ETAS RTA-FBL v1.3.3
GM Port

User Guide

www.etas.com

ETAS RTA-FBL | User Guide V1.3.3 EN – 12.2024

Contents

1 Safety and Privacy Information

1.1 Intended Use

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

1.2 Target Group

This user manual is targeted to users of RTA-FBL. Users include individuals generating RTA-FBL instances using the provided configuration tools, integrators who integrate the FBL into the final ECU, and test engineers ensuring the correct behaviour of the bootloader within the complete integrated system.

1.3 Classification of Safety Messages

The safety messages used here warn of dangers that can lead to personal injury or damage to property:

⚠ DANGER

DANGER indicates a hazardous situation that, if not avoided, will result in death or serious injury.

⚠ WARNING

WARNING indicates a hazardous situation that, if not avoided, could result in death or serious injury.

⚠ CAUTION

CAUTION indicates a hazardous situation that, if not avoided, could result in minor or moderate injury.

NOTICE

NOTICE indicates a situation that, if not avoided, could result in damage to property.

1.4 Safety Information

This software is qualified following the ETAS PEP project to QM level. It does not meet ISO 26262 ASIL requirements.

1.5 Definitions and Abbreviations

Term/Abbreviation	**Definition**
ADC	Analogue to Digital Convertor
AR	AUTOSAR
Application Software (Application Software)	This is the software that executes the control logic of the ECU
AUTOSAR	AUTomotive Open System Architecture
BLSM	Bootloader State Manager
BSW	Basic Software
CAN	Controller Area Network
CAN FD	CAN Flexible Datarate
Dcm	Diagnostic Communication Manager
DiD	Data iDentifier
DPS	Development Programming System – A tester tool provided by GM
ECU	Electronic Control Unit
FBL	Flash Bootloader
Fee	Flash EEPROM Emulation
HSM	Hardware Security Module (SP)
MCAL	Micro-Controller Abstraction Layer
NRC	Negative Response Code from the ECU
NvM	Non-Volatile Memory
OS	Operative System
PEC	Program Error Code as defined in [3]
RTA-x	The ETAS suite of embedded SW products
SP	Security Peripheral (HSM)
UDS	Unified Diagnostic Services

1.6 References

Ref.	**Document Name**	**Ver.**
[1]	GB6000 Unified Diagnostic Services Specification	v 2.2 (March 17, 2017)
[2]	GB6001 Diagnostic Infrastructure Specification	MY22 Version: Version: 4.1 (02/06/2019)
[3]	GB6002 Bootloader Specification	Version 2.0 Jan-13-2021

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 the RTA-FBL Port for GM must be installed and used in order to allow you to create a complete GM RTA-FBL bootloader instance. It includes important steps required for integrating RTA-FBL with your Application Software. For targets that support boot update, this chapter also details how to create a boot updater and prepare a bootloader image for flashing.
Chapter 4	This chapter contains ETAS references for customer support.

2 Introduction to ETAS RTA-FBL

This chapter 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 Chapter 3.

2.1 What is a Flash Bootloader?

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

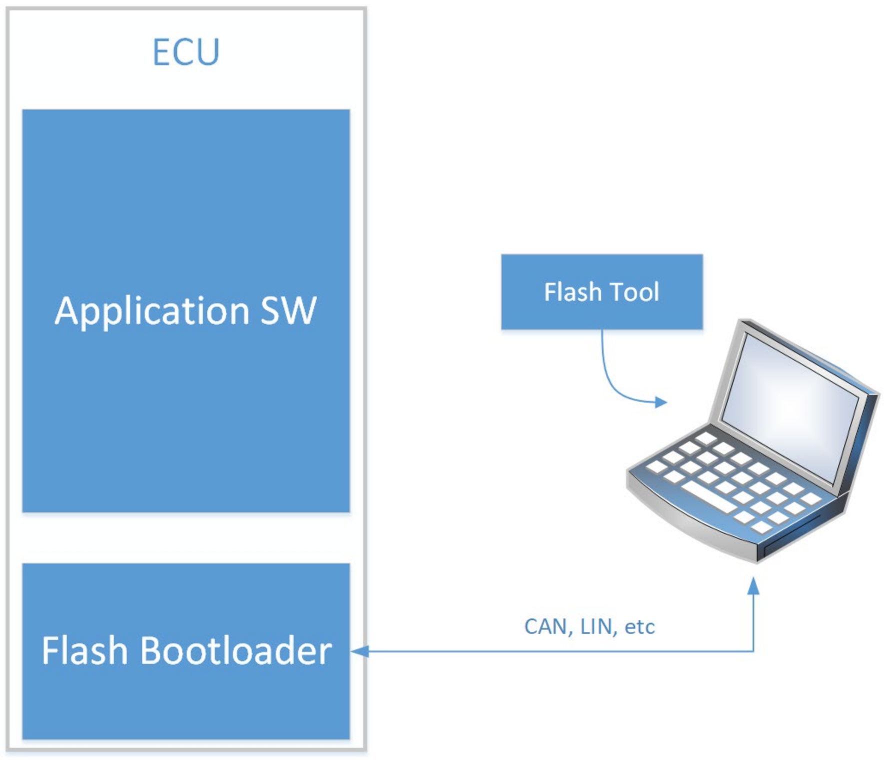


Figure 1: High level flashing process

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

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

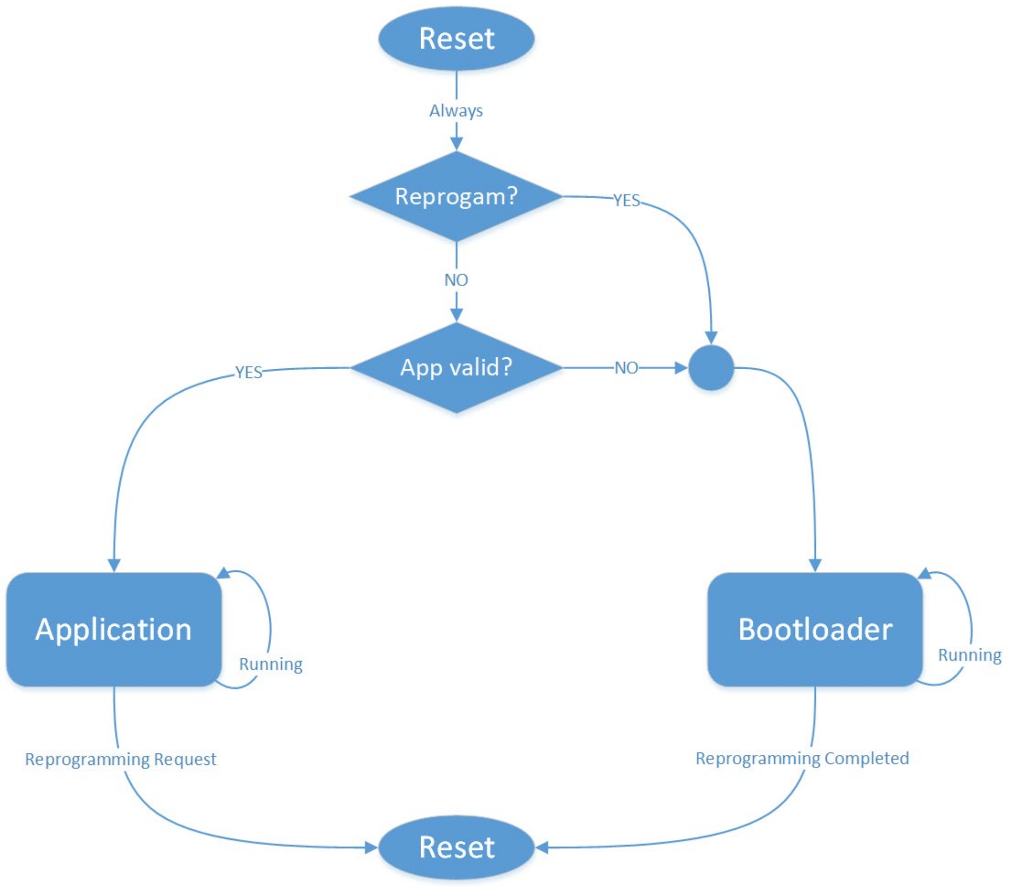


Figure 2: Boot loading flowchart

2.2 What is RTA-FBL?

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

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

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

Basing the underlying SW architecture on AUTOSAR allows support of other communication protocols such as CAN-FD, Ethernet, FlexRay, LIN.

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

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

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

2.3 The Flash Tool (Tester)

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

2.4 The OEM-defined Programming Sequence

The tester communicates with the ECU by sending messages over a communication bus according to a defined protocol. For example, some ports of ETAS' FBL supports UDS on the CAN protocol. This means that requests are made to the ECU over a CAN bus, and the messages sent and received comply with the UDS standard ISO 14229-1[2]. The allowed message sequence sent to the ECU, as well as the expected response from the ECU differs across OEMs. Therefore, the ETAS FBL supports different OEM standards for ECU reprogramming. These are called "OEM ports" or just "ports". This guide specifically addresses the RTA-FBL port that implements the reprogramming standard described in [3]. Each port supports one or more hardware "targets".

2.5 Target Dependencies and the Flash Driver

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

2.6 Interaction with the Application using NvM

A Bootloader and the Application Software may need to share data. For example, a Tester may read or write data such as the ECU serial number both when the ECU is running in boot-loader mode and when running its Application Software (e.g. by using UDS ReadDataByIndentifier and WriteDataByIdentifier commands). Typically, this will mean that both the Bootloader and the Application Software will need to be able to read and write the same non-volatile memory. Where non-volatile memory is implemented by EEPROM emulation in flash such sharing may introduce technical challenges because the Bootloader and Application Software must use the same algorithms and data-structures when emulating EEPROM. (For example, if the application uses an AUTOSAR Fee module for EEPROM emulation then the Bootloader may need to use the same Fee module). The requirements for compatibility between the FBL and Application Software for your port are detailed in Chapter 3.

2.7 One and Two-Stage Bootloaders

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

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

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

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

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

2.8 Updating the bootloader

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

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

An instance of ETAS’s FBL is generated based on the chosen OEM specification that defines the reprogramming sequence, the chosen hardware target, and the specific configurations that are allowed within the scope of the OEM specification. The tool for generating this FBL instance is an ISOLAR-AB plugin, which is included with your purchased core license. An FBL generated using this plugin is described as “an instance of RTA-FBL”. The plugin creates bootloader code as well as a full RTA-BSW project with configuration that is needed to support the bootloader functionality. In the same generation process, the plugin therefore optionally also invokes RTA-BSW to generate an instance of the BSW. Alternatively, the user can open the RTA-BSW project created by the RTA-FBL plugin to inspect the generated configuration. FBL generation also results in some ports in the generation of an MCAL project that can be adapted. Further details relevant to your port are provided in Chapter 3.

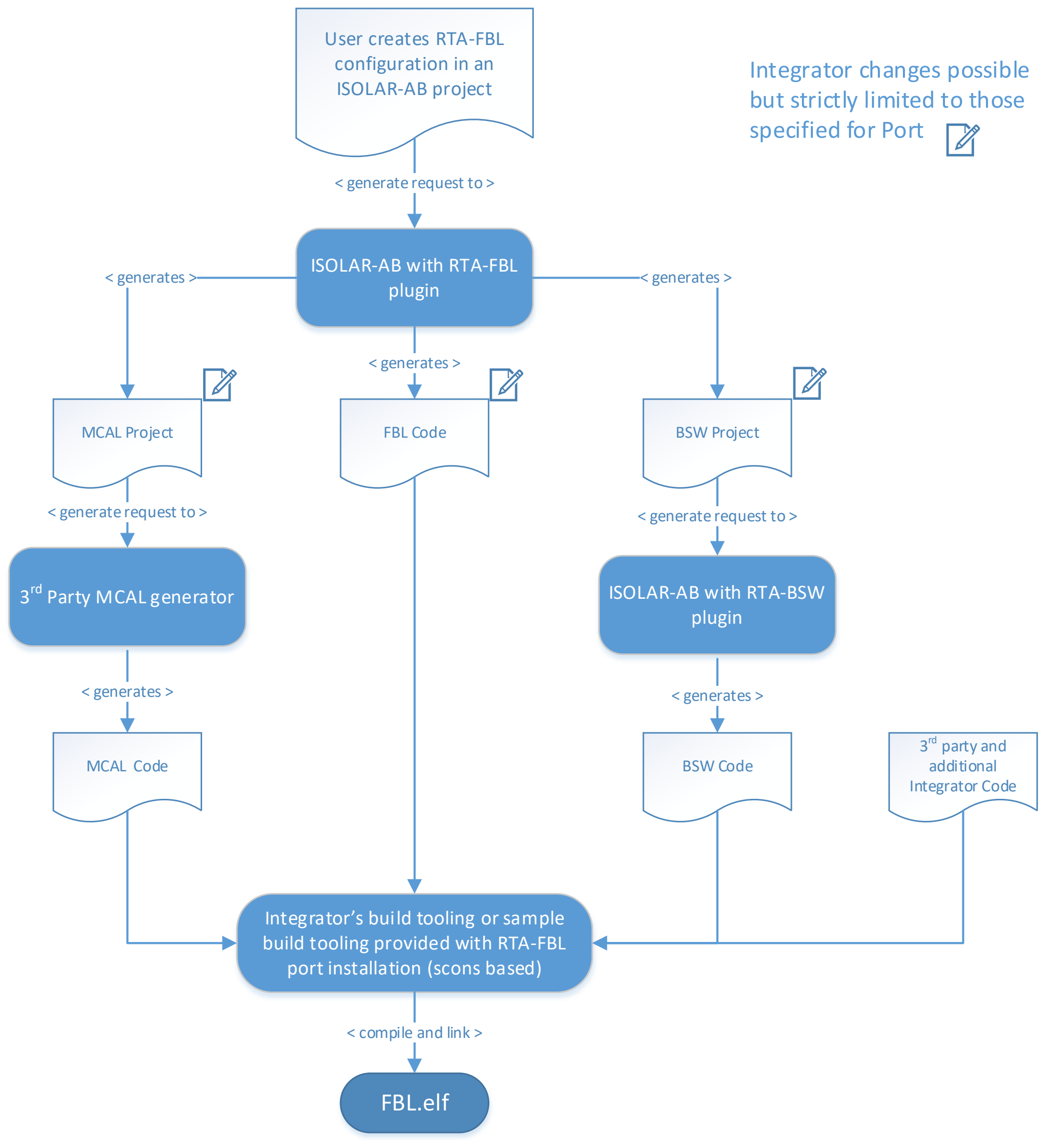

User creates RTA-FBL configuration in an ISOLAR-AB project
< generate request to >
Integrator changes possible but strictly limited to those specified for Port
ISOLAR-AB with RTA-FBL plugin
< generates >
< generates >
< generates >
MCAL Project
FBL Code
BSW Project
< generate request to >
< generate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

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

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

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

__IMPORTANT:__ RTA-FBL tests are carried out by ETAS for various FBL configurations that create for each configuration different bootloader code, an MCAL project and a BSW project. Since the integrator can make adaptations to specified sample code, the generated MCAL project and the generated BSW project, this may result in a final software stack that is not tested. For this reason, it is ultimately the integrator's responsibility to test that the complete bootloader works with any changes made to any code or projects generated by RTA-FBL. Please read the important integrator guidelines provided in Chapter 3 for information relevant to your port.

