**Automotive and Discrete Group**
**Automotive Digital Division**

Infotainment Business Unit
Teseo SDK user manual

# 1 Introduction

The Standard Development Kit (SDK) for Teseo products provides a pool of libraries and sources to start developing custom applications based on ST GNSS library. Besides, it provides an IDE configuration to simplify developing.

This document describes what is delivered with SDK and how to use it inside IDE. Information available here applies to these products in Teseo family:

- Teseo2 (STA8088 and all variants);
- Teseo3 (STA8089/8090 and all variants).

All provided descriptions refer to Teseo3 family, but they are valid also for Teseo2 family. Regarding references to file or folder names, `sta8090` keyword refers to Teseo3 family. For Teseo2 family, the same file or folder can be found using `sta8088` keyword.

Any difference is highlighted.

# 2 Contents

## 2.1 Index

## 2.2 List of Tables

## 2.3 List of Figures

# 3 Document Management

## 3.1 Revision History

| Rev | Date | Author | Notes |
|---|---|---|---|
| 1.0 | 2014/06/05 | F. Boggia | First release |
| 1.1 | 2015/04/28 | F. Boggia | Updated to SDK rev 2.1.1 |
| 1.2 | 2015/06/16 | V. Delaunay | Add probe plug-in |
| 1.3 | 2015/12/03 | V. Delaunay | Add memory setup menu |
| 1.4 | 2015/12/07 | S. Chambrillon | Minor updates to improve understanding. |
| 1.5 | 2015/12/14 | F. Boggia | Updated to SDK rev 2.1.5 |
| 1.6 | 2015/12/17 | S. Chambrillon | Creation of Appendix B: RAM and CPU status |
| 1.7 | 2016/02/09 | PY. Quemeneur | Add 32KB option in Appendix B |
| 2.0 | 2016/03/15 | F. Boggia<br>S. Chambrillon | Updated to SDK rev 2.2.0<br>Several reworks/updates in all the document. |
| 2.1 | 2016/06/13 | S. Chambrillon | Update of Appendix B<br>Added Appendix C |
| 3.0 | 2016/07/22 | F. Boggia<br>S. Chambrillon | Updated to SDK rev 2.2.1 |
| 3.1 |  | F. Boggia<br>S. Chambrillon | Updated to SDK rev |

**Table 1: Revision history**

## 3.2 Acronyms

| Keyword | Definition |
|---|---|
| BSP | Board Support Package |
| GAE | GNU tools for Arm Embedded processors |
| GCC | GNU C Compiler |
| GNSS | Global Navigation Satellite System |
| GPS | Global Positioning System |
| IDE | Integrated Development Environment |
| LLD | Low Level Drivers |

| | |
|---|---|
| NVM | Non Volatile Memory |
| QZSS | Quasi-Zenith Satellite System |
| RVCT | Real View Compiler Toolchain (from ARM) |
| RTOS | Real Time Operative System |
| SBAS | Satellite Based Augmentation System |
| SDK | Software Development Toolkit |

**Table 2. Acronyms**

# 4 SDK content

## 4.1 Software delivery

Here is the software content which is provided with SDK for Teseo products:

- GNSS library for multi-constellation positioning (GPS/Glonass/Galileo/Beidou[1] QZSS/SBAS);
- ST OS20+ RTOS library;
- FreeRTOS library;
- LLD drivers library;
- Services to integrate drivers into gpOS (see Section 5.2 for more details about gpOS);
- Eclipse IDE projects for different configurations.

## 4.2 Tools

### 4.2.1 External tools needed

SDK for Teseo products is built using:

- ARM RVCT 5.04 update 2 (available standalone and from ARM DS-5 5.19);
- GCC for ARM embedded 4.7 2014q2 (GCC 4.7.4);

ARM libraries are built with backward compatibility to RVCT 3.1, 4.0 and 4.1. GCC libraries are compatible with Sourcery Codebench Lite.

SDK also provides projects to be used in Eclipse IDE. This tool is available from:

http://www.eclipse.org

The use of this IDE is not mandatory. Customer is free to build up projects in its preferred IDE or developing environment.

For debugging, SDK provides:

- Scripts for Lauterbach Trace32 software. Information can be obtained from:

  http://www.lauterbach.com/

- Scripts for Segger Ozone. Information can be obtained from:

  https://www.segger.com/ozone.html

[1] Beidou constellation is available only for Teseo3 family.

In next section we will describe how to setup and configure Eclipse to be used with SDK for Teseo products. In section 5 we will describe the SDK architecture.

*Note:* *Installation of RVCT compiler, GCC compiler, Trace32 software and Segger software is not described here.*

### 4.2.2 Provided tools

This section summarizes all tools available to support the Teseo products SDK users.

- **FWConfig**

Allows to modify generated binary image configuration before downloading it into device's flash.

- **Firmware Upgrade**

Firmware upgrade tool (in-field GNSS firmware update).

