

ETAS RTA-FBL_STLA v2.1.0



User Manual

Copyright

The data in this document may not be altered or amended without special notification from ETAS GmbH. ETAS GmbH undertakes no further obligation in relation to this document. The software described in it can only be used if the customer is in possession of a general license agreement or single license. Using and copying is only allowed in concurrence with the specifications stipulated in the contract.

Under no circumstances may any part of this document be copied, reproduced, transmitted, stored in a retrieval system or translated into another language without the express written permission of ETAS GmbH.

© Copyright 2024 ETAS GmbH, Stuttgart

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

ETAS RTA-FBL_STLA 2.1.0 - User Manual R01 EN – 11.2024

Contents

1	Introduction.....	6
1.1	Intended Use.....	6
1.2	Safety Information.....	6
1.3	Revision History.....	6
1.4	Definition and Abbreviations.....	12
1.5	References.....	13
1.6	About this Document.....	14
1.7	Chapter Description.....	14
2	Introduction to ETAS RTA-FBL.....	15
2.1	What is a Flash Bootloader?.....	15
2.2	What is RTA-FBL?.....	16
2.3	The Flash Tool (Tester).....	17
2.4	The OEM-defined Programming Sequence.....	17
2.5	Target Dependencies and the Flash Driver.....	17
2.6	Interaction with the Application using NvM.....	18
2.7	One and Two-Stage Bootloaders.....	18
2.8	FBL generation with the RTA-FBL ISOLAR-AB plugin.....	18
2.9	General architecture of RTA-FBL.....	20
2.10	Setting up your environment to generate an RTA-FBL instance.....	21
3	Installing RTA-FBL.....	22
4	The STLA Port.....	25
4.1	RTA-FBL_STLA Architecture.....	25
4.2	Stellantis Download Sequence.....	27
4.3	Creating and building an RTA-FBL instance.....	28
4.3.1	Project creation.....	29
4.3.2	Configuration and Generation of FBL and BSW.....	32
4.3.2.1	FblRegion.....	36
4.3.2.2	FblGeneral.....	38
4.3.2.3	FblCore.....	40
4.3.2.4	FblCan.....	41
4.3.2.5	FblSec.....	43
4.3.2.6	FblNvmBlock.....	43
4.3.2.7	FblNvmData.....	44

- 4.3.2.8 FblDid 45
- 4.3.2.9 FblFota 46
- 4.3.2.10 FblFotaRollbackRegion 47
- 4.3.3 Files created during generation 48
- 4.3.4 The RTA-FBL instance for the Dummy Target 49
 - 4.3.4.1 Dummy Target Memory Layout 50
- 4.4 Security Stack 51
 - 4.4.1 FBL callout for SecStack integration 51
 - 4.4.1.1 Startup 52
 - 4.4.1.2 Jump to application 52
 - 4.4.1.3 Periodic 53
 - 4.4.1.4 HTA suspension and activation 54
 - 4.4.1.5 Secure Download 54
 - 4.4.1.6 ADA - Challenge request 55
 - 4.4.1.7 ADA - Verify response and periodic certificate verification 56
 - 4.4.1.8 ADA - Certificate management 57
 - 4.4.1.9 ADA - Periodic 59
 - 4.4.1.10 FOTA - CSR and identity key generation 59
 - 4.4.1.11 FOTA - Sign data with identity key 61
 - 4.4.1.12 FOTA - Periodic 62
 - 4.4.2 FCA Security Manager (FSM) 62
 - 4.4.2.1 FSM CertStore, DisavowedCertificateList and TrustStore 62
 - 4.4.2.2 FSM Trusted Boot 64
 - 4.4.2.3 FSM Trusted Download 67
 - 4.4.3 STLA Security Manager (SSM) 68
 - 4.4.3.1 SSM CertStore, DisavowedCertificateList and TrustStore 68
 - 4.4.3.2 SSM Trusted Boot 70
 - 4.4.3.3 SSM Trusted Download 71
- 4.5 Supported targets 71
- 4.6 Integrator guidelines 72
 - 4.6.1 FBL: Memory Layout Adaptation 72
 - 4.6.2 FBL: User Functions 73
 - 4.6.2.1 Initialization 73
 - 4.6.2.2 Shutdown 74

4.6.2.3	Watchdog	74
4.6.2.4	Application validation	75
4.6.2.5	Software Identification Update	75
4.6.2.6	External memory reprogramming	76
4.6.2.7	Authenticated Diagnostic Access	78
4.6.2.8	Diagnostic	79
4.6.2.9	ECU Identity	80
4.6.2.10	FOTA Rollback	81
4.6.3	FBL: BSW adaptation	81
4.6.4	FBL: MCAL adaptation	82
4.6.5	FBL: OS adaptation	82
4.6.6	FBL: BLSM adaptation	83
4.6.7	Application Software: NvM layout adaptation	83
4.6.8	ASW: Boot Jump Handling	84
4.6.9	FBL: NvM adaption	87
4.6.10	FBL: DID adaption	90
4.7	Bootloader Update	91
4.8	Authenticated Diagnostic Access	94
4.8.1	Authenticated security access and Delay timer	94
4.8.2	Certificate expiration and monotonic counter	96
4.8.3	Role and Security policy management	97
4.9	FOTA Rollback	97
4.9.1	FOTA Rollback configuration	97
4.9.2	FOTA Rollback functional behavior	98
4.10	FOTA ECU Identity	100
5	Privacy	102
5.1	Privacy Statement	102
5.2	Data Processing	102
5.3	Data and Data Categories	102
5.4	Technical and Organizational Measures	102
6	ETAS Contact Addresses	103
6.1	ETAS HQ	103
6.2	ETAS Subsidiaries and Technical Support	103

1 Introduction

This user manual introduces the RTA-FBL port for STLA. It provides an overview of the RTA-FBL architecture and software design. It also provides detailed information of the STLA 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 Intended Use

This software release is qualified for series production usage.

The scope of the project is to implement a Flash Bootloader for STLA OEM. A Flash Bootloader is a piece of software that resides in a permanent partition of the ECU's flash memory. The purpose of Flash Bootloader is to establish the ECU entry point upon power up or power on reset and to enable flash programming of application software and calibration data via a diagnostic protocol on some physical channel. The Flash Bootloader implements the startup sequence when the ECU is powered up or after power on reset. Flash programming of the ECU is required when application software or calibration data is missing or an update to these is required.

1.2 Safety Information

The bootloader software is provided as 'commercial off-the-shelf' (COTS) software and is not certified according to ISO 26262:2018 or similar. If used in a safety relevant system developed according to ISO 26262:2018, the integrator should qualify the software according to ISO 26262-8:2018 clause 12 'qualification of software components'.

1.3 Revision History

Ver.	Author	Date	Change (Why, What)
0.1	Francesco Ficili	29/11/2018	First version.
1.0	Andrew Borg	03/02/2019	First release.
1.1	Daniele Cloralio	29/05/2020	STLA Port version
1.2	Daniele Cloralio	15/07/2020	Minor changes
1.3	Daniele Cloralio	22/10/2020	Typo and minor changes
1.4	Francesco Sfragara	05/11/2020	Minor changes in section 4.3 and 3
1.5	Daniele Cloralio	27/09/2022	Updates for release RTA-FBL STLA 2.0 Proto1

1.6	Mohamed Salem	15/05/2023	Updates for release RTA-FBL STLA 2.0 Proto2. Chapters: <ul style="list-style-type: none">- Updates in section 4.6.2 new user interfaces for ADA- Updates in section 4.6.7 to add new Nvm entry for ADA- Updates in section 4.6.8 to handle ADA feature during Jump from the ASW- Added section 4.8 for ADA feature description
-----	---------------	------------	--

1.7	Jacopo Filippi	18/06/2023	<p>Updates for release RTA-FBL STLA 2.0 Proto3. Features:</p> <ul style="list-style-type: none"> - Moved SecHal from target to INFRA - Added DID2001 user callback - Added FBL Did and NvM configuration to the plugin user interface. <p>FBL parameters:</p> <ul style="list-style-type: none"> - Deleted FbIDidEcuDiagnosticVariant, FbIDidSupplierId, FbIDidEcuDiagnosticVersion, FbIDidFdn, FbIDidAlgorithmIdReprogramming, FbIDidPtEslmHardwareNumber, FbIDidEbomEcuPartNumber, FbIDidCodepEcuPartNumber, FbIDidElsmEcuSoftwareNumber, FbIDidElsmEcuSwCalibrationNumber, FbIDidElsmEcuSwApplicationNumber, FbIDidCodepAssemblyPartNumber, FbIDidElsmEcuHardwareNumber, FbIDidSupplierEcuHardwarePartNumber, FbIDidSupplierEcuSoftwarePartNumber, FbIDidEbomAssemblyPartNumber, FbIDidHwSupplierId, FbIDidSwSupplierId, FbIDidEcuSerialNumber, FbIDidSupplierManEcuSwVersion, FbIDidSupplierManEcuHwVersion, FbIDidPolicyType, FbIDidErotan (replaced by FbIDid) - Added FbIDid (Now user can configure DIDs relevant parameter in the FBL module) - Added FbINvmBlock (Now user can configure the NvM blocks relevant parameter in the FBL module) - Added FbINvmData (Now user can configure the NvM data relevant parameter in the FBL module) <p>Chapters:</p> <ul style="list-style-type: none"> - Added chapter 4.6.9 FBL: NvM adaption - Added chapter 4.6.10 FBL: Did adaption
-----	----------------	------------	---

1.8	Jacopo Filippi	29/09/2023	<p>Updates for release RTA-FBL STLA 2.0 Proto4.</p> <p>Features:</p> <ul style="list-style-type: none"> - Added Rollback - Added Ecu Identity <p>FBL parameters:</p> <ul style="list-style-type: none"> - Added FblFotaRollback - Added FblFotaRollbackRegion - Added FblEraseMaxSizePerCycle <p>Chapters:</p> <ul style="list-style-type: none"> - Updated 4.3.2.3 FblCore (Added FblFotaRollbackRegion) - Added 4.3.2.9 FblFota - Added 4.3.2.10 FblFotaRollbackRegion - Updated 4.6.2.6 External memory reprogramming (Added UserFlash_FlashRead) - Updated 4.6.2.8 Diagnostic (Added new callouts for DID) - Added 4.6.2.9 ECU Identity - Added 4.6.2.10 FOTA Rollback - Updated 4.6.7 Application Software: NVM layout adaptation - Updated 4.6.10 FBL: DID adaption (Added Fbl_Port_<DID_SHORT_NAME>ReadFunc) - Added 4.9 FOTA Rollback - Added 4.10 FOTA ECU Identity
-----	----------------	------------	--

1.9	Jacopo Filippi	18/12/2023	<p>Updates for release RTA-FBL STLA 2.0 Proto5.</p> <p>Features:</p> <ul style="list-style-type: none"> - Added DCL support and DID 0x2031 - Updated Rollback - Added F1AA <p>Chapters:</p> <ul style="list-style-type: none"> - Added 4.4.1 CertStore, DisallowCertificateList and TrustStore - Added 4.4.2 Authenticated Boot - Added 4.4.3 Authenticated Download - Updated 4.6.2.10 FOTA Rollback to add new information on ISOLAR configuration and the description of the updates. - Update 4.6.2.8 Diagnostic with F1AA informations. - Update 4.8.1 Authenticated security access and Delay timer. Failure counter is stored in NvM.
1.10	Jacopo Filippi	08/03/2024	<p>Updates for release RTA-FBL STLA 2.0 Proto6.</p> <p>Features:</p> <ul style="list-style-type: none"> - Added "Extended reset" feature - Mod Secure Boot - Del Dids 0xF1A7, 0x100A and 0x100B - Mod Rollback - Mod Erase <p>Chapters:</p> <ul style="list-style-type: none"> - Mod 4.4 Security Stack: added subchapters 4.4.1 FBL callout for SecStack integration, 4.4.2 CertStore, DisallowedCertificateList and TrustStore - Mod 4.9 FOTA Rollback with new features.

2.0.0	Jacopo Filippi	02/07/2024	<p>RTA-FBL_STLA qualified release</p> <p>Chapters:</p> <ul style="list-style-type: none"> - Mod Stellantis Download Sequence (Erased force pending response) - Mod Bootloader Update (Added information about boot updater process) - Mod FOTA ECU Identity (Added information about ECDSA signature) - Del <i>How to Flash the ECU with INCA and the ProF Script</i> (INCA ProF is no more delivered with the plugin)
2.1.0_Proto1	Margherita Capodieci	31/07/2024	<p>Updates for release RTA-FBL STLA 2.1.0_Proto1.</p> <p>Features:</p> <ul style="list-style-type: none"> - Added Standard CAN ID support <p>FBL parameters:</p> <ul style="list-style-type: none"> - Added FblCanType <p>Chapters:</p> <ul style="list-style-type: none"> - Mod FblCan (Added FblCanType)

2.1.0_Proto2	Jacopo Filippi	29/10/2022	<p>Updates for release RTA-FBL STLA 2.1.0_Proto2.</p> <p>Features:</p> <ul style="list-style-type: none"> - Added SSM 3.0.0 support - Added External region verification - Added DIDs: 0x2951, 0x2955 <p>FBL parameters:</p> <ul style="list-style-type: none"> - Del FblSecTsMainAddress - Del FblSecTsBackupAddress - Add FblSecStackType - Add FblSleepWakeupJumpToAppDelay - Add FblFlashBufferSize - Mod FblRegionType parameter of regions. - Add FblRegionSecTrustedBootMode to FblRegion - Add FblRegionSecCompatibilityId to FblRegion <p>Chapters:</p> <ul style="list-style-type: none"> - Mod FblRegion (Added parameters) - Mod FblGeneral (Added parameters) - Mod FblSec (Mod parameters) - <p>Mod Security Stack (Added SSM support)</p> <ul style="list-style-type: none"> • Mod External memory reprogramming
2.1.0	Jacopo Filippi	22/11/2024	RTA-FBL_STLA qualified release

1.4 Definition and Abbreviations

Term/Abbreviation	Definition
ADC	Analogue to Digital Convertor
ADA	Authenticated Diagnostic Access
AR	AUTOSAR
Application Software (ASW)	This is the software that executes the control logic of the ECU
AUTOSAR	AUTomotive Open System Architecture
BLSM	Bootloader State Manager

BSW	Basic Software
CAN	Controller Area Network
CAN FD	CAN Flexible Datarate
CS	Cert Store
CSR	Certificate Signing Request
CRC	Cyclic Redundancy Code - a CRC module is provided in RTA-BSW
Dcm	Diagnostic Communication Manager
DID	Data IDentifier
DCL	Disallowed Certificate List
DLL	Dynamic Link Library
ECU	Electronic Control Unit
FBL	Flash Bootloader
FOTA	Firmware Over The Air
Fee	Flash EEPROM Emulation
FW	Firmware
HW	Hardware
ISR	Interrupt Service Routine
MCAL	Micro-Controller Abstraction Layer
NvM	Non-Volatile Memory
OS	Operative System
RTA-x	The ETAS suite of embedded SW products
SFB	Signer Firmware Block
SFBH	Signer Firmware Block Header
SFBD	Signer Firmware Block Data
STLA	Stellantis
S&K	Seed And Key
SW	Software
TB	Trusted Boot
TBT	Trusted Boot Table
TS	Trust Store
UDS	Unified Diagnostic Services

1.5 References

Ref.	Document Name	Ver.
[1]	CS.00101_ECU FLASH Reprogramming Requirements	Rev. D
[2]	CS.00102_Standardized Diag Data	Rev. F
[3]	CS.00092_AUTHENTICATED DIAGNOSTICS ACCCESS	Rev. B

[4]	FBLDid_DefaultValues.pdf	2.1.0
[5]	FBLNvmBlock_DefaultValues.pdf	2.1.0
[6]	FBLNvmData_DefaultValues.pdf	2.1.0
[7]	SD.00078_Certificate_Formats.pdf	Rev. C
[8]	SD.00015_03_DCL	Rev. 0

1.6 About this Document

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

1.7 Chapter Description

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

2 Introduction to ETAS RTA-FBL

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

2.1 What is a Flash Bootloader?

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

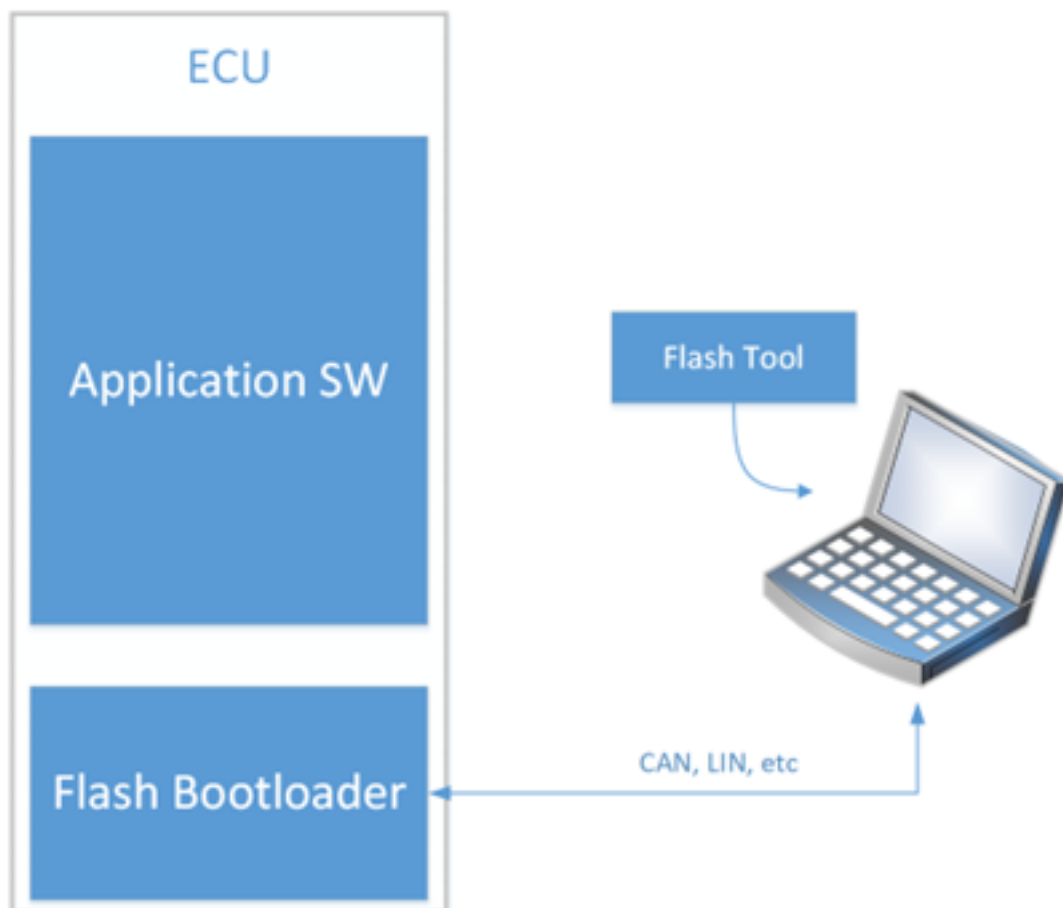


Figure 2.1. High level flashing process

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

SW already present in the ECU. This decision depends on two items of state in the ECU: whether a reprogramming request flag has been set by the Application SW before the last reset, and whether the Application SW currently programmed in the ECU is valid.

A classic boot loading sequence showing this decision is depicted in Figure 2.2. Note that the Application SW is only started if the Application SW is valid and the reprogramming request flag is not set. In any other case, the FBL enters the Bootloader state and communicates with the tester to reprogram the ECU.

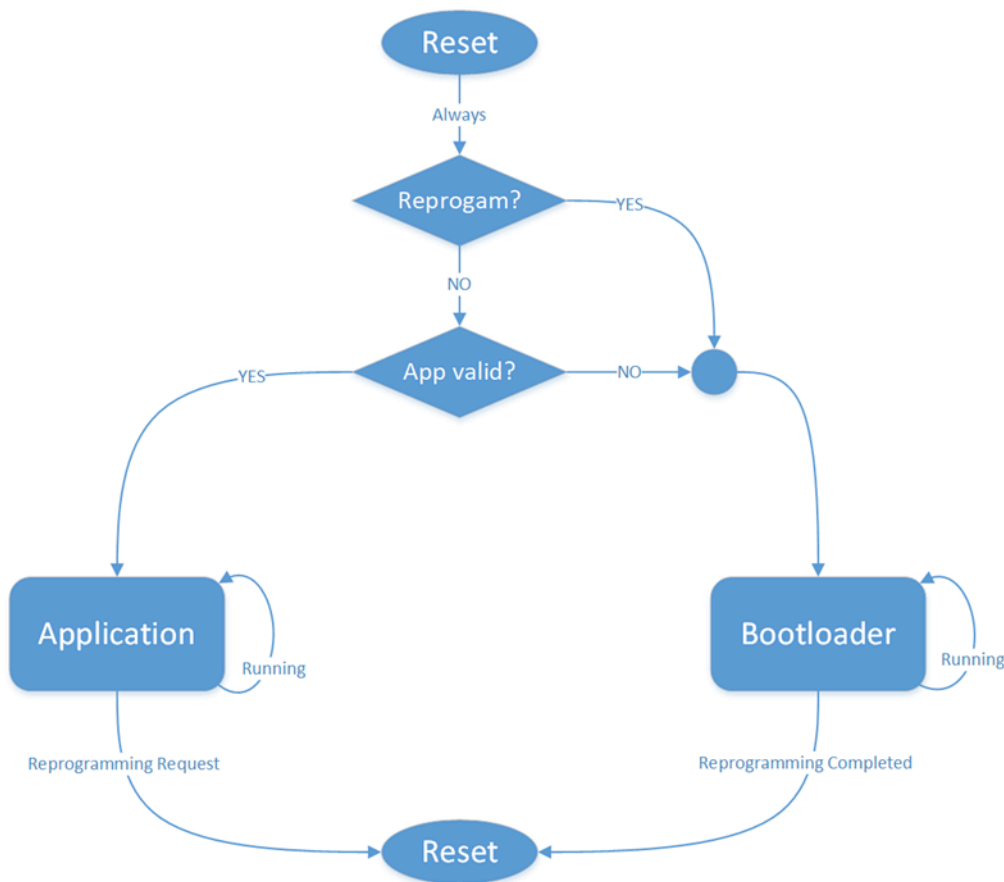


Figure 2.2.Boot loading flowchart

2.2 What is RTA-FBL?

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

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

- MCAL: provided by silicon vendor

- BSW: provided by ETAS (RTA-BSW)

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

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

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

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

2.3 The Flash Tool (Tester)

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

2.4 The OEM-defined Programming Sequence

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

For example, the RTA-FBL port that implements [1] supports all the targets described in the section Supported targets.

2.5 Target Dependencies and the Flash Driver

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

part of the MCAL.