2.10 General architecture of RTA-FBL

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

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

As noted in Section 2.9, you will need to install a number of tools in order to generate a complete instance of RTA-FBL with all required modules as shown in Figure 4. A number of integration steps will also be required to build your software. Details for your specific OEM port and target are also given in Chapter 3, including the folder structure of a generated RTA-FBL instance that contains the code for the modules in Figure 4.

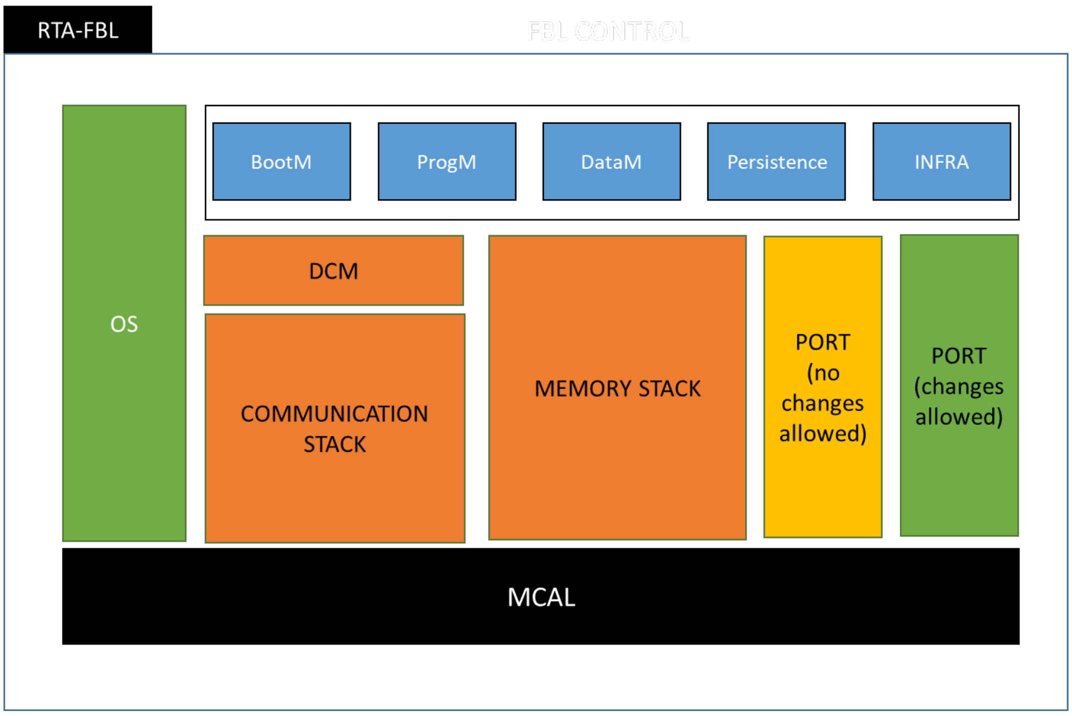


Figure 4: General architecture of an RTA-FBL instance

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

In order to generate an instance of RTA-FBL, you will need to install the tools shown in Table 1. Once you have the above packages, you will be able to generate an instance of RTA-FBL. In order to build the instance, you will also need to have installed the 3rd party MCAL as well as the relevant compiler toolchain required by your target as described in your GM FBL Target Guide.

Table 1: Tool versions

Tool Name	Version	Description
ISOLAR-AB	9.1.0	RTA-FBL configurator tool.
RTA-FBL GM Port	1.3.3	FBL generator plugin for ISOLAR-AB.
.NET framework	3.5	This is required by the ETAS license management. In most cases, you will already have this installed on your machine.

3 RTA-FBL GM Port

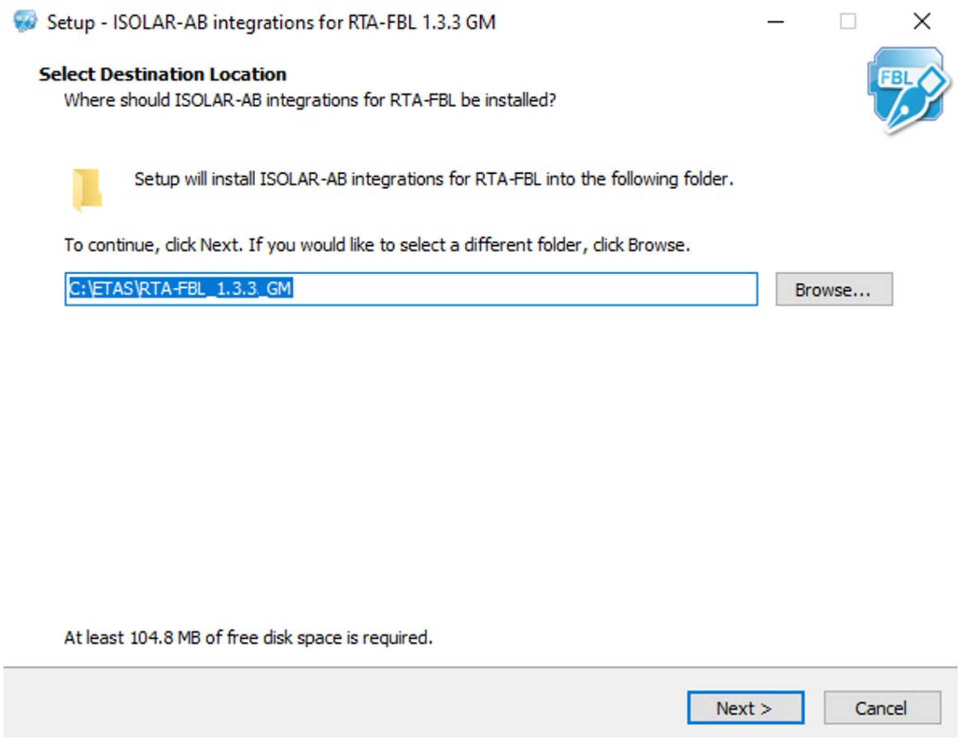
This chapter describes the GM Port of RTA-FBL. It provides specific information relevant to this port that expands on the general RTA-FBL features described in 2. This chapter assumes that the reader is familiar with the GM Specifications in [1], [2] and [3], and all relevant referenced specifications therein. Reference is made to these documents when describing the configuration and implementation-specific features of the GM Port of RTA-FBL.

3.1 Installation

This section describes the installer for the GM port of RTA-FBL. As noted in Section 2.11, you need to install this package in addition to ISOLAR-AB and RTA-BSW. This installer is described further in this section.

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

Step 1: Execute the file RTA_FBL_1.3.3_GM.exe. When the welcome window is displayed, select the desired installation folder by typing the desired location or by clicking “Browse”. Then click “Next”.


Setup - ISOLAR-AB integrations for RTA-FBL 1.3.3 GM

Select Destination Location

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

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

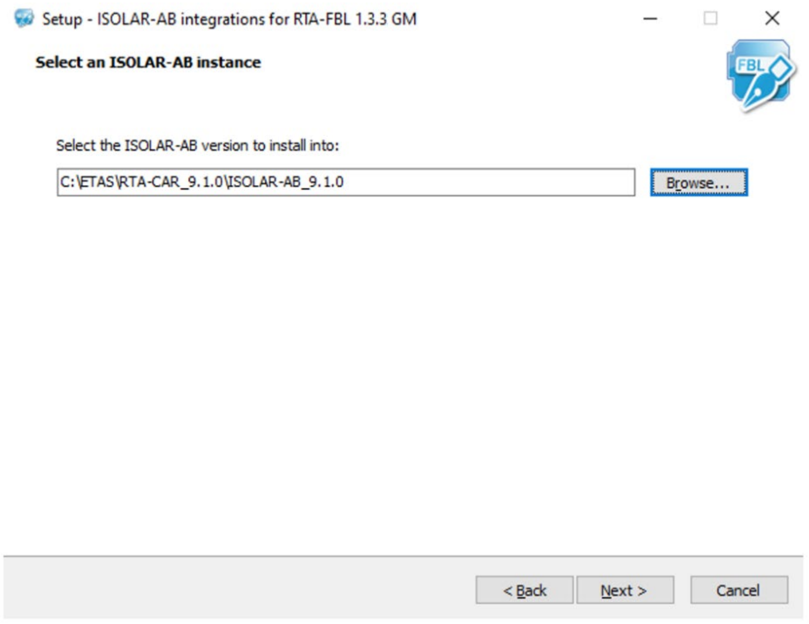
To continue, click Next. If you would like to select a different folder, click Browse.

C:\ETAS\RTA-FBL_1.3.3_GM

At least 104.8 MB of free disk space is required.

Figure 5: Welcome window

Step 2: Select the ISOLAR-AB version that will support the plugin by using "Browse". The minimum required version is 9.1.0. Then click "Next".


Setup - ISOLAR-AB integrations for RTA-FBL 1.3.3 GM

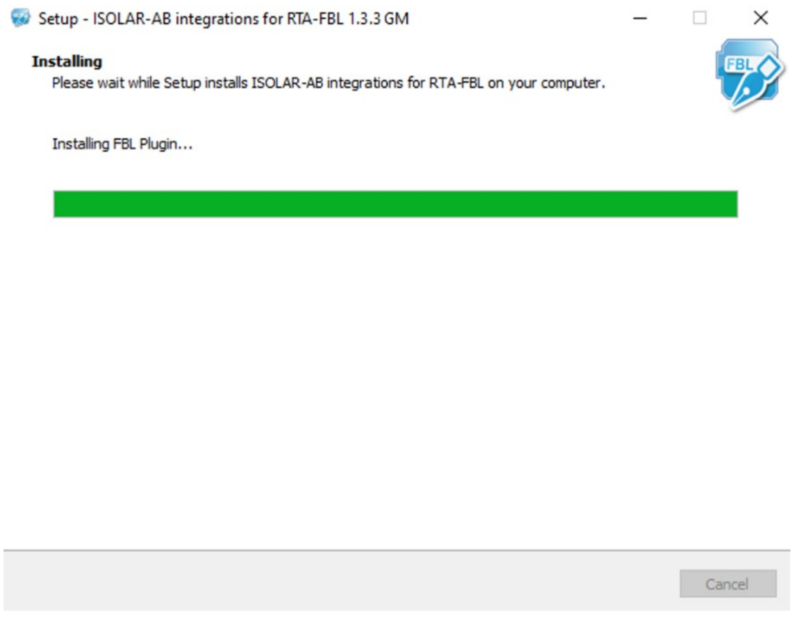
Select an ISOLAR-AB instance

Select the ISOLAR-AB version to install into:

C:\ETAS\RTA-CAR_9.1.0\ISOLAR-AB_9.1.0

Figure 6: ISOLAR-AB version selection

Step 3: Wait until the installation is complete.


Setup - ISOLAR-AB integrations for RTA-FBL 1.3.3 GM

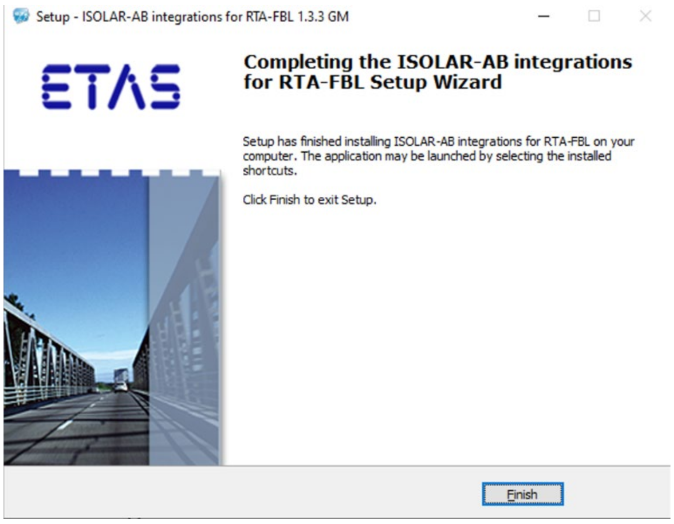
Installing

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

Installing FBL Plugin...

Figure 7: Installation progress

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



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

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

Click Finish to exit Setup.

Figure 8: Installed Components

3.2 Licensing

To be able to work with an ETAS software product, you require a license. This section contains basic details on ETAS License. Details concerning the scope of the licenses and other legal aspects can be found in “Terms and Conditions”.

3.2.1 ETAS License Models

There are two different license models available for licensing RTA-FBL software:

Machine-Named License, Local

- A license of this type is managed by the user himself/herself.
- As it is linked to a particular PC (more precisely: to the MAC address of the Ethernet adapter), it is valid wherever the PC is used.
- When you change your PC, you require a new license

Concurrent (or Floating) License, Server-Based

- The licenses are provided for the specific user names. Several users share a limited number of licenses.

How to get a License

Contact the responsible person, if your company has a tool coordinator and server-based license management for ETAS software. Otherwise (in case of a machine-named license) you obtain your license from the ETAS license-portal (the URL is shown on your Entitlement Certificate).

There are three ways of logging in on the welcome page:

Activation ID: Once you have logged in, a specific activation is visible and can be managed – the activation ID is shown on your Entitlement Certificate.

Entitlement ID: All activations of the entitlement are visible and can be managed (e.g. for a company with just one entitlement).

E-mail and password: All activations of the entitlements assigned to the user account are visi-ble and can be managed (e.g. for a tool coordinator responsible for several entitlements).

If you need help in the portal, click Help link.

What Information is required?

Information on the hosts must be entered to activate licenses:

Machine-named license: The MAC address of the Ethernet adapter to which the license is to be bound is required.

Concurrent (floating) license: You need a server host or a server triad.

Note: If this data changes (e.g. due to changes in the hardware or a change of user), the license must be given a “rehost”. This procedure is also described in the portal help file.

License File

The result of your activities is the provision of a file <name>.lic with which you can license your software in the ETAS License Manager.

