Contents

[1. Introduction 2](#_Toc39136251)

[1.1. Preprocessing 2](#_Toc39136252)

[1.2. Compilation 2](#_Toc39136253)

[1.3. Assembly 3](#_Toc39136254)

[1.4. Linking 3](#_Toc39136255)

[2. make Workflow 3](#_Toc39136256)

[2.1. Structure 3](#_Toc39136257)

[2.2. Dependencies 3](#_Toc39136258)

[2.3. Standardization 4](#_Toc39136259)

[3. Features 4](#_Toc39136260)

[4. Architecture 7](#_Toc39136261)

[4.1. Makefile Structure 7](#_Toc39136262)

[4.2. Output files 7](#_Toc39136263)

[4.2.1. Mainstream 7](#_Toc39136264)

[4.2.2. Extended 9](#_Toc39136265)

[5. Integration and Configuration 10](#_Toc39136266)

[5.1. Configuration options 10](#_Toc39136267)

[5.2. Adding new Modules 12](#_Toc39136268)

[6. Walkthrough 12](#_Toc39136269)

[6.1. First use 12](#_Toc39136270)

[6.2. Normal workflow 14](#_Toc39136271)

[7. FAQ 14](#_Toc39136272)

Figures

[Figure 1 Overview of the compilation process 2](#_Toc34122762)

[Figure 2 Software Component Distribution 3](#_Toc34122763)

[Figure 3 Dependency Tracking 4](#_Toc34122764)

[Figure 4 Makefile structure 4](#_Toc34122765)

[Figure 5 Feature Overview 6](#_Toc34122766)

[Figure 7 Build Environment Components 7](#_Toc34122767)

[Figure 6 Eclipse Build Procedure 13](#_Toc34122768)

Tables

[Table 1 make commands 5](#_Toc34122769)

[Table 2 Mainstream Platform output file list 9](#_Toc34122770)

[Table 3 Extended Platform output file list 10](#_Toc34122771)

[Table 4 Configuration Option Summary 12](#_Toc34122772)

# Introduction

The purpose of the Build Environment is to compile the source code found in the software modules into machine language so that the MCU can execute the program as per the instructions given.

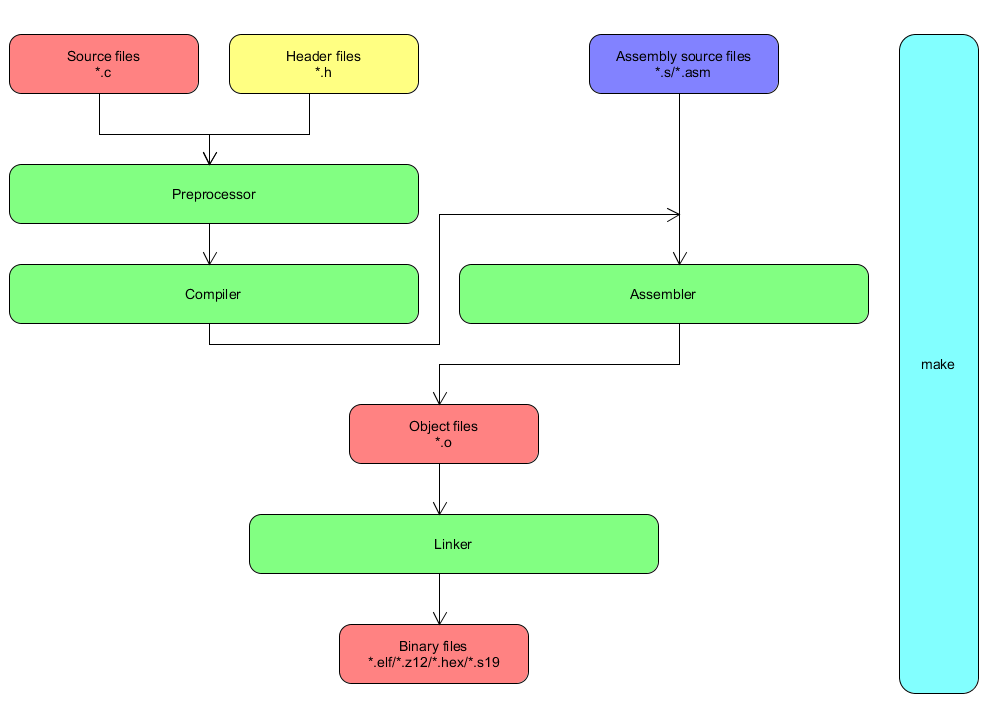


Figure 1 Overview of the compilation process

The compilation process can be split into four separate stages:

## 1.1. Preprocessing

The first stage of the build process is called preprocessing. In this stage, lines starting with a # character are interpreted by the preprocessor. These commands form a simple macro language with its own syntax and semantics.

This language is used to reduce repetition in source code by providing functionality to inline files, define macros, and to conditionally omit code.

Before interpreting commands, the preprocessor does some initial processing. This includes joining continued lines (lines ending with a \) and stripping comments.

## 1.2. Compilation

The second stage of compilation is confusingly enough called compilation. In this stage, the preprocessed code is translated to assembly instructions specific to the target processor architecture. These form an intermediate human readable language.

The existence of this step allows for C code to contain inline assembly instructions and for different assemblers to be used.

Some compilers also supports the use of an integrated assembler, in which the compilation stage generates machine code directly, avoiding the overhead of generating the intermediate assembly instructions and invoking the assembler.

## 1.3. Assembly

During this stage, an assembler is used to translate the assembly instructions to object code. The output consists of actual instructions to be run by the target processor. The contents of this file is in a binary format.

## 1.4. Linking

The object code generated in the assembly stage is composed of machine instructions that the processor understands but some pieces of the program are out of order or missing. To produce an executable program, the existing pieces have to be rearranged and the missing ones filled in. This process is called linking.