2.6 Interaction with the Application using NvM

A Bootloader and the Application Software may need to share data. For example, a Tester may read or write data such as the ECU serial number both when the ECU is running in boot-loader mode and when running its Application Software (e.g. by using UDS ReadDataByIdentifier 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 the section The STLA Port.

2.7 One and Two-Stage Bootloaders

There are two broad models for bootloaders and the model type for the boot-loader described in [1] is described in more detail in The STLA Port.

- 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:
 - The Primary Bootloader can in principal be smaller because it does not need to include the code to write to flash (although space savings will be limited in practice if the Primary Bootloader also needs to include a flash driver to write to non-volatile memory implemented with flash).
 - 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.
 - The Secondary Bootloader can be used to work around bugs in the boot-loader installed on the ECU when it was manufactured.

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

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

An instance of ETAS's FBL is generated based on the chosen OEM specification

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

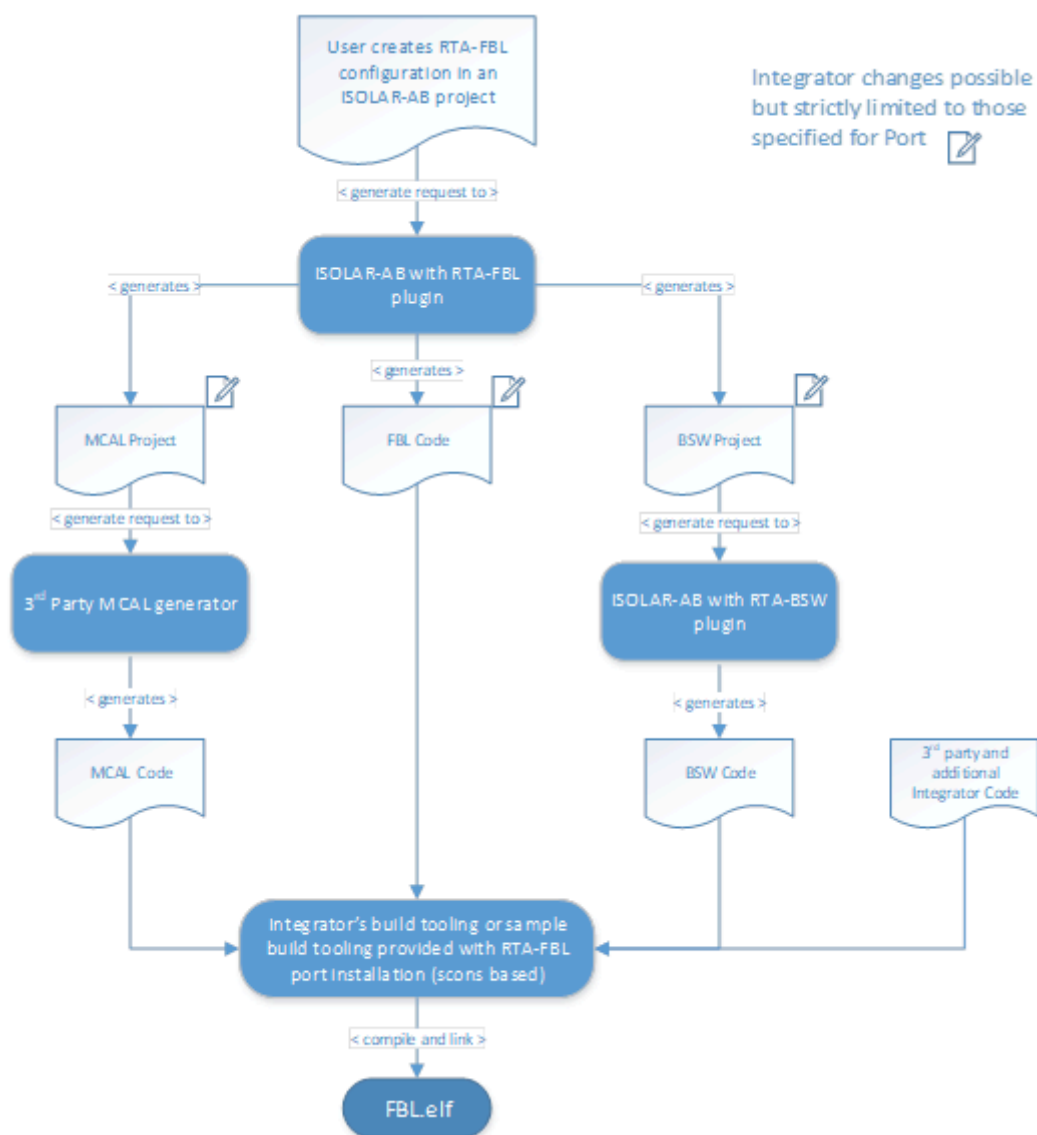


Figure 2.3. The process of generating an RTA-FBL instance

The tool process for generating an RTA-FBL instance is shown in Figure 2.3.

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

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



WARNING

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 the section The STLA Port for information relevant to your port.

2.9 General architecture of RTA-FBL

An instance of RTA-FBL consists of five types of module as shown within the complete RTA-FBL architecture in Figure 2.4.

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 Integrator guidelines. 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 FBL generation with the RTA-FBL ISOLAR-AB plugin, 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 2.4. 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 The STLA Port, including the folder structure of a generated RTA-FBL instance that contains the code for the modules in Figure 2.4.

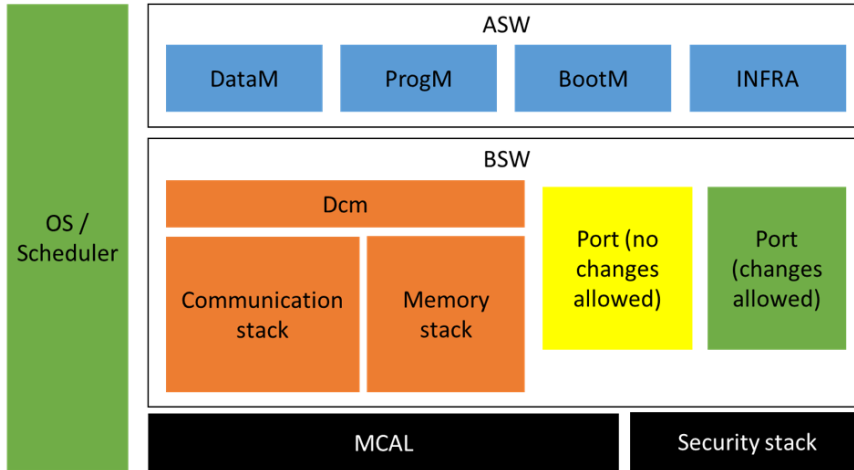


Figure 2.4. General architecture of an RTA-FBL instance

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

In order to generate an instance of RTA-FBL, you will need to install the tools shown in Table 2.1. 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 STLA FBL Target Guide.

Table 2.1. Tool versions

Tool Name	Version	Description
RTA-CAR	9.2.1	RTA-FBL configurator tool.
RTA-BSW	5.1.0	AUTOSAR BSW code generation tool.
RTA-FBL STLA Port	2.1.0	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 Installing RTA-FBL

This section describes the installer for RTA-FBL. As noted in Setting up your environment to generate an RTA-FBL instance, you need to install this package in addition to ISOLAR-AB. This installer is described further in this section. In order to install RTA-FBL, follow the instructions below. At the end of this installation, the PC needs to restart.

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

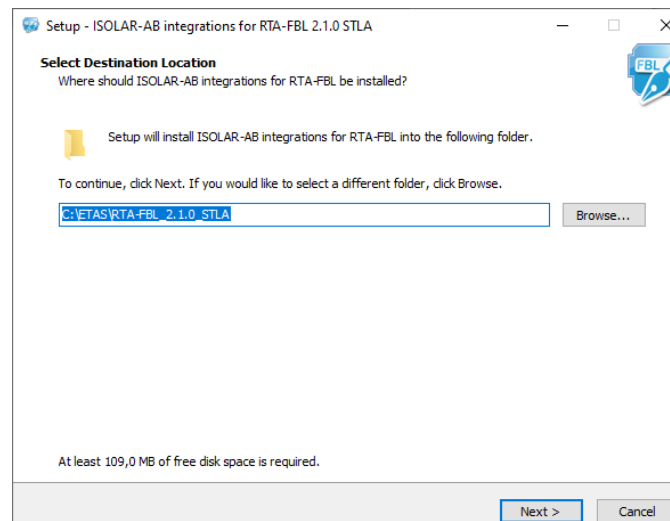


Figure 3.1. Welcome window

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".

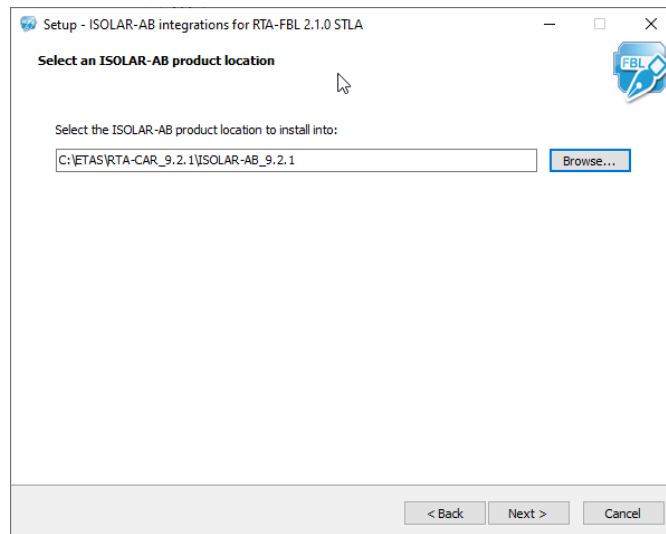


Figure 3.2.ISOLAR integration

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

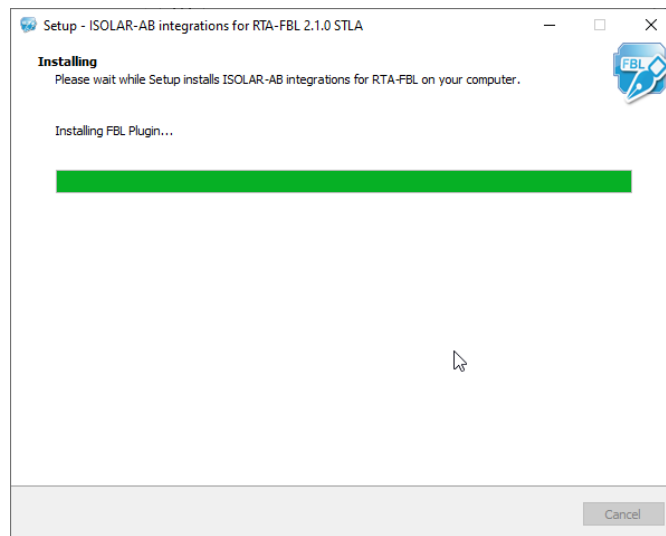


Figure 3.3.Installation Ongoing

Step 4: Once the installation completes, click on "Finish" to close the installer.

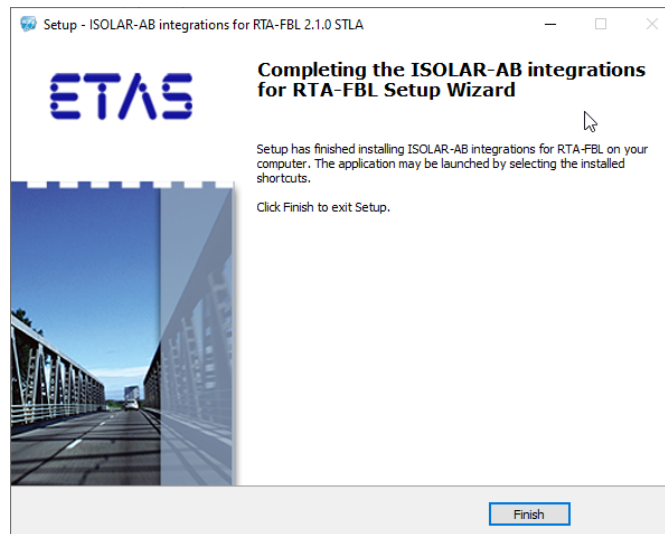


Figure 3.4. Installation finished

4 The STLA Port

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

4.1 RTA-FBL_STLA Architecture

Figure 4.1 provides a high-level view of RTA-FBL architecture for Stellantis. The communication, memory and diagnostic stacks are based on RTA-BSW and support the AUTOSAR architecture and methodology for source code configuration and generation. The rest of the components, except for the MCAL, are provided by ETAS and Escrypt. The modules that comprise the RTA-FBL instance for a Stellantis 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 Stellantis-specific port module (in yellow): this is generated by the RTA-FBL ISOLAR-AB plugin when the Stellantis port is selected. This module implements the bootloader features that are specific to the Stellantis specification [1].

Some of the api contained in this modules shall be populated by the integrator. These api are described in the chapters FBL User Functions and FBL callout for SecStack integration

4. The Stellantis-specific sample modules (in green): these are generated by the RTA-FBL ISOLAR-AB plugin when the Stellantis port is selected and may be modified by the integrator:
 - The OS is a basic cyclic scheduler that can be replaced by any other scheduler (e.g. a fully-configured RTA-OS) as long as the calls to the relevant main functions are made at the correct periods as in the provided samples. See FBL: OS adaptation for further details on how to adapt this module.
 - The BLSM contains code for initializing the Bootloader. Changes can be made here by the integrator if other modules are to be integrated (e.g. other BSW modules) but changes should not be made to the functions that interact with the core FBL modules. See FBL: BLSM adaptation for further details on how to adapt this module.
5. Third-party software modules (in red): these are security modules provided by Escrypt that should not be modified by the integrator. These modules are not generated with RTA-FBL and shall be added manually. If Escrypt solution (CycurHSM and Stellantis Wrapper) is not used, the integrator shall add a compatible security stack with the proper integration code.

- The MCAL modules (in black); the modules shown are those required by the Stellantis 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.

Note: Fee and FlsLoader module may be present or not based on your target.

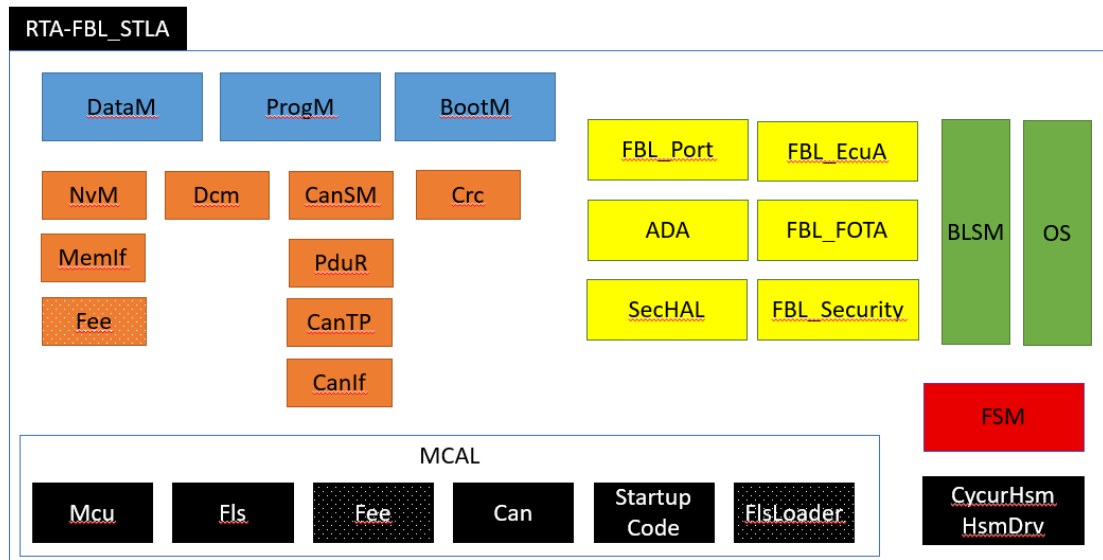


Figure 4.1. Stellantis architecture of an RTA-FBL instance

4.2 Stellantis Download Sequence

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

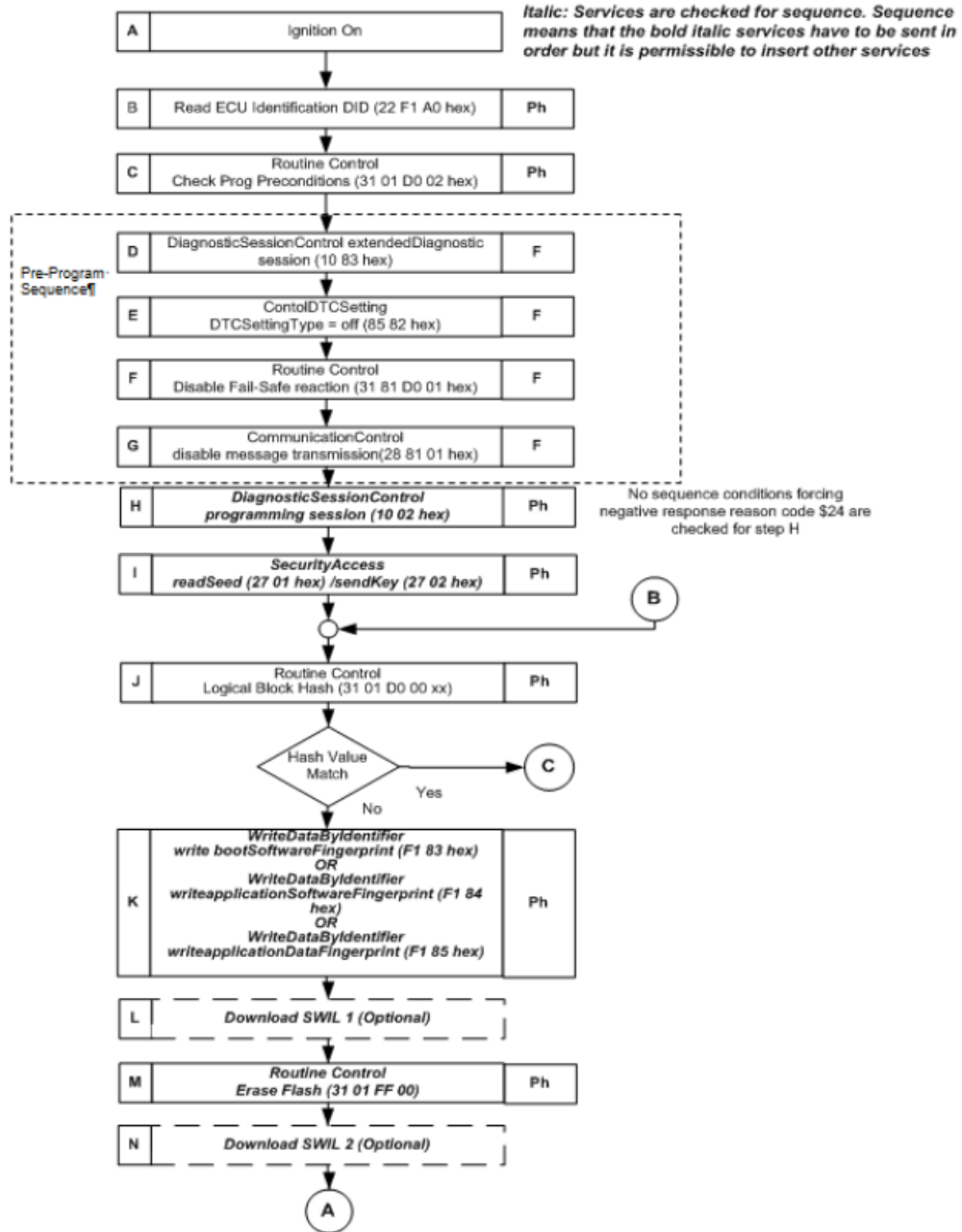


Figure 4.2.Flash Download Sequence Part 1

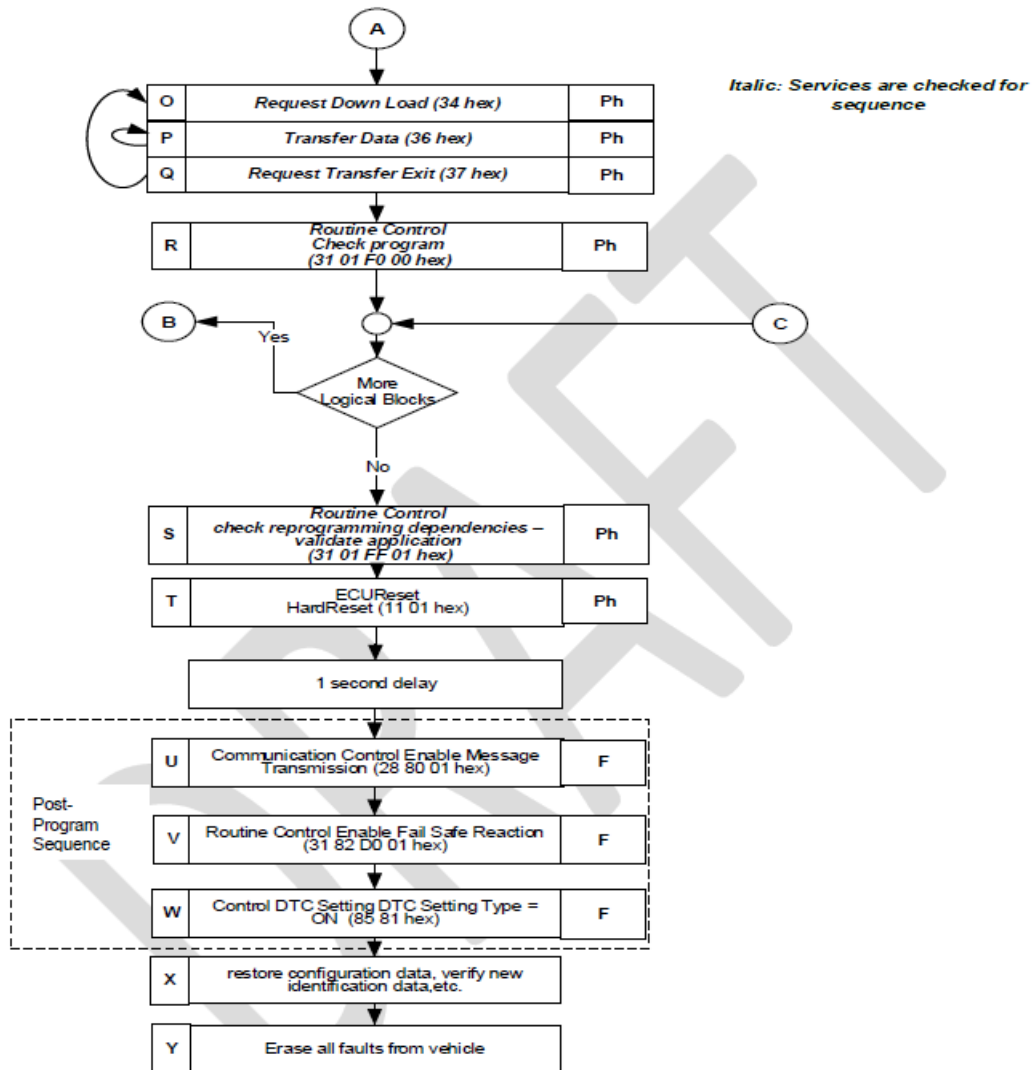


Figure 4.3. Flash Download Sequence Part 2

Particular attention should be paid to service \$FF00 (Routine Control Erase Flash). This operation is blocking for the Bootloader and, depending on the target, might take some time. In order to avoid a timeout, the ECU will force a Pending Response (\$78) on the communication bus every 4 seconds until the end of the operation.