Open ETAS License Manager:

- Click All Programs®->ETAS®License Management®->ETAS License Manager on Windows 7 Start menu.

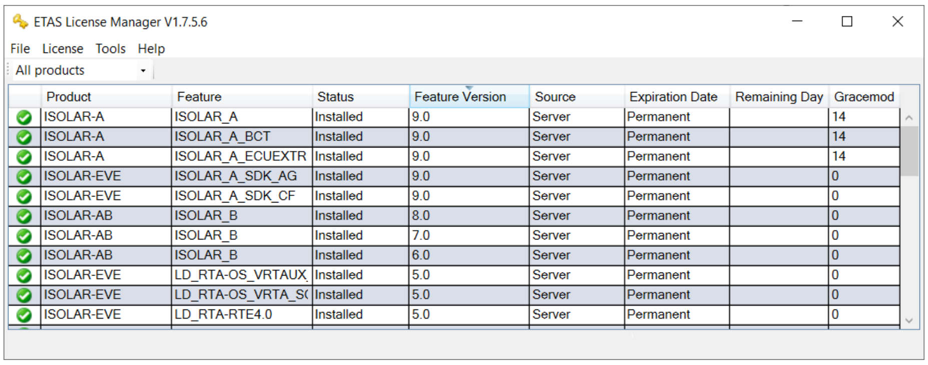
  <or>

- Click ETAS License Manager directly on “Apps” view in the Windows 8 / Windows 8.1 / Windows 10 Start menu.

Check License Status:

- Open the ETAS License Manager.
- Check the “Status” column.

The ETAS License Manager contains one entry for each installed product. The symbol at the beginning of the entry and the “Status” column entry indicates if a valid license has already been obtained or not.

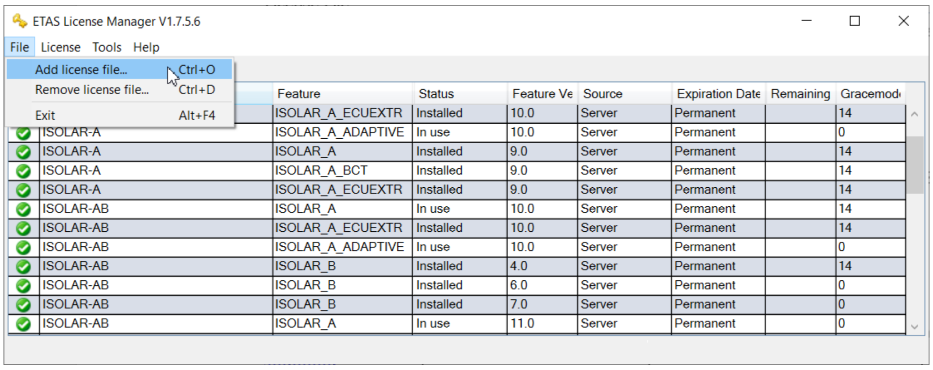


	Product	Feature	Status	Feature Version	Source	Expiration Date	Remaining Day	Gracemod
✓	ISOLAR-A	ISOLAR_A	Installed	9.0	Server	Permanent		14
✓	ISOLAR-A	ISOLAR_A_BCT	Installed	9.0	Server	Permanent		14
✓	ISOLAR-A	ISOLAR_A_ECUEXTR	Installed	9.0	Server	Permanent		14
✓	ISOLAR-EVE	ISOLAR_A_SDK_AG	Installed	9.0	Server	Permanent		0
✓	ISOLAR-EVE	ISOLAR_A_SDK_CF	Installed	9.0	Server	Permanent		0
✓	ISOLAR-AB	ISOLAR_B	Installed	8.0	Server	Permanent		0
✓	ISOLAR-AB	ISOLAR_B	Installed	7.0	Server	Permanent		0
✓	ISOLAR-AB	ISOLAR_B	Installed	6.0	Server	Permanent		0
✓	ISOLAR-EVE	LD_RTA-OS_VRTAUX	Installed	5.0	Server	Permanent		0
✓	ISOLAR-EVE	LD_RTA-OS_VRTA_S(	Installed	5.0	Server	Permanent		0
✓	ISOLAR-EVE	LD_RTA-RTE4.0	Installed	5.0	Server	Permanent		0

Figure 9. License Manager (Installed Licenses)

To add a License file:

- Open the ETAS License Manager.
- Select the menu, **File->Add License File.**


ETAS License Manager V1.7.5.6
File License Tools Help
Add license file... Ctrl+O
Remove license file... Ctrl+D
Exit Alt+F4

		Feature	Status	Feature Ve	Source	Expiration Date	Remaining	Gracemod
		ISOLAR_A_ECUEXTR	Installed	10.0	Server	Permanent		14
✓	ISOLAR-A	ISOLAR_A_ADAPTIVE	In use	10.0	Server	Permanent		0
✓	ISOLAR-A	ISOLAR_A	Installed	9.0	Server	Permanent		14
✓	ISOLAR-A	ISOLAR_A_BCT	Installed	9.0	Server	Permanent		14
✓	ISOLAR-A	ISOLAR_A_ECUEXTR	Installed	9.0	Server	Permanent		14
✓	ISOLAR-AB	ISOLAR_A	In use	10.0	Server	Permanent		14
✓	ISOLAR-AB	ISOLAR_A_ECUEXTR	Installed	10.0	Server	Permanent		14
✓	ISOLAR-AB	ISOLAR_A_ADAPTIVE	In use	10.0	Server	Permanent		0
✓	ISOLAR-AB	ISOLAR_B	Installed	4.0	Server	Permanent		14
✓	ISOLAR-AB	ISOLAR_B	Installed	6.0	Server	Permanent		0
✓	ISOLAR-AB	ISOLAR_B	Installed	7.0	Server	Permanent		0
✓	ISOLAR-AB	ISOLAR_A	In use	11.0	Server	Permanent		0

Figure 10. Add a License

The “Add License File” window opens.

- Next to the “Select License File” field, click the [. . .] button to select the .lic file.
- Click OK.

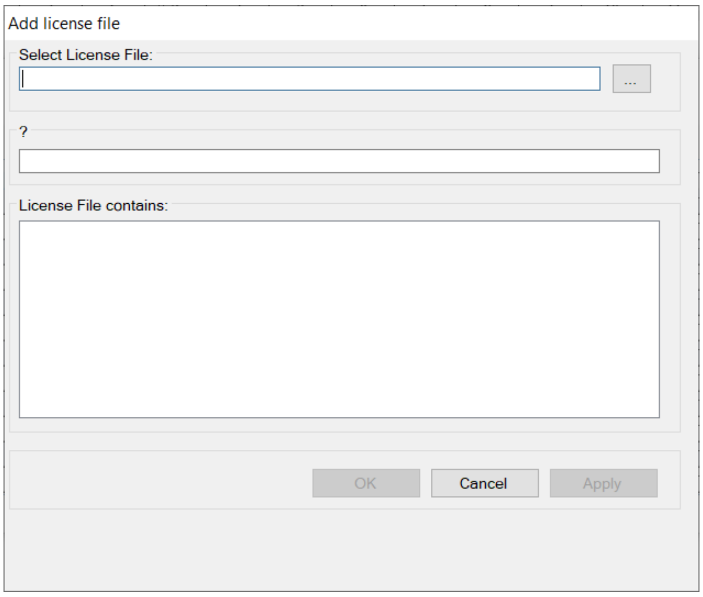


Figure 11. Add a License File.

The “ETAS License Manager” window shows information on the selected license. The “Feature Version” column shows the version number of the license, not the version number of the software.

Warning: If the green symbol is not displayed, there might be a problem with the license file or the license relates to another product. Additional information on ETAS License Manager can be found in the “Online Help” of the ETAS License Manager.

3.3 Supported and unsupported features

This version targets supports all the features described in [3] with the following exceptions noted:

- Bit-difference library: This optional feature is not supported.
- LZMA compression: This feature is not supported. Note that GM do not require this feature to be supported if the download time requirements are met. If downloads do not meet these requirements, consider using ARLE compression which is supported in this port.
- Multiple processors: Only one processor is supported in this version. Therefore, only one application and its association calibration modules will be accepted for download by the bootloader. This also limits the maximum number of calibration partitions to 14 and the maximum number of calibration modules to 19 (calibration module Ids 2 through to 20).
- The optional integrity check validation of the bootloader during initialization is not supported.
- ECU Regionalization: This feature is not currently supported.
- Bootloader Secure Parameters: This feature is not currently supported.
- Security Peripheral Update: While this feature is supported, a dedicated memory area for storing the downloaded update package is required. You cannot use an application or calibration space to store the SP update package.

The integrator should also note the following:

- Bootloader update is not required in [3]. Although bootloader update is not natively supported in the port, support may have been implemented for your chosen target. Please see your GM FBL Target Guide for additional information on whether bootloader update is supported for your target.
- The block size set in the configuration of the bootloader as described in Section 3.5.2 must be respected for all blocks sent from the tester except for the last block that can be smaller than the configured block size.
- As result of a limitation in the types of frames supported by RTA-BSW, functional addresses should only be used for single-frame communication when using CAN. All multi-frame communication should be done using physical Can Ids for CAN communication.

The following services and subservices as described in [2] and [3] are supported:

Service	Subfunction	Important config	Comments
0x10 – DiagnosticSessionControl	01	P2ServerMax is set to 0.1 seconds and P2StartServerMax is set to 5 seconds.	Default session
	02		Programming Session
	03		Extended Session
0x22 – ReadDataByIdentifier	N/A	All DiDs noted in [2] to be required by the bootloader are supported	Used to read data from the Fbl
0x27 – SecurityAccess	01,03,05		Note that in boot mode, requesting a seed automatically unlocks the ECU. The returned seed consist of 31 bytes all set to 0. No key needs to be sent, and the pairing Send Key subfunctions are not supported.
0x28 – CommunicationControl	00	N/A	enableRxAndTx
	03	N/A	disableRxAndTx

0x31 – Routine-Control	01	0xFF00 as identifier	Erase memory region
	01	0x0209 as identifier	Set PSI
	01	0xFF01 as identifier	Check programming de-pendencies
	01	0x0300 as identifier	Ephemeral Test Memory Erase. Not supported, and therefore returns 0x10 to indicate this.
	01 03	0x3C2 as identifier	SP Programming
0x34 – Re-questDownload	N/A	N/A	Request for download
0x36 – Transfer-Data	N/A	N/A	Transfer of blocks (both application and calibra-tion)
0x37 – Re-questTransferExit		N/A	Compete transfer
0x3E – Tester-Present	00	N/A	Tester present
0x85 – Con-trolDTCSettings	01	N/A	Set On: Configured, but DTC is not supported while in boot mode.
	02	N/A	Set Off: Configured, but DTC is not supported while in boot mode.

3.4 GM RTA-FBL Architecture

Figure 12 provides a high-level view of RTA-FBL architecture for GM. The communication, memory and diagnostic stacks are based on RTA-BSW and support the AUTOSAR architecture and methodology for source code configuration and generation. The rest of the components, except for the MCAL, are provided by ETAS. The modules that comprise the RTA-FBL instance for a GM 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 GM-specific port modules (in yellow); these are generated by the RTA-FBL ISOLAR-AB plugin when the GM port is selected and must not be modified by the integrator. The Port module implements the bootloader features that are specific to the GM specification [1][2][3] whereas the ECL is the ETAS Crypto library used for message digest and signature calculation.
4. The GM-specific sample modules (in green); these are generated by the RTA-FBL ISOLAR-AB plugin when the GM port is selected and may be modified by the integrator:
   - The OS is a basic cyclic scheduler that can be replaced by any other scheduler (e.g. a fully-configured RTA-OS) as long as the calls to the relevant main functions are made at the correct periods as in the provided samples. See Section 0 for further details on how to adapt this module.
   - The BLSM contains code for initializing the Bootloader. Changes can be made here by the integrator if other modules are to be integrated (e.g. other BSW modules) but changes should not be made to the functions that interacts with the core FBL modules. See Section 3.5.10 for further details on how to adapt this module.

5. The GM-specific third-party modules (in violet); these are not generated by the RTA-FBL ISOLAR-AB plugin and must be integrated separately by the integrator to add support for the Security Peripheral. See Section 3.5.17 for further details.
6. The MCAL modules and the Port-Target interface module (in black); the modules shown are those required by the GM 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 when populating the functions described in Section 3.5.7. The GM port ships with a dummy target that contains no MCAL. Please see your target user guide for a sample MCAL project that has been tested with the full bootloader stack.

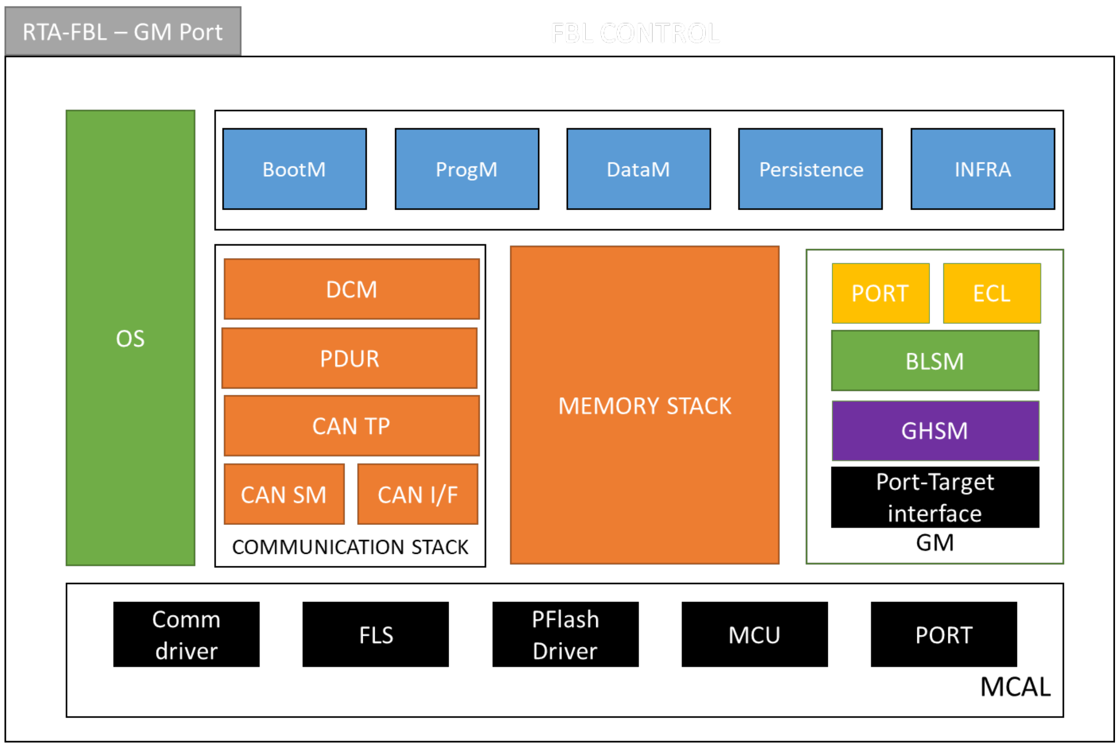


Figure 12: GM architecture of an RTA-FBL instance

3.5 Creating and building an RTA-FBL instance

This section explains how to create an ISOLAR-AB project to configure and generate an instance of RTA-FBL compliant with the GM bootloader specification in [3]. The tooling described in this section has been tested with Windows 10.

