RTA-FBL STANDARD PORT
USER MANUAL
Status: RELEASED

Contents

1 Introduction

This user manual introduces the RTA-FBL Standard port. It provides an overview of the RTA-FBL architecture and software design. It also provides detailed information of the Standard 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)**
1.0.0	Daniele Cloralio	21/10/2021	First version.
1.1.0	Daniele Cloralio	24/02/2022	Added section 3.6 Minor changes in other sections
1.2.0	Daniele Cloralio	16/03/2022	Updates for RTA-FBL 1.0.0

1.2 Definition and Abbreviations

Term/Abbreviation	**Definition**
ADC	Analogue to Digital Convertor
AR	AUTOSAR
Application Software (Application Software)	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
Dcm	Diagnostic Communication Manager
DiD	Data iDentifier
RiD	Routine iDentifier
ECU	Electronic Control Unit
FBL	Flash Bootloader
Fee	Flash EEPROM Emulation
MCAL	Micro-Controller Abstraction Layer
NRC	Negative Response Code from the ECU
NvM	Non-Volatile Memory
OS	Operative System
RTA-x	The ETAS suite of embedded SW products
UDS	Unified Diagnostic Services

1.3 References

Ref.	Document Name	Ver.
[1]	ISO 14229-1	2013

1.4 About this Document

This document provides a detailed description of ETAS' RTA-FBL Standard Port. 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 the RTA-FBL Standard Port must be installed and used to allow you to create a complete RTA-FBL bootloader instance. It includes important steps required for integrating RTA-FBL with your Application Software.
Chapter 4	This chapter explains how to flash an ECU with an RTA-FBL bootloader using INCA.
Chapter 5	This chapter contains important privacy information.
Chapter 6	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 3.

2.1 What is a Flash Bootloader?

A Flash Bootloader (FBL) is embedded SW that allows the reprogramming of an ECU with new Application Software together with its calibration data 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 Software. The FBL updates the ECU's non-volatile memory with this new Application Software.

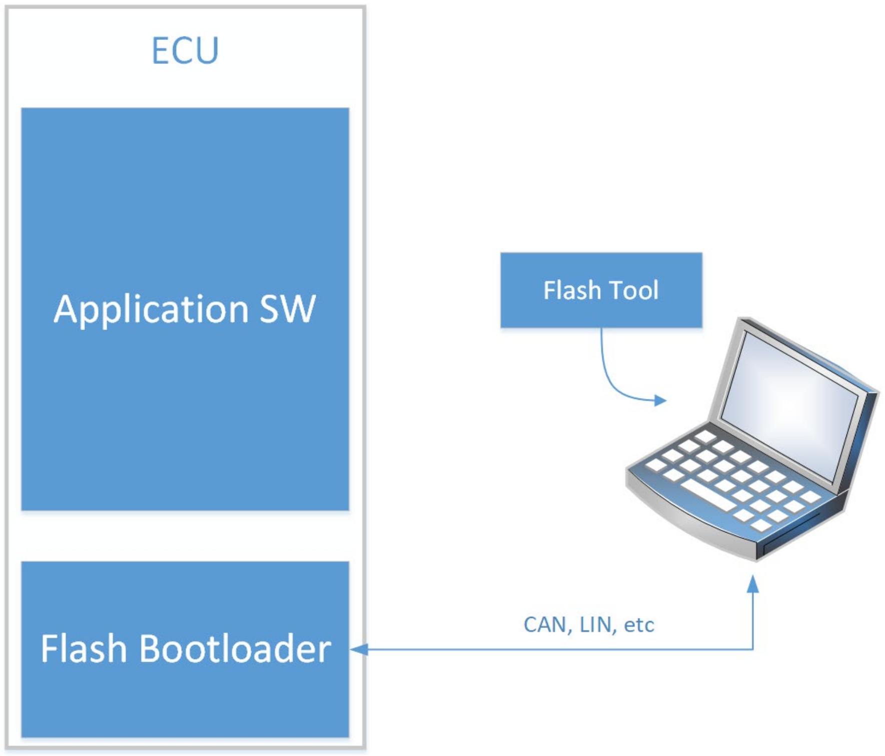


Figure 1: High level flashing process

The FBL is a standalone program. It has a separate run-time with respect to the Application Software, and so the FBL and the Application Software never run concurrently. After startup, the FBL always runs first as it needs to decide whether it is to wait for new Application Software to be sent from a tester, or if it is to start the Application Software 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 Software before the last reset, and whether the Application Software currently programmed in the ECU is valid.

A classic boot loading sequence showing this decision is depicted in Figure 2. Note that the Application Software is only started if the Application Software 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.

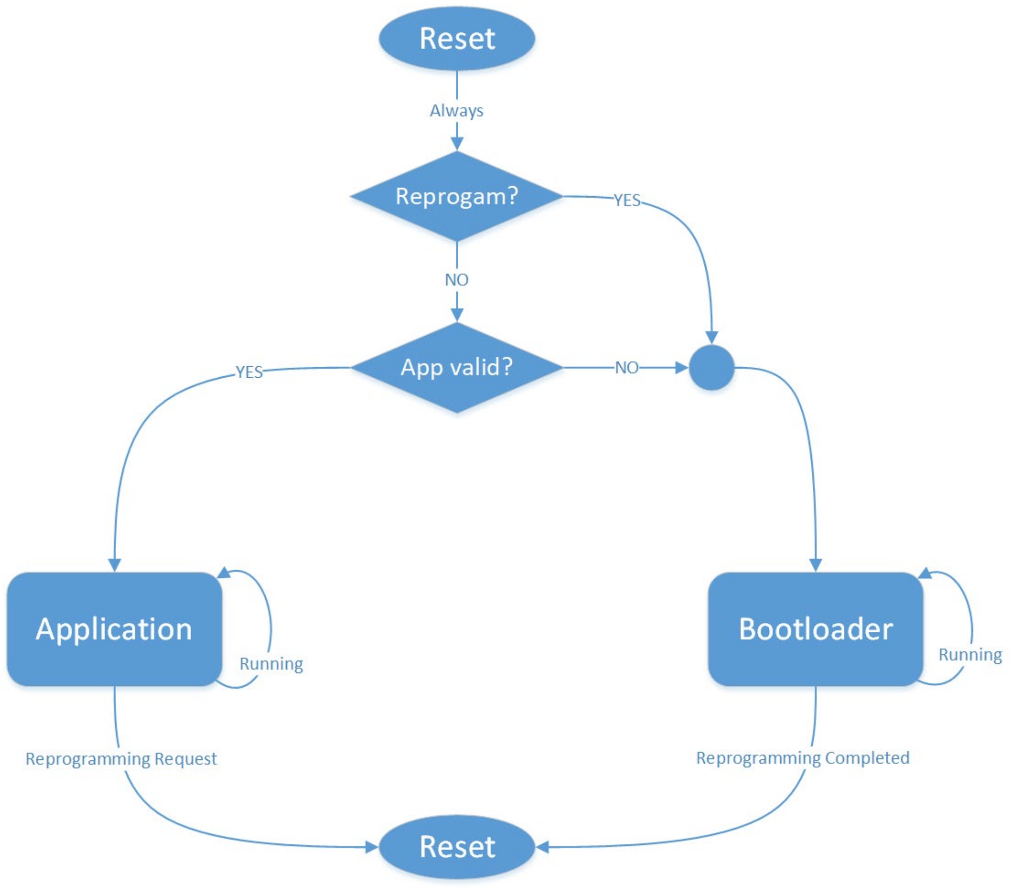


Figure 2: Boot loading flowchart

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 and MCAL configuration) 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)

Basing the underlying SW architecture on AUTOSAR allows support of other communication protocols such as CAN-FD, 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, BSW and MCAL configuration 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. For example, some ports of ETAS' FBL support 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 [3]. Each port supports one or more hardware "targets".

2.5 Target Dependencies and the Flash Driver

An FBL will necessarily 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 3.

2.7 One and Two-Stage Bootloaders

There are two broad models for bootloaders and the model type for the bootloader described in [3] is described in more detail in Section 3.

- 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 principle 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 Updating the bootloader

A bootloader specification might require that the bootloader be able to update itself. The way that this is done may also be prescribed by that specification, or the specification may allow the implementer to devise a proprietary solution. Bootloader update usually includes downloading a "Bootloader Updater" in place of the application, which then updates the main bootloader. Integrity of the ECU must be maintained so that a failure during bootloader update does not result in bricking of the ECU. Support for bootloader update for your port (if any) is described in Section 3.

2.9 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 3.

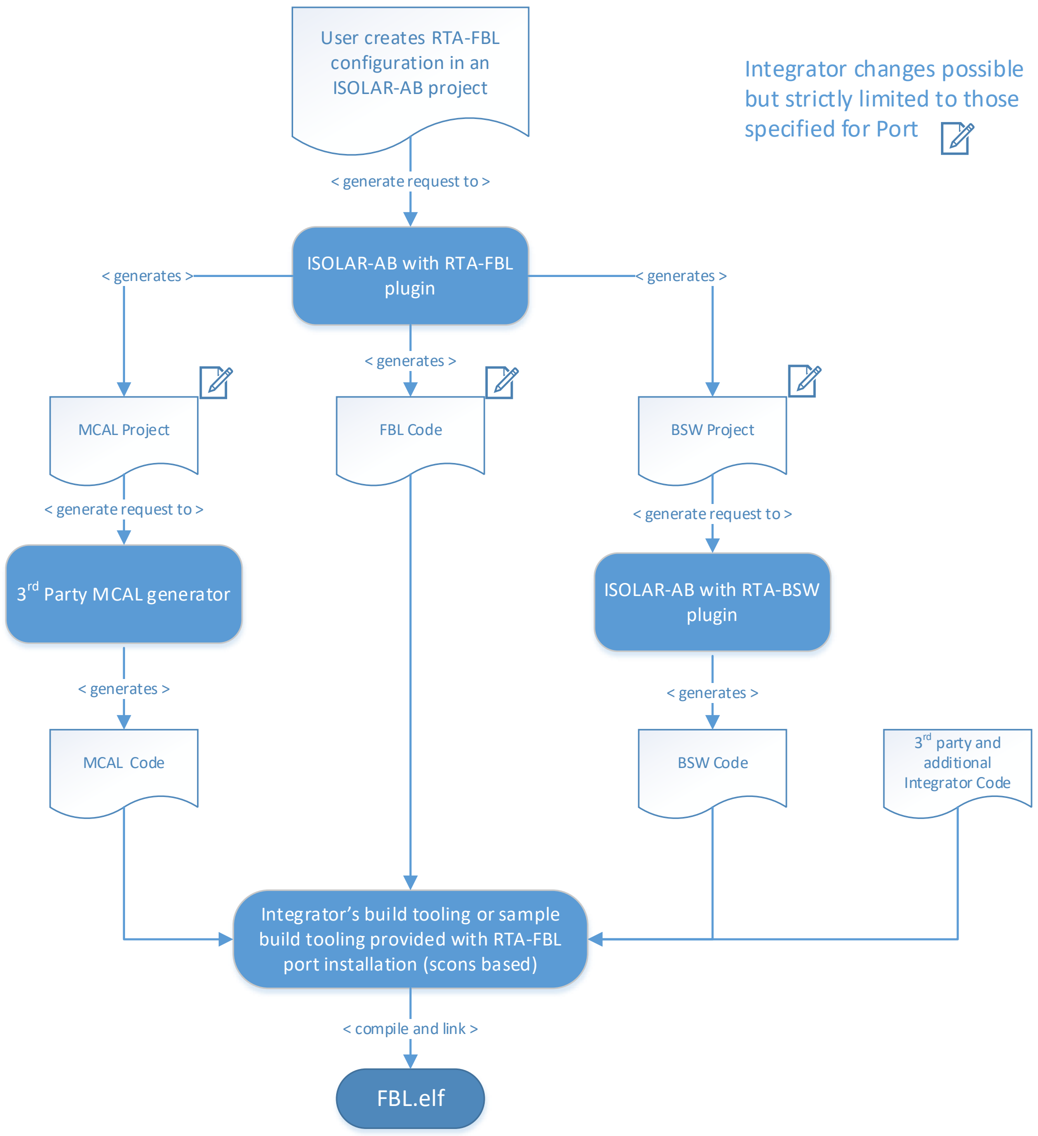

User creates RTA-FBL configuration in an ISOLAR-AB project
< generate request to >
ISOLAR-AB with RTA-FBL plugin
Integrator changes possible but strictly limited to those specified for Port
< generates >
< generates >
< generates >
MCAL Project
FBL Code
BSW Project
< generate request to >
< generate request to >
3rd Party MCAL generator
ISOLAR-AB with RTA-BSW plugin
< generates >
< generates >
MCAL Code
BSW Code
3rd party and additional Integrator Code
Integrator's build tooling or sample build tooling provided with RTA-FBL port installation (scons based)
< compile and link >
FBL.elf

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 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 3 for information relevant to your port.

2.10 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 3. 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 2.9, 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 3, including the folder structure of a generated RTA-FBL instance that contains the code for the modules in Figure 4.

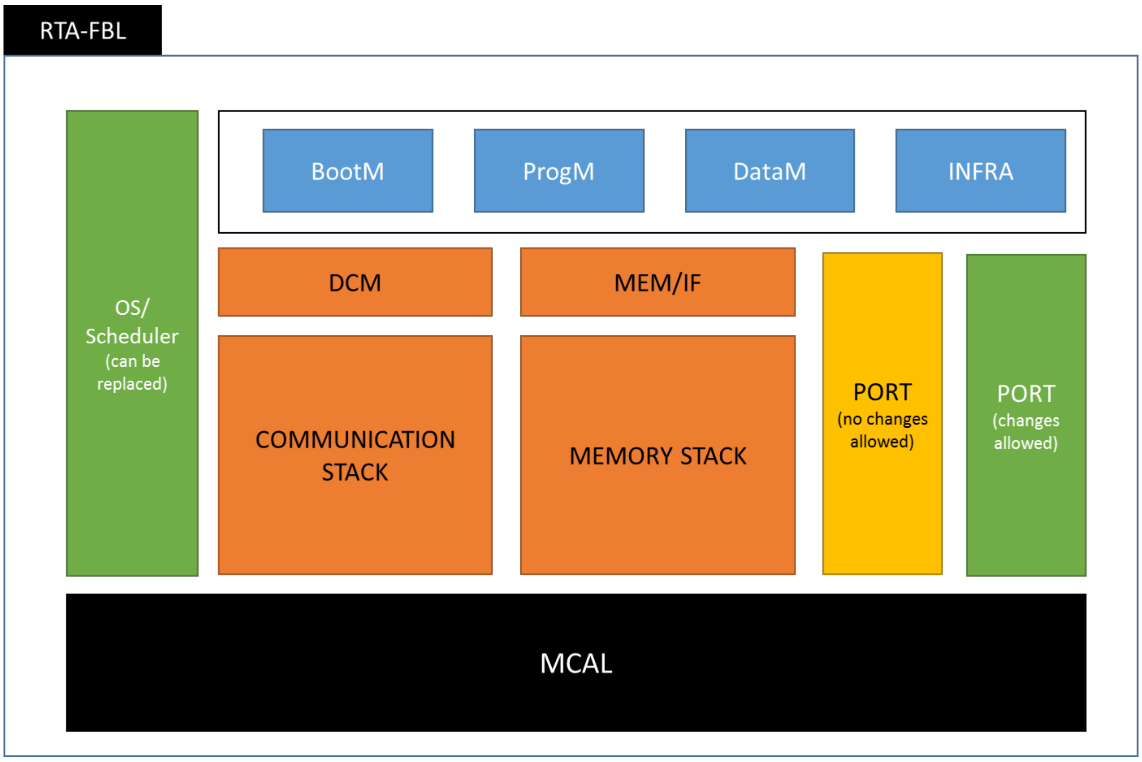


Figure 4: General architecture of an RTA-FBL instance

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

In order to generate an instance of RTA-FBL, you will need to install the tools shown in Table 1. 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 required by your target as described in your Standard FBL Target Guide.

Table 1: Tool versions

Tool Name	**Version**	**Description**
RTA-CAR	9.2	RTA-FBL configurator tool.
RTA-FBL Standard Port	1.0.0	FBL generator tool.
.NET framework	3.5	This is required by the ETAS license management. In most cases, you will already have this installed on your machine.

3 RTA-FBL Standard Port

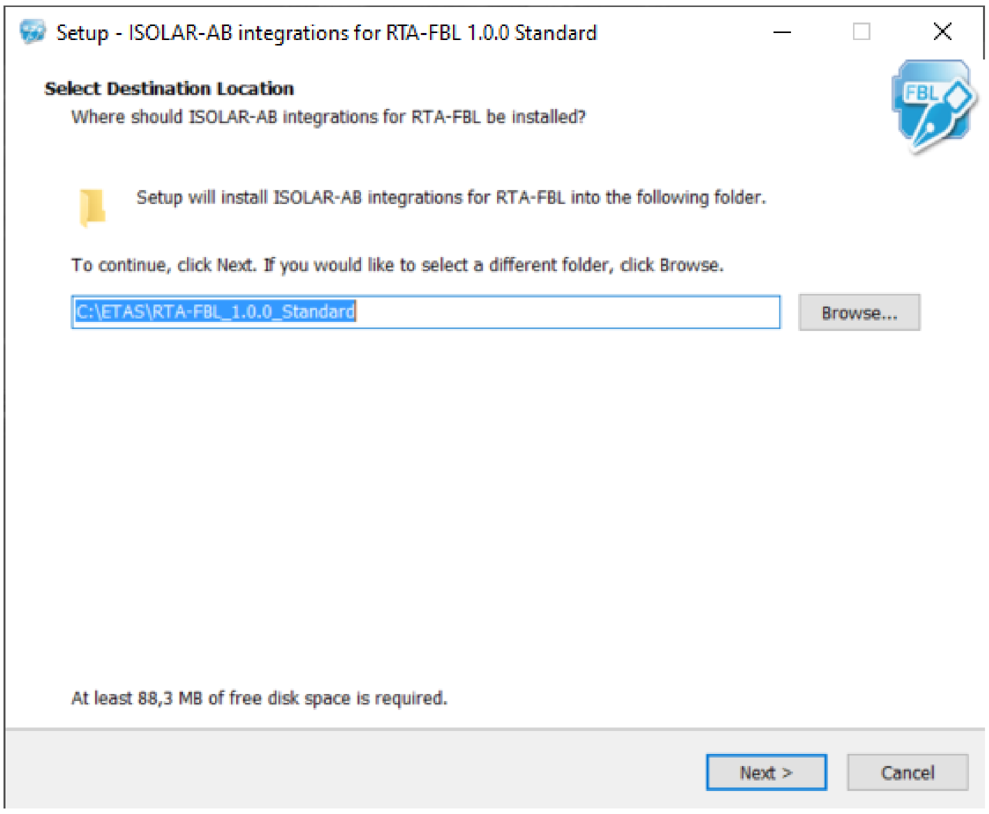
This chapter describes the Standard Port of RTA-FBL. It provides specific information relevant to this port that expands on the general RTA-FBL features described in Chapter 2. This chapter assumes that the reader is familiar with the ISO standard in [1] and common Flash Bootloader functionalities described in 2.

3.1 Installation

This section describes the installer for the Standard port of RTA-FBL. As noted in Section 2.11, 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 RTA_FBL_v1.0.0_Standard.exe. When the welcome window is displayed, select the desired installation folder by typing the desired location or by clicking "Browse". Then click "Next".


Setup - ISOLAR-AB integrations for RTA-FBL 1.0.0 Standard

Select Destination Location

Where should ISOLAR-AB integrations for RTA-FBL be installed?

Setup will install ISOLAR-AB integrations for RTA-FBL into the following folder.

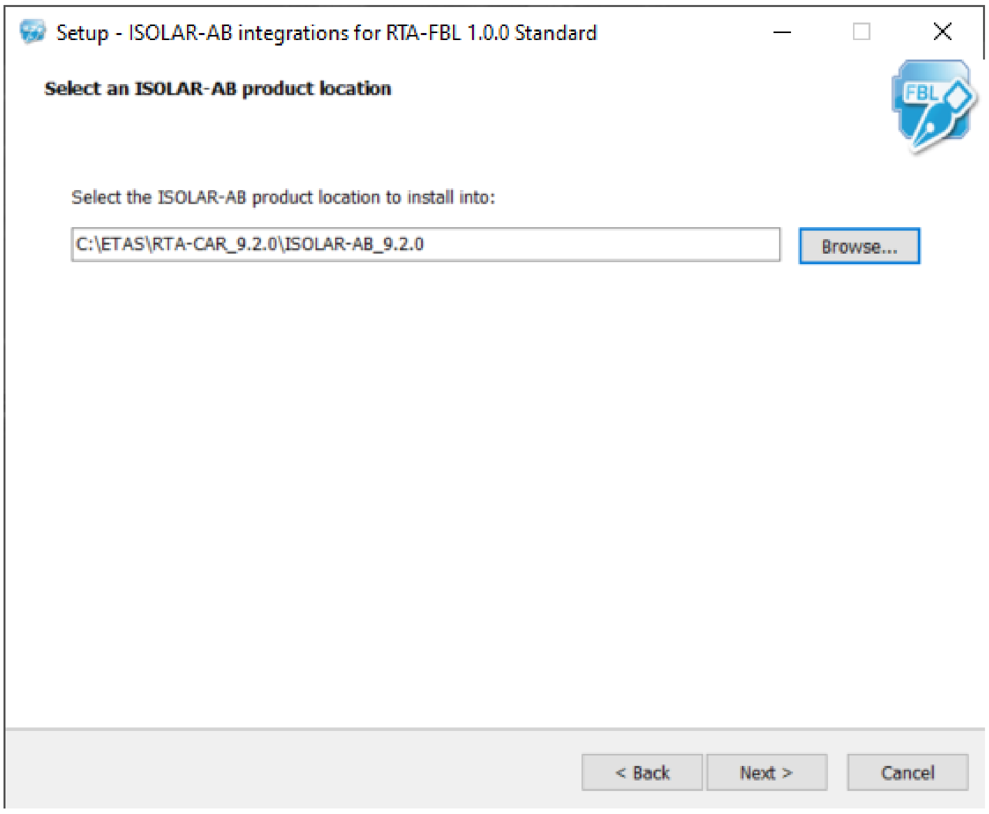
To continue, click Next. If you would like to select a different folder, click Browse.

C:\ETAS\RTA-FBL_1.0.0_Standard

At least 88,3 MB of free disk space is required.

Figure 5: Welcome window

Step 2: Select the ISOLAR-AB version that will support the plugin by using "Browse". The minimum required version is 9.2 Then click "Next".