4.3 Creating and building an RTA-FBL instance

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

4.3.1 Project creation

A new FBL project is created in ISOLAR-AB. As shown in Figure 4.4, create a new RTA-CAR project by clicking the “New Project” dropdown button and selecting “RTA-CAR Project”.

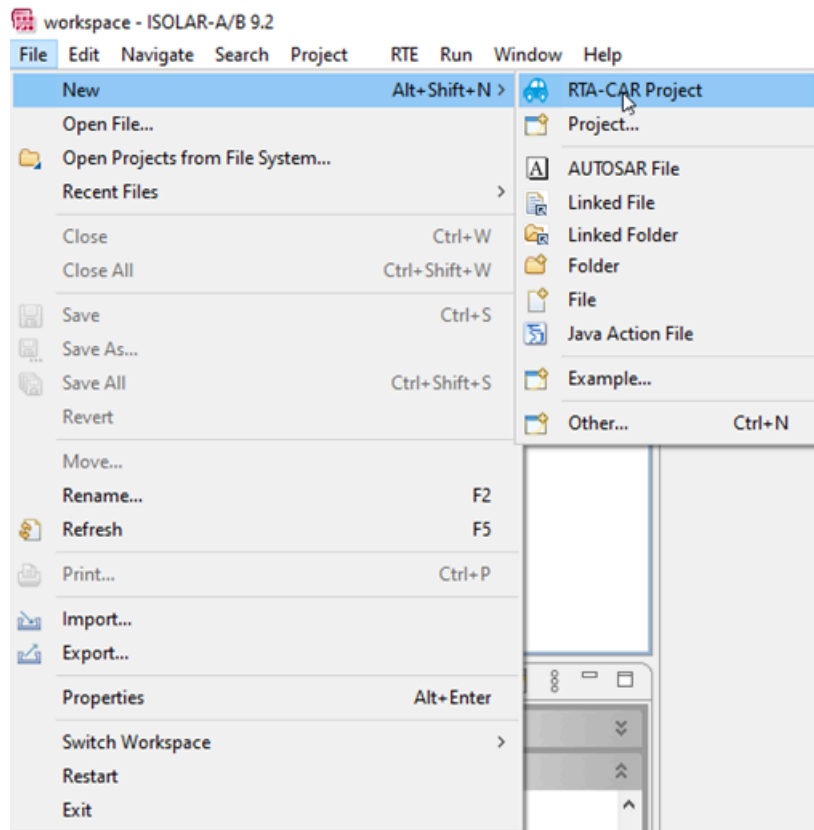


Figure 4.4. RTA-CAR project creation

If RTA-CAR Project is not present, select “Project” and search for “RTA-CAR Project” in the new window, as shown in Figure 4.5.

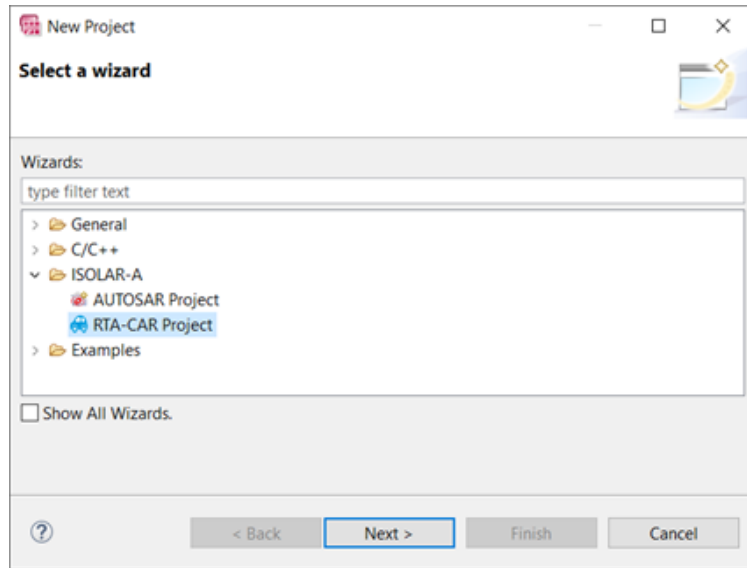


Figure 4.5.RTA-CAR project

In the New RTA-CAR Project window, select RTA-CAR bootloader project as shown in Figure 4.5.

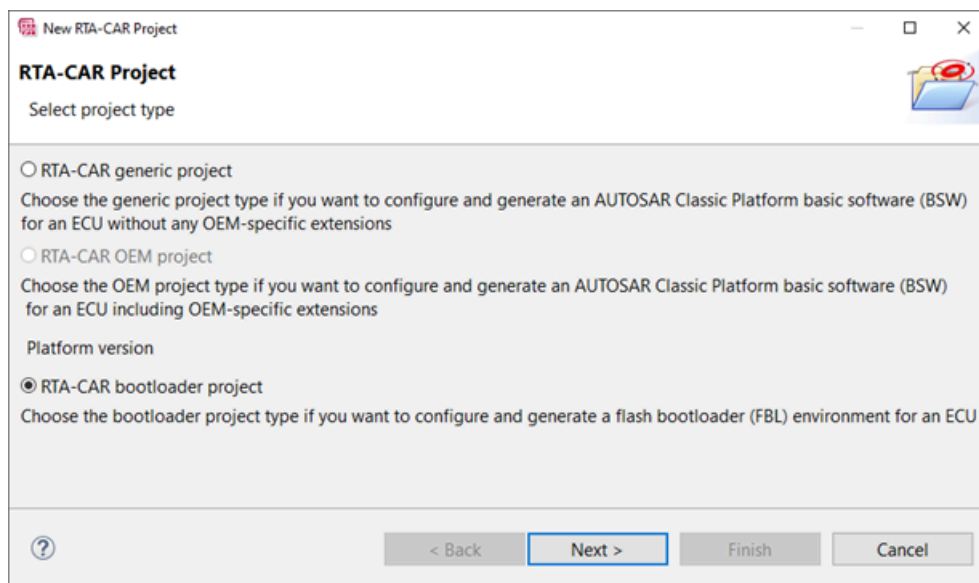


Figure 4.6.New RTA-CAR Bootloader Project

Next, choose a name for your project and select the RTA-FBL target from the dropdown list as shown in Figure 4.7. 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 2.1.0.STLA plugin under RTA FBL Tools.

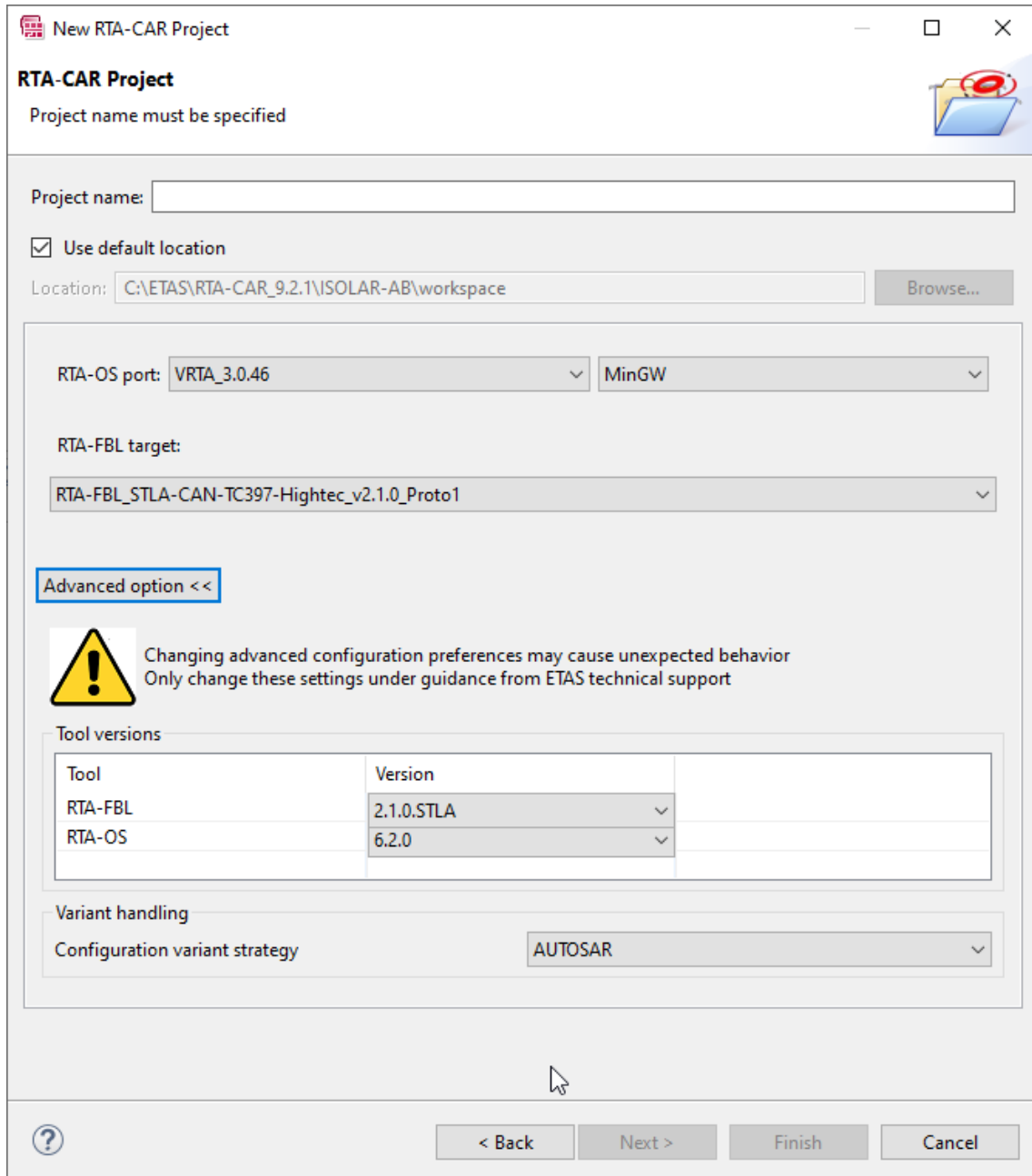


Figure 4.7.Select Target

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

Figure 4.8 shows the result of a successful project creation in the console window.

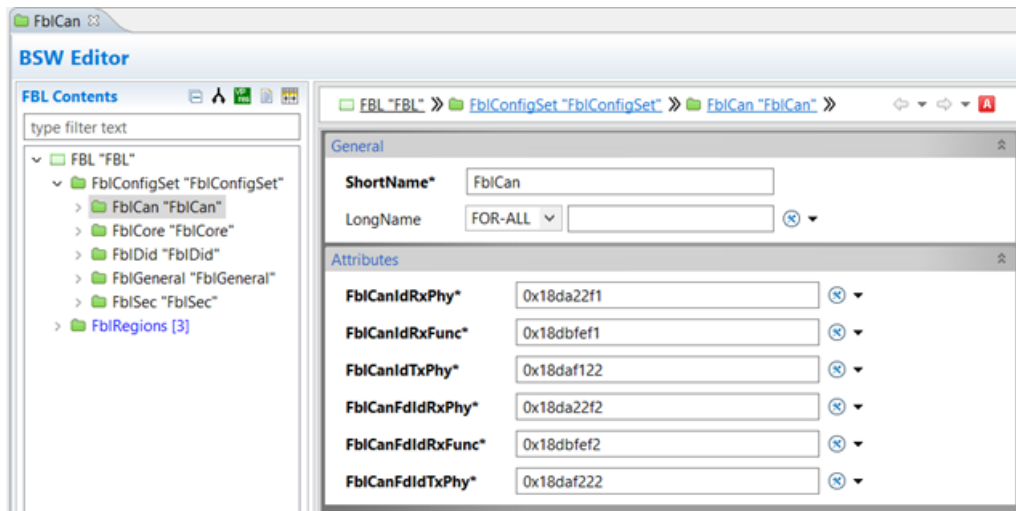


Figure 4.10. Edit Base Configuration Parameters

Once complete, the user can generate the RTA-FBL instance first by clicking on “Open RTA Code Generator dialog...” as shown in Figure 4.11 and then, in the opened RTA Code Generator window, by clicking Run after selecting only RTA-FBL_STLA as shown in Figure 4.12.

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

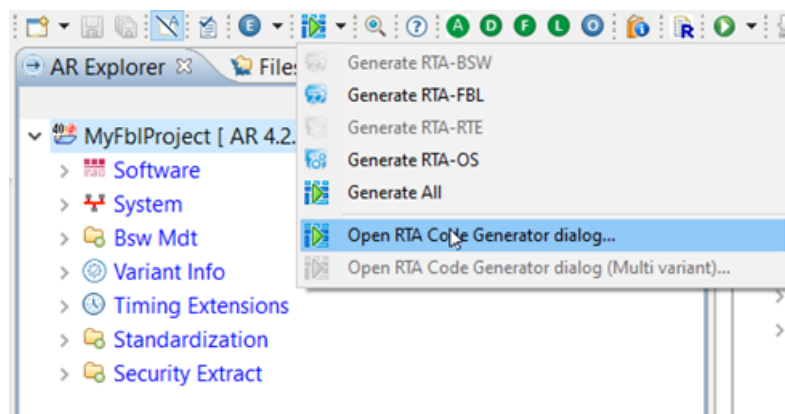


Figure 4.11. Open RTA Code Generator Dialog

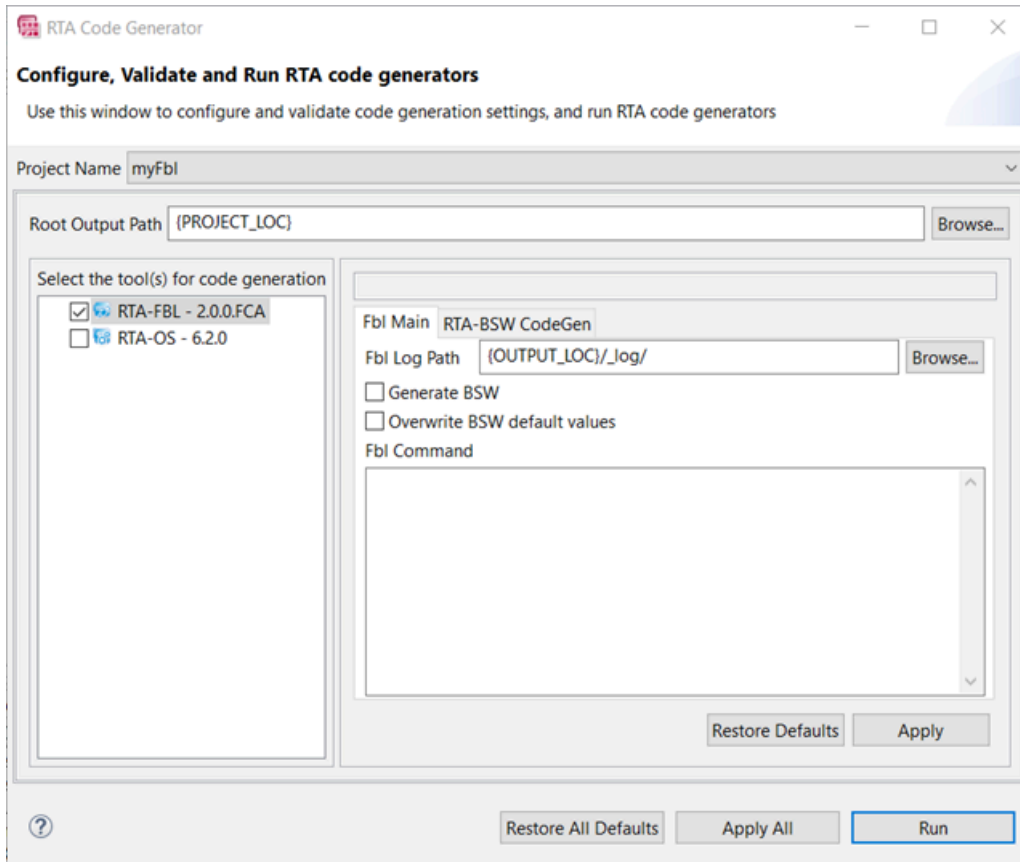


Figure 4.12.RTA Code Generator

Note the two options that are available:

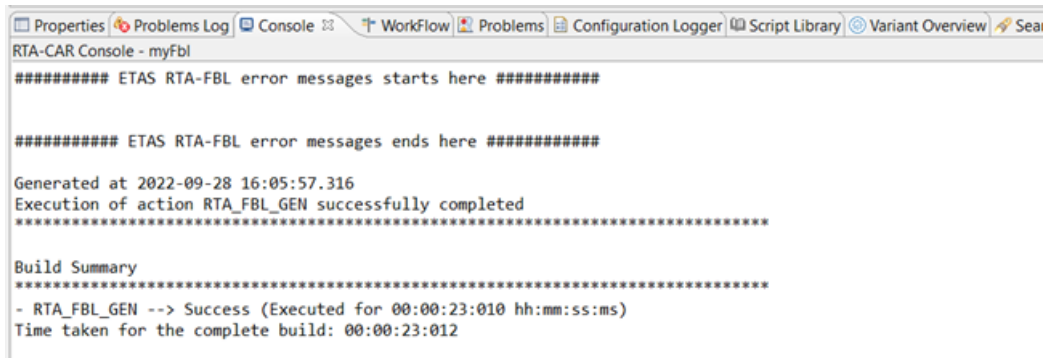
- Generate BSW: This will automatically generate the BSW after FBL generation using the BSW configuration generated by the FBL generator.
- Overwrite BSW default values: If this option is selected, any manual changes you have made to the BSW configuration after the last FBL generation will be lost and 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).

WARNING

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 4.13 shows the result

of a successful generation in the console window.



```

RTA-CAR Console - myFbl
##### ETAS RTA-FBL error messages starts here #####

##### ETAS RTA-FBL error messages ends here #####

Generated at 2022-09-28 16:05:57.316
Execution of action RTA_FBL_GEN successfully completed
*****

Build Summary
*****
- RTA_FBL_GEN --> Success (Executed for 00:00:23:010 hh:mm:ss:ms)
Time taken for the complete build: 00:00:23:012

```

Figure 4.13. 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 4.14.

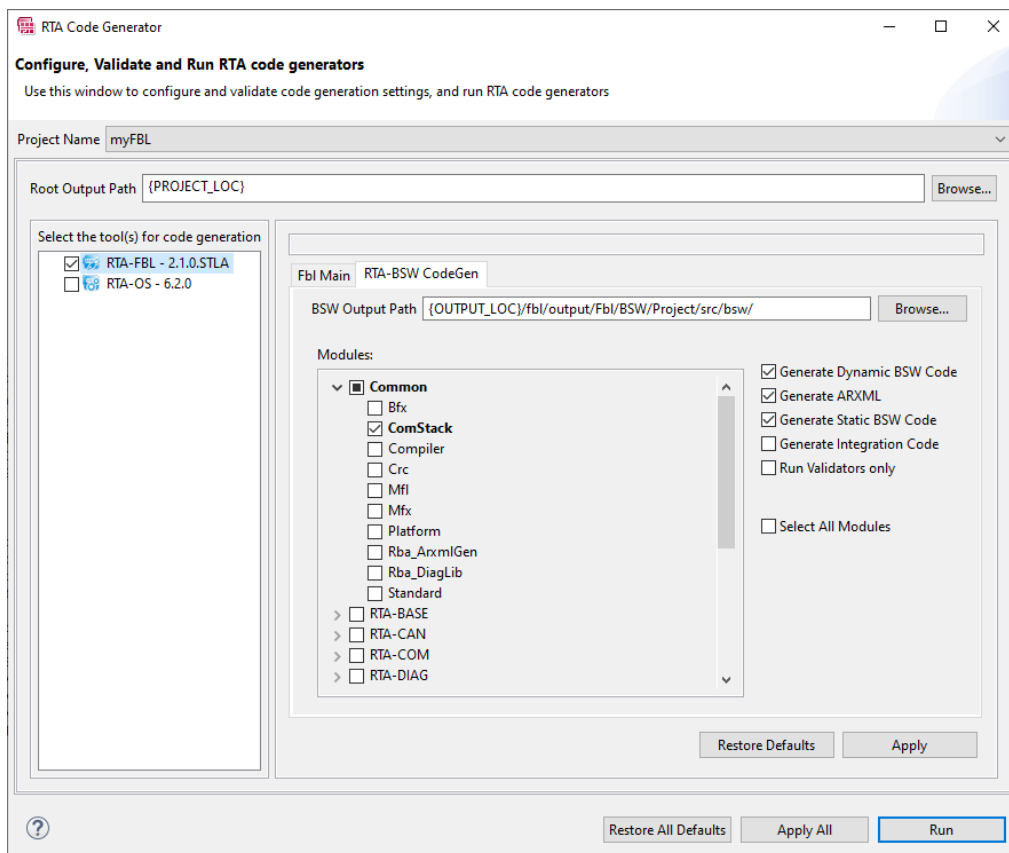


Figure 4.14. RTA-BSW CodeGen tab

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 4.15. Upon successful generation, the popup message in Figure 4.16 is shown.

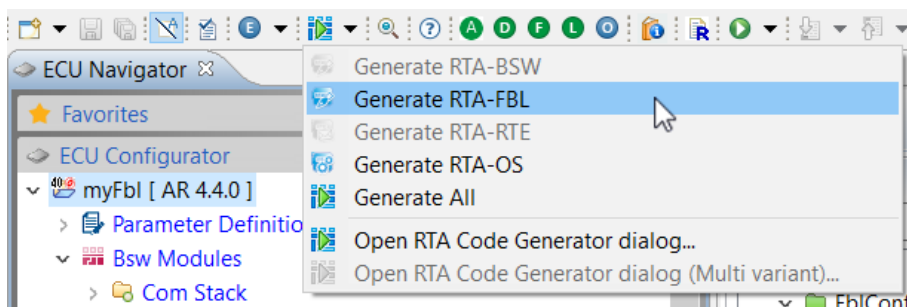


Figure 4.15. Generate RTA-FBL

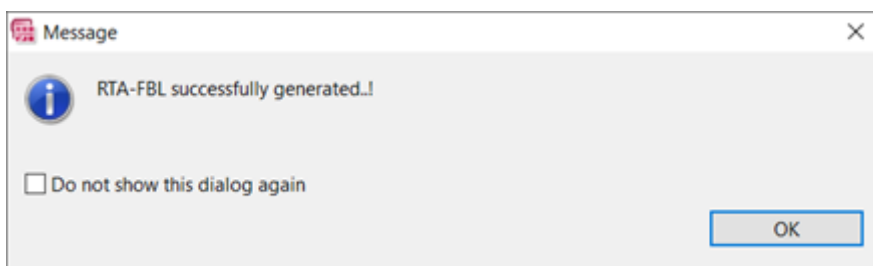


Figure 4.16. Successful generation

Following sections describe the parameters that the user can configure. The letters 'M' and 'O' are used to indicate "Mandatory" and "Optional" respectively. If O* or M* is specified, then see the Description for exceptions. The column "Requires BSW Re-Gen" 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 Target Guide.