3.5.1 Project creation

A new FBL project is created in ISOLAR-AB. As shown in Figure 13, create a new RTA-CAR project by clicking the “New” dropdown button and selecting “RTA-CAR Project”.

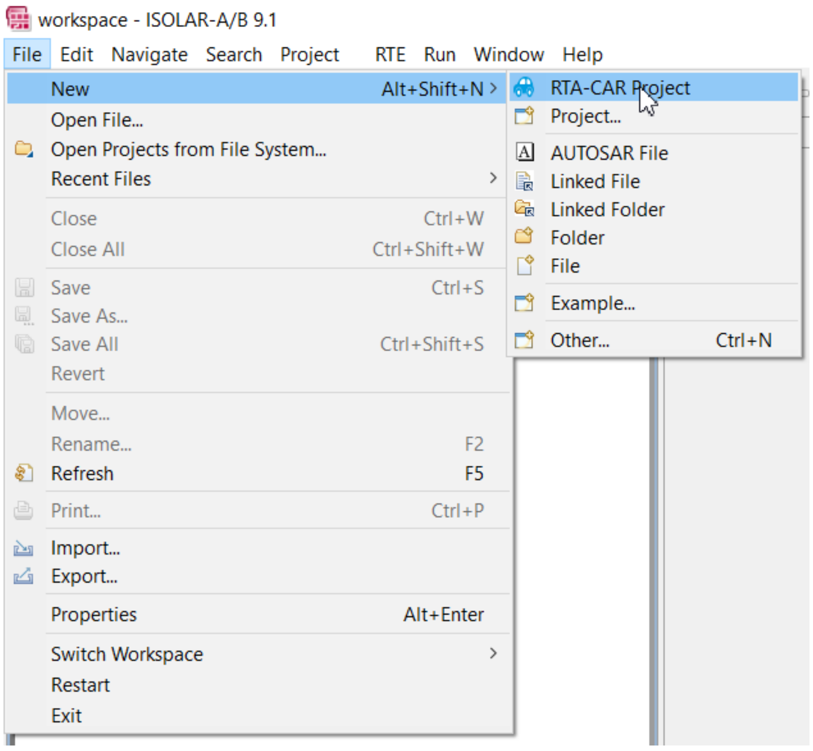


Figure 13: RTA-CAR project creation

If RTA-CAR Project is not present, select “Project” and search for “RTA-CAR Project” in the new window, as shown in Figure 14.

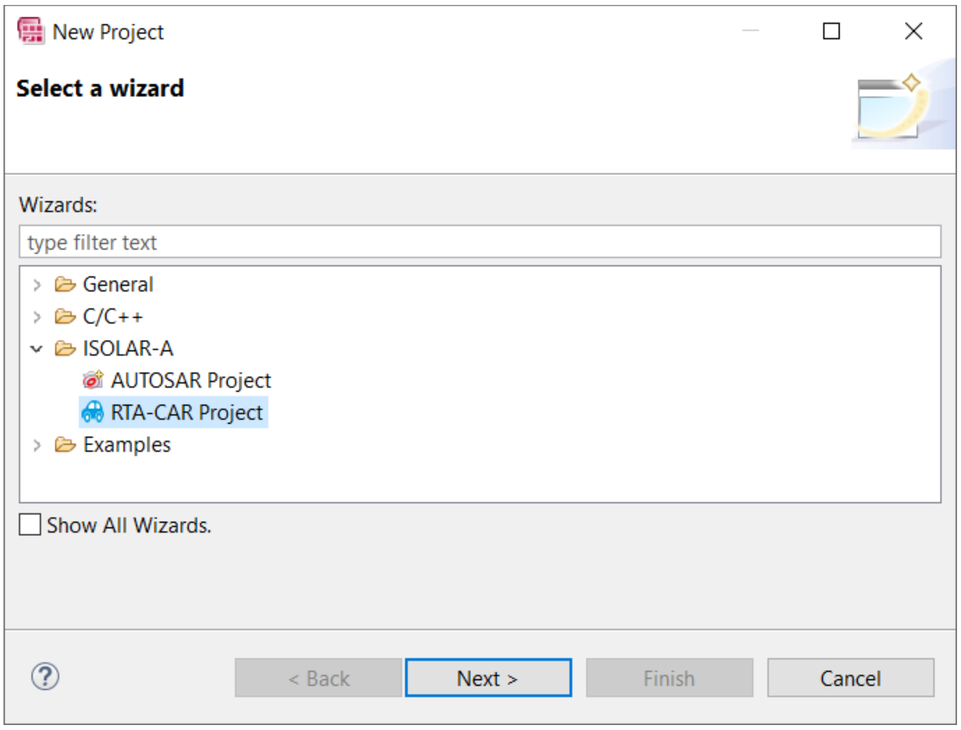
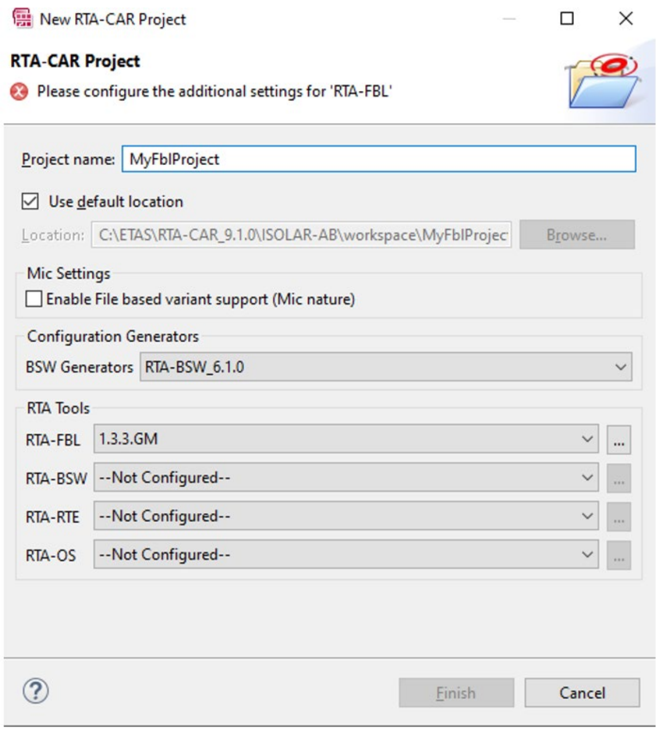


Figure 14: RTA-CAR project

In the New RTA-CAR Project window, choose a name for your project and select the 1.3.3.GM plugin under RTA Tools as shown in Figure 15. Note that the other RTA-Tools are not to be configured. The BSW Generator can be set to any BSW generator you have available. This BSW generator will not be used during BSW generation as RTA-FBL uses its own internal BSW generator.


New RTA-CAR Project

RTA-CAR Project

Please configure the additional settings for 'RTA-FBL'

Project name: MyFblProject

☑ Use default location

Location: C:\ETAS\RTA-CAR_9.1.0\ISOLAR-AB\workspace\MyFblProjec

Mic Settings

☐ Enable File based variant support (Mic nature)

Configuration Generators

BSW Generators: RTA-BSW_6.1.0

RTA Tools

RTA-FBL: 1.3.3.GM

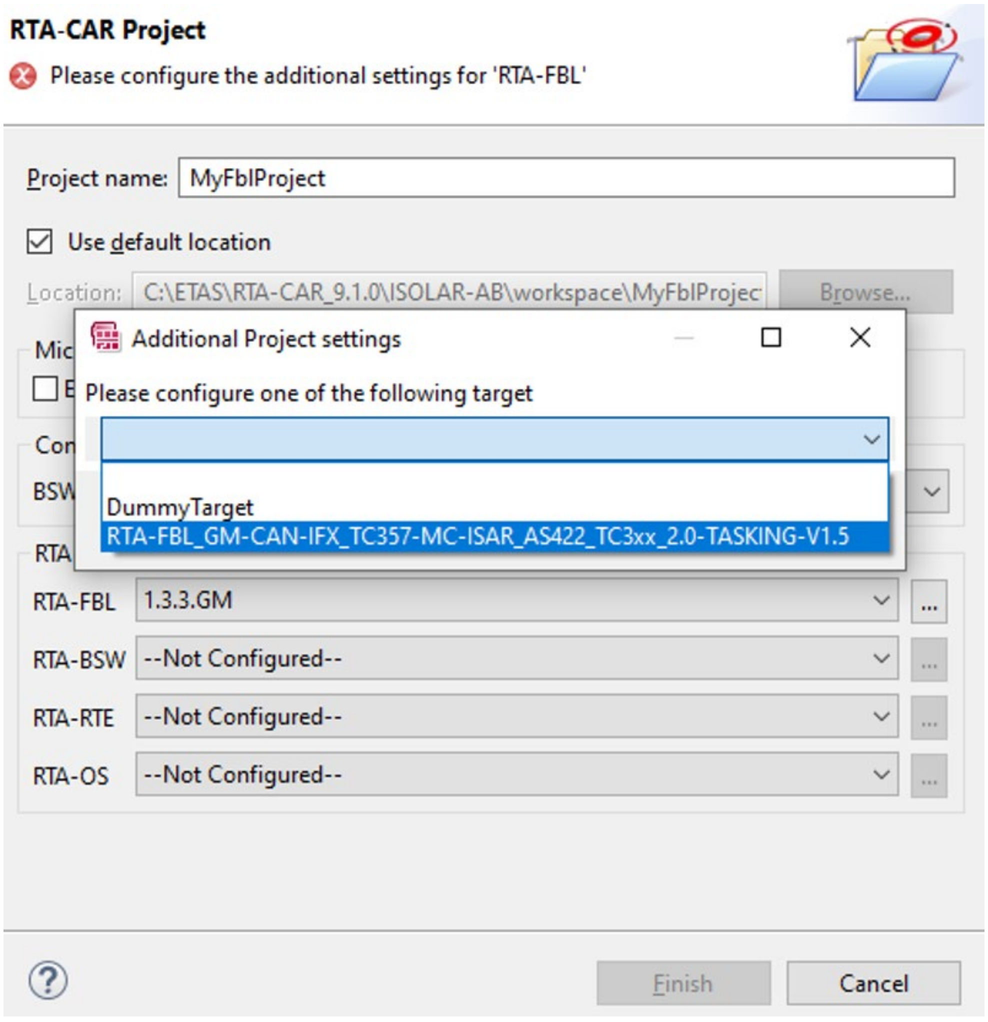
RTA-BSW: --Not Configured--

RTA-RTE: --Not Configured--

RTA-OS: --Not Configured--

Figure 15: New RTA-FBL Project

Next, click on the three dots icon to open the window “Additional Project Settings” and select the target from the dropdown list as shown in Figure 16.



RTA-CAR Project

Please configure the additional settings for 'RTA-FBL'

Project name: MyFblProject

☑ Use default location

Location: C:\ETAS\RTA-CAR_9.1.0\ISOLAR-AB\workspace\MyFblProjec

Additional Project settings

Please configure one of the following target

DummyTarget

RTA-FBL_GM-CAN-IFX_TC357-MC-ISAR_AS422_TC3xx_2.0-TASKING-V1.5

RTA-FBL: 1.3.3.GM

RTA-BSW: --Not Configured--

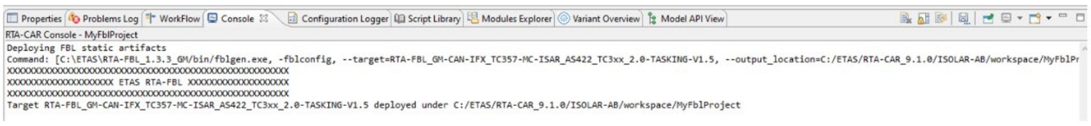
RTA-RTE: --Not Configured--

RTA-OS: --Not Configured--

Figure 16: Select Target

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

Figure 17 shows the result of a successful project creation in the console window.



```
RTA-CAR Console - MyFblProject
Deploying FBL static artifacts
Command: [C:\ETAS\RTA-FBL_1.3.3_GM/bin/fblgen.exe, -fblconfig, --target=RTA-FBL_GM-CAN-IFX_TC357-MC-ISAR_AS422_TC3xx_2.0-TASKING-V1.5, --output_location=C:/ETAS/RTA-CAR_9.1.0/ISOLAR-AB/workspace/MyFblPr
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXX ETAS RTA-FBL XXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Target RTA-FBL_GM-CAN-IFX_TC357-MC-ISAR_AS422_TC3xx_2.0-TASKING-V1.5 deployed under C:/ETAS/RTA-CAR_9.1.0/ISOLAR-AB/workspace/MyFblProject
```

Figure 17: Console window upon successful project creation

3.5.2 Configuration and Generation of FBL and BSW

Next, complete the FBL configuration parameters. In the AR Explorer view, double click on one of the items under Bsw > Bsw Module Description > FBL, as shown in Figure 18.