- **X-Loader**

Firmware flashing tool (production line flashing)

- **Teseo-Suite setup**

Allows to install the Teseo Suite tool. The Teseo Suite tool is a 32-bits software which provides a powerful tool for evaluation, performance analysis and configuration of ST GNSS receivers.

## 4.3 Documentation

This section summarizes all documents available to support the Teseo products SDK users.

### 4.3.1 Documents related to SDK

- **Release Note**

Describes SDK content.

- **Teseo products SDK usage**

Current document.

- **OS20+ Specification**

OS20+ real-time kernel description.

- **FreeRTOS specification**

Describes all information for FreeRTOS usage with ST GNSS Positioning SW including implementation, compilation and debugging.

- **GNSSLib API Specification**

Provides all APIs available in ST GNSS library for all ST GNSS microcontrollers.

### 4.3.2 Tools related documentation

- **Firmware Configuration**

Provides details about each supported parameter including procedures for changing and saving the firmware configuration.

- **Firmware Upgrade user manual**

Describes how to use the firmware upgrade tool.

- **XLoader user manual**

Describes the usage of TeseoIII XLoader tool.

- **Teseo-Suite user manual**

Contains the information necessary for correct use of the Teseo-Suite tool and describes all its functionality.

- **GNSS Performance Report**

### 4.3.3 Application notes

- **SBAS Automatic Search Algorithm Description**

Overview of the algorithm implemented for the Automatic Search of the SBAS satellite.

- **Antenna Detection Application Note**

Description of antenna detection mechanism via hardware schematics and description of the firmware running on Teseo chipset.

- **Adaptive low power management**

Application note to select and configure properly low power modes.

- **ST GNSS NMEA specification and commands**

Provides an overview of the various NMEA commands and messages for the STMicroelectronics´ GNSS Systems.

- **FreeRTOS PVT application example**

Provides PVT example using FreeRTOS to help the first steps in application design.

- **Low Power applications examples (only Teseo3)**

Provides low power examples to help the first steps in application design.

- **NMEA over CAN (only Teseo3)**

Provides an example on how setup an application to send GNSS information over CAN bus.

### 4.3.4 DRAW specific documentation

*Note: Only provided with DRAW plug-in delivery*

- **DRAW Firmware Configuration**

Provide details about specific Dead Reckoning parameters including procedures for changing and saving the firmware configuration.

- **DRAW NMEA Interface**

Provide the DR extension to the standard NMEA messages for the STMicroelectronics´ GNSS Systems.

- **ST DRAW Installation and Calibration**

Provides an overview on the steps needed to manage ST DRAW™ installation and calibration.

- **ST DRAW Installation Parameters**

Contains an explanation on the installation parameters and some examples.

- **ST DR API Specifications**

Provides all available DR APIs.

- **ST DRAW Map Matching Feedback**

Explains the way to feed TESEO DRAW™ with map data fed back by customer application aiming to increase position accuracy during GNSS outages.

- **ST DRAW Access Raw Sensor Data**

Describe how activating and using the “Access Raw Sensor Data” feature with the Teseo products SDK DR.

- **ST DRAW Sensors over UART**

Describe the way to feed Teseo DRAW sensor module using properly formatted strings sent through the UART port.

# 5 SDK software architecture

The SDK for Teseo products is organized as showed in Figure 1.