4.3.2.1 FblRegion

This container allows the configuration of the memory regions of your ECU. For each region the parameters details are listed in Table 4.1.

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

Table 4.1. Configuration parameters FblRegion of the Stellantis port of RTA-FBL

Parameter	Description	Requires BSW Re-Gen
FblRegionAddressLow	Specifies the low address of the region	Yes
FblRegionAddressHigh	Specifies the high address of the region	Yes
FblRegionMaxAttemptCounter	Specifies the maximum number of reprogramming attempts for the region. When the number of attempts reaches this threshold, it will not be possible to reprogram this block anymore.	Yes
FblRegionType	Specifies the region type: <ul style="list-style-type: none"> - BOOT_BLOCK Region used for a Boot block - APPLICATION_BLOCK Region used for an application block - DATA_BLOCK Region used for a data block - TS_MAIN_BLOCK: region used for TS main block - TS_BACKUP_BLOCK: region used for TS backup block - DCL_BACKUP_BLOCK: region used for DCL backup block 	Yes
FblRegionExternalFlashSupport	Specifies the region type: <ul style="list-style-type: none"> - INTERNAL host PFLASH region: erasing and writing is handled by the FlashBootloader using MCAL APIs - EXTERNAL external memory device: erasing and writing is handled by the user via callbacks. 	Yes

FblRegionID	Specifies the identification number of the region. The number uniquely identifies the region, and it is used by CDA tool to address the block during download	Yes
FblRegionSecTrustedBootMode	If the FSM or the SSM is used, this parameter specifies the Trusted boot verification mode for the region.	Yes
FblRegionSecCompatibilityId	If the FSM or the SSM is used, this parameter specifies the compatibility id of the region.	Yes

4.3.2.2 FblGeneral

Table 4.2 provides a description of each parameter for the container FblGeneral.

Table 4.2. Configuration parameters FblGeneral of the Stellantis port of RTA-FBL

Parameter	O/M	Description	Requires BSW Re-Gen
FblEcuType	M	<p>This parameter indicates which ECU Type is selected. As per STLA specification [1] three different ECU types are supported:</p> <ul style="list-style-type: none"> - ECU Type A (ECU_A) ECU supporting Standard reprogramming: there is no specific security support, the Download is unlocked by a security level 0x1 (Seed&Key). The FW blocks are not protected with authenticity check (signature verification) but only against consistency (CRC check). - ECU Type B (ECU_B) ECU supporting Standard Reprogramming and Cyber security Level 4: there is partial security support, the Download is unlocked by a security level 0x1 (Seed&Key). The FW blocks are protected with authenticity check (signature verification). To accomplish this the ECU must maintain an internal certificate database (TrustStore). - ECU Type C (ECU_C) ECU supporting Authenticated Reprogramming and Cyber security Level 4 and 5: there is full security support, the Download is unlocked by a security level 0x11 (ADA - Challenge-Response). The FW blocks are protected with authenticity check (signature verification). To accomplish this the ECU must maintain an internal certificate database (TrustStore). 	Yes

FblSleepWakeup	M	Specifies whether the ECU is a +15 or +30 node and it shall support the Bootloader Sleep/Wakeup Mechanism of [1].	No
FblSleepWakeupJump ToAppDelay	1	Allows to configure the time in ms that the FBL shall wait before jumping to the application if an unwanted reset is detected.	No
FblFlashBufferSize	M	Allows to configure the buffer used to access the flash area during read and write operation. This parameter defines the size of the buffer allocated in the RAM.	No
FblBlockSize	O	Allows the user to configure the download block size in bytes.	No
FblDidC5Support	O	Allows to configure the Erotan (\$F196) of the FlashBootloader, for details refer to [2].	No

¹: Mandatory if FblSleepWakeup is true.

4.3.2.3 FblCore

Table 4.3 provides a description of each parameter for the container FblCore.

Table 4.3. Configuration parameters FblCore of the Stellantis port of RTA-FBL

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

FblEraseMaxSize PerCycle	O	Allows to configure the maximum number of bytes that can be erased in one erase cycle. If this parameter is set, the logical block erase operation is split according to the bank memory layout and this parameter. If this parameter is not set, the logical block erase operation is split according to the bank memory layout.	No
-----------------------------	---	---	----

4.3.2.4 FblCan

The Can parameters details of FblCan container are listed in Table 4.4. For additional information on the MCAL Can configuration, please refer to your FBL Target Guide.

Table 4.4. Configuration parameters FblCan of the Stellantis port of RTA-FBL

Parameter	O/M	Description	Requires BSW Re-Gen
FblCanType	O	Allows to configure the Can addressing mode. The value can be: <ul style="list-style-type: none"> - STANDARD: 11-bit addressing mode is supported. - EXTENDED: 29-bit addressing mode is supported. If parameter is not set, then the Can addressing mode is set to Extended.	Yes
FblCanIdRxPhy	M	The physical receive Can ID. The value can be: <ul style="list-style-type: none"> - an integer between 1 and 0x1FFFFFF if Can addressing mode is set to Extended. - an integer between 1 and 0x7FF if Can addressing mode is set to Standard. Both addressing mode 29 bits and 11 bits are supported.	Yes

FblCanIdRxFunc	M	<p>The functional receive Can ID. The value can be:</p> <ul style="list-style-type: none"> - an integer between 1 and 0x1FFFFFFF if Can addressing mode is set to Extended. - an integer between 1 and 0x7FF if Can addressing mode is set to Standard. <p>Both addressing mode 29 bits and 11 bits are supported.</p>	Yes
FblCanIdTxPhy	M	<p>The physical transmit Can ID. The value can be:</p> <ul style="list-style-type: none"> - an integer between 1 and 0x1FFFFFFF if Can addressing mode is set to Extended. - an integer between 1 and 0x7FF if Can addressing mode is set to Standard. <p>Both addressing mode 29 bits and 11 bits are supported.</p>	Yes
FblCanFdidRxPhy	M	<p>The physical receive Can FD ID. The value can be:</p> <ul style="list-style-type: none"> - an integer between 1 and 0x1FFFFFFF if Can addressing mode is set to Extended. - an integer between 1 and 0x7FF if Can addressing mode is set to Standard. <p>Both addressing mode 29 bits and 11 bits are supported.</p>	Yes
FblCanFdidRxFunc	M	<p>The functional receive Can FD ID. The value can be:</p> <ul style="list-style-type: none"> - an integer between 1 and 0x1FFFFFFF if Can addressing mode is set to Extended. - an integer between 1 and 0x7FF if Can addressing mode is set to Standard. <p>Both addressing mode 29 bits and 11 bits are supported.</p>	Yes

FblCanFdidTxPhy	M	<p>The physical transmit Can FD ID. The value can be:</p> <ul style="list-style-type: none"> - an integer between 1 and 0x1FFFFFFF if Can addressing mode is set to Extended. - an integer between 1 and 0x7FFF if Can addressing mode is set to Standard. <p>Both addressing mode 29 bits and 11 bits are supported.</p>	Yes
-----------------	---	---	-----

4.3.2.5 FblSec

This section describes the security related parameters. Some of these parameters depend on the FSM and the CycurHsm integration. Check if the CycurHsm is supported in your FBL Target Guide.

Table 4.5. Configuration parameters FblSec of the Stellantis port of RTA-FBL

Parameter	O/M	Description	Requires BSW Re-Gen
FblSecSk1, FblSecSk2	M ¹	Security constants for Seed&Key algorithm to unlock level 0x01. Mandatory for ECU Type A and ECU Type B.	No
FblSecStackType	M	Specifies the sec stack type. Refer to the chapter Security Stack for more information.	No
FblSecTsCapacity	M ²	Specifies the maximum number of certificates that could be stored in the Trustore.	No
FblSecCsCapacity	M ²	Specifies the number of certificates that could be stored in the CertStore.	No
FblSecCrlCapacity	M ²	Specifies the number certificates that could be stored in the Certificate Revocation List.	No
FblSecTargetName	M ²	Specifies the Target Name for all Signed Firmware Blocks, for details refer to SD.00015	No

¹: This parameter is mandatory only if EcuType is set to ECU_A or ECU_B

²: This parameter is mandatory only if EcuType is set to ECU_B or ECU_C and your target integrates the CycurHsm integration.

4.3.2.6 FblNvmBlock

This container allows to add or modify the default configuration of the non volatile

memory blocks. This parameter affects FBL_DataM, Nvm and Fee. It's not possible to delete the default non volatile memory blocks.

Table 4.6. Configuration parameters FblNvmBlocks of the Stellantis port of RTA-FBL

Parameter	O/M	Description	Requires BSW Re-Gen
FblNvmBlock	M	ShortName of the NvM block. This parameter shall be unique or it can have the same name of the a default NvmBlock. Unicity check is only performed on FBL configured parameters so BSW modules are excluded.	Yes
FblNvmBlockId	M	Identifier of the NvmBlock. This identifier will be propagated to the NvM and the Fee modules This identifier shall be unique. Unicity check is only performed on FBL configured parameters so BSW modules are excluded..	Yes
FblNvmBlockSize	M	Size of the NvmBlock. This identifier will be propagated to the NvM and the Fee modules	Yes

Refert to FBL: NvM adaption for more details about default values and how to manage NvM.

4.3.2.7 FblNvmData

This container allows to add or modify the default configuration of the non volatile memory data. This parameter affects FBL_DataM. It's not possible to delete the default non volatile memory data.

Table 4.7. Configuration parameters FblNvmBlocks of the Stellantis port of RTA-FBL

Parameter	O/M	Description	Requires BSW Re-Gen
FblNvmData	M	ShortName of the NvM data. This parameter shall be unique or it can have the same name of the a default NvM data. Unicity check is only performed on FBL configured parameters so BSW modules are excluded..	Yes
FblNvmBlockName	M	ShortName of the NvmBlock that contains the data. The FblNvmBlock shall be present in the FblNvmBlock configuration.	Yes

FblNvmBlockOffset	M	Offset of the data inside the NvmBlock that contains the data.	Yes
FblNvmDataSize	M	Size of the data.	Yes
FblNvmData DefaultValue	M	Allows to configure the default value of the NvM data. The default value shall be and hexadecimal string like 00FF00 (3 byte size data). Eventually the default value can be 1 byte that will be copied over the whole size. For example an NvM data of size 5 can have the default value set to 01. This configuration will generate the following default value 0101010101.	Yes

Refert to Chapter FBL: NvM adaption for more details about default values and how to manage NvM.

4.3.2.8 FblDid

This container allows to add or modify the default configuration of the DID. It's not possible to delete the default DIDs. This parameter affect the dcm configuration and the FBL_Port source code.

Table 4.8. Configuration parameters FblDid of the Stellantis port of RTA-FBL

Parameter	O/M	Description	Requires BSW Re-Gen
FblDid	M	ShortName of the Did. This parameter shall be unique or it can have the same name of a default DID.	Yes
FblDidIdentifier	M	Identifier of the DID. A value in the range between 0 and 0xFFFF.	Yes
FblDidReadInSession	O	Allows to configure the available session for reading the DID. This can be a string containing D for Default session, P for Programming session and E for Extended session. For example the string DPE will allows the DID to be read in all the sessions.	Yes
FblDidWriteInSession	O	Allows to configure the available session for writing the DID. This can be a string containing D for Default session, P for Programming session and E for Extended session. For example the string DPE will allows the DID to be read in all the sessions.	Yes

FblDidType	M	<p>Allows to configure the type of the DID.</p> <p>Sets the DID type, select among:</p> <ul style="list-style-type: none"> - NVM_DATA read/write data direct from NvM. FBL_DataM will manage the dcm callback. - CALLBACK read/write through a callback. FBL_Port will manage the dcm callback. - USER_CBK read/write through a callback. The user shall define the dcm callback. <p>If the DID default type is CALLBACK, the user can change the type to USER_CBK to exclude the default call-back implementation.</p> <p>If the DID default type is NVM_DATA, the user can change the type to USER_CBK or CALLBACK and this makes no difference. The user shall define the dcm callback.</p>	Yes
FblDidNvmDataName	O*	<p>Allows to configure the Nvm data associated to the DID. If FblDidType is set to NVM_DATA, this field is mandatory. The short name reported shall be a valid short name configured in the FblNvmData table.</p>	Yes
FblDidCbKReadSize	O*	<p>Allows to configure size of the data available in the read DID callback. If FblDidType is set to CALLBACK or USER_CBK, this field is mandatory.</p>	Yes
FblDidCbKWriteSize	O*	<p>Allows to configure the size of the data available in the write DID callback. If FblDidType is set to CALLBACK or USER_CBK, this field is mandatory.</p>	Yes

Refert to FblFotaRollbackRegion for more details about default values and how to manage DIDs.

4.3.2.9 FblFota

This container allows to manage the configurations for the FOTA.

Table 4.9.FblFota configuration container

Parameter	O/M	Description	Requires BSW Re-Gen
FblFotaType	O	Allows to configure the FOTA type. The value can be: <ul style="list-style-type: none"> - FOTA_DISABLED: FOTA features is disabled. - FOTA_SMALL_TARGET_ECU: FOTA features is enabled. If parameter is not set, then the FOTA feature is disabled.	Y
FblFotaRollback	O	Allows to enable or disable the FOTA rollback feature. If parameter is not set, then the FOTA rollback feature is disabled. This parameter can be set to ENABLED only if the FblFotaType is not set to FOTA_DISABLED	Y
FblFotaCanFIdRxPhy	O	The physical receive Can ID. It is an integer between 1 and 0xFFFFFFFF. Only addressing mode 29 bits is supported.	Y
FblFotaCanFIdRxFunc	O	The functional receive Can ID. It is an integer between 1 and 0xFFFFFFFF. Only addressing mode 29 bits is supported.	Y
FblFotaCanFIdTxPhy	O	The physical trasmit Can ID. It is an integer between 1 and 0xFFFFFFFF. Only addressing mode 29 bits is supported.	Y

4.3.2.10 FblFotaRollbackRegion

This container allows to manage the FotaRollback blocks.

Table 4.10.Configuration parameters FblDid of the Stellantis port of RTA-FBL

Parameter	O/M	Description	Requires BSW Re-Gen
FblFotaRollbackRegionAddressLow	M	Specifies the low address of the region	N

FblFotaRollback RegionAddressHigh	M	Specifies the high address of the region	N
FblFotaRollback RegionExternalFlash Support	M	Specifies the region type: <ul style="list-style-type: none"> - INTERNAL host PFLASH region: erasing and writing is handled by the FlashBootloader using MCAL APIs - EXTERNAL external memory device: erasing and writing is handled by the user via callbacks. 	N
FblRegionId	M	Specifies the identification number of the region that is used by the rollback region.	N

4.3.3 Files created during generation

When you generate an instance of the Stellantis 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 4.11 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 FBL 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 4.11. 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 core-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/BLSM	The BLSM contains code for initializing the Bootloader. The functions in ./src/BLSM_CallOuts.c can be changed as described in FBL: BLSM adaptation, but the functions in BLSM_Main.c should not be changed. It is the integrator's responsibility to ensure that any changes made in the BLSM do not affect the bootloader's correct functionality.
./fbl/output/Fbl/INFRA/OS	The OS contains the cyclic scheduler that calls the module main functions. The OS is provided as a fully functioning and tested sample, but the integrator may replace the OS as described in FBL: OS adaptation. For example, the integrator may wish to use RTA-OS in order to more easily configure interrupts for other software integrated with RTA-FBL. It is the integrator's responsibility to ensure that any changes made to the OS do not affect the bootloader's correct functionality.
./fbl/output/Fbl/INFRA/Port	This folder contains the code that implements port-specific functionality. 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/Tools/BootUpdater	This folder contains the BootUpdater sample, for details refer to Bootloader Update.
./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.
./fbl/output/Fbl/INFRA/SecHAL	This folder contains interface between the CycurHsm and the FBL

4.3.4 The RTA-FBL instance for the Dummy Target

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

The FBL for your target will have undergone an in-depth testing using the

compiler and MCAL that you have chosen. All targets use a common base that require the tools as described in Setting up your environment to generate an RTA-FBL instance.

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.

4.3.4.1 Dummy Target Memory Layout

In order to allow the user to experiment with different memory space configurations, the dummy target is set up to mimic the memory layout of Infineon's TC233 processor. This processor has a memory layout as shown in Table 4.12. Memory regions of a space must begin on sector boundaries and the bootloader reserves the first sector (i.e. the memory between 0xA0000000 and 0xA01FFFFF). You can experiment with different configurations of Application, Calibration and Bootloader space if you have not yet received your Target package. For example, if you configure a space that uses a memory region that is not on a region boundary or that enters a disallowed space and note the error returned by the FBL generator.

Table 4.12. Memory layout of the Dummy Target

Bank	Sector	Start	End	Comment
0	0	0xA0000000	0xA0003FFF	Reserved for FBL
	1	0xA0004000	0xA0007FFF	Available for Application/Calibration
	2	0xA0008000	0xA000BFFF	
	3	0xA000C000	0xA000FFFF	
	4	0xA0010000	0xA0013FFF	
	5	0xA0014000	0xA0017FFF	Not available for Application/Calibration
	6	0xA0018000	0xA001BFFF	
	7	0xA001C000	0xA001FFFF	Available for Application/Calibration
	8	0xA0020000	0xA0027FFF	
	9	0xA0028000	0xA002FFFF	
	10	0xA0030000	0xA0037FFF	
	11	0xA0038000	0xA003FFFF	
	12	0xA0040000	0xA0047FFF	
	13	0xA0048000	0xA004FFFF	
	14	0xA0050000	0xA0057FFF	
	15	0xA0058000	0xA005FFFF	
	16	0xA0060000	0xA006FFFF	Not available for Application/Calibration
17	0xA0070000	0xA007FFFF		

1	18	0xA0080000	0xA008FFFF	Available for Application/Calibration
	19	0xA0090000	0xA009FFFF	
	20	0xA00A0000	0xA00BFFFF	
	21	0xA00C0000	0xA00DFFFF	
	22	0xA00E0000	0xA00FFFFFF	
2	23	0xA0100000	0xA013FFFF	
	24	0xA0140000	0xA017FFFF	
3	25	0xA0180000	0xA01BFFFF	
	26	0xA01C0000	0xA01FFFFFF	

4.4 Security Stack

Depending on the configuration of the `FblSecStackType`, the integrator may need to add the security modules provided by Escript:

- Stellantis Security Manager (SSM),
- FCA Security Manager (FSM),
- CycurHSM.

The `FblSecStackType` is set to:

- *USR*: the FBL provides some callout to integrate a user security stack. More information in chapter FBL callout for SecStack integration.
- *FSM*: the FBL uses the CycurHsm and the FSM. More information in chapter FCA Security Manager (FSM).
- *SSM*: the CycurHsm and the SSM shall be integrated. More information in chapter STLA Security Manager (SSM).

If your target supports Escript modules, your FBL Target Guide will provide information on how to integrate the security stack.

The security stacks provided by Escript supports the following features:

- Trusted boot
- Trusted download
- ADA
- X509v3 management
- Lv9 features for FOTA feature.

4.4.1 FBL callout for SecStack integration

The FBL provides the following callouts to integrate the security stack. In this section you will find only the callout provided by the FBL. The Security features will be presented in the next chapters. All the following apis can be found in the `FBL_Security`, `ADA` and `FBL_FOTA` modules. These apis can provides the integra-

tion of the CysurHsm/Fsm or can be used to integrate an external Security stack.

4.4.1.1 Startup

This section provides all the apis that can be used in the startup phase to initialize the HTA and request the Secure Boot.

Prototype	void Fbl_Sec_Usr_DrvInitZero (void)
Parameter	none
Return Code	none
Functional Description	This api is called during the startup phase. It is requested just after the MCAL startup code and before any other initialization.
Location	FBL_Security.c
Pre-Conditions	none

Prototype	void Fbl_Sec_Usr_DrvInitOne (void)
Parameter	none
Return Code	none
Functional Description	This api is called during the startup phase. It is requested just after the NvM initialization. This api is used by the CysurHsm.
Location	FBL_Security.c
Pre-Conditions	none

Prototype	void Fbl_Sec_Usr_DrvInitThree (void)
Parameter	none
Return Code	none
Functional Description	This api is called during the startup phase. It is requested just after the Dcm initialization. If the FBL jumps to the application, this api is not
Location	FBL_Security.c
Pre-Conditions	none

4.4.1.2 Jump to application

This section provides all the apis that can be used to check the application validity and to perform any HTA action before jumping to the application.

Prototype	boolean Fbl_Sec_Usr_ApplicationIsValid (uint8* blockId)
Parameter	[out] blockId: it points to an array of size equal to DID201F_-LOGICAL_BLOCK_BYTES. Each bit of this array refers to the validity of the correspondent logical block as required by the DID 0x201F.

Return Code	TRUE: The software and the TS are trustable FALSE: The software and the TS are not trustable
Functional Description	This api is called before jumping to the application to enable or disable the application jump. This api is also called by the DID 0x201F to retrieve the invalid software parts.
Location	FBL_Security.c
Pre-Conditions	none

Prototype	<code>boolean Fbl_Sec_Usr_ApplicationJumpPreHook(void)</code>
Parameter	none
Return Code	none
Functional Description	This api is called only if the jump to the application is requested and just before the actual jump. It performs any action required by the HTA before jumping to the application like disabling security features that shall not be supported in the application.
Location	FBL_Security.c
Pre-Conditions	none

4.4.1.3 Periodic

This section provides all the apis that can be used to perform any periodic task related to the HTA.

Prototype	<code>void Fbl_Sec_Usr_MainFunction_1ms (void)</code>
Parameter	none
Return Code	none
Functional Description	Periodic function scheduled by the OS every 1ms
Location	FBL_Security.c
Pre-Conditions	none

Prototype	<code>void Fbl_Sec_Usr_MainFunction_5ms (void)</code>
Parameter	none
Return Code	none
Functional Description	Periodic function scheduled by the OS every 5ms
Location	FBL_Security.c
Pre-Conditions	none

Prototype	void Fbl_Sec_Usr_MainFunction_10ms (void)
Parameter	none
Return Code	none
Functional Description	Periodic function scheduled by the OS every 10ms
Location	FBL_Security.c
Pre-Conditions	none

4.4.1.4 HTA suspension and activation

This section provides all the apis that can be used to enable the HTA while erasing or writing the PFLASH.