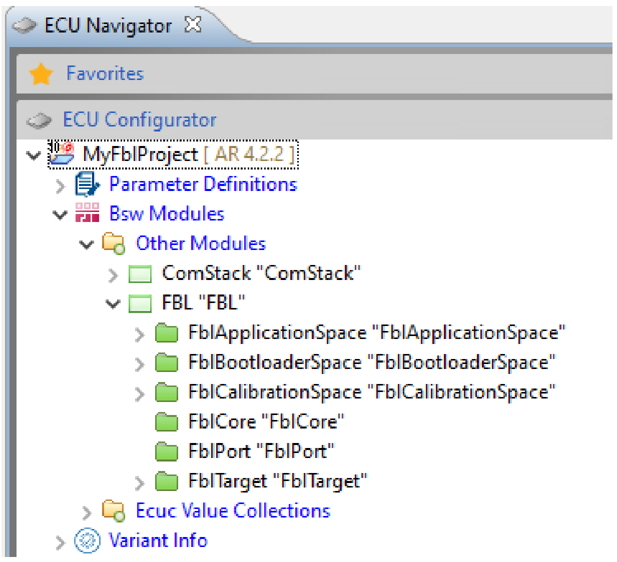
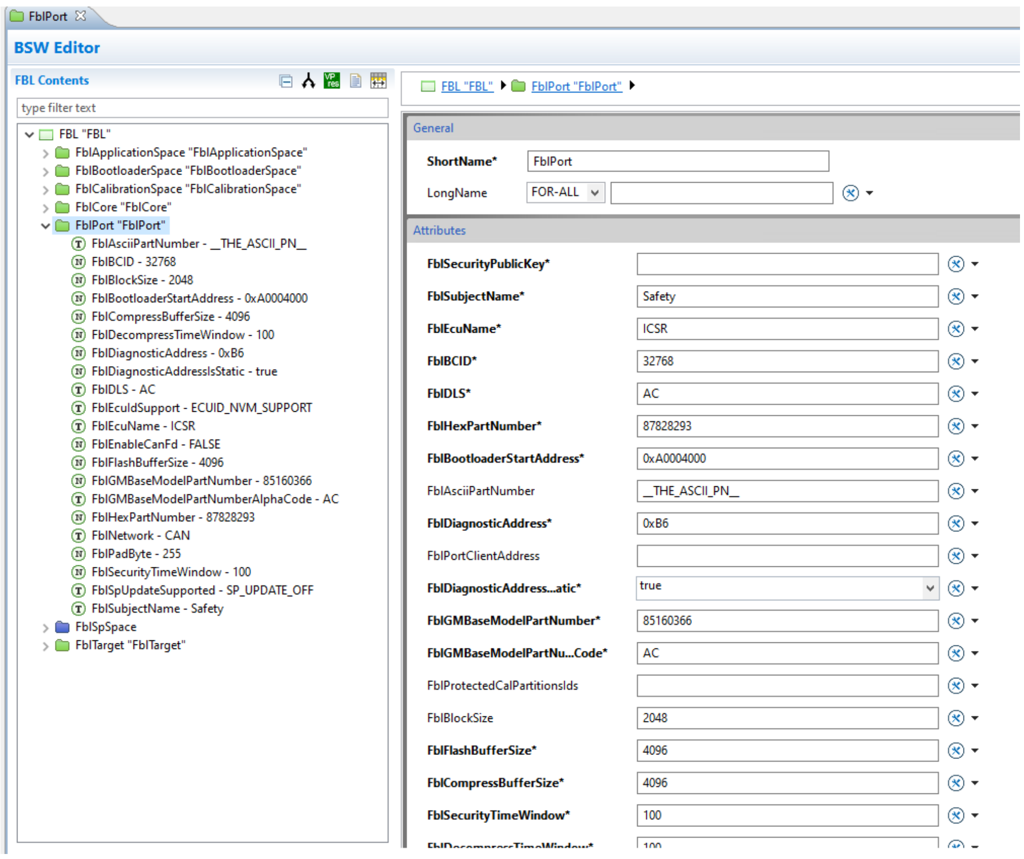


Figure 18: Accessing the FBL configuration parameters

The user can now edit the base configuration parameters in the RTA-FBL Editor window. Figure 19 shows an example of the port-specific configuration parameters. An explanation of each parameter is provided at the end of this section.



- FblAsciiPartNumber - __THE_ASCII_PN__
- FblBCID - 32768
- FblBlockSize - 2048
- FblBootloaderStartAddress - 0xA0004000
- FblCompressBufferSize - 4096
- FblDecompressTimeWindow - 100
- FblDiagnosticAddress - 0xB6
- FblDiagnosticAddressIsStatic - true
- FblDLS - AC
- FblEcuIdSupport - ECUID_NVM_SUPPORT
- FblEcuName - ICSR
- FblEnableCanFd - FALSE
- FblFlashBufferSize - 4096
- FblGMBaseModelPartNumber - 85160366
- FblGMBaseModelPartNumberAlphaCode - AC
- FblHexPartNumber - 87828293
- FblNetwork - CAN
- FblPadByte - 255
- FblSecurityTimeWindow - 100
- FblSpUpdateSupported - SP_UPDATE_OFF
- FblSubjectName - Safety

ShortName*	FblPort
LongName	FOR-ALL
FblSecurityPublicKey*	
FblSubjectName*	Safety
FblEcuName*	ICSR
FblBCID*	32768
FblDLS*	AC
FblHexPartNumber*	87828293
FblBootloaderStartAddress*	0xA0004000
FblAsciiPartNumber	__THE_ASCII_PN__
FblDiagnosticAddress*	0xB6
FblPortClientAddress	
FblDiagnosticAddress...atic*	true
FblGMBaseModelPartNumber*	85160366
FblGMBaseModelPartNu...Code*	AC
FblProtectedCalPartitionsIds	
FblBlockSize	2048
FblFlashBufferSize*	4096
FblCompressBufferSize*	4096
FblSecurityTimeWindow*	100

Figure 19: Edit Configuration Parameters

Once complete, the user can generate the RTA-FBL instance first by clicking on “Open RTA Code Generator dialog…” as shown in Figure 20 and then, in the opened RTA Code Generator window, by clicking Run as shown in Figure 21.

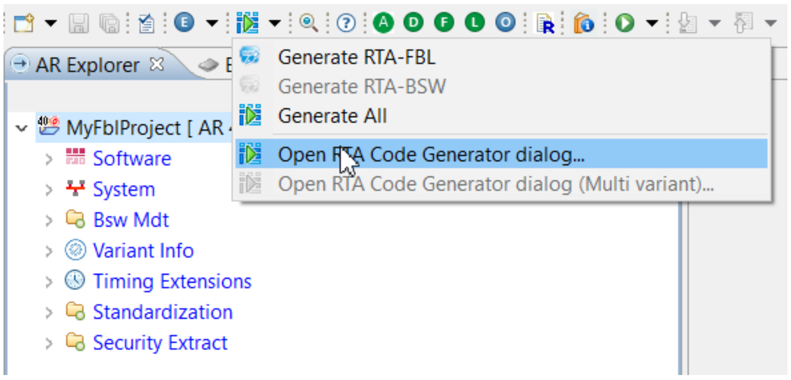
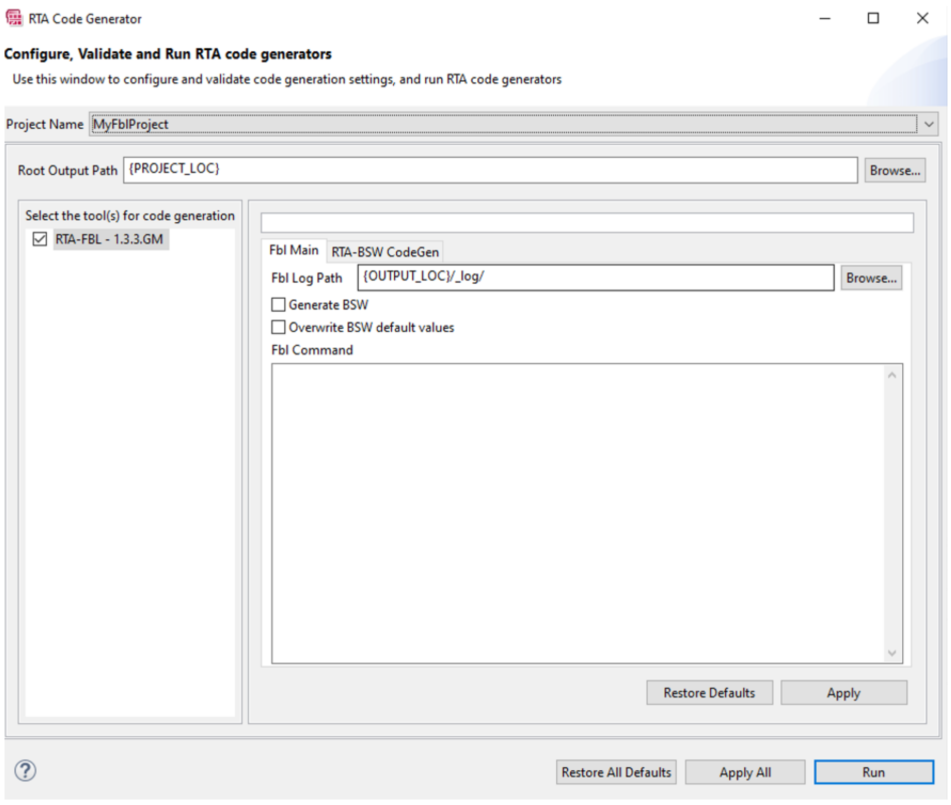


Figure 20: Open RTA Code Generator Dialog



RTA Code Generator

Configure, Validate and Run RTA code generators

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

Project Name: MyFblProject

Root Output Path: {PROJECT_LOC}

Select the tool(s) for code generation: ☑ RTA-FBL - 1.3.3.GM

Fbl Main | RTA-BSW CodeGen

Fbl Log Path: {OUTPUT_LOC}/_log/

☐ Generate BSW

☐ Overwrite BSW default values

Fbl Command

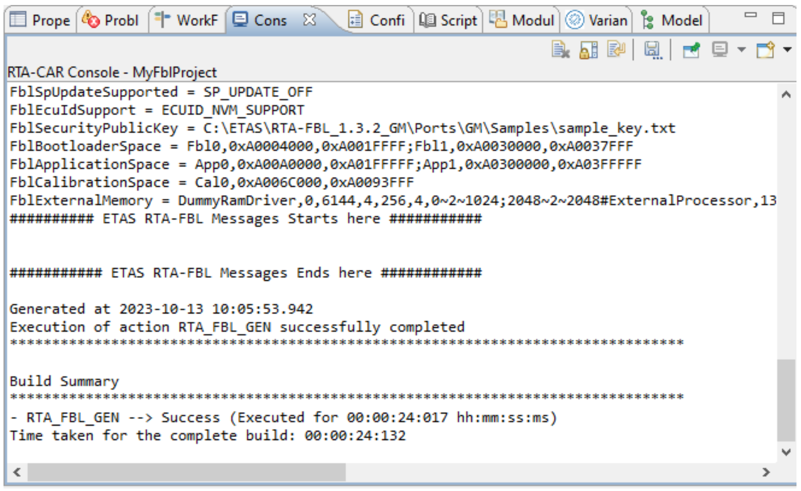
Figure 21: RTA Code Generator

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).

- **<u>IMPORTANT</u>**: 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 selected. The configuration in this file should never be modified as these values are completely defined by the configuration of the bootloader.

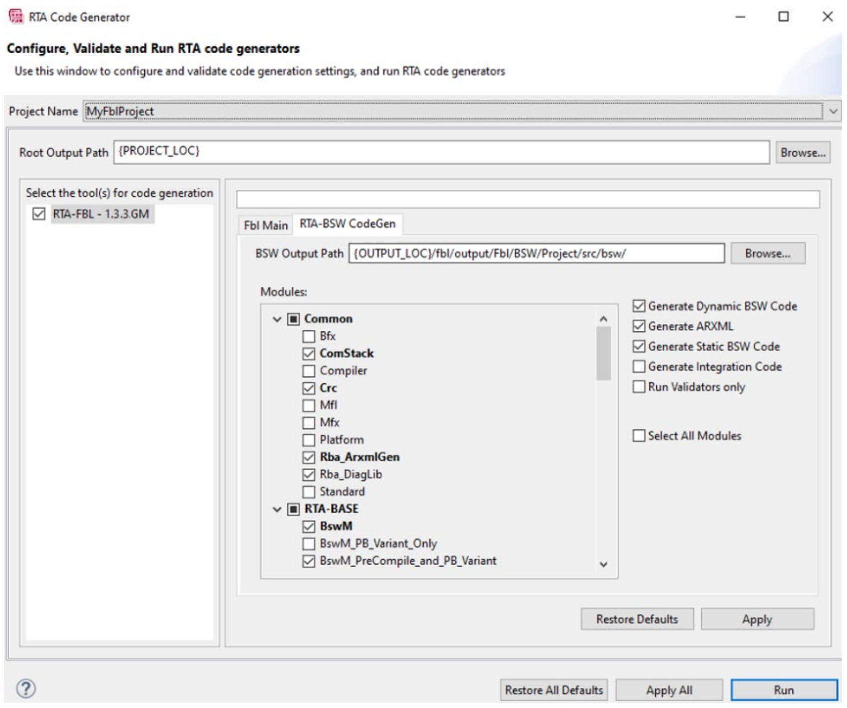
On clicking Run, the RTA-FBL instance is generated. Figure 22 shows the result of a successful generation in the console window.



```
FblSpUpdateSupported = SP_UPDATE_OFF
FblEcuIdSupport = ECUID_NVM_SUPPORT
FblSecurityPublicKey = C:\ETAS\RTA-FBL_1.3.2_GM\Ports\GM\Samples\sample_key.txt
FblBootloaderSpace = Fbl0,0xA0004000,0xA001FFFF;Fbl1,0xA0030000,0xA0037FFF
FblApplicationSpace = App0,0xA00A0000,0xA01FFFFF;App1,0xA0300000,0xA03FFFFF
FblCalibrationSpace = Cal0,0xA006C000,0xA0093FFF
FblExternalMemory = DummyRamDriver,0,6144,4,256,4,0~2~1024;2048~2~2048#ExternalProcessor,13
########## ETAS RTA-FBL Messages Starts here ###########


########### ETAS RTA-FBL Messages Ends here ############

Generated at 2023-10-13 10:05:53.942
Execution of action RTA_FBL_GEN successfully completed
********************************************************************************

Build Summary
********************************************************************************
- RTA_FBL_GEN --> Success (Executed for 00:00:24:017 hh:mm:ss:ms)
Time taken for the complete build: 00:00:24:132
```

Figure 22: Console Window on Successful Generation

To complete the FBL instance, the user must generate the BSW code by selecting the BSW modules for which the code should be generated in the RTA-BSW CodeGen tab of the RTA Code Generator window, as shown in Figure 23.



RTA Code Generator

Configure, Validate and Run RTA code generators

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

Project Name: MyFblProject

Root Output Path: {PROJECT_LOC}

Select the tool(s) for code generation
- ✓ RTA-FBL - 1.3.3.GM

Fbl Main | RTA-BSW CodeGen

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

Modules:
- ■ Common
  - ☐ Bfx
  - ✓ ComStack
  - ☐ Compiler
  - ✓ Crc
  - ☐ Mfl
  - ☐ Mfx
  - ☐ Platform
  - ✓ Rba_ArxmlGen
  - ✓ Rba_DiagLib
  - ☐ Standard
- ■ RTA-BASE
  - ✓ BswM
  - ☐ BswM_PB_Variant_Only
  - ✓ BswM_PreCompile_and_PB_Variant

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

Figure 23: 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 24. Upon successful generation, the popup message in Figure 25 is shown.

