 3	This chapter explains how RTA-FBL must be installed in order to allow you to create a complete RTA-FBL bootloader instance.
Chapter 4	This chapter introduces the RTA-FBL Port for FCA. It includes important integration steps required for integrating RTA-FBL with your Application Software.
Chapter 5	This chapter explains how to flash an ECU with an RTA-FBL bootloader using INCA.
Chapter 6	This chapter contains important privacy information.
Chapter 7	This chapter contains ETAS references for customer support.

2 Introduction to ETAS RTA-FBL

This section introduces basic FBL concepts independently of a particular OEM port or hardware target. It also introduces ETAS' FBL product, RTA-FBL, and provides information that is common to all ports and targets. Specific information about your port and the targets supported in this port are detailed in Section 4

2.1 What is a Flash Bootloader?

A Flash Bootloader (FBL) is embedded SW that allows the reprogramming of an ECU with new Application SW using a standard communication channel. The FBL works in combination with an external tool that runs as a desktop application (often called a Flash Tool or Tester Tool). This tool communicates with the FBL executing on the ECU to transfer the new Application SW. The FBL updates the ECU's non-volatile memory with this new Application SW.

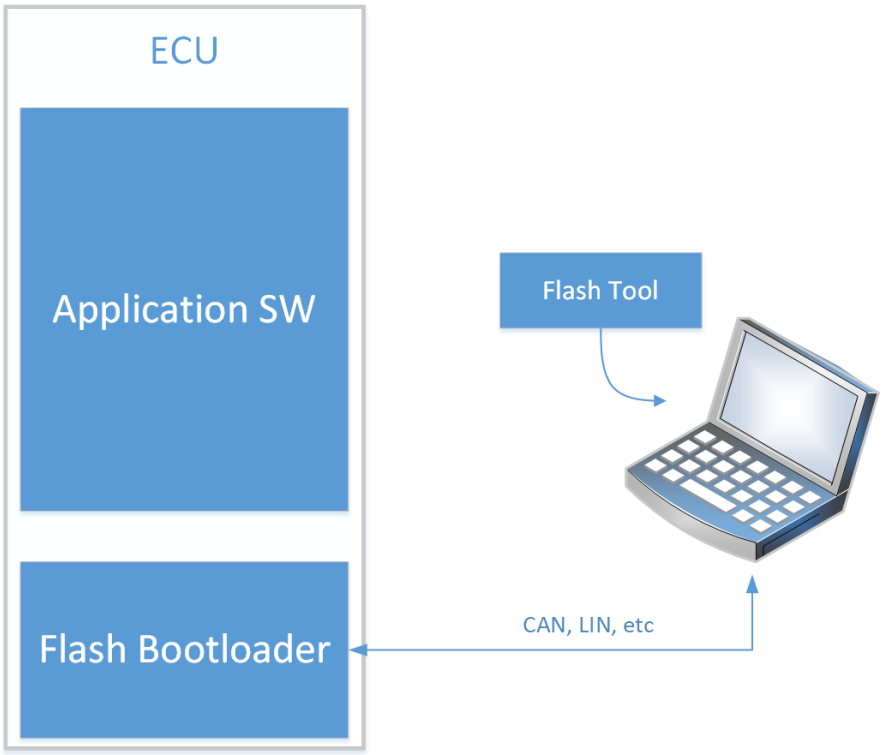


Figure 1: High level flashing process

The FBL is a standalone program. It has a separate run-time with respect to the Application SW, and so the FBL and the Application SW never run concurrently. After startup, the FBL always runs first as it needs to decide whether it is to wait for new Application SW to be sent from a tester, or if it is to start the Application SW already present in the ECU. This decision depends on two items of state in the ECU: whether a reprogramming request flag has been set by the Application SW before the last reset, and whether the Application SW currently programmed in the ECU is valid.

A classic boot loading sequence showing this decision is depicted in Figure 2. Note that the Application SW is only started if the Application SW is valid and the reprogramming request flag is not set. In any other case, the FBL enters the Bootloader state and communicates with the tester to reprogram the ECU.

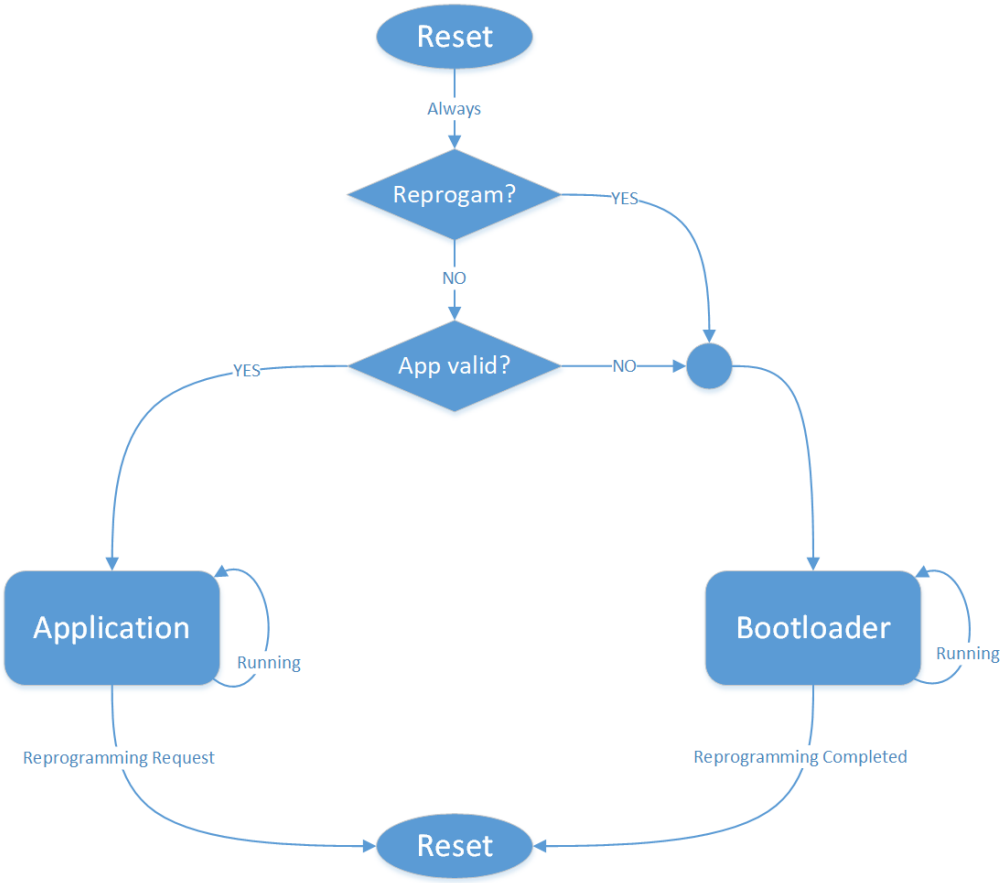


Figure 2: Boot loading flowchart

2.2 What is RTA-FBL?

RTA-FBL is ETAS' bootloader product offering. It allows integrators to create Flash Bootloader software according to a specific OEM specification. RTA-FBL generates source code (flash boot loader modules and basic software) from user configuration. This significantly reduces the user effort required to get the flash bootloader up and running and integrated with the application software.

RTA-FBL leverages the following layers defined by the AUTOSAR standard architecture:

- MCAL: provided by silicon vendor
- BSW: provided by ETAS (RTA-BSW)

Although RTA-FBL ports currently support CAN and CAN-FD, basing the underlying SW architecture on AUTOSAR allows support of other communication protocols such as Ethernet, FlexRay, LIN.

RTA-FBL satisfies requirements from different OEMs for different HW architectures by creating ports that integrate with the core RTA-FBL product. The clear separation between core (which is OEM independent and target independent) and port (which is OEM-dependent with support for one or more targets) makes it possible to support a wide range of OEM FBL requirements and allows quick porting to new targets.

RTA-FBL generates source code and BSW files through the following components:

- rtafblgen: an executable for FBL generation
- RTA-FBL GUI: a user interface for configuring the parameters used by rtafblgen for generation. The configuration options depend on the OEM port and selected target.

2.3 The Flash Tool (Tester)

The Flash Tool, or Tester, is a desktop application that handles the PC-side of the flashing process. In general, the tester is used when the bootloader is in production and access to the ECU is limited to non-debug communication protocols such as CAN, Ethernet and FlexRay.

2.4 The OEM-defined Programming Sequence

The tester communicates with the ECU by sending messages over a communication bus according to a defined protocol. The ETAS FBL supports the UDS on the CAN protocol. This means that requests are made to the ECU over a CAN bus, and the messages sent and received comply with the UDS standard ISO 14229-1[2]. The allowed message sequence sent to the ECU, as well as the expected response from the ECU differs across OEMs. Therefore, the ETAS FBL supports different OEM standards for ECU reprogramming. These are called "OEM ports" or just "ports". This guide specifically addresses the RTA-FBL port that implements the reprogramming standard described in [1]. Each port supports one or more hardware "targets". For example, the RTA-FBL port that implements [1] supports all the targets described in Section 4.6.

2.5 Target Dependencies and the Flash Driver

An FBL will naturally contain several dependencies on the underlying microcontroller target. In addition to the typical drivers such as communication, port and timer drivers is the driver used by the bootloader to write the FLASH memory of the ECU. This is target dependent code (usually provided by the silicon vendor), because each different target could have different flash memory properties (i.e. different technology, layout, endurance, etc.). The flash driver typically forms part of the MCAL.

2.6 Interaction with the Application using NvM

A Bootloader and the Application Software may need to share data. For example, a Tester may read or write data such as the ECU serial number both when the ECU is running in boot-loader mode and when running its Application Software (e.g. by using UDS ReadDataByIndentifier and WriteDataByIdentifier commands). Typically, this will mean that both the Bootloader and the Application Software will need to be able to read and write the same non-volatile memory. Where non-volatile memory is implemented by EEPROM emulation in flash such sharing may introduce technical challenges because the Bootloader and Application Software must use the same algorithms and data-structures when emulating EEPROM. (For example, if the application uses an Autosar Fee module for EEPROM emulation then the Bootloader may need to use the same Fee module). The requirements for compatibility between the FBL and Application Software for your port are detailed in Section 4.

2.7 One and Two-Stage Bootloaders

There are two broad models for bootloaders and the model type for the bootloader described in [1] is described in more detail in Section 4.

- Single-stage: In this model, the complete Bootloader is stored on the ECU (in flash), including the code used to write a new application to flash.
- Two-stage: In this model, a Primary Bootloader is stored in the ECU. This Primary Bootloader is able to start the application running or download a Secondary Bootloader into RAM. The Primary Bootloader is not able to write to the flash used to store the application. Programming flash with a new application is done by the Secondary Bootloader. There are three advantages to the two-stage approach:
    1. The Primary Bootloader can in principal be smaller because it does not need to include the code to write to flash (although space savings will be limited in

practice if the Primary Bootloader also needs to include a flash driver to write to non-volatile memory implemented with flash).

2. Since the Primary Bootloader does not contain the code to write to flash, the application is less likely to corrupt itself or the bootloader because faulty code in the application cannot jump to the flash reprogramming driver.
3. The Secondary Bootloader can be used to work around bugs in the bootloader installed on the ECU when it was manufactured.

Rather than an independent Secondary Bootloader, some OEMs use a single-stage Bootloader that only excludes the flash driver used to write to the flash that stores the application. Instead, the driver used to write to flash is downloaded and stored in RAM during the programming sequence. This is sometimes referred to as a software "interlock".

2.8 FBL generation with the RTA-FBL ISOLAR-AB plugin

An instance of ETAS's FBL is generated based on the chosen OEM specification that defines the reprogramming sequence, the chosen hardware target, and the specific configurations that are allowed within the scope of the OEM specification. The tool for generating this FBL instance is an ISOLAR-AB plugin, which is included with your purchased core license. An FBL generated using this plugin is described as "an instance of RTA-FBL". The plugin creates bootloader code as well as a full RTA-BSW project with configuration that is needed to support the bootloader functionality. In the same generation process, the plugin therefore optionally also invokes RTA-BSW to generate an instance of the BSW. Alternatively, the user can open the RTA-BSW project created by the RTA-FBL plugin to inspect the generated configuration. FBL generation also results in some ports in the generation of an MCAL project that can be adapted. Further details relevant to your port are provided in Section 4.

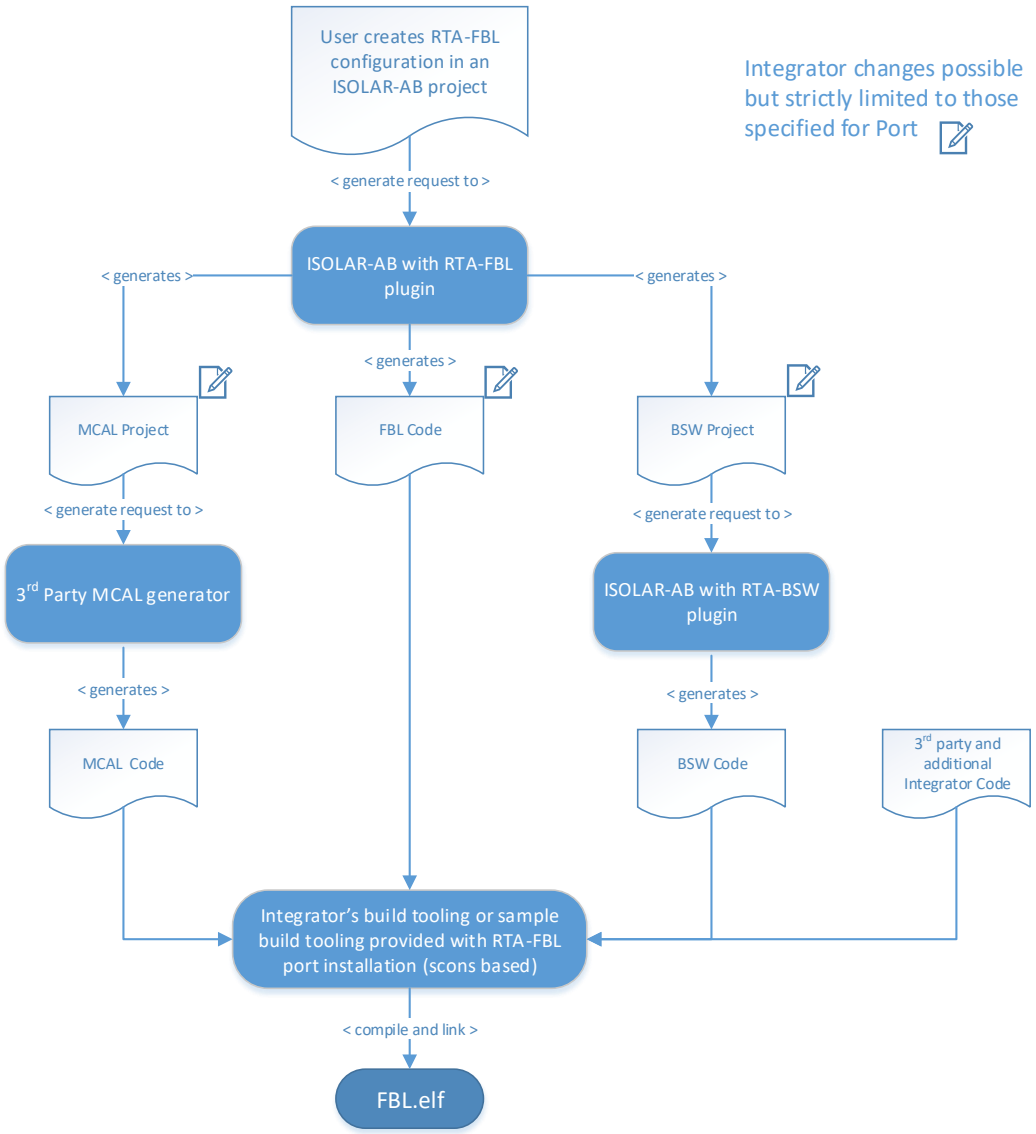


Figure 3: The process of generating an RTA-FBL instance

The tool process for generating an RTA-FBL instance is shown in Figure 3. ETAS-provided tooling allows the integrator to create the bootloader-specific application code (through the RTA-FBL plugin for ISOLAR-AB), and the BSW code (through the RTA-BSW plugin for ISOLAR-AB). The MCAL code must be created using a 3rd party tool, typically provided by the silicon vendor.

Note that the RTA-FBL ISOLAR-AB plugin generates source code that includes some sample code that may require modification by the integrator. The integrator also has the option to add further integration code. Finally, all source code needs to be integrated and built using either the sample build scripts provided with RTA-FBL (and based on scons) or the integrator's own build toolchain.

IMPORTANT: RTA-FBL tests are carried out by ETAS for various FBL configurations that create for each configuration different bootloader code, an MCAL project and a BSW project. Since the integrator can make adaptations to specified sample code, the generated MCAL project and the generated BSW project, this may result in a final software stack that is not tested. For this reason, it is ultimately the integrator's responsibility to test that the complete bootloader works with any changes made to any code or projects generated by RTA-FBL. Please read the important integrator guidelines provided in Section 4 for information relevant to your port.

2.9 General architecture of RTA-FBL

An instance of RTA-FBL consists of five types of module as shown within the complete RTA-FBL architecture in Figure 4. These are:

1. Core bootloader modules (in blue): these are generated from the RTA-FBL ISOLAR-AB plugin and must not be modified.
2. BSW modules (in orange): these are standard AUTOSAR BSW modules generated by RTA-BSW and must not be modified.
3. Port-specific bootloader modules (in yellow): these are generated by the RTA-FBL ISOLAR-AB plugin and must not be modified. They implement the bootloader features that are specific to an OEM.
4. Port-specific bootloader modules (in green) generated from the RTA-FBL ISOLAR-AB plugin that can be modified by the integrator as discussed in Section 4.7. For example, the scheduler with callouts to main functions is provided in all ports as a sample OS, and can be modified. Most ports will also include integration code that can be used as provided in samples or completed by the integrator.
5. 3rd-party modules, and in particular the MCAL.

As noted in Section 2.8, you will need to install a number of tools in order to generate a complete instance of RTA-FBL with all required modules as shown in Figure 4. A number of integration steps will also be required to build your software. Details for your specific OEM port and target are also given in Section 4, including the folder structure of a generated RTA-FBL instance that contains the code for the modules in Figure 4.

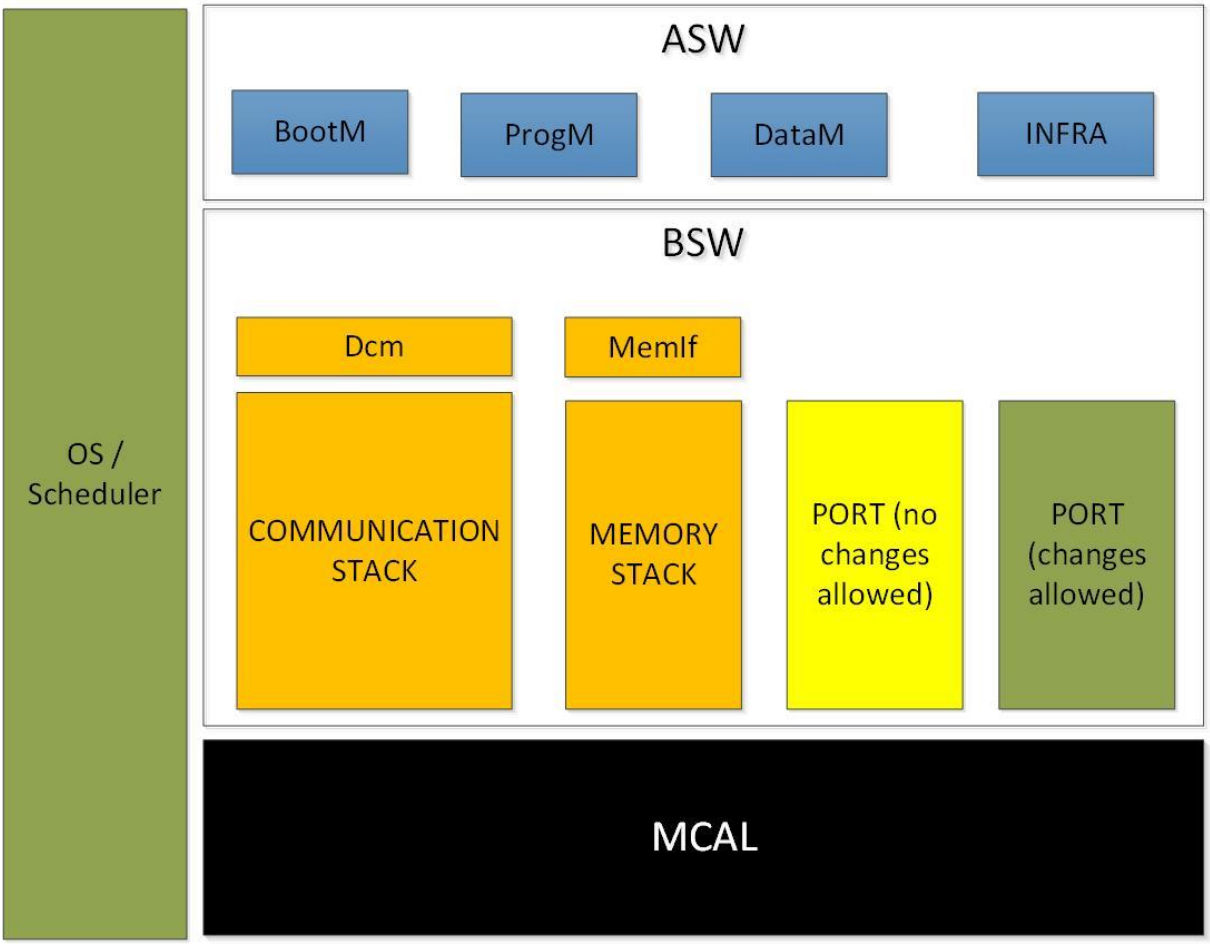


Figure 4: General architecture of an RTA-FBL instance

2.10 Setting up your environment to generate an RTA-FBL instance

In order to generate an instance of RTA-FBL, you will need to install:

1. ISOLAR-AB; the supported version depends on the BSW used by the Port, and it is 8.0.1.
2. The RTA-FBL installation package that contains the RTA-FBL plugin for ISOLAR-AB that drives the FBL generation process, and includes the core RTA-FBL modules and the correct RTA-BSW version, that for this port is 5.1.