The linker will arrange the pieces of object code so that functions in some pieces can successfully call functions in other ones. It will also add pieces containing the instructions for library functions used by the program.

# make Workflow

## 2.1. Structure

The environment is designed in such a way that no matter the number of software components(e.g. Application/Bootloader/etc.) exist in the project and no matter the number of software modules(e.g. ADC, DIA, etc.) we are able to easily configure them in groups in order to be able to control all aspects of the compilation process (e.g. compiler options, source file locations, source file generation methods, etc.).

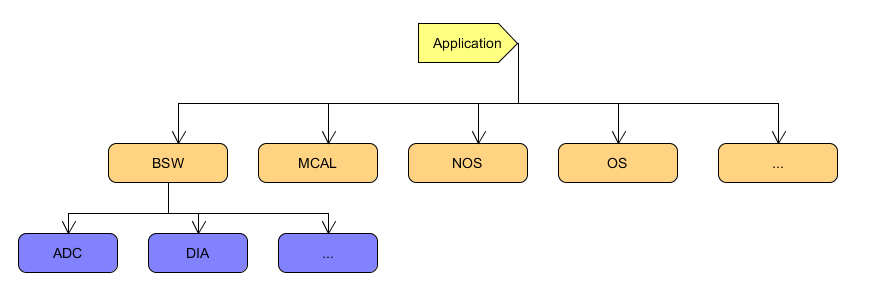


Figure 2 Software Component Distribution

*Note:* Each third-party software module that will be used in our Application will come with its own set of compiler options and restrictions. In most cases if these conditions are not respected, warranty on the third-party software supplier will be lost.

## 2.2. Dependencies

Dependencies are the mechanism that make uses in order to know what input each artifact (object files, binary files, etc.) needs. This also leads to a very well optimized build process where we can avoid full Application compilation for small modifications.

One advantage of structuring software components in the manner mentioned above is that a dependency between a configuration file (e.g. GENy configuration, Tresos MCAL arxml) and the generated source files can be created, so that every time the configuration is modified, the build will know to trigger the generation of the source files automatically, thus reducing the probability of human error.

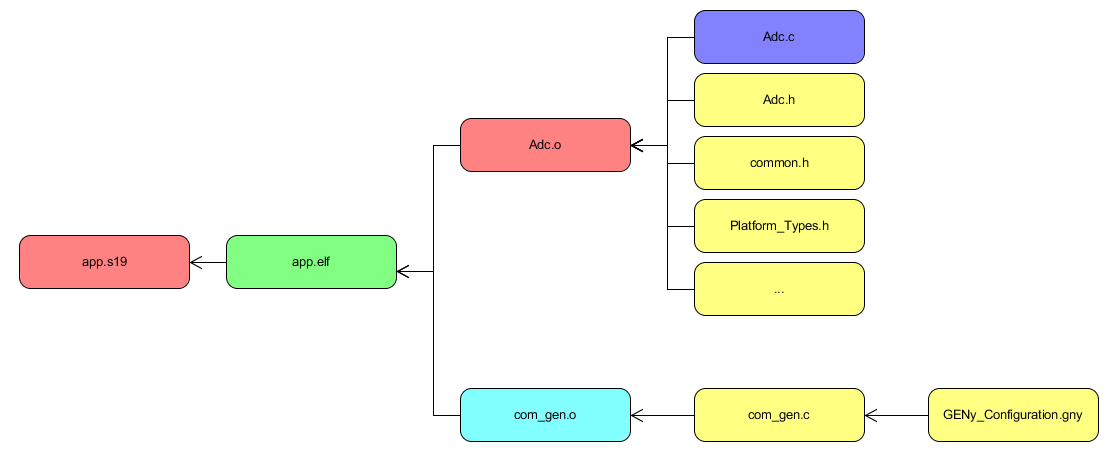


Figure 3 Dependency Tracking

*Example\_1:* If *Adc.c* is modified then make will trigger the compilation process for *Adc.o* then the linking process for *app.elf* and then the binary post processing for *app.s19*.

*Example\_2:* Ifthe GENy\_Configuration.gny is modified then make will trigger the generation process of the *com\_gen.c* source file and then it will continue the flow as mentioned in *Example\_1*.

## 2.3. Standardization

Since it is important to be able to use this environment no matter the microcontroller or the compiler, it is designed in such a way that it can be split into two components, a core environment that consists of helper functions, python scripts and generic targets for compilation and linking, and a configuration part where we have all the information needed for designing our Application(module groups, source files, include paths, software versioning, compiler options, compiler selection, etc.).

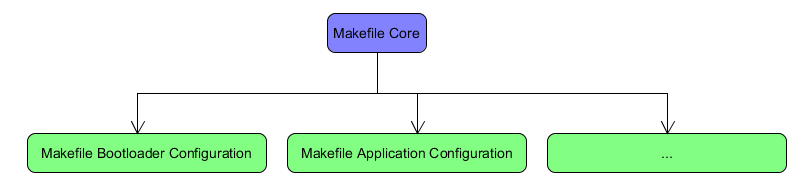


Figure 4 Makefile structure

# Features

The following features are available for use in the build environment:

* Compilation process

|  |  |
| --- | --- |
| Target | Description |
| make <variant> | build the selected software component |
| make <variant>-clean | cleans the selected software component |
| make <variant>-eclipse | generates the eclipse project and workspace for the selected component |
| make <variant>-sanity\_checks FILTER=<source file> | runs vera++ on the selected source file and checks basic coding rules  e.g. make err-sanity\_checks FILTER=Adc.c |
| make all | cleans and builds all the available software components and sets BUILD\_MODE to DEBUG |
| make all-release | cleans and builds all the available software components and sets BUILD\_MODE to RELEASE |
| make all-eclipse | generates the eclipse project and workspace for all the available components |
| make help | prints the available commands |