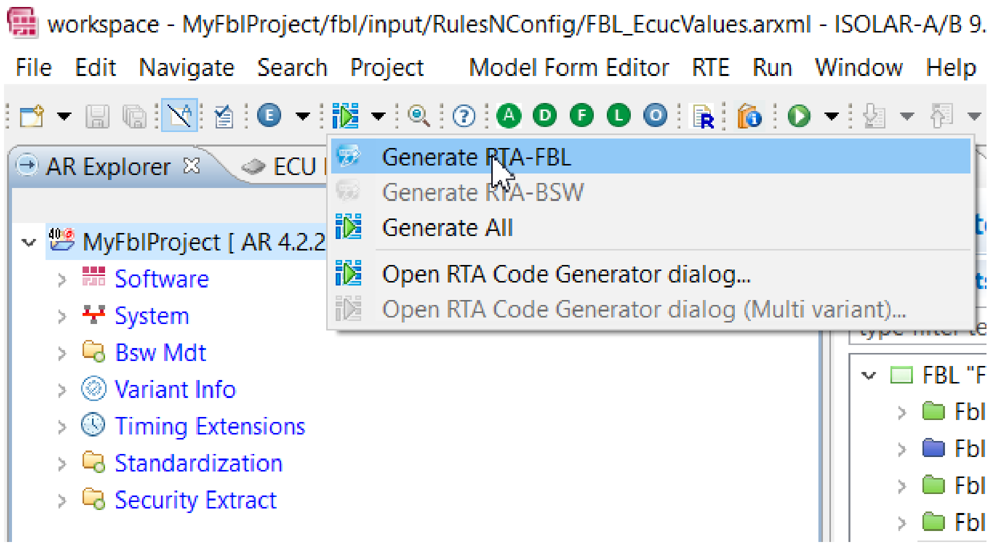
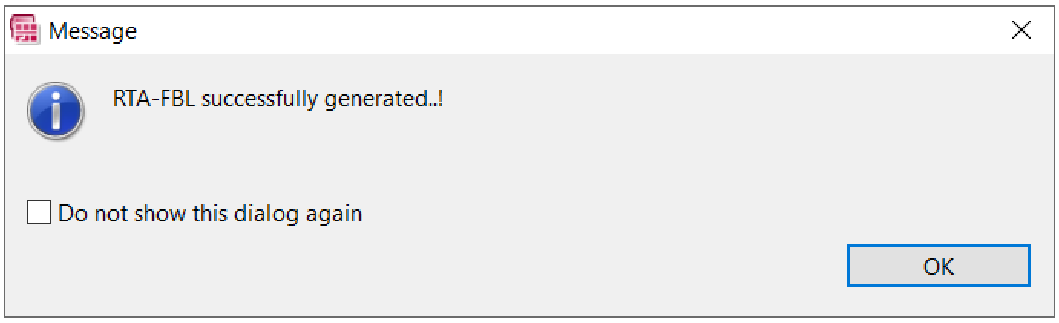


Figure 24: Generate RTA-FBL


Message

RTA-FBL successfully generated..!

Figure 25: Successful generation

Table 2 describes the parameters that the user can configure. The letters 'N', 'Y' 'M' and 'O' are used to indicate "No", "Yes", "Mandatory" and "Optional" respectively. The column "Requires BSW regen." indicates whether the BSW needs to be re-generated in case the associated parameter has been changed. Note that the Application, Calibration and Bootloader spaces each contain sub containers with each sub container containing a region with a high and low address. As required by [3], there must be at least one application space configured. The allowed range for each region in a space that can be specified is different for each target and can be found in your GM FBL Target Guide.

Note that your target may also specify parameters that are unique to that target, or restrict allowed configuration ranges for some parameters. These will also be listed in your GM FBL Target Guide.

Table 2: Configuration parameters of the GM port of RTA-FBL

Config Group	Parameter	Description	Requires BSW regen.	Optional or Mandatory
Fbl {Application, Calibration, Bootloader, Sp} Space	Fbl<space_name>RegionAddressHigh	These parameters of a region in the Application, Calibration, Bootloader and SP spaces define the memory range of that region. Note that for the SP space, only one region can be configured. The BootloaderSpace parameters are also used to configure the boot updater if this is supported for your target. The generated boot updater application will be set up to erase and program the memory regions defined by the BootloaderSpace parameter.	N	An Application Space and Bootloader Space is required. All other spaces are optional.
	Fbl<space_name>RegionAddressLow			
FblPort	FblSecurityPublicKey	The Security Public Key stored in the Boot Info Block. Select a text file that contains this key in hex. This file should contain 512 characters that define the 256-byte hex value of the key. A sample key is provided for you in the file /Ports/GM/Samples/sample_key.txt of your bootloader installation.	N	M
	FblSubjectName	The ECU Subject Name stored in the Boot Info Block. Enter 16 ASCII characters that define this parameter	N	M
	FblEcuName	The ECU Name stored in the Boot Info Block. Enter 8 ASCII characters that define this parameter.	N	M
	FblBCID	The BCID stored in the Boot Info Block. Enter an integer value between 0 and 0xFFFF.	N	M
	FblDLS	The DLS stored in the Boot Info Block. Enter 2 ASCII characters that define this parameter.	N	M

	FblHexPartNumber	The Hex Part Number stored in the Boot Info Block. Enter an integer value between 0 and 0xFFFFFFFF.	N	M
	FblPortClientAddress	The default client address. Enter an integer value between 0xF1 and 0xF6.	N	O
	FblAsciiPartNumber	The ASCII Part Number stored in the Boot Info Block. Enter 16 ASCII characters that de-fine this parameter. Sets the FBL generator parameter AsciiPartNumber.	N	O
	FblDiagnosticAddress	The diagnostic address of this ECU. Enter an integer value between 0x1 and 0xFF. If Diag-nosticAddressIsStatic is set, then this is the diagnostic address of the ECU. If it is not set, then the diagnostic address is read from NvM and this value is the default ROM value in case the read from NvM fails.	Y(****)	M
	FblDiagnosticAddressIsStatic	Specifies if the Diagnostic Address is static or if it needs to be read from NvM. Can be set to either “true” or “false”.	Y(****)	M
	FblGMBaseModelPartNumber	The GM Base Model Part Number required for DiD 0xF1CC. Enter an integer value be-tween 0 and 0xFFFFFFFF.	N	M
	FblGMBaseModelPartNumberAlpha-Code	The GM Base Model Part Number Alpha Code required for DiD 0xF1DC. Enter 2 ASCII char-acters that define this parameter.	N	M
	FblProtectedCalPartitionsIds	The calibration partitions which are pro-tected. Enter a comma-separated list of Ids with the smallest allowed Id being 2. Leave empty if no calibration partitions are pro-tected.	N	O
	FblBlockSize	The download block size in bytes. Enter an in-teger value between 16 and 4096 that is a	N	O

		multiple 16. If no value is entered, then the default value of 256 is used.		
	FblFlashBufferSize	The size of the flash buffer in bytes. Enter an integer value between 1024 and 6144 that is a multiple of the target write alignment and greater than the block size and any header.	N	M
	FblCompressBufferSize	The size of the compress buffer in bytes. Enter an integer value between 16 and 6144 that is a multiple 16 and greater than the block size.	N	M
	FblSecurityTimeWindow	The maximum amount of security processing in microseconds that can be done before yielding control back to the OS. Enter an integer value between 1 and 1000000.	N	M
	FblMessageDigestBlockSize	The maximum number of bytes that can be processed during message digest validation before yielding control back to the OS. If not specified, the default is set to 1024 bytes.	N	O
	FblDecompressTimeWindow	The maximum amount of decompression processing in microseconds that can be done before yielding control back to the OS. Enter an integer value between 1 and 1000000.	N	M
	FblPadByte	The byte to be used for padding. Enter an integer value between 0 and 0xFF. Leave empty if padding is not required.	N	O
	FblPersistence	Specifies whether the RTA-CAR NvM stack is to be generated (set parameter to NVM_RTA_CAR), or whether hook functions to be defined by the user to handle persistence of non-volatile data should be generated (set parameter to NVM_USER).	Y	M
	FblNetwork	The underlying can network (CAN or Ethernet). Note that this value depends on the	Y	M

		communication channel supported by the target.		
	FblEthPhysicalAddress	The physical MAC address of the Ethernet ECU. Enter a value in the format FF:FF:FF:FF:FF:FF.	Y	M(*)
	FblEthRemoteIpAddress	The remote IP address of the device connect-ing to this ECU. Enter a value in the format 111:111:111:111.	Y	M(*)
	FblEthRemotePort	The remote IP port of the Ethernet ECU. En-ter a value between 1 and 65535.	Y	M(*)
	FblEthStaticIp	The local, static IP address of the Ethernet ECU. Enter a value in the format 111:111:111:111.	Y	M(*)
	FblEthNetMask	The network mask of the Ethernet ECU in CIDR Notation. Set a value between 0 and 32 that describes the number of significant bits defining the network number or prefix of the IP address.	Y	M(*)
	FblEthLocalPort	The local IP port of the Ethernet ECU. Enter a value between 1 and 65535.	Y	M(*)
	FblEthNetworkType	The network type that can be IP_NET-WORK_1_INFOTAINMENT or IP_NET-WORK_2_ACTIVE_SAFETY. Together with the diagnostic address, this defines the logical ad-dress of the ECU.	Y	M(*)
	FblEnableCanFd	This specifies whether CanFd support is to be enabled. Can be set to either “true” or “false”.	Y	M(**)
	FblEcuIdSupport	Specifies if the user callback function Fbl_Port_GetEcuIdUserHook() is to be called to read the Ecu Id, or if the Ecu Id is to be read from NvM. If set to ECUID_USER_SUP-PORT, then the callback function is called. If	N	O

		set to ECUID_NVM_SUPPORT, then the Ecu Id is read from NvM. If not specified, then the default behavior is to use NvM (i.e. the behavior is as if ECUID_NVM_SUPPORT is configured).		
	FblReprogrammingRequestFlag	For targets that support the reprogramming request flag in RAM, this parameter can be set to REPROGRAMMING_FLAG_NVM or REPROGRAMMING_FLAG_RAM to indicate whether the jump from Application to FBL should be done via NvM or via a RAM flag.	Y	O
	FblSpUpdateSupported	Specifies whether updating of the security peripheral is to be supported (set to SP_UPDATE_ON) or not (SP_UPDATE_OFF). Note that even if SP update is not required, this parameter should be set to SP_UPDATE_ON if compatibility checks between the HSM application and software application is required.	N	M
	FblBlSpblCid	The FBL-HSMFBL compatibility id. Enter an integer value between 0 and 65535.	N	M(***)
FblCore	EraseTimeout	Max time in milliseconds for erase flash before timing out. Enter an integer value between 1 and 100000.	N	M
	VerifyTimeout	Max time in milliseconds for PSI setting before timing out. Enter an integer value between 1 and 100000.	N	M
	WriteTimeout	Max time in milliseconds for write on flash before timing out. Enter an integer value between 1 and 100000.	N	M
	StartAddress	Memory address of first instruction of application software.	N	M

(*) The parameter is present only if FblNetwork = ETH

(**) The parameter is present only if FblNetwork = CAN

(***) The parameter is required only if FblSpUpdateSupported = SP_UPDATE_ON

(****) For CAN configurations this is used as part of the Can Id. For Ethernet configurations, this is used in PDU headers.

3.5.3 Timing consideration for SBA ticket validation

The SBA ticket validation is done during bootloader initialization. This process could take in the order of a few seconds to complete. During this time, an erase request which is required before programming any partition will return a pending NRC. This must be considered in the EraseTimeout unless your tester programming script (your utility file in DPS) already provides a delay after entering the programming session.

The DiD F0F2 for reading the Boot Initialization Status also needs to wait for the SBAT ticket validation to complete before it can provide a valid response. In the case of this DiD, you will get a general reject response (NRC 0x10) until the SBAT ticket has been validated.

3.5.4 Files created during generation

When you generate an instance of the GM RTA-FBL using the RTA-FBL plugin for ISOLAR-AB, a series of files are created within a number of folders that you then use to build your RTA-FBL instance. Table 3 summarizes the folder structure created for the GM port. Additional folders will be created that contain the target-specific elements, such as target code and sample build scripts. See your RTA-FBL GM Target Guide for details of the content of these additional folders.

Table 3: 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. Do not modify these files.
./fbl/output/Fbl/Bootloader	This contains the core (port-independent and target-independent) modules.
./fbl/output/Fbl/BootUpdater	This contains the boot updater for this project. The boot updater is a completely separate application that is generated by the fbl generator but must be built independently of the bootloader. This folder is created only if your target supports boot update.
./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 any changes do not affect the bootloader's correct functionality.
./fbl/output/Fbl/INFRA/BCL	This is the ETAS crypto library (ECL).
./fbl/output/Fbl/INFRA/BLSM	The BLSM contains code for initializing the Bootloader. The functions in ./src/BLSM_CallOuts.c can be changed as described in Section 3.5.10, but the functions in BLSM_Main.c should not be changed. It is the integrator's responsibility to test any changes made in the BLSM do not affect the bootloader's correct functionality.

./fbl/output/Fbl/INFRA/OS	The OS contains the cyclic scheduler that calls the module main functions. The OS is provided as a fully functioning and tested sample, but the integrator may replace the OS as described in Section 0. 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 test 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. This code should not be modified by the integrator with the exception of the file FBL_PortUserCode.c as described in Section 3.5.7.
./fbl/output/Fbl/INFRA/Stubs	This folder contains stub code necessary due to the AUTOSAR architecture. Files in this folder should not be modified by the integrator.

3.5.5 The RTA-FBL instance for the Dummy Target

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

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

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

Dummy Target Memory Layout

To allow the user to experiment with different memory space configurations, the dummy target is set up to mimic the memory layout of Infineon's TC275 processor. This processor has a memory layout as shown in Table 4. Memory regions of a space must begin on sector boundaries and the bootloader reserves the first sector (i.e. the memory between 0xA0000000 and 0xA0003FFF). You can experiment with different configurations of Application, Calibration and Bootloader space if you have not yet received your Target package. For example, if you configure a space that uses a memory region that is not on a region boundary or that enters enter a disallowed space and note the error returned by the FBL generator.