Setup - ISOLAR-AB integrations for RTA-FBL 1.0.0 Standard

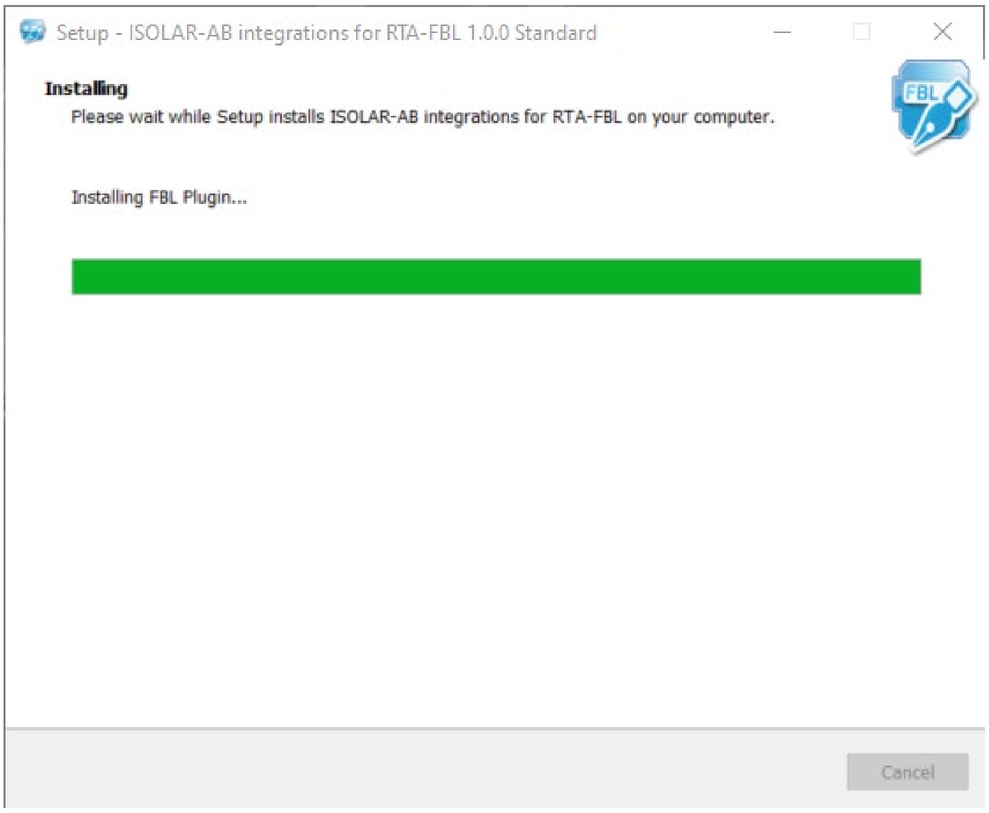
Select an ISOLAR-AB product location

Select the ISOLAR-AB product location to install into:

C:\ETAS\RTA-CAR_9.2.0\ISOLAR-AB_9.2.0

Figure 6: ISOLAR-AB version selection

Step 3: Wait until the installation is complete.


Setup - ISOLAR-AB integrations for RTA-FBL 1.0.0 Standard

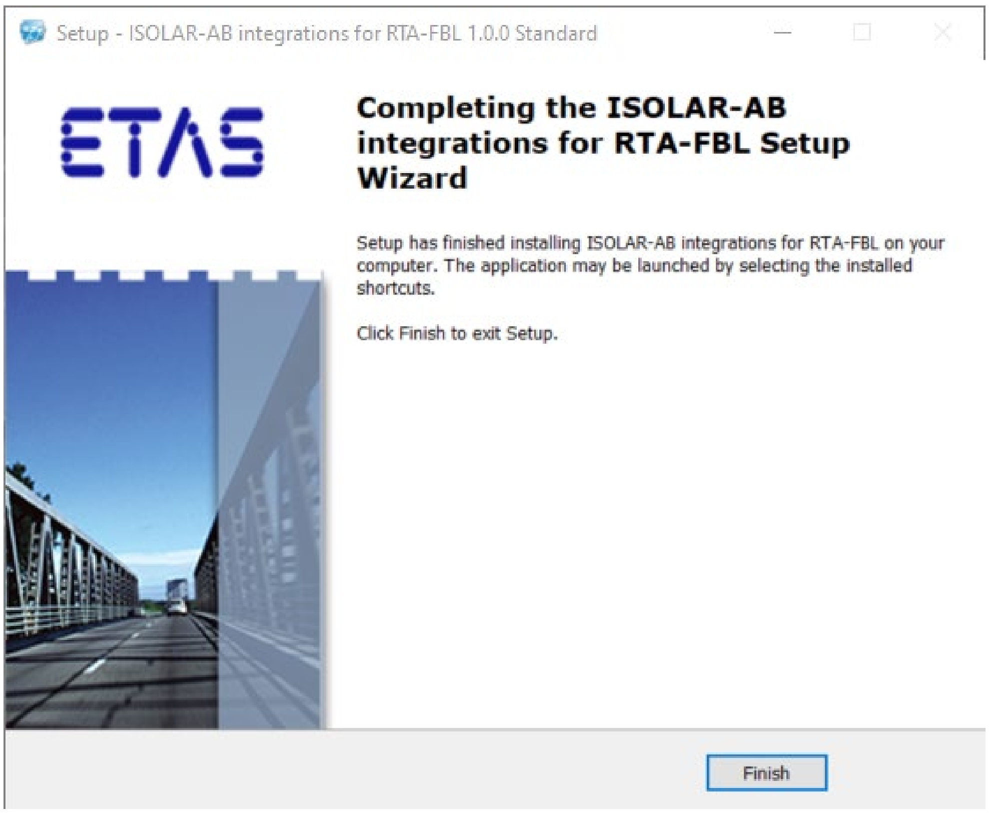
Installing

Please wait while Setup installs ISOLAR-AB integrations for RTA-FBL on your computer.

Installing FBL Plugin...

Figure 7: Installation progress

Step 4: After the installation is completed, click on "Finish" to close the installer.


Setup - ISOLAR-AB integrations for RTA-FBL 1.0.0 Standard

Completing the ISOLAR-AB integrations for RTA-FBL Setup Wizard

Setup has finished installing ISOLAR-AB integrations for RTA-FBL on your computer. The application may be launched by selecting the installed shortcuts.

Click Finish to exit Setup.

Figure 8: Installed Components

3.2 RTA-FBL Standard Architecture

Figure 9 provides a high-level view of RTA-FBL architecture for Standard Port. 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. The modules that comprise the RTA-FBL instance for this 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 Standard port modules (in yellow); these are generated by the RTA-FBL ISOLAR-AB plugin and must not be modified by the integrator. The Port module implements the bootloader features that are described in 3.3 whereas the ECL is the ETAS Crypto library used for signature calculation.
4. The sample modules (in green); these are generated by the RTA-FBL ISOLAR-AB 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 3.5.8 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 interacts with

the core FBL modules. See Section 3.5.9 for further details on how to adapt this module.

5. The MCAL modules and the Port-Target interface module (in black); the modules shown are those required by the Standard 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. The Standard port ships with a dummy target that contains no MCAL. Please see your target user guide for a sample MCAL project that has been tested with the full bootloader stack.

The full list of files created during generation is described in 3.5.3

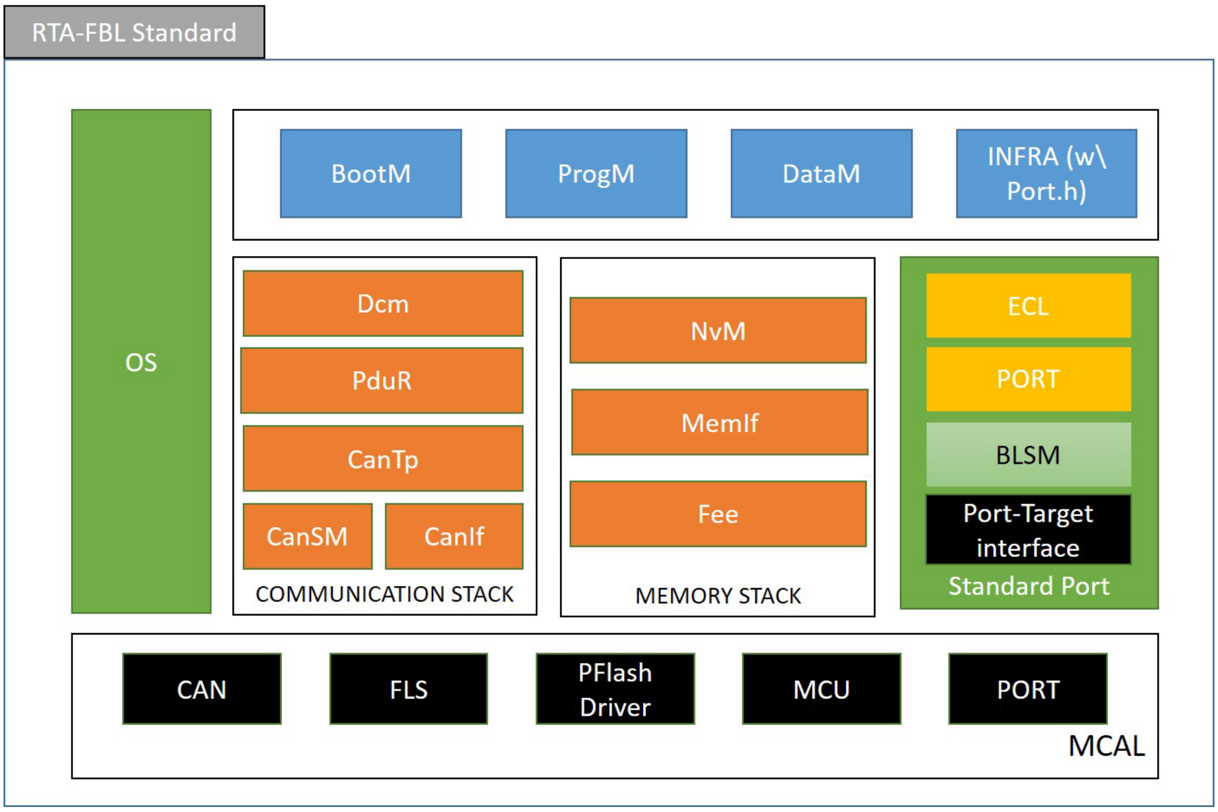


Figure 9: Architecture of an RTA-FBL Standard instance

3.3 Supported services

The following services and subservices as described in [1] are supported:

Service	Subfunction	Important config	Comments
0x10 – DiagnosticSessionControl	01	P2ServerMax is set to 0.05 seconds and P2StarServerMax is set to 5 seconds.	Default Session according to [1]
	02		Programming Session according to [1]

0x11 – ECU Reset	01	N/A	Used to reset the Fbl according to [1]
0x22 – ReadDataByIdentifier	N/A	0x0100	Used to read Fbl Programming Counter, for details refer to 3.3.1
		0x0101	Used to read Application Programming Counter, for details refer to 3.3.1
		0x0102	Used to read Application Data Programming Counter, for details refer to 3.3.1
		0xF180	Used to read Boot Software Identification, for details refer to 3.3.1 and [1]
		0xF181	Used to read Application Software Identification, for details refer to 3.3.1 and [1]
		0xF182	Used to read Application Data Software Identification, for details refer to 3.3.1 and [1]
		0xF183	Used to read Boot Software Fingerprint, for details refer to 3.3.1 and [1]
		0xF184	Used to read Application Software Fingerprint, for details refer to 3.3.1 and [1]
		0xF185	Used to read Application Data Software Fingerprint, for details refer to 3.3.1 and [1]
		0xF18C	Used to read ECU Serial Number, for details refer to 3.3.1 and [1]
0x2E - WriteDataByIdentifier	N/A	0xF183 – available in Programming Session in Unlocked state	Used to write Boot Software Fingerprint, for details refer to 3.3.1 and [1]
		0xF184– available in Programming Session in	Used to write Application Software

		Unlocked state	Fingerprint, for details refer to 3.3.1 and [1]
		0xF185 – available in Programming Session in Unlocked state	Used to write Application Data Software Fingerprint, for details refer to 3.3.1 and [1]
0x27 – SecurityAccess	01	Available in Programming Session, 10.0 seconds delay time after 3 consecutive wrong attempts	requestSeed for ECU Unlock according to [1]
	02		sendKey for ECU Unlock according to [1]. Key is computed as follow: `Key = Seed ^ FblSeedKeyConstant1 ^ FblSeedKeyConstant1`
0x28 – CommunicationControl	00	N/A	enableRxAndTx: Configured, but it has no effect while in boot mode.
	03	N/A	disableRxAndTx: Configured, but it has no effect while in boot mode.
0x31 – RoutineControl	01	0xFF00 - available in Programming Session in Unlocked state	Erase memory region, for details refer to 3.3.2 and [1]
	01	0xF000 - available in Programming Session in Unlocked state	Verify memory region, for details refer to 3.3.2 and [1]
	01	0xFF01 - available in Programming Session in Unlocked state	Check programming dependencies, for details refer to 3.3.2 and [1]
0x34 – RequestDownload	N/A	Available in Programming Session in Unlocked state	Request for download according to [1]. Supported dataFormatIdentifier is 00 hex. Supported addressAndLengthFormatIdentifier is 44 hex.
0x36 – TransferData	N/A	Available in Programming Session in Unlocked state	Transfer of data according to [1]
0x37 – RequestTransferExit	N/A	Available in Programming Session in Unlocked state	Complete data transfer according to [1]. transferRequestParamet

			erRecord not required.
0x3E – TesterPresent	00	N/A	Tester present according to [1]
0x85 – ControlDTCSettings	01	N/A	Set On: Configured, but DTC is not supported while in boot mode.
	02	N/A	Set Off: Configured, but DTC is not supported while in boot mode.

3.3.1 DIDs

Bootloader Programming Counter – 0x0100

The Bootloader Programming Counter tracks the number of reprogramming times. An FBL which has never been reprogrammed by a diagnostic tool will report a programming counter of zero. The programming counter is incremented after the new downloaded software is considered valid with a positive response to RID 0xF000 - Verify memory region.

Upon requesting data identifier 0x0100 with the diagnostic service Read Data By Identifier, the FBL will return the Boot Software Programming Counter data record, matching following table:

Byte	Parameter Name / Description	Byte Value
#1	ReadDataByIdentifier Response SID	0x62
#2	dataIdentifier High Byte	0x01
#3	dataIdentifier Low Byte	0x00
#4	Bootloader Identification Region	0x00
#5	Number of Bootloader Programming Counter High Byte	0x00-0xFF
#6	Number of Bootloader Programming Counter Low Byte	0x00-0xFF

Application Programming Counter – 0x0101

The Application Programming Counter tracks the number of application regions reprogramming times. An FBL whose Application blocks have never been reprogrammed by a diagnostic tool will report a programming counter of zero. The programming counter is incremented after the new downloaded software is considered valid with a positive response to RID 0xF000 - Verify memory region.

Upon requesting data identifier 0x0101 with the diagnostic service Read Data By Identifier, the FBL will return the Application Software Programming Counter data record, matching following table:

Byte	Parameter Name / Description	Byte Value
#1	ReadDataByIdentifier Response SID	0x62
#2	dataIdentifier High Byte	0x01

#3	dataIdentifier Low Byte	0x01
#4	Application Identification Region – Block #1	0x00-0xFF
#5	Number of Application Programming Counter High Byte – Block #1	0x00-0xFF
#6	Number of Application Programming Counter Low Byte – Block #1	0x00-0xFF
..	..	..
n·3+1	Application Identification Region – Block #n(*)	0x00-0xFF
n·3+2	Number of Application Programming Counter High Byte – Block #n(*)	0x00-0xFF
n·3+3	Number of Application Programming Counter Low Byte – Block #n(*)	0x00-0xFF

(*) n: Number of configured application regions

Application Data Programming Counter – 0x0102

The Application Data Programming Counter tracks the number of calibration regions reprogramming times. An FBL whose Calibration blocks have never been reprogrammed by a diagnostic tool will report a programming counter of zero. The programming counter is incremented after the new downloaded software is considered valid with a positive response to RID 0xF000 - Verify memory region.

Upon requesting data identifier 0x0102 with the diagnostic service Read Data By Identifier, the FBL will return the Application Data Programming Counter data record, matching following table:

Byte	Parameter Name / Description	Byte Value
#1	ReadDataByIdentifier Response SID	0x62
#2	dataIdentifier High Byte	0x01
#3	dataIdentifier Low Byte	0x02
#4	Application Data Identification Region – Block #1	0x00-0xFF
#5	Number of Application Data Programming Counter High Byte – Block #1	0x00-0xFF
#6	Number of Application Data Programming Counter Low Byte – Block #1	0x00-0xFF
..	..	..
n·3+1	Application Data Identification Region – Block #n(*)	0x00-0xFF
n·3+2	Number of Application Data Programming Counter High Byte – Block #n(*)	0x00-0xFF
n·3+3	Number of Application Data Programming Counter Low Byte – Block #n(*)	0x00-0xFF

(*) n: Number of configured calibration regions

Application Programming Status – 0x0103

The Application Programming Status reports the reprogramming status of each application region.

Upon requesting data identifier 0x0103 with the diagnostic service Read Data By Identifier, the FBL will return the Application Programming Status data record, matching following table:

Byte	Parameter Name / Description	Byte Value
#1	ReadDataByIdentifier Response SID	0x62
#2	dataIdentifier High Byte	0x01
#3	dataIdentifier Low Byte	0x03
#4	Application Identification Region – Block #1	0x00-0xFF
#5	Application Programming Status – Block #1	0x00 – Block in erased state or partially reprogrammed 0x10 – Signature verification failed by RID 0xF000 0x11 – Signature verification success by RID 0xF000 0x20 – CRC verification failed by RID 0xF000 0x21 – CRC verification success by RID 0xF000 0xF0 – Check dependencies failed because software deemed incompatible by the user 0xFF – Check dependencies success by RID 0xFF01
..	..	..
n·2+2	Application Identification Region – Block #n(*)	0x00-0xFF
n·2+3	Application Programming Status – Block #n(*)	0x00 – Block in erased state or partially reprogrammed 0x10 – Signature verification failed by RID 0xF000 0x11 – Signature verification success by RID 0xF000 0x20 – CRC verification failed by RID 0xF000 0x21 – CRC verification success by RID 0xF000 0xF0 – Check dependencies failed because software deemed incompatible by the user 0xFF – Check dependencies success by RID 0xFF01

(*) n: Number of configured application regions

Application Data Programming Status – 0x0104

The Application Data Programming Status reports the reprogramming status of each calibration region.

Upon requesting data identifier 0x0104 with the diagnostic service Read Data By Identifier, the FBL will return the Application Data Programming Status data record, matching following table:

Byte	Parameter Name / Description	Byte Value
#1	ReadDataByIdentifier Response SID	0x62
#2	dataIdentifier High Byte	0x01
#3	dataIdentifier Low Byte	0x04
#4	Application Data Identification Region – Block #1	0x00-0xFF
#5	Application Data Programming Status – Block #1	0x00 – Block in erased state or partially reprogrammed 0x10 - Signature verification failed by RID 0xF000 0x11 - Signature verification success by RID 0xF000 0x20 – CRC verification failed by RID 0xF000 0x21 – CRC verification success by RID 0xF000 0xF0 – Check dependencies failed because software deemed incompatible by the user 0xFF – Check dependencies success by RID 0xFF01
..	..	..
n·2+2	Application Data Identification Region – Block #n(*)	0x00-0xFF
n·2+3	Application Data Programming Status – Block #n(*)	0x00 – Block in erased state or partially reprogrammed 0x10 - Signature verification failed by RID 0xF000 0x11 - Signature verification success by RID 0xF000 0x20 – CRC verification failed by RID 0xF000 0x21 – CRC verification success by RID 0xF000 0xF0 – Check dependencies failed because software deemed incompatible by the user 0xFF – Check dependencies success by RID 0xFF01

(*) n: Number of configured calibration regions

Bootloader Software Identification – 0xF180

DID 0xF180 is used to reference the vehicle manufacturer specific ECU Bootloader Software Identification record, according to [1].

Upon requesting data identifier 0xF180 with the diagnostic service Read Data By Identifier, the FBL will return the Bootloader Software Identification data record, matching following table:

Byte	Parameter Name / Description	Byte Value
#1	ReadDataByIdentifier Response SID	0x62
#2	dataIdentifier High Byte	0xF1
#3	dataIdentifier Low Byte	0x80
#4	Bootloader Software Identification – Number of Bootloader regions	0x01
#5	Bootloader Software Identification – Byte #1	0x00-0xFF
#6	Bootloader Software Identification – Byte #2	0x00-0xFF
#7	Bootloader Software Identification – Byte #3	0x00-0xFF

Application Software Identification – 0xF181

DID 0xF181 is used to reference the vehicle manufacturer specific ECU Application Software Number(s), according to [1].

Upon requesting data identifier 0xF181 with the diagnostic service Read Data By Identifier, the FBL will return the Application Software Identification data record, matching following table:

Byte	Parameter Name / Description	Byte Value
#1	ReadDataByIdentifier Response SID	0x62
#2	dataIdentifier High Byte	0xF1
#3	dataIdentifier Low Byte	0x81
#4	Application Software Identification – Number of Application regions(*)	0x00-0xFF
#5	Application Software Identification – Byte #1	0x00-0xFF
#6	Application Software Identification – Byte #2	0x00-0xFF
#7	Application Software Identification – Byte #3	0x00-0xFF
#8	Application Software Identification – Byte #4	0x00-0xFF
#9	Application Software Identification – Byte #5	0x00-0xFF

(*) Number of application regions configured within FblRegions

Application Data Software Identification - 0xF182

DID 0xF182 is used to reference the vehicle manufacturer specific ECU Application Data Identification record, according to [1].