Table 1 make commands

* Dynamic Eclipse Project Generation: By using the inputs in the configuration part of the environment (basically what modules we are using) we can dynamically generate and update the eclipse project at any time.
  + This will also add all the defines/includes we are using in order to have correct indexing.
  + It will generate build targets so you can build from eclipse, and tool launchers, so you can do things like open the GENy configuration.
* Automatically generate source files at build
  + Currently the generation process for the NVP and ERH modules were integrated into the build flow, so anytime you modify the configuration files the build knows that it has to generate the files if the step was not performed manually.
  + GENy tool was integrated and can generate source files automatically from the \*.gny config
* PDF Report Generation: During the release build, the environment will generate a pdf file with all the information that it has available, source files, compiler options, defines.
  + It will also check that the source files that you currently have in the sandbox are the latest revision in the PTC project.
* vera++
  + Define custom rules in python for code analysis.
* [TBD] Peer Review Analyzer
  + Parses Peer Reviews and checks that the last reviewed revision is up to date with PTC revision

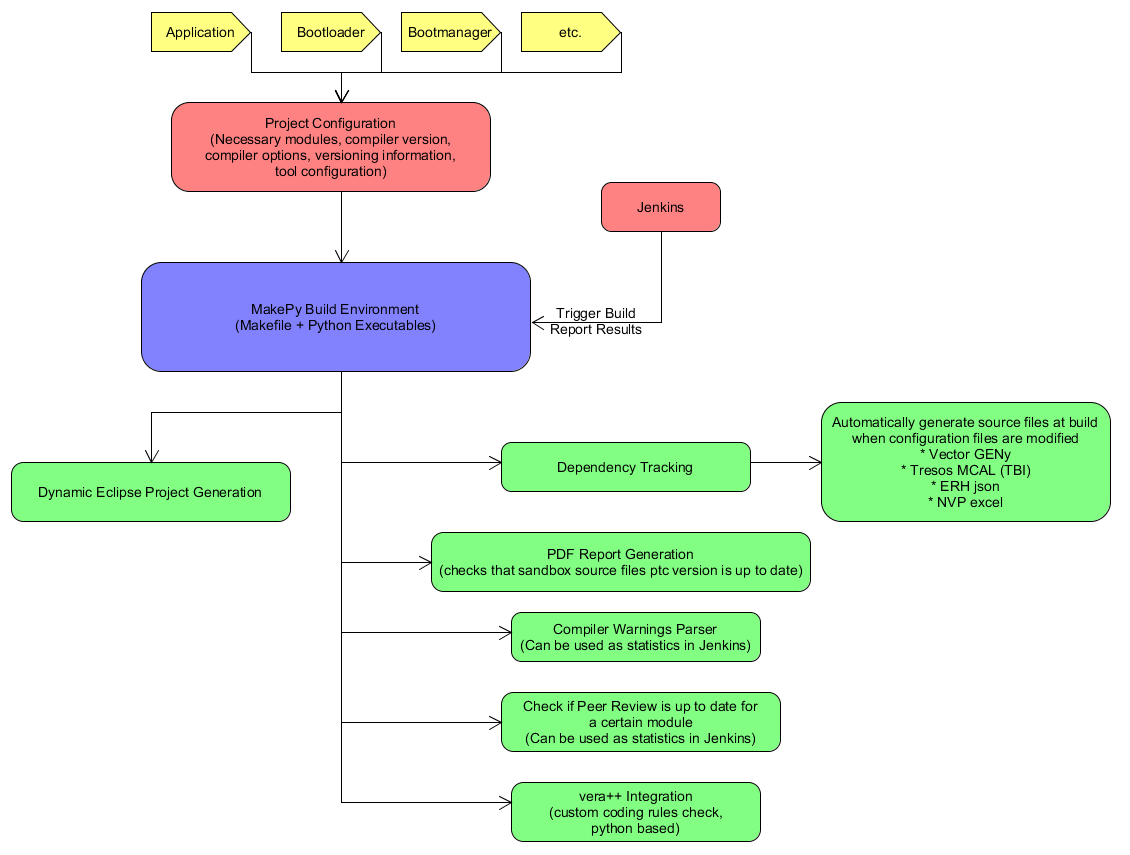


Figure 5 Feature Overview

# Architecture

## 4.1. Makefile Structure

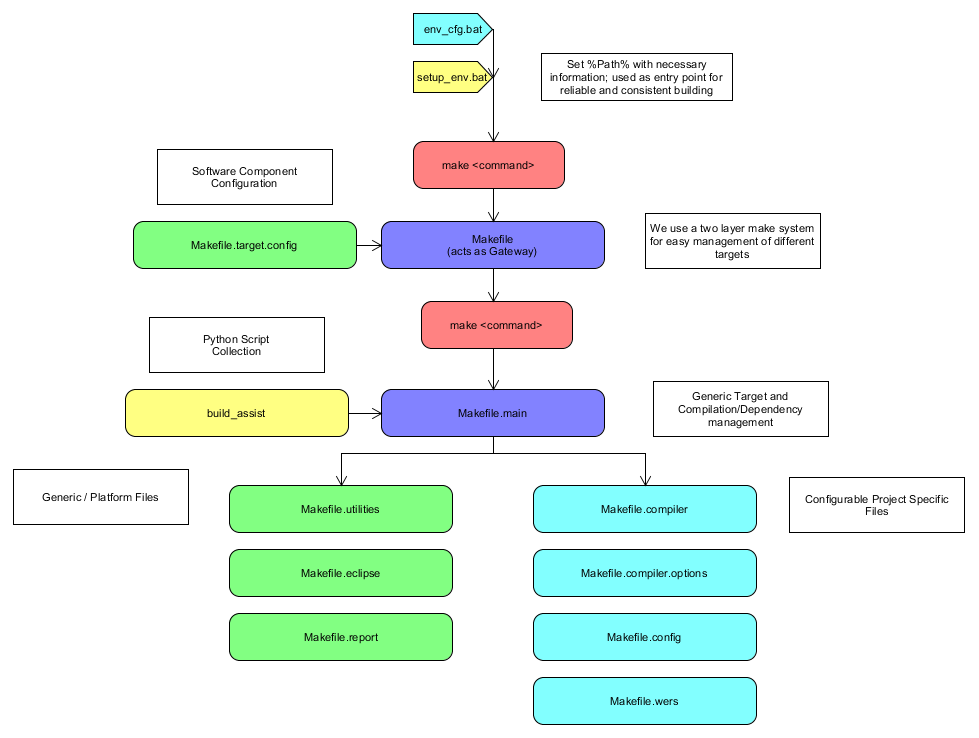


Figure 6 Build Environment Components

## 4.2. Output files

### 4.2.1. Mainstream

|  |  |  |
| --- | --- | --- |
| Folder | File | Description |
| Application | AEC\_Calibration.S00 | File generated by ERH/Config/AEC\_Configurator.jar |
|  | app.elf | File generated with cvdwarf.exe from app.z12 |
|  | app.map | Output of clnk.exe |
|  | app.s19 | File generated with chex.exe from app.z12 |
|  | app.sym | Output of clnk.exe |
|  | app.z12 | Output of clnk.exe |
|  | app\_crc.s19 | app.s19 + Checksum Calculation |
|  | app\_crc\_customer.s19 | app\_crc.s19 + Fill-up |
|  | CalibrationFlash.S00 | File generated by NVP/Config/SBE\_4G\_NVP\_layout.xls |
|  | Default.S00 | File generated by NVP/Config/SBE\_4G\_NVP\_layout.xls |
|  | Default\_customer.s19 | Default.S00 + Fill-up |
|  | DynamicE2P.S00 | File generated by NVP/Config/SBE\_4G\_NVP\_layout.xls |
|  | E2P\_Study.S00 | File generated by NVP/Config/SBE\_4G\_NVP\_layout.xls |