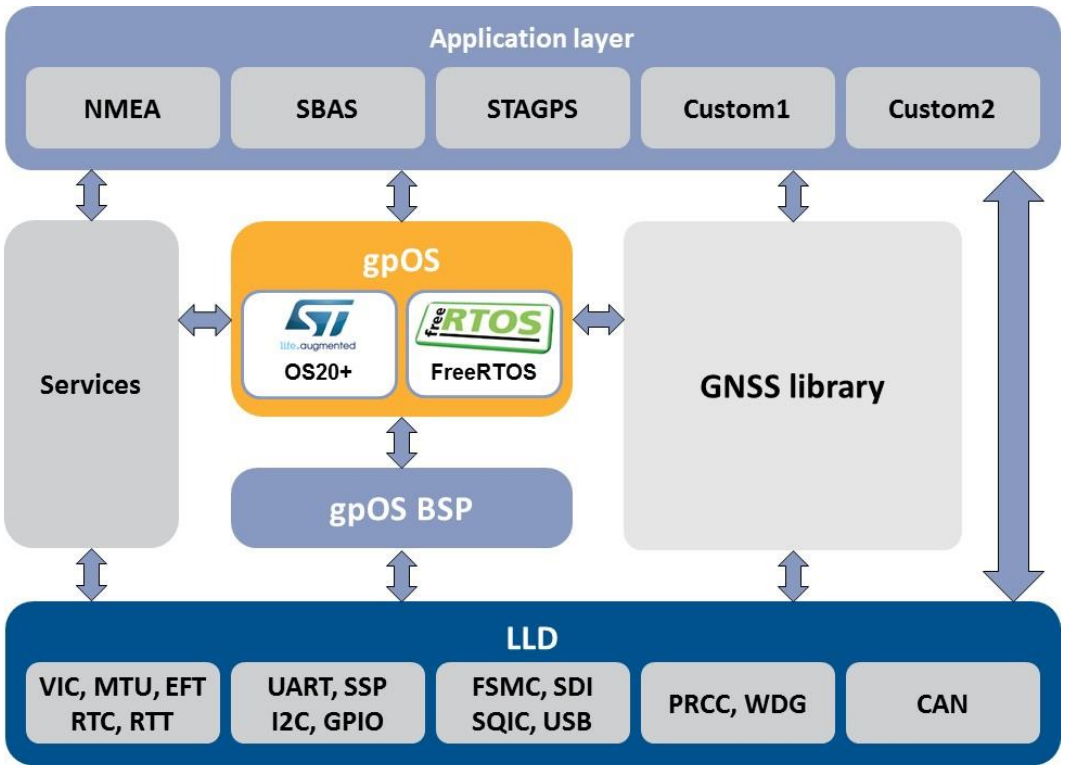


**Figure 1: SDK software architecture**

All represented blocks are described in details in the following sections.

## 5.1 LLD layer

The LLD (Low Level Drivers) layer is a collection of drivers to interface Teseo products peripherals. These drivers are very basic and provides a simple access to devices features. The common characteristics of all drivers are:

- Zero RAM memory occupation: all drivers do not allocate any random access memory. If memory is needed to the driver, it will be allocated by the application when the driver is used.
- Monolithic structure: each driver only implements specific device features, never referencing to other drivers. This means that interfacing between the drivers must be realized at application level. For example, to implement a module that uses an UART, it must use UART driver for UART specific APIs, VIC driver for interrupts handling and GPIO driver for alternate functions programming if needed.
- OS independency: drivers do not refer to any operative system at all.

## 5.2 gpOS layer

gpOS stands for Generic Positioning Operating System. It is an operating system abstraction layer on which GNSS library and modules were developed. This layer can be ported to different operating systems. SDK provides two versions:

- a proprietary operating system (OS20+);
- a FreeRTOS adaptation layer.

gpOS interfaces the Teseo products through a specific layer named BSP (Board Support Package) that implements the API needed by gpOS for timing and interrupt controlling using LLD layer. This ensures portability of gpOS to different architectures.

## 5.3 Services

This module implements a set of services for some peripherals. Currently SDK provides services for:

- ADC
- CAN
- FSMC
- FSW (only Teseo3)
- GPIO
- I2C
- MCU
- MSP (SPI master protocol only)
- MTU
- PWR (only Teseo3)
- SDI
- SQI
- SSP (SPI master protocol only)
- UART
- USB (VCOM protocol only)

These services are not full. These modules are provided as sources to customers to be customized to their needs.

*Note: FSMC and SQI services are used by GNSS library for specific tasks. Sources are provided to let customers modify them to fulfil to their needs, but care must be taken to not break their usage in GNSS library.*

## 5.4 GNSS library layer

This library implements all API for GNSS multi-constellation positioning.

## 5.5 Application layer

This layer implements some optional standard ST modules (NMEA, SBAS, STAGNSS …) plus customers' specific modules.

### 5.5.1 ST modules

SDK for Teseo products provides a list of modules, some mandatory, some optional, that can be included or not in a customer application:

- NVM (mandatory): this module implements non-volatile data handling for GNSS library. ST provides libraries to use RAM, NOR or SQI to save data. Customer can use this module as well to handle its own specific non-volatile data.
- SBAS (optional): this module implements SBAS receiving and decoding for GNSS library.
- RTCM (optional): this module implements RTCM receiving and decoding for GNSS library.
- ST binary protocol (optional): this module implements ST binary protocol.
- NMEA (optional): this module implements standard NMEA sentences production. It also provides proprietary commands that can be sent on NMEA port to perform specific actions.
- FATFS (optional): this module is an open source hardware independent implementation of FAT/FAT32 file systems for embedded systems. For more info and usage manual, please refer to http://elm-chan.org/fsw/ff/00index_e.html.
- SDLOG (optional): this module implements logging of debug and NMEA information on a SD card connected to Teseo products.
- STAGNSS (optional): this module implements best-in-class self-assisted GNSS algorithm.

Optional modules can be enabled in project by using `PROJ_OPT_FEATURES` variable in a target configuration makefile as described in Section 6.4.3 and Table 15.

*Note: By default, STAGNSS is not used even if its compilation is enabled. To enable it, FWconfig tool must be used on swconfig library (either ARM or GCC library).*

### 5.5.2 Custom modules

The user can build its own application over the explained system. The number of custom modules is only limited by memory available. In principle, there is no limitation to number of OS services (tasks, semaphores, message queues...) that the user can access.

As drivers layer (LLD) is not OS controlled, the critical resource access for a specific peripheral must be implemented by the user. An example to do this is provided in OS services sources.