Section 3 explains how to install package (3). Please see the installation guides for ISOLAR-AB and RTA-BSW for information on how to install these packages.

Once you have the above packages, you will be able to generate an instance of RTA-FBL. In order to build the instance, you will also need to have installed the 3rd party MCAL as well as the relevant compiler toolchain. See Section 4.6 for further information specific to your target.

3 Installing RTA-FBL

This section describes the installer for RTA-FBL. As noted in Section 2.10, you need to install this package in addition to ISOLAR-AB. This installer is described further in this section.

In order to install RTA-FBL, follow the instructions below. At the end of this installation, the PC needs to restart.

Step 1: Execute the file setup.exe from the root folder of the installation CD. When the destination location window is displayed, select your preferred folder and click "Next".

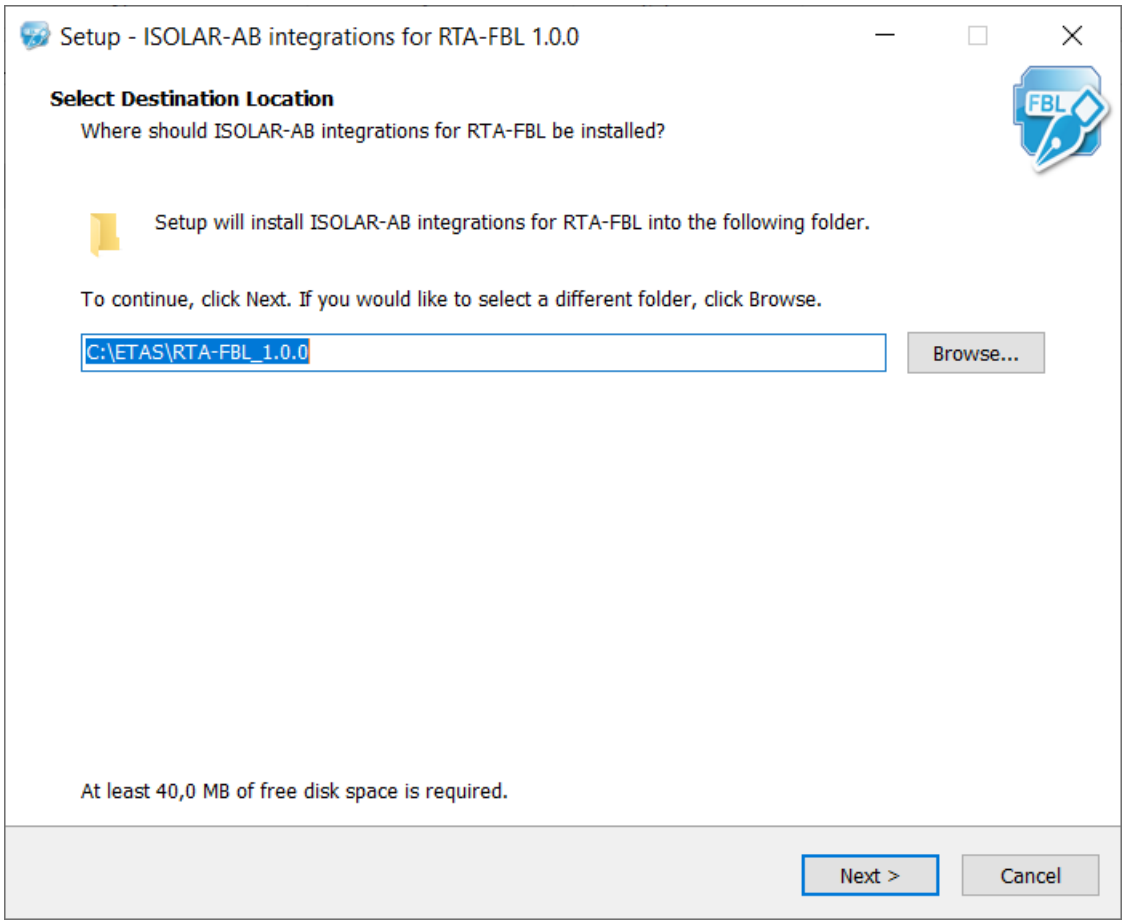


Figure 5: Welcome window

Step 2: Select the ISOLAR version you want to install the plugin into and click "Next".

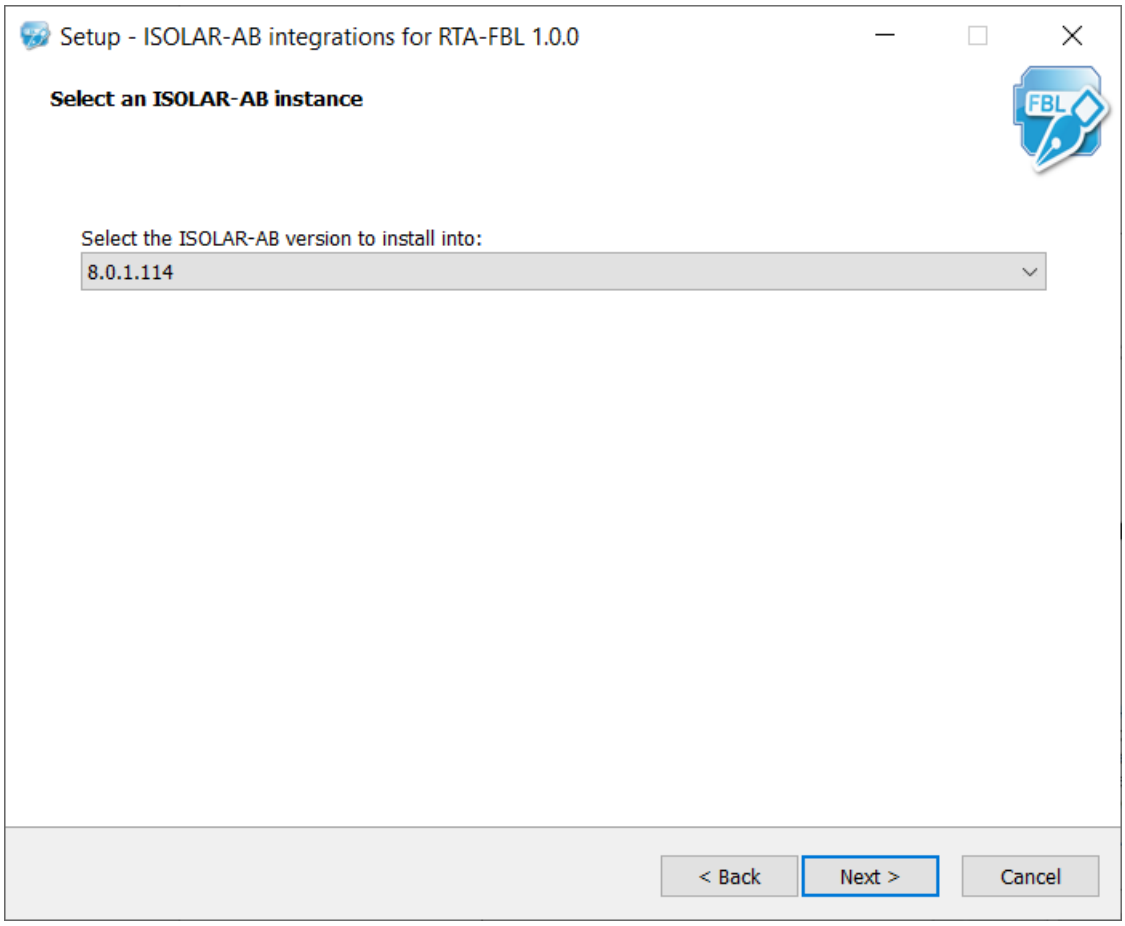


Figure 6: ISOLAR integration

Step 3: Wait for the software required for RTA-FBL to be installed.

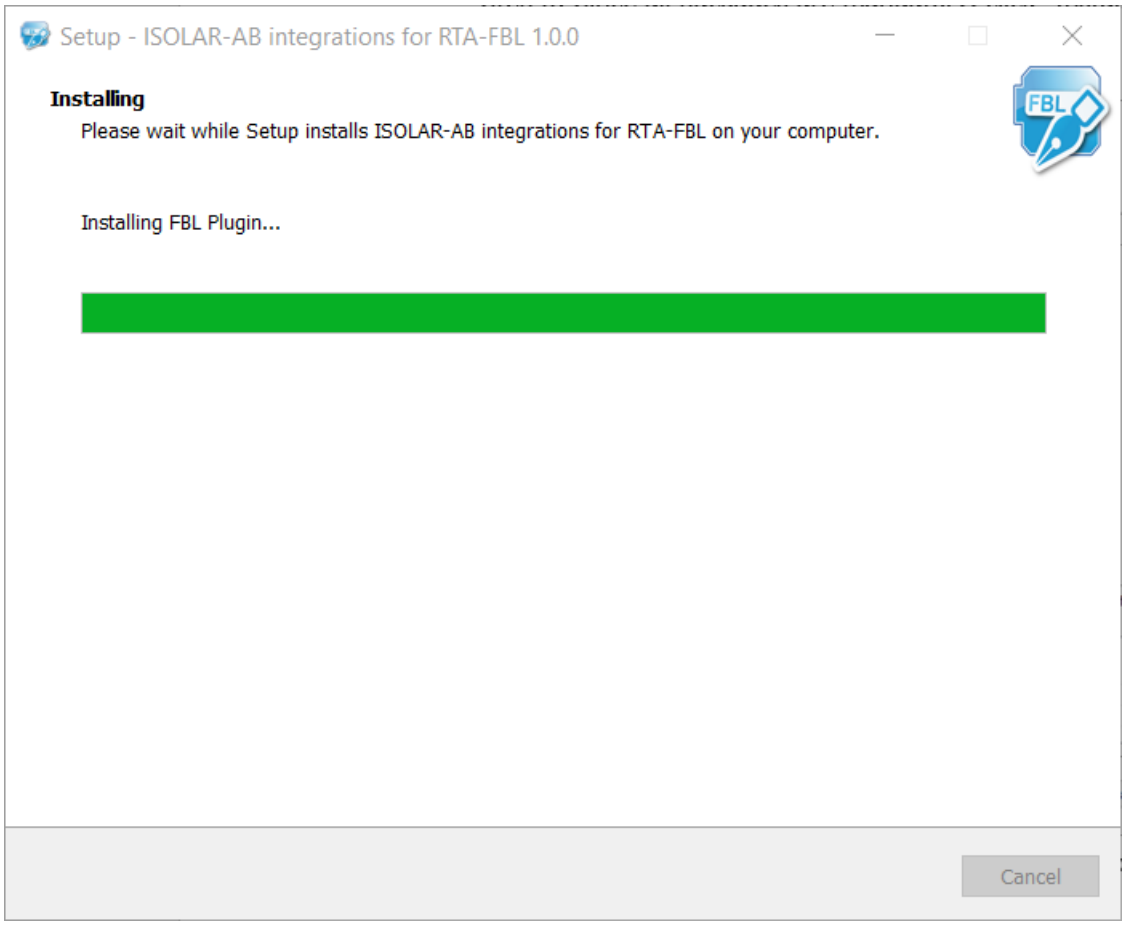


Figure 7: Installation Ongoing

Step 5: Once the installation completes, click on "Finish" to close the installer.

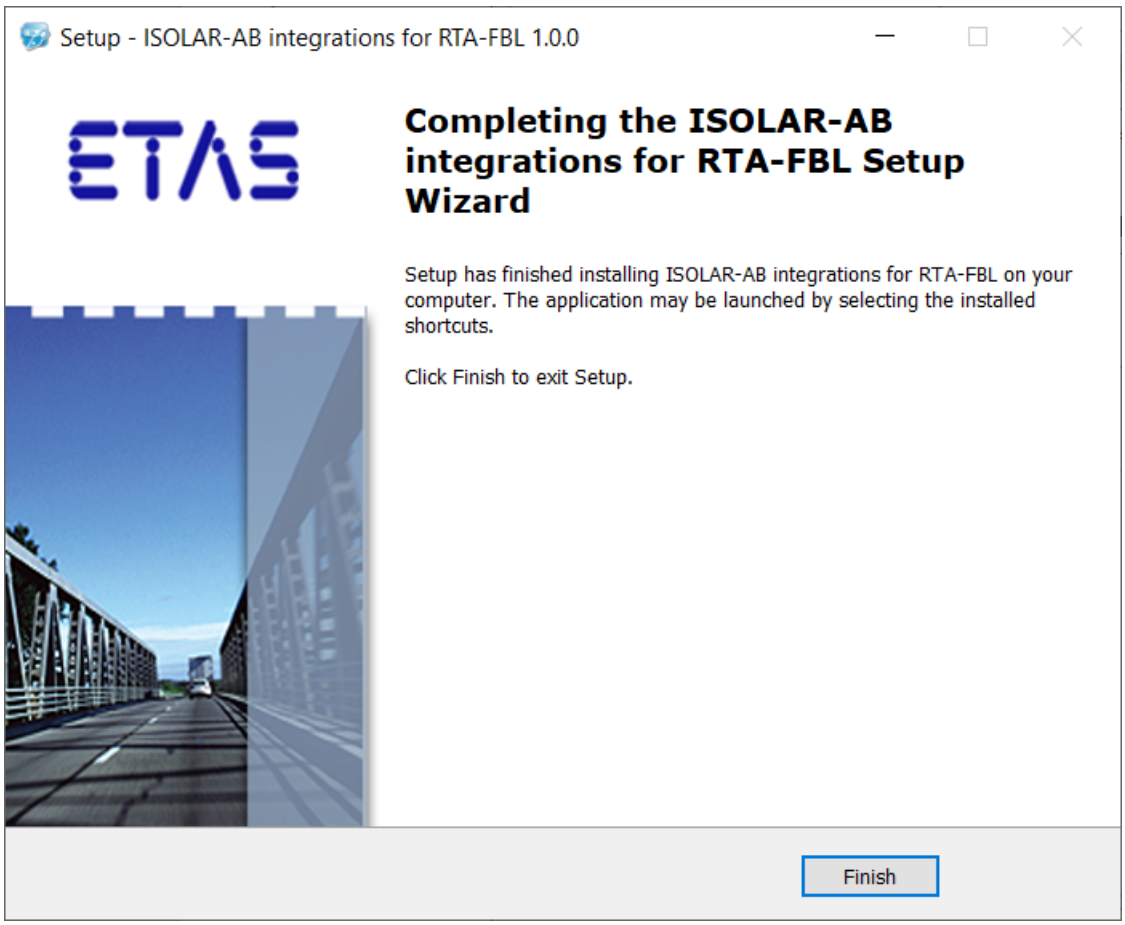


Figure 8: Installation finished

4 The FCA Port

This chapter describes the FCA Port of RTA-FBL. It provides specific information relevant to this port that expands on the general RTA-FBL features described in Chapter 2. This chapter assumes that the reader is familiar with the FCA Bootloader Specification in [1] and all relevant referenced specifications therein. Reference is therefore made to [1] only in describing the configuration and implementation-specific features of RTA-FBL.

4.1 FCA RTA-FBL Architecture

Figure 9 provides a high-level view of RTA-FBL architecture for FCA. The communication, memory and diagnostic stacks are based on RTA-BSW and support the AUTOSAR architecture and methodology for source code configuration and generation. The rest of the components, except for the MCAL, are provided by ETAS and Escrypt. The modules that comprise the RTA-FBL instance for a FCA port are:

1. Core bootloader modules (in blue): these are generated from the RTA-FBL ISOLAR-AB plugin and must not be modified by the integrator.
2. Standard AUTOSAR BSW modules (in orange): these are generated by RTA-BSW and should not be modified by the integrator.
3. The FCA-specific port module (in yellow): this is generated by the RTA-FBL ISOLAR-AB plugin when the FCA port is selected. This module implements the bootloader features that are specific to the FCA specification [1].
4. The FCA-specific sample modules (in green): these are generated by the RTA-FBL ISOLAR-AB plugin when the FCA port is selected and may be modified by the integrator:
    - The OS is a basic cyclic scheduler that can be replaced by any other scheduler (e.g. a fully-configured RTA-OS) as long as the calls to the relevant main functions are made at the correct periods as in the provided samples. See Section 4.7.9 for further details on how to adapt this module.
    - The BLSM contains code for initializing the Bootloader. Changes can be made here by the integrator if other modules are to be integrated (e.g. other BSW modules) but changes should not be made to the functions that interact with the core FBL modules. See Section 4.7.10 for further details on how to adapt this module.
5. Third-party software modules (in red): these are security modules provided by Escrypt that should not be modified by the integrator. These modules are not generated with RTA-FBL and shall be added manually. If Escrypt solution (CycurHSM and FCA Wrapper) is not used, the integrator shall add a compatible security stack with the right integration code.
6. The MCAL modules (in black); the modules shown are those required by the FCA port of RTA-FBL. The integrator may add additional modules required for a specific ECU. For example, the ADC module would likely be required if the integrator wishes to check the battery voltage or other system operating conditions required for the specific ECU.

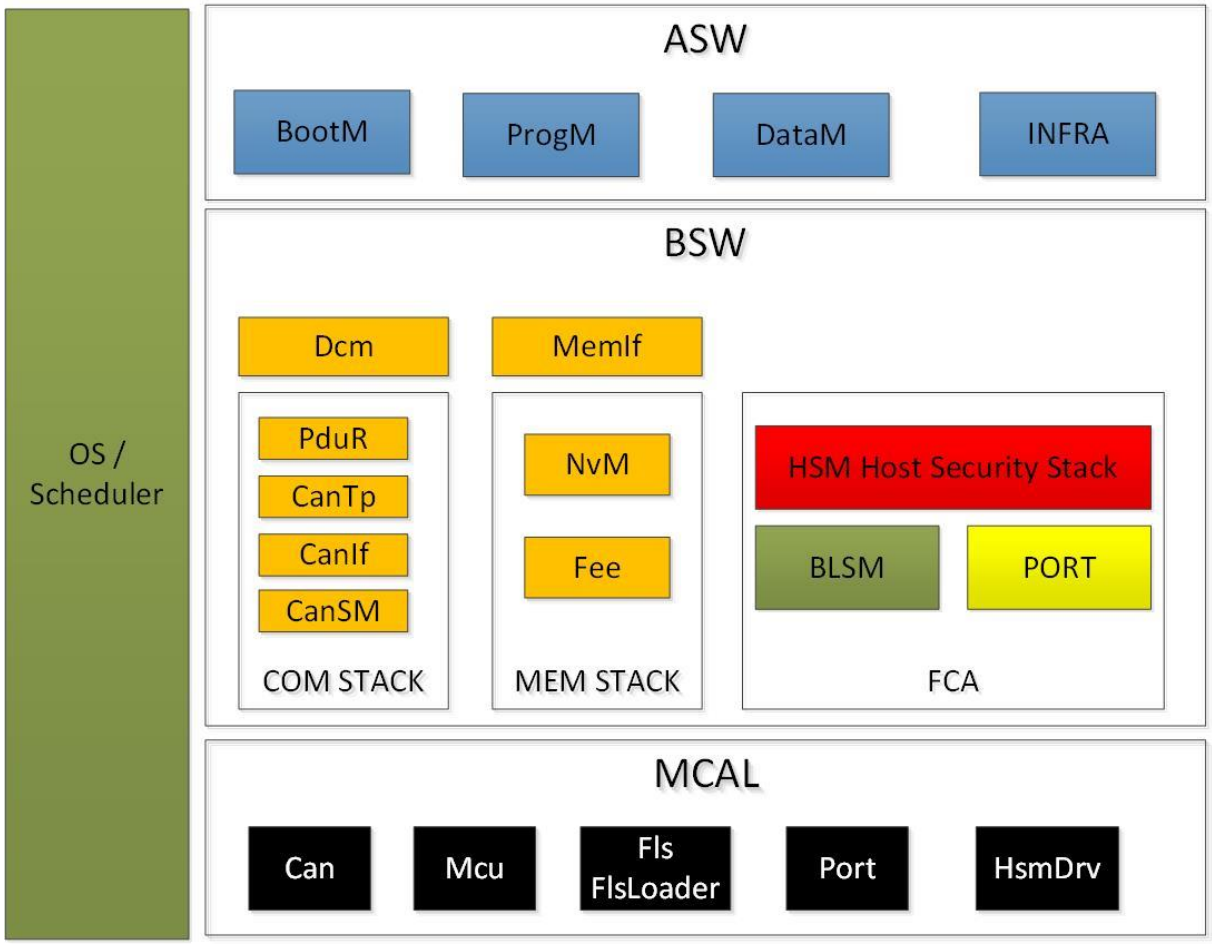


Figure 9: FCA architecture of an RTA-FBL instance

4.2 FCA Download Sequence

The download sequence is according to [1] and depicted below:

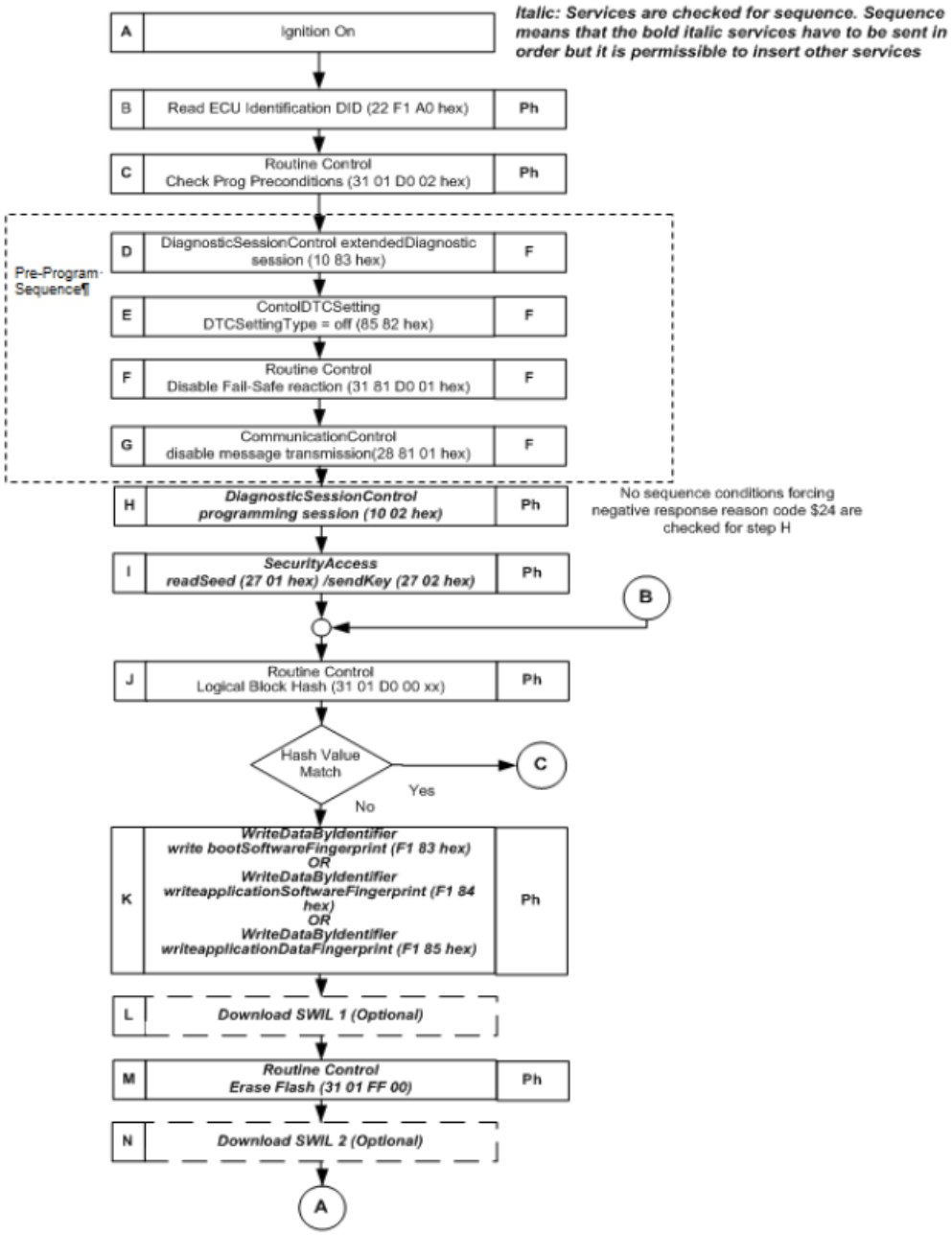


Figure 10 - Flash Download Sequence Part 1

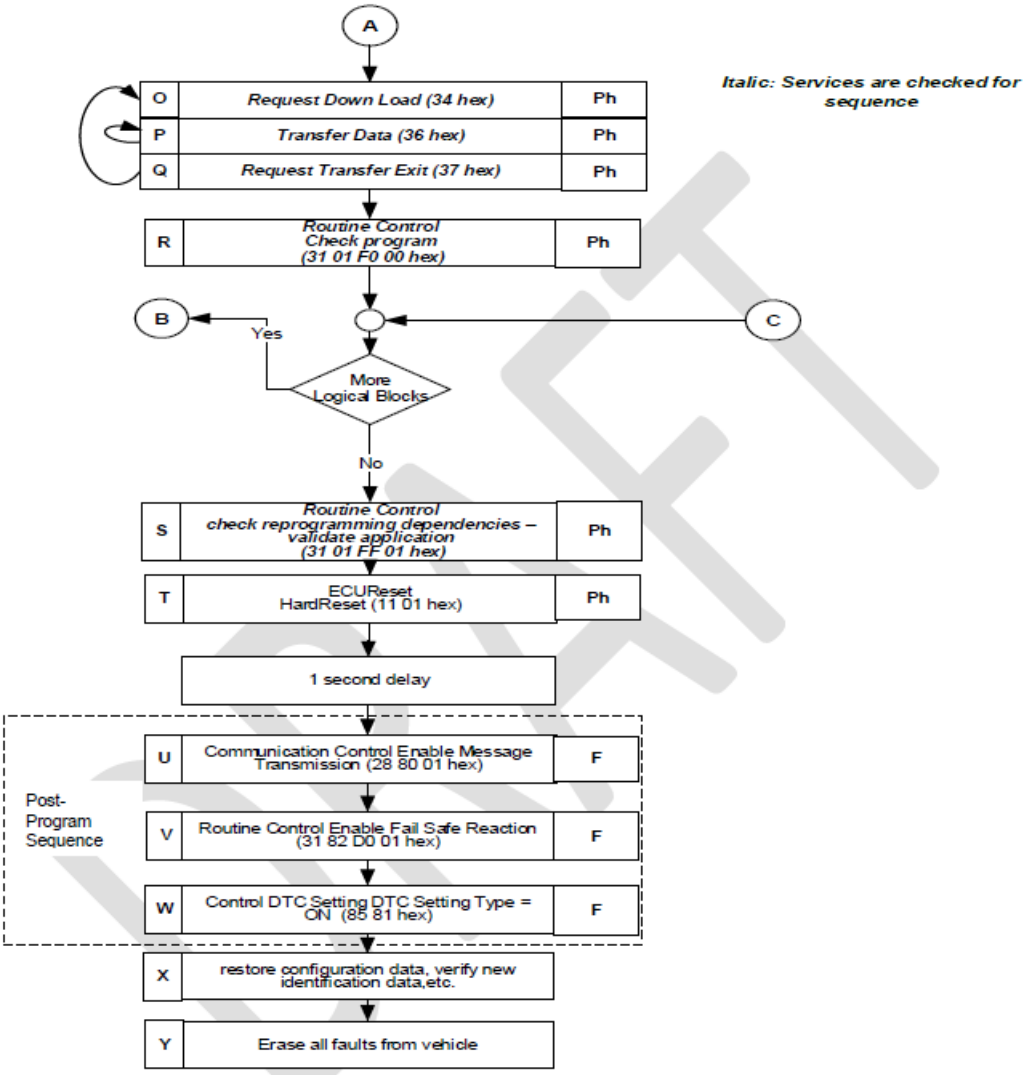


Figure 11 - Flash Download Sequence Part 2

4.3 Creating and building an RTA-FBL instance

This section explains how to create an ISOLAR-AB project in order to configure and generate an instance of RTA-FBL compliant with the FCA bootloader specification. The tooling described in this section has been tested with Windows 10.

4.3.1 Project creation

A new FBL project is created in ISOLAR-AB. As shown in Figure 12, create a new RTA-CAR project by clicking the "New Project" dropdown button and selecting "RTA-CAR Project".

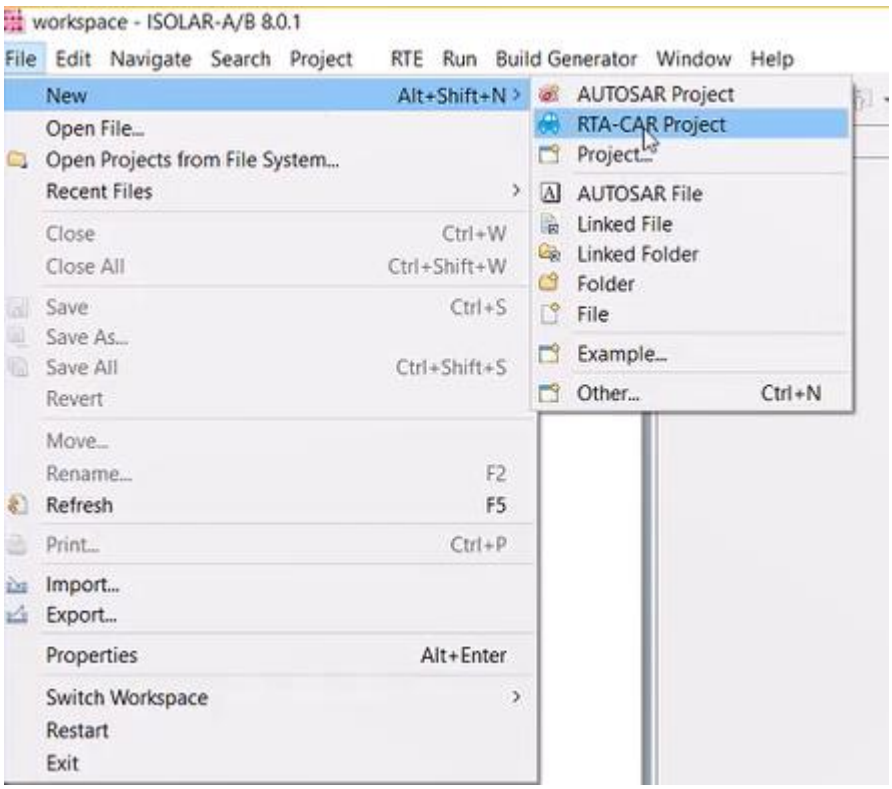


Figure 12: RTA-CAR project creation

Il RTA-CAR Project is not present, select "Project" and search for "RTA-CAR Project" in the new window, as shown in Figure 13.

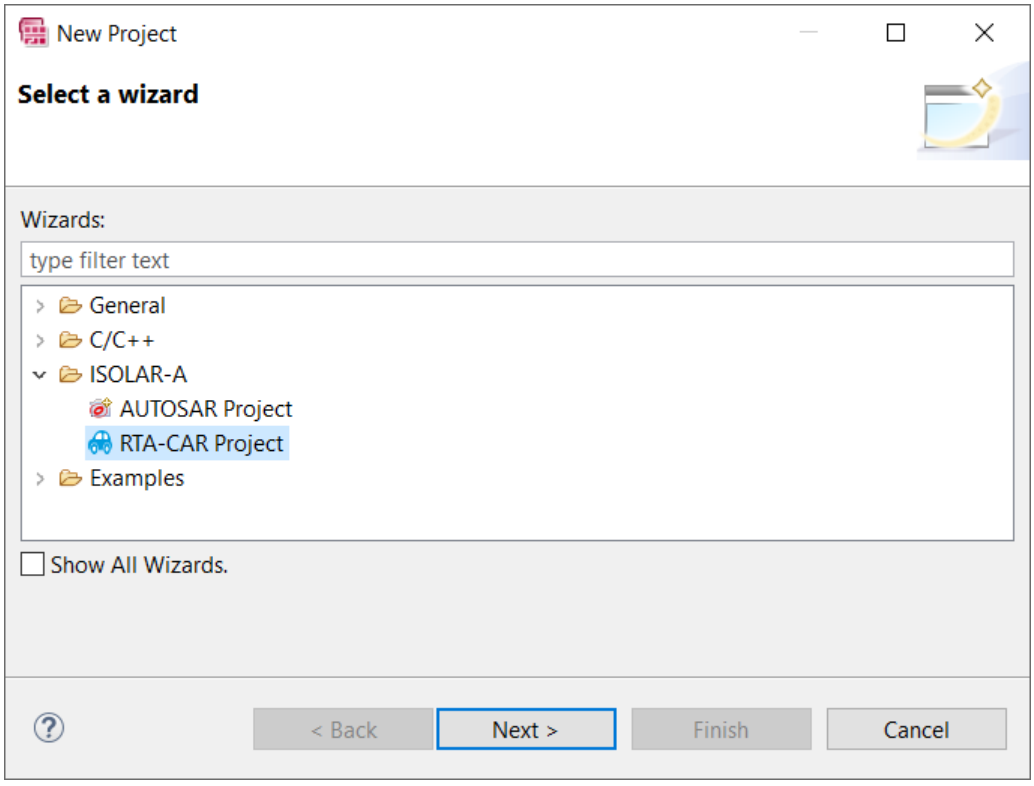


Figure 13: RTA-CAR project

In the New RTA-CAR Project window, choose a name for your project and select the 1.0.0.FCA plugin under RTA Tools as shown in Figure 14.

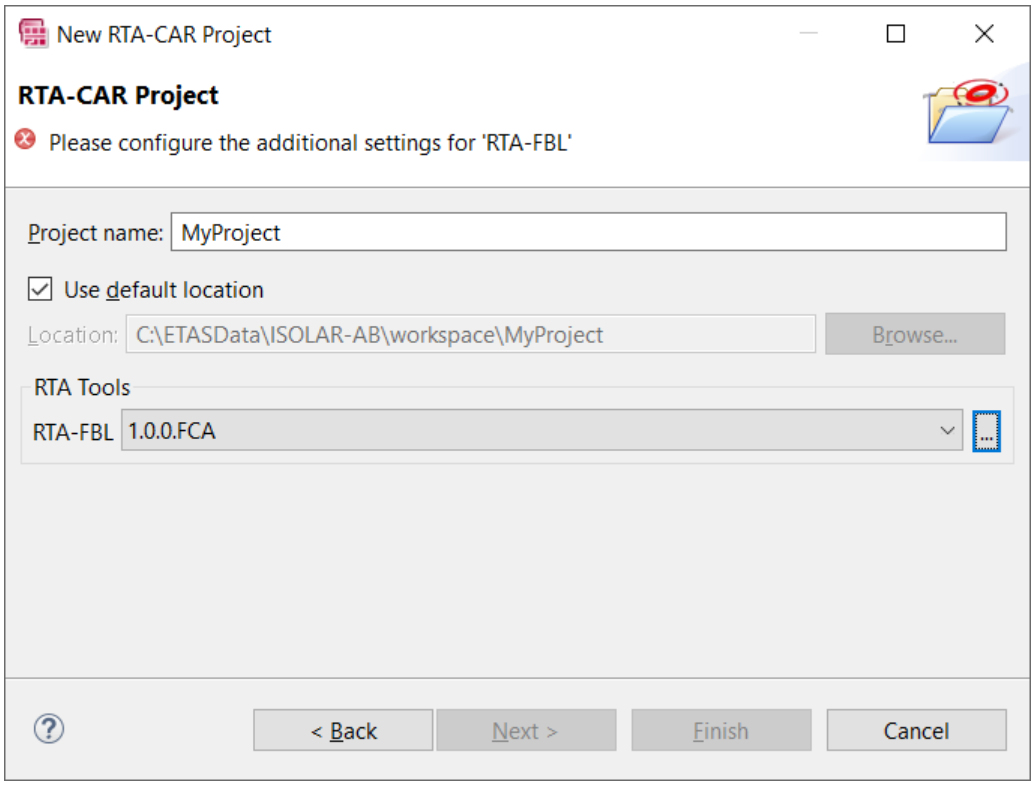


Figure 14: New RTA-FBL Project

Next, click on the three dots icon to open the window "Additional Project Settings" and select the target from the dropdown list as shown in Figure 15.

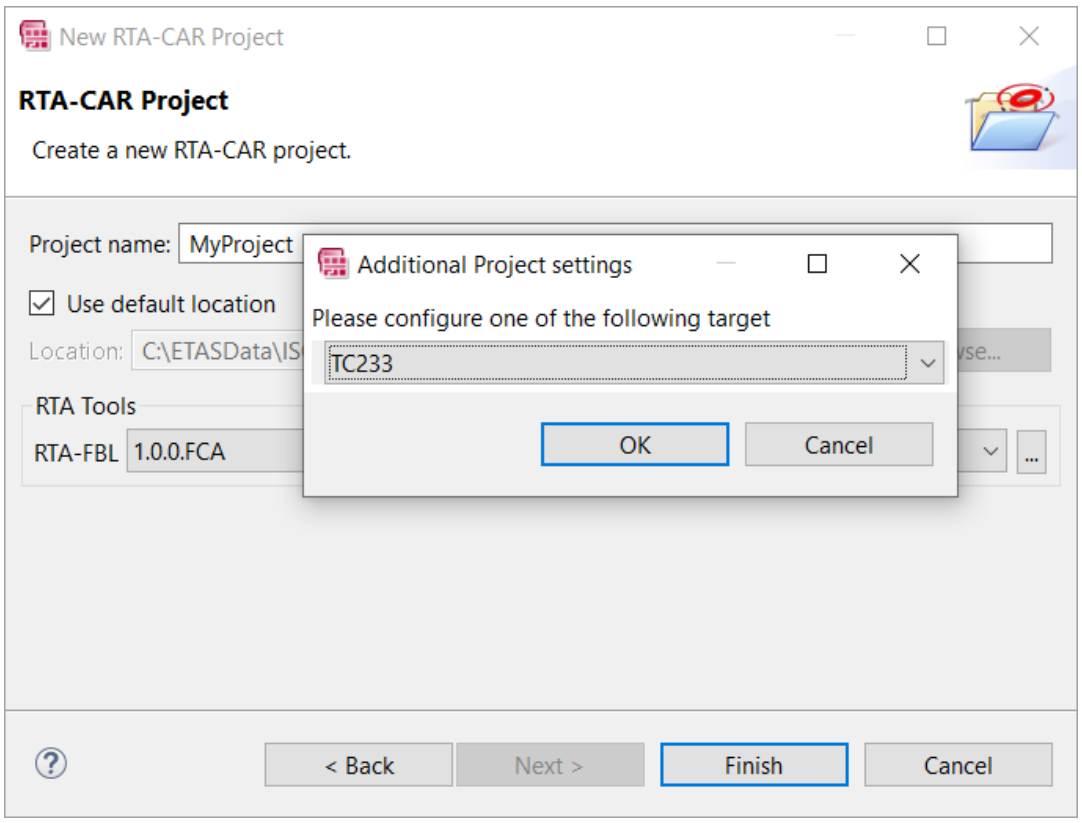


Figure 15: Select Target

Once complete, clicking the Finish button will result in the creation of the FBL project.

Figure 16 shows the result of a successful project creation in the console window.

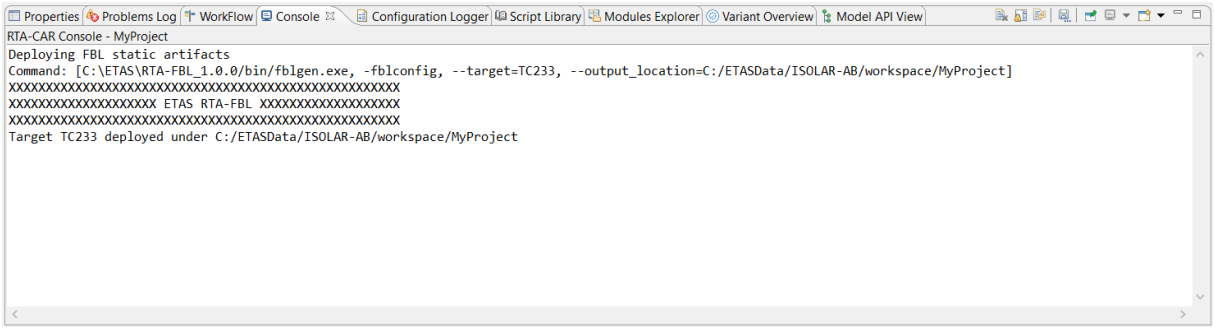


Figure 16: Console window upon successful project creation

4.3.2 Configuration and Generation of FBL and BSW

Next, complete the base configuration parameters. In the AR Explorer view, double click on FblConfigSet under Bsw > Bsw Module Description > FBL, as shown in Figure 17.

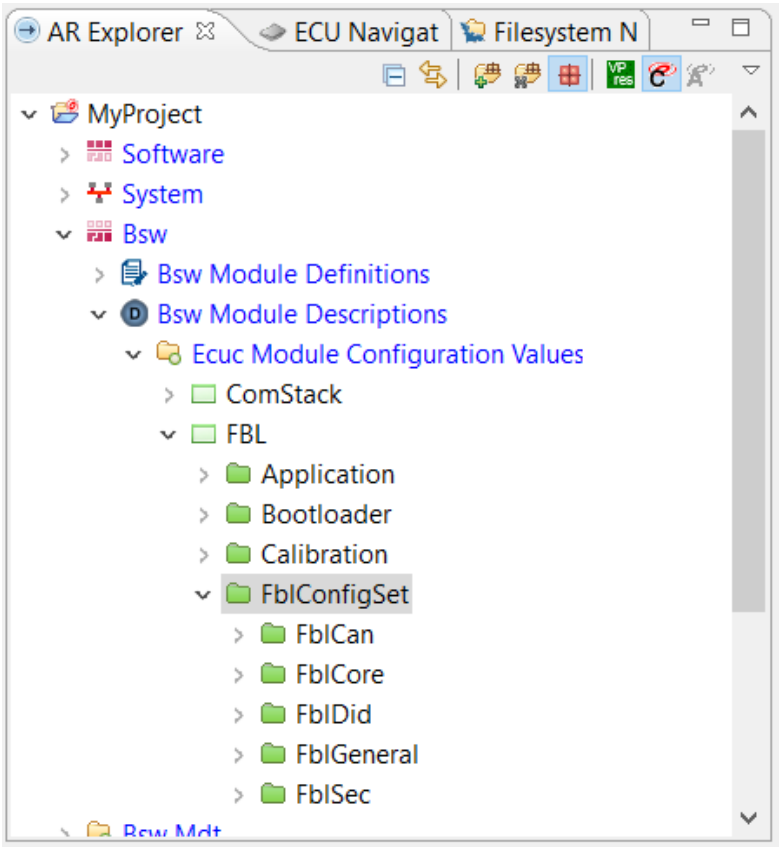


Figure 17: FblConfigSet

The user can now edit the base configuration parameters in the RTA-FBL Editor window. Figure 18 shows as an example the configuration parameters for CAN communication. An explanation of each parameter is provided in Section 4.3.2.