|  | EepSymbol.sym | File generated by NVP/Config/SBE\_4G\_NVP\_layout.xls |
|  | FullFlash.s19 | fbl.s19 + Unified\_Calibration\_customer.s19 + Default\_customer.s19 + app\_crc\_customer.s19 + Validation Flag in EEPROM |
|  | Unified\_Calibration.s19 | AEC\_Calibration.S00 + CalibrationFlash.S00 |
|  | Unified\_Calibration\_customer.s19 | Unified\_Calibration.s19 + Fill-up |
| Bootloader | fbl.elf | File generated with cvdwarf.exe from fbl.z12 |
|  | fbl.map | Output of clnk.exe |
|  | fbl.s19 | File generated with chex.exe from fbl.z12 |
|  | fbl.sym | Output of clnk.exe |
|  | fbl.z12 | Output of clnk.exe |
| CanFlash | app\_crc\_customer.s19 | Application Binary that will be used for SWDL |
|  | c\_key.txt | Secret key used in signature verification in SWDL process |
|  | Default\_customer.s19 | Default E2P Binary that will be used for SWDL |
|  | fbl.s19 | Bootloader Binary that can be used for SWDL if Bootloader\_Updater is implemented |
|  | FlashDrv.s19 | Secondary Bootloader Binary that will be used for SWDL |
|  | Unified\_Calibration\_customer.s19 | Calibration Binary that will be used for SWDL |
| DIA | DIA\_ReadDataByIdentifier.c | Source file used at build time, used to check compiled Software Version |
| ERH | ErhDataDictionnary.arxml | Arxml file used by ERH/Config/AEC\_Configurator.jar |
| NVP | Nvp\_Generated.c | Source file used at build time, used to check NVP Layout |
|  | Nvp\_Generated.h | Source file used at build time, used to check NVP Layout |
|  | SBE\_4G\_NVP\_layout.xls | Generator file used at build time |
| Production | PP4GM000.034 | FullFlash.s19 |
|  | PP4GM000.F34 | FAB File generated by NVP/Config/SBE\_4G\_NVP\_layout.xls |
|  | PP4GM000.F34.xls | Generator file used at build time(NVP/Config/SBE\_4G\_NVP\_layout.xls) |
| Release\_log | Application\_build\_report.pdf | Report containing all files compiled, including compiler options, defines |
|  | Bootloader\_build\_report.pdf | Report containing all files compiled, including compiler options, defines |
| SAP | UNI\_FAB\_NVP\_SBE\_PP4G\_R034M.SAP | SAP File containing FullFlash.s19 + FAB File |
|  | UNI\_NVP\_SBE\_PP4G\_R034M.SAP | SAP File containing FullFlash.s19 + E2P\_Study.S00 file |
|  | UNI\_SBE\_PP4G\_R034M.SAP | SAP File containing FullFlash.s19 |
| Secondary\_Bootloader | c\_key.txt | Secret key used in signature verification in SWDL process |
|  | FlashDrv.s19 | Secondary Bootloader Binary that will be used for SWDL |