## 5.6 Memory organization

The Figure 2 shows how memory is organized in typical SDK targets.

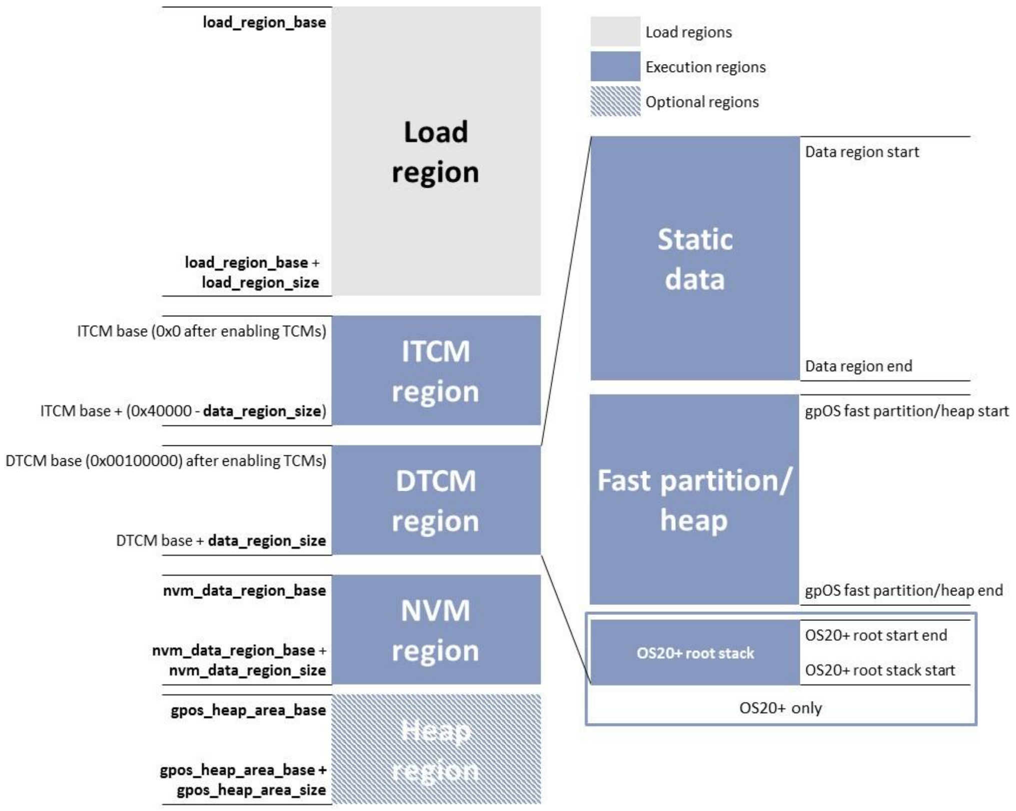

load_region_base
Load region
load_region_base + load_region_size
ITCM base (0x0 after enabling TCMs)
ITCM region
ITCM base + (0x40000 - data_region_size)
DTCM base (0x00100000) after enabling TCMs)
DTCM region
DTCM base + data_region_size
nvm_data_region_base
NVM region
nvm_data_region_base + nvm_data_region_size
gpos_heap_area_base
Heap region
gpos_heap_area_base + gpos_heap_area_size
Load regions
Execution regions
Optional regions
Static data
Data region start
Data region end
Fast partition/ heap
gpOS fast partition/heap start
gpOS fast partition/heap end
OS20+ root stack
OS20+ root start end
OS20+ root stack start
OS20+ only

**Figure 2: Memory map of SDK software**

The memory configuration is implemented in the scatter file used in SDK. The ARM scatter file and GCC map file have the capability to be pre-processed by C compiler before being used as memory region definition file by linker. So, some of the regions are defined by some parameters (in bold) that are passed by Eclipse. These parameters are described in Section 6.4.2.

The **Load region** is the region were the binary image created by linker will be stored. The image will contain:

- Code to be executed in place;
- Code and data to be relocated in another region;
- Constant data.

All regions in blue are **execution regions**. They are created at runtime by GNSS application image:

- **ITCM region** contains code and data needed at runtime, like exception vectors.
- **DTCM region** contains data needed at runtime. It can be divided in three subregions:
  - **Static data**: data statically allocated by software
  - **Fast partition/heap**: data not statically allocated. This will be used as fast partition if optional heap is defined, or as heap if no external heap is configured.
  - **OS20+ root stack** (for OS20+ projects only): region used for root task stack.
- **NVM region** is used to store data relevant for GNSS application. It can be allocated in either RAM or Flash memory. Its size depends on GNSS features enabled in specific configuration. See Section 6.4.2 for details.
- **Heap region** (optional) is a region used as C library heap. Usually it is allocated in external RAM, if available. If there is no memory that can be used as heap, the size of this region must be set to zero. In this way, part of DTCM region will be used as heap instead of fast partition.