Prototype	Fbl_Sec_StatusT Fbl_Sec_Usr_Suspend(void)
Parameter	none
Return Code	FBL_SEC_STATUS_OK: HTA suspension passes FBL_SEC_STATUS_ERROR: HTA suspension fails
Functional Description	Service to suspend the HTA
Location	FBL_Security.c
Pre-Conditions	none

Prototype	Fbl_Sec_StatusT Fbl_Sec_Usr_Activate(void)
Parameter	none
Return Code	FBL_SEC_STATUS_OK: HTA activation passes FBL_SEC_STATUS_ERROR: HTA activation fails
Functional Description	Service to activate the HTA
Location	FBL_Security.c
Pre-Conditions	none

4.4.1.5 Secure Download

This section provides all the apis that can be used to start and check the software verification.

Prototype	Fbl_Sec_StatusT Fbl_Sec_Usr_VerifyDownload (uint8 SfbIndex)
Parameter	[in] SfbIndex: index of the signed firmware block that shall be verified
Return Code	FBL_SEC_STATUS_OK: verification is requested. FBL_SEC_STATUS_BUSY: verification request is in progress. FBL_SEC_STATUS_ERROR: verification request fails.

Functional Description	It requests the verification process of a signed firmware block. The verification process is asynchronous, and the output of the verification can be checked polling the FBL_Sec_VerifyDownloadResult.
Location	FBL_Security.c
Pre-Conditions	none

Prototype	Fbl_Sec_StatusT FBL_Sec_Usr_VerifyDownloadResult(void)
Parameter	none
Return Code	FBL_SEC_STATUS_OK: verification ends successfully. FBL_SEC_STATUS_NOT_REQUESTED: verification is not requested. FBL_SEC_STATUS_BUSY: verification request is in progress. FBL_SEC_STATUS_ERROR: verification ends failing.
Functional Description	It checks the result of the verification process of a signed firmware block
Location	FBL_Security.c
Pre-Conditions	none

4.4.1.6 ADA - Challenge request

This section provides all the apis that can be used to generate the challenge required by the Authenticated Security access (\$27 11).

Prototype	ADA_Sec_StatusT ADA_Sec_Usr_GenerateChallenge (const uint8* certificate)
Parameter	[in] certificate: pointer to the ADA certificate
Return Code	ADA_SEC_STATUS_OK: the job is scheduled ADA_SEC_STATUS_BUSY: HTA is busy and the job is not scheduled ADA_SEC_STATUS_ERROR: HTA is faulty and the job is not scheduled
Functional Description	It requests the random seed generation, it stores the ADA certificates, and it can request the certificate verification. This process is asynchronous, and the output of the verification can be checked polling the ADA_Sec_Usr_GetChallengeStatus.
Location	ADA_SecStackUserCode.c
Pre-Conditions	none

Prototype	ADA_Sec_StatusT ADA_Sec_Usr_GetChallengeStatus(uint8* randomSeed)
Parameter	[in] certificate: pointer to the ADA certificate

Return Code	<p>ADA_SEC_STATUS_OK: seed is available and job is successfully terminated.</p> <p>ADA_SEC_STATUS_BUSY: seed is not available and job is executing.</p> <p>ADA_SEC_STATUS_REVOKED_CERT: seed is not available and job is successfully terminated.</p> <p>ADA_SEC_STATUS_EXPIRED_CERT: seed is not available and job is successfully terminated.</p> <p>ADA_SEC_STATUS_ERROR: seed is not available and job is successfully terminated.</p>
Functional Description	It checks the status of the HTA job triggered by the functions ADA_Sec_Usr_GenerateChallenge and it returns the random seed to the FBL.
Location	ADA_SecStackUserCode.c
Pre-Conditions	none

4.4.1.7 ADA - Verify response and periodic certificate verification

This section provides all the apis that can be used to verify the response and the certificate required by the Authenticated Security access (§27 12). The ADA certificate can be verified both during the §27 11 and §27 12. By default, the ADA certificate is stored in the host NvM, and it's verified periodically. Check the Authenticated Diagnostic Access section for more details.

Prototype	ADA_Sec_StatusT ADA_Sec_Usr_VerifyResponse (const uint8* key)
Parameter	[in] key: pointer to the ADA signed random number array
Return Code	<p>ADA_SEC_STATUS_OK: the job is scheduled</p> <p>ADA_SEC_STATUS_BUSY: HTA is busy and the job is not scheduled</p> <p>ADA_SEC_STATUS_ERROR: HTA is faulty and the job is not scheduled</p>
Functional Description	<p>This api is called during the dcm callback for the service §27 12.</p> <p>It requests the verification of the Signed Challenge and the ADA certificate. This process is asynchronous, and the output of the verification can be checked polling the ADA_Sec_Usr_VerifyResponseStatus.</p>
Location	ADA_SecStackUserCode.c
Pre-Conditions	none

Prototype	ADA_Sec_StatusT ADA_Sec_Usr_VerifyCertificate (const uint8* certificate)
Parameter	[in] certificate: pointer to the ADA certificate

Return Code	ADA_SEC_STATUS_OK: the job is scheduled ADA_SEC_STATUS_BUSY: HTA is busy and the job is not scheduled ADA_SEC_STATUS_ERROR: HTA is faulty and the job is not scheduled
Functional Description	It periodically requests the verification of the ADA certificate. This process is asynchronous, and the output of the verification can be checked polling the ADA_Sec_Usr_VerifyResponseStatus.
Location	ADA_SecStackUserCode.c
Pre-Conditions	none

Prototype	ADA_Sec_StatusT ADA_Sec_Usr_VerifyResponseStatus(void)
Parameter	none
Return Code	ADA_SEC_STATUS_OK: ADA certificate is valid job is successfully terminated. ADA_SEC_STATUS_BUSY: job is executing. ADA_SEC_STATUS_REVOKED_CERT: ADA certificate is revoked. The job ends with an error. ADA_SEC_STATUS_EXPIRED_CERT: ADA certificate is expired. The job ends with an error. ADA_SEC_STATUS_ERROR: ADA certificate is not valid or the signature is not valid. The job ends with errors.
Functional Description	It checks the status of the HTA job triggered by the functions: ADA_Sec_Usr_VerifyResponse and ADA_Sec_Usr_VerifyCertificate. If the request comes from the ADA_Sec_Usr_VerifyCertificate and the certificate is stored in a protected area, this api shall ADA_SEC_STATUS_OK since any other result will trigger a reset.
Location	ADA_SecStackUserCode.c
Pre-Conditions	none

4.4.1.8 ADA - Certificate management

This section provides all the apis that can be used to access to the information stored in the ADA certificate.

Prototype	ADA_Sec_StatusT ADA_Sec_Usr_GetCertificateInfo (ADA_Sec_Usr_CertificateInfoT* certInfo)
Parameter	[out] certInfo: a structure containing the serialNumber and the commonName of the ADA certificate.
Return Code	ADA_SEC_STATUS_OK: ADA certificate is verified and data are available. ADA_SEC_STATUS_ERROR: ADA certificate is not verified and data are not available.

Functional Description	After a successful verification of the ADA certificate, this api is used to retrieve the serialNumber and the commonName and store these information in the DID 2030.
Location	ADA_SecStackUserCode.c
Pre-Conditions	none

Prototype	uint8 ADA_Sec_Usr_GetAuthRole(void)
Parameter	None
Return Code	0x00U if no role is set or the current role defined in the ADA certificate.
Functional Description	It retrieves the role of a valid ADA certificate. If the ADA certificate is not valid, it returns 0x00U.
Location	ADA_SecStackUserCode.c
Pre-Conditions	none

Prototype	void ADA_Sec_Usr_GetCertificate(uint8 * data)
Parameter	[out] data: pointer to the ADA certificate
Return Code	none
Functional Description	This api is called to store the ADA certificate in the host NVM. It retrieves the ADA certificate from the security stack.
Location	ADA_SecStackUserCode.c
Pre-Conditions	none

Prototype	ADA_Sec_StatusT ADA_Sec_Usr_GetCertificateLatestSavedDate (uint64* latestSavedDate)
Parameter	[out] latestSavedDate: latest saved date formatted as unix time
Return Code	ADA_SEC_STATUS_OK: latest Saved Date was successfully calculated. ADA_SEC_STATUS_ERROR: latest Saved Date was not successfully calculated.
Functional Description	This api is called during the startup to retrieve the last saved date value from the validated certificate chain.
Location	ADA_SecStackUserCode.c
Pre-Conditions	none

Prototype	ADA_Sec_StatusT ADA_Sec_Usr_GetCertificateLatestSavedDate (uint64* latestSavedDate)
Parameter	[out] latestSavedDate: latest saved date formatted as unix time
Return Code	ADA_SEC_STATUS_OK: latest Saved Date was successfully calculated. ADA_SEC_STATUS_ERROR: latest Saved Date was not successfully calculated.
Functional Description	This api is called during the startup to retrieve the last saved date value from the validated certificate chain.
Location	ADA_SecStackUserCode.c
Pre-Conditions	none

Prototype	Std_ReturnType ADA_Sec_Usr_certParseDate(uint64* timeStamp, uint8* dateString, uint8 length)
Parameter	[in] dateString: input ascii string [in] length: input ascii string length [out] timestamp: pointer to the time stamp
Return Code	E_OK: valid date E_NOT_OK: invalid date
Functional Description	It converts an x509v3 GeneralizedTime od UTCTime into a timestamp.
Location	ADA_SecStackUserCode.c
Pre-Conditions	none

4.4.1.9 ADA - Periodic

Prototype	void ADA_Sec_Usr_Mainfunction_1ms (void)
Parameter	none
Return Code	none
Functional Description	Periodic function scheduled by the OS every 1ms
Location	ADA_SecStackUserCode.c
Pre-Conditions	none

4.4.1.10 FOTA - CSR and identity key generation

This section provides all the apis required by the FBL during the and the private key and to sign with the identity key generation. This process is triggered by the RC 0xD00A.

Prototype	FBL_FOTA_Sec_StatusT FBL_FOTA_Sec_Usr_GenerateIdentityKeyData (void)
Parameter	none

Return Code	FBL_FOTA_SEC_STATUS_OK: job is triggered successfully. FBL_FOTA_SEC_STATUS_BUSY: HTA is busy, and job is not triggered. FBL_FOTA_SEC_STATUS_ERROR: HTA is faulty, and job is not triggered.
Functional Description	It triggers the identity key generation. This is an asynchronous function that trigger the job. FBL_FOTA_Sec_Usr_GetGenIdKeyStatus can be used for polling the result.
Location	FBL_FOTA_SecStackUserCode.c
Pre-Conditions	none

Prototype	FBL_FOTA_Sec_StatusT FBL_FOTA_Sec_Usr_GetGenIdKeyStatus (void)
Parameter	none
Return Code	FBL_FOTA_SEC_STATUS_OK: job ends successfully. FBL_FOTA_SEC_STATUS_BUSY: job is still running. FBL_FOTA_SEC_STATUS_ERROR: job ends with errors.
Functional Description	Periodic function scheduled by the OS every 1ms
Location	FBL_FOTA_SecStackUserCode.c
Pre-Conditions	none

Prototype	FBL_FOTA_Sec_StatusT FBL_FOTA_Sec_Usr_GenerateCSR (void)
Parameter	none
Return Code	FBL_FOTA_SEC_STATUS_OK: job is triggered successfully. FBL_FOTA_SEC_STATUS_BUSY: HTA is busy, and job is not triggered. FBL_FOTA_SEC_STATUS_ERROR: HTA is faulty, and job is not triggered.
Functional Description	It triggers the CSR generation. This is an asynchronous function that trigger the job. FBL_FOTA_Sec_Usr_GetCSRStatus can be used for polling the result.
Location	FBL_FOTA_SecStackUserCode.c
Pre-Conditions	none

Prototype	FBL_FOTA_Sec_StatusT FBL_FOTA_Sec_Usr_GetCSRStatus (void)
Parameter	none
Return Code	FBL_FOTA_SEC_STATUS_OK: job ends successfully. FBL_FOTA_SEC_STATUS_BUSY: job is still running. FBL_FOTA_SEC_STATUS_ERROR: job ends with errors.

Functional Description	It retrieves the status of the CSR generation process.
Location	FBL_FOTA_SecStackUserCode.c
Pre-Conditions	none

4.4.1.11 FOTA - Sign data with identity key

This section provides all the apis required by the FBL during the data signature calculation required by the DIDs 0xF1B7 and 0xF1B8.

As reported in the [2], the DIDs 0xF1B7 and 0xF1B8 reserve up to 256 bytes for the signature but it's up to the HTA implementation to define the signature algorithm. An api is provided to retrieve this information from the HTA.

Prototype	FBL_FOTA_Sec_StatusT FBL_FOTA_Sec_Usr_SignWithIdentityKey (const uint8* tbsData, uint32 tbsDataLen)
Parameter	[in] tbsData: The data to be signed with the identity key [in] tbsDataLen: Length of the data to be signed
Return Code	FBL_FOTA_SEC_STATUS_OK: job is triggered successfully. FBL_FOTA_SEC_STATUS_BUSY: HTA is busy, and job is not triggered. FBL_FOTA_SEC_STATUS_ERROR: HTA is faulty, and job is not triggered.
Functional Description	It triggers the FOTA Sign with Identity Key process. This is an asynchronous function that trigger the job. FBL_FOTA_Sec_Usr_GetSignWithIdKeyStatus can be used for polling the result.
Location	FBL_FOTA_SecStackUserCode.c
Pre-Conditions	none

Prototype	FBL_FOTA_Sec_StatusT FBL_FOTA_Sec_Usr_GetSignWithIdKeyStatus (void)
Parameter	none
Return Code	FBL_FOTA_SEC_STATUS_OK: job ends successfully. FBL_FOTA_SEC_STATUS_BUSY: job is still running. FBL_FOTA_SEC_STATUS_ERROR: job ends with errors.
Functional Description	It retrieves the status of the sign with identity key process.
Location	FBL_FOTA_SecStackUserCode.c
Pre-Conditions	none

Prototype	uint32 FBL_FOTA_Sec_Usr_GetSignatureLength(void)
Parameter	none
Return Code	uint32 containing the signature length

Functional Description	It retrieves the signature length provided by the HTA. The maximum size of the signature is 256 as reported in the DID 0xF1B7 and 0xF1B8.
Location	FBL_FOTA_SecStackUserCode.c
Pre-Conditions	none

Prototype	FBL_FOTA_Sec_StatusT FBL_FOTA_Sec_Usr_GetDataSignature (uint8* SignedData)
Parameter	[out] SignedData: pointer to an array that contains the signed data.
Return Code	uint32 containing the signature length
Functional Description	It retrieves the signature length provided by the HTA
Location	FBL_FOTA_SecStackUserCode.c
Pre-Conditions	none

4.4.1.12 FOTA - Periodic

Prototype	void FBL_FOTA_Sec_Usr_Mainfunction_1ms
Parameter	none
Return Code	none
Functional Description	Periodic function scheduled by the OS every 1ms
Location	FBL_FOTA_SecStackUserCode.c
Pre-Conditions	none

4.4.2 FCA Security Manager (FSM)

4.4.2.1 FSM CertStore, DisavowedCertificateList and TrustStore

The CS (Cert Store) contains the certificate chain of trust and the DCL (Disavowed Certificate List) according to the SD.00078 and the SD.00125.

The user shall assemble the CS and DCL considering that:

- CS and DCL shall have a separate SFBH
- CS and DCL shall be signed by a certificate that has the contains the extKeyUsage OID 1.3.6.1.4.1.29858.3.1.2.5
- CS and DCL shall be concatenated.

An example of a CS and DCL is reported in the figure below.

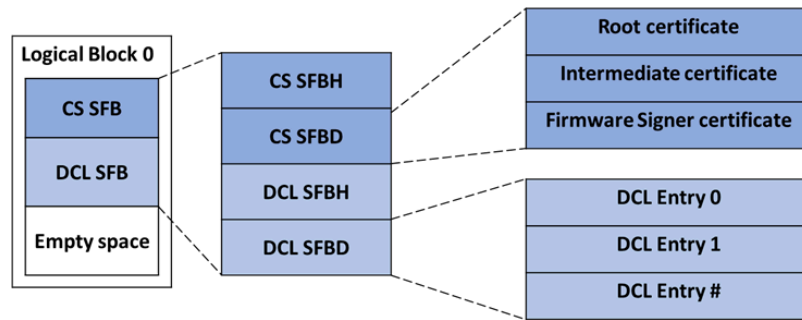


Figure 4.17. CertStore creation

The FBL calculates the CS and DCL SFBs position inside the logical block 0 as follow:

- The start address of the CS SFB is fixed and matches the start address of the logical block 0.
- The size of the CS SFB is dynamic and depends on the field size of the CS SFBH.
- The start address of the DCL SFB is dynamic and it's calculated at the run time by the FBL as: CS SFB start address +
- CS SFB size.
- The size of the DCL SFB is dynamic and depends on the field size of the DCL SFBH and the DCL SFB start address.

The logical block 1 shall not be used since both the CS and the DCL are part of the Logical block 0.

The CS contains the certificate that shall be added to the TS. It's not necessary that the CS contains all the certificate in the TS, but instead should contain those certificates to be added to the TS.

The DCL shall be always present, and it shall contain at least one certificate filled with 0xFFs if no certificate needs to be removed.

If only the DCL is to be installed, then the new Truststore will be generated from the old Truststore and the new DCL

As reported in the SD.00015_03, each DCL entry shall contains two fields: the issuer common name and the certificate serial number. Since the issuer common name and the serial number may have a size smaller than the reserved area, these two fields shall be padded as reported in the below figure.

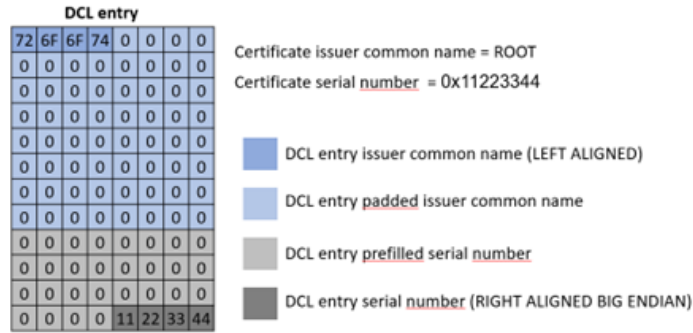


Figure 4.18.DCL entry format

The FSM interprets the certificate serial number field in the DCL in big endian representation.

The CS and the DCL are stored in the PFLASH and the whole content of the Logical Block 0 is protected with a CMAC by the CycurHsm and verified at the startup and before the usage.

The DID 2031 reports the last 10 DCL entries available.

At the first startup, the TS is generated based on the content of the CS only.

If a new CS/DCL is downloaded the TS is updated as follow:

- If a new valid certificate is added to the CS, it is also added to the TS.
- if a certificate is added to the DCL, the same certificate and all the leaf certificates that are validated by that certificate are removed from the TS.

NOTE
 It's suggested to leave the DCL "empty" (one entry filled 0xFFs) in the initial Certificate.

NOTE
 A DCL entry shall be never removed otherwise the related certificate can be readded to the CS and to the TS.

4.4.2.2 FSM Trusted Boot

The authenticated boot is responsible to prevent that a corrupted or illegally modified software executes on the ECU.

The FBL defines the Authenticated Boot table that contains all the information related to the Logical block that shall be verified by the HTA during the Authenticated boot phase.

A logical block may be verified in the following ways:

- `TB_BOOT_MODE_SECURE`: The Logical block CMAC verification is performed at the startup by the HTA. If the Logical block CMAC verification fails, the HTA doesn't release the Host and the FBL doesn't execute. This mode doesn't allow any recovery. This is the mode used to verify the FBL.
- `TB_BOOT_MODE_AUTHENTIC`: The Logical block CMAC verification is requested by the FBL during the startup. The FBL doesn't jump to the application until the HTA verify the logical block. If the Logical block CMAC verification fails, the FBL doesn't jump to the application and remains in boot. A new download is possible, and the corrupted region can be recovered. This is the default mode used to verify the APPs and the CALs.
- `TB_BOOT_MODE_AUTHENTIC_EXTENDED_RESET`: The Logical block CMAC verification is requested by the FBL during the startup. The FBL doesn't wait for the HTA verification and jump to the application. If the Logical block CMAC verification fails, the HTA triggers a reset. At the following reset the FBL detects that the HTA trigger the reset and erase all the regions marked with this flag. This mode can be used to decrease the startup time



NOTE

Only the application can be marked with the trusted boot mode `TB_BOOT_MODE_AUTHENTIC_EXTENDED_RESET`.

The FBL configuration of the TBT (Trusted Boot Table) is stored in the variable `AuthBootTable` located in the `Sec_Hal_PrivCfg.c` file. The `AuthBootTable` contains the FBL configuration of the blocks that shall be verified by the `CycurHsm`.

`Sec_Hal_UserCfg.h` can be updated to change the Trusted Boot mode for the Application and the Calibration entries. While it's not possible to change the FBL Trusted Boot Mode, it's possible to change the one of the calibrations and application blocks. By default, FBL block is set as `TB_BOOT_MODE_SECURE` and other as `TB_BOOT_MODE_AUTHENTIC`.

The `CycurHsm` provides the api `HSM_GetTBVerifTable` to retrieve information about the current blocks in the `CycurHsm` TBT. The two TBT, the one in the FBL and the one in the `CycurHsm` are different.

The FBL TBT contains the configuration of the logical blocks that shall be part of the `CycurHsm` TBT. `Trustore` is not part of the FBL TBT. The `CycurHsm` TBT contains the current area for which the `CycurHsm` has calculated the CMAC. At the first startup the `CycurHsm` TBT is empty, and it's populated after the first initialization and after each download.

FSM Bootloader first run

The FBL requires a valid CS to be flashed on the ECU before the first run of the FBL.

During the first run, the FBL performs the following tasks:

1. Initialize the HSM.
2. Request the CS verification and the TS generation from the HSM.
3. Store the TS and the TS backup in the PFLASH.
4. Request the TS protection from the HSM. During this step the lifecycle of the HSM is set to INFIELD1 and the CMAC is calculated for both the TS.
5. Verify the TS.
6. Calculate the CMAC of the configured logical block that are marked as TB_BOOT_MODE_SECURE.



NOTE

the lifecycle of the HSM switches to INFIELD1 despite any of the previous step fails. This will prevent the TS generation. The HSM shall be re-flashed to generate a new valid TS.

FSM Startup sequence

After the first run, the startup sequence is reported in the figure below.