Table 2 Mainstream Platform output file list

### 4.2.2. Extended

|  |  |  |
| --- | --- | --- |
| Folder | File | Description |
| Application | app.elf | Output of ilinkarm.exe |
|  | app.map | Output of ilinkarm.exe |
|  | app.s19 | File generated with ielftool.exe from app.elf |
|  | app\_crc.s19 | app.s19 + Checksum Calculation |
|  | app\_crc\_customer.s19 | app\_crc.s19 + Fill-up |
|  | CalibrationFlash.S00 | File generated by NVP/Config/SBE\_4G\_NVP\_layout.xls |
|  | Default.S00 | File generated by NVP/Config/SBE\_4G\_NVP\_layout.xls |
|  | Default\_customer.s19 | Default.S00 + Fill-up |
|  | E2pDump.S00 | File generated by NVP/Config/SBE\_4G\_NVP\_layout.xls |
|  | EepSymbol.sym | File generated by NVP/Config/SBE\_4G\_NVP\_layout.xls |
|  | FullFlash.s19 | fbl.s19 + Unified\_Calibration\_customer.s19 + Default\_customer.s19 + app\_crc\_customer.s19 + Validation Flag in EEPROM |
|  | Unified\_Calibration.s19 | AEC\_Calibration.S00 + CalibrationFlash.S00 |
|  | Unified\_Calibration\_customer.s19 | Unified\_Calibration.s19 + Fill-up |
| CanFlash | app\_crc\_customer.s19 | Application Binary that will be used for SWDL |
|  | Default\_customer.s19 | Default E2P Binary that will be used for SWDL |
|  | Unified\_Calibration\_customer.s19 | Calibration Binary that will be used for SWDL |
| DIA | DIA\_ReadDataByIdentifier.c | Source file used at build time, used to check compiled Software Version |
| NVP | Nvp\_Generated.c | Source file used at build time, used to check NVP Layout |
|  | Nvp\_Generated.h | Source file used at build time, used to check NVP Layout |
|  | SBE\_4G\_NVP\_layout.xls | Generator file used at build time |
| Production | PP4GE000.010 | FullFlash.s19 |
| Release\_log | Application\_build\_report.pdf | Report containing all files compiled, including compiler options, defines |
| SAP | UNI\_SBE\_PP4G\_R010E.SAP | SAP File containing FullFlash.s19 |