Upon requesting data identifier 0xF182 with the diagnostic service Read Data By Identifier, the FBL will return the Application Data Software Identification data record, matching following table:

Byte	Parameter Name / Description	Byte Value
#1	ReadDataByIdentifier Response SID	0x62

#2	dataIdentifier High Byte	0xF1
#3	dataIdentifier Low Byte	0x82
#4	Application Data Software Identification – Number of Calibration regions(*)	0x00-0xFF
#5	Application Data Software Identification – Byte #1	0x00-0xFF
#6	Application Data Software Identification – Byte #2	0x00-0xFF
#7	Application Data Software Identification – Byte #3	0x00-0xFF
#8	Application Data Software Identification – Byte #4	0x00-0xFF
#9	Application Data Software Identification – Byte #5	0x00-0xFF

(*) Number of calibration regions configured within FblRegions

Bootloader Software Fingerprint – 0xF183

DID 0xF183 is used to reference the vehicle manufacturer specific ECU Bootloader Software Fingerprint identification record, according to [1]. It can be used to store relevant information (e.g. tester identification or reprogramming date) during the bootloader reprogramming process.
Upon requesting data identifier 0xF183 with the diagnostic service Read Data By Identifier, the FBL will return the Bootloader Software Fingerprint data record, matching following table:

Byte	Parameter Name / Description	Byte Value
#1	ReadDataByIdentifier Response SID	0x62
#2	dataIdentifier High Byte	0xF1
#3	dataIdentifier Low Byte	0x83
#4	Bootloader Identification Region	0x00
#5	Bootloader Software Fingerprint – Byte #1	0x00-0xFF
#6	Bootloader Software Fingerprint – Byte #2	0x00-0xFF
#7	Bootloader Software Fingerprint – Byte #3	0x00-0xFF
#8	Bootloader Software Fingerprint – Byte #4	0x00-0xFF
#9	Bootloader Software Fingerprint – Byte #5	0x00-0xFF

The Bootloader Software Fingerprint data record can be written into FBL's NvM, upon requesting data identifier 0xF183 with the diagnostic service Write Data By Identifier, matching following table:

Byte	Parameter Name / Description	Byte Value
#1	WriteDataByIdentifier Request SID	0x2E
#2	dataIdentifier High Byte	0xF1
#3	dataIdentifier Low Byte	0x83
#4	Bootloader Identification Region	0x00
#5	Bootloader Software Fingerprint – Byte #1	0x00-0xFF
#6	Bootloader Software Fingerprint – Byte #2	0x00-0xFF
#7	Bootloader Software Fingerprint – Byte #3	0x00-0xFF
#8	Bootloader Software Fingerprint – Byte #4	0x00-0xFF
#9	Bootloader Software Fingerprint – Byte #5	0x00-0xFF

Application Software Fingerprint – 0xF184

DID 0xF184 is used to reference the vehicle manufacturer specific ECU Application Software Fingerprint identification record, according to [1]. It can be used to store relevant information (e.g. tester identification or reprogramming date) during the application reprogramming process.
Upon requesting data identifier 0xF184 with the diagnostic service Read Data By Identifier, the FBL will return the Application Software Fingerprint data record, matching following table:

Byte	Parameter Name / Description	Byte Value
#1	ReadDataByIdentifier Response SID	0x62
#2	dataIdentifier High Byte	0xF1
#3	dataIdentifier Low Byte	0x84
#4	Application Software Identification Region – Block #1	0x00-0xFF
#5	Application Fingerprint (Byte #1) – Block #1	0x00-0xFF
#6	Application Fingerprint (Byte #1) – Block #2	0x00-0xFF
#7	Application Fingerprint (Byte #1) – Block #3	0x00-0xFF
#8	Application Fingerprint (Byte #1) – Block #4	0x00-0xFF
#9	Application Fingerprint (Byte #1) – Block #5	0x00-0xFF
..	..	..
(n-1)·6+4	Application Identification Region – Block #n(*)	0x00-0xFF
(n-1)·6+5	Application Fingerprint (Byte #1) – Block #n(*)	0x00-0xFF
(n-1)·6+6	Application Fingerprint (Byte #2) – Block #n(*)	0x00-0xFF
(n-1)·6+7	Application Fingerprint (Byte #3) – Block #n(*)	0x00-0xFF
(n-1)·6+8	Application Fingerprint (Byte #4)	0x00-0xFF

	– Block #n(*)	
(n-1)·6+9	Application Fingerprint (Byte #5) – Block #n(*)	0x00-0xFF

(*) n: Number of application regions configured within FblRegions

The Application Software Fingerprint data record can be written into FBL's NvM, upon requesting data identifier 0xF184 with the diagnostic service Write Data By Identifier, matching following table:

Byte	Parameter Name / Description	Byte Value
#1	WriteDataByIdentifier Request SID	0x2E
#2	dataIdentifier High Byte	0xF1
#3	dataIdentifier Low Byte	0x84
#4	Application Identification Region	0x00-0xFF
#5	Application Fingerprint – Byte #1	0x00-0xFF
#6	Application Fingerprint – Byte #2	0x00-0xFF
#7	Application Fingerprint – Byte #3	0x00-0xFF
#8	Application Fingerprint – Byte #4	0x00-0xFF
#9	Application Fingerprint – Byte #5	0x00-0xFF

Application Data Software Fingerprint – 0xF185

DID 0xF185 is used to reference the vehicle manufacturer specific ECU Application Data Software Fingerprint identification record, according to [1]. It can be used to store relevant information (e.g. tester identification or reprogramming date) during the calibration reprogramming process.
Upon requesting data identifier 0xF185 with the diagnostic service Read Data By Identifier, the FBL will return the Application Software Fingerprint data record, matching following table:

Byte	Parameter Name / Description	Byte Value
#1	ReadDataByIdentifier Response SID	0x62
#2	dataIdentifier High Byte	0xF1
#3	dataIdentifier Low Byte	0x85
#4	Application Data Software Identification Region – Block #1	0x00-0xFF
#5	Data Fingerprint (Byte #1) – Block #1	0x00-0xFF
#6	Data Fingerprint (Byte #1) – Block #2	0x00-0xFF
#7	Data Fingerprint (Byte #1) – Block #3	0x00-0xFF
#8	Data Fingerprint (Byte #1) – Block #4	0x00-0xFF
#9	Data Fingerprint (Byte #1) – Block #5	0x00-0xFF
..	..	..
(n-1)·6+4	Application Data Software Identification Region – Block #n(*)	0x00-0xFF
(n-1)·6+5	Data Fingerprint (Byte #1) – Block #n(*)	0x00-0xFF
(n-1)·6+6	Data Fingerprint (Byte #2) – Block #n(*)	0x00-0xFF

(n-1)·6+7	Data Fingerprint (Byte #3) – Block #n(*)	0x00-0xFF
(n-1)·6+8	Data Fingerprint (Byte #4) – Block #n(*)	0x00-0xFF
(n-1)·6+9	Data Fingerprint (Byte #5) – Block #n(*)	0x00-0xFF

(*) n: Number of calibration regions configured within FblRegions

The Application Data Software Fingerprint data record can be written into FBL's NvM, upon requesting data identifier 0xF185 with the diagnostic service Write Data By Identifier, matching following table:

Byte	Parameter Name / Description	Byte Value
#1	WriteDataByIdentifier Request SID	0x2E
#2	dataIdentifier High Byte	0xF1
#3	dataIdentifier Low Byte	0x85
#4	Application Data Identification Region	0x00-0xFF
#5	Data Fingerprint – Byte #1	0x00-0xFF
#6	Data Fingerprint – Byte #2	0x00-0xFF
#7	Data Fingerprint – Byte #3	0x00-0xFF
#8	Data Fingerprint – Byte #4	0x00-0xFF
#9	Data Fingerprint – Byte #5	0x00-0xFF

ECU Serial Number – 0xF18C

The DID 0xF18C is used to reference the ECU (server) serial number, according to [1].

Upon requesting data identifier 0xF18C with the diagnostic service Read Data By Identifier, the FBL will return the ECU Serial Number data record, matching following table:

Byte	Parameter Name / Description	Byte Value
#1	ReadDataByIdentifier Response SID	0x62
#2	dataIdentifier High Byte	0xF1
#3	dataIdentifier Low Byte	0x8C
#4	ECU Serial Number – Byte #1	0x00-0xFF
#5	ECU Serial Number – Byte #2	0x00-0xFF
#6	ECU Serial Number – Byte #3	0x00-0xFF

3.3.2 RIDs

Erase Memory – 0xFF00

RID 0xFF00 shall be used to start the FBL memory erase routine. The Control option is used to identify the block that should be erased and reprogrammed.
The Erase Memory Routine can be triggered upon requesting routine identifier 0xFF00 with the diagnostic service Routine Control, matching following table:

Byte	Parameter Name / Description	Byte Value
#1	RoutineControl Request SID	0x31
#2	sub-function	0x01
#3	routineIdentifier High Byte	0xFF
#4	routineIdentifier Low Byte	0x00
#5	Identification Region	0x00-0xFF

Verify Download – 0xF000

RID 0xF000 shall be used to start the FBL verification process routine for the last downloaded block. The Control option depends on the configuration parameter FblSoftwareVerificationType, described in 3.5.2

- If FblSoftwareVerificationType = SIGNATURE:
  The Verify Download Routine can be triggered upon requesting routine identifier 0xF000 with the diagnostic service Routine Control, matching following table:

Byte	Parameter Name / Description	Byte Value
#1	RoutineControl Request SID	0x31
#2	sub-function	0x01
#3	routineIdentifier High Byte	0xF0
#4	routineIdentifier Low Byte	0x00

The FBL will compute the software signature, using algorithm RSASSA-PKCS1-v1_5 with 2048 bit public key specified via the configuration parameter FblSecurityPublicKey.
The FBL will then compare the computed signature with one contained in the last 256 bytes of the software and return NRC 0x72 – General Programming Failure if the two do not match.

- If FblSoftwareVerificationType = CRC:
  The Verify Download Routine can be triggered upon requesting routine identifier 0xF000 with the diagnostic service Routine Control, matching following table:

Byte	Parameter Name / Description	Byte Value
#1	RoutineControl Request SID	0x31
#2	sub-function	0x01
#3	routineIdentifier High Byte	0xF0
#4	routineIdentifier Low Byte	0x00
#5	CRC Value – Byte #1	0x00-0xFF
#6	CRC Value – Byte #2	0x00-0xFF
#7	CRC Value – Byte #3	0x00-0xFF
#8	CRC Value – Byte #4	0x00-0xFF

The FBL will compute the software CRC, using CRC32H04C11DB7 polynomial and 0xFFFFFFFF as initial value.

The FBL will then compare the computed CRC with one received in the Control Option and return NRC 0x72 – General Programming Failure if the two do not match.

Check Programming Dependencies – 0xFF01

RID 0xFF01 shall be used to start the FBL Programming Dependencies process, according to [1].

The Programming Dependencies Routine can be triggered upon requesting routine identifier 0xF000 with the diagnostic service Routine Control, matching following table:

Byte	Parameter Name / Description	Byte Value
#1	RoutineControl Request SID	0x31
#2	sub-function	0x01
#3	routineIdentifier High Byte	0xFF
#4	routineIdentifier Low Byte	0x01

The FBL will verify that all mandatory regions have been programmed successfully, and trigger an additional user callout for further compatibility check.

If previous verification fails, the FBL will return NRC 0x72 – General Programming Failure.

3.4 Reprogramming Sequence

Reprogramming Sequence is based on Section 15 from [1], and described in Figure 10.

Orange step is optional, and can be executed any time while in boot mode if necessary.

For each service details, please refer to 3.3 and [1]

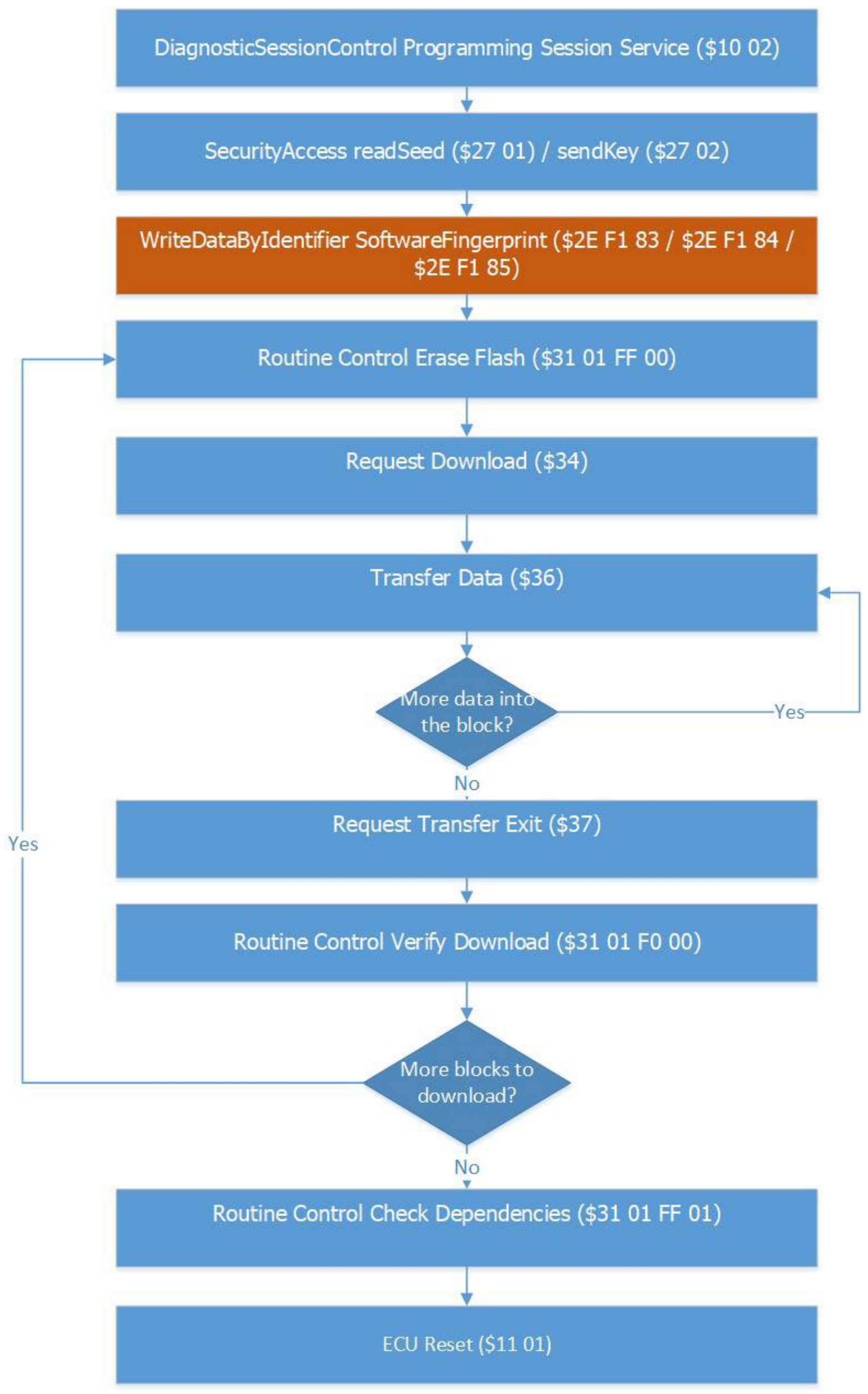

DiagnosticSessionControl Programming Session Service ($10 02)
SecurityAccess readSeed ($27 01) / sendKey ($27 02)
WriteDataByIdentifier SoftwareFingerprint ($2E F1 83 / $2E F1 84 / $2E F1 85)
Routine Control Erase Flash ($31 01 FF 00)
Request Download ($34)
Transfer Data ($36)
More data into the block?
Yes
No
Request Transfer Exit ($37)
Routine Control Verify Download ($31 01 F0 00)
More blocks to download?
Yes
No
Routine Control Check Dependencies ($31 01 FF 01)
ECU Reset ($11 01)

Figure 10: RTA-FBL Standard reprogramming sequence

3.5 Creating and building an RTA-FBL instance

This section explains how to create an ISOLAR-AB project to configure and generate an instance of Standard RTA-FBL port.

The tooling described in this section has been tested with Windows 10.

3.5.1 Project creation

A new FBL project is created in ISOLAR-AB. As shown in Figure 11, create a new RTA-CAR project by clicking the "New" dropdown button and selecting "RTA-CAR Project".

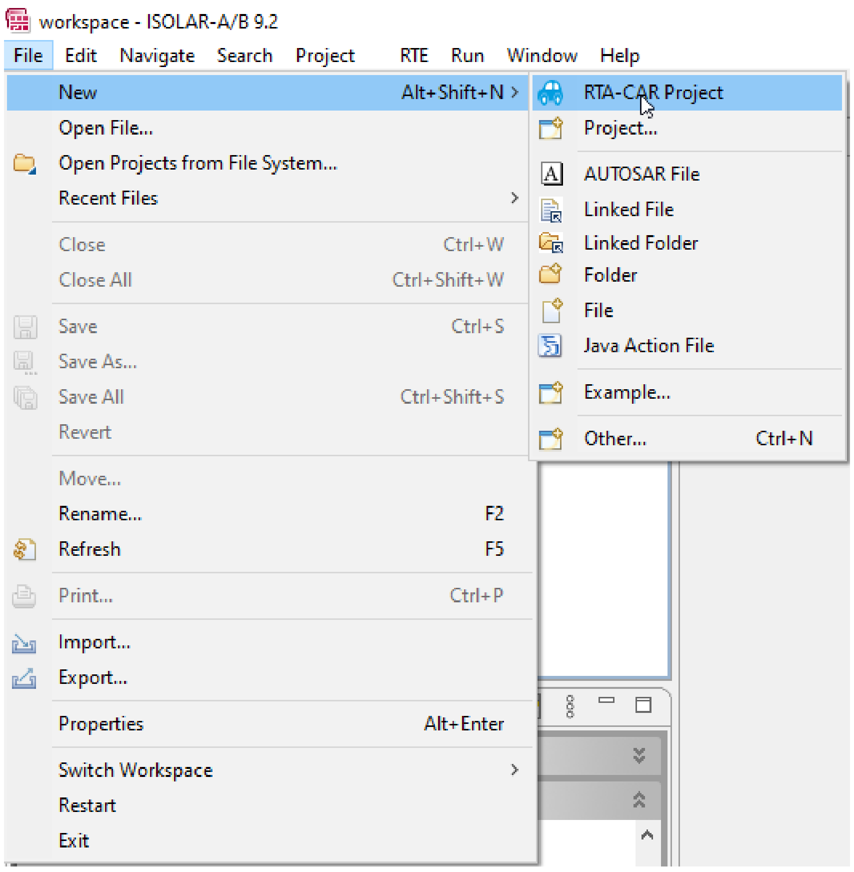
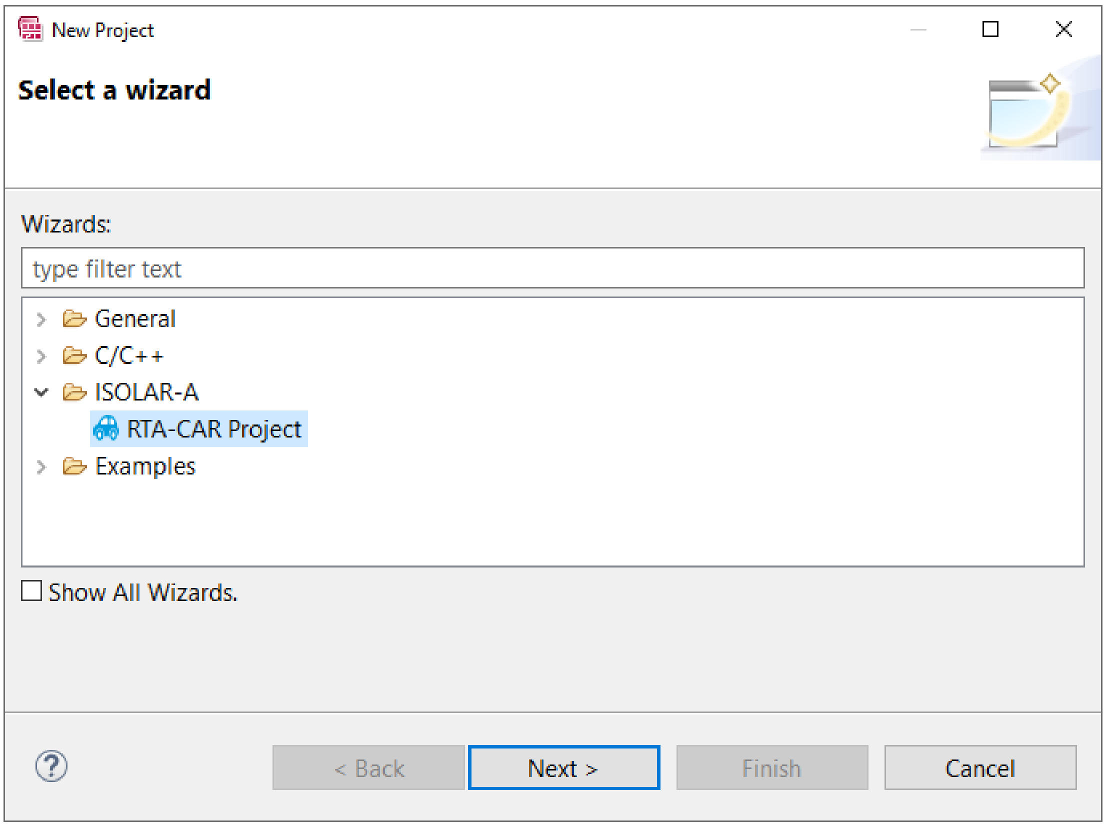


Figure 11: RTA-CAR project creation

If RTA-CAR Project is not present, select "Project…" and search for "RTA-CAR Project" in the new window, as shown in Figure 12.


New Project

Select a wizard

Wizards:

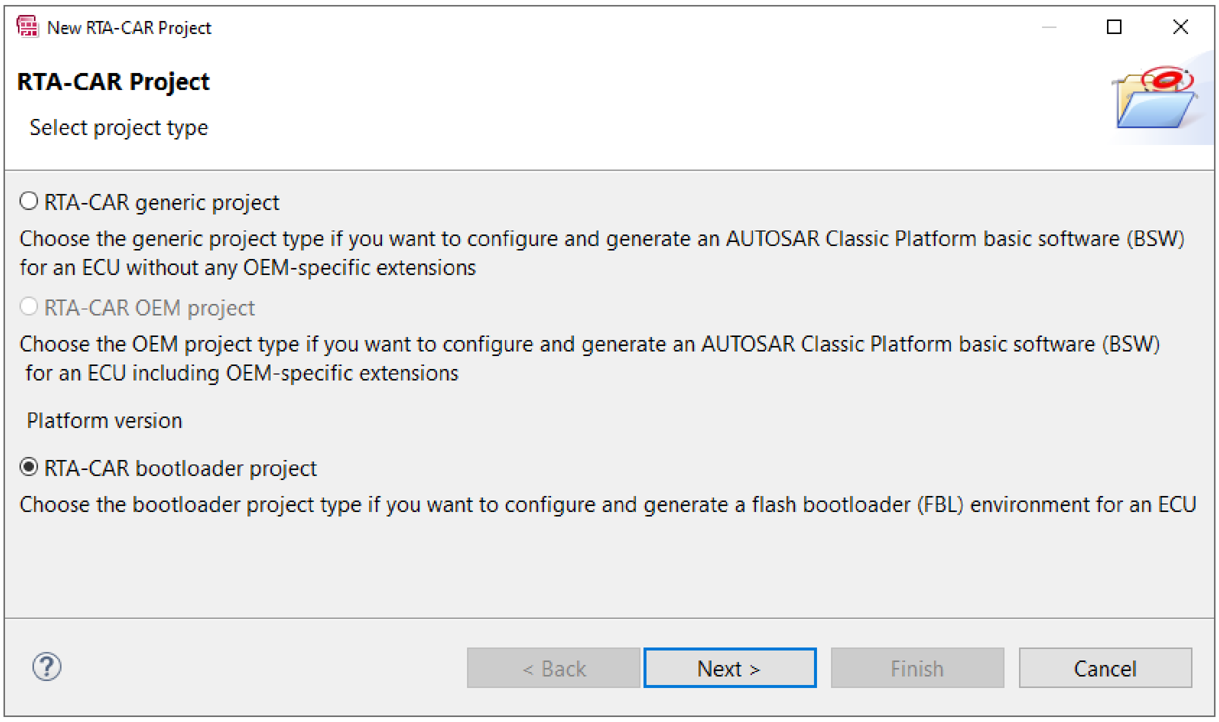
type filter text

- General
- C/C++
- ISOLAR-A
  - RTA-CAR Project
- Examples

☐ Show All Wizards.

Figure 12: RTA-CAR project

In the New RTA-CAR Project window, select RTA-CAR bootloader project as shown in Figure 13.