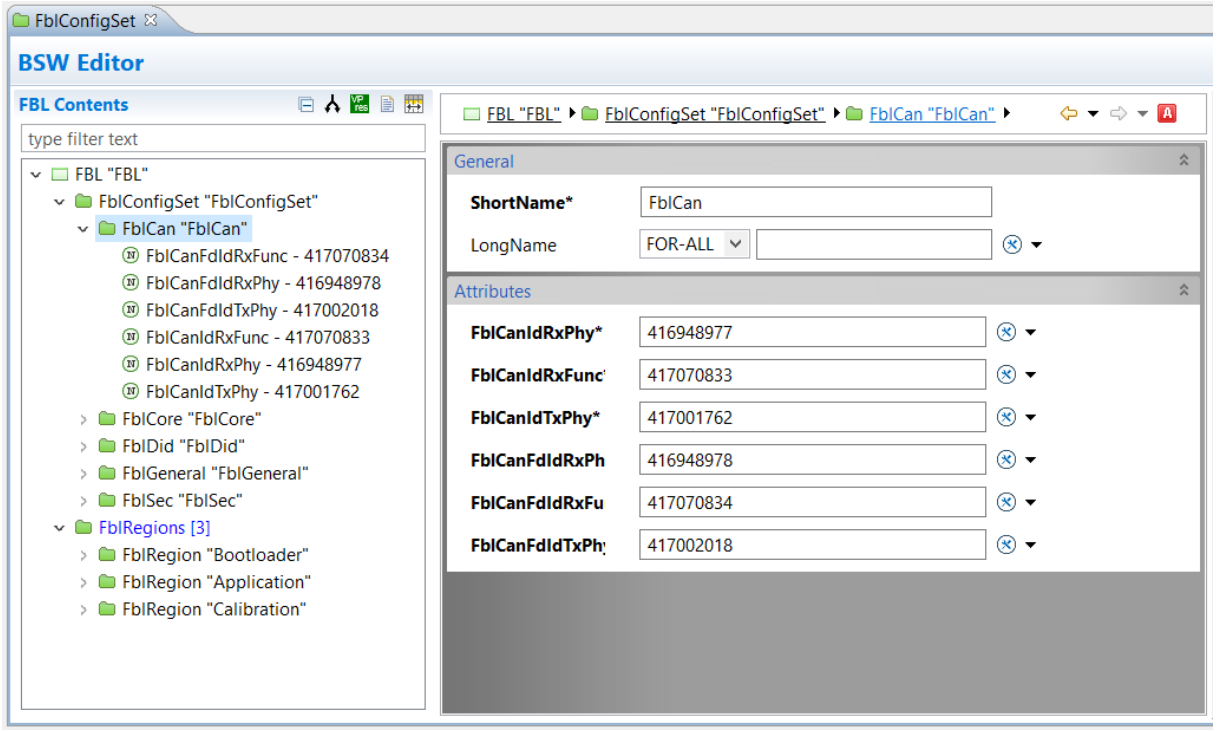


Figure 18: Edit Base Configuration Parameters

Once complete, the user can generate the RTA-FBL instance first by clicking on "Open RTA Code Generator dialog…" as shown in Figure 19 and then, in the opened RTA Code Generator window, by clicking Run as shown in Figure 20.

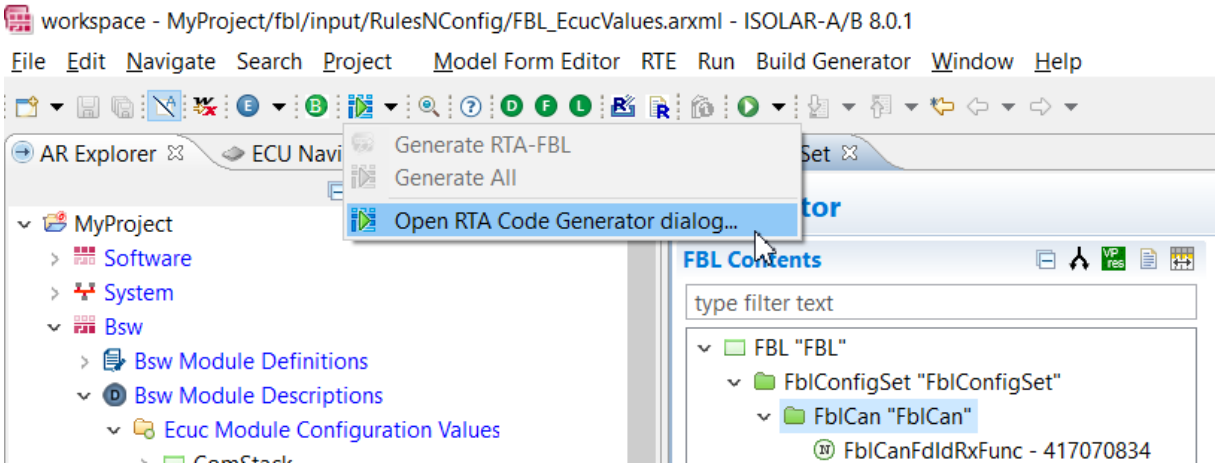


Figure 19: Open RTA Code Generator Dialog

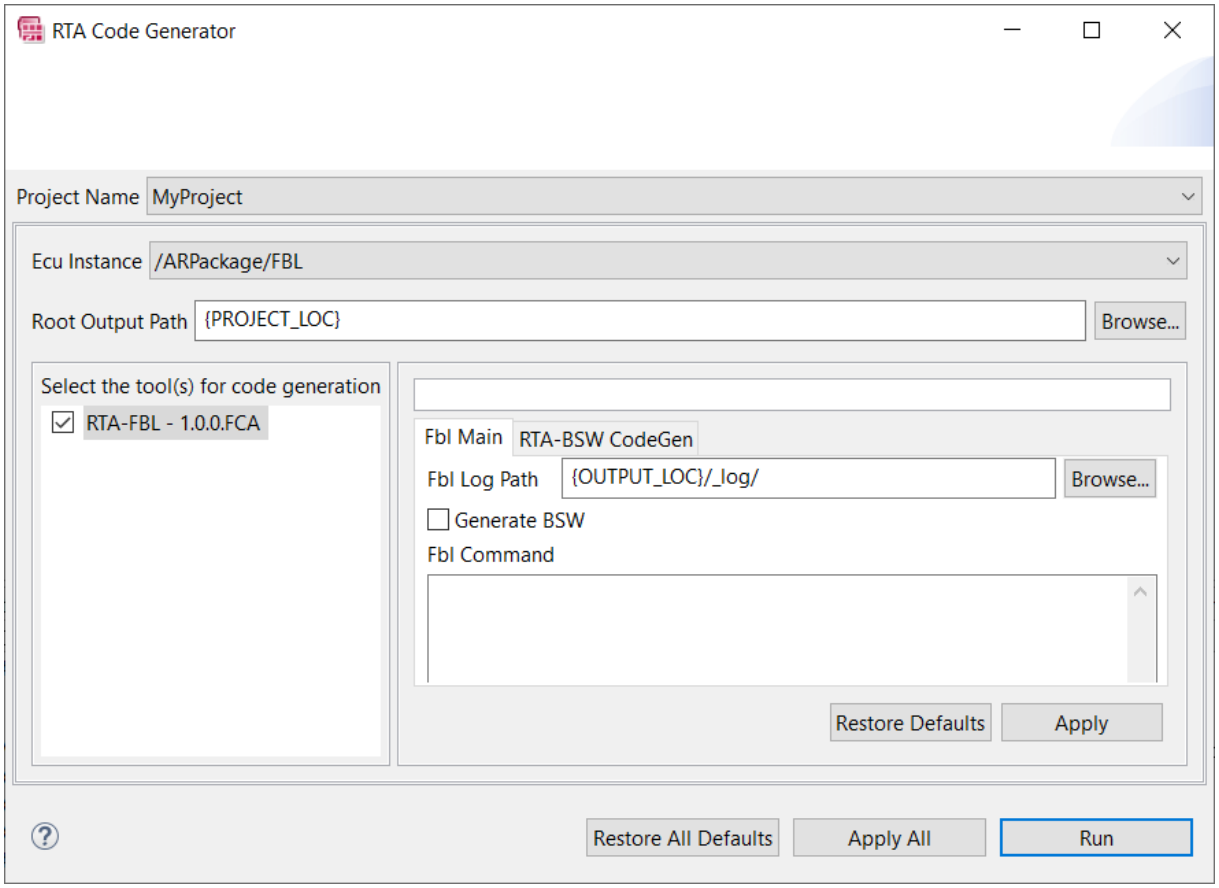


Figure 20: RTA Code Generator

On clicking Run, the RTA-FBL instance is generated. Figure 21 shows the result of a successful generation in the console window.

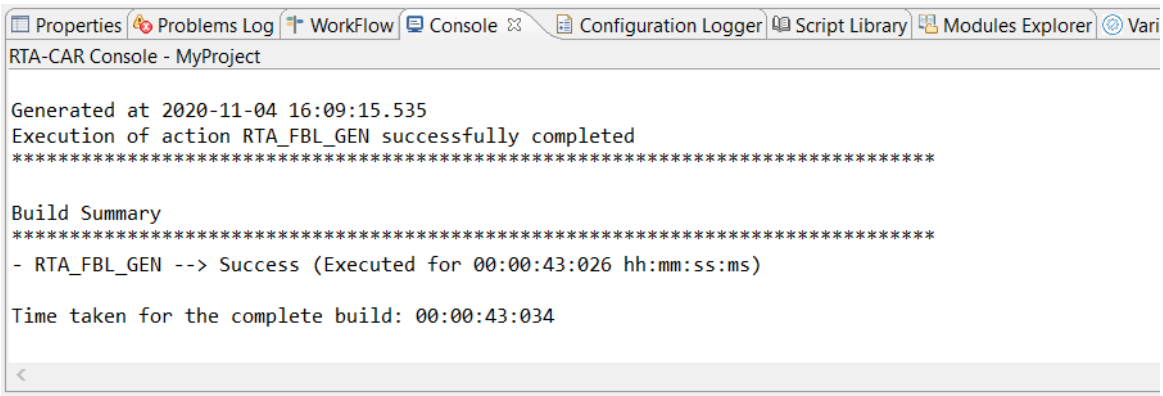


Figure 21: Console Window on Successful Generation

To complete the FBL instance, the user must generate the BSW code by selecting the BSW modules for which the code should be generated in the RTA-BSW CodeGen tab of the RTA Code Generator window, as shown in Figure 22.

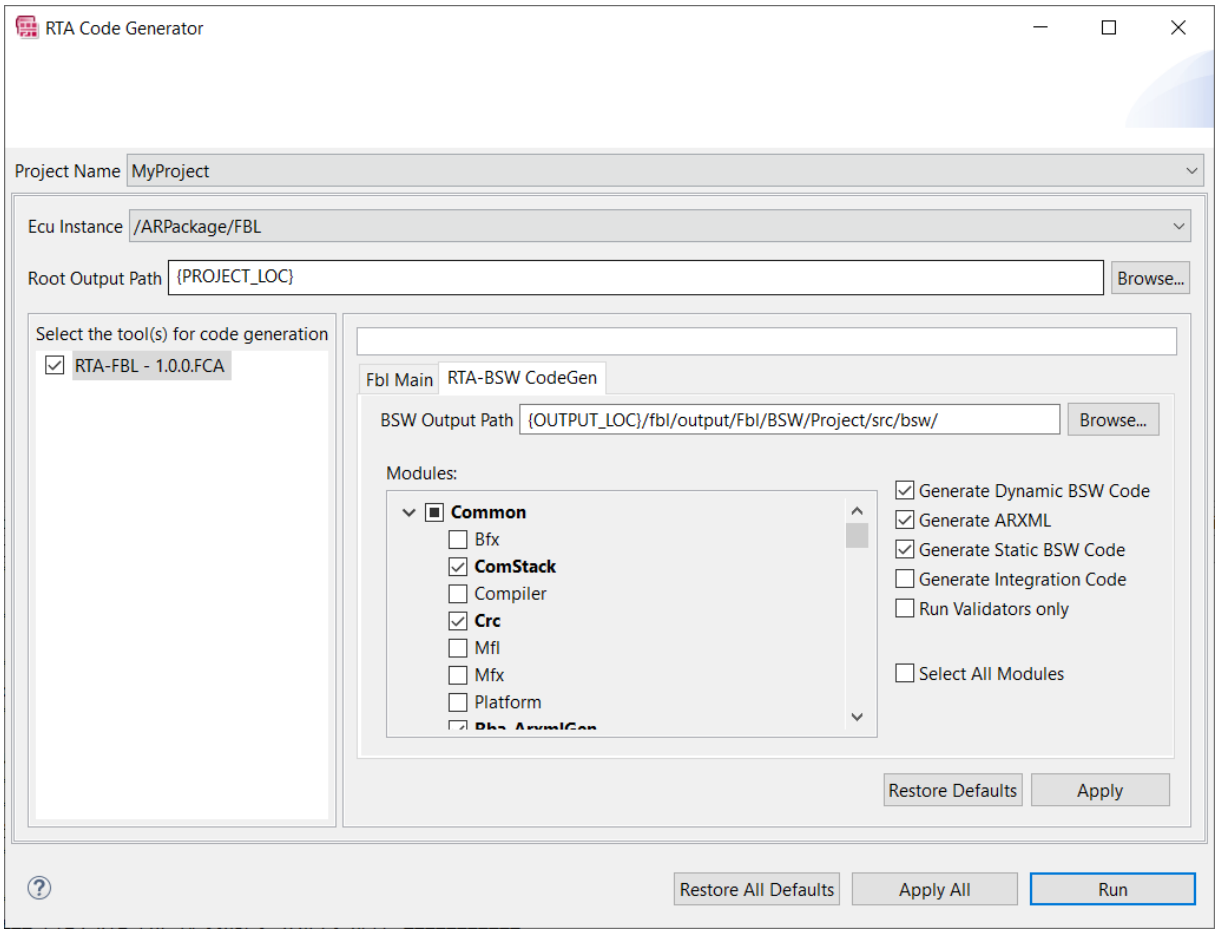


Figure 22: RTA-BSW CodeGen tab

Once complete, check the box Generate BSW in Fbl Main tab of the RTA Code Generator window and click Run.

The user can re-generate the BSW code by clicking on Generate RTA-FBL as shown in Figure 23. Upon successful generation, the popup message in Figure 24 is shown.

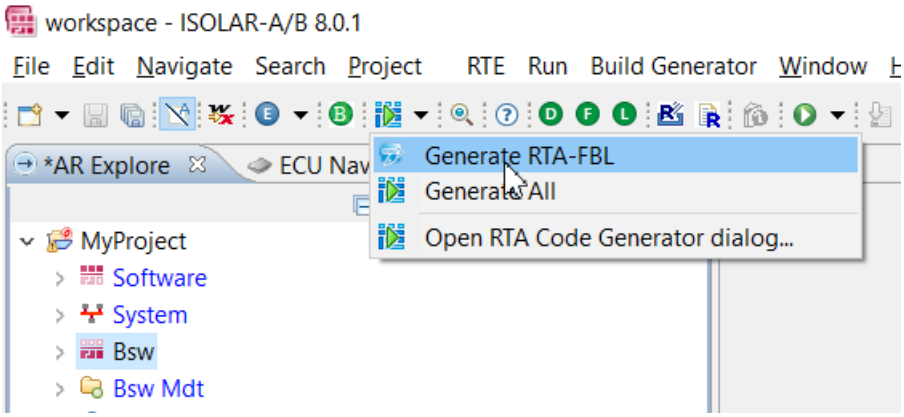


Figure 23: Generate RTA-FBL

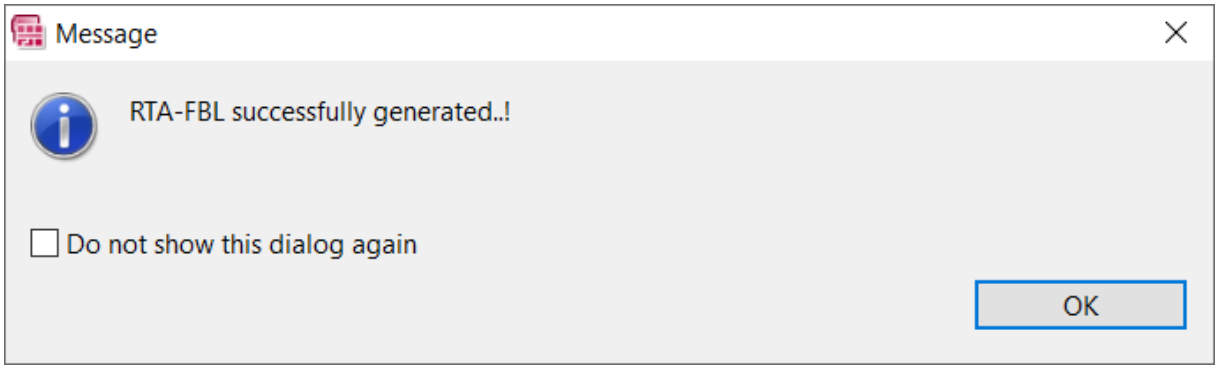


Figure 24: Successful generation

The following paragraphs describe the base configuration parameters that the user can configure. The column Requires BSW Re-Gen in the following tables indicates whether the BSW needs to be re-generated in case that parameter has been changed.

For each new region define the parameters as shown in Table 1.

Table 1: Configuration parameters FblGeneral of the FCA port of RTA-FBL

Parameter	Description	Requires BSW Re-Gen
FblRegionAddressLow	Specifies the low address of the region	Yes
FblRegionAddressHigh	Specifies the high address of the region	Yes
FblRegionMaxAttemptCounter	Specifies the maximum number of reprogramming attempts for the region. When the number of attempts reaches this threshold, it will not be possible to reprogram this block anymore.	Yes
FblRegionType	Specifies the region type: - 0 → Boot region - 1 → App region - 2 → Data region	Yes
FblRegionExternalFlashSupport	Specifies if erasing and writing the region should be handled using MCAL APIs or it is an external region	Yes

	and it is handled by the user.	
FblRegionID	Specifies the index of the region. The index uniquely identifies the region, and it is used by CDA tool to address the block during download	Yes

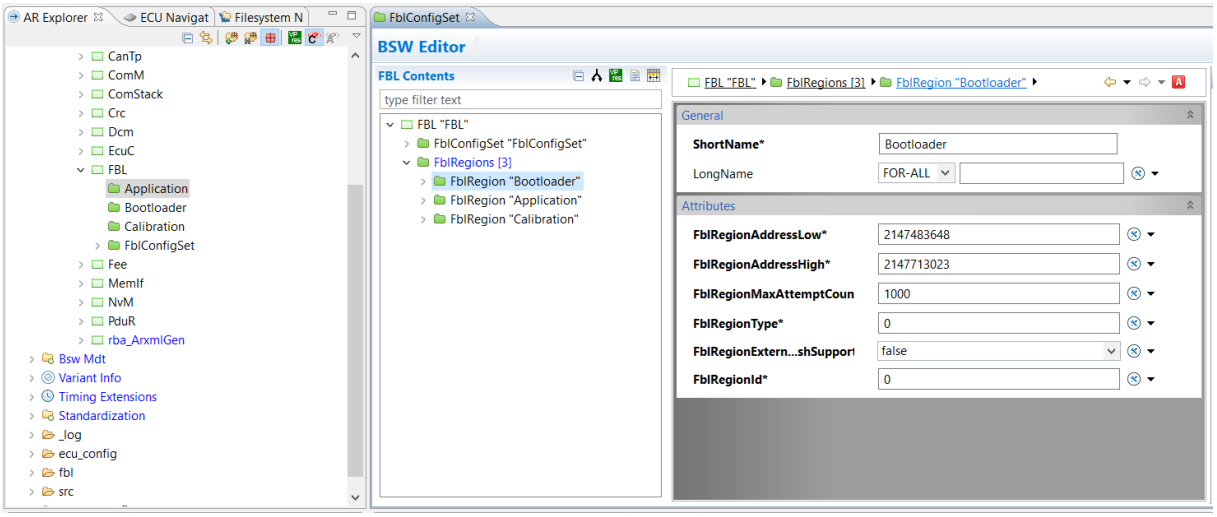


Figure 25: RTA-FBL FCA regions configuration

Table 2 provides a description of each parameter for the container FblGeneral together with whether that parameter is Optional or Mandatory (O/M). If O* or M* is specified, then see the Description for exceptions.

Figure 26 shows the FBL Editor window for FblGeneral.

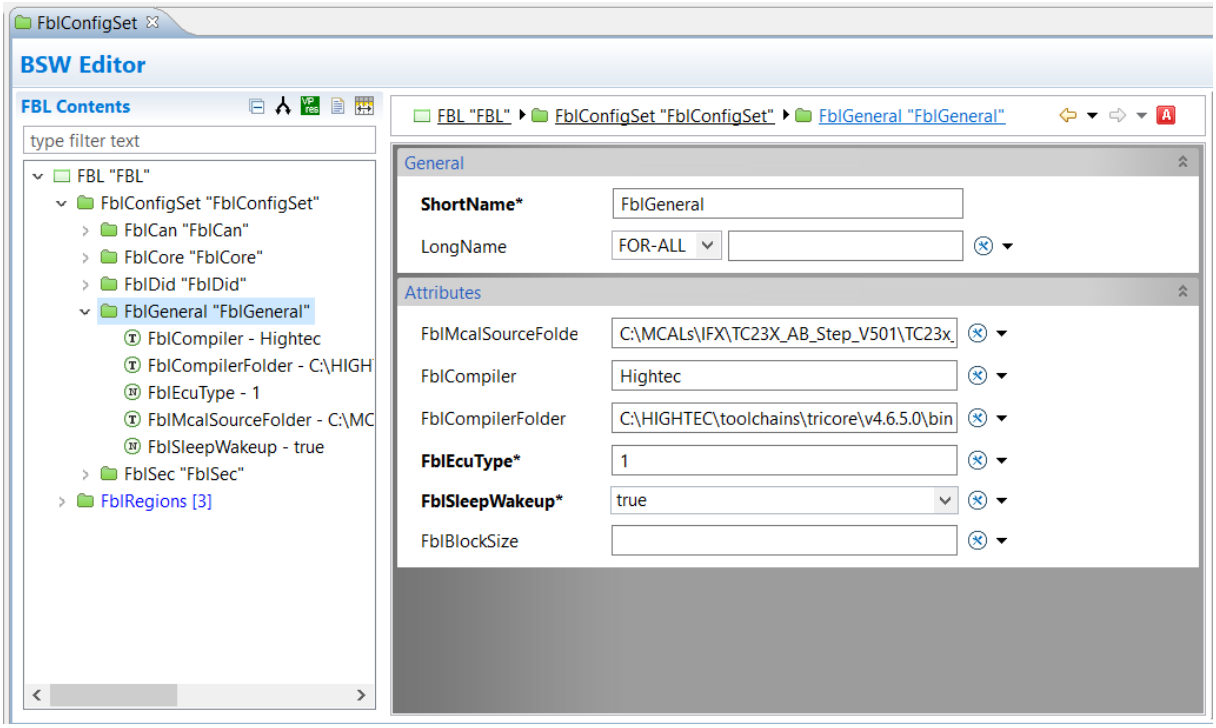


Figure 26: FblGeneral

Table 2: Configuration parameters FblGeneral of the FCA port of RTA-FBL

Parameter	O/M	Description	Requires BSW Re-Gen
FblMcalSourceFolder	O	Specifies the folder where target specific MCAL is installed.	No
FblCompiler	O	Specifies which compiler to use when generating the sample scons build scripts. If this parameter is not specified, then these sample scripts are not created. In this case, you are responsible for creating the build scripts required for creating your bootloader. In most cases, you would initially specify this parameter to create these scripts, but if you then integrate additional code in your FBL and manually modify these scripts, you might want to remove this parameter in order to avoid overwriting your changes. If the compiler you are using is not available, you will have to create your own build scripts. Note that RTA-FBL supports any compiler that is supported by the MCAL, but each target only provides a small set of sample build scripts for a limited set of compilers and is tested only using the compilers and versions as described in Section 4.6.	No
FblCompilerFolder	O	The bin path where the compiler is installed.	No
FblEcuType	M	This parameter indicates which ECU Type is selected. As per FCA specification [1] three different ECU types are supported: - ECU Type A (FblEcuType = 0): no specific security support, the Download is unlocked by a security level 0x1 (Seed&Key). The FW blocks are not protected with authenticity check (signature verification) but only against consistency (CRC check). - ECU Type B (FblEcuType = 1): partial security support, the Download is unlocked by a security level 0x1 (Seed&Key). The FW blocks are protected with authenticity check (signature verification). To accomplish this the ECU must maintain an internal certificate database (TrustStore). - ECU Type C (FblEcuType = 2): full security support, the Download is unlocked by a security level 0x11 (ADA – Challenge-Response). The FW blocks are protected with authenticity check (signature verification). To accomplish this the ECU must maintain an internal certificate database (TrustStore).	Yes
FblSleepWakeup	M	Specifies whether the ECU is a +15 or +30 node and it shall support the Bootloader Sleep/Wakeup Mechanism of [1].	No
FblBlockSize	O	Allows the user to configure the download block size in bytes.	No

Table 3 provides a description of each parameter for the container FblCore together with whether that parameter is Optional or Mandatory (O/M). If O* or M* is specified, then see the Description for exceptions.