Table 3 Extended Platform output file list

# Integration and Configuration

## 5.1. Configuration options

Please find below all the possible configuration options and their effects:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Configuration Option | Location | Application | | | Bootloader | Comment |
| - | - | Extended | Mainstream | Mainstream | |  |
| PROJECT\_STEM | Makefile.config.mk | PP4G\_EXTENDED | PP4G\_MAINSTREAM | PP4G\_MAINSTREAM | | Use to compose PROJECT\_NAME |
| PROJECT\_NAME | $(PROJECT\_STEM)\_APPLICATION | $(PROJECT\_STEM)\_APPLICATION | $(PROJECT\_STEM)\_FBL | | PROJECT\_NAME used for eclipse project/ report generation |
| GENY\_GENERATION | FALSE | FALSE | TRUE | | Switch used for automatic generation with GENy |
| PARSE\_LINKER\_FILE | FALSE | FALSE | TRUE | | Add object files automatically to the linker file |
| BOLO\_EEP\_VALIDATION | FALSE | TRUE | FALSE | | Add validation header in FullFlash file |
| PRINT\_WARNINGS | FALSE | FALSE | FALSE | | Switch used for compiler warning printing in eclipse console(N/A for extended) |
| S19\_CRC\_CALCULATION | TRUE | TRUE |  | | Switch used for for app\_crc.s19 generation |
| AEC\_CALIBRATION\_SUPPORT | FALSE | TRUE |  | | Mainstream specific file |
| DYNAMIC\_E2P\_SUPPORT | FALSE | TRUE |  | | Mainstream specific file |
| FAB\_FILE\_SUPPORT | FALSE | TRUE |  | | Enable or disable FAB file processing |
| NVM\_CHECKSUM\_SAP\_FILE\_GEN | FALSE | TRUE |  | | Create NVP Corrupt SAP Image for SWV Testing |
| NVM\_CHECKSUM\_SAP\_FILE\_START\_ADDR |  | 0x10037C |  | | Start Addr for NVP Corruption |
| NVM\_CHECKSUM\_SAP\_FILE\_END\_ADDR |  | 0x1005D5 |  | | End Addr for NVP Corruption |
| BOOTLOADER\_SUPPORT | FALSE | TRUE |  | | Define Bootloader support |
| BOOTMANAGER\_SUPPORT | TRUE | FALSE |  | | Define Bootmanager support |
| EEP\_VALIDATION\_START\_ADDR |  | 0x100000 |  | | Address used with BOLO\_EEP\_VALIDATION switch |
| EEP\_VALIDATION\_END\_ADDR |  | 0x100010 |  | | Address used with BOLO\_EEP\_VALIDATION switch |
| EEP\_VALIDATION\_FLAG |  | 0x00 0xFF 0xFF 0xFF 0x00 0xFF 0xFF 0xFF 0xF0 0xFF 0xFF 0xFF 0xD1 0xFF 0xFF 0xFF |  | | Pattern that will be places between EEP\_VALIDATION\_START\_ADDR and EEP\_VALIDATION\_END\_ADDR |
| LINKER\_FILE\_STEM | err | err | PP4G\_Fbl | | Used to identify the linker file |
| CYCLONE\_UNIV\_NVP\_SAP\_GENERATION | FALSE | TRUE |  | | Create SAP files with EEPROM |
| CYCLONE\_UNIV\_FLASH\_ALGORITHM | $(CYCLONE\_UNIV\_INSTALATION\_DIR)/cyclone\_universal/supportFiles\_ARM/NXP/S32K1xx/freescale\_s32k144f512m150N77P\_all.arp | $(CYCLONE\_UNIV\_INSTALATION\_DIR)/cyclone\_universal/supportFiles\_HC12\_HCS12/freescale\_9s12zvc192\_1x16x96k\_all.s12zp |  | | Used for SAP file generation |
| ALV\_CRC\_SUPPORTED | FALSE | TRUE |  | | Mainstream specific CRC algorithm |
| HEXVIEW\_CRC\_SUPPORTED | TRUE | FALSE |  | | Extended specific CRC algorithm |
| APPL\_CRC\_ADDR | 0x7FFFC | 0xFF6DE0 |  | | Address where CRC shall be places |
| APPL\_CRC\_RANGES | 0x0-0x7FFFB/0x1000-0x2FFF | 0xFD0000-0xFF6DE0,0xFF6DE4-0xFF7000 |  | | Address range over which CRC shall be calculated |
| APPL\_EXCLUSION\_RANGE | 0x1FFF9000-0x20002FFF |  |  | | Address space that will be removed from binary file |
| APPL\_RANGE | 0x20000 0x7FFFC | 0xFD0000 0xFF7000 |  | | Application address space that will be filled up |
| CALIBRATION\_RANGE | 0x22000 0x23000 | 0xFF7000 0xFF7800 |  | | Calibration address space that will be filled up |
| DEFAULT\_NVP\_RANGE | 0x21000 0x22000 | 0xFF7800 0xFF8000 |  | | Default\_E2P address space that will be filled up |
| BOOTMANAGER\_RANGE |  |  |  | | Bootmanager address space that will be filled up |
| AEC\_CALIBRATION\_FILE |  | $(OUTPUT\_DIR)/aec\_calibration/AEC\_Calibration.S00 |  | | Mainstream specific file generated by NVP |
| CALIBRATION\_FLASH\_FILE | $(OUTPUT\_DIR)/nvp/CalibrationFlash.S00 | $(OUTPUT\_DIR)/nvp/CalibrationFlash.S00 |  | | File generated by NVP |
| DEFAULT\_FILE | $(OUTPUT\_DIR)/nvp/Default.S00 | $(OUTPUT\_DIR)/nvp/Default.S00 |  | | File generated by NVP |
| DYNAMIC\_E2P\_FILE |  | $(OUTPUT\_DIR)/nvp/DynamicE2P.S00 |  | | Mainstream specific file generated by NVP |
| E2P\_STUDY\_FILE | $(OUTPUT\_DIR)/nvp/E2pDump.S00 | $(OUTPUT\_DIR)/nvp/E2P\_Study.S00 |  | | File generated by NVP |
| E2P\_SYMBOL\_FILE | $(OUTPUT\_DIR)/nvp/EepSymbol.sym | $(OUTPUT\_DIR)/nvp/EepSymbol.sym |  | | File generated by NVP |
| FAB\_FILE |  | $(OUTPUT\_DIR)/nvp/Fab\_\_\_\_\_.F00 |  | | Mainstream specific file generated by NVP |
| FULL\_MEMORY\_FILE | $(OUT\_DIR)/FullFlash.$(SREC\_SUFFIX) | $(OUT\_DIR)/FullFlash.$(SREC\_SUFFIX) |  | | Flash file containing all configured blocks(Application, Calibration, Default , Bootloader, etc.) |
| UNIFIED\_CALIBRATION\_FILE | $(OUT\_DIR)/Unified\_Calibration.$(SREC\_SUFFIX) | $(OUT\_DIR)/Unified\_Calibration.$(SREC\_SUFFIX) |  | | Mainstream specific file, is build from AEC\_Calibration and Calibration\_Flash |
| BOOTLOADER\_FILE |  | $(RELEASE\_DIR)/../Bootloader/fbl.s19 |  | | Bootloader flash file |
| BOOTMANAGER\_FILE | $(TOOLS\_DIR)/BootManager/Workspace/Debug/Exe/BootManager.s19 |  |  | | Bootmanager flash file |
| ELF\_FILE\_GENERATION | Makefile.compiler.options.mk | FALSE | TRUE | TRUE | | Switch to enable generation of elf file |
| PREPROCESS\_AS\_SUPPORT | TRUE | FALSE | FALSE | | Switch to enable pre-processing of assembler files |
| GENERATE\_LINKER\_CMD | TRUE | FALSE | FALSE | | Extended specific linker command |
| APPL\_VERSION\_REV1 | Makefile.wers.mk | XXXXXXXX.XXX | XXXXXXXX.XXX |  | | Original SW Version |
| APPL\_VERSION\_REV2 | PP4GE000.010 | PP4GM000.034 |  | | New SW Version |
| APPL\_FAB\_VERSION | PP4GE000.F10 | PP4GM000.F34 |  | | Fab File Version |
| SAP\_NAME | SBE\_PP4G\_R010E | SBE\_PP4G\_R034M |  | | SAP File Name |

Table 4 Configuration Option Summary

## 5.2. Adding new Modules

In order to add new files to the compilation process you must do the following:

1. Add the component to the component list if it does not exist in Makefile.config.mk:

**COMPONENTS\_LIST :**= BSW

1. Create the two variables that control the sources and include directory

**BSW\_SOURCES\_DIR** = $(SOURCES\_DIR)/$(1)/Implementation/src

**BSW\_INCLUDES\_DIR** = $(SOURCES\_DIR)/$(1)/Implementation/inc

1. Add the component list (each component will be placed instead of $(1) in the variables above)

**BSW\_COMPONENTS\_LIST :**= PMP NVP ATM BFE BFS BMM BPA BSR CIL DIA EOL ERH ESM HWA LSM MIC MMG PAL PCM PRE PRO PWM RCM SCM SFR STM TIM WDG SBC NvmIf SpiIf PwmIf AdcIf TL\_Lib PortIf VDA