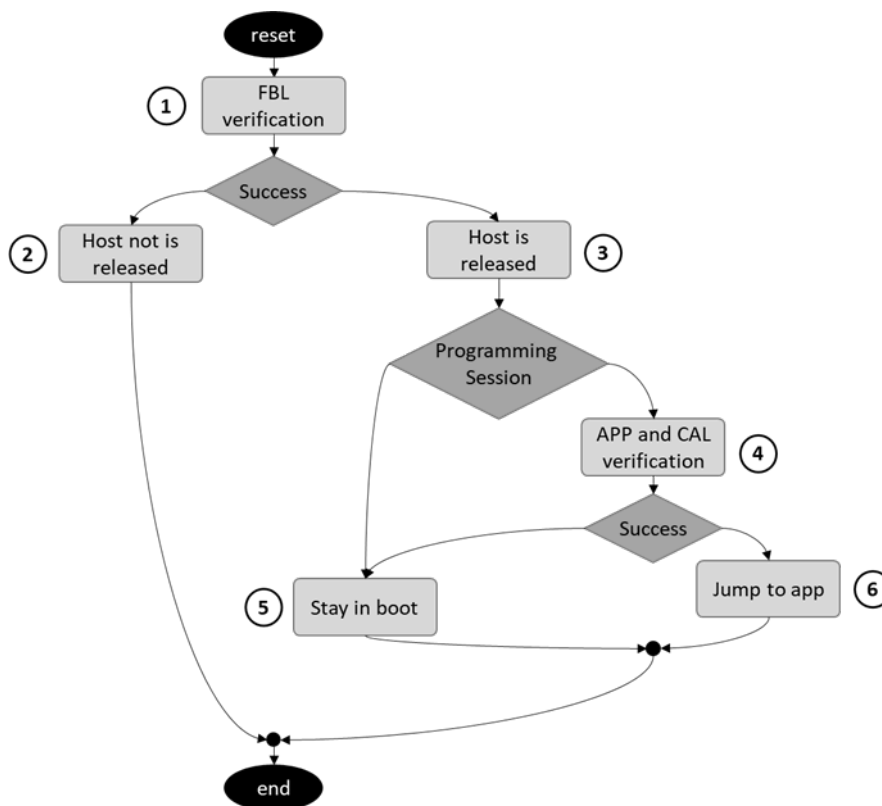


Figure 4.19. Authenticated boot Startup sequence

1. At the reset the HSM starts before the Host and verify the FBL (The FBL block verification mode is flagged as TB_BOOT_MODE_SECURE)
2. If verification fails, the HSM doesn't release the host.
3. After a successful verification, the HSM release the Host and the FBL starts. In this step the TS is verified, and remedial action are setup to recover from a corrupted TS. TS and TS backup shall be aligned.
4. If the programming session is not requested, the FBL requests the verification of the application and calibration logical blocks in the Trusted Boot Table.
 - If the FBL detects a reset triggered by the HTA, the FBL invalidate the block and erase the blocks marked as TB_BOOT_MODE_AUTHENTIC_EXTENDED_RESET from the Trusted Boot Table.
 - The FBL starts with all the blocks marked as TB_BOOT_MODE_AUTHENTIC and wait until the HSM finishes the verification.
 - If the FBL doesn't detect a reset triggered by the HTA, then the FBL moves to the blocks marked as TB_BOOT_MODE_AUTHENTIC_EXTENDED_RESET and just start the verification without waiting the output.
5. If verification fails (either TS, app or cal) or any blocks define in the Auth-BootTable is not present in the Trusted Boot Table, then the ECU remains in the FBL.
6. Once the SW verification is fine the FBL requests the jump to the application after calling ecy_hsm_Csai_Protection_DisableSecCritFunctions. This API disables some CycurHSM features. This is done only if no regions are marked as TB_BOOT_MODE_AUTHENTIC_EXTENDED_RESET, otherwise it's up to the App to call this API.

 **NOTE**

The above startup sequence doesn't report any MCAL initialization or any other check to determine application can run.

 **NOTE**

At the first startup only TS and FBL are present in the Trusted Boot Table. Cals and Apps are added to the Trusted Boot Table after a successful RC 0xF000 verify download.

4.4.2.3 FSM Trusted Download

The verification process is managed by the FSM and the CycurHSM and all the information can be found in their manuals.

As a summary:

- The verification process is triggered by the RID 0xF000 Check Program.
- The FBL requests to the CyscurHsm the verification of the downloaded block
- The CyscurHsm responds with a positive or negative response
- In case of positive response, the CyscurHsm provides the updated TS.
- The FBL store the TS in the flash area and requests the protection.

After a successful verification, the FSM doesn't include the SFBH and the CS area in the CyscurHsm TBT entry. The FBL updates the CyscurHsm TBT to include all the Logical Block area.

4.4.3 STLA Security Manager (SSM)

4.4.3.1 SSM CertStore, DisavowedCertificateList and TrustStore

The SSM integration code provided by the FBL uses the *Certstore as a Truststore* approach. This means that the CertStore and the Truststore are equal. Moreover, the logical block 1 shall not be used since both the CS and the DCL are part of the logical block 0. CS and TS main will be used as synonyms. The logical block 0 contains the CS/TS main and the DCL main.

The user needs to define the location of the CS/DCL region, the TS backup and the DCL backup. These regions shall have an appropriate size.

The user needs to configure the FBL with the following parameters:

- FblSecStackType shall be set to SSM
- FblRegion shall define an entry with the FblRegionId set to 0 for the CS/DCL logical block.
- FblRegion shall define an entry with the FblRegionType set to TS_BACK-UP_BLOCK for the TS backup
- FblRegion shall define an entry with the FblRegionType set to DCL_BACK-UP_BLOCK for the DCL backup
- For each FblRegion entry, the FblRegionSecTrustedBootMode shall be defined.
- For each FblRegion entry, the FblRegionSecCompatibilityId shall be defined.
- FblSecTsCapacity shall contains the number of the certificate in the TS. This parameter will be used to check the TS main and TS backup regions
- FblSecCsCapacity shall contains the number of the certificate in the CS. This parameter will be used to check the CS region.
- FblSecCrlCapacity shall contains the number of the dcl entry in the DCL. This parameter will be used to check the DCL main and DCL backup regions.
- FblSecTargetName shall contain the Ecu target name.

FblSecTsCapacity and FblSecCsCapacity shall be equal since the *Certstore as a Truststore* approach is used.

The CS (Cert Store) contains the certificate chain of trust and the DCL (Dis-avowed Certificate List) according to the SD.00078 and the SD.00125.

The user shall assemble the CS and DCL considering that:

- CS and DCL shall have a separate SFBH
- CS and DCL shall be signed by a certificate that has the contains the extKeyUsage OID 1.3.6.1.4.1.29858.3.1.2.5
- CS and DCL shall be concatenated and the DCL shall be aligned to the first available sector after the CS.

An example of a CS and DCL is reported in the figure below.

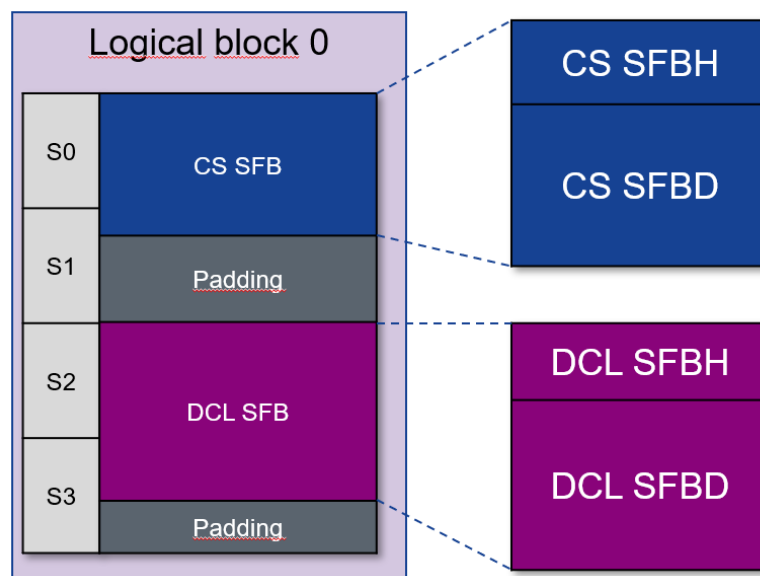


Figure 4.20.CertStore creation

The FBL calculates the CS and DCL SFBs position inside the logical block 0 as follow:

- The start address of the CS SFB is fixed and matches the start address of the logical block 0 configured.
- The size of the CS SFB is dynamic and depends on the field size of the CS SFBH.
- The maximum size of the CS SFB is set based on the parameter FblSec-sCapacity (check Table 4.5).
- The start address of the DCL SFB is fixed and it's located at the start address of the next sector
- The size of the DCL SFB is dynamic and depends on the field size of the DCL SFBH and the DCL SFB start address.

The DCL shall be always present, and it shall contain at least one dcl entry filled

with 0xFFs if no certificate needs to be removed.

As reported in the SD.00015_03, each DCL entry shall contains two fields: the issuer common name and the certificate serial number. Since the issuer common name and the serial number may have a size smaller than the reserved area, these two fields shall be padded as reported in the figure Figure 4.18. The SSM uses the same DCL entry of the FSM.

The DID 2031 reports the last 10 DCL entries available.

At the first startup, the TS backup and the DCL backup are generated based on the content of the CS/DCL block.

At each startup, the TS main, the TS backup, the DCL main and the DCL backup are verified and if any of them is invalid, it's restored:

- if the TS main is invalid and the TS backup is valid, the TS main is restored based on the content of the TS backup.
- if the TS backup is invalid and the TS main is valid, the TS backup is restored based on the content of the TS main.
- if both the TS main and the TS backup are invalid, the ECU is not able to recover and it will be stuck in the initialization phase.
- if the DCL main is invalid and the DCL backup is valid, the DCL main is restored based on the content of the DCL backup.
- if the DCL backup is invalid and the DCL main is valid, the DCL backup is restored based on the content of the DCL main.
- if both the DCL main and the DCL backup are invalid, the ECU is not able to recover and it will be stuck in the initialization phase.

If a new CS/DCL block is downloaded the TS backup and the DCL backup are updated after the verification of the CS SFB and the DCL SFB.



NOTE

It's mandatory to have the DCL in the initial CertStore.



NOTE

A DCL entry shall be never removed otherwise the related certificate can be readded to the CS and to the TS.

4.4.3.2 SSM Trusted Boot

The FBL implements the same logic of the FSM. Refer to FSM Trusted Boot

SSM Bootloader first run

The FBL requires a valid CS to be flashed on the ECU before the first run of the FBL.

During the first run, the FBL performs the following tasks:

1. Initialize the HSM.
2. Set the CycurHsm lifecycle to INFIELD1
3. Add the FBL to the TBT as trusted boot mode TB_BOOT_MODE_SECURE.
4. Request the CS verification and generate the TS backup and the DCL backup.
5. Verify the TS.
6. Verify the DCL.

SSM Startup sequence

The FBL implements the same logic of the FSM. Refer to FSM Startup sequence

4.4.3.3 SSM Trusted Download

The verification process is managed by the SSM and the CycurHSM and all the information can be found in their manuals.

As a summary:

The verification process triggered by the RID 0xF000 Check Program follows the following steps:

- Verification of the CS SFB. At the end of the this step the CS TBT entry is updated.
- Verification of the DCL SFB. At the end of the this step the DCL main TBT entries are updated.
- Updating of the TS backup. This step will not be reached if the CS/DCL SFBs are not valid. At the end of the this step the TS backup TBT entry is updated.
- Updating of the DCL backup. At the end of the this step the DCL backup TBT entries are updated.

If the verification of a new downloaded CS/DCL fails, the CS/DCL region is restored at the following startup. A new download is possible since the verification of a new downloaded block is based on the TS backup and the DCL backup.

After a successful verification, the SSM doesn't include the SFBH and the CS area in the CycurHsm TBT entry. The FBL updates the CycurHsm TBT to include all the Logical Block area.

4.5 Supported targets

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

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

4.6 Integrator guidelines

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

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

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

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

- NvM layout adaptation
- Boot jump handling

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

4.6.1 FBL: Memory Layout Adaptation

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

An example of a typical memory layout is depicted in Figure 4.21.

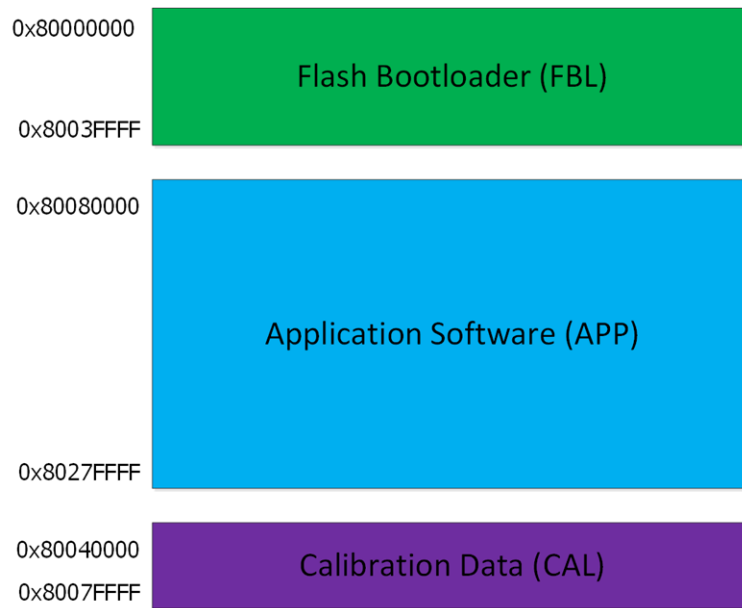


Figure 4.21. Sample memory layout

4.6.2 FBL: User Functions

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

Few of them must be filled correctly to be fully compliant with Stellantis norms, while others are optional.

4.6.2.1 Initialization

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

Prototype	<code>void Fbl_Port_UserConfigInitOne (void)</code>
Parameter	none
Return Code	none
Functional Description	Called at system startup during Flash Bootloader initialization, before the NvM has loaded the data flash.
Location	FBL_PortUserCode.c
Pre-Conditions	none

Prototype	<code>void Fbl_Port_UserConfigInitTwo (void)</code>
Parameter	none
Return Code	none

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

4.6.2.2 Shutdown

The user function `Fbl_Port_GoToSleep` should be filled to put the ECU in sleep mode, according to the ECU hardware configuration and sleep strategy. The callout is triggered according to [1] when the timers of section 5.2.6 Bootloader Sleep/Wakeup Mechanism are expired. When this functionality is enabled, the NvM block `NvM_ShutdownCorrectBlock` could be used to detect whether the previous shutdown was normal or abnormal, according to [1]

Prototype	<code>void Fbl_Port_GoToSleep (void)</code>
Parameter	none
Return Code	none
Functional Description	Called to put the system in shutdown
Location	FBL_PortUserCode.c
Pre-Conditions	none

4.6.2.3 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.

Prototype	<code>void Fbl_Port_WatchDogInitialise (void)</code>
Parameter	none
Return Code	none
Functional Description	Called to initialize the watchdog
Location	FBL_PortUserCode.c
Pre-Conditions	none
Prototype	<code>void Fbl_Port_WatchDogRefresh (void)</code>
Parameter	none
Return Code	none
Functional Description	Called to refresh the watchdog

Location	FBL_PortUserCode.c
Pre-Conditions	none

4.6.2.4 Application validation

At the end of the download the callout `Fbl_Port_UserValidApplication` is triggered to verify that the application is valid and compatible. The callout should be filled with application specific checks. When requesting the routine \$FF01 - Check Programming Dependencies the callout is executed only if the signature or CRC check on the blocks has been positive, otherwise a negative response is returned without triggering the user callback. Please note that when bit # 1 of DID 2010 is not set, the DTC P0602-00 is returned by FBL. The same should be done by the application if it is executed in limp mode.

Prototype	<code>boolean Fbl_Port_UserValidApplication (uint8 * isSwHwCompatible, uint8 * isSwDataCompatible)</code>
Parameter	<p><code>isSwHwCompatible</code>: pointer to software validity flag, it is used to update bit #1 and bit #5 of DID 2010</p> <ul style="list-style-type: none"> - Bit 1 Programming Status - Application - Bit 5 Software not Compatible with Hardware <p><code>isSwDataCompatible</code>: pointer to data validity flag, it is used to update bit #2 and bit #6 of DID 2010</p> <ul style="list-style-type: none"> - Bit 2 Programming Status - Data - Bit 6 Software not Compatible with Application Data
Return Code	<p>TRUE: the application will be executed; this value should be returned to execute a valid application or an application in limp mode</p> <p>FALSE: the application will not be executed and the ECU will remain in boot mode</p>
Functional Description	Called after an application software download, to verify the application validity and compatibility.
Location	FBL_PortUserCode.c
Pre-Conditions	none

4.6.2.5 Software Identification Update

After an application software update is successfully performed, the callout `Fbl_Port_DownloadSuccess` is triggered to execute user specific code. This callout could be used to update software identification values in NVM with the newer values, or to store the odometer for the last flash programming, or any other ECU specific use case. The callout is executed at the end of the download only if the signature check has been positive and the application has been considered valid.

Prototype	<code>void Fbl_Port_UserDownloadSuccess (void)</code>
Parameter	none

Return Code	none
Functional Description	Called after a successful download to execute user code
Location	FBL_PortUserCode.c
Pre-Conditions	none

4.6.2.6 External memory reprogramming

If a region is configured as EXTERNAL using the FBL parameter FblRegionExternalFlashSupport, following user callouts are triggered to manage erase and writing processes.

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. It will result in NRC 0x78 to RID 0xFF00 when Transport Protocol timeout expires. FBL_USER_FLASH_RESULT_FAILURE: Erase completed with failure. It will result in NRC 0x72 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. It will result in NRC 0x78 to Transfer Data service when Transport Protocol timeout expires. FBL_USER_FLASH_RESULT_FAILURE: Write completed with failure. It will result in NRC 0x72 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	<code>UserFlash_ReturnType UserFlash_CalculateBlockHash (uint32 TargetAddress, uint32 Length, uint8* Hash)</code>
Parameter	TargetAddress: address of the external memory device to calculate the Hash. Length: size of the data whose Hash should be computed. Hash: pointer to computed hash.
Return Code	FBL_USER_FLASH_RESULT_SUCCESS: Hash calculation completed successfully FBL_USER_FLASH_RESULT_PROCESSING: Hash calculation is in progress and requires additional time. It will result in NRC 0x78 to RID 0xD000xx service when Transport Protocol timeout expires. FBL_USER_FLASH_RESULT_FAILURE: Hash verification completed with failure. It will result in NRC 0x72 to RID 0xD000xx
Functional Description	Triggered when RID 0xD000 - Logical Block Hash is received for a region configured as external. The integrator should calculate the Hash of the memory region using the proper algorithm. Please note a cryptographic library for LTC_RIPEMD-160 is present in your generated code that could be used.
Pre-Conditions	None

Prototype	<code>UserFlash_ReturnType UserFlash_VerifyBlockSignature (uint32 TargetAddress, uint32 Length)</code>
Parameter	TargetAddress: address of the start address of the external region to verify the signature. Length: size of the region to be verified Hash: pointer to computed hash.
Return Code	FBL_USER_FLASH_RESULT_SUCCESS: Signature verification is done successfully FBL_USER_FLASH_RESULT_PROCESSING: Signature verification is in progress and requires additional time. It will result in NRC 0x78 to RID 0xF000xx service when Transport Protocol timeout expires. FBL_USER_FLASH_RESULT_FAILURE: Signature verification is done with failure. It will result in positive response with the status record byte set to 0x04 as a response to RID 0xF000 i.e., 0xF0 00 04

Functional Description	Triggered when RID 0xF000 - Check program is received for a region configured as external. The integrator should verify the signature of the memory region using the proper algorithm. Please note a cryptographic library for LTC_RIPEMD-160 is present in your generated code that could be used.
Pre-Conditions	None

Prototype	UserFlash_ReturnType UserFlash_FlashRead (uint32 TargetAddress, uint32 Length, uint8* DestAddressPtr)
Parameter	TargetAddress: address of the external memory device to be read. Length: size of the external memory device to be read. DestAddressPtr: output buffer.
Return Code	FBL_USER_FLASH_RESULT_SUCCESS: Read completed successfully. FBL_USER_FLASH_RESULT_PROCESSING: Read is in progress and requires additional time. FBL_USER_FLASH_RESULT_FAILURE: Read completed with failure.
Functional Description	The integrator should add the code to read from the external memory device.
Pre-Conditions	None

4.6.2.7 Authenticated Diagnostic Access

Prototype	Std_ReturnType ADA_UserCode_SecurityPolicyService (uint8 service, uint8 subService, uint32 userRole)
Parameter	Service : Requested diagnostic service. subService : Requested diagnostic subservice. userRole : Current active role mask.
Return Code	E_OK : Service is allowed. E_NOT_OK : Service is not allowed.
Functional Description	This function handles the service security policy. It checks if the requested service with the current active role is allowed or not.
Pre-Conditions	None

Prototype	Std_ReturnType ADA_UserCode_SecurityPolicyRID (uint8 rid, uint8 subService, uint32 userRole)
Parameter	rid : Requested diagnostic RID. subService : Requested diagnostic RID subservice. userRole : Current active role mask.
Return Code	E_OK : RID is allowed. E_NOT_OK : RID is not allowed.

Functional Description	This function handles the RID security policy. It checks if the requested RID with the current active role is allowed or not.
Pre-Conditions	None

Prototype	Std_ReturnType ADA_UserCode_SecurityPolicyDID (uint8 rid, uint8 subService, uint32 userRole)
Parameter	did : Requested diagnostic DID. subService : Requested diagnostic DID subservice. userRole : Current active role mask.
Return Code	E_OK : DID is allowed. E_NOT_OK : DID is not allowed.
Functional Description	This function handles the DID security policy. It checks if the requested DID with the current active role is allowed or not.
Pre-Conditions	None

4.6.2.8 Diagnostic

Prototype	Std_ReturnType Fb1_UserCode_DID2001_OdometerReadFunc (uint8* Data)
Parameter	Data: pointer to the data content fo the DID. The data size is 2 according to the [2] unless the FbIDid is updated.
Return Code	E_OK : DID is allowed. E_NOT_OK : DID negative response.
Functional Description	This function handles the value of the DID 2001.
Pre-Conditions	None

Prototype	Std_ReturnType Fb1_UserCode_DIDF1B9_FotaScomoId1ReadFunc (uint8* Data)
Parameter	Data: pointer to the data content fo the DID. The data size is 33 according to the [2] unless the FbIDid is updated.
Return Code	E_OK : DID is allowed. E_NOT_OK : DID negative response.
Functional Description	This function handles the value of the DID F1B9.
Pre-Conditions	None

Prototype	Std_ReturnType Fb1_UserCode_DIDF1BA_FotaScomoId2ReadFunc (uint8* Data)
Parameter	Data: pointer to the data content fo the DID. The data size is 33 according to the [2] unless the FbIDid is updated.
Return Code	E_OK : DID is allowed. E_NOT_OK : DID negative response.