New RTA-CAR Project

RTA-CAR Project

Select project type

○ RTA-CAR generic project

Choose the generic project type if you want to configure and generate an AUTOSAR Classic Platform basic software (BSW) for an ECU without any OEM-specific extensions

○ RTA-CAR OEM project

Choose the OEM project type if you want to configure and generate an AUTOSAR Classic Platform basic software (BSW) for an ECU including OEM-specific extensions

Platform version

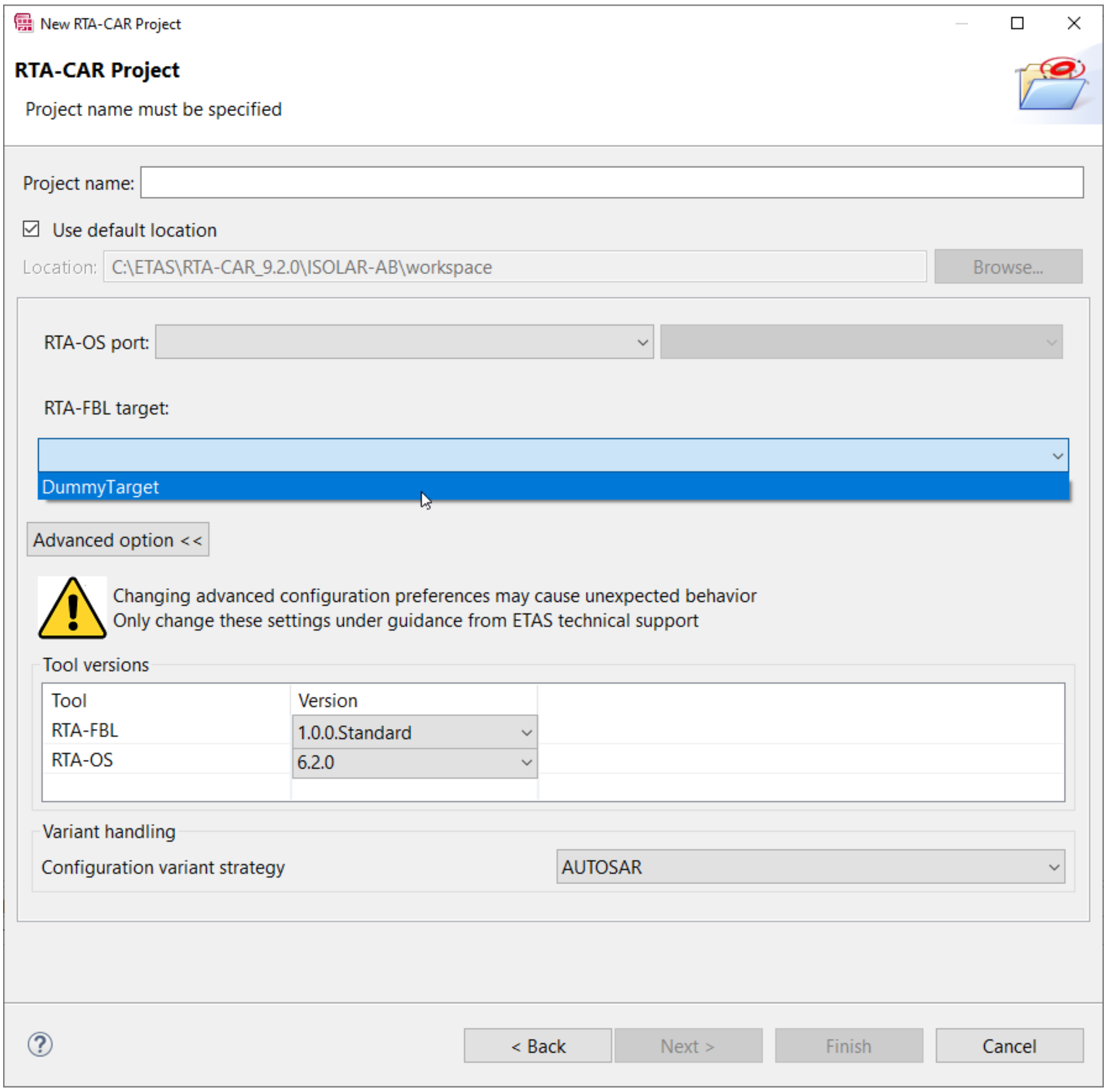
[✓ RTA-CAR bootloader project]

Choose the bootloader project type if you want to configure and generate a flash bootloader (FBL) environment for an ECU

Figure 13: New RTA-CAR Bootloader Project

Next, choose a name for your project and select the RTA-FBL target from the dropdown list as shown in Figure 14. Note that a target for RTA-OS port must also be selected for a proper project creation, despite the tool can be unused.

If you have multiple RTA-FBL tools installed, click on Advanced option and select 1.0.0 .Standard plugin under RTA FBL Tools.


New RTA-CAR Project

RTA-CAR Project

Project name must be specified

Project name:

☑ Use default location

Location: C:\ETAS\RTA-CAR_9.2.0\ISOLAR-AB\workspace

RTA-OS port:

RTA-FBL target: DummyTarget

Changing advanced configuration preferences may cause unexpected behavior
Only change these settings under guidance from ETAS technical support

Tool versions

Tool	Version
RTA-FBL	1.0.0.Standard
RTA-OS	6.2.0

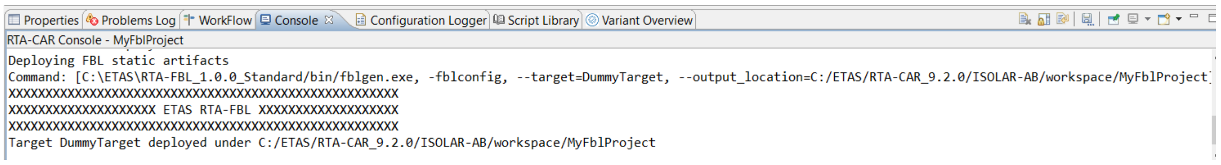
Variant handling

Configuration variant strategy: AUTOSAR

Figure 14: Select Target

Once complete, clicking the Finish button will result in the creation of the RTA-CAR bootloader project.

Figure 15 shows the result of a successful project creation in the console window.



```
RTA-CAR Console - MyFblProject
Deploying FBL static artifacts
Command: [C:\ETAS\RTA-FBL_1.0.0_Standard/bin/fblgen.exe, -fblconfig, --target=DummyTarget, --output_location=C:/ETAS/RTA-CAR_9.2.0/ISOLAR-AB/workspace/MyFblProject]
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXX ETAS RTA-FBL XXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Target DummyTarget deployed under C:/ETAS/RTA-CAR_9.2.0/ISOLAR-AB/workspace/MyFblProject
```

Figure 15: Console window upon successful project creation

3.5.2 Configuration and Generation of FBL and BSW

Next, complete the FBL configuration parameters. In the ECU Navigator view, right click on FBL under Bsw Modules and select Open With > BSW Editor, as shown in Figure 16.

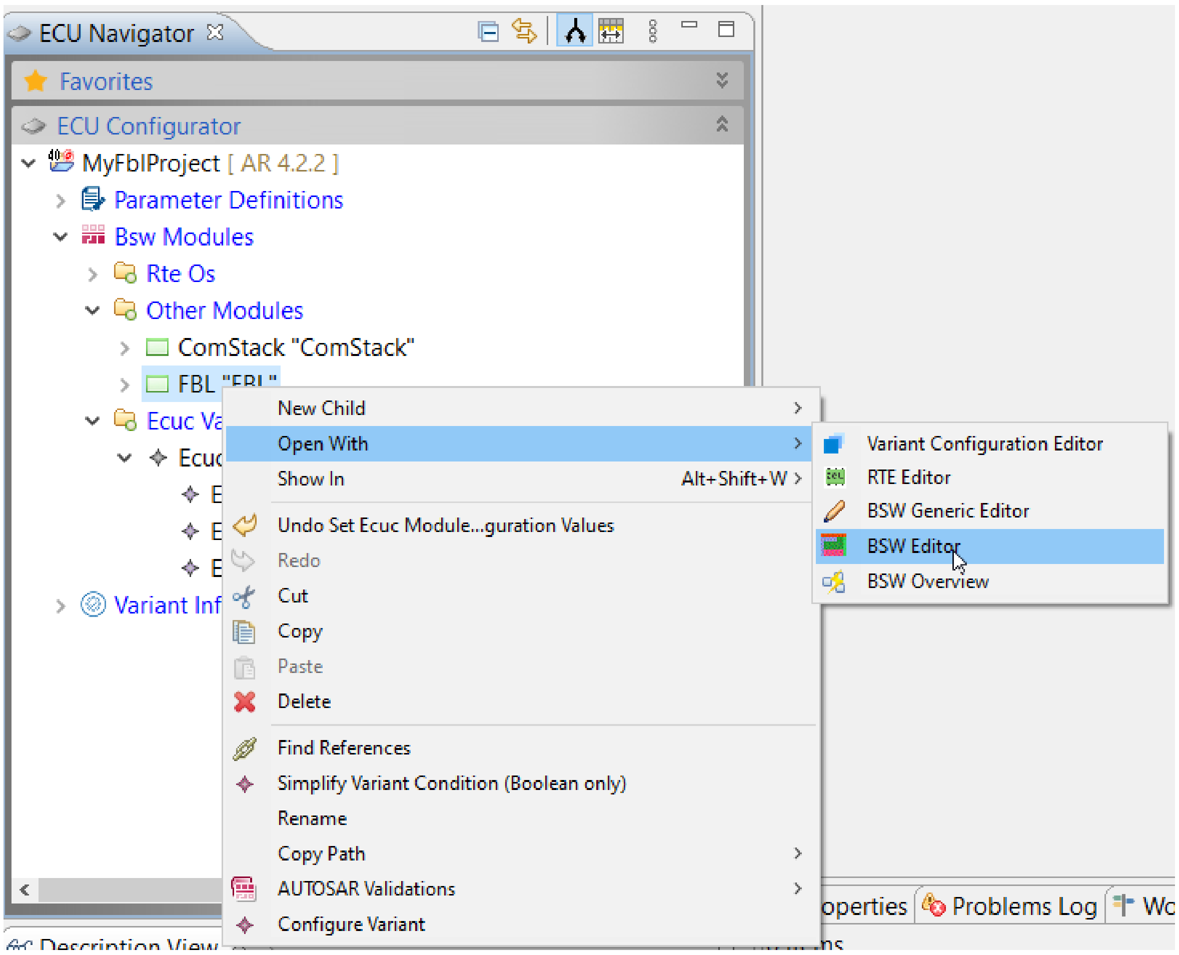
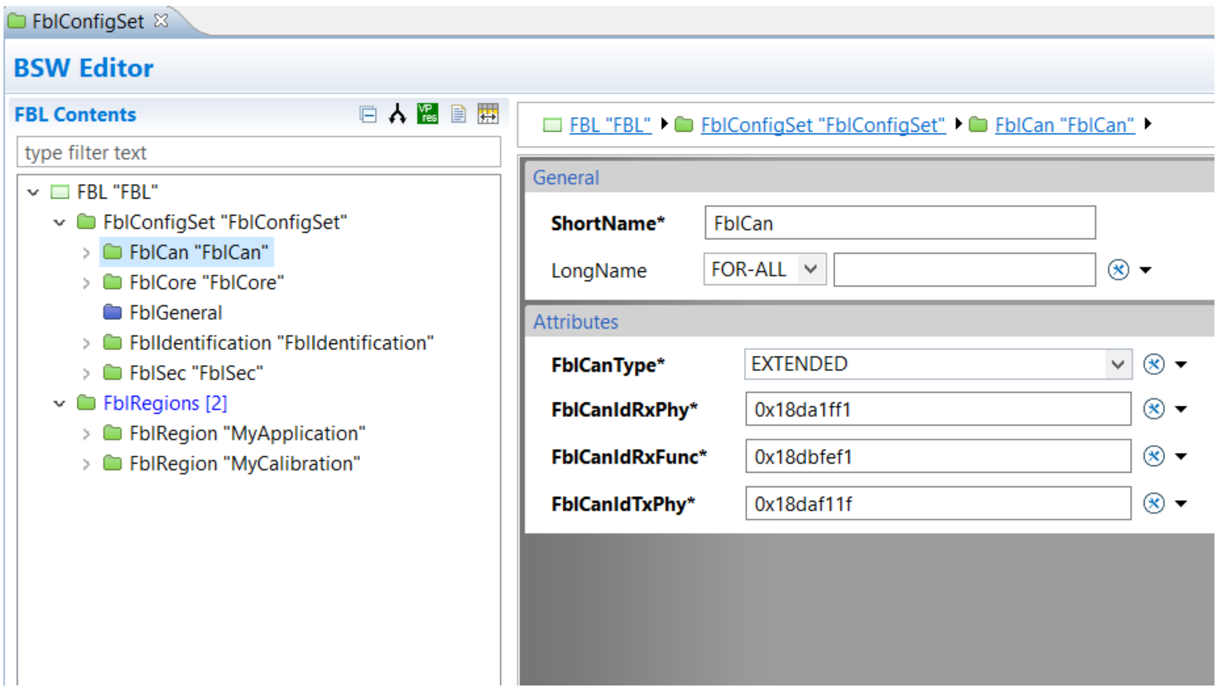


Figure 16: Accessing the FBL configuration parameters

The user can now edit the base configuration parameters in the RTA-FBL Editor window. Figure 17 shows as an example the port-specific configuration parameters. An explanation of each parameter is provided at the end of this section.



FblConfigSet

BSW Editor

FBL Contents

type filter text

```
FBL "FBL"
  FblConfigSet "FblConfigSet"
    FblCan "FblCan"
    FblCore "FblCore"
    FblGeneral
    FblIdentification "FblIdentification"
    FblSec "FblSec"
  FblRegions [2]
    FblRegion "MyApplication"
    FblRegion "MyCalibration"
```

FBL "FBL" ▸ FblConfigSet "FblConfigSet" ▸ FblCan "FblCan" ▸

General

ShortName*	FblCan
LongName	FOR-ALL

Attributes

FblCanType*	EXTENDED
FblCanIdRxPhy*	0x18da1ff1
FblCanIdRxFunc*	0x18dbfef1
FblCanIdTxPhy*	0x18daf11f

Figure 17: Edit 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 18 and then, in the opened RTA Code Generator window, by clicking Run after selecting only RTA-FBL – 1.0.0.Standard as shown in Figure 19.

RTA-OS should be generated only if the user wants to replace the FlashBootloader scheduler with a version of RTA-OS that is has configured for his target.

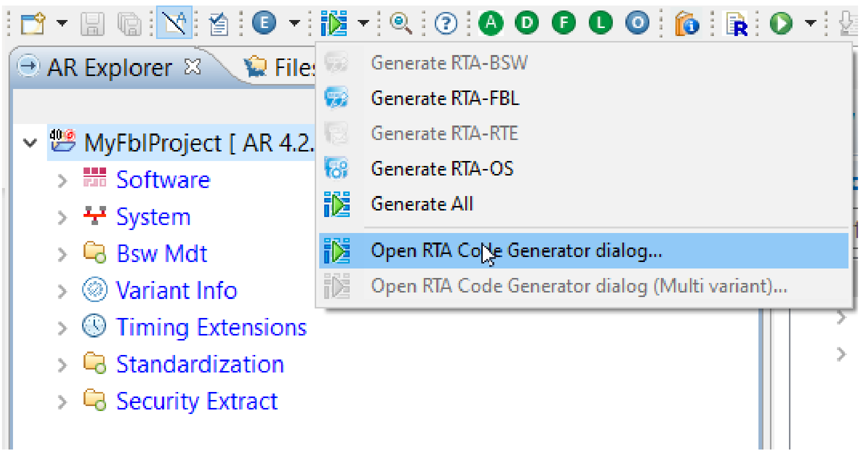
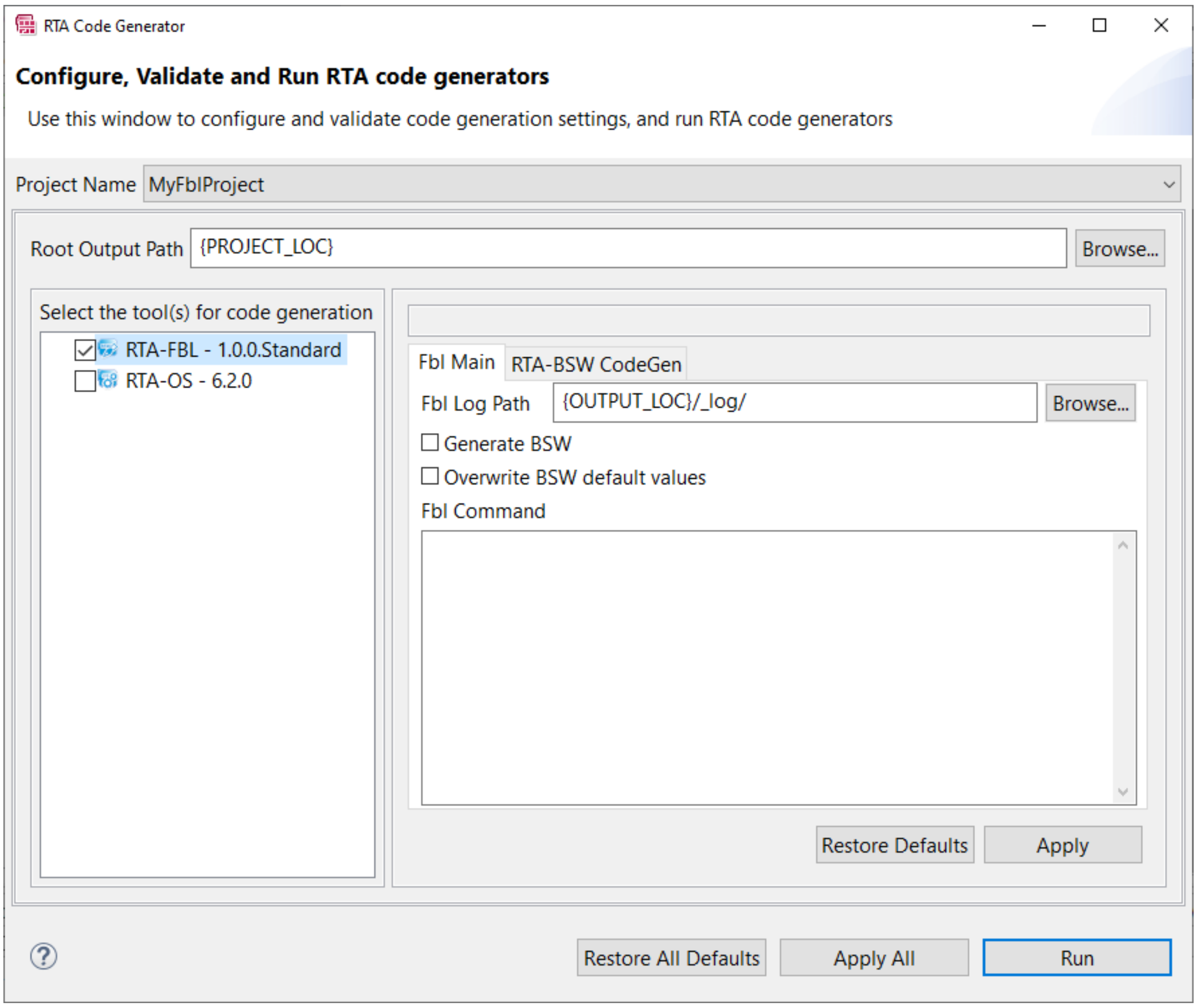


Figure 18: Open RTA Code Generator Dialog


RTA Code Generator

Configure, Validate and Run RTA code generators

Use this window to configure and validate code generation settings, and run RTA code generators

Project Name: MyFblProject

Root Output Path: {PROJECT_LOC}

Select the tool(s) for code generation:
- ☑ RTA-FBL - 1.0.0.Standard
- ☐ RTA-OS - 6.2.0

Fbl Main | RTA-BSW CodeGen

Fbl Log Path: {OUTPUT_LOC}/_log/

☐ Generate BSW

☐ Overwrite BSW default values

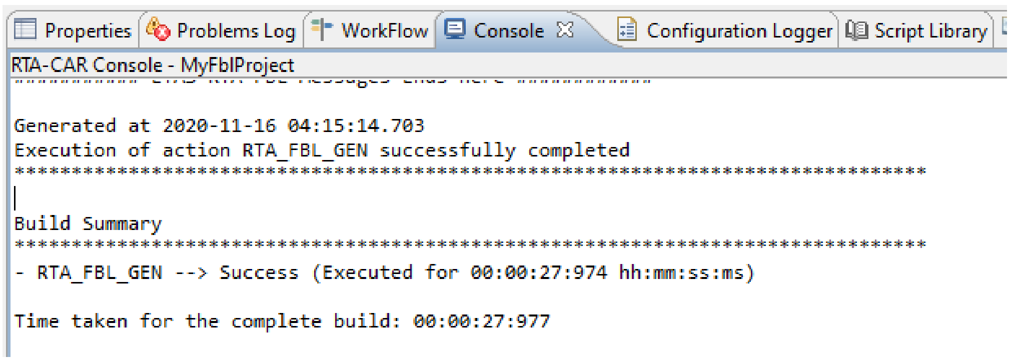
Fbl Command

Figure 19: RTA Code Generator

Note the two options that are available:

- Generate BSW: This will automatically generate the BSW after FBL generation using the BSW configuration generated by the FBL generator.
- Overwrite BSW default values: If this option is selected, any manual changes you have made to the BSW configuration after the last FBL generation will be lost and overwritten by default values. Note that this option should only be selected once you have generated the BSW at least once (using the option "Generate BSW" as described above).
    - **IMPORTANT**: The FBL generator will always overwrite all BSW configuration held in the configuration file Fblgen_EcucValues.arxml, even if the "Overwrite BSW default values" option is not selected. The configuration in this file cannot be modified as these values are completely defined by the configuration of the bootloader.

On clicking Run, the RTA-FBL instance is generated. Figure 20 shows the result of a successful generation in the console window.