Relocation of code is done by the software. ARM toolchains provide relocation mechanism handled by linker and libraries, configured using scatter file. Instead, GCC toolchains do not have any automated relocation algorithm and it must implemented manually. SDK provides assembly files for GCC to relocate regions needed by GNSS application.

Note: *If another region must be defined by custom application using GCC toolchain, it must be relocated by modifying properly `crt0_FR.s` in `sta8090\gpsapp\gnu` folder*

Relocation is essential also when flash operation must occurs on load region memory. During a write or erase operation performed on flash used to store load region, no fetch (code or data) can be performed. For this reason, execution regions are used to relocated specific portion of code like:

- Flash algorithms needed to perform write and erase operations
- Exception vectors (like interrupts and system calls) and all code and data needed to handle them.

Note: *As execution regions are relocated during startup, code and data that will be relocated MUST not be accessed before relocation has been executed.*

ITCM and DTCM regions are allocated in ITCM and DTCM of ARM946 core, so they are accessed at fastest speed. For this reason, code and/or data can be relocated in those regions to improve their performance.

To improve CPU usage for GNSS library, some code and data are moved in ITCM and DTCM and some other can be moved as well. A selection of modules is specified in scatter and map files. They can be moved by specifying them in `PROJ_FAST_ENABLE` option in a target configuration makefile, as described in Section 6.4.4. Available flags are described in Table 17.

*Note: If the heap size is not enough for customer application, a new heap region located for example in an external RAM can be created and used in gpOS using the `<service>_create_p` APIs of gpOS.*

## 5.7 Image formats

Teseo products SDK offers the capability to build different kind of images. After linking, the image will be converted to a pure binary file. So there are different ways the image can be loaded to the target:

- Using debugger: the executable produced by the linker can be easily loaded to the target using a debugger.
- Using XLoader protocol: a plain image can be flashed directly to the target by a proper sequence of boot pins setup and reset.
- Using FW upgrade protocol: an upgrade image can be flashed

The default of SDK projects is to produce executable and plain binary images to be used with XLoader. This behaviour can be changed by enabling FW upgrade option in projects as explained in Section 0.

When this option is enabled, the build system will create usual executable together with a couple of binaries:

- Binary with _BOOT extension: this binary can be flashed with XLoader protocol. It contains the bootloader and the application image.
- Binary with _UPG extension: this binary can be used with FW upgrade protocol to update an existing image.

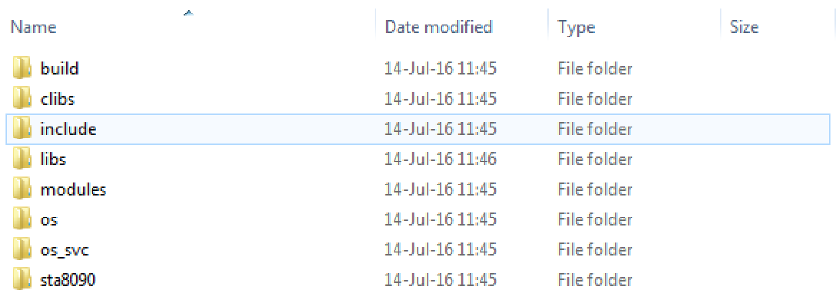
*Note: FW upgrade protocol is not available for targets which load region is set to RAM.*

# 6 Using SDK in Eclipse

## 6.1 Preparing SDK

Extract Teseo products SDK in a folder.

*Note:* ***The extraction folder for SDK must not contain spaces!***



| Name | Date modified | Type | Size |
|---|---|---|---|
| build | 14-Jul-16 11:45 | File folder | |
| clibs | 14-Jul-16 11:45 | File folder | |
| include | 14-Jul-16 11:45 | File folder | |
| libs | 14-Jul-16 11:46 | File folder | |
| modules | 14-Jul-16 11:45 | File folder | |
| os | 14-Jul-16 11:45 | File folder | |
| os_svc | 14-Jul-16 11:45 | File folder | |
| sta8090 | 14-Jul-16 11:45 | File folder | |

**Figure 3: Contents of SDK package.**

Let's name extraction folder `<sdk_folder>`.

## 6.2 Eclipse setup

SDK projects based on Eclipse are using the following packages that must be downloaded from Internet:

| **Name of package** | **Link** |
|---|---|
| Java Runtime Edition | http://java.com/en/download/manual.jsp |
| Eclipse IDE for C/C++ Developers | http://www.eclipse.org/downloads/packages/eclipse-ide-cc-developers/neon1a |

**Table 3: Packages needed for SDK usage.**

*Note:* *32-bit and 64-bit combination of Java and Eclipse can be used.*

Here are the installation steps:

- Install Java Runtime Edition using its setup;
- Extract Eclipse package in any folder;
- Execute `eclipse\eclipse.exe` from extraction folder;
- Select Eclipse default workspace and continue with program starting.

*Note:* *This procedure was tested on Windows Windows7 64-bit using Java RE 1.8u111 and Eclipse Neon 1a, both 32-bit and 64-bit.*

Figure 4 shows an open Eclipse empty workspace.