Figure 27 shows the FBL Editor window for FblCore.

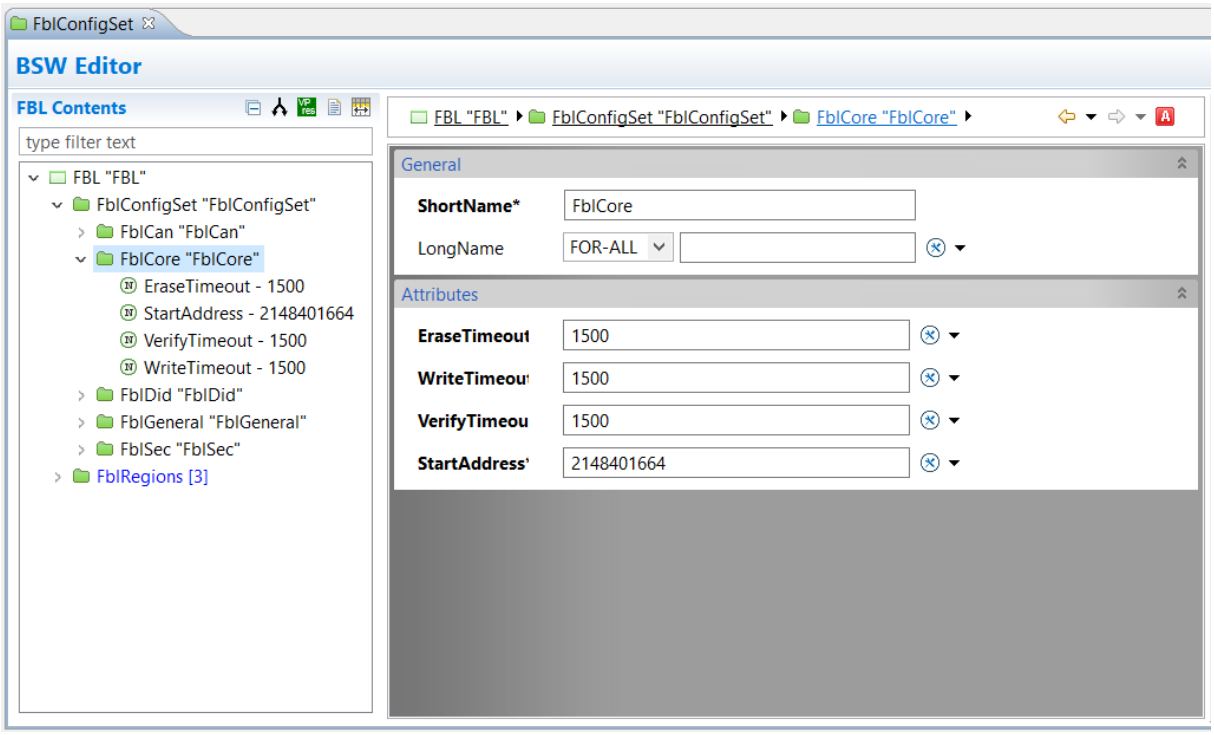


Figure 27: FblCore

Table 3: Configuration parameters FblCore of the FCA port of RTA-FBL

Parameter	O/M	Description	Requires BSW Re-Gen
EraseTimeout	M	Allows to configure the maximum time in microseconds for erase flash operation before timing out.	No
StartAddress	M	The start address of the application software. The bootloader will jump to this address if the application is valid and no reprogramming request has been made.	No
VerifyTimeout	M	Allows to configure the maximum time in microseconds for flash verification operation before timing out.	No
WriteTimeout	M	Allows to configure the maximum time in microseconds for write on flash operation before timing out.	No

Table 4 provides a description of each parameter for the container FblCan together with whether that parameter is Optional or Mandatory (O/M). If O* or M* is specified, then see the Description for exceptions.

Figure 28 shows the FBL Editor window for FblCan.

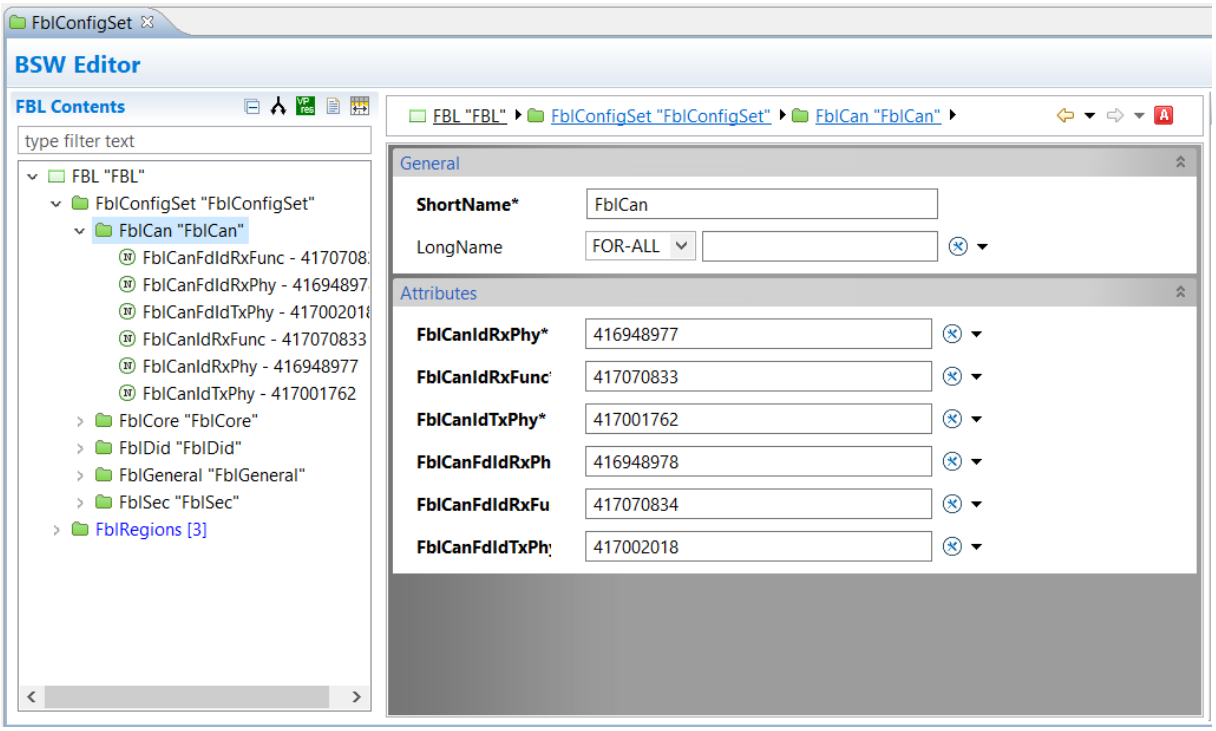


Figure 28: FblCan

Table 4: Configuration parameters FblCan of the FCA port of RTA-FBL

Parameter	O/M	Description	Requires BSW Re-Gen
FblCanIdRxPhy	M	The physical receive Can ID. It is an integer between 1 and 0x1FFFFFFF. Only addressing mode 29 bits is supported.	Yes
FblCanIdRxFunc	M	The functional receive Can ID. It is an integer between 1 and 0x1FFFFFFF. Only addressing mode 29 bits is supported.	Yes
FblCanIdTxPhy	M	The physical trasmit Can ID. It is an integer between 1 and 0x1FFFFFFF. Only addressing mode 29 bits is supported.	Yes
FblCanFdIdRxPhy	M	The physical receive Can FD ID. It is an integer between 1 and 0x1FFFFFFF. Only addressing mode 29 bits is supported.	Yes
FblCanFdIdRxFunc	M	The functional receive Can FD ID. It is an integer between 1 and 0x1FFFFFFF. Only addressing mode 29 bits is supported.	Yes
FblCanFdIdTxPhy	M	The physical transmit Can FD ID. It is an integer between 1 and 0x1FFFFFFF. Only addressing mode 29 bits is supported.	Yes

Table 5 provides a description of each parameter for the container FblSec together with whether that parameter is Optional or Mandatory (O/M). If O* or M* is specified, then see the Description for exceptions.

Figure 29 shows the FBL Editor window for FblSec.

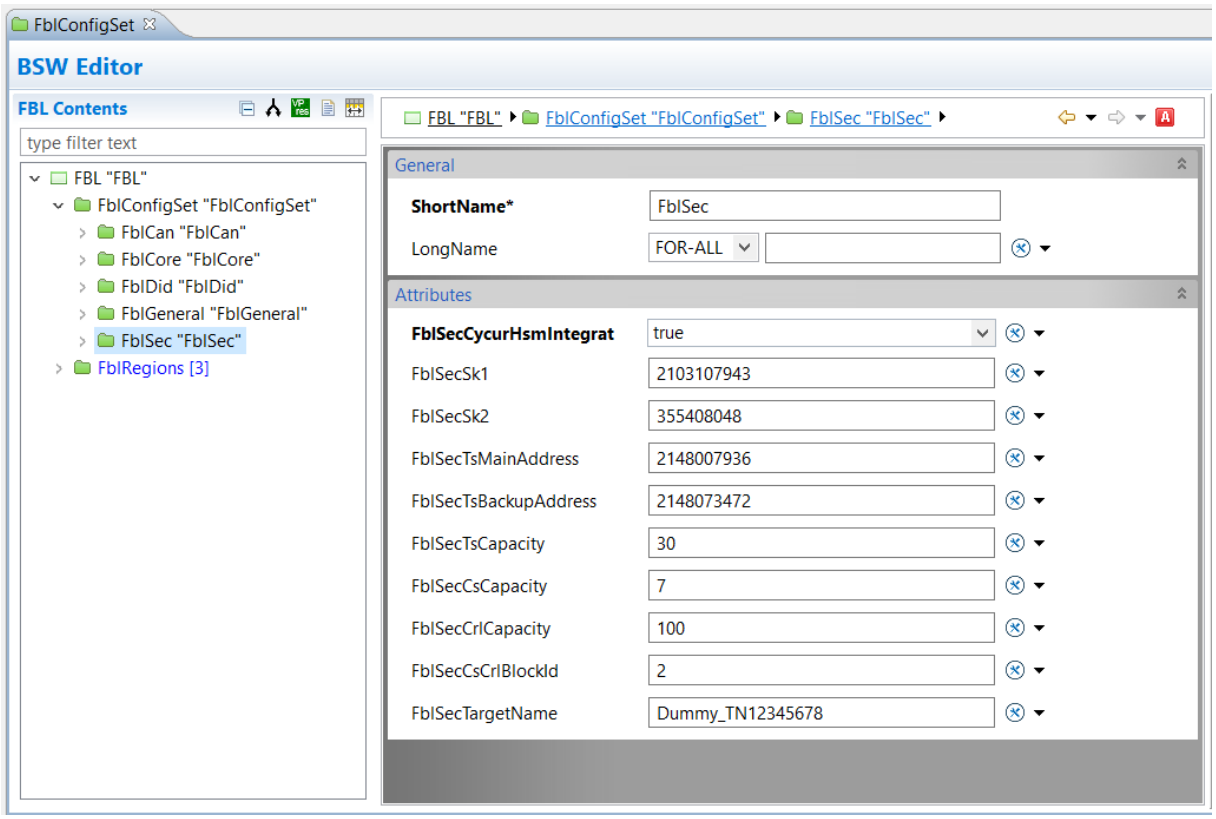


Figure 29: FblSec

Table 5: Configuration parameters FblSec of the FCA port of RTA-FBL

Parameter	O/M	Description	Requires BSW Re-Gen
FblSecCycurHsmIntegration	M	Allows to enable or disable the integration code for CycurHSM 2.x This parameter must be configured for ECU Type B and ECU Type C	No
FblSecSk1, FblSecSk2	M*	Security constants for Seed&Key algorithm to unlock level 0x01, mandatory for ECU Type A and ECU Type B.	No
FblSecTargetName	M*	Specifies the Target Name for Signed Firmware Blocks, for details refer to [1]. This parameter must be configured if CycurHSM sample code is enabled. This parameter is not present, hence not configurable, if CycurHSM sample code is not provided for the selected target.	No
FblSecTsMainStartAddress	O	Specifies the address of the main Trustore Block. This parameter must be configured if CycurHSM sample code is enabled. This parameter is not present, hence not configurable, if CycurHSM sample code is not provided for the selected target.	No

FblSecTsBackup StartAddress	O	Specifies the address of the second Trustore Block, the backup copy. This parameter must be configured if CycurHSM sample code is enabled. This parameter is not present, hence not configurable, if CycurHSM sample code is not provided for the selected target.	No
FblSecTsCapacity	O	Specifies the maximum number of certificates that could be stored in the Trustore. This parameter must be configured if CycurHSM sample code is enabled. This parameter is not present, hence not configurable, if CycurHSM sample code is not provided for the selected target.	No
FblSecCsCapacity	O	Specifies the number of certificates that could be stored in the CertStore. This parameter must be configured if CycurHSM sample code is enabled. This parameter is not present, hence not configurable, if CycurHSM sample code is not provided for the selected target.	No
FblSecCrlCapacity	O	Specifies the number certificates that could be stored in the Certificate Revocation List. This parameter must be configured if CycurHSM sample code is enabled. This parameter is not present, hence not configurable, if CycurHSM sample code is not provided for the selected target.	No
FblSecCsCrlBlockId	O	Specifies the index of the region that should be the CertStore and CRL. The chosen index should be configured in the memory regions as a data region. This parameter must be configured for ECU Type B and ECU Type C	No

Table 6 provides a description of each parameter for the container FblDid together with whether that parameter is Optional or Mandatory (O/M). If O* or M* is specified, then see the Description for exceptions.

Figure 30 shows the FBL Editor window for FblDid.

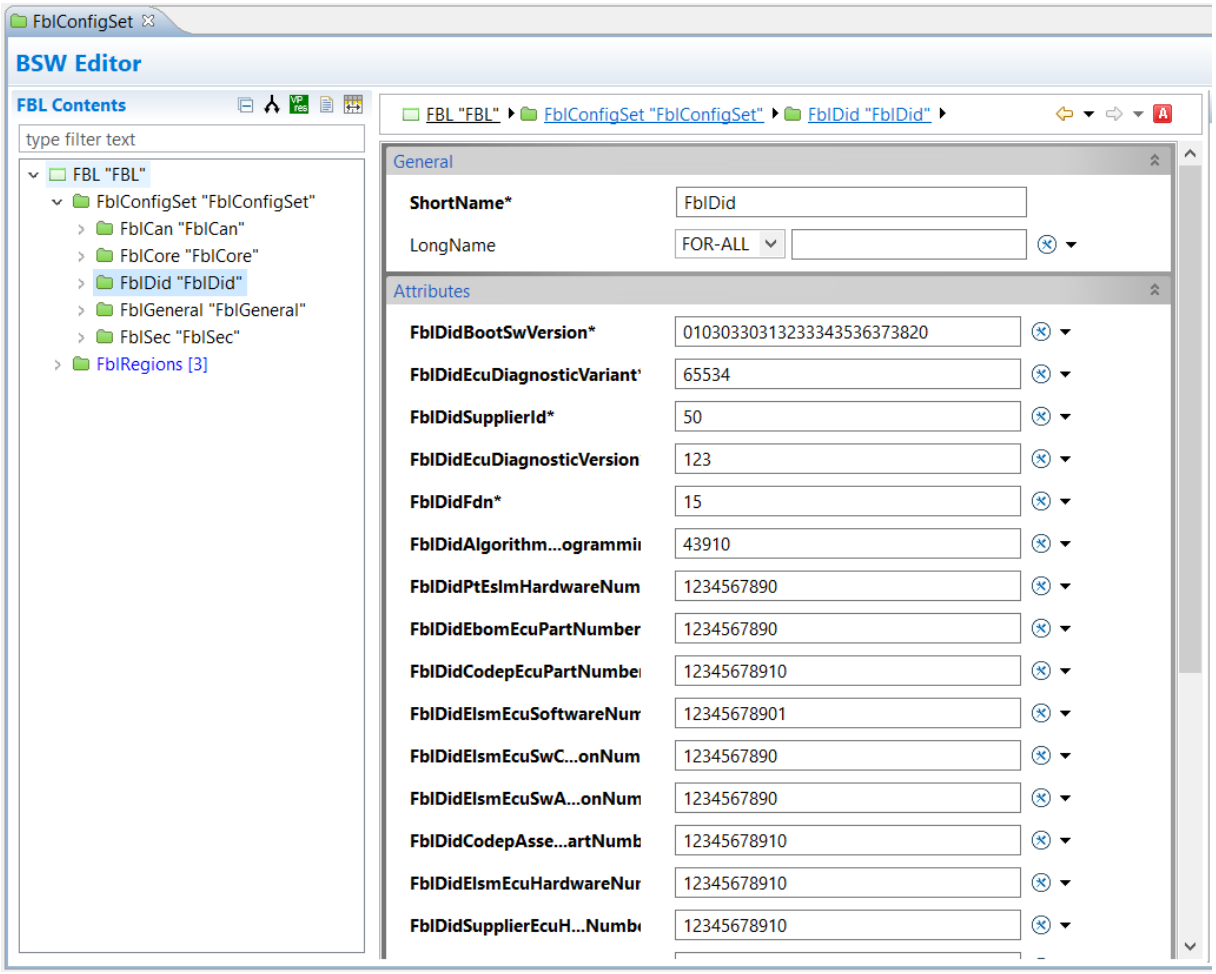


Figure 30: FblDid

Table 6: Configuration parameters FblDid of the FCA port of RTA-FBL

<table>
<tr><th>Parameter</th><th>O/M</th><th>Description</th><th>Requires BSW Re-Gen</th></tr>
<tr><td>FblDidBootSwVersion</td><td>M</td><td>String of 13 hexadecimal bytes value for DID $F180
The format shall be according to [2]
<table>
<tr><th>Byte</th><th>Boot SW Version Information</th><th>Hex Range</th></tr>
<tr><td>0</td><td>SW - Year</td><td>00-3F</td></tr>
<tr><td>1</td><td>SW – Week</td><td>01-34</td></tr>
<tr><td>2</td><td>SW – Patch Level</td><td>00-FF</td></tr>
<tr><td>3 - 12</td><td>SW Identification</td><td>00-FF</td></tr>
</table></td><td>Yes</td></tr>
<tr><td>FblDidEcuDiagnosticVariant</td><td>M</td><td>Specifies Diagnostic Variant value for DID $F110, it should be in range 0x01 - 0xFFFE</td><td>Yes</td></tr>
<tr><td>FblDidSupplierId</td><td>M</td><td>Specifies Supplier ID value for DID $F110, it should be in range 0x00 - 0xFFFF</td><td>Yes</td></tr>
</table>

FblDidEcuDiagnosticVersion	M	Specifies Diagnostic Version value for DID $F110, it should be in range 0x00 - 0xFFFFFFFF	Yes
FblDidFdn	M	Specifies Flash Definition Number value for DID $F110, it should be in range 0x00 - 0xFFFF	Yes
FblDidAlgorithmIdReprogramming	M	Specifies value for DID $F1A4, it should be in range 0x00 - 0xFFFF	Yes
FblDidPtEslmHardwareNumber	M	Specifies value for DID $F188, it should be an 11 bytes value. Each byte should be in range 20; 30-39; 41-5A	Yes
FblDidEbomEcuPartNumber	M	Specifies value for DID $F132, it should be a 10 bytes value. Each byte should be in range 20; 30-39; 41-5A	Yes
FblDidCodepEcuPartNumber	M	Specifies value for DID $F187, it should be a 11 bytes value. Each byte should be in range 20; 30-39; 41-5A	Yes
FblDidElsmEcuSoftwareNumber	M	Specifies value for DID $F188, it should be an 11 bytes value. Each byte should be in range 20; 30-39; 41-5A	Yes
FblDidElsmEcuSwApplicationNumber	M	Specifies value for DID $F18B, it should be a 10 bytes value.	Yes
FblDidElsmEcuSwCalibrationNumber	M	Specifies value for DID $F18A, it should be a 10 bytes value.	Yes
FblDidCodepAssemblyPartNumber	M	Specifies value for DID $F188, it should be an 11 bytes value. Each byte should be in range 20; 30-39; 41-5A	Yes
FblDidElsmEcuHardwareNumber	M	Specifies value for DID $F191, it should be an 11 bytes value. Each byte should be in range 20; 30-39; 41-5A	Yes
FblDidSupplierEcuHardwarePartNumber	M	Specifies value for DID $F192, it should be an 11 bytes value. Each byte should be in range 20; 30-39; 41-5A	Yes
FblDidSupplierEcuSoftwarePartNumber	M	Specifies value for DID $F194, it should be an 11 bytes value. Each byte should be in range 20; 30-39; 41-5A	Yes
FblDidEbomAssemblyPartNumber	M	Specifies value for DID $F194, it should be a 10 bytes value. Each byte should be in range 20; 30-39; 41-5A	Yes
FblDidHwSupplierId	M	Specifies value for DID $F154, it should be in range 0x00 - 0xFFFF	Yes
FblDidSwSupplierId	M	Specifies value for DID $F155, it should be in range 0x00 - 0xFFFF	Yes
FblDidEcuSerialNumber	M	Specifies value for DID $F18C, it should be a 15 bytes value. Each byte should be in range 30-39; 41-5A	Yes
FblDidSupplierManEcuSwVersion	M	Specifies value for DID $F195, it should be in range 0x00 - 0xFFFF	Yes
FblDidSupplierManEcuHwVersion	M	Specifies value for DID $F193, it should be in range 0x00 - 0xFF	Yes
FblDidPolicyType	M*	Specifies value for DID $2954, it should be in range 0x00 - 0xFF. It is mandatory for ECU Type B and ECU Type C.	Yes
FblDidErotan	O	Specifies value for DID $F196, it should be a 15 bytes value. If the parameter is not filled, the DID will not be created.	Yes
FblDIDF122Size	O	Specifies a particular size for DID $F122. If not configured, DID $F122 and its corresponding NvM bock will be generated with default size.	Yes

4.3.3 Files created during generation

When you generate an instance of the FCA RTA-FBL using the RTA-FBL plugin for ISOLAR-AB, files are created within a number of folders that you then use to build your RTA-FBL instance. Table 7 summarizes the folder structure created for the FCA port.