Table 4: Memory layout of the Dummy Target

Bank	Sector	Start	End	Comment
0	0	0xA0000000	0xA0003FFF	Reserved for FBL
	1	0xA0004000	0xA0007FFF	Available for Applica-tion/Calibration
	2	0xA0008000	0xA000BFFF	
	3	0xA000C000	0xA000FFFF	
	4	0xA0010000	0xA0013FFF	
	5	0xA0014000	0xA0017FFF	Not available for Applica-tion/Calibration
	6	0xA0018000	0xA001BFFF	
	7	0xA001C000	0xA001FFFF	Available for Applica-tion/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 Applica-tion/Calibration
	17	0xA0070000	0xA007FFFF	
1	18	0xA0080000	0xA008FFFF	Available for Applica-tion/Calibration
	19	0xA0090000	0xA009FFFF	
	20	0xA00A0000	0xA00BFFFF	
	21	0xA00C0000	0xA00DFFFF	
	22	0xA00E0000	0xA00FFFFF	
2	23	0xA0100000	0xA013FFFF	
	24	0xA0140000	0xA017FFFF	
3	25	0xA0180000	0xA01BFFFF	
	26	0xA01C0000	0xA01FFFFF	
4	0	0xA0200000	0xA0203FFF	
	1	0xA0204000	0xA0207FFF	
	2	0xA0208000	0xA020BFFF	
	3	0xA020C000	0xA020FFFF	
	4	0xA0210000	0xA0213FFF	
	5	0xA0214000	0xA0217FFF	
	6	0xA0218000	0xA021BFFF	
	7	0xA021C000	0xA021FFFF	

	8	0xA0220000	0xA0227FFF	
	9	0xA0228000	0xA022FFFF	
	10	0xA0230000	0xA0237FFF	
	11	0xA0238000	0xA023FFFF	
	12	0xA0240000	0xA0247FFF	
	13	0xA0248000	0xA024FFFF	
	14	0xA0250000	0xA0257FFF	
	15	0xA0258000	0xA025FFFF	
	16	0xA0260000	0xA026FFFF	
	17	0xA0270000	0xA027FFFF	
5	18	0xA0280000	0xA028FFFF	
	19	0xA0290000	0xA029FFFF	
	20	0xA02A0000	0xA02BFFFF	
	21	0xA02C0000	0xA02DFFFF	
	22	0xA02E0000	0xA02FFFFF	
6	23	0xA0300000	0xA033FFFF	
	24	0xA0340000	0xA037FFFF	
7	25	0xA0380000	0xA03BFFFF	
	26	0xA03C0000	0xA03FFFFF	

Integrator guidelines

Section 3.5.1 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),
- OS adaptation (optional),
- BLSM adaptation (optional).

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

- NvM layout adaptation,
- Boot jump handling.

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

- Completion of target-specific user functions
- C-code startup and trap table updates,
- MCAL adaptation,
- Integration of the SP update library if SP update is required.

3.5.6 FBL: Memory Layout Adaptation

To integrate the FBL in your application the first step to do is decide how to set up your memory regions. This is done using the configuration tool as described in Section 3.5.2. The allowed range for your target is described in you RTA-FBL GM Target Guide. An example of a typical memory layout is depicted in Figure 26.

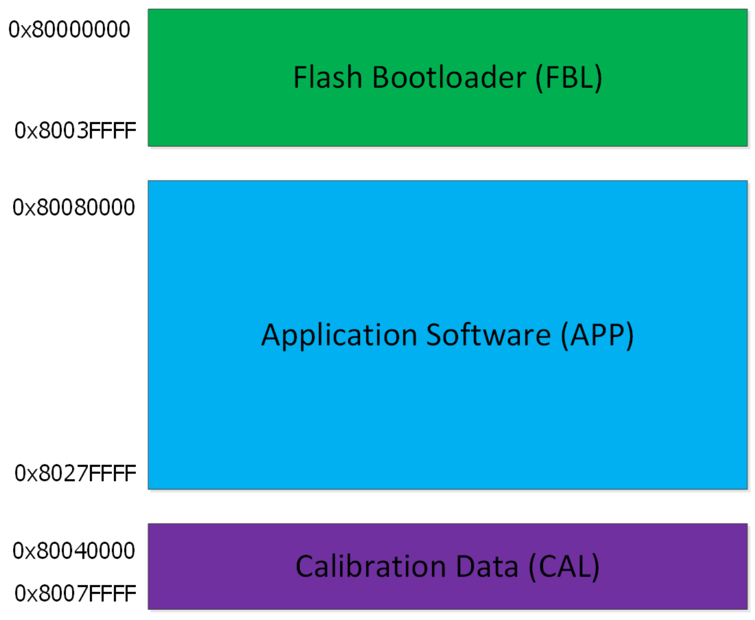


Figure 26 - Sample Memory Layout

3.5.7 FBL: User Functions

You will find all user functions that you need to complete in the generated file FBL_PortUserCode.c within the \Ports\GM\INFRA\Port\src folder of your generated FBL instance. The exception is the functions for Persistence which are found in FBL_PersistenceUser.c/h and Fbl_PersistenceUserCfg.c/h. in the Bootloader\Persistence folder.

Sleep

The function `Fbl_Port_Sleep` is called by the bootloader when no UDS messages are received after 20 seconds. The integrator should modify this code to put the ECU to sleep. The integrator must also implement the wakeup logic that will result in the ECU restarting from reset.

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 from within the 20ms task (`OsTask_20ms`), but the integrator can call `Fbl_Port_WatchDogRefresh` at whatever rate is deemed suitable.

Hook Functions

On some targets, the follow hook functions are provided. See your GM target guide for further information on whether your target supports these hook functions:

- `Fbl_Port_TargetWritePreHook`: This function is called before the start of writing to flash
- `Fbl_Port_TargetWritePostHook`: This function is called after the end of writing to flash

- `Fbl_Port_TargetErasePreHook`: This function is called before the start of flash erase
- `Fbl_Port_TargetErasePostHook`: This function is called after the end of flash erase
- `Fbl_Port_CheckSignaturePreHook`: This function is called before the start of signature validation
- `Fbl_Port_CheckSignaturePostHook`: This function is called after the end of signature validation
- `Fbl_Port_CheckMessageDigestPreHook`: This function is called before the start of signature validation
- `Fbl_Port_CheckMessageDigestPostHook`: This function is called after the end of signature validation
- `Fbl_Port_PreHsmUpdateHook`: This function is called before the start of HSM update in the Mini FBL
- `Fbl_Port_PostHsmUpdateHook`: This function is called after the end of HSM update in the Mini FBL
- `Fbl_Port_ExcuteHsmUpdateHook`: This function is called in the Mini FBL while waiting for completion of HSM update

Get Programming Conditional Flags – Status User Bits

The function `Fbl_Port_GetProgrammingConditionalFlagsSta-tusUserBits` is called by the bootloader when servicing the routing control 0xFF01 (Check Programming Dependencies). It needs to return a value for the third byte (Byte 2) when servicing this routine control. The integrator must set bits 3-7 are as per the GM specification [2]. Note that bits 0-2 will be cleared if set.

Get ECU Type and Current Programming Capability

The function `Fbl_Port_GetECUTypeAndCurrentProgrammingCapability` is called by the bootloader when servicing the routing control 0xFF01 (Check Programming Dependencies). It needs to return a value for the fourth byte (Byte 3) when servicing this routine control. The default value to accept programming conditions is 0x7 since this sets bits 0-2 to the default values as per the GM specification [2].

Get Additional Programming Conditional Flags Status

The function `Fbl_Port_GetAdditionalProgrammingCondition-alFlagsStatus` is called by the bootloader when servicing the routing control 0xFF01 (Check Programming Dependencies). It needs to return a value for the fifth byte (Byte 4) when servicing this routine control.

Get the ECU Id

The function `Fbl_Port_GetEcuIdUserHook` is called by the bootloader at initialization when the FBL generator parameter FblEcuIdSupport is set to ECUID_USER_SUPPORT. You need to return the 16-byte value of the ECU Id depending on how this is provisioned for your ECU.

Persistence for user management of non-volatile data

If the configuration parameter FblPersistence is set to NVM_USER, then the following functions must be completed in FBL_PersistenceUser.c:

- `Fbl_Persistence_Init`: Used to provide logic for initializing the NvM system
- `Fbl_Persistence_MainFunction`: The main function called from the DataM module
- `Fbl_Persistence_PersistenceReady`: A function that returns whether the NvM system has completed setup. When this returns true, the overlying modules can assume that calls to read and write to NvM are available.

The following functions must also be completed in FBL_PersistenceUserCfg.c for each NvM data item with name <name>.

- Fbl_Persistence_<name>ReadFunc: Read the NvM data item <name>. This function receives a reference where the current NvM data is to be copied.
- Fbl_Persistence_<name>WriteFunc: Write data to the provided NvM data item <name>. A reference is passed to this function, but the data at this location cannot be assumed to persist after the return of the caller. Therefore, the data passed to this function should be copied by the user provided NvM system before returning form this call.
- Fbl_Persistence_<name>IsPersisted: Return whether or not the data item name has persisted to non-volatile memory.
- Fbl_Persistence_<name>GetRawMirrorReference: Return a reference to the RAM data for data item <name>. The callers of this function guarantee to never modify this data. This function is used as an optimization to avoid the copying required when using Fbl_Persistence_<name>ReadFunc.

The integrator should note that when FblPersistence is set to NVM_USER, the NvM modules of RTA-CAR may still be generated on some targets in order to allow a complete BSW generation to occur. The generated files are however not used during the building processes of the instance.

3.5.8 FBL: BSW adaptation

The BSW modules needed by RTA-FBL and generated in the generated BSW project are listed in Table 5. This list is the minimum setup needed for the basic FBL. The only exception is that the values of STmin in [2] is specified as a range between 0 and 100 microseconds. This value has been set to 0 in the default generated BSW configuration. However, you may change this value in the BSW configuration if in testing you realize your setup is resulting in lost CAN frames. You may need to do this if you set larger values than 100 microseconds for the FBL configuration parameters FblSecurityTimeWindow and FblDecompressTimeWindow. System testing of CAN targets is typically done with an STmin value set to 32 microseconds.

If changes are necessary in order to fulfill non-standard bootloader integration requirements (i.e. requirements not covered in [1], [2] and [3]), then it is strongly recommend that you first contact ETAS' support before making these changes. The integrator must always test the complete FBL after making any modifications to the generated BSW project.

Table 5: BSW modules list

BSW Module(s)	Notes
Dcm	The diagnostic communication module.
MemIf; Fee; Rba_FeeFs1, NvM	Memory stack modules for the NVM.
CanIf; CanSM; CanTp; CanTp_Precom-pile, ComM; ComStack; PduR	Can communication stack modules. (*)
EthIf; EthSM; EthTrcv; SoAd; TcpIp; PduR	Ethernet communication stack modules. (*)
Crc	Uses for CRC calculation in NvM.
Rba_ArxmlGen; Rba_DiagLib; Standard; BswM; BswM_Precompile_PB_Variant	Additional modules required for build.

(*) They are mutually exclusive and depend on the network the target supports

3.5.9 FBL: OS adaptation

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

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

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

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

Finally, the #define `FBL_MEASURE_TASKS` can be set in Os_Tasks.h to measure the amount of time spent tasks during the execution of a download sequence. This can help you to benchmark the amount of time spent in each co-routine. Note that this feature is should only be used for development and is only supported when the bootloader is entered directly (that is, not from a programming session in the application). The data collected for each routine is described in Table 6. The routines that are captured are:

- `Can_MainFunction_Write` (for CAN targets only)
- `Can_MainFunction_Read` (for CAN targets only)
- `Can_MainFunction_BusOff` (for CAN targets only)
- `CanTp_MainFunction` (for CAN targets only)
- `CanSM_MainFunction` (for CAN targets only)
- `Eth_MainFunction` (for Ethernet targets only)

- `EthIf_MainFunctionRx_EthIfPhysCtrlRxMainFunctionPriorityProcessing` (for Ethernet targets only)
- `EthIf_MainFunctionState` (for Ethernet targets only)
- `EthSM_MainFunction` (for Ethernet targets only)
- `SoAd_MainFunction` (for Ethernet targets only)
- `EthTrcv_MainFunction` (for Ethernet targets only)
- `NvM_MainFunction`
- `Fee_MainFunction`
- `Fls_MainFunction`
- `BLSM_MainFunction`
- `Fbl_Port_SecurityMainFunction`
- `Fbl_Port_MainFunction`
- `Dcm_MainFunction`
- `Fbl_ProgM_MainFunction`
- `Fbl_BootM_MainFunction`
- `Fbl_DataM_MainFunction`

Table 6: Timing benchmark variables

Variable	Description
`timing_total_ms<routine>`	The total amount of time in milliseconds spent in routine <routine>
`timing_low_10us<routine>`	The lowest amount of time in tens of microseconds spent in routine <routine>
`timing_high_10us<routine>`	The highest amount of time in tens of microseconds spent in routine <routine>
`timing_average_10us<routine>`	The average amount of time in tens of microseconds spent in routine <routine>
`iterations_<routine>`	The number of times that the routine <routine> executed

3.5.10 FBL: BLSM adaptation

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

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

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

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

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

3.5.12 FBL: MCAL adaptation

The MCAL modules needed by RTA-FBL for all targets are listed in Figure 3. The list is the minimum setup needed for the basic FBL functionalities (i.e. communication, flashing, etc.). The list does not include customer specific adaptations like external watchdog, external transceivers, external EEPROM, etc. See your RTA-FBL GM Target Guide for further information on MCAL modifications for your target.

Table 7: MCAL modules list

MCAL Module	Notes
Can	CAN driver (*)
Eth	Ethernet driver (*)
Flash Driver	Driver for FLASH erase and programming. This includes the handling of PFLASH and DFLASH, so in some MCALs these drivers are implemented by two different modules.
Mcu	Provides core functionality such as clock handling, mcu reset, etc.
Port	Provides interface to port pin peripheral.

(*) They are mutually exclusive and depend on the network the target supports

3.5.13 FBL: Integrating an External CAN Transceiver into an RTA-FBL Project

Earlier sections provided an explanation of how the BSW (Section 3.5.8), OS (Section 3.5.9) BLSM (Section 3.5.10) and MCAL (Section 3.5.12) could be adapted. This section provides a brief overview of how a Can Transceiver can be integrated into the FBL by adapting these modules. It serves as an example of how to integrate a Can Transceiver that has already been integrated into an application that uses a newer version of RTA-BSW (e.g., RTA-BSW 6.1) than is supported by the bootloader (RTA-BSW 5.1). Therefore, the integration engineering work should be limited to transferring the existing working transceiver logic in the application to the bootloader.