1. Add the necessary compiler options/assembler options in the Makefile.compiler.option.mk file:

**COMP\_OPT\_BSW :**= -o $(OBJ\_DIR)/ \

--no\_wrap\_diagnostics \

-e \

--cpu Cortex-M4 \

--fpu None \

--debug \

--dlib\_config '**$(subst** \,/,$(COMPILER\_INC)/DLib\_Config\_Normal.h**)**' \

--endian little \

--cpu\_mode thumb \

-Ol \

--no\_cse \

--no\_unroll \

--no\_inline \

--no\_code\_motion \

--no\_tbaa \

--no\_clustering \

--no\_scheduling \

**$(addprefix** -D,$(COMP\_DEFINES)**)** \

**$(addprefix** -I,$(COMP\_INCLUDES)**)**

# Walkthrough

## 6.1. First use

1. Navigate to the ***s:\Scripts\VirtualDrive\core\*** directory and run the following scripts:
   1. ***Mount\_S\_Drive.bat*** (optional)
   2. ***RemoveReadOnly.bat***
2. Open the ***env\_cfg.bat*** file and make sure the following configuration(tool selection) is correct in regards to the project context.
   1. Specify the installation directory of the compiler:

set COMPILER\_DIR=c:\Program Files (x86)\IAR Systems\Embedded Workbench 8.0\arm\bin\

*OR*

set COMPILER\_DIR= c:\COSMIC\CXS12Z\_438

* 1. Specify the number of CPU threads that will be used by make(leave blank for maximum available):

set THREAD=

* 1. Specify what tool versions will be used:

set SRECORD\_BASE=srecord-1.63-win32

set ECLIPSE\_BASE=eclipse-cpp-2019-12

set CYCLONE\_BASE=cyclone\_universal

set HEXVIEW\_BASE=HexView-1.12.02

set PYTHON\_BASE=python-3.7.1

set VERA\_BASE=vera++

set ZIP\_INSTALLER\_BASE=zip\_installer-1.0.1-py3-none-any

set BUILD\_ENV\_BASE=build\_assist-1.0.1-py3-none-any

* 1. Specify the location where you wish to install the necessary tools.

set "SW\_DEV\_TOOLS\_DIR=c:\Prog"

1. Open a cmd window in the ***s:\Tools\Build\_Env\Workspace\Build\*** directory.
2. Run the ***setup\_env.bat*** command.

*Note:* This might take [15-30s] since all the necessary tools will be installed to the configured directory.

1. Run the ***make all-eclipse*** command in order to generate the eclipse workspace. Please see ***Table 1 make commands*** for more details.
2. After issuing the make all-eclipse command, the cmd window can be closed, and the compilation process can continue from eclipse.

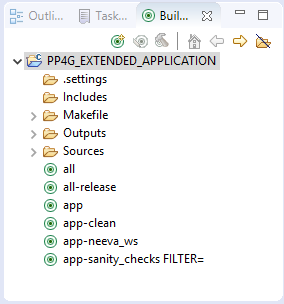
******

Figure 7 Eclipse Build Procedure

## 6.2. Normal workflow

1. Navigate to the ***s:\Scripts\VirtualDrive\core\*** directory and run the following scripts:
   1. ***Mount\_S\_Drive.bat*** (optional)
   2. ***RemoveReadOnly.bat***
2. Navigate to the ***s:\Tools\Build\_Env\Workspace\Build\***  directory.
3. Launch the ***eclipse.bat*** file in order to start the eclipse environment.

***OR***

1. Launch the ***build\_window.lnk*** in order to start the cmd environment.

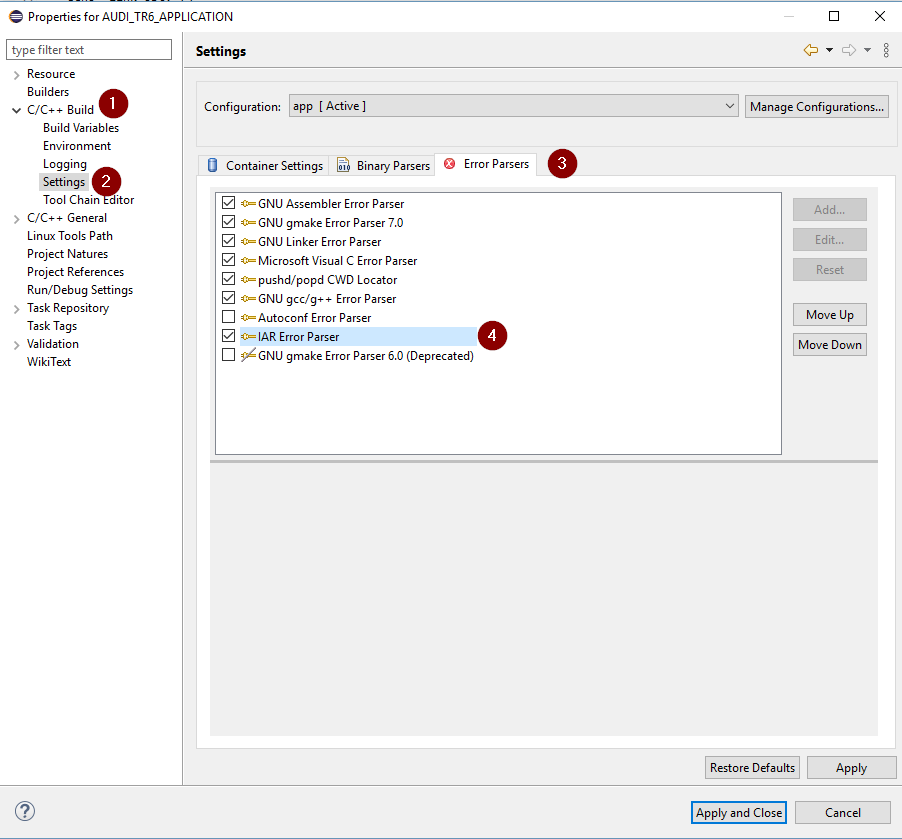
*Note 1:* When adding new modules to the Application the eclipse project must be regenerated.