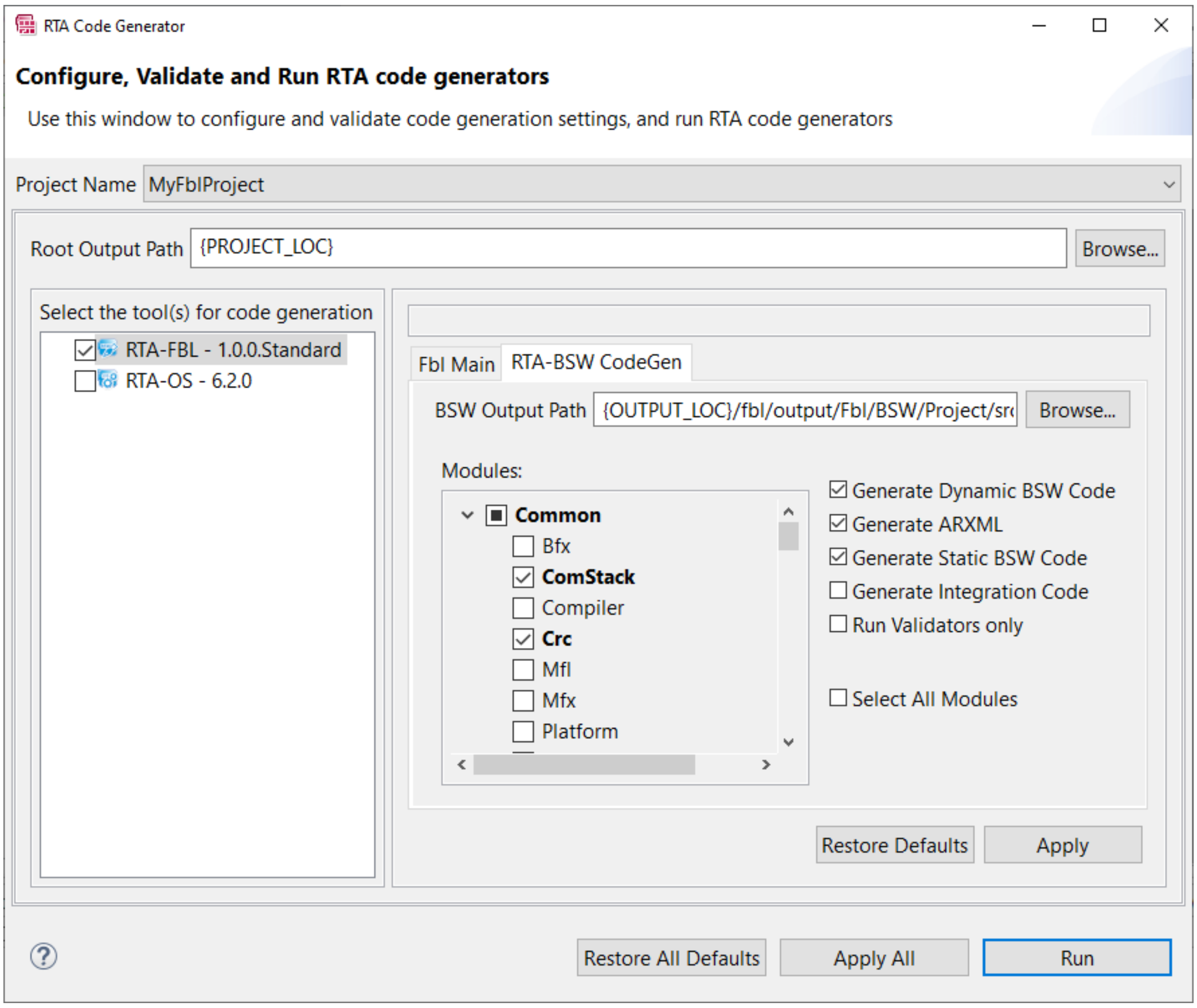
RTA-CAR Console - MyFblProject

```
Generated at 2020-11-16 04:15:14.703
Execution of action RTA_FBL_GEN successfully completed
********************************************************************************

Build Summary
********************************************************************************
- RTA_FBL_GEN --> Success (Executed for 00:00:27:974 hh:mm:ss:ms)

Time taken for the complete build: 00:00:27:977
```

Figure 20: 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.

If you have previously generated the RTA-FBL instance, the configured modules are already selected, as shown in Figure 21.


RTA Code Generator

Configure, Validate and Run RTA code generators

Use this window to configure and validate code generation settings, and run RTA code generators

Project Name: MyFblProject

Root Output Path: {PROJECT_LOC}

Select the tool(s) for code generation:
- [✓] RTA-FBL - 1.0.0.Standard
- [ ] RTA-OS - 6.2.0

Tabs: Fbl Main | RTA-BSW CodeGen

BSW Output Path: {OUTPUT_LOC}/fbl/output/Fbl/BSW/Project/src

Modules:
- ■ Common
  - [ ] Bfx
  - [✓] ComStack
  - [ ] Compiler
  - [✓] Crc
  - [ ] Mfl
  - [ ] Mfx
  - [ ] Platform

- [✓] Generate Dynamic BSW Code
- [✓] Generate ARXML
- [✓] Generate Static BSW Code
- [ ] Generate Integration Code
- [ ] Run Validators only
- [ ] Select All Modules

Figure 21: RTA-BSW CodeGen tab

Once complete, check the box Generate BSW in the 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 22. Upon successful generation, the popup message in Figure 23 is shown.

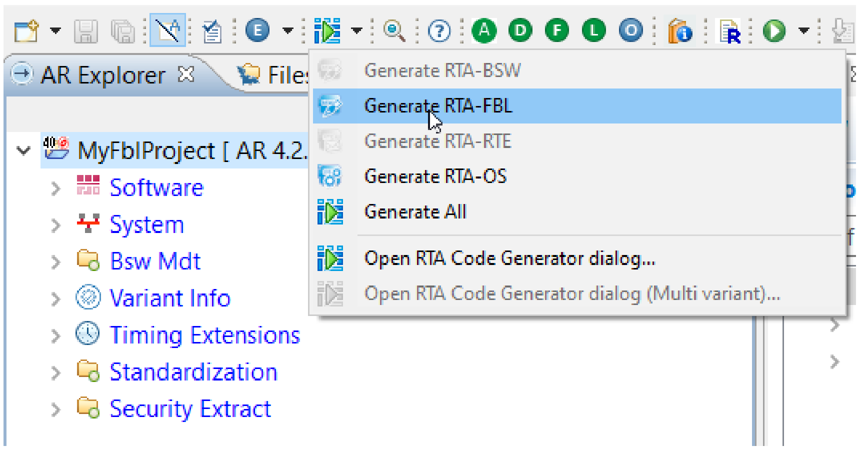
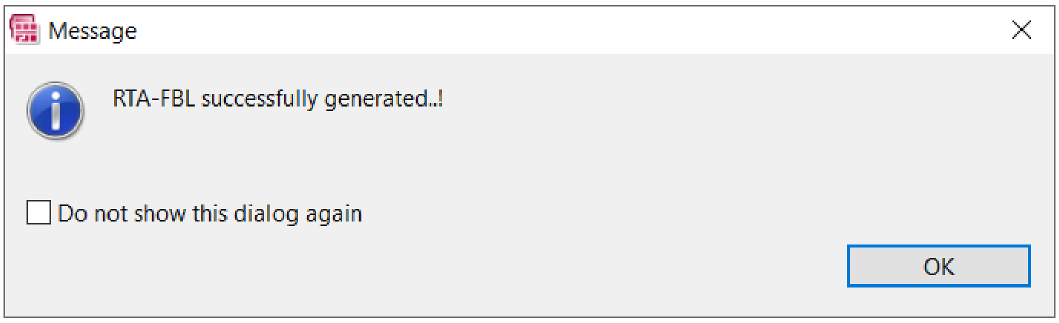


Figure 22: Generate RTA-FBL


Message

RTA-FBL successfully generated..!

Do not show this dialog again

OK

Figure 23: Successful generation

Following sections describe the parameters that the user can configure. The letters 'M' and 'O' are used to indicate "Mandatory" and "Optional" respectively. The column "Requires BSW regen." indicates whether the BSW needs to be re-generated in case the associated parameter has been changed.

Note that your target may also specify parameters that are unique to that target. These will also be listed in your Standard FBL Target Guide.

FblRegions

This container allows the configuration of the memory regions of your ECU. For each region the parameters details are listed in Table 2.

The allowed range for each FblRegion that can be specified is different for each target and can be found in your Standard FBL Target Guide.

Table 2: Configuration parameters FblRegions of RTA-FBL Standard

Parameter	**Description**	**Requires BSW Re-Gen**	**Optional or Mandatory**
FblRegionAddressLow	Specifies the low address of the region. Enter a Program Flash address valid for your target. See your RTA-FBL Standard Target Guide for details of the allowed range.	Yes	M

FblRegionAddressHigh	Specifies the high address of the region. Enter a Program Flash address valid for your target. See your RTA-FBL Standard Target Guide for details of the allowed range.	Yes	M
FblRegionType	Specifies the region type from below list: • INTERNAL_BOOTLOADER_REGION, • INTERNAL_APPLICATION_REGION, • EXTERNAL_APPLICATION_REGION, • INTERNAL_CALIBRATION_REGION, • EXTERNAL_CALIBRATION_REGION Internal regions erasing and writing are handled by the FBL, while for external regions user callouts are provided.	Yes	M
FblRegionId	Specifies a unique number to identify each region in UDS services. Note that number "0" is reserved for the FBL itself, and the FblRegionId of all regions must be contiguous.	Yes	M
FblRegionOptional	Specifies if the region must be downloaded or it is optional. If set to TRUE, the region is marked as optional and even if it is not downloaded, Check Programming Dependencies RID will return success. For an INTERNAL_BOOTLOADER_REGION this parameter is not relevant and it would be ignored.	Yes	M

FblCan

The Can parameters details of FblCan container are listed in Table 3.

For additional information on the MCAL Can configuration, please refer to your Standard FBL Target Guide.

Table 3: Configuration parameters FblCan of RTA-FBL Standard

Parameter	Description	Requires BSW Re-Gen	Optional or Mandatory
FblCanType	Specifies the CAN communication type from below list: • EXTENDED, • EXTENDED_FD, • STANDARD, • STANDARD_FD EXTENDED refers to 29 bits Can id configuration, while STANDARD refers to 11 bits Can id configuration. CAN-FD can be enabled selecting EXTENDED_FD or STANDARD_FD.	Yes	M

FblCanIdRxPhy	Allows to configure the CAN Id of UDS Physical Request. Depending on FblCanType configuration, you should enter a 29-bit CAN id or 11-bit CAN id.	Yes	M
FblCanIdRxFunc	Allows to configure the CAN Id of UDS Functional Request. Depending on FblCanType configuration, you should enter a 29-bit CAN id or 11-bit CAN id.	Yes	M
FblCanIdTxPhy	Allows to configure the CAN Id of UDS Response. Depending on FblCanType configuration, you should enter a 29-bit CAN id or 11-bit CAN id.	Yes	M

FblCore

The parameters details of FblCore container are listed in Table 4.

This parameters set is common to all RTA-FBL ports.

Table 4: Configuration parameters FblCore of RTA-FBL Standard

Parameter	Description	Requires BSW Re-Gen	Optional or Mandatory
EraseTimeout	Max time in milliseconds for erase Program Flash before timing out. You should enter an integer value between 1 and 100000.	No	M
WriteTimeout	Max time in milliseconds for writing Program Flash before timing out. You should enter an integer value between 1 and 100000.	No	M
VerifyTimeout	Max time in milliseconds for verifying Program Flash before timing out. You should enter an integer value between 1 and 100000.	No	M
StartAddress	Memory address of the first instruction of application software.	No	M

FblIdentification

The Identification parameters details of the FblIdentification container are listed in Table 5.

For additional DIDs details please refer to 3.3.1

Table 5: Configuration parameters FblIdentification of RTA-FBL Standard

Parameter	Description	Requires BSW Re-Gen	Optional or Mandatory
FblEcuSerialNumberSupport	Specifies if the user callback function	No	O

	Fbl_Port_GetEcuSerialNumberUserHook() is to be called to read the Ecu Serial Number for DID 0xF18C, or if the Ecu Serial Number is to be read from NvM. If set to ECUSERIALNUMBER_USER_SUPPORT, then the callback function is called. If set to ECUID_NVM_SUPPORT, then the Ecu Serial Number is read from NvM. If not specified, then the default behavior is to use NvM (i.e. the behavior is as if ECUSERIALNUMBER_NVM_SUPPORT is configured).		
BootSoftwareIdentification	Specifies default ROM value for DID $F180, stored in NvM block NvM_DIDF180_BootSoftwareIdentification. Note that the number of bootloader region is added as first byte. It should be in range 0x00 - 0xFFFFFF	No	M
ApplicationSoftwareIdentification	Specifies default ROM value for DID $F181, stored in NvM block NvM_DIDF181_ApplicationSoftwareIdentification. Note that the number of application regions is added as first byte. It should be in range 0x00 - 0xFFFFFFFFFF	No	M
ApplicationDataSoftwareIdentification	Specifies default ROM value for DID $F182, stored in NvM block NvM_DIDF182_ApplicationDataSoftwareIdentification. Note that the number of application regions is added as first byte. It should be in range 0x00 - 0xFFFFFFFFFF	No	M
EcuSerialNumber	Specifies default ROM value for DID $F18C, stored in NvM block NvM_DIDF18C_EcuSerialNumber. It should be in range 0x00 - 0xFFFFFF.	No	O

FblSec

The security related parameters details that can be configured within FblSec container are listed in Table 6.

Table 6: Configuration parameters FblSec of RTA-FBL Standard

Parameter	Description	Requires BSW Re-Gen	Optional or Mandatory
FblSoftwareVerificationType	Specifies how the newly reprogrammed software should be verified. It could be: • CRC, • SIGNATURE. The CRC verification is based on CRC32H04C11DB7 polynomial and	Yes	M

	0xFFFFFFFF as initial value. The SIGNATURE verification is based on RSASSA-PKCS1-v1_5 with 2048 bit key. This parameter affects RID 0xF000, see sections 3.3.2 for details Note that if "SIGNATURE" is selected, INCA Prof is generated to create the application signature, see section 4.1 for details		
FblSecurityPublicKey	Specifies the public key used by the FBL when FblSoftwareVerificationType is SIGNATURE. It could contain the 256-byte hex value of the key or the absolute path of a text file with the key. Sample keys with the different allowed input formats are provided for you under C:\ETAS\RTA-FBL_1.0.0_Standard\Ports\Standard\Samples	No	O
FblSeedKeyConstant1	Specify a first security constant to customize the Seed&Key algorithm to unlock level 0x01. It should be in range 0x00 - 0xFFFFFFFF. It is recommended to use a 4 bytes value for more unpredictability.	No	M
FblSeedKeyConstant2	Specify a second security constant to customize the Seed&Key algorithm to unlock level 0x01. It should be in range 0x00 - 0xFFFFFFFF. It is recommended to use a 4 bytes value for more unpredictability.	No	M

FblGeneral

The general parameters details that can be configured within FblGeneral container are listed in Table 7.

Table 7: Configuration parameters FblGeneral of RTA-FBL Standard

Parameter	Description	Requires BSW Re-Gen	Optional or Mandatory
FblBlockSize	The download block size in bytes, used for Transfer Data service. Select one of he allowed values, depending on your target memory alignment. If no value is entered, then a target specific default value is used.	No	O
FblSleepTimer	Specifies if the bootloader should enter in sleep mode after a defined amount of time in Default Session without an active diagnostic communication.	No	O

	This is to prevent KL +30 ECU without a valid application to be always awake, since the bootloader doesn't manage Network Management. Select one of the allowed values: • DISABLED, • 20, • 60, • 300. If FblSleepTimer equals "DISABLED", user callout `Fbl_Port_GoToSleep` is never triggered. The other values specify the number of seconds after which the callout `Fbl_Port_GoToSleep` should be triggered.		

3.5.3 Files created during generation

When you generate an instance of the Standard RTA-FBL using the RTA-FBL plugin for ISOLAR-AB, a series of files is created within a number of folders that you then use to build your RTA-FBL instance. Table 8 summarizes the folder structure created for this port. Additional folders which contain the target-specific elements, such as target code and sample build scripts, will be created. See your RTA-FBL Standard Target Guide for details of the content of these additional folders.

Please note that the executable generated using our sample build scripts includes debug symbols, to make debugging/troubleshooting easier. However, we recommend disabling/stripping debug symbols from your final production builds, as debug symbols may constitute a security risk in some use-cases.

Table 8: Files created by RTA-FBL generation

L1 Folder	Description
./	The home of the RTA-CAR project
./fbl/input	Internal files for RTA-CAR created during project creation, with FBL module configuration. Do not manually edit these files.
./fbl/output/Fbl/Bootloader	This contains the core (port-independent and target-independent) modules.
./fbl/output/Fbl/BSW	This folder contains the RTA-BSW project used to generate the FBL BSW modules. You can investigate the configuration used for the BSW modules of the FBL. If the configuration in the project is manually changed and a new BSW generated, then it is the integrator’s responsibility to test that these changes do not affect the bootloader’s correct functionality.
./fbl/output/Fbl/INFRA/ECL	This is the ETAS crypto library (ECL). It should not be modified by the integrator.
./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 3.5.9, but the functions in BLSM_Main.c should not be changed. It is the integrator’s responsibility to test that any change made to the BLSM does 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 3.5.8. For example, the integrator may wish to use RTA-OS in order to more easily configure interrupts for other software integrated within the RTA-FBL. It is the integrator’s responsibility to test that any change made to the OS does not affect the bootloader’s correct functionality.
./fbl/output/Fbl/INFRA/Port	This folder contains the code that implements port-specific functionality. The file FBL_Port.c should not be modified by the integrator.

	The other files may be modified depending on your ECU's use cases. It is the integrator's responsibility to test that any change made to the Port folder does not affect the bootloader's correct functionality.
./fbl/output/Fbl/INFRA/Stubs	This folder contains stub code necessary due to the AUTOSAR architecture. Files in this folder should not be modified by the integrator.

3.5.4 The RTA-FBL instance for the Dummy Target

The dummy target provided with the Standard Port cannot be built. You can only use the generated code as a reference to explore how different parameters change the generated FBL instance. Your Standard FBL Target Guide will provide information on how to build an instance of the bootloader for your real target.

The FBL for your target will have undergone an in-depth testing using the compiler and MCAL that you have chosen. The Standard FBL Target Guide for your target will indicate the tools and their versions that you must have to create a buildable FBL instance. All targets use a common base that require the tools as described in Table 1.

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 TC275 processor. This processor has a memory layout as shown in Table 9. Memory regions of a space must begin on sector boundaries and the bootloader reserves the first sector (i.e. the memory between 0xA0000000 and 0xA0003FFF). You can experiment with different configurations of Application and Calibration regions if you have not yet received your Target package. For example, if you configure a region that uses an address that is not on a region boundary or that enters a disallowed space note the error returned by the FBL generator.

Table 9: Memory layout of the Dummy Target

Bank	Sector	Start	End	Comment
0	0	0x80000000	0x80003FFF	Reserved for FBL
	1	0x80004000	0x80007FFF	Available for Application/Calibration
	2	0x80008000	0x8000BFFF	
	3	0x8000C000	0x8000FFFF	
	4	0x80010000	0x80013FFF	
	5	0x80014000	0x80017FFF	Not available for Application/Calibration
	6	0x80018000	0x8001BFFF	
	7	0x8001C000	0x8001FFFF	Available for Application/Calibration
	8	0x80020000	0x80027FFF	
	9	0x80028000	0x8002FFFF	
	10	0x80030000	0x80037FFF	

	11	0x80038000	0x8003FFFF	
	12	0x80040000	0x80047FFF	
	13	0x80048000	0x8004FFFF	
	14	0x80050000	0x80057FFF	
	15	0x80058000	0x8005FFFF	
	16	0x80060000	0x8006FFFF	Not available for Application/Calibration
	17	0x80070000	0x8007FFFF	
1	18	0x80080000	0x8008FFFF	Available for Application/Calibration
	19	0x80090000	0x8009FFFF	
	20	0x800A0000	0x800BFFFF	
	21	0x800C0000	0x800DFFFF	
	22	0x800E0000	0x800FFFFF	
2	23	0x80100000	0x8013FFFF	
	24	0x80140000	0x8017FFFF	
3	25	0x80180000	0x801BFFFF	
	26	0x801C0000	0x801FFFFF	
4	0	0x80200000	0x80203FFF	
	1	0x80204000	0x80207FFF	
	2	0x80208000	0x8020BFFF	
	3	0x8020C000	0x8020FFFF	
	4	0x80210000	0x80213FFF	
	5	0x80214000	0x80217FFF	
	6	0x80218000	0x8021BFFF	
	7	0x8021C000	0x8021FFFF	
	8	0x80220000	0x80227FFF	
	9	0x80228000	0x8022FFFF	
	10	0x80230000	0x80237FFF	
	11	0x80238000	0x8023FFFF	
	12	0x80240000	0x80247FFF	
	13	0x80248000	0x8024FFFF	
	14	0x80250000	0x80257FFF	
	15	0x80258000	0x8025FFFF	
	16	0x80260000	0x8026FFFF	
	17	0x80270000	0x8027FFFF	
5	18	0x80280000	0x8028FFFF	
	19	0x80290000	0x8029FFFF	
	20	0x802A0000	0x802BFFFF	
	21	0x802C0000	0x802DFFFF	
	22	0x802E0000	0x802FFFFF	
6	23	0x80300000	0x8033FFFF	
	24	0x80340000	0x8037FFFF	
7	25	0x80380000	0x803BFFFF	
	26	0x803C0000	0x803FFFFF	

Integrator guidelines

Section 3.5 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 (optional)
- BSW module 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.

Finally, the integrator may also need to make changes to default generated target code. The integration guidelines for your target will be provided in your RTA-FBL Standard Target Guide. For most targets, you will likely need to consider:

- C-code startup and trap table updates,
- MCAL adaptation,
- Completion of target-specific user functions (optional).

3.5.5 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 3.5. The allowed range for your target is described in you RTA-FBL Standard Target Guide. An example of a typical memory layout is depicted in Figure 24.

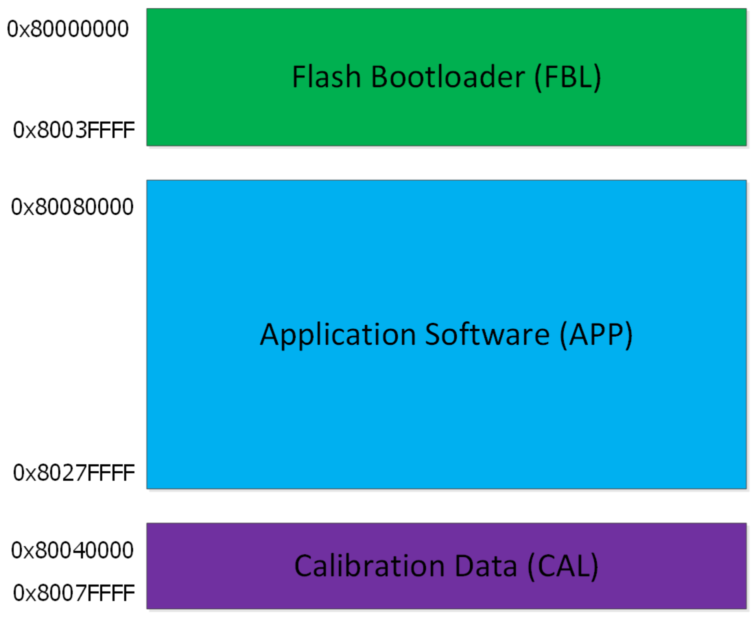


Figure 24 - Sample Memory Layout

3.5.6 FBL: User Functions

You will find all user functions that you may need to complete in the generated files within the \Ports\Standard\INFRA\Port\src folder of your generated FBL instance.

Sleep

The function `Fbl_Port_GoToSleep` is called by the bootloader when no UDS messages are received after a configured amount of time. The integrator should modify this code to put the ECU to sleep. The function is triggered depending on the FBL parameter FblSleepTimer.

Watchdog

The FBL does not implement any watchdog functionality. As an example, for the integrator, the call `Fbl_Port_WatchDogInitialise` is called from `Os_Start` and the user should place the code that initializes the watchdog in this function. The function `Fbl_Port_WatchDogRefresh` must then be called to pet the watchdog from within a cyclic OS function. In the provided OS, this is called every 100ms, but the integrator can call `Fbl_Port_WatchDogRefresh` at whatever rate is deemed suitable.

Get the ECU Serial Number

The function `Fbl_Port_GetEcuSerialNumberUserHook` is called by the bootloader when the FBL parameter FblEcuSerialNumberSupport is set to ECUSERIALNUMBER_USER_SUPPORT. You need to return the 3 bytes value of the ECU Serial Number for DID $F18C depending on how this is provisioned for your ECU.