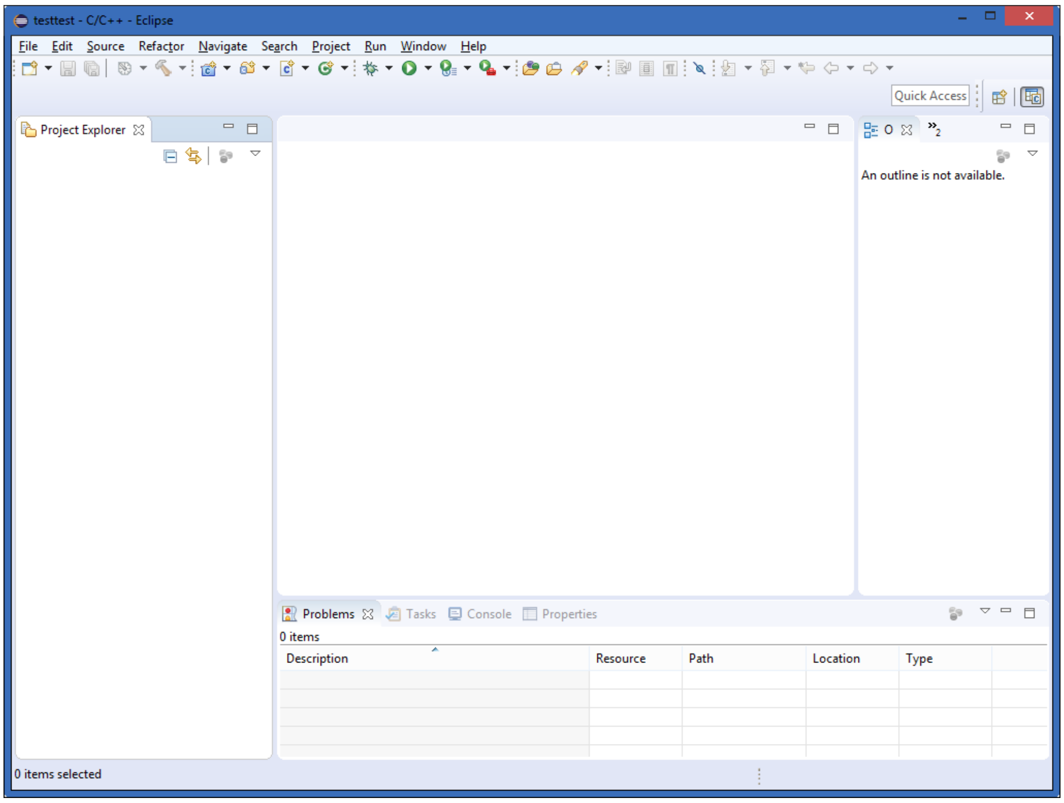

testtest - C/C++ - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Project Explorer
An outline is not available.
Problems Tasks Console Properties
0 items
Description Resource Path Location Type
0 items selected

Figure 4: Eclipse default window.

## 6.3 Opening a project

Teseo products SDK is provided with a set of preconfigured projects that can be used. Table 4 resumes the list of the available projects and their features:

| Project name | Features |
|---|---|
| binary | This project can be used to build an exact copy of the binaries distributed with Teseo products binary package. |
| gnssapp_freertos_gae | This project can be used to develop FreeRTOS applications based on GNSS libraries using GCC for ARM embedded compiler |
| gnssapp_freertos_rvct | This project can be used to develop FreeRTOS applications based on GNSS libraries using ARM RVCT compiler. |
| gnssapp_os20_rvct | This project can be used to develop OS20+ applications based on GNSS libraries using ARM RVCT compiler. |

Table 4: Available projects.

When DRAW plugin is installed, more projects will be available as showed in Table 5.

| Project name | Features |
|---|---|
| binary_dr | This project can be used to build an exact copy of the DRAW binaries distributed with Teseo products DRAW binary package. |
| gnssapp_dr_freertos_gae | This project can be used to develop FreeRTOS applications based on DRAW and GNSS libraries using GCC for ARM embedded compiler |
| gnssapp_dr_os20_rvct | This project can be used to develop OS20+ applications based on DRAW and GNSS libraries using ARM RVCT compiler. |

**Table 5: More projects when DRAW plugin is installed.**

Most configurable option are shared between all projects. There is a subset of options related to compiler used in a project that differs between ARM RVCT and GCC for ARM embedded based projects. They will be shown in Section 6.4.