Table 7: Files created by RTA-FBL generation

L1 Folder	Description
. (files in this location)	This contains the root of the project.
./fbl/output/Fbl/Bootloader	This contains the core (port-independent and core-independent) modules.
./fbl/output/Fbl/BSW	This contains the BSW modules.
./fbl/output/Fbl/INFRA/BLSM	The BLSM contains code for initializing the Bootloader. The functions in ./src/BLSM_CallOuts.c can be changed as described in Section 4.7.10, but the functions in BLSM_Main.c should not be changed. It is the integrator's responsibility to ensure that any changes made in the BLSM do not affect the bootloader's correct functionality.
./fbl/output/Fbl/INFRA/OS	The OS contains the cyclic scheduler that calls the module main functions. The OS is provided as a fully functioning and tested sample, but the integrator may replace the OS as described in Section 4.7.9. For example, the integrator may wish to use RTA-OS in order to more easily configure interrupts for other software integrated with RTA-FBL. It is the integrator's responsibility to ensure that any changes made to the OS do not affect the bootloader's correct functionality.
./fbl/output/Fbl/INFRA/Port	This folder contains the code that implements port-specific functionality.
./fbl/output/Fbl/Tools/Prof	This folder contains the Prof to execute a download using INCA, for details refer to Section 5.

4.3.4 Building the RTA-FBL instance.

Sample build scripts are provided for a limited set of compilers. See Section 4.6 for more details on this topic.

If you have the compiler for which sample build scripts are supported on your target, then you can build the RTA-FBL instance, by running the batch file Build_FBL.bat. In order to first clean the output directory before building the FBL, use CleanAndBuild_FBL.bat. If you do not have a compiler for which sample build scripts can be generated, then you will need to create these yourselves. It is recommended in this case that you first create the sample build scripts for any supported compiler, and then adapt these scripts.

4.3.5 The RTA-FBL instance for the Dummy Target

The user can select a dummy target when creating a new project (please refer to section 4.3.1). The dummy target provided with the FCA Port cannot be built. You can only use the generated code as a reference to explore how different parameters change the generated FBL instance.

The FBL for your target will have undergone an in-depth testing using the compiler and MCAL that you have chosen. All targets use a common base that require the tools as described in section 2.10.

Note that although different compilers supported by your MCAL, as well as other MCAL versions for this target should work, these have not been tested. If you do need to generate your bootloader for a different MCAL/compiler combination than that listed above, it is recommended that you first contact ETAS support team.

Dummy Target Memory Layout

In order to allow the user to experiment with different memory space configurations, the dummy target is set up to mimic the memory layout of Infineon's TC233 processor. This processor has a memory layout as shown in Table 8. Memory regions of a space must begin on sector boundaries and the bootloader reserves the first sector (i.e. the memory between 0xA0000000 and 0xA01FFFFF). You can experiment with different configurations of Application, Calibration and Bootloader space if you have not yet received your Target package. For example, if you configure a space that uses a memory region that is not on a region boundary or that enters enter a disallowed space and note the error returned by the FBL generator.

Table 8: Memory layout of the Dummy Target

Bank	Sector	Start	End	Comment
0	0	0xA0000000	0xA0003FFF	Reserved for FBL
	1	0xA0004000	0xA0007FFF	Available for Application/Calibration
	2	0xA0008000	0xA000BFFF	
	3	0xA000C000	0xA000FFFF	
	4	0xA0010000	0xA0013FFF	
	5	0xA0014000	0xA0017FFF	Not available for Application/Calibration
	6	0xA0018000	0xA001BFFF	
	7	0xA001C000	0xA001FFFF	Available for Application/Calibration
	8	0xA0020000	0xA0027FFF	
	9	0xA0028000	0xA002FFFF	
	10	0xA0030000	0xA0037FFF	
	11	0xA0038000	0xA003FFFF	
	12	0xA0040000	0xA0047FFF	
	13	0xA0048000	0xA004FFFF	
	14	0xA0050000	0xA0057FFF	
	15	0xA0058000	0xA005FFFF	
	16	0xA0060000	0xA006FFFF	Not available for Application/Calibration
	17	0xA0070000	0xA007FFFF	
1	18	0xA0080000	0xA008FFFF	Available for Application/Calibration
	19	0xA0090000	0xA009FFFF	
	20	0xA00A0000	0xA00BFFFF	
	21	0xA00C0000	0xA00DFFFF	
	22	0xA00E0000	0xA00FFFFF	
2	23	0xA0100000	0xA013FFFF	
	24	0xA0140000	0xA017FFFF	
3	25	0xA0180000	0xA01BFFFF	

	26	0xA01C0000	0xA01FFFFF	

4.4 Security

Depending on the configuration of the FBL created instance, the integrator may need to add two security modules provided by Escrypt:

- FCA Security Manager (FSM)
- CycurHSM

The generated FBL contains stubs of this two stacks in order to allow the user to test the others FBL functionalities. By default the stubs throw some compilation errors when the project is build, in order to warn the user that the security functionalities are not present. To suppress this errors the following define:

#define SUPPRESS_FBL_SEC_STUBS_ERROR_MSG STD_ON

must be set for the project (either at the highest level in the building chain or locally on the stub files).

4.5 Replace the Security Stub Files

The security stack stubs file are located at the following path "<generated_location>/Fbl/INFRA/Port/src/Security/Stubs".

Stubs can be replaced with real security stacks by selecting the following option in file fbl_settings.py:

```
# Target as defined by the compiler
CPU_TARGET          = 'tc23xx'

# Building options for security stack
SEC_STUB_ENABLED    = True
```

If the Stub enabling option is set to **True**, stubs are used for security, while if are set to **False** real stack are used (always use capital for first letter).

The generated code contains also the folders with needed building scripts for FSM[1], HSM and Cycurlib (FSM, HsmHost and Cycurlib folder). The user needs to copy and paste the correct content from the FSM[1] and HSM deliveries into "<generated_location>/Fbl/INFRA/SecStack/", keeping the generated folder structure:

- **Fsm** folder structure:
  - \inc (from fhwp)
  - \src (from fhwp)
    - \hsm_hw
    - \hsm_sw
  - \Cycurlib (from fhwp)
- **Hsm** folder structure:

[1] Also called as *FHWP*

- CSAI (from ecy_hsm)
  - \api
  - \src
- Host_Applet (from ecy_hsm)
  - \api
  - \src
- Host_Mcal (from ecy_hsm_TC23x_HT)
  - \api
  - \src
- Host_Proxy (from ecy_hsm)
  - \api
  - \src
- Host_Storage from ecy_hsm
  - \api
  - \src

If the integrator does not use CycurHSM solution, the integration steps will be different and he should refer to the security stack used.

4.6 Supported targets

RTA-FBL is a hardware independent FlashBootloader, using the abstraction layers provided by AUTOSAR: the integrator could integrate any targets and MCALs, depending on the customer needs.

This port has been developed and tested with different MCALs and compilers, please contact ETAS if you are interested to know the targets already used.

4.7 Integrator guidelines

Section 4.3 demonstrated how an RTA-FBL project is created in the ISOLAR-AB plugin and the RTA-FBL instance generated. This section explains how and where the integrator can modify this generated instance, as well as integrate the control Application Software on the ECU. This may require adaptation of the FBL as well as adaptations of your Applications Software.

The integrator may need to make the following changes to the default generated FBL:

- Memory layout adaptation,
- Completion of user functions,
- BSW module adaptation (optional),
- C-code startup and trap table updates (optional),
- MCAL adaptation (optional),
- OS adaptation (optional),
- BLSM adaptation (optional).

The integrator may need to make the following changes to the Application Software:

- NvM layout adaptation,
- Boot jump handling.

The integrator may need to make additional changes not described in this User Manual to support specific use cases for his ECU. It is the integrator's responsibility to ensure that any changes made do not affect the bootloader's correct functionality.

4.7.1 FBL: Memory Layout Adaptation

To integrate the FBL in your application the first step to do is decide how to set up your memory regions. This is done using the configuration tool as described in Section 4.3.2. The allowed memory range depends on your target.

An example of a typical memory layout is depicted in Figure 31.

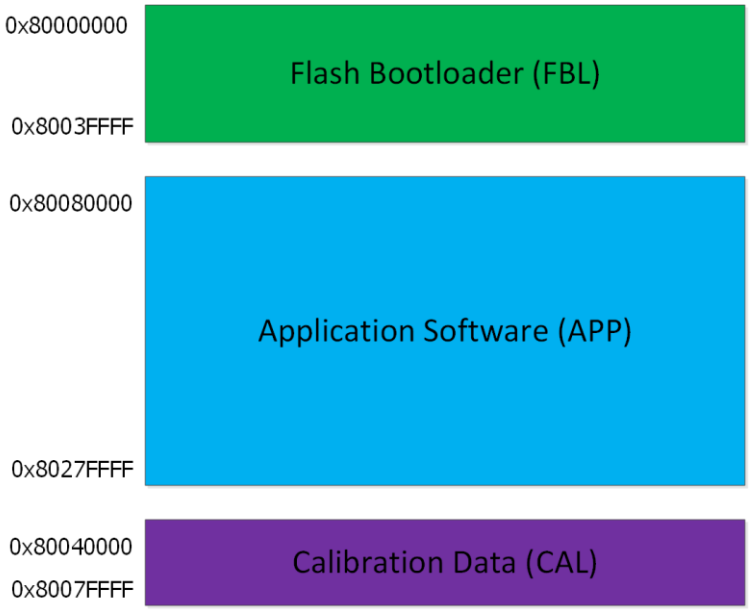


Figure 31: Sample memory layout

4.7.2 FBL: User Functions

This port provides some functions that need to be adapted by the integrator.

Few of them must be filled correctly to be fully compliant with FCA norms, while others are optionals. These can be found in "<generated_location>/Fbl/INFRA/Port/src/FBL_PortUserCode.c.":

4.7.3 Initialization

During system initialization, user has two callouts that could be used for ECU init, useful in case specific hardware initialization is needed (e.g. enable a CAN transceiver or manage the operating mode of a SBC)

Prototype	`void Fbl_Port_UserConfigInitOne ( void )`
Parameter	none
Return Code	none
Functional Description	Called at system startup during Flash Bootloader initialization, before the NvM has loaded the data flash.
Pre-Conditions	none

Prototype	`void Fbl_Port_UserConfigInitTwo ( void )`

Parameter	none
Return Code	none
Functional Description	Called at system startup during Flash Bootloader initialization, after the NvM has loaded the data flash (thus only if the application is not executed)
Pre-Conditions	none

4.7.4 Shutdown

The user function `Fbl_Port_GoToSleep` should be filled to put the ECU in sleep mode, according to the ECU hardware configuration and sleep strategy. The callout is triggered according to [REF-01] when the timers of section 5.2.6 Bootloader Sleep/Wakeup Mechanism are expired

Prototype	`void Fbl_Port_GoToSleep ( void )`
Parameter	none
Return Code	none
Functional Description	Called to put the system in shutdown
Pre-Conditions	none

4.7.5 Application validation

At the end of the download the callout `Fbl_Port_UserValidApplication` is triggered to verify that the application is valid and compatible. The callout should be filled with application specific checks.

When requesting the routine $FF01 – Check Programming Dependencies the callout is executed only if the signature or CRC check on the blocks has been positive, otherwise a negative response is returned without triggering the user callback.

Please note that when bit # 1 of DID 2010 is not set, the DTC P0602-00 is returned by FBL. The same should be done by the application if it is executed in limp mode.

Prototype	`boolean Fbl_Port_UserValidApplication (uint8 * isSwHwCompatible, uint8 * isSwDataCompatible )`
Parameter	`isSwHwCompatible`: pointer to software validity flag, it is used to update bit #1 and bit #5 of DID 2010 **Bit 1 Programming Status - Application** **Bit 5 Software not Compatible with Hardware** `isSwDataCompatible`: pointer to data validity flag, it is used to update bit #2 and bit #6 of DID 2010 **Bit 2 Programming Status - Data** **Bit 6 Software not Compatible with Application Data**
Return Code	`TRUE`: the application will be executed; this value should be returned to execute a valid application or an application in limp mode `FALSE`: the application will not be executed and the ECU will remain in boot mode

Functional Description	Called after an application software download, to verify the application validity and compatibility.
Pre-Conditions	none

4.7.6 Software Identification Update

After an application software update is successfully performed, the callout `Fbl_Port_DownloadSuccess` is triggered to execute user specific code. This callout could be used to update software identification values in NVM with the newer values, or to store the odometer for the last flash programming, or any other ECU specific use case.

The callout is executed at the end of the download only if the signature check has been positive and the application has been considered valid.

Prototype	`void Fbl_Port_UserDownloadSuccess ( void )`
Parameter	none
Return Code	none
Functional Description	Called after a successful download to execute user code (e.g. update software identification)
Pre-Conditions	none

4.7.7 FBL: BSW adaptation

The BSW modules needed by RTA-FBL and generated in the generated BSW project are listed in Table 9. This list is the minimum setup needed for the basic FBL. The integrator could modify the generated BSW project, but should not modify any of the modules of the Dcm or memory stack except as described in this section. As with the MCAL, the integrator must test the complete FBL after making any modifications to the generated BSW project.

Table 9: MCAL modules list

BSW Module(s)	**Notes**
Dcm	The diagnostic communication module
Mem/IF; Fee; NvM	Memory stack modules for the NVM
CanIf; CanSM; CanTp; ComM; ComStack; PduR	Communication stack modules
Crc	Uses for CRC calculation when verifying the downloaded application/calibration.

4.7.8 FBL: MCAL adaptation

The MCAL modules needed by RTA-FBL are listed in Table 10. The list is the minimum setup needed for the basic FBL functionalities (i.e. communication, flashing, etc.). The list does not include customer specific adaptations like external watchdog, external transceivers, external EEPROM, etc. If the integrator also wishes to implement additional user function , then other modules such as the ADC would likely be required. The integrator is then responsible for testing of the complete FBL after the MCAL integration.

Table 10: MCAL modules list

MCAL Module	Notes
Can	CAN driver
Flash Driver	Driver for FLASH erase and programming. This includes the handling of PFLASH and DFLASH, so in some cases could be made by two different modules (i.e. IFX MCALs use Fls for DFLASH and FlsLoader for PFLASH).
Mcu	Provides core functionality such as clock handling, mcu reset, etc.
Port	Provides interface to port pin peripheral.

4.7.9 FBL: OS adaptation

The OS provided with this port is based on a simple cyclic scheduler. This OS does not support interrupts and is non-preemptive. If you need to integrate additional code to the bootloader, you will likely need to adapt this OS. This might involve adding co-routines to the existing tasks or adding new tasks. Adding a new co-routine simply requires adding the function call with the relevant task body in "Fbl/INFRA/Os/src/Os_Tasks.c". If you need to add a new task with a different frequency then follow these steps:

1. Add the task to the task list in "Fbl/INFRA/Os/inc/Os_Tasks.h"
2. Add the task to Os_TaskTable in Os_SchTbl in "Fbl/INFRA/Os/inc/Os_Tasks.c"
3. Create the task body in "Fbl/INFRA/Os/inc/Os_Tasks.c"

The frequency used to derive the task periods is defined by SYSTEM_FREQ_HZ in "Fbl/INFRA/Os/inc/Os.h". You can change this value to match your clock frequency in order to ensure that the tasks are executed at the correct rate. The timer used is target dependent, but you can also change this by adapting the function OsPort_InitOsTimerResource in "Fbl/INFRA/Os/inc/Os_Port.c" and the macro GET_SYSTEM_TIMER in "Fbl/INFRA/Os/inc/Os_Port.h".

IMPORTANT: The integrator is responsible for ensuring that any modifications made to the OS are tested to ensure that the FBL continues to operate as expected. In particular, moving the existing co-routines into a different order or within other tasks will likely result in incorrect behavior.

4.7.10 FBL: BLSM adaptation

The BLSM is used primarily to initialize the BSW and MCAL modules and to start the bootloader. An integrator may need to adapt the BLSM to make the initialization calls for additional modules. This will involve modifying one or more of the Fbl_Port_BLSM_DriverInit functions in "Fbl/INFRA/BLSM/src/BLSM_CallOuts.c". It is strongly recommended that while additional init functions can be added, the existing init functions calls are not moved from their current location with the Fbl_Port_BLSM_DriverInit calls.

In choosing where to add your init functions, note that the NvM is only set up at the end of Fbl_Port_BLSM_DriverInitOne. Therefore, if your integrated code requires the NVM, you should add it in Fbl_Port_BLSM_DriverInitTwo.

IMPORTANT: The integrator is responsible for ensuring that any modifications made to the BLSM are tested to ensure that the FBL continues to operate as expected.

4.7.11 ASW: NvM layout adaptation

Adaptation of the NvM is usually required as the application would rarely already incorporate the FBL NvM layout. This is because the NvM is the interaction mechanism between application and FBL. In particular, the application writes a flag in NvM and then resets, in order to allow the FBL to handle the reprogramming request and to know that it has been issued. Moreover, the FBL could have other internal NvM blocks that need to be copied in case of a page swap by the application. Therefore, the correct integration of NvM layout comprise the complete copy of FBL NvM blocks on the application NvM layout. In order for the layout to be consistent, the Fee persistent IDs of the blocks must match between the application and FBL:

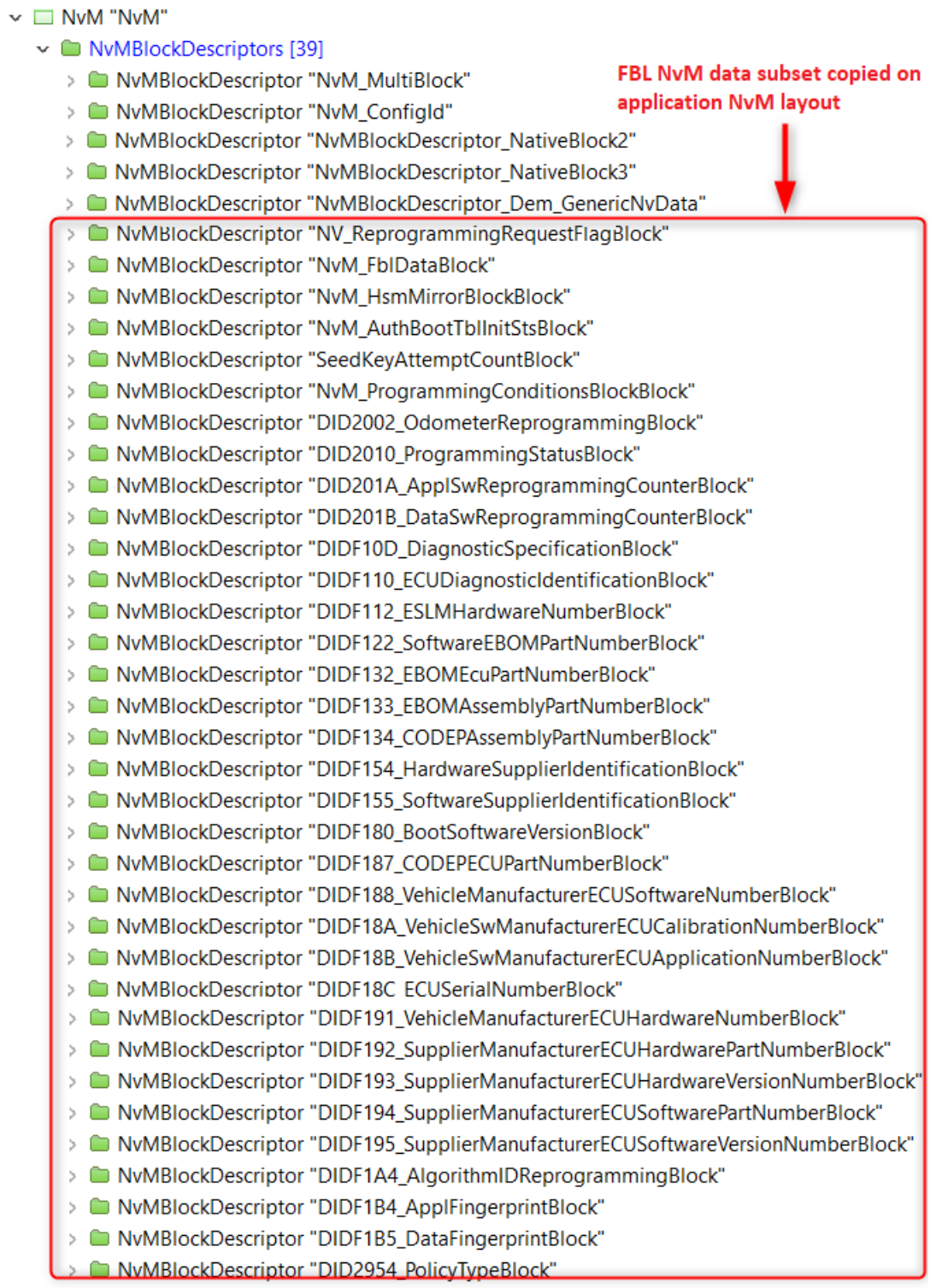


Figure 32: FBL NvM blocks subset on the Application NvM layout

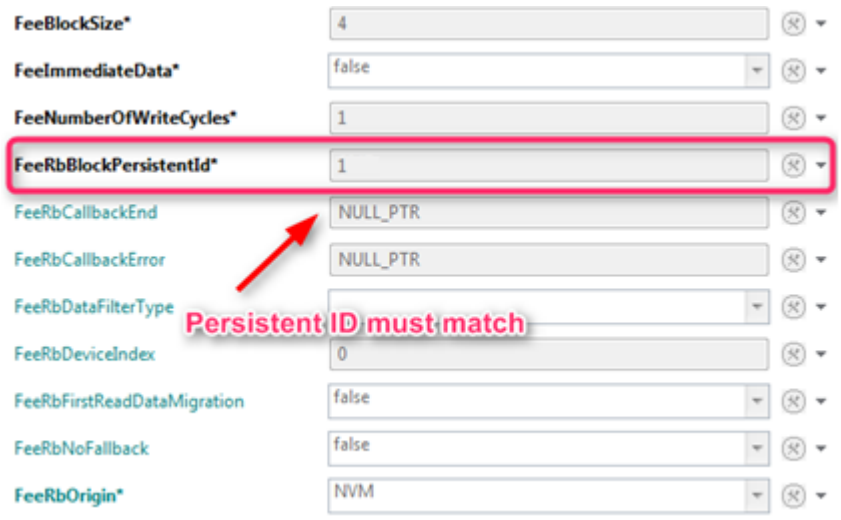


Figure 33: Persistent ID of a block

This completes the update of the BSW, so it **could be re-generated** now in order to allow the update of SWC Description of service components. Note that if you already are using the Persistent IDs generated by fblgen, you can change these values as long as you keep them consistent with the values in the Application.