Functional Description	This function handles the value of the DID F1BA.
Pre-Conditions	None

Prototype	Std_ReturnType Fb1_UserCode_DIDF1BB_FotaScomoId3ReadFunc (uint8* Data)
Parameter	Data: pointer to the data content fo the DID. The data size is 33 according to the [2] unless the FbIDid is updated.
Return Code	E_OK : DID is allowed. E_NOT_OK : DID negative response.
Functional Description	This function handles the value of the DID F1BB.
Pre-Conditions	None

Prototype	Std_ReturnType Fb1_UserCode_DIDF1BC_FotaScomoId4ReadFunc (uint8* Data)
Parameter	Data: pointer to the data content fo the DID. The data size is 33 according to the [2] unless the FbIDid is updated.
Return Code	E_OK : DID is allowed. E_NOT_OK : DID negative response.
Functional Description	This function handles the value of the DID F1BC.
Pre-Conditions	None

Prototype	Std_ReturnType Fb1_UserCode_DIDF1AA_StandardVersionInformationTypeReadFunc (uint8* Data)
Parameter	Data: pointer to the data content fo the DID. The data size is 11 according to the [2] unless the FbIDid is updated in the configurartion. The integrator can use the following define to set the value of the DID: DID_F1AA_RELEASE_VERSION, DID_F1AA_LICENSE_NUMBER, DID_F1AA_AUTOSAR_REVISION and DID_F1AA_SUPPLIER_ID
Return Code	E_OK : DID is allowed. E_NOT_OK : DID negative response.
Functional Description	This function handles the value of the DID F1AA.
Pre-Conditions	None

4.6.2.9 ECU Identity

Prototype	Std_ReturnType Fb1_Port_UserCheckCsrCondition (void)
Parameter	None

Return Code	E_OK : CSR generation allowed. E_NOT_OK : CSR generation not allowed.
Functional Description	This function enables or disables the CSR generation that is triggered by the RC 0xD00A
Pre-Conditions	None

4.6.2.10 FOTA Rollback

Prototype	Std_ReturnType Fbl_Port_UserCheckRollbackCondition (void)
Parameter	None
Return Code	E_OK: Rollback allowed. E_NOT_OK : Rollback not allowed.
Functional Description	This function enable or disable the Rollback feature that is triggered by the RC 0xD006
Pre-Conditions	None

Prototype	void Fbl_Port_UserFinalizeRollback (void)
Parameter	None
Return Code	None
Functional Description	This function can be used for customization after a successful Rollback.
Pre-Conditions	None

4.6.3 FBL: BSW adaptation

The BSW modules needed by RTA-FBL and configured in the generated BSW project are listed in Table 4.70. 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 4.70.MCAL modules list

BSW Module(s)	Notes
Dcm	The diagnostic communication module
Mem/IF; Fee; NvM	Memory stack modules for the NVM
CanIf; CanSM; CanTp; ComM; ComStack; PduR	Communication stack modules

Crc	Uses for CRC calculation when verifying the downloaded application/calibration.
-----	---

4.6.4 FBL: MCAL adaptation

The MCAL modules needed by RTA-FBL for all targets are listed in Table 4.71. 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 FBL Target Guide for further information on MCAL modifications for your target.

Table 4.71.MCAL modules list

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

4.6.5 FBL: OS adaptation

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

with a different frequency, then follow these steps:

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

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

**WARNING**

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

4.6.6 FBL: BLSM adaptation

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

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

**WARNING**

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

4.6.7 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.

The NvM memory layout of the FBL shall be a subset of the NvM memory layout of the Application. In order for the layout to be consistent, the Fee persistent IDs of the shared blocks must match between the application and FBL.

The default memory layout of the FBL is reported in [5].

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 FBL Target Guide for details of Fee Sectors configuration.

4.6.8 ASW: 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 shown in Figure 4.22.

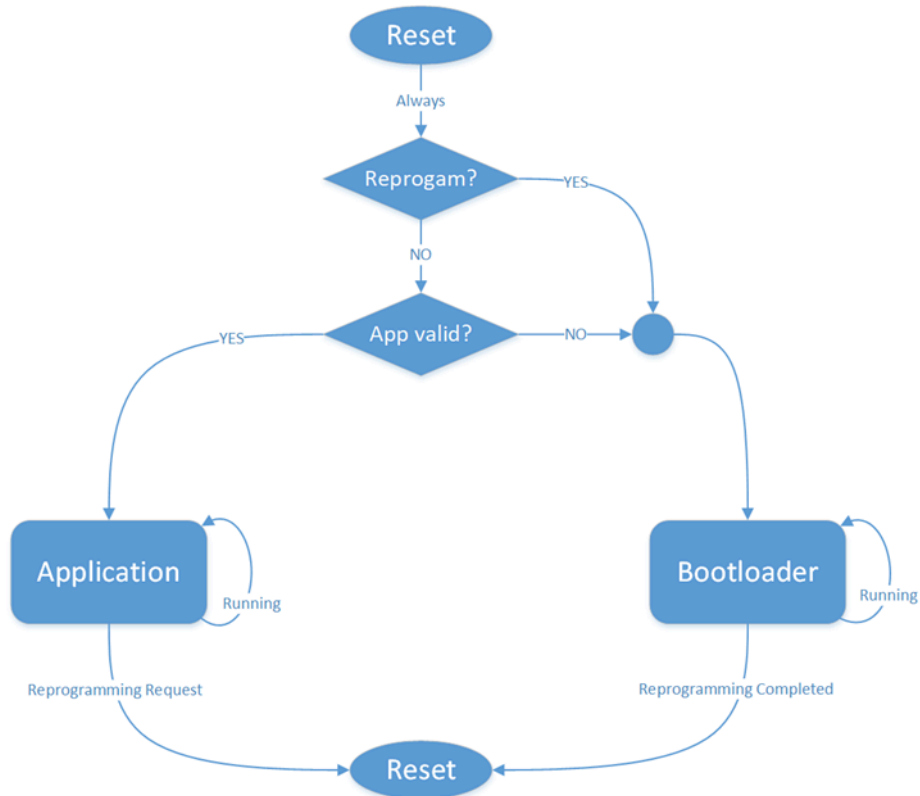


Figure 4.22. Handling of jump logic

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 4.23 for API description).

This callout must be implemented by the user. The goal is to store the required information in a place and format, the bootloader can access and understand.

The two required information are:

- The reprogramming request flag, a uint32 that shall be set by the application exactly to `0xA AFF55AAUL`,
- The parameter `ProgConditions` of the type `Dcm_ProgConditionsType` provided by Dcm.

[SWS_Dcm_00543] [

Service name:	Dcm_SetProgConditions	
Syntax:	<pre>Std_ReturnType Dcm_SetProgConditions(Dcm_OpStatusType OpStatus, Dcm_ProgConditionsType * ProgConditions)</pre>	
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 (NR 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 4.23.Dcm_SetProgConditions API from AUTOSAR SWS

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. Dcm uses the return value from the function DcmApp1_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 4.72 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. The NvM block to store the reprogramming request flag is NV_ReprogrammingRequestFlagBlock. The NvM block to store the programming conditions is NvM_ProgrammingConditionsBlock.

Table 4.72. 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
SubFnclId	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 DcmDslProtocolRx-TesterSourceAddr is correctly configured for each DcmDs/DcmDsl-Protocol/DcmDslConnections in BSW configuration.
ElapsedTime	18	4	Total elapsed time - Set by Dcm
ReprogrammingRequest	22	1	Reprogramming 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	Those bytes are currently 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_B00T 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.

If the ECU supports the ADA, the ECU can be kept authenticated after the jump just storing the certificate used for the authenticated security access in the NvM entry Nvm_ADACertificate.

The NvM entry Nvm_ADAsavedDate can be used to store the last NotBefore value of a valid certificate.



NOTE

Make sure to persist all NvM blocks before the reset.

4.6.9 FBL: NvM adaption

This chapter explains how to manage the NvM configuration through the FBL module and perform the following tasks:

- Use the default configuration stored in the RTA-FBL
- Add new NvM blocks
- Modify the default NvM blocks (id and size)
- Add new data in the NvM blocks
- Modify the default data of the NvM blocks (move data to other blocks, change id, size or default value)

The FBL will propagate the configuration to the impacted modules NvM, Fee and FBL_DataM. You can still manually configure your own NvM blocks with the Nvm/Fee module. These blocks will not be managed by the FBL and the FBL will not have visibility of that modules. That means that the FBL_DataM won't produce any API to manage the block.

The FBL default configuration is reported in [5] and [6].

In order to keep the default configuration, the fbINvmBlock and FblNvmData container shall be left empty. The RTA-FBL will use the default configuration stored in the generator.

You can add new NvmBlock container and NvmData container or modify the default configuration through the FBL modules as show in the below figure.

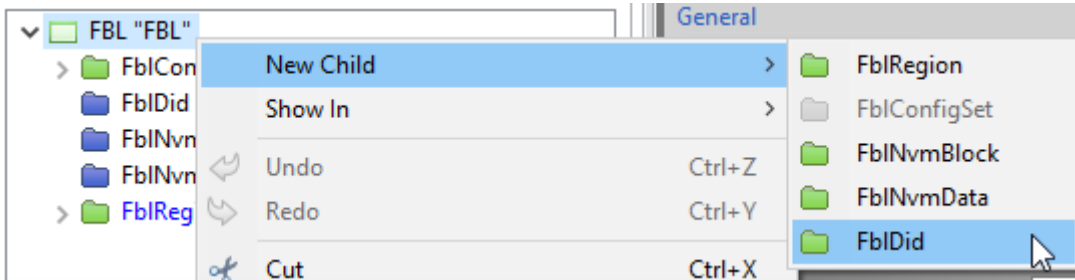


Figure 4.24. Menu to add new child container for FbINvmBlock, FbINvmData and FbINvmDid

Once the new child FbINvmBlock container is added to the configuration, you can create your block as show in the figure below.

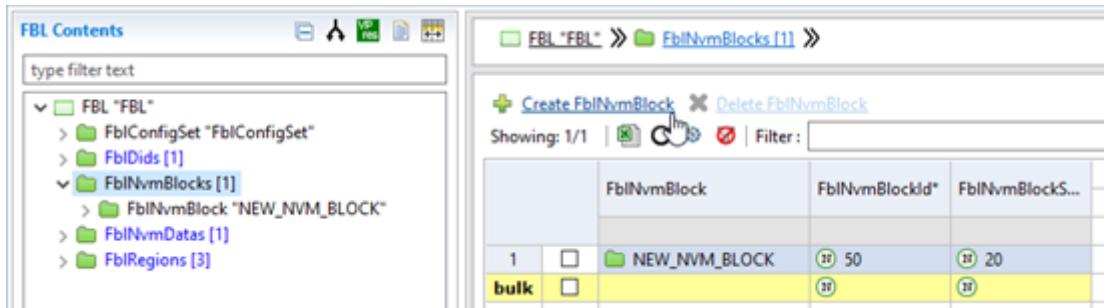


Figure 4.25. ISOLAR GUI for FbINvmBlock container with a new block configured. At the top two buttons allow the user to create or delete an entry

In the newly created FbINvmBlock you shall configure a unique ShortName. The ShortName can be the same of the ShortName of a default value. In this case you won't generate a new NvM block but you will overwrite the default NvM block.

You shall also define a block id and the size.

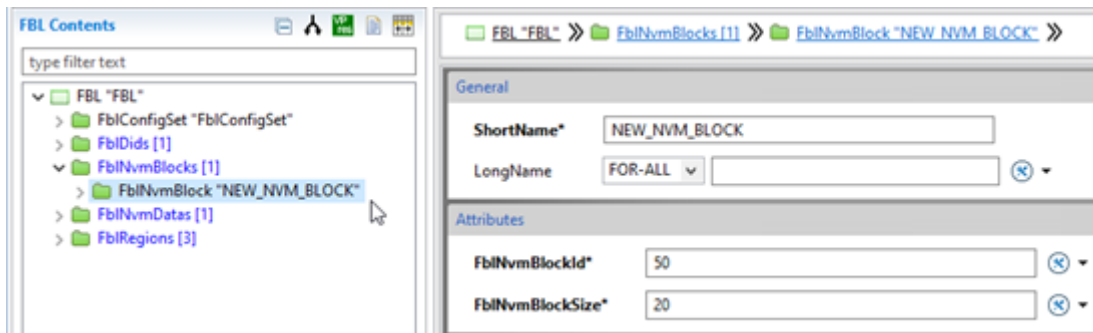


Figure 4.26. ISOLAR GUI for FbINvmBlock entry with a new block configured.

Once the FbINvmBlock is configured, you can populate it with one or more FbINvmData.

You can create the new child FbINvmData container and populate it with your

FbINvmData (same procedure as FbINvmBlock).

For each FbINvmData, you shall configure the ShortName. The ShortName can be the same of the ShortName of a default value. In this case you won't generate a new NvM data but you will overwrite the default NvM data.

You shall also configure the FbINvmBlock that contains the data, the offset inside the block, the size of the data and the default value.

The RTA-FBL will check that the FbINvmBlock configured for the FbINvmData exists and that it can contain the FbINvmData.

The default value is a string value that shall contain the hexadecimal value. For example if you configure the size to 3 you can set the value as follow:

- '20' all bytes will be automatically filled with the value generating the following value {0x20U, 0x20U, 0x20U}
- '010203' the bytes will be set as follow {0x01U, 0x02U, 0x03U}



NOTE

the FBL will not check the content of the FbINvmData so refer to [2].

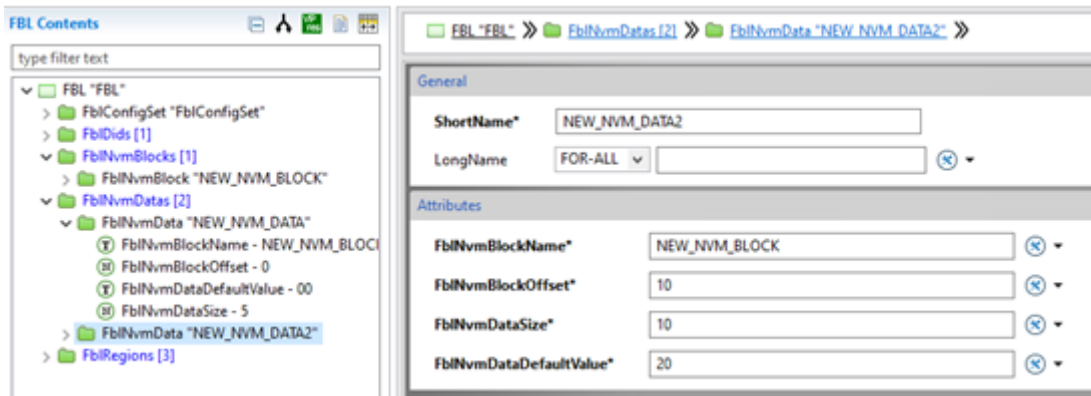


Figure 4.27. ISOLAR GUI for FbINvmData entry with a new data configured.

Once the FBL and the BSW are generated, you will find your block propagated in the NvM and Fee modules.

You can access the FbINvmData through the FBL_DataM APIs:

- FbI_DataM_Nvm< SHORT_NAME >ReadFunc: read data from the non-volatile memory
- FbI_DataM_Nvm< SHORT_NAME >WriteFunc: write data to the non-volatile memory
- FbI_DataM_Nvm< SHORT_NAME >IsPersisted: check data is actually written in the non-volatile memory

4.6.10 FBL: DID adaption

This chapter explains how to manage the DIDs configuration through the FBL module and perform the following tasks:

- Use the default configuration stored in the RTA-FBL
- Add new DIDs
- Modify the default DIDs (id, type or size)

The FBL will propagate the configuration to the impacted modules: Dcm and FBL_DataM.

You can still manually configure your own DIDs within the configuration of the dcm. These DIDs will not be managed by the FBL and the FBL will not have visibility of that DIDs. That means that the FBL_DataM cannot be used to manage that DID.

The FBL Did default configuration is reported in [4].

In order to keep the default configuration, the FblDids container shall be left empty. The plugin-in will use the default hard coded configuration.

In order to add a new DID, you shall create a new FblDids container and rename it as FblDids. Inside the FblDids container you can create a new FblDid pressing on the "Create FblDid" button on the top left of the BSW Editor.

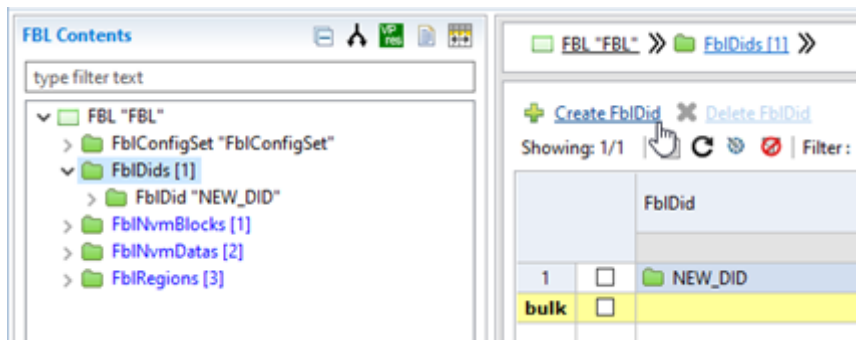


Figure 4.28. ISOLAR GUI for FblDid container with a new DID configured. At the top two buttons allow the user to create or delete an entry

The new FblDid shall have a unique FblDid ShortName. This means that the Short-Name shall not be equal to any default DIDs name or to other FblDid ShortName in the ISOLAR configuration. If the FblDid ShortName is equal to a default DID name, then the default DID will be overwritten.

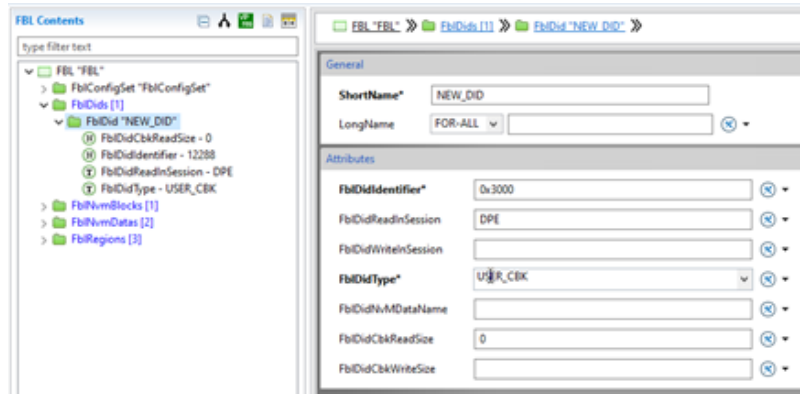


Figure 4.29. ISOLAR GUI for FbIDid entry with a new block configured.

For each DID you shall define the session in which the DID can be read or can be written. For this value you can use D for default session, E for extended session or P for programming session. So the string 'DPE' enables all the sessions.

You shall also define the FblDidType with one of the following values:

- NVM_DATA read/write data direct from NvM. FBL_DataM will manage the dcm callback.
- CALLBACK read/write through a callback. FBL_Port will manage the dcm callback.
- USER_CBK read/write through a callback. you shall define the dcm callback.

If the DID default type is CALLBACK, you can change the type to USER_CBK to overwrite the default callback implementation. In this case, you can define the callback in any source file you want. The default callback implementation is managed by the FBL_Port.c component and they will be excluded with a preprocessor #define. These #define are generated in the FBL_Port_Cfg.h file.

If the DID default type is NVM_DATA, the user can change the type to USER_CBK or CALLBACK and this makes no difference.

In case the NVM_DATA is set, you shall define the FblNvmData associated to the DID. In this case, the FblNvmData short name shall be defined in the FblNvmData table.

In case the CALLBACK is set, the user shall define the FblDidCbkReadSize and the FblDidCbkwriteSize.

The following api shall be populated:

```

FUNC(Std_ReturnType, DCM_APPL_CODE) Fbl_Port_<DID_SHORT_NAME>ReadFunc
(
    P2VAR(uint8, AUTOMATIC, DCM_APPL_DATA) dataOut1
)
    
```

4.7 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 a special application, named Boot Updater.

The Boot Updater is downloaded just like a normal application: in its code flash it holds the new flash bootloader and, once started, replaces the existing Bootloader by the new version.

Please note a boot manager is required to ensure the bootloader update process is reliable.

On reset, the boot manager will check if the FBL is valid and jump to the bootloader if it is valid. If the bootloader is not valid, then the boot manager will jump to the application. This is because if the bootloader is invalid, then this would signify the boot updater application must have failed in a previous update attempt and will need to run again. Note that the boot manager is the only non-replaceable software and is used to ensure boot update is failure safe. It will hold a reference to the start address of the bootloader and the start address of the application, that cannot be changed across bootloader updates.

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

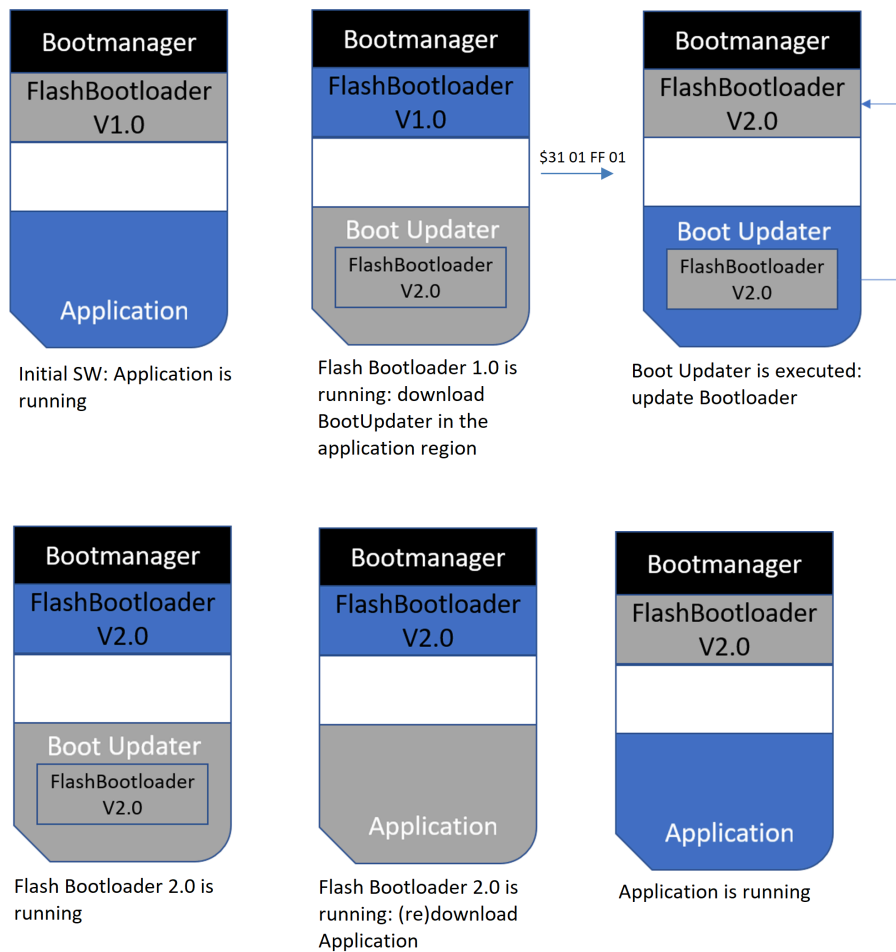


Figure 4.30. Example of the sequence of flash memory blocks updates during a bootloader update

process.