To open a project, restart Eclipse and set the folder `<sdk_folder>\build\eclipse\prj` as workspace, as shown in Figure 5:

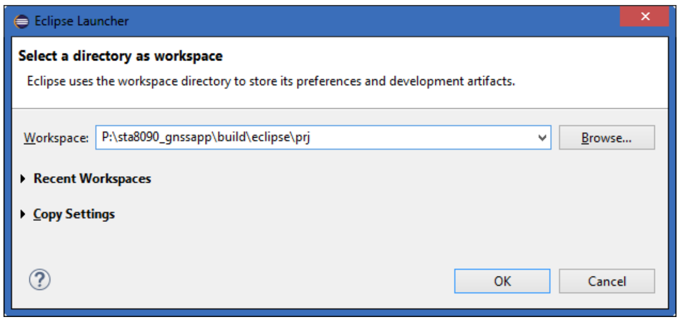
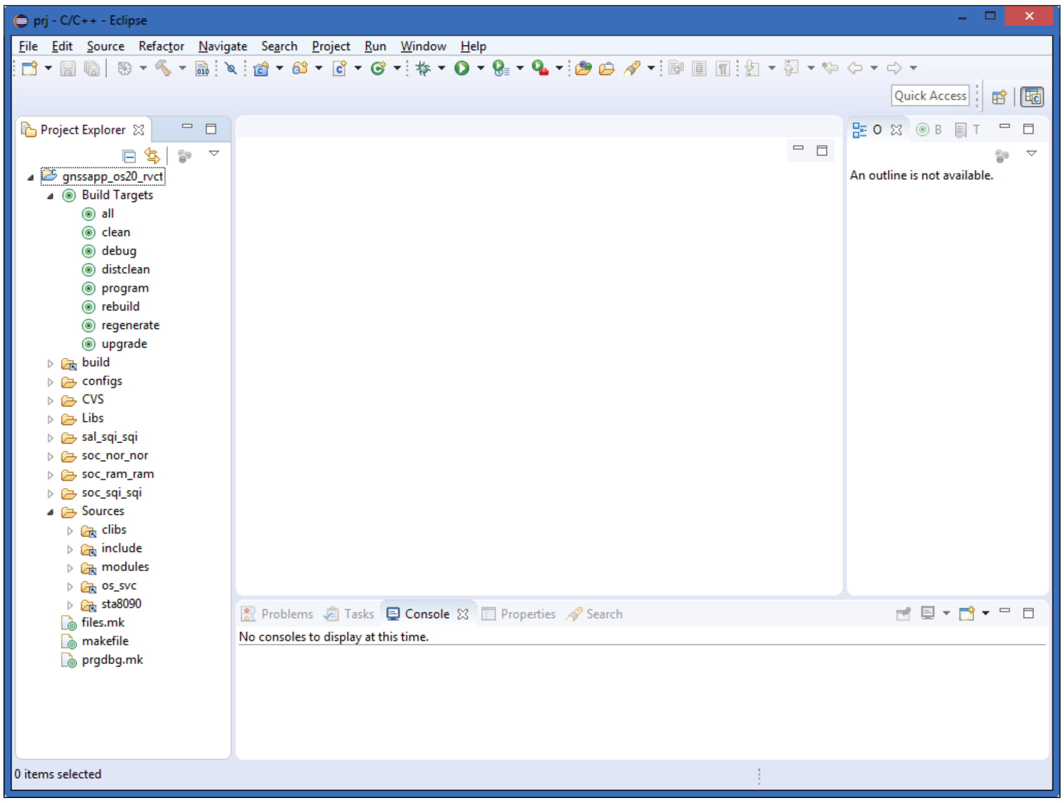

Eclipse Launcher
Select a directory as workspace
Eclipse uses the workspace directory to store its preferences and development artifacts.
Workspace: P:\sta8090_gnssapp\build\eclipse\prj
Browse...
Recent Workspaces
Copy Settings
OK
Cancel

**Figure 5: Selecting workspace folder in Eclipse.**

At this point:

- Open **File** menu, **New** submenu and select **Project**.
- In **General**, select **Project**.
- In **Project name** text field, enter the name of the desired project (as chosen from Table 4) and click **Finish**.
- In **Window** menu, **Show View** submenu, select **C/C++ Projects**.

At this point Eclipse is ready to build SDK configurations.



An outline is not available.

No consoles to display at this time.

0 items selected

**Figure 6: Eclipse window with open project (gnssapp_os20_rvct shown as example).**

## 6.4 Configuring a project

The first step to configure a project is to select which target must be used. You can select the target by right clicking the project name in **Project Explorer** window, as showed in Figure 7.

*Note: There are other ways to select the project, like opening the project properties or using the drop down menu to the right of **Build** icon. Note that some of these methods would also start building process.*

Each project is provided with a specific configuration makefile. The name of this file can be configured opening the project properties and opening the C/C++ Build/Environment options page, as showed in Figure 8.

The name can be set changing the `PROJ_CFG` option with proper filename (without extension). This file will be searched in subfolder `configs` of project folder.