Application Validation

The function `Fbl_Port_UserValidApplication` is called by the bootloader when all the regions have been downloaded successfully, within RID 0xFF01 processing. The integrator may verify if the new software is compatible and return `FALSE` in case of hardware/software incompatibility or software modules incompatibility. If the function returns `TRUE`, after the reset the application will be executed by the bootloader. If the function returns `FALSE`, RID 0xFF01 will reply with NRC 0x72 and Programming Status DIDs will report 0xF0

External Memory Reprogramming

The functions within FBL_PortUserFlashCode.c are triggered if a region is configured as EXTERNAL using the FBL parameter FblRegionType.

Note that just one function among `UserFlash_VerifyBlockCRC` and `UserFlash_VerifyBlockSignature` is triggered by the Fbl, depending on the FBL parameter FblSoftwareVerificationType.

Prototype	`UserFlash_ReturnType UserFlash_FlashErase ( uint32 TargetAddress, uint32 Length )`
Parameter	`TargetAddress`: low address of the external memory device to be erased `Length`: size of the external memory device to be erased
Return Code	`FBL_USER_FLASH_RESULT_SUCCESS`: Erase completed successfully. `FBL_USER_FLASH_RESULT_PROCESSING`: Erase is in progress and requires additional time. `FBL_USER_FLASH_RESULT_FAILURE`: Erase completed with failure. It

	will result in NRC 0x78 to RID 0xFF00
Functional Description	Triggered when RID 0xFF00 – Erase Memory is received for a region configured as external. The integrator should start the erase process of the memory device using its own driver.
Pre-Conditions	None

Prototype	`UserFlash_ReturnType UserFlash_FlashWrite ( uint32 TargetAddress, uint32 Length, const uint8 *SourceAddressPtr )`
Parameter	`TargetAddress`: address of the external memory device where data should be written `Length`: size of the data to be written `*SourceAddressPtr`: pointer to data
Return Code	`FBL_USER_FLASH_RESULT_SUCCESS`: Write completed successfully `FBL_USER_FLASH_RESULT_PROCESSING`: Write is in progress and requires additional time `FBL_USER_FLASH_RESULT_FAILURE`: Write completed with failure. It will result in NRC 0x78 to Transfer Data service
Functional Description	Triggered each time data from Transfer Data service is processed. Note that `Length` depends on the Fbl parameter FblBlockSize. The integrator should start the write process of the memory device using its own driver.
Pre-Conditions	None

Prototype	`UserFlash_ReturnType UserFlash_VerifyBlockCRC ( uint32 TargetAddress, uint32 Length, uint32 CRC )`
Parameter	`TargetAddress`: address of the external memory device to calculate the CRC. `Length`: size of the data whose CRC should be verified `CRC`: Checksum received by the tester.
Return Code	`FBL_USER_FLASH_RESULT_SUCCESS`: CRC verification completed successfully `FBL_USER_FLASH_RESULT_PROCESSING`: CRC verification is in progress and requires additional time `FBL_USER_FLASH_RESULT_FAILURE`: CRC verification completed with failure. It will result in NRC 0x78 to RID 0xF000 VerifyDownload
Functional Description	Triggered when RID 0xF000 - VerifyDownload is received for a region configured as external. The integrator should calculate the CRC of the memory region and compare it with the received one. If they match, the result should be success.
Pre-Conditions	None

Prototype	`UserFlash_ReturnType UserFlash_VerifyBlockSignature ( uint32 TargetAddress, uint32 Length, uint8* Signature )`
Parameter	`TargetAddress`: address of the external memory device to

	calculate the signature. `Length`: size of the data whose signature should be verified `Signature`: Pointer to 256 bytes of signature received by the tester.
Return Code	`FBL_USER_FLASH_RESULT_SUCCESS`: Signature verification completed successfully `FBL_USER_FLASH_RESULT_PROCESSING`: Signature verification is in progress and requires additional time `FBL_USER_FLASH_RESULT_FAILURE`: Signature verification completed with failure. It will result in NRC 0x78 to RID 0xF000 VerifyDownload
Functional Description	Triggered when RID 0xF000 - VerifyDownload is received for a region configured as external. The integrator should calculate the signature of the memory region and compare it with the received one. If they match, the result should be success. Note that Etas ECL may be used for signature verification if needed.
Pre-Conditions	None

Trusted Boot

RTA-FBL Standard does not natively support the so-called Trusted Boot functionality, to determine whether the ECU software has been tampered or replaced with a malicious one.

RTA-FBL Standard offers you anyway the possibility to add this feature, providing a set of APIs to interface with the fbl and fulfill the most common cybersecurity use cases.

The available interfaces are within FBL_Security.c and described below.

Prototype	`Std_ReturnType Fbl_Security_PreHostEraseHook (void )`
Parameter	None
Return Code	`E_OK`: Erase can be executed by the host `E_NOT_OK`: Erase can't be executed, fbl will return NRC 0x78
Functional Description	Triggered before the host start erasing PFLASH. It may be needed to inform the Hardware Trust Anchor about the incoming erase operation (e.g. to suspend a run time manipulation detection). If the host is not allowed to erase the PFLASH in that context, the erase operation could be aborted, returning a NRC 0x78.
Pre-Conditions	None

Prototype	`void Fbl_Security_PostHostEraseHook (void )`
Parameter	None
Return Code	None
Functional Description	Triggered after the host has completed erasing PFLASH. It may be needed to inform the Hardware Trust Anchor about the end of the erase operation (e.g. to resume a run time manipulation detection).

Pre-Conditions	None

Prototype	`Std_ReturnType Fbl_Security_PreHostWriteHook (void )`
Parameter	None
Return Code	`E_OK`: Write can be executed by the host `E_NOT_OK`: Write can't be executed, fbl will return NRC 0x78
Functional Description	Triggered before the host start writing PFLASH. It may be needed to inform the Hardware Trust Anchor about the incoming erase operation (e.g. to suspend a run time manipulation detection). If the host is not allowed to write the PFLASH in that context, the write operation could be aborted, returning a NRC 0x78. Note that this callout is triggered several times during the reprogramming phase, every time a Transfer Data block is processed.
Pre-Conditions	None

Prototype	`void Fbl_Security_PostHostWriteHook (void )`
Parameter	None
Return Code	None
Functional Description	Triggered after the host has completed writing to PFLASH. It may be needed to inform the Hardware Trust Anchor about the end of the write operation (e.g. to resume a run time manipulation detection).
Pre-Conditions	None

Prototype	`Std_ReturnType Fbl_Security_TrustedBootVerification (void )`
Parameter	None
Return Code	`E_OK`: Trusted Boot verification success `E_NOT_OK`: Trusted Boot verification failure
Functional Description	Triggered at startup to ensure the ECU has a trusted application. The Trusted Boot Verification should be synchronous and should determine if the application can be executed or the ECU software has been tampered. Note that this callout is triggered only if all mandatory regions have been successfully downloaded into the ECU.
Pre-Conditions	None

Prototype	`Fbl_Security_JobResultType Fbl_Security_UpdateAuthSWTable ( const uint8 SwPartIndex )`
Parameter	`SwPartIndex:` Index of the updated region. Note that fbl will trigger the callout using FblRegionId as index
Return Code	`FBL_PORT_SEC_JOB_RESULT_SUCCESS`: Authenticated trusted boot table has been updated `FBL_PORT_SEC_JOB_RESULT_PROCESSING`: Authenticated trusted boot table update is in progress and requires additional time

	FBL_PORT_SEC_JOB_RESULT_FAILURE: Authenticated trusted boot table update has failed
Functional Description	Triggered after a region has been updated, during RID 0xF000 – Verify Download. The integrator should trigger an asynchronous operation to update the Hardware Trust Anchor internal reference table used for Trusted Boot verification. A common use case is to update the reference CMAC, with a new calculation on the new software.
Pre-Conditions	None

Name	Fbl_Security_AuthSwTableType		
Type	Struct		
Members	uint8	SwPartID	An index to identify the software region to be verified.
	uint8	SwPartAddr	Low address of the software to be verified.
	uint8	SwPartLen	Size of the software to be verified.
Description	Example of the type of data that could be used by Fbl_Security_UpdateAuthSWTable and Fbl_Security_TrustedBootVerification Note that the structure AuthSwTable[] is created as example, with the regions configured with the first FBL generation. It is up to the integrator to decide which software parts should be verified and update the AuthSwTable[] structure accordingly.		

3.5.7 FBL: BSW adaptation

The BSW modules needed by RTA-FBL and generated in the generated BSW project are listed in Table 10. This list is the minimum setup needed for the basic FBL.

If changes are necessary in order to fulfill non-standard bootloader integration requirements, you are allowed to modify the BSW generated configuration. You are not allowed to modify any parameters within Fblgen_EcucValues.arxml: these configuration parameters are highlighted in brick red and not editable by the user. This file will always be overwritten during generation.

The integrator must always test the complete FBL after making any modifications to the generated BSW project.

Table 10: BSW modules list

BSW Module(s)	**Notes**
Dcm	The diagnostic communication module.
MemIf; Fee; Rba_FeeFs1, NvM	Memory stack modules for the NVM.
CanIf; CanSM; CanTp; CanTp_Precompile, ComM; ComStack; PduR	Can communication stack modules.
Crc	Uses for CRC calculation in NvM.

BSW Module(s)	**Notes**
Rba_ArxmlGen; Rba_DiagLib; Standard; BswM; BswM_Precompile_PB_Variant	Additional modules required for build.

3.5.8 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_SchTbl.c"
3. Create the task body in "Fbl/INFRA/Os/inc/Os_Tasks.c"

The OS is driven from a timer that is provided by the target through the macro `GET_SYSTEM_TIMER`. Your target will also export the #define `OS_TICK_TH` that is used as the tick counter for the OS. The rate at which this is ticked is target-dependent and can usually be set by the integrator. Please review your RTA-FBL Standard Target Guide for more information on how to set this rate.

IMPORTANT: The integrator is responsible to ensure 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.

3.5.9 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 within 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 to ensure that any modifications made to the BLSM are tested to ensure that the FBL continues to operate as expected.

3.5.10 FBL: C-code startup and trap table updates (optional)

The startup code and trap code used for your target is described in your RTA-FBL Standard Target Guide. The integrator may change these if required following the target guidelines, but is then responsible for testing of the complete FBL.

3.5.11 FBL: MCAL adaptation

The MCAL modules needed by RTA-FBL for all targets are listed in Figure 3. 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. See your RTA-FBL Standard Target Guide for further information on MCAL modifications for your target.

Table 11: 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 MCALs these drivers are implemented by two different modules.
Mcu	Provides core functionality such as clock handling, mcu reset, etc.
Port	Provides interface to port pin peripheral.

3.5.12 Application Software: 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 specific Fbl flag in NvM and then resets, in order to allow the FBL to handle the reprogramming request and to know that this request 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 application should take care of the unknown blocks configured in the FBL.

RTA-FBL Standard foresees some Mandatory and Optional blocks to be introduced in the Application NvM layout, while suggest to not includes some FBL private blocks. In order for the layout to be consistent, the Fee persistent IDs of the shared blocks must match between the application and FBL. Table 12 shows the NvM blocks and their Ids that must be also configured in the application. Note that if you already are using the Persistent IDs generated by fblgen in your application and do want to change these, then you can instead change these values in the FBL instance’s BSW configuration as long as you keep them consistent with the values in the Application.

Table 12: Fbl NvM configuration blocks

Block Name	**Persistent ID**	**Bytes**	**Fbl-App Shared Block**
NvM_ProgrammingConditionsBlock	2	31	Mandatory (*)
NvM_DIDF180_BootSoftwareIdentificationBlock	3	4	Optional (**)
NvM_DIDF181_ApplicationSoftwareIdentificationBlock	4	6	Optional (**)
NvM_DIDF182_ApplicationDataSoftwareIdentificationBlock	5	6	Optional (**)
NvM_DIDF18C_EcuSerialNumberBlock	6	3	Optional (**)

NvM_DID0100_ProgrammingCounterBootBlock	7	3	Optional (**)
NvM_DID0101_ProgrammingCounterApplBlock	8	n·3 (****)	Optional (**)
NvM_DID0102_ProgrammingCounterDataBlock	9	n·3 (****)	Optional (**)
NvM_DIDF183_BootFingerprintBlock	10	5	Optional (**)
NvM_DIDF184_ApplicationFingerprintBlock	11	n·5 (****)	Optional (**)
NvM_DIDF185_ApplicationDataFingerprintBlock	12	n·5 (****)	Optional (**)
NvM_DID0103_ProgrammingStatusApplBlock	13	n·2 (****)	Optional (**)
NvM_DID0104_ProgrammingStatusCalBlock	14	n·2 (****)	Optional (**)
NvM_ValidityFlag	15	4	No (***)
NvM_<region>ValidityFlag	16	4	No (***)

(*) Must be shared by Fbl and Application to handle the $10 02 jump
(**) Could be shared if both Fbl and Application need to access this NvM data
(***) Fbl private data management, strongly suggested to not share with Application
(****) Size of this NvM block depends on the number of FblRegions configured

Another parameter that needs to be aligned between Application and Bootloader is the Fee Sector Layout. The sector layout depends on application and target characteristics (data memory sizes mainly). The configuration of the Fee Sectors needs to match between FBL and Application, otherwise the Fee will recognize it as invalid and re-format the Data Memory (thus deleting the existing content). See your RTA-FBL Standard Target Guide for details of Fee Sectors configuration.

3.5.13 Application Software: Boot Jump Handling

To reprogram the ECU when an application is valid and running, it is necessary for the application to signal to the bootloader that reprogramming is required after the next reset. This sequence is show in Figure 25.

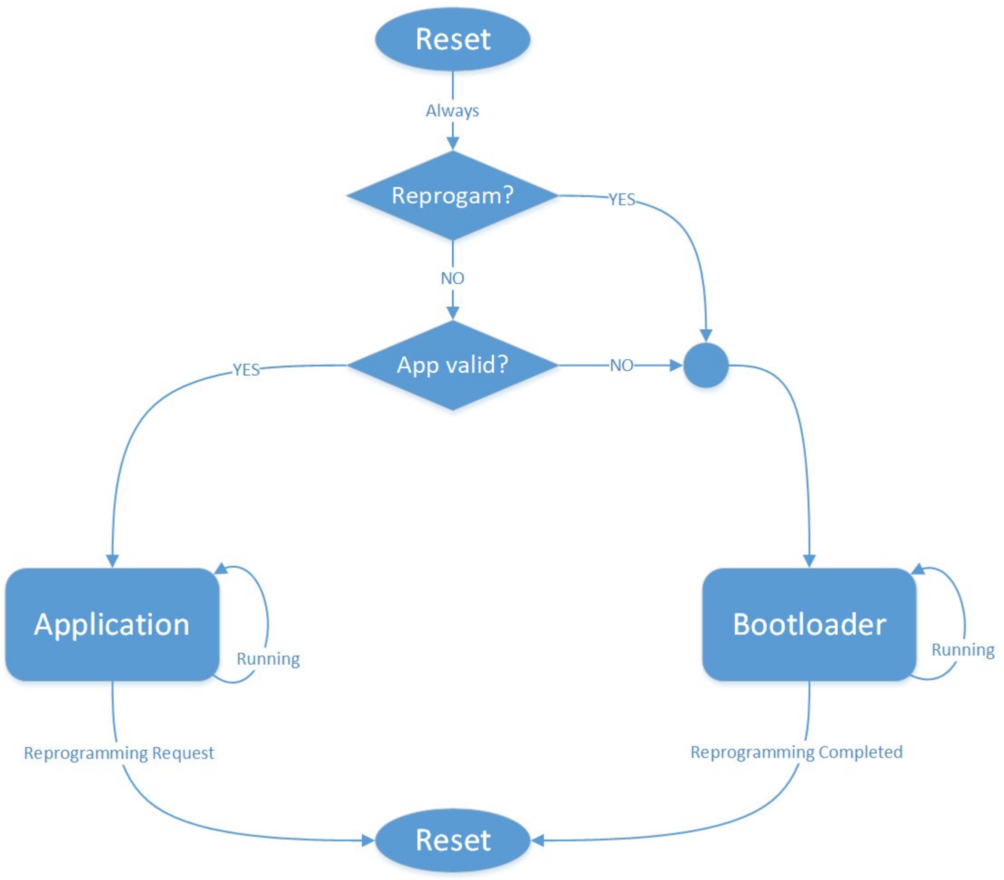
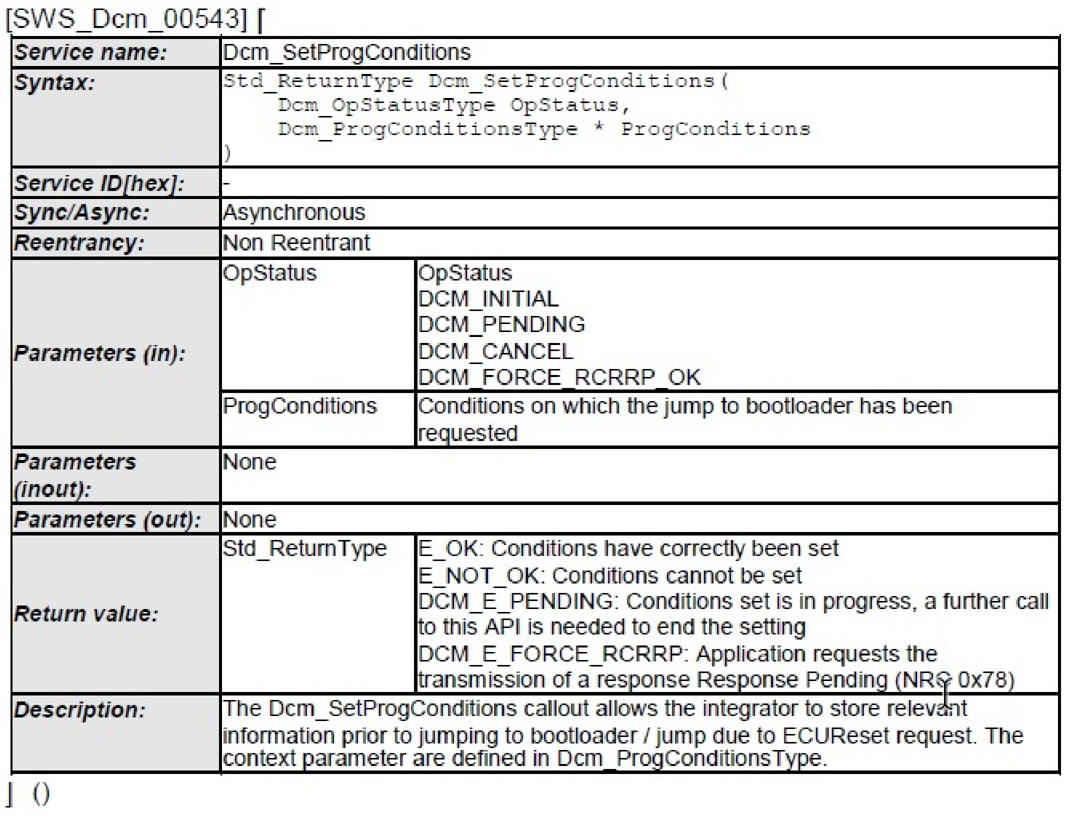


Figure 25: Handling of jump logic

In an AUTOSAR stack, the module responsible for the reception of the tester requests and triggering the jump to the bootloader is the Diagnostic Communication Manager (Dcm). For storing the information required by the bootloader, the Dcm will execute the application callout `Dcm_SetProgConditions` (see Figure 26 for API description).

This callout must be implemented by the user. The goal is to hand over the information in the parameter `ProgConditions` of the type `Dcm_ProgConditionsType` provided by Dcm and store it in a place and format, the bootloader can access and understand.

Note that the callout `Dcm_SetProgConditions` allows the return value `DCM_E_PENDING` which results in the `Dcm_MainFunction` calling `Dcm_SetProgConditions` in each subsequent cycle until it returns `E_OK`, before the Dcm continues with the jump to bootloader.



[SWS_Dcm_00543] ⌈

Service name:	Dcm_SetProgConditions	
Syntax:	`Std_ReturnType Dcm_SetProgConditions( Dcm_OpStatusType OpStatus, Dcm_ProgConditionsType * ProgConditions )`	
Service ID[hex]:	-	
Sync/Async:	Asynchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	OpStatus	OpStatus DCM_INITIAL DCM_PENDING DCM_CANCEL DCM_FORCE_RCRRP_OK
	ProgConditions	Conditions on which the jump to bootloader has been requested
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Conditions have correctly been set E_NOT_OK: Conditions cannot be set DCM_E_PENDING: Conditions set is in progress, a further call to this API is needed to end the setting DCM_E_FORCE_RCRRP: Application requests the transmission of a response Response Pending (NRC 0x78)
Description:	The Dcm_SetProgConditions callout allows the integrator to store relevant information prior to jumping to bootloader / jump due to ECUReset request. The context parameter are defined in Dcm_ProgConditionsType.	

⌋ ()

Figure 26: Dcm_SetProgConditions API from AUTOSAR SWS

Dcm uses the return value from the function `DcmAppl_DcmGetStoreType` to fill in `ProgConditions` parameter of `Dcm_SetProgConditions`. The recommended return value is `DCM_WARMRESPONSE_TYPE`

We strongly recommend to serialize the `ProgConditions` structure as shown in Table 13 in order to make sure that there are no dependencies on the structure of the data due to the compiler, compiler options, or the BSW version. You should be able to get all the values needed to create data by using the contents of the programming conditions sent as an input parameter to the function `Dcm_SetProgConditions` called from the Dcm when the Programming Session is entered in the application.

Note that `freeForProjectUse` bytes are used as reprogramming request flag and should contain exactly the data shown in Table 13.

The NvM block to store the programming conditions is NvM_ProgrammingConditionsBlock (id 2 in Table 12)

Table 13: Programming conditions data to be set in NvM as required by RTA-BSW

Name	First byte index	Size in Bytes	Description
ProtocolId	0	1	Active Protocol ID – Set by Dcm to identify the protocol on which Jumping is initiated
Sid	1	1	Active Service Identifier – Set by Dcm
SubFncId	2	1	Active Subfunction Id – Set by Dcm
StoreType	3	1	Storing Type used for Storing the information – Warm Request/Warm Response/Warm Init
SessionLevel	4	1	Active Session Level which needs to be stored

			– Set by Dcm
SecurityLevel	5	1	Active Security Level which needs to be stored – Set by Dcm
ReqResLen	6	1	Total Request/Response length including SID and Subfunc – Set by Dcm
NumWaitPend	7	1	Number of wait response pending triggered – Set by Dcm
ReqResBuf	8	8	Request / Response buffer – Set by Dcm
TesterSourceAddr	16	2	Tester diagnostic address. Note that the Dcm sets the Tester Source Address only if `DcmDslProtocolRxTesterSourceAddr` is correctly configured for each `DcmDs/DcmDslProtocol/DcmDslConnections` in BSW configuration.
ElapsedTime	18	4	Total elapsed time – Set by Dcm
ReprogramingRequest	22	1	Reprograming of ECU requested or not – Not Used.
ApplUpdated	23	1	Application has to be updated or not – Not Used.
ResponseRequired	24	1	Response has to be sent by flashloader or application – Set by Dcm
freeForProjectUse	25	6	Used to store the Fbl reprogramming request flag. This flag is used by the Fbl to recognize the application has received a reprogramming request from the tester and wants the Fbl to start the reprogramming flow and not jump into application. The first 4 bytes should contain the `uint32 0xAAFF55AA`. While the remaining 2 bytes are not used.