*Note 2:* The eclipse workspace is shared between the Application/Bootmanager/Bootloader software components. By issuing ***make all-eclipse***, the build environment will generate the eclipse project for all the available software components.

# FAQ

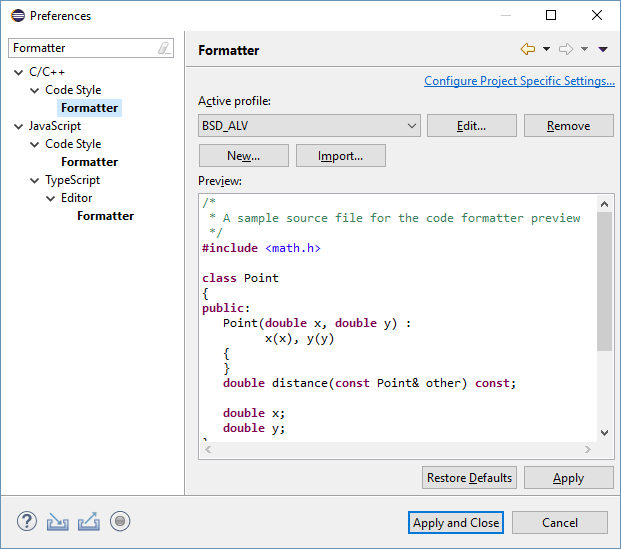
1. Eclipse does not index compiler warnings for IAR

* *Right click on project properties for \*\*\*\_APPLICATION*

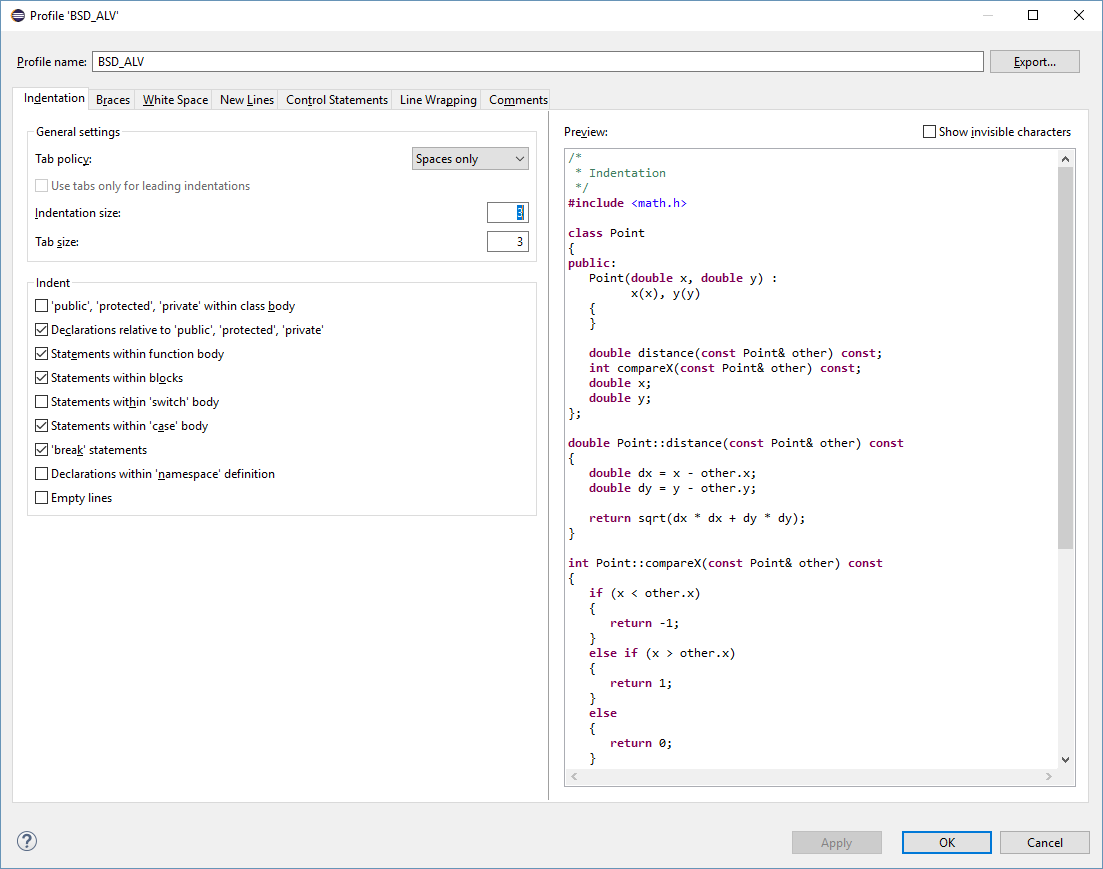


1. Tabs and spaces are not respected

* *Windows -> Preferences -> Code Formatter -> Edit BSD\_ALV*

**

* *Change indentation size to 4, click Apply*
* *Change indentation size back to 3, click Apply and Ok*

**

* *Click Apply & Close on the Code Formatter window*

1. Fatal error when launching pip

If you encounter the following error:

***Fatal error in launcher: Unable to create process using '"c:\\_devapp\python71\python.exe"  "c:\My\_prog\python-3.7.1\Scripts\pip.exe" uninstall -y build\_assist -q -q'***

* Please navigate to C:/Prog/python-3.7.1 and open a cmd window
* Run the following command:

python -m pip install --upgrade pip