4.7.12 ASW: Boot Jump Handling

Once memory layout, Dcm and NvM configuration are compatible between Application and FBL, the two entities are compatible and can "communicate". What is missing at this stage is to integrate the Application to the FBL jump logic. This can be done in multiple ways, but in the end, apart from other OEM-specific steps, it can be done with the sequence highlighted in Figure 34.

Before jumping into bootloader, the application has to be sure that the needed data is saved in NvM. The data shared between the FBL and the application to manage the boot jump are the reprogramming request flag and the Dcm reprogramming conditions. The flag is checked at startup by the FBL to decide if jumping in application or remaining in boot mode, while the Dcm reprogramming conditions are used to send the final response to $10 02 request, if the application wants to send only a NRC $78 and the FBL is in charge of sending the final response.

The reprogramming request flag is a uint32 and it shall be set by the application to `FBL_BOOTM_REPROGRAMMING_REQUEST_FLAG_STATE_ON`. For details on Dcm reprogramming conditions structure, please refer to AUTOSAR specification of the structure of type `Dcm_ProgConditionsType`

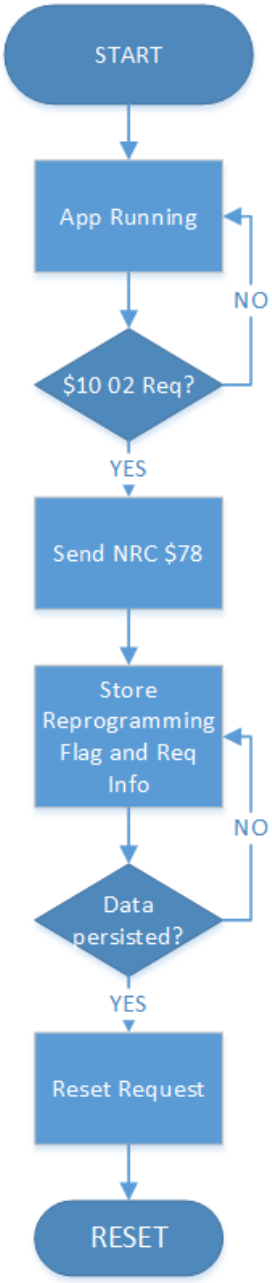


Figure 34: Handling of jump logic

5 How to Flash the ECU with INCA and the ProF Script

This section explains how to perform the download process with INCA.

Step1: Install INCA and ES58x driver

Before starting the actual download process, INCA and the HW interface driver must be installed on the machine that will be used to flash the ECU. In this example, we will use an ES58x as HW interface.

1. Please make sure you are using the official released INCA V7.2.x version package, and make sure the valid node-locked license is also installed on your testing PC,
2. Make sure that the used HW interface driver is installed correctly.

Step2: Setup the environment

Launch INCA and add a new database using the "New" button on the toolbar and name it with your preferred db name.

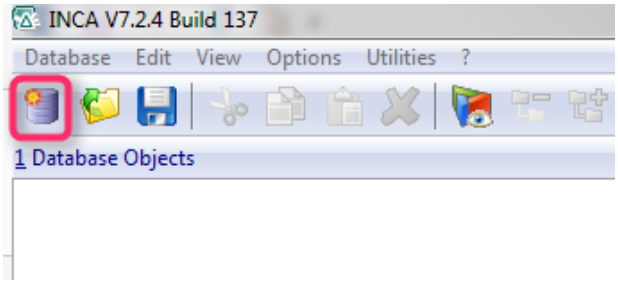


Figure 35: New database creation

Right-click on the created top folder ("DEFAULT") and select Add→Workspace.

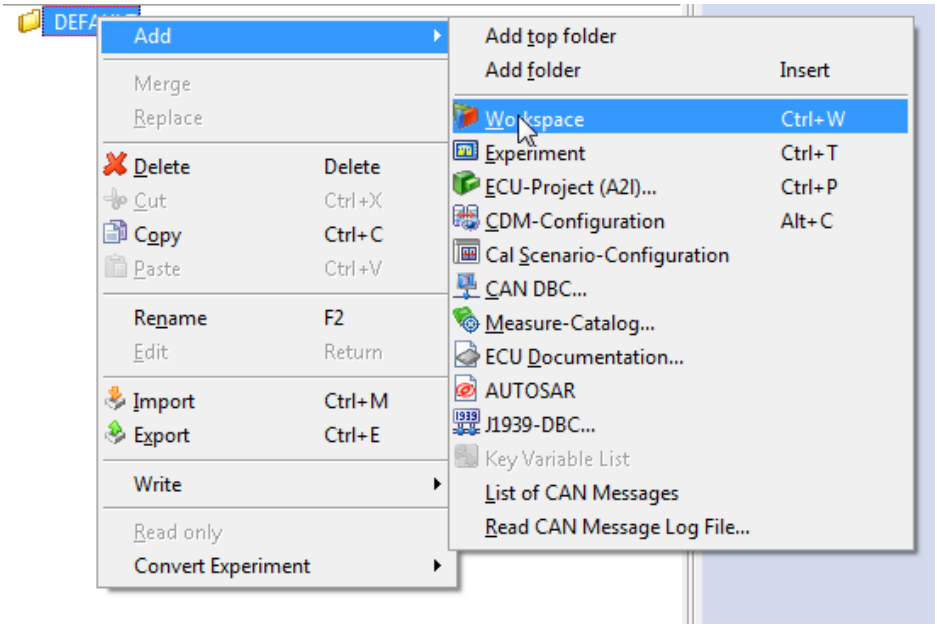


Figure 36: Adding a workspace

Then, right-click again from the "DEFAULT" folder and select Add→ECU-Project (A2L):

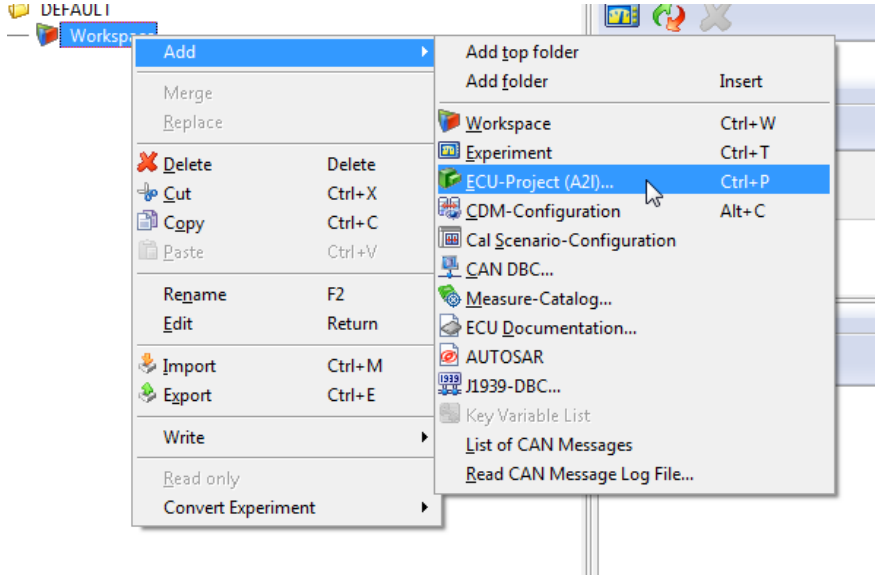


Figure 37: Adding an ECU project

From the dialog window navigate to the where the ProF script is located in your delivery and select the file "ECU_dummy.a2l", and following this, from the same path, choose the file "ECU_dummy.hex".

Left-click on the newly created Workspace and select the HW icon on the bottom-right window.

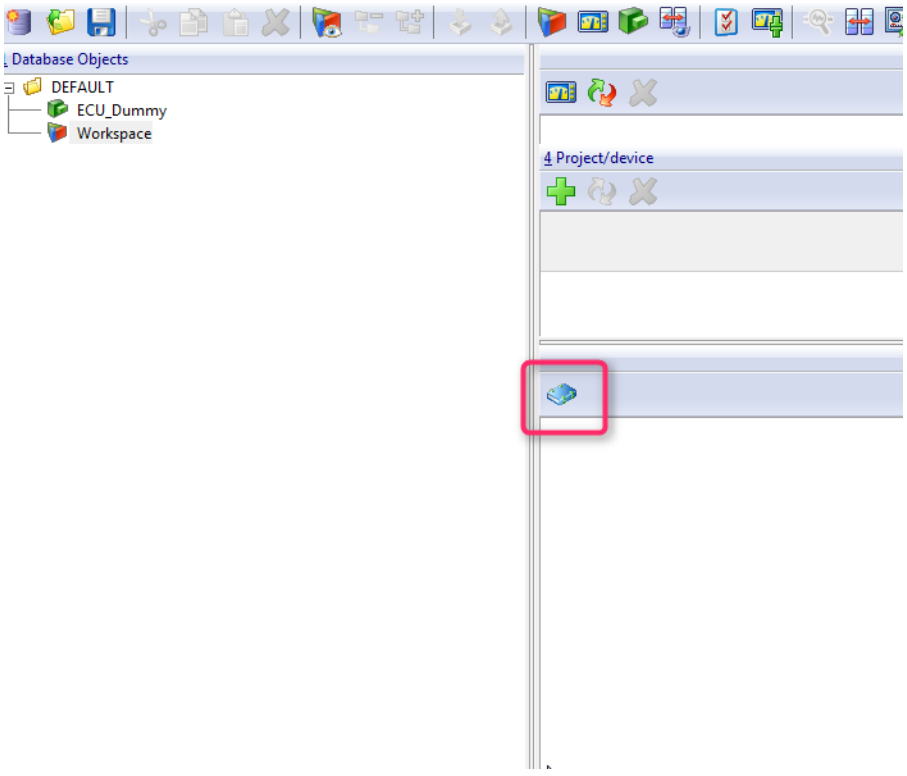
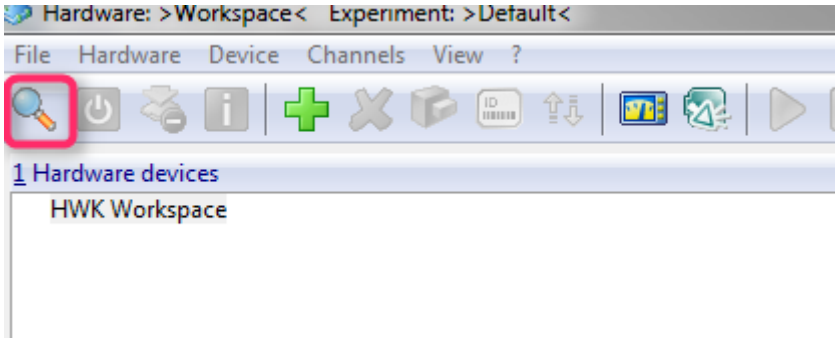


Figure 38: HW icon

From the newly opened window select the "Search for connected devices" option on the toolbar, then select USB (and click OK), then UDS (and click OK) and finally associate the ECU_dummy project, as shown in Figure 41.



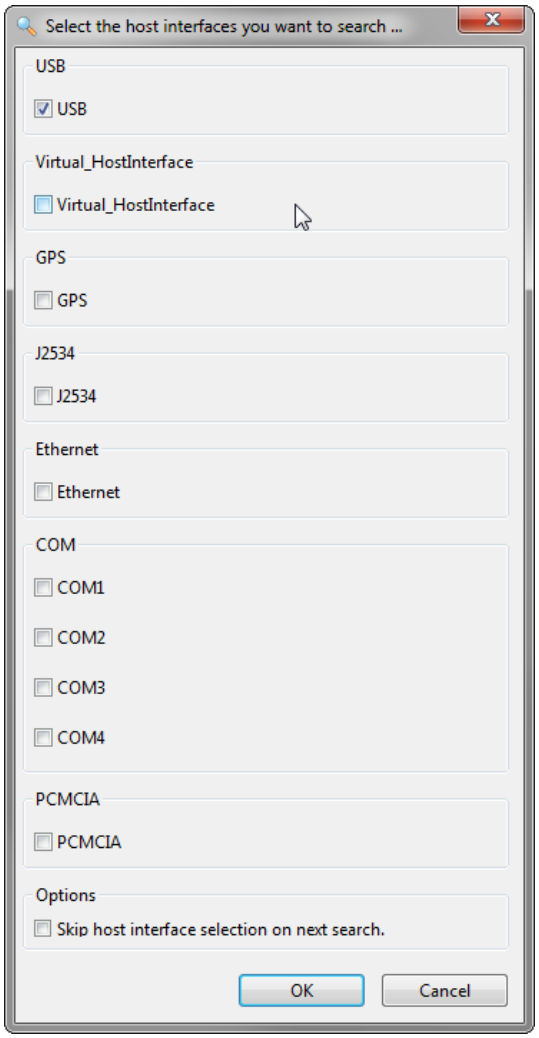


Figure 39: Search of HW interface

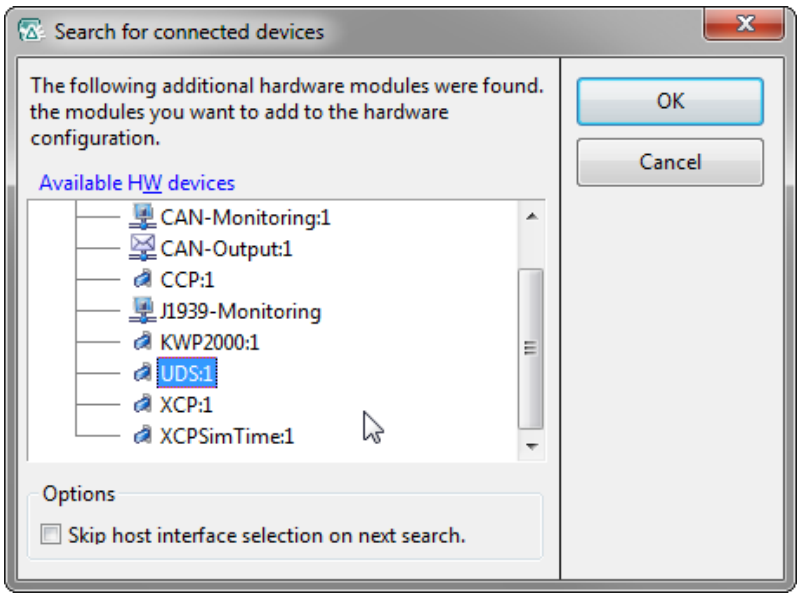


Figure 40: Selection of UDS interface

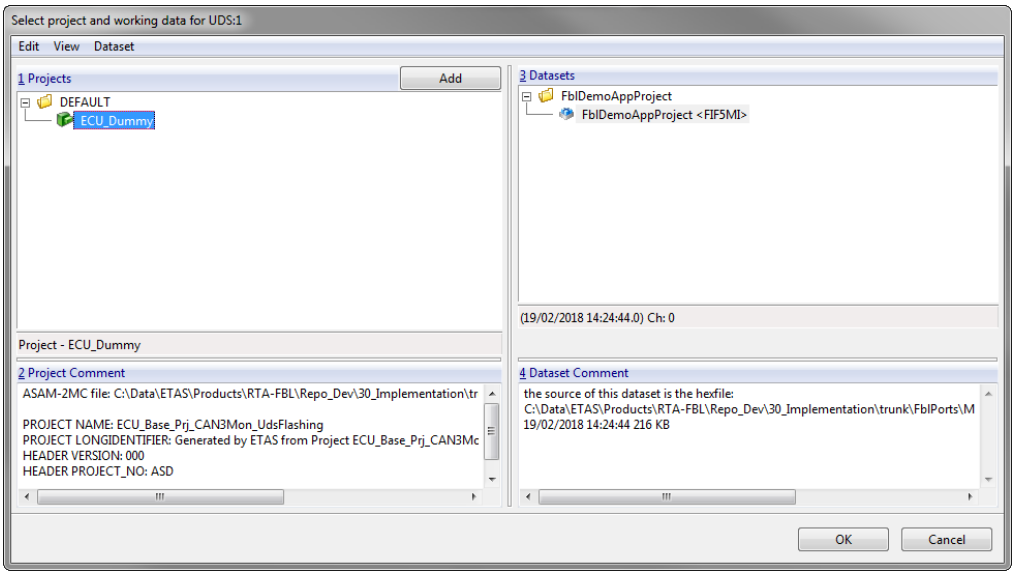


Figure 41: Association of ECU_dummy project

Now you can click the "Initialize Hardware" button on the toolbar and the devices should initialize and appear as connected on the left 

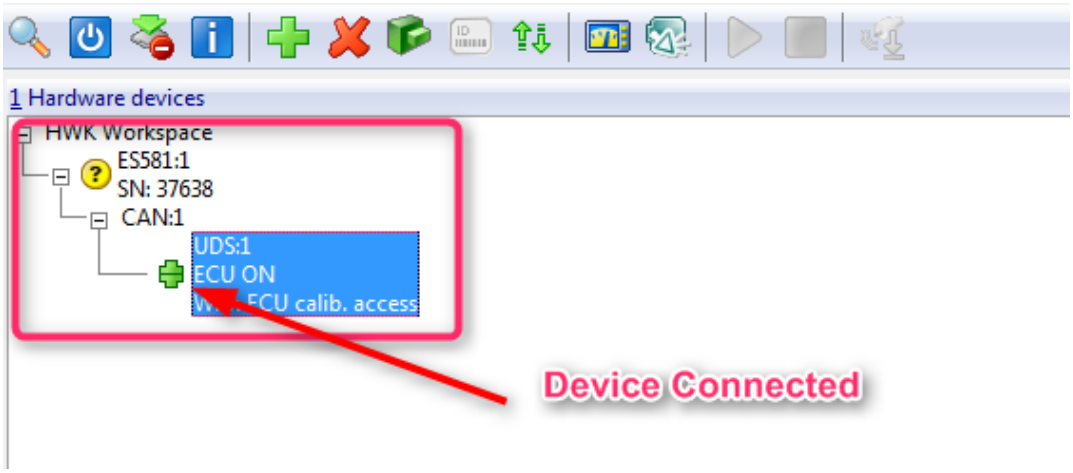


Figure 42: Device connection

Step3: Install ProF script

Now you can click the "Manage Memory Page" icon on the toolbar.

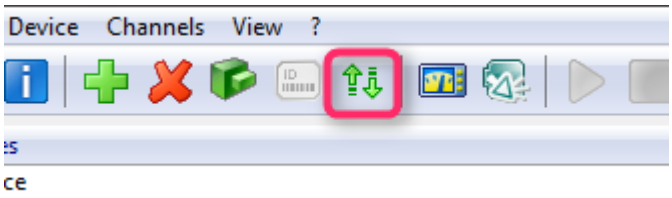


Figure 43: Manage memory page selection

From the "Utilities" menu of the popup window, select the option "Configure PROF".

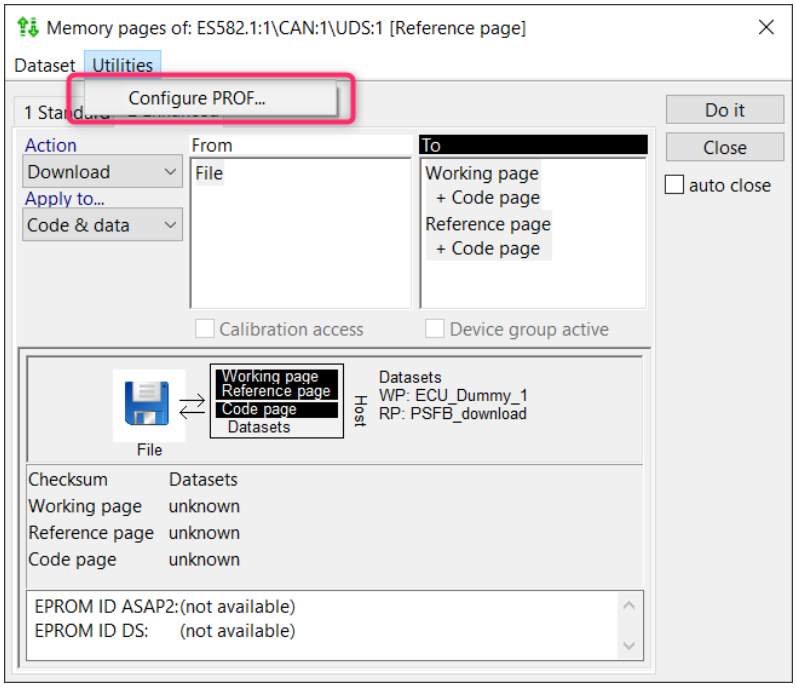


Figure 44: ProF configuration selection

In the new window, select Install and navigate to the path <generated_location>\Tools\Prof\Install\CAN or <generated_location>\Tools\Prof\Install\CANFD, depending on the needed bus type.