Note that, although this example assumes that the transceiver is supported by the version of RTA-BSW used in the application, it is also possible to integrate 3rd party transceivers that are not supported in RTA-BSW. This would have to be done in the same way that you would have had to integrate the transceiver in the application. As is the case with the integration of all third-party code, testing of the entire integrated system should be carried out.

Also, note that any new code integrated in the following steps should be added to the build system.

BSW Configuration

To integrate a CAN Transceiver into an RTA-FBL project the following BSW configuration should be added to the FBL project:

- CanTrcv,
- CanIf (Configuration already exists in FBL, so it is only necessary to add configuration that references the CanTrcv module),
- CanSM (Configuration already exists in FBL, so it is only necessary to add configuration that references the CAN Transceiver),
- Stubbed MCAL module for CanTrcv references (Spi or Dio; RTA-BSW will only use this as a reference).

The RTA-FBL Generator includes RTA-BSW 5.1. If a CAN Transceiver is not supported in RTA-BSW 5.1, but it is in a subsequent version of the tool (e.g. RTA-BSW 6.1), then it is possible to integrate a newer version of the CanTrcv module using a newer version of RTA-BSW. This would require the following steps:

1. Generate the CanTrcv module using the newer version of RTA-BSW.
2. Copy over CanTrcv and MCAL (Spi or Dio) config as well as the parts of CanIf and CanSM that reference the CanTrcv.
3. Copy over the CanTrcv paramdef from the newer RTA-BSW project.
4. Regenerate the original RTA-BSW project to reflect the changes in configuration to CanIf and CanSM. Note that CanTrcv and Rba_Trcv generation should be turned off during this step to prevent an error being thrown by RTA-BSW.
5. Copy over the generated CanTrcv code from the newer RTA-BSW.

MCAL Configuration

The Spi or Dio configuration required to communicate with the CanTrcv should be added to the MCAL project and the code regenerated. The names of the configuration items should match the names in the stubbed MCAL module added to the BSW project.

Integration Code in the OS and BLSM

BSW initialization functions should be added to the Fbl_Port_BLSM_DriverInit functions in BLSM_Callouts.c. The CAN Transceiver initialization function should be called after the MCAL initialization function (Spi or Dio) and before the initialization function for CanIf. Certain CAN transceivers may require a mainfunction to be executed periodically. This should be added to one of the tasks in Os_Tasks.c. The steps required in this section are explained in more detail in sections 3.5.9 and 3.5.10.

Wakeup Interrupts

Even though interrupts are not supported by the cyclic OS provided in RTA-FBL, wakeup interrupts can still be added to the interrupt table as the bootloader OS scheduler will not be running until wakeup is complete. The process of implementing the interrupts is target and transceiver specific and is the responsibility of the integrator.

3.5.14 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 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 correct integration of NvM layout comprise the complete copy of FBL NvM blocks on the application NvM layout. In order for the layout to be consistent, the Fee persistent IDs of the blocks must match between the application and FBL. Table 8 shows the NvM blocks and their Ids that must be also configured in the application. Note that if you already are using the Persistent IDs generated by fblgen in your application and do want to change these, then you can instead change these values in the FBL instance's BSW configuration as long as you keep them consistent with the values in the Application.

Table 8: Fbl NvM data

Persistent Id*	Name	Size in Bytes	Description
2	Reprogramming Request Flag	4	The reprogramming request flag that is set by the application when it enters the Programming session. The bootloader clears this flag on startup.
3	Signature Bypass Ticket	822	The SBA ticket is set by the application and read by the bootloader to implement bypass features as described in [3].
4	Key NBID	2	Used internally by the FBL to guarantee consistency of the Key NBID.
5	Ecu Id	16	The ECU Id is written by the application or during ECU provisioning and read as part of the response to DiD 0xF0F3. Note that this NvM value is always present, even if FblEcuIdSupport is set to ECUID_USER_SUPPORT.
6	App NBID	2	Used internally by the FBL to guarantee consistency of the App NBID.
7	Programming Conditions	31	Used to save programming conditions that are read by the bootloader on startup. See 3.5.15 for more information.
8	GM End Model Part Number	4	The GM End Model Part Number is written by the application and read as part of the response to DiD 0xF1CB.
9	GM End Model Part Number Alpha Code	2	The GM End Model Part Number Alpha Code is written by the application and read as part of the response to DiD 0xF1DB.
10	Diagnostic Address	1	The diagnostic address of the ECU. This value is not written by the bootloader but can be set up on ECU commissioning. It is only used if the bootloader is generated with DiagnosticAddressIsStatic not set.

*Note, if the target supports jumping to boot in Ram and FblReprogrammingRequestFlag is true, then the persistent Id columns are decremented by 1 and the Reprogramming Request Flag NvM block is excluded.

3.5.15 Application Software: Boot Jump Handling

To reprogram the ECU when an application is valid and running, it is necessary for the application to signal to the bootloader that reprogramming is required after the next reset. This is done by setting the reprogramming request flag (id 2 in Table 8) NVM and then entering the programming session. For some targets, the jump to boot can be done via a RAM flag. In this case, the NvM block is not configured. This sequence is show in Figure 27.

The programming conditions (id 7 in Table 8) must also be set. The data bytes are used to build the contents of a BSW structure of type `Dcm_ProgConditionsType`. They must be serialized as shown in Table 1 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 the values needed to create data by using the contents of the programming conditions sent as an input parameter to the function `Dcm_SetProgConditions` called from the Dcm when the Programming Session is entered in the application. Note that the Tester Source address (`TesterSourceAddr`) is set by the Dcm only if DcmDslProtocolRxTesterSourceAddr is correctly configured for each DcmDslConnections in BSW configuration.

The freeForProjectUse (id 25 in Table 8) must also be set. As the first byte of freeForProjectUse is used to activate DCM connections in bootloader after jumping from application to bootloader.

For example, if the Application received programming diagnostic session control in extended CAN2.0 connection, Then the Application must set freeForProjectUse[0] to be equal 2. So that the bootloader will activate extended CAN2.0 connection in the warm start.

For FBL that supporting both CAN 2.0 and CAN FD protocols the application should set freeForProjectUse[0] to activate the desired connection as in table 9:

Table 9: Active Connection based on freeForProjectUse for FBL that support CAN2.0 and CAN FD protocols.

ProgConditions->freeForProjectUse[0] value	Active connection in fbl after jumping from application to bootloader
1	extended CANFD connection
2	extended CAN2.0 connection
3	standard CAN2.0 connection
4	standard CANFD connection

For FBL that supporting only CAN 2.0 protocol then the application should set freeForProjectUse[0] to activate the desired connection as in table 10:

Table 10: Active Connection based on freeForProjectUse for FBL that support CAN2.0 protocol.

ProgConditions->freeForProjectUse[0] value	Active connection in fbl after jumping from application to bootloader
2	extended CAN2.0 connection
3	standard CAN2.0 connection

Finally, note that it is important that the application check that the writes to NvM data are persisted (and not just present in the NvM's mirror RAM) before the reset takes place.

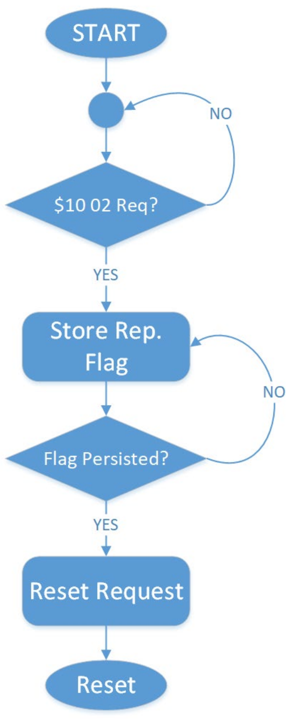


Figure 27: Handling of jump logic

Table 11: Programming conditions data to be set in NvM

Name	First byte index	Size in Bytes	Description
ProtocolId	0	1	Active Protocol ID – Should be set to 0x03 to indicate UDS
Sid	1	1	Active Service Identifier – Should be set to 80 to indicate the response to service 0x10 (session control)

SubFncId	2	1	SubFncId – Should be set to 2 to indicate request to enter the programming session
StoreType	3	1	Storing Type used for Storing the information – Should be set to 3
SessionLevel	4	1	Active Session – Should be set to 3 to indicate the extended session
SecurityLevel	5	1	Active Security – Should be set to 1 to indicate the ECU is unlocked
ReqResLen	6	1	Response Length – Should be set to 6 to indicate the response length should be made of 6 bytes
NumWaitPend	7	1	Number of waitpends triggered – Should be set to 1
ReqResBuf	8	8	Request / Response buffer – Should be set to the response parameters for the diagnostic session change; the first four bytes are required here as the response bytes to this UDS request.
TesterSourceAddr	16	2	Tester diagnostic address – Should be set to a value between 0xF1 and 0xF6
ElapsedTime	18	4	Total elapsed time – This should be retained from the Dcm provided value when `Dcm_SetProg-Conditions` is called.
ReprogramingRequest	22	1	Reprograming of ECU requested – This value should be set to 1.
ApplUpdated	23	1	Application has to be updated or not – This value is ignored as the programming request flag uses its own NvM data or making a programming request as shown in Table 8.
ResponseRequired	24	1	Response has to be sent by flashloader or applica-tion – This should be set to 1.
freeForProjectUse	25	6	The first byte of freeForProjectUse is used to acti-vate the desired `DcmDslMainConnection` when jumping from Boot to Application

3.5.16 Implementing a back door

The GM specification does not provide a back-door mechanism to force the ECU to enter boot mode when the application needs to be reprogrammed in situations where the appli-cation is unable to successfully enter the programming session. This could happen if there is a bug in the jump logic of application or if the start address of the application is not at the address jumped to by the bootloader. If the integrator wishes to implement such a backdoor mechanism, then this should be done by setting the reprogramming flag in NvM (persistent Id 2) to 0xAAFF55AA and causing a reset. This should be done in `Fbl_Port_BLSM_DriverInitOne` in the generated file BLSM_Callouts.c before the function `Fbl_BootM_CheckEcuState` is called. This value can also be specified using the define `FBL_BOOTM_REPROGRAMMING_REQUEST_FLAG_STATE_ON` in BootM.h.

3.5.17 Security Peripheral Update

Some GM target ports support update of the security peripheral (HSM) as per [3]. To enable this feature on these targets and generate the FBL instance using FblSpUpdateSupported set to SP_UPDATE_ON, you will need to add a number of files that would be delivered separately from the installer RTA-FBL GM Port installer. Please refer to your target guide for information on where to copy these files to.

3.6 Preparing an application/calibration for download

The FBL expects binary files sent to ECU to be programmed to follow the format described in [3]. Therefore, the user is expected to prepare the binary using the required envelopes, header data and info section. If a binary is to be compressed using the supported ARLE mechanism, then this must be done as described in [3]. The bootloader will reject any incorrectly formed binary and return the relevant NRC and set the PEC as described in [3].

In preparing the application header, note that no Alignment Padding is required in the Application Signed Header. Therefore, the bootloader expects that the application info section will follow immediately after the header in the downloaded binary.

3.7 Boot Update

Boot update is supported for some targets. A separate application (the “boot updater”) is created that you would download to your ECU to replace your application. This application has one calibration module that contains the binary of the new bootloader.

3.7.1 The bootmanager and the boot update process

Figure 28 shows the process of boot update. Targets that support boot update will always have a bootmanager. On reset, the bootmanager will check if the FBL is valid and jump to the bootloader if it is valid. If the bootloader is not valid, then the bootmanager 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 bootmanager is the only non-replaceable software and is used to ensure boot update is failure safe. It will hold a refence to the start address of the bootloader (see parameter FblBootloaderStartAddress in Section 3.5.2) that cannot be changed across bootloader updates.

Note that boot update will only be allowed by GM with an SBAT preset as GM will not sign either of the 2 modules.

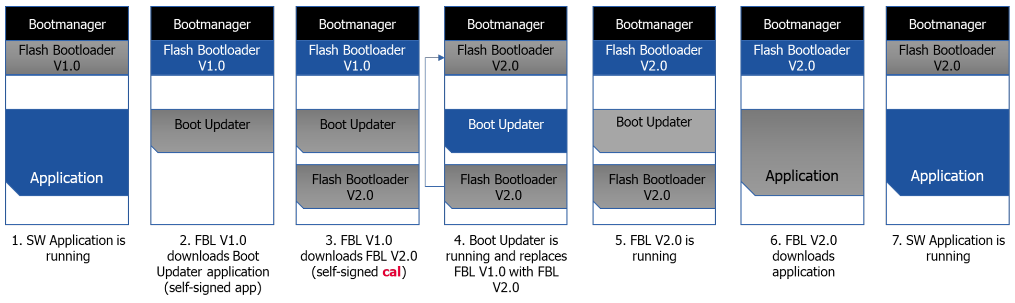


Figure 28: The boot update process

3.7.2 Preparing boot update modules

To update the bootloader, you will need to prepare two binaries: the boot updater binary and the new bootloader binary. These two binaries can be signed using the same tooling that you use to create your self-signed binaries for testing. No special tooling is required to create the boot updater application module and bootloader calibration module. They are treated by the bootloader in the same way as ordinary applications and calibration modules and must therefore have valid signed headers and info sections.

Note that the bootloader you generate will always have a bootmanager for targets that support this feature. The bootmanager should not be placed in the calibration module binary since it should never be flashed by the boot updater. Therefore, when creating the calibration module with the new bootloader, ensure that only the data that is to be flashed to the regions defined by the configuration parameter FblBootloaderSpace is contained in the module.

4 Contact Information

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 website: www.etas.com/hotlines

ETAS Headquarters

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

Figures

Revision History

Version	Author	Date	Change (Why, What)
1.0	Andrew Borg	22/06/2020	First version.
1.1.0	Andrew Borg	09/11/2020	Updated for RTA-CAR integration.
1.2.0	Andrew Borg	02/08/2021	Updated for RTA-FBL GM V1.2. This update includes support for Ethernet and SP (HSM) update.
1.3.0 RC 1	Andrew Borg	21/09/2022	Typo fixes; added section on transceiver integration; added information on boot updater
1.3.0	Jacopo Filippi	28/10/2022	Update version and screenshot
1.3.1	Andrew Borg, Alessandro Sassone	30/06/2023	Minor updates and conversion to new template. Added Licensing Section.
1.3.2	Andrew Borg	23/11/2023	Minor updates for release with bug fixes. Added information for jump-to-boot in RAM for supported targets.
1.3.3	Mahmoud Shaaban	5/12/2024	Minor updates for release with bug fixes. Updates for Can+CanFD support and integration of user-defined NvM.