As depicted in previous pictures, the boot updater is executed during RID 0xFF01 - Check Programming Dependencies processing. The user macro FBL_PORT_CHECK_BOOT_UPDATER is used to detect whether a boot updater was downloaded and should be executed during RID processing or not. While the actual boot update is in progress, one or more UDS NRC \$78 may be issued to prevent UDS timeouts to expire. Positive response to RID 0xFF01 is only sent by FlashBootloader V2.0 after the boot update process has been completed.

Please note even if tester writes bootSoftwareFingerprint (\$F183) addressing logical block #0 (i.e. the bootloader logical block), the FBL expects the download of a boot updater application, and the above-mentioned process would take place.

The plugin includes a sample boot updater application.

The boot updater application is generated using the same parameters of the FBL.

Table 4.73.FBL parameters used by the Boot Updater.

Parameter	Description
FblEraseMaxSizePerCycle	Split the erase routine in multiple calls.
FblRegion	Configures the erase routines and the target linker-script
StartAddress	Identifies the application used as boot updater.

The boot updater application implements the following logic:

1. Reorganize the TBT: The FBL is removed from the TBT and the Boot Updater is marked as TB_BOOT_MODE_SECURE.
2. calls the BootUpdater_PreEraseHook(). The integrator may use this api for custom operation (ex: update the NvM, refresh the watchdog).
3. Erase the FBL. The CycurHsm suspension api are already managed by the Boot Updater erase functions.
4. calls the BootUpdater_PostEraseHook(). The integrator may use this api for custom operation
5. calls the BootUpdater_PreWriteHook(). The integrator may use this api for custom operation
6. Write the new FBL. The CycurHsm suspension api are already managed by the Boot Updater write functions.
7. calls the BootUpdater_PostWriteHook(). The integrator may use this api for custom operation
8. Restore the TBT: The FBL is added to the TBT and it's marked as TB_BOOT_MODE_SECURE and the Boot Updater trusted boot is restored.
9. Invalidate the NvM validity flags of the Boot Updater since at the next startup it shall run.
10. calls the BootUpdater_BootUpdateSuccess(). The integrator may use this api for custom operation.

11. Reset.

If any error occurs, the api `BootUpdater_BootUpdateFailed()` is called and the failing state is stored in the variable `BootUpdater_StateError`.

The Boot Updater share the same BSW of the FBL.

In particular at the startup the Boot Updater reads the `TesterSourceAddr` stored the in the Nvm block `NvmProgConditions`.

The dcm is not initialized, and the Boot Updater directly calls the `CanIf_Transmit` to send the pending response. The variable `TesterSourceAddr` is used to identify the tester and choose the correct Pdu.

4.8 Authenticated Diagnostic Access

According to the CS.00092, CS.00101 and CS.00121 the ADA module supports the following features:

- Authenticated security access.
- Delay timer.
- Certificate expiration and monotonic counter.
- Role management.
- Secure policy management.
- Authenticated security access from App. Check ASW: Boot Jump Handling for more details.

See the following chapter for more information.

4.8.1 Authenticated security access and Delay timer

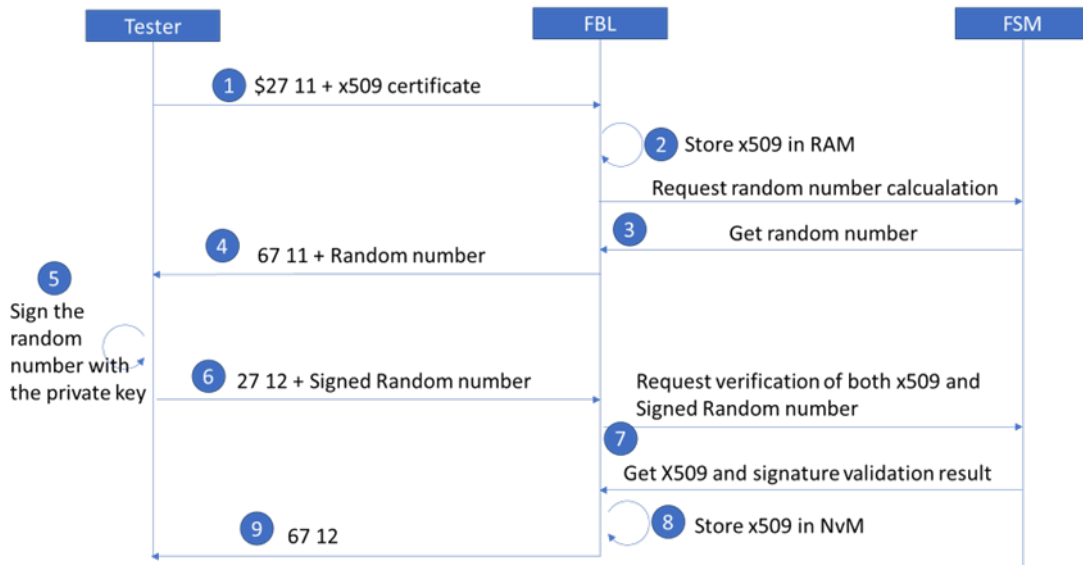


Figure 4.31. Sequence diagram reporting a successful security access with services 27/11 and 27/12

The above figure reports the step required by the ADA:

1. The tester requests the service 27 11 and send the x509v3 certificate to the FBL.
2. The FBL store the certificate in the RAM. The x509v3 certificate is not verified during the service \$27 11 callback routine.
3. The FBL requests the generation of a random number generation to the HTA.
4. The FBL responds to the tester with a positive response.
5. The tester signs the random number.
6. The tester sends the signed random number.
7. The FBL requests the verification of the x509v3 certificate and the signed random number to the HTA.
8. The FBL store the x509v3 certificate in the NvM entry Nvm_ADACertificate.
9. The FBL responds to the tester with a positive response.

The Authenticated security access supports the NRC reported in the figure below with the following exception:

- X509v3 certificate is validated during the service \$27 12. The service \$27 11 handles only the random number generation.
- At the startup the Delay timer is reset and the user shall wait that this timer expires before request an ADA.

Annex A
Support Diagrams

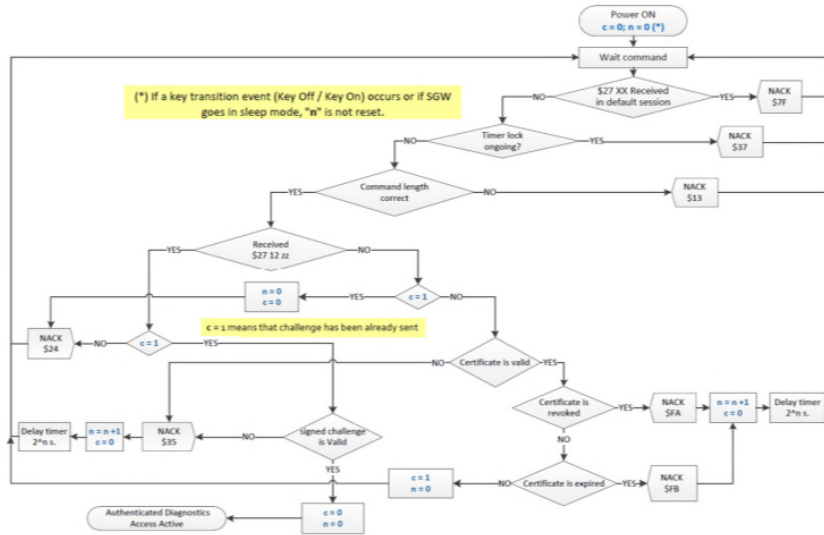


Figure A-1 - UDS Service 27 Negative Response Codes (NRC) Management for SGW

NOTE1: \$2711 and \$2712 commands are requested to be implemented in Extended Session, but if the SGW is in boot condition and Extended Session is not supported in boot, SGW shall in any case manage \$2711 and \$2712 commands.

NOTE2: See Figures A-2 and A-3 for actual NRCs and NRC implementation notes

NOTE3: NACK = Negative Acknowledgment

Figure 4.32. Authenticated security access workflow

4.8.2 Certificate expiration and monotonic counter

The last available NotBefore entry of a valid x509v3 certificate is stored in the Nvm entry Nvm_ADAsavedDate.

At the startup the x509v3 certificate of the cert store are used to define the initial value of the Nvm_ADAsavedDate.

During an authenticate security access, The Nvm_ADAsavedDate entry is used to verify the NotAfter value of x509v3 certificate. If the x509v3 certificate is valid and its NotBefore value is greater than the stored Nvm_ADAsavedDate then the Nvm_ADAsavedDate is updated.

When the ECU is authenticated by the ADA, a monotonic counter is set to the NotBefore value of the x509v3 certificate used for the ADA.

Until the ECU is authenticated, the FBL periodically increase the monotonic counter and it checks if the certificate is still valid or not. If the certificate expires, due to the monotonic counter reaching the NotAfter value of the certificate itself, then the Nvm_ADAsavedDate is updated with the current monotonic counter value to avoid that the same certificate can be reused. Then a reset is call and the current session is terminated.

4.8.3 Role and Security policy management

X509v3 certificates for ADA store a role that is used for the secure policy management. Since the certificate is stored on the FBL NVM, the FBL checks periodically the certificate validity.

If the certificate is no more valid, the certificate is removed from the NVM entry and the current session is terminated.

When the ECU is authenticated, the role is compared against the security policy. It's up to the integrator to define the Security policy populating the APIs in the file ADA_UserCode.c.

4.9 FOTA Rollback

This section describes the rollback feature.

As reported in the CS.00180, if a FOTA process fails, the FOTA master can request the rollback to all the ECU and restore the previous version of the software.

The ECU shall store a valid software version in a backup area. It's up to the application to create the backup.

The FOTA rollback can be requested both in the application or in the FBL. Once it's requested the ECU shall automatically move to the boot loader mode and restore the software from the backup.

The FOTA master will poll the ECU to request feedbacks on the process.

Once the FOTA rollback ends, the ECU remains in the boot loader mode until the FOTA master requests a reset.

Once the FOTA rollback starts, the ECU will respond only to the FOTA master until a reset is requested.

It's not possible to define a backup region for the FBL. If the Fsm and the CyclicHsm are integrated, it's not possible to define a backup region for the main TS and backup TS.

4.9.1 FOTA Rollback configuration

To enable and configure the feature, the ISOLAR parameter FblFotaRollback to ENABLE. This option is available only if FblFotaType is not set to FOTA_DISABLED and FblEcuType is set to ECU_C.

The generator performs the following checks:

- Rollback region ids are unique (No multiple rollback region for one region)
- Rollback region id links to a valid block region id.
- Rollback region size is equal to the block region size.
- All region id links to a rollback region id except for the FBL.
- Rollback region shall not overlap to other regions (block regions, trust stores).
- The CS and the FBL shall not have a rollback region.

4.9.2 FOTA Rollback functional behavior

The Rollback can be requested both by the FBL and the App. The Rollback is requested by writing 0x01 in the NvM data NvmRollbackStatus.

If the application manages the rollback request, then the value of the NvmRollbackStatus shall be persisted and a reset shall be called.

If the Fbl manages the rollback request, then the user can reject the requests implementing the callout Fbl_Port_UserCheckRollbackCondition. It's up to the application software to populate the backup regions and the UserCheckRollbackCondition can be used to inform the FBL that the backup is available.

It's not possible to requests a rollback if a previous rollback is already executing. The ECU will report a negative response with the NRC set to "conditions not correct".

The figure below shows the flowchart of the RC 0xD006 sub 01 when the request is managed by the FBL.

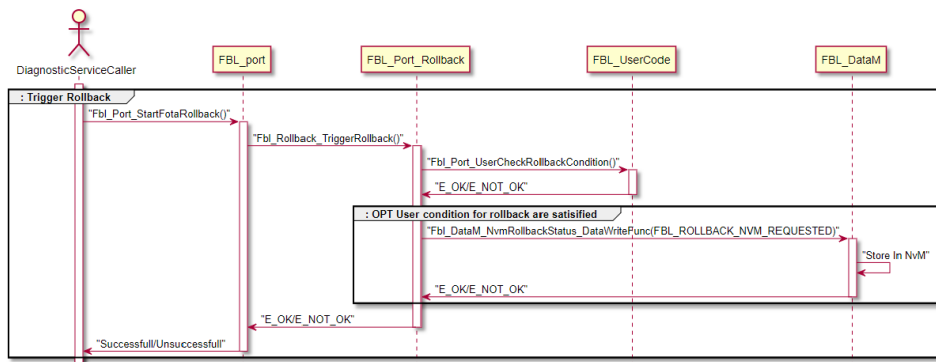


Figure 4.33. Flowchart of the RC 0xD006 01.

After the reset, the FBL detects the rollback requests and it remains in the default session.

If the FBL_Port_Rollback_Manager detects that a rollback is requested, it updates the status of the NvM data NvmRollbackStatus to 0x02 (Pending execution).

At the beginning of the rollback, the NvM data related to the DID 0x2010 and the logical block status are invalidated.

The pending execution status is not used to inform the tester about the execution status since there is a dedicated api to perform this task. This status is used as a recovery action after an exception is thrown and a reset is performed.

During the erase process, if the backup region is external then the UserFlash_FlashRead is used to read from the external device.

After a successful download, the fingerprints are restored and the user callout Fbl_Port_UserFinalizeRollback is called.

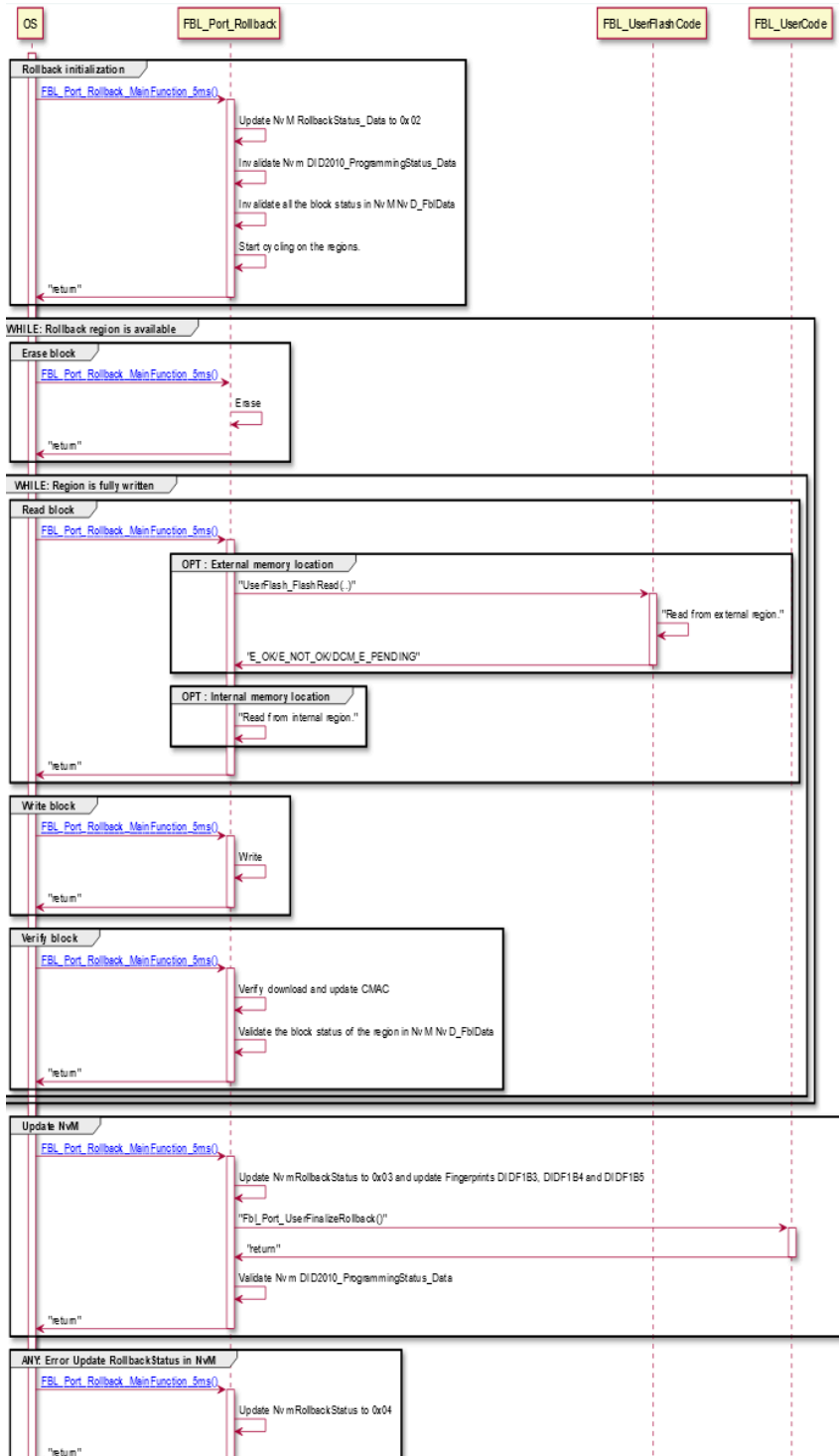


Figure 4.34. Simplified flowchart of the Rollback manager with the callout calls.

If the Rollback is successful, the FBL update the NvM data NvmRollbackStatus and the information about the DID 0x2010. The ECU remains in boot mode until the FOTA Master requests a reset.

4.10 FOTA ECU Identity

This section describes the ecu identity feature.

The ECU Identity generation is triggered by the RC 0xD00A. As shown in the figure below, the Dcm callouts is called multiple times and it returns a pending response until the private key and the CSR are generated and stored in the NVM.

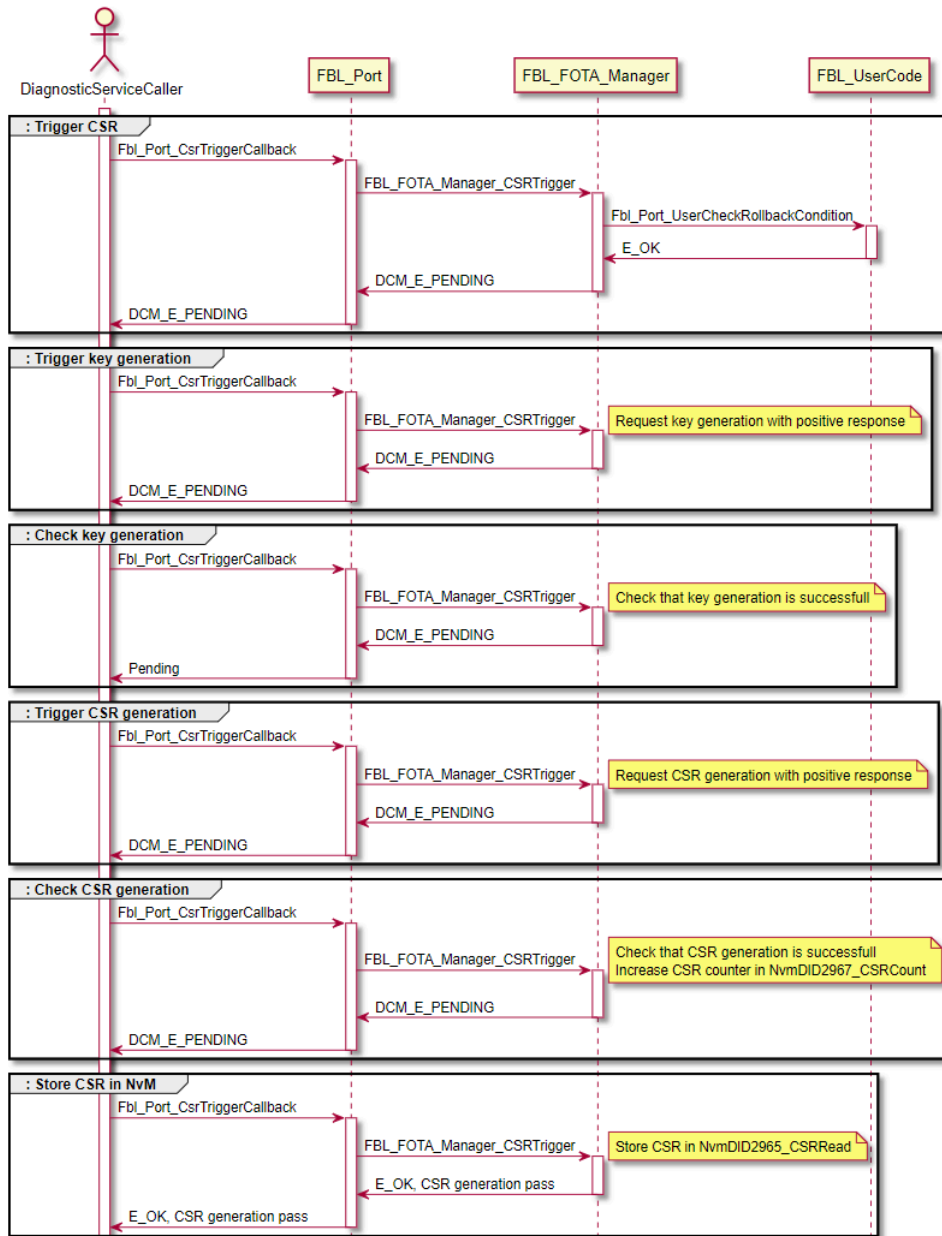


Figure 4.35. Simplified flowchart of a successful Ecu identity generation with the callout calls.

In case of any error during the RC 0xD00A execution, the CSR counter and the CSR remains unchanged.

After the CSR is generated:

- the CSR value is stored in the NvmDID2965_CSRRead and it can be read with the DID 0x2965
- The DIDF1B7 and the DIDF1B8 can be used to sign the Software Id Signature and the Hardware Id Signature

The algorithm supported by the CysurHSM for the signature of the content of the DID 0xF1B7 and 0xF1B8 is the ECDSA.

This algorithm generates a random signature also if the digest and the key are the same.

Since the FOTA master requires that the FOTA target ECUs send always the same signature if the digest doesn't change, the FBL provides the following strategy.

The FBL provides 4 Nvm data to store the digest and the signature of the two dids: Nvm_DIDF1B7_Signature_Data, Nvm_DIDF1B7_Digest_Data, Nvm_DIDF1B8_Signature_Data and Nvm_DIDF1B8_Digest_Data.

When the dids are called the FBL shall calculate the new signature and the new digest.

If the new digest is equal to the digest stored in the Nvm, the FBL sends the signature stored in the Nvm and ignore the new calculated digest and signature

If the new digest is not equal to the digest stored in the Nvm, the FBL stores the new signature and the new digest in the Nvm and it sends back to the tester the new calculated signature.

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

6.1 ETAS HQ

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

6.2 ETAS Subsidiaries and Technical Support

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