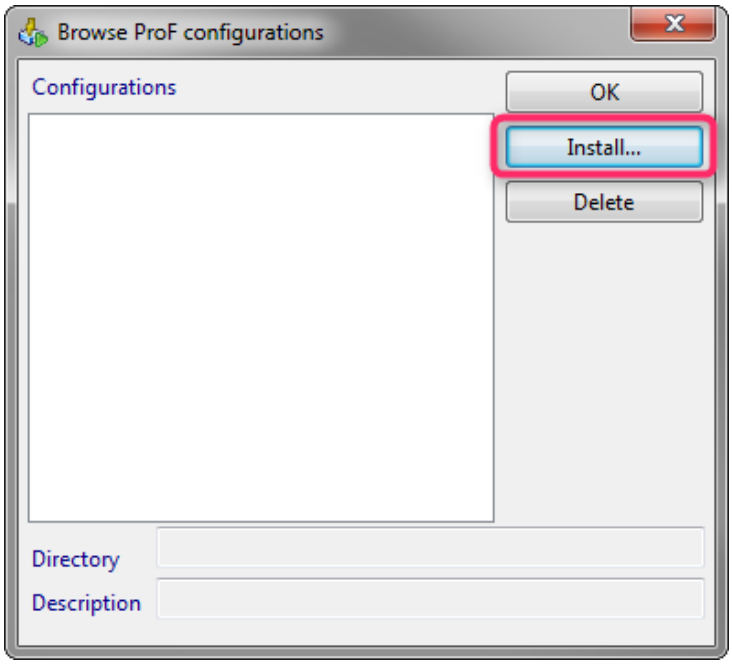


Figure 45: Install ProF script

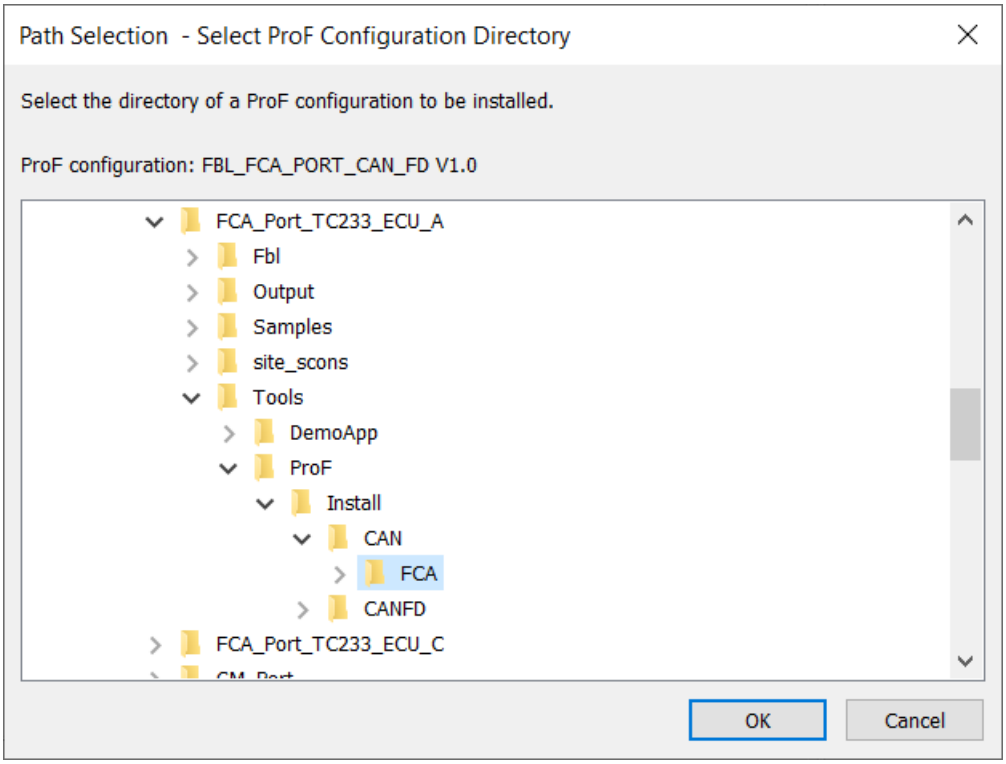


Figure 46: Selection of ProF script to install

When you click "OK", the ProF script will be installed.

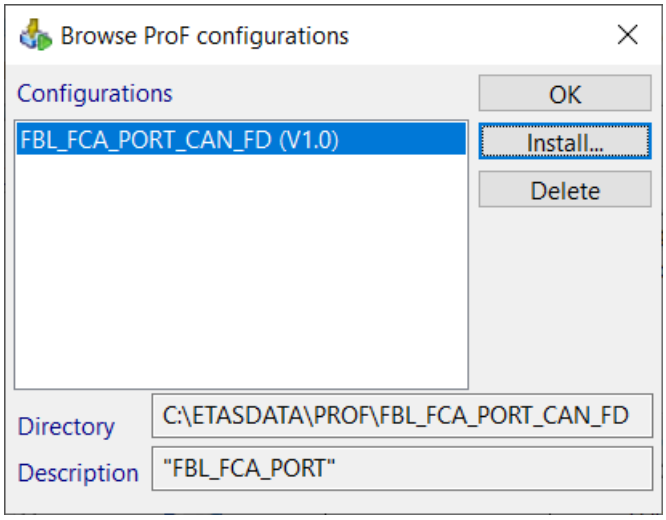


Figure 47: ProF script installation confirmation

Step4: Flash the ECU

To start the download process switch to the "Enhanced" tab on the "Manage memory page" window, select "Flash Programming" as Action and click on "Do It".

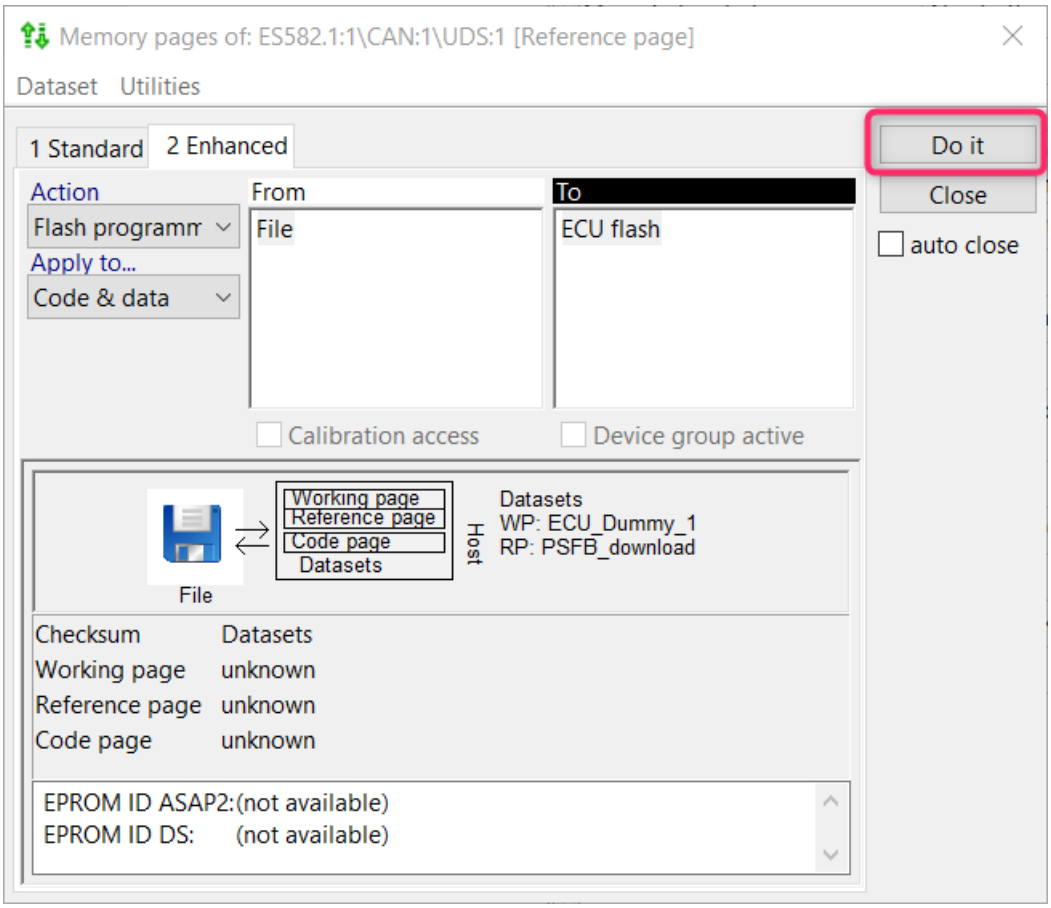


Figure 48: Download Process Start

From the new dialog window, select the .hex file you want to download and click on "Open".

From the ProF mask that is displayed select the option you want to use and click "OK" to start the download process.

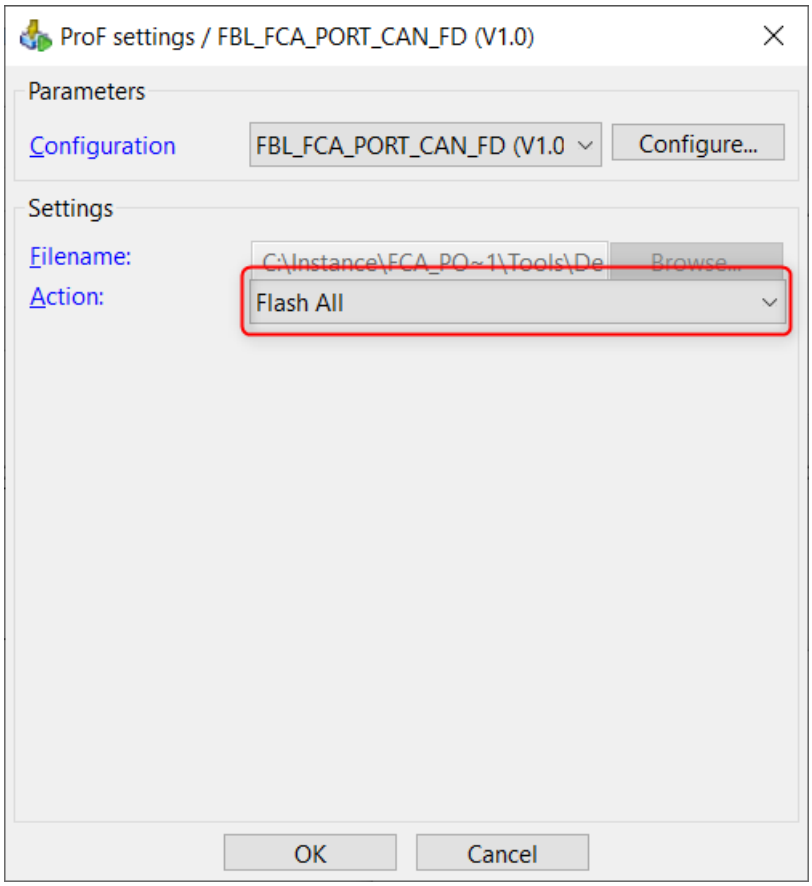


Figure 49: Option Selection

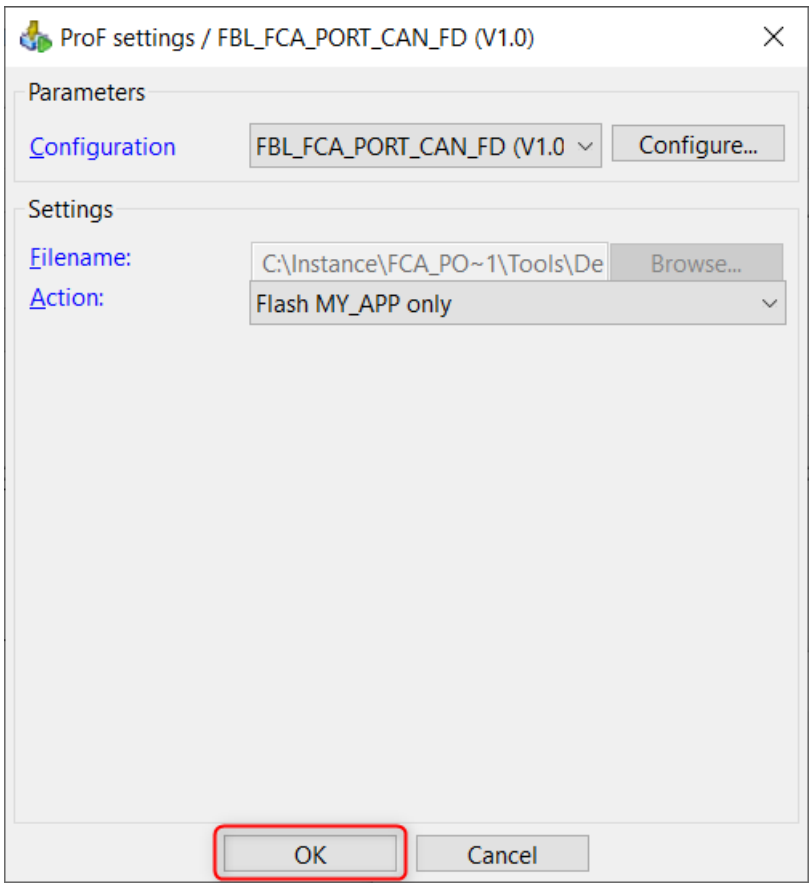


Figure 50: Start Download

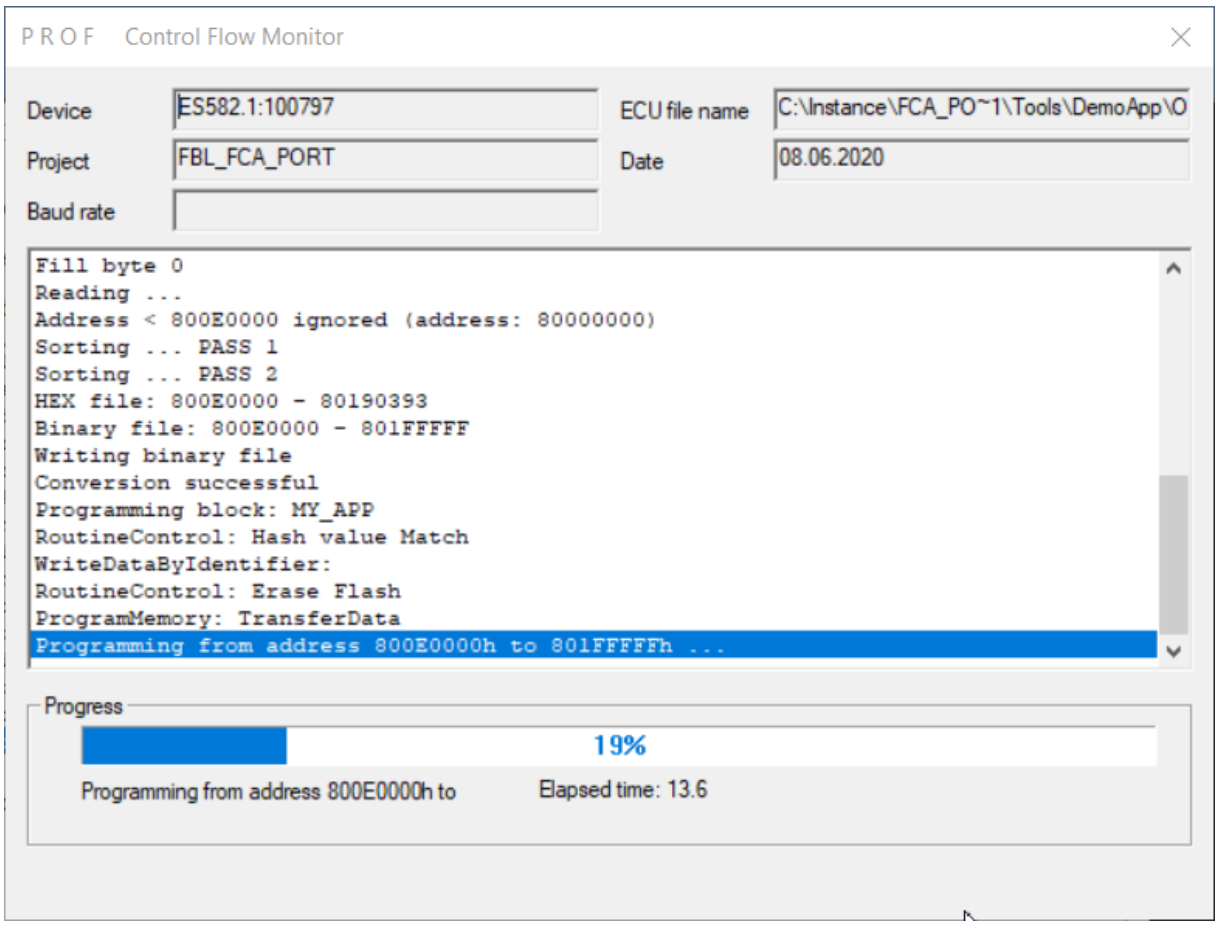


Figure 51: Download in progress

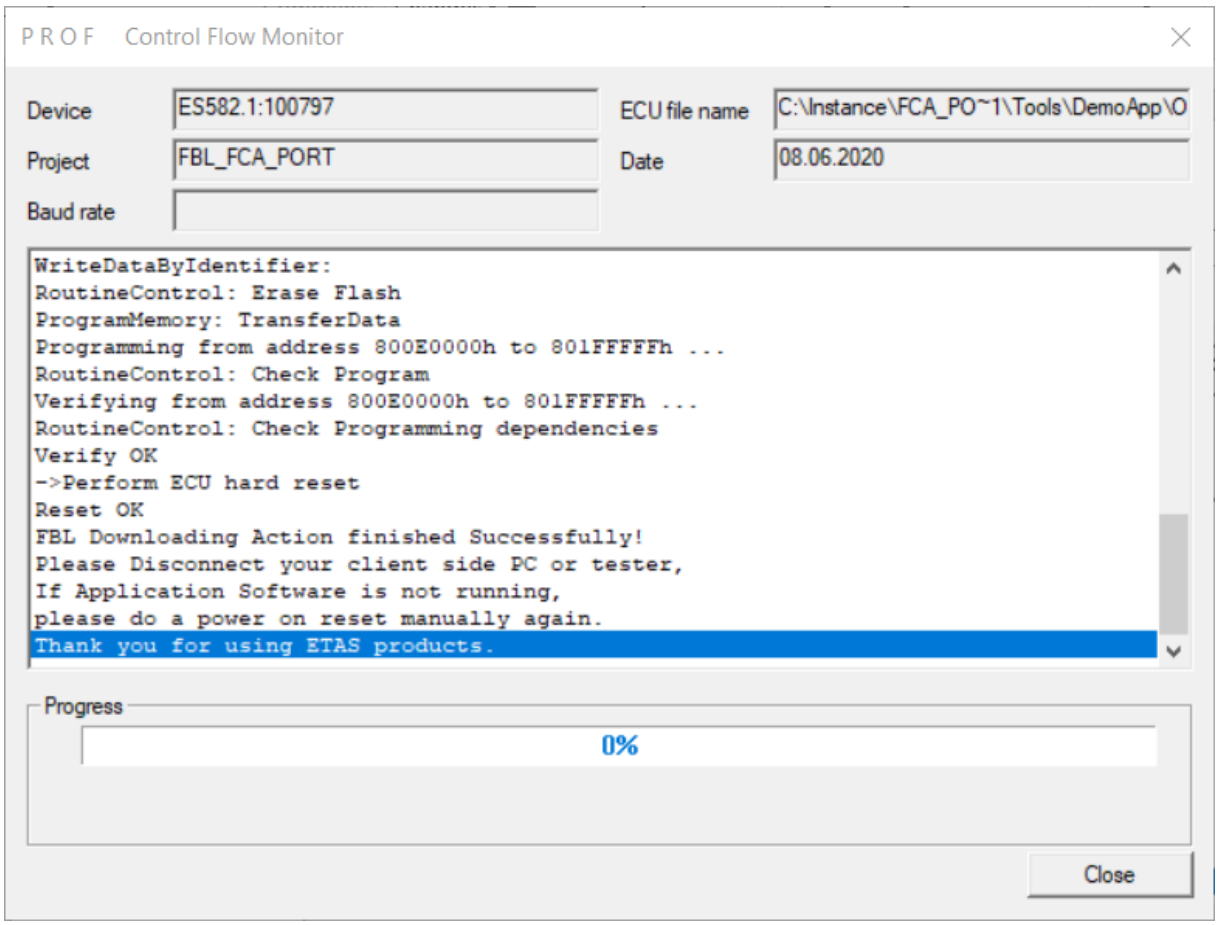


Figure 52: Download Completed

6 Privacy

6.1 Privacy Statement

Your privacy is important to ETAS so we have created the following Privacy Statement that informs you which data are processed in RTA-FBL, which data categories RTA-FBL uses, and which technical measure you have to take to ensure the users privacy. Additionally, we provide further instructions where this product stores and where you can delete personal or personal-related data.

6.2 Data Processing

Note that personal or personal-related data respectively data categories are processed when using this product. The purchaser of this product is responsible for the legal conformity of processing the data in accordance with Article 4 No. 7 of the General Data Protection Regulation (GDPR). As the manufacturer, ETAS GmbH is not liable for any mishandling of this data.

6.3 Data and Data Categories

When using the ETAS License Manager in combination with user-based licenses, particularly the following personal or personal-related data respectively data categories can be recorded for the purposes of license management:

- Communication data: IP address,
- User data: UserID, WindowsUserID.

6.4 Technical and Organizational Measures

This product does not itself encrypt the personal or personal-related data respectively data categories that it records. Ensure that the data recorded are secured by means of suitable technical or organizational measures in your IT system. Personal or personal-related data in log files can be deleted by tools in the operating system.

7 ETAS Contact Addresses

ETAS HQ

ETAS GmbH
Borsigstraße 24 Phone: +49 711 3423-0
70469 Stuttgart Fax: +49 711 3423-2106
Germany WWW: www.etas.com

ETAS Subsidiaries and Technical Support

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 subsidiaries WWW: www.etas.com/en/contact.php