To instruct the Dcm to call the `Dcm_SetProgConditions` callback when the Programming Session is requested you must ensure the Dcm knows a jump to Bootloader is required. This is achieved with the parameter `DcmConfigSet/DcmDsp/DcmDspSession/DcmSessionRows/DcmDspSessionForBoot` that must be set to DCM_OEM_BOOT for Programming Session.

When receiving a programming session request, it is suggested to instruct the Dcm to send a NACK $78 on transition to boot, in order to allow timing extension (P2*). This configuration is needed to avoid that a long Bootloader startup time breaks an UDS timeout (since the $10 02 response will be sent by the bootloader after the jump). This is achieved setting the parameter `DcmConfigSet/DcmDsl/DcmDslProtocol/DcmDslProtocolRows/DcmSendRespPendOnTransToBoot` to TRUE.

3.6 Bootloader Update

The Flash Bootloader can be reprogrammed via UDS to upgrade the FBL of an ECU in which the debug port is not accessible: this is achieved using the Flash Bootloader Updater.

The Flash Bootloader Updater is downloaded just like a normal application (with FblRegionId = 0). In its code flash it holds the new flash bootloader, once started, replaces the existing Bootloader by the new version.

If a power failure or a reset occurs during the update process, at next power on the Bootmanager will execute again the Bootloader Updater. When the update process is completed, the new Flash Bootloader is executed, and the application can be reprogrammed.

The figures below show a conceptual overview of the update process:

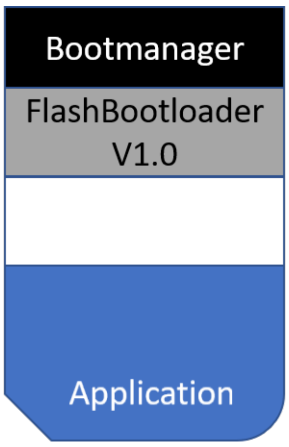


Figure 29. Initial SW: Application is running

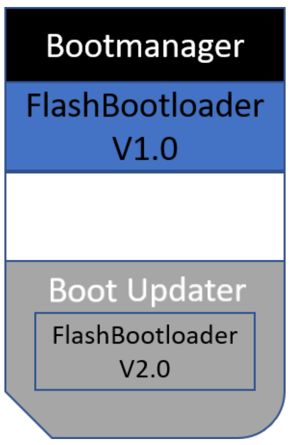


Figure 28. Flash Bootloader 1.0 is running: download Boot Updater in the application region

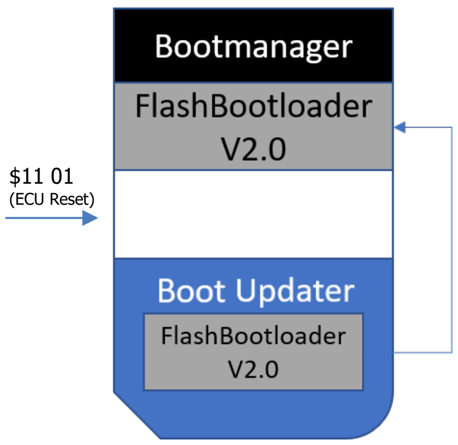


Figure 27. Boot Updater is running: update Bootloader

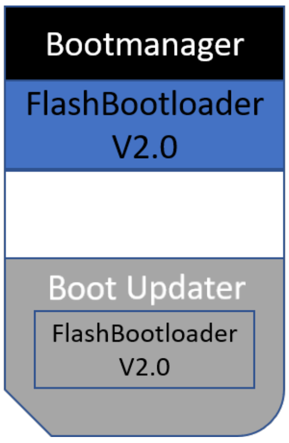


Figure 31. Flash Bootloader 2.0 is running

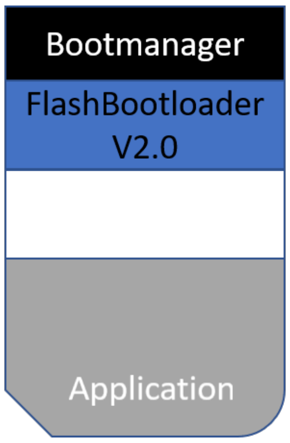


Figure 30. Flash Bootloader 2.0 is running: (re-)download Application

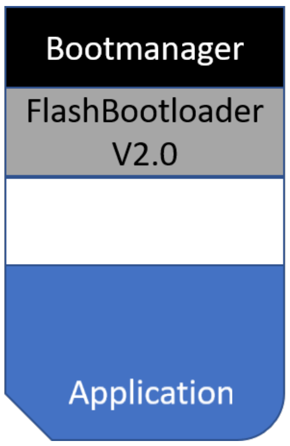


Figure 32. Application is running

When the Boot Updater is running (as shown in Figure 29) one or more UDS NRC $78 may be issued, depending on the flash bootloader size to update and the target underlying characteristics, affecting erase and write speed.

When the update is completed and the new Flash Bootloader is running, (as shown in Figure 30) it replies with $51 01 UDS response, to signal the end of the procedure.

Please note that the Bootmanager is required to have a reset-safe bootloader update, and thus it cannot be updated via UDS. Any customer-specific Bootmanager upgrade must be addressed by the customer itself with an appropriate means (e.g. program/debug port on the ECU PCB).

Your target will likely include a sample boot updater application, please refer to your RTA-FBL Standard Target Guide.

4 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 HW interface 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.

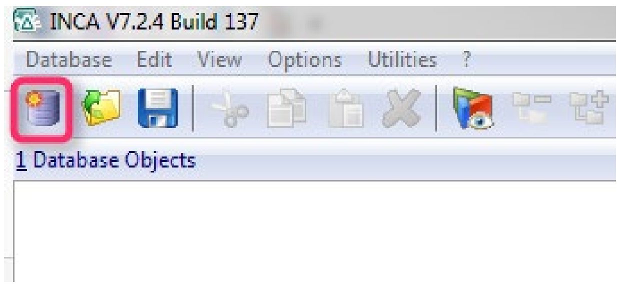


Figure 33: New database creation

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

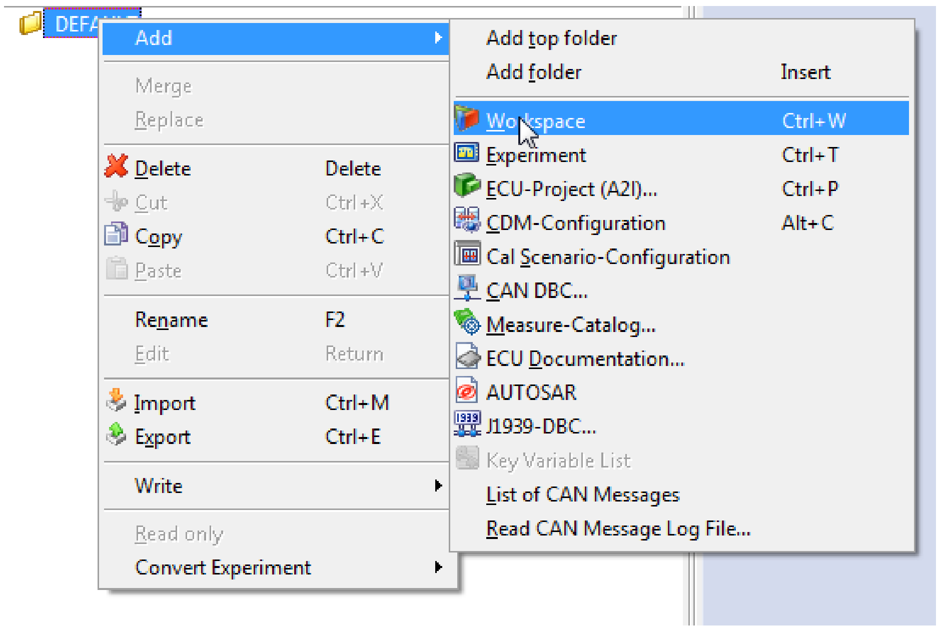


Figure 34: Adding a workspace

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

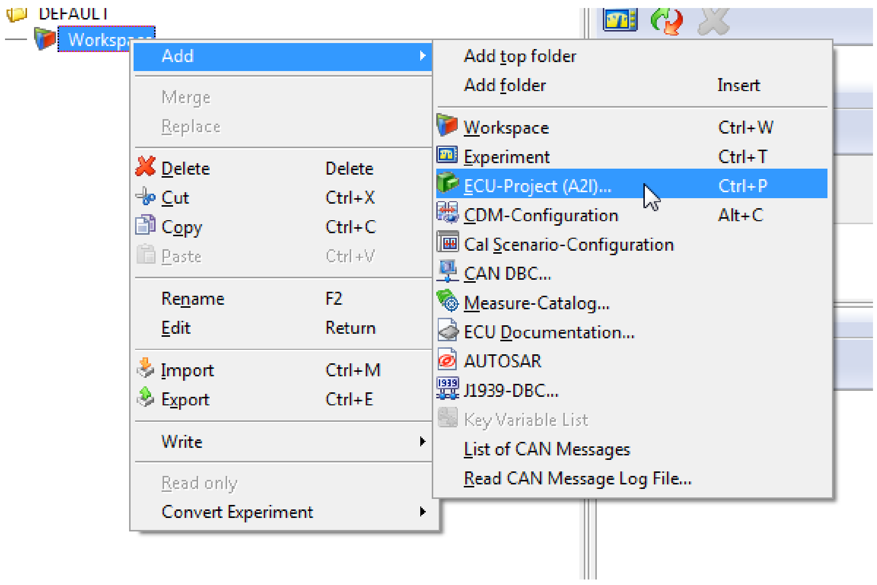


Figure 35: Adding an ECU project

From the dialog window navigate to the where the ProF script is located in your RTA-FBL Standard Instance (<output_location>\fbl\output\Tools\ProF\Install) and select the file “ECU_dummy.a2l”, and following this, from the same path, choose the file “FAKE_APPL.hex”.

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

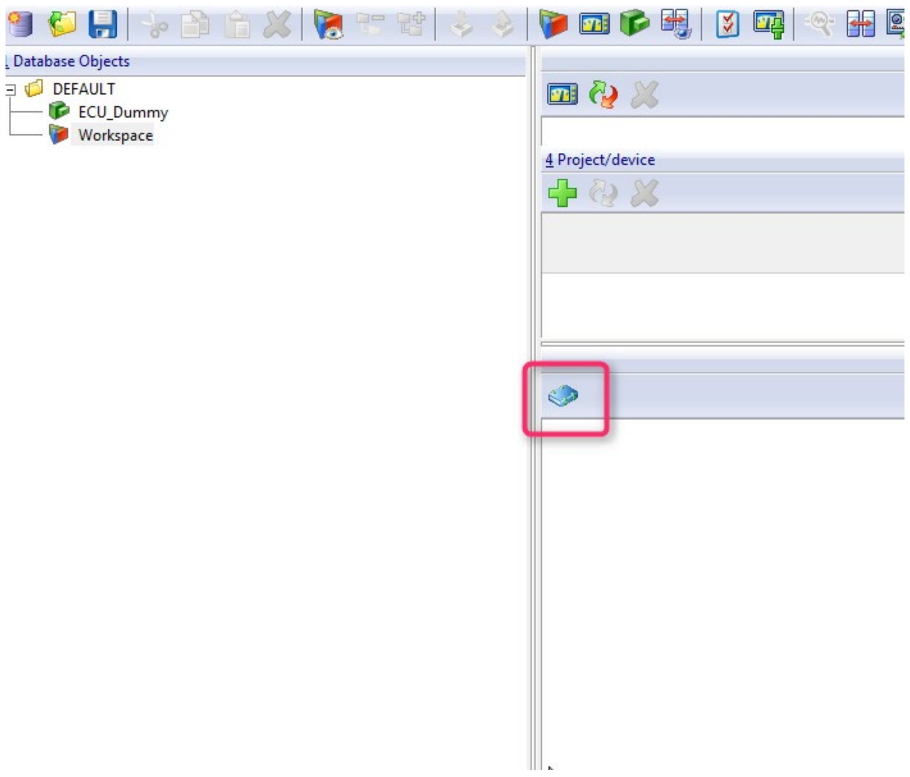
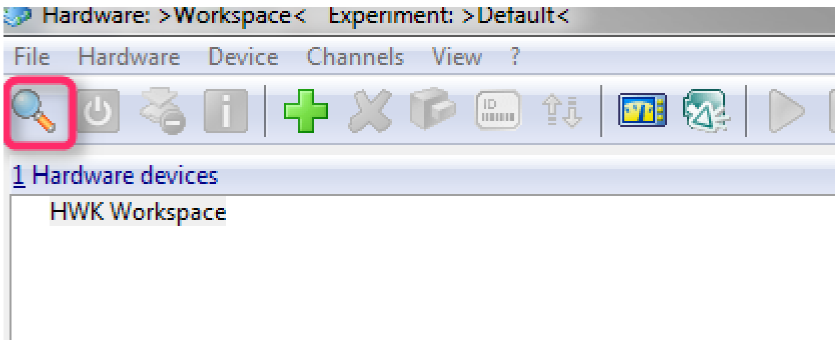
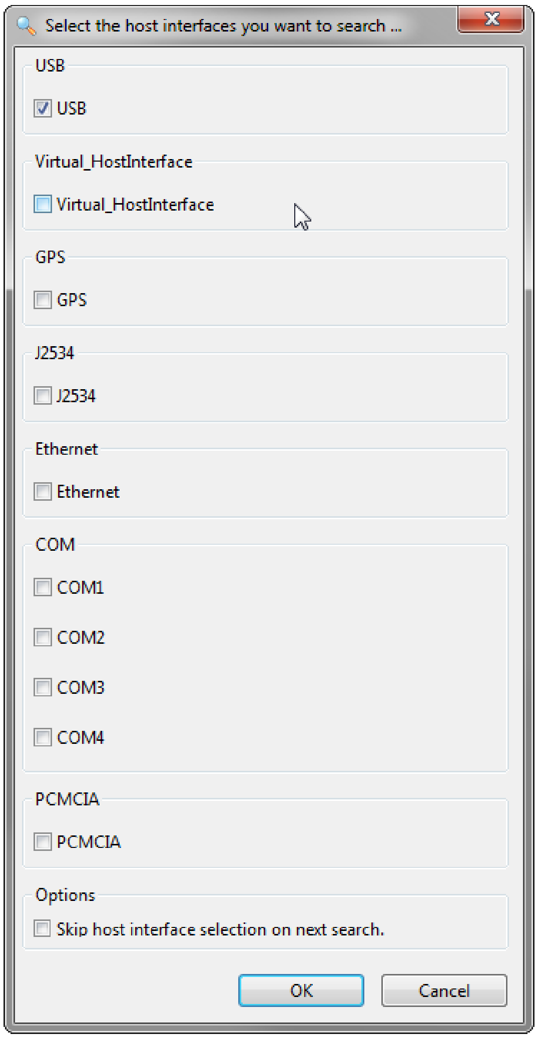


Figure 36: HW icon

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 39.


Hardware: >Workspace< Experiment: >Default<
File Hardware Device Channels View ?
1 Hardware devices
HWK Workspace


Select the host interfaces you want to search ...

USB
- [✓] USB

Virtual_HostInterface
- [ ] Virtual_HostInterface

GPS
- [ ] GPS

J2534
- [ ] J2534

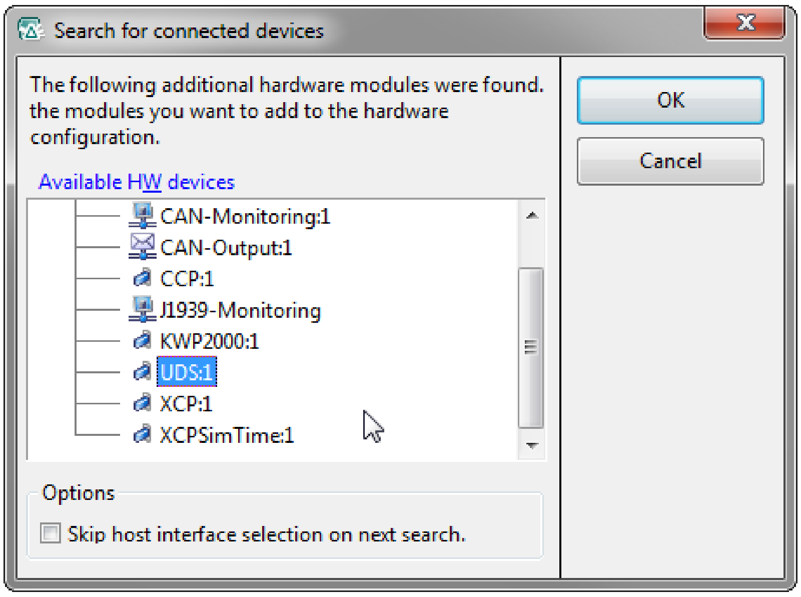
Ethernet
- [ ] Ethernet

COM
- [ ] COM1
- [ ] COM2
- [ ] COM3
- [ ] COM4

PCMCIA
- [ ] PCMCIA

Options
- [ ] Skip host interface selection on next search.

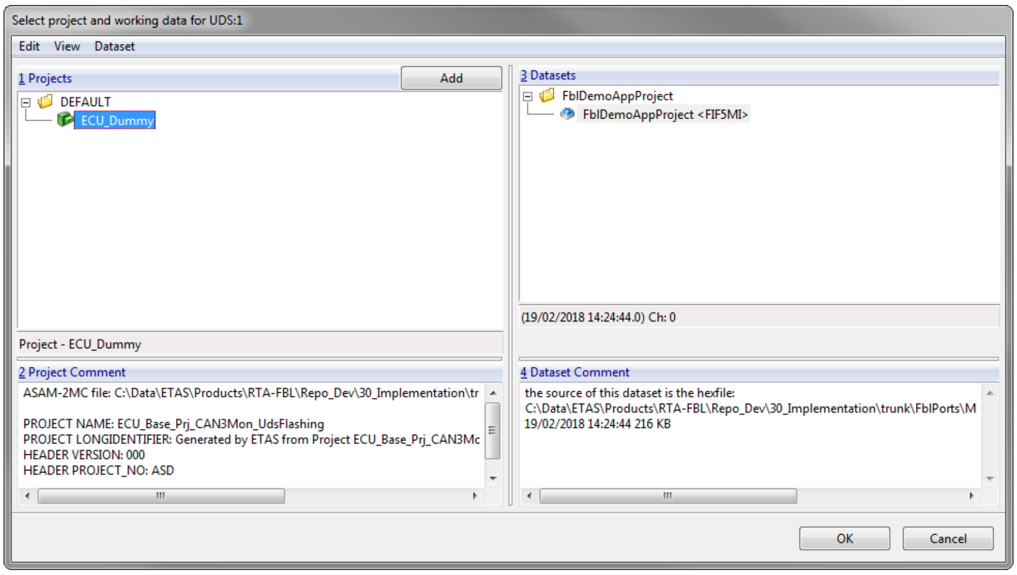
Figure 37: Search of HW interface



The following additional hardware modules were found. the modules you want to add to the hardware configuration.

☐ Skip host interface selection on next search.

Figure 38: Selection of UDS interface



(19/02/2018 14:24:44.0) Ch: 0

Project - ECU_Dummy

ASAM-2MC file: C:\Data\ETAS\Products\RTA-FBL\Repo_Dev\30_Implementation\tr

PROJECT NAME: ECU_Base_Prj_CAN3Mon_UdsFlashing
PROJECT LONGIDENTIFIER: Generated by ETAS from Project ECU_Base_Prj_CAN3Mc
HEADER VERSION: 000
HEADER PROJECT_NO: ASD

the source of this dataset is the hexfile:
C:\Data\ETAS\Products\RTA-FBL\Repo_Dev\30_Implementation\trunk\FblPorts\M
19/02/2018 14:24:44 216 KB

Figure 39: 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

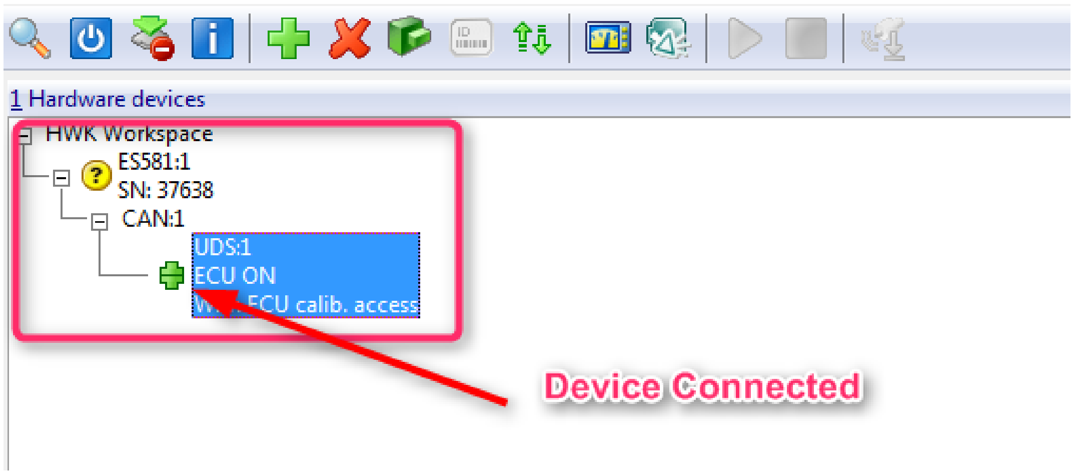


Figure 40: Device connection

Step3: Install ProF script

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

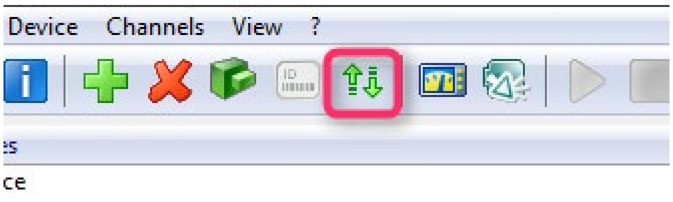
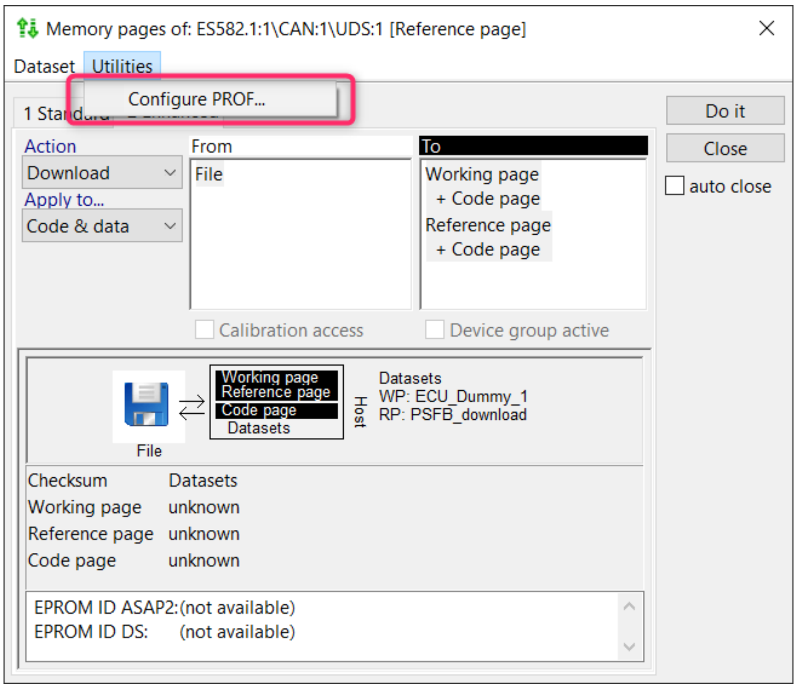


Figure 41: Manage memory page selection

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



Datasets
WP: ECU_Dummy_1
RP: PSFB_download

Checksum	Datasets
Working page	unknown
Reference page	unknown
Code page	unknown

EPROM ID ASAP2:(not available)
EPROM ID DS: (not available)

Figure 42: ProF configuration selection

In the new window, select Install and navigate to the path of RTA-FBL generated instance <output_location>\fbl\output\Tools\ProF\Install\Standard_FBL\

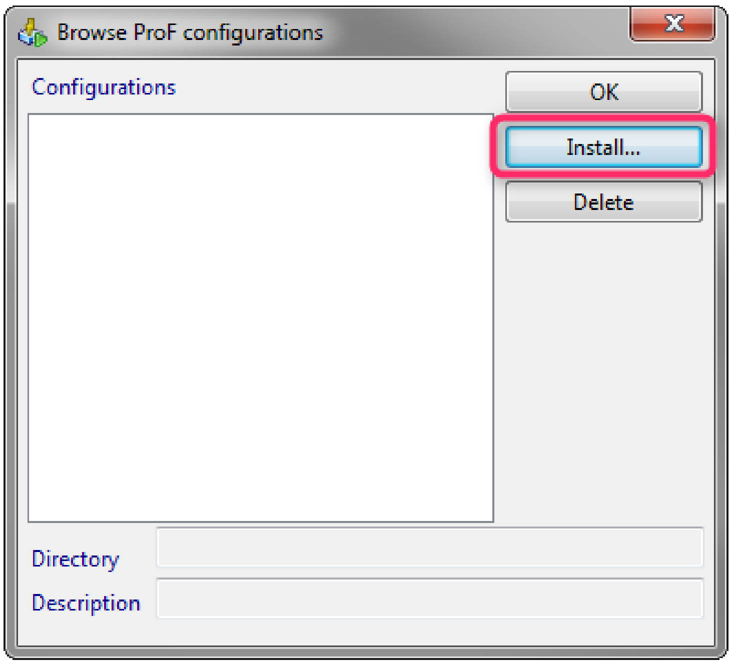
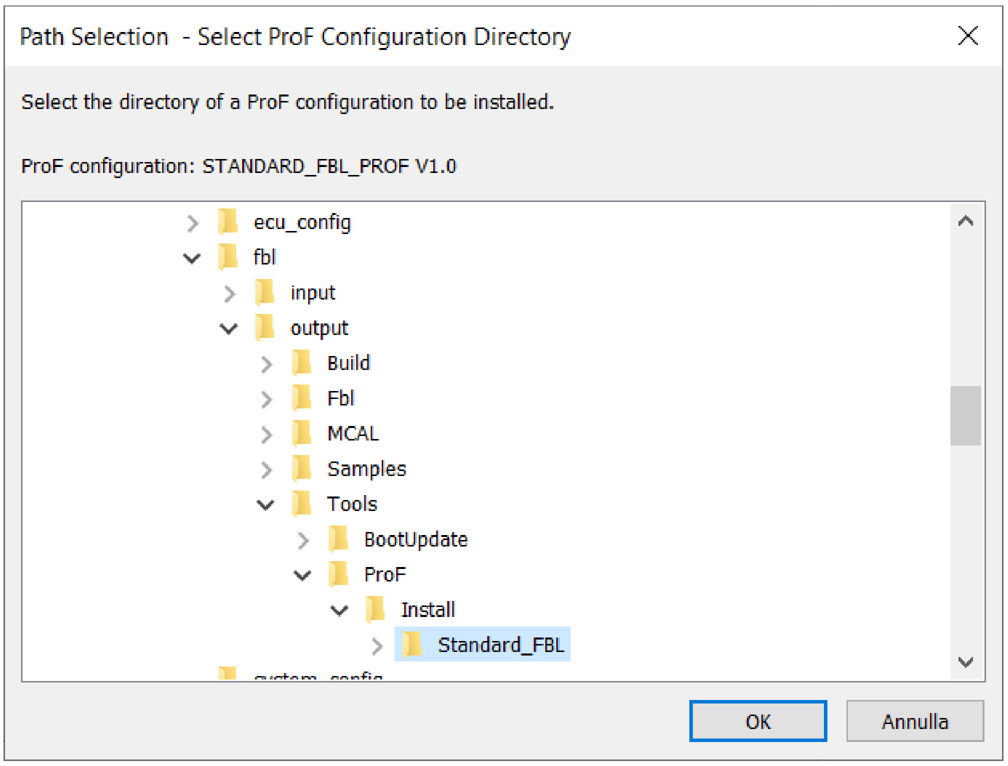


Figure 43: Install ProF script

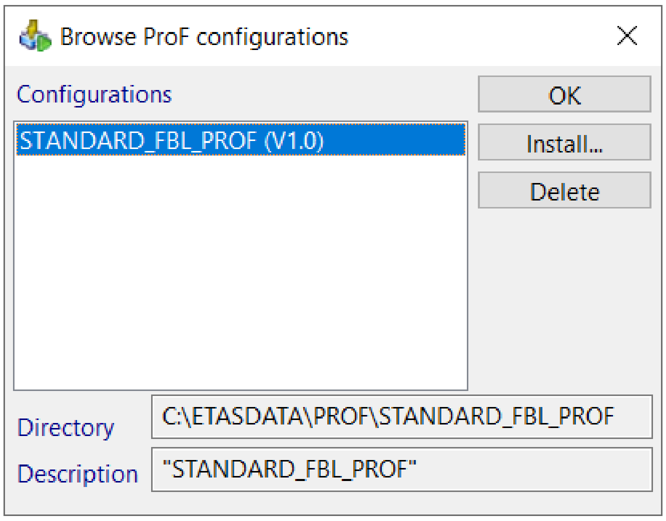


Select the directory of a ProF configuration to be installed.

ProF configuration: STANDARD_FBL_PROF V1.0

Figure 44: Selection of ProF script to install

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



Browse ProF configurations

Configurations: STANDARD_FBL_PROF (V1.0)

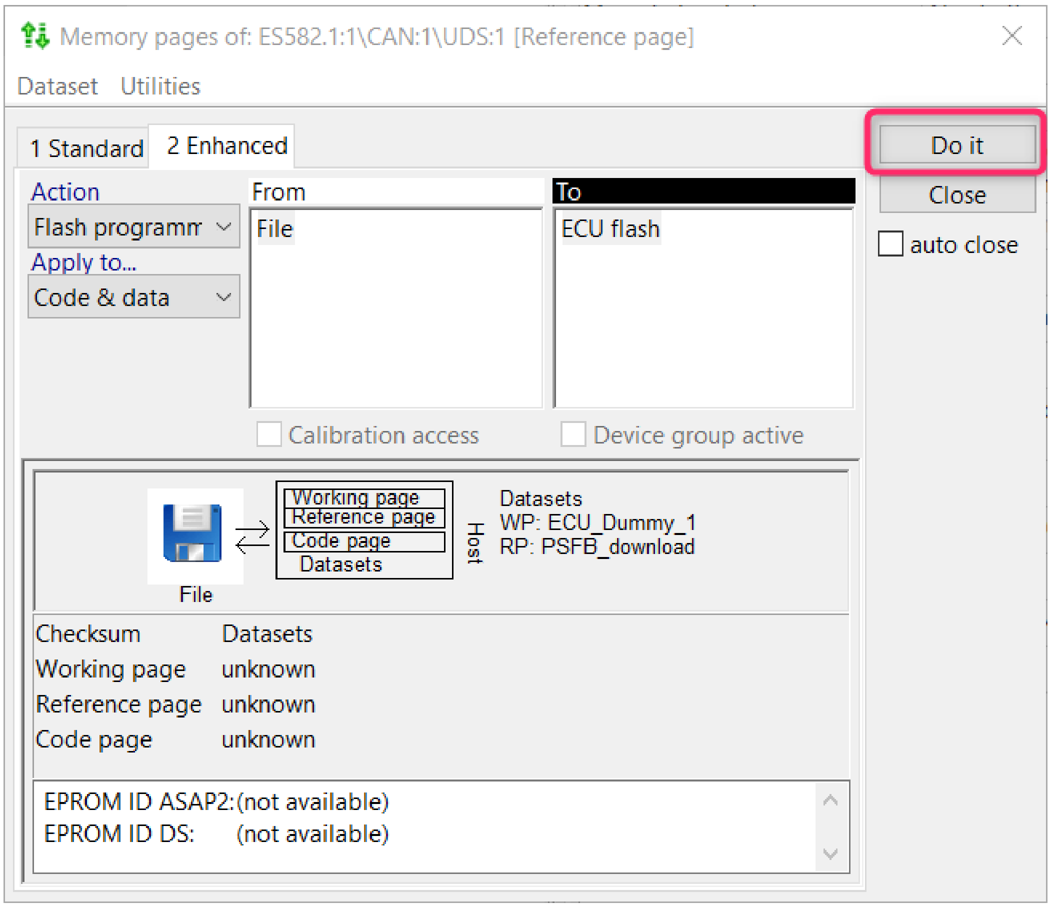
Directory: C:\ETASDATA\PROF\STANDARD_FBL_PROF

Description: "STANDARD_FBL_PROF"

Figure 45: 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".



Memory pages of: ES582.1:1\CAN:1\UDS:1 [Reference page]

Action: Flash programm

Apply to...: Code & data

From: File

To: ECU flash

Datasets
WP: ECU_Dummy_1
RP: PSFB_download

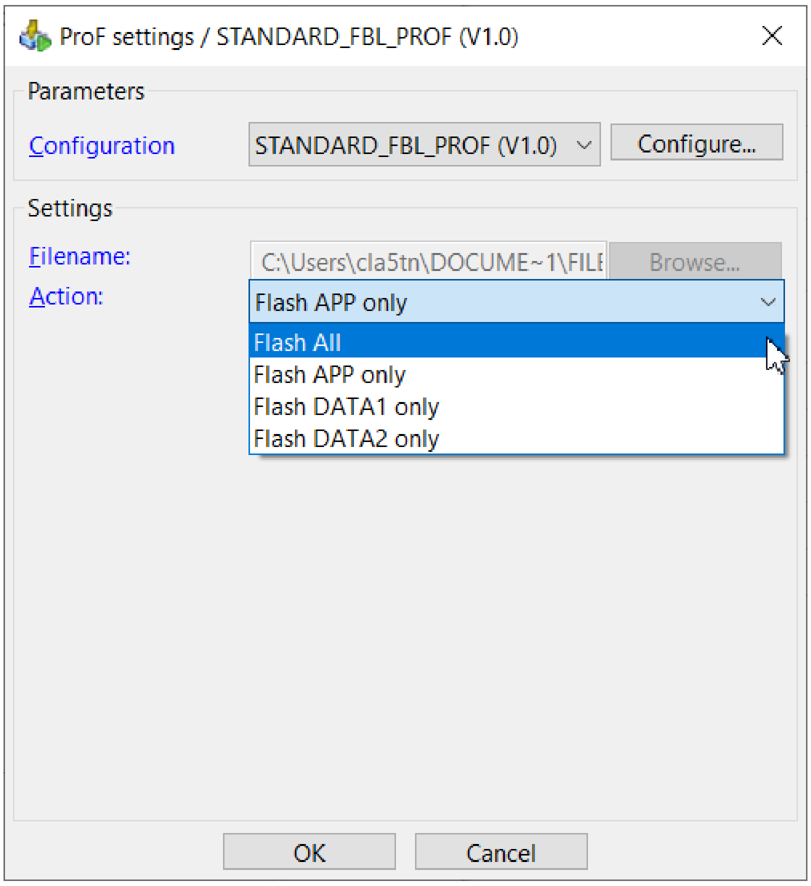
Checksum	Datasets
Working page	unknown
Reference page	unknown
Code page	unknown

EPROM ID ASAP2: (not available)
EPROM ID DS: (not available)

Figure 46: 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. The ProF will be created according to the regions you have configured: you can choose to reprogram only a specific region or all the regions.


ProF settings / STANDARD_FBL_PROF (V1.0)

Parameters

Configuration: STANDARD_FBL_PROF (V1.0)

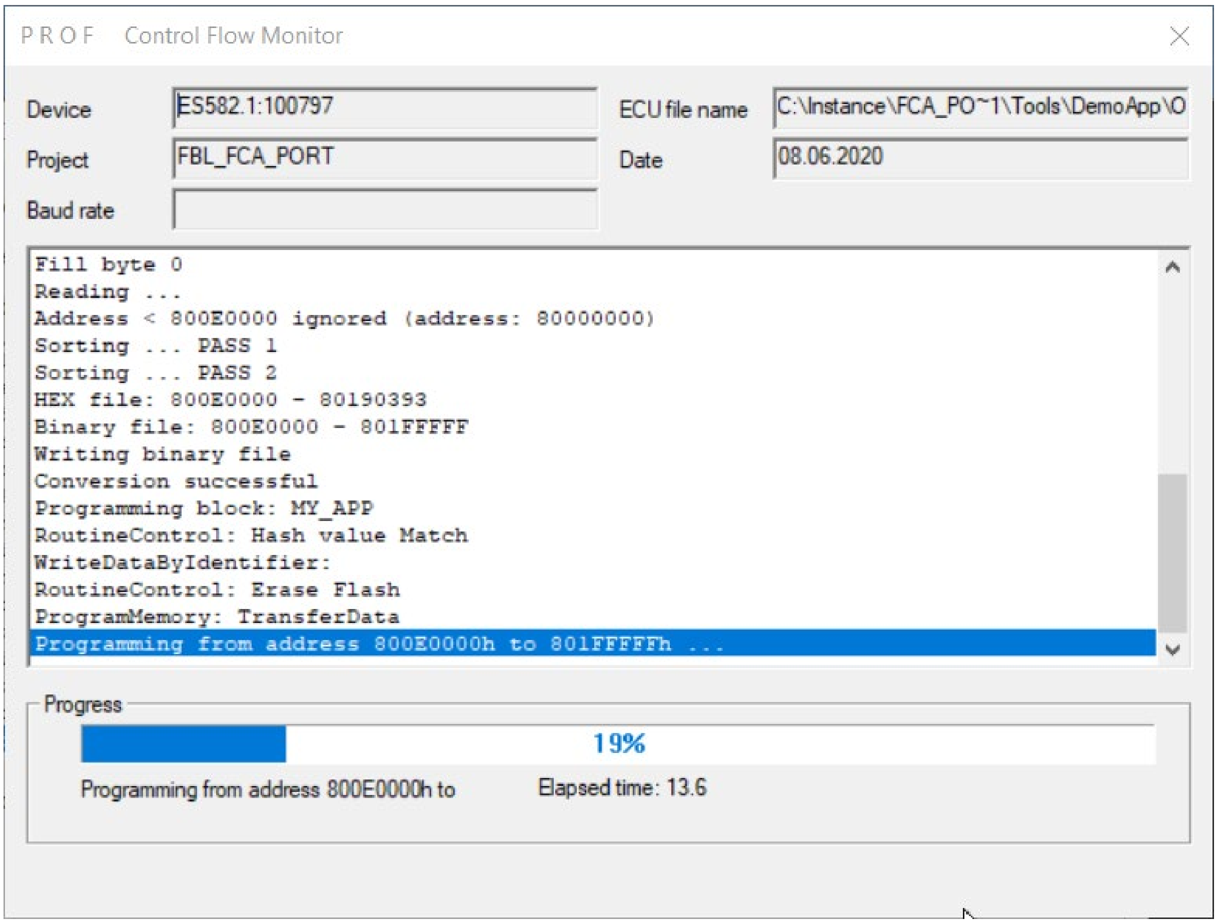
Settings

Filename: C:\Users\cla5tn\DOCUME~1\FILE

Action: Flash APP only

- Flash All
- Flash APP only
- Flash DATA1 only
- Flash DATA2 only

Figure 47: Option Selection



Device: ES582.1:100797
Project: FBL_FCA_PORT
Baud rate:
ECU file name: C:\Instance\FCA_PO~1\Tools\DemoApp\O
Date: 08.06.2020

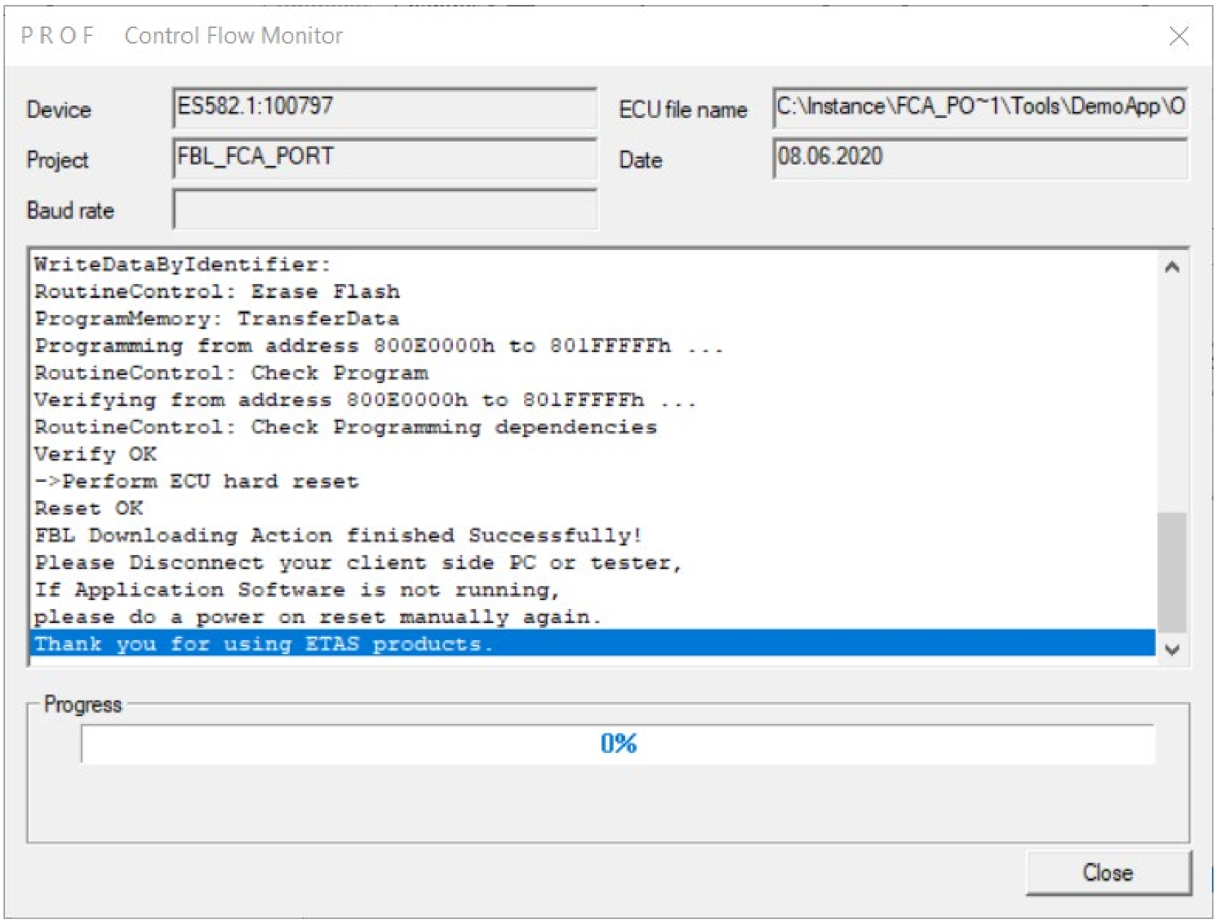
```
Fill byte 0
Reading ...
Address < 800E0000 ignored (address: 80000000)
Sorting ... PASS 1
Sorting ... PASS 2
HEX file: 800E0000 - 80190393
Binary file: 800E0000 - 801FFFFF
Writing binary file
Conversion successful
Programming block: MY_APP
RoutineControl: Hash value Match
WriteDataByIdentifier:
RoutineControl: Erase Flash
ProgramMemory: TransferData
Programming from address 800E0000h to 801FFFFh ...
```

19%

Programming from address 800E0000h to

Elapsed time: 13.6

Figure 48: Download in progress



Device: ES582.1:100797
ECU file name: C:\Instance\FCA_PO~1\Tools\DemoApp\O
Project: FBL_FCA_PORT
Date: 08.06.2020
Baud rate:

```
WriteDataByIdentifier:
RoutineControl: Erase Flash
ProgramMemory: TransferData
Programming from address 800E0000h to 801FFFFFh ...
RoutineControl: Check Program
Verifying from address 800E0000h to 801FFFFFh ...
RoutineControl: Check Programming dependencies
Verify OK
->Perform ECU hard reset
Reset OK
FBL Downloading Action finished Successfully!
Please Disconnect your client side PC or tester,
If Application Software is not running,
please do a power on reset manually again.
Thank you for using ETAS products.
```

Figure 49: Download Completed

4.1 Seed & Key Constants

A text file located in <output_location>\fbl\output\Tools\ProF\Install\Standard_FBL\ProF\keys.txt is created to store RTA-FBL configuration parameters FblSeedKeyConstant1 and FblSeedKeyConstant2.

This file is copied by the Prof during [INIT] section in the fixed location C:\ETASData\RTA-FBL\STANDARD_FBL, using the RTA-FBL output location as absolute path. If you have moved the Prof in a different path, ensure to manually copy over this file, since the batch copy will fail.

It is necessary to have the keys.txt file in the proper location (C:\ETASData\RTA-FBL\STANDARD_FBL) because it is needed by the SeedAndKey.dll to unlock the FlashBootloader during the reprogramming.

4.2 Application signature handling

If RTA-FBL configuration parameter FblSoftwareVerificationType equals "SIGNATURE", the Fbl expects the signature for software verification to be in the last 256 bytes of the downloaded region.

The Fbl public key, configured via the parameter FblSecurityPublicKey, should match the private key used for signature calculation.

Before downloading each region, INCA computes the signature of the data to be downloaded and writes the result to the last 256 bytes of the software, using the tool Signature.exe located within the Prof folder.

The private key used is stored into <output_location>\fbl\output\Tools\ProF\Install\Standard_FBL\ProF\PrivateKey.pem in PEM format and copied by INCA during Prof installation.

The private key provided within INCA Prof, matches the sample public key under C:\ETAS\RTA-FBL_1.0.0_Standard\Ports\Standard\Samples.

5 Privacy

5.1 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.

5.2 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.

5.3 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.

5.4 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.

6 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 contacts:

ETAS subsidiaries	www.etas.com/en/contact.php
ETAS RTA Technical Support Website	https://rtahotline.etas.com/
ETAS RTA Technical Support Email	rta.hotline@etas.com