*Note: New targets can be created from project Properties, and custom configuration makefiles (based on ready-to-go makefiles) can be associated to them.*

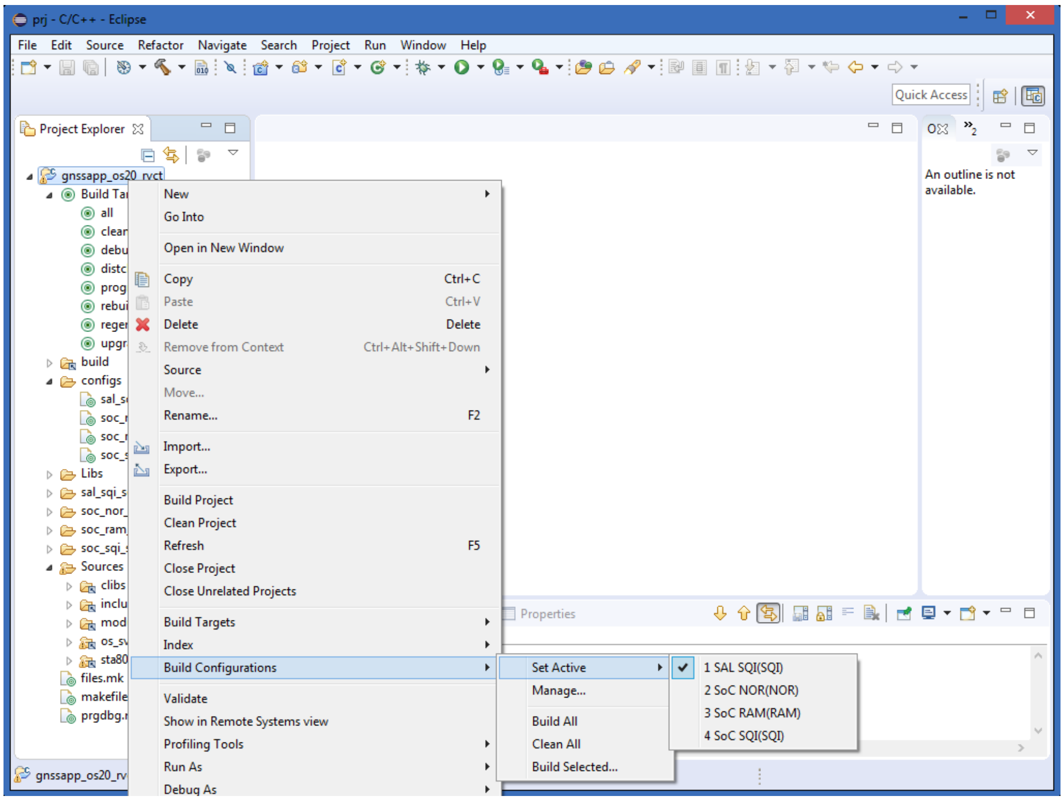
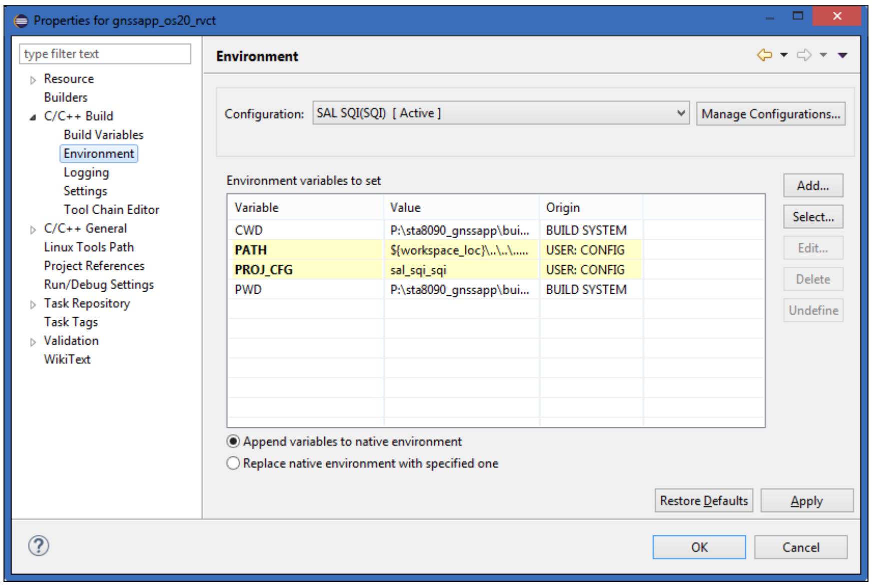


Figure 7: Selecting target.



**Environment**

Configuration: SAL SQI(SQI) [ Active ] Manage Configurations...

Environment variables to set

| Variable | Value | Origin |
|---|---|---|
| CWD | P:\sta8090_gnssapp\bui... | BUILD SYSTEM |
| PATH | ${workspace_loc}\..\..\..... | USER: CONFIG |
| PROJ_CFG | sal_sqi_sqi | USER: CONFIG |
| PWD | P:\sta8090_gnssapp\bui... | BUILD SYSTEM |

[✓ Append variables to native environment]
Replace native environment with specified one

Figure 8: Configuring target configuration makefile.

Configuration makefile includes a set of make variables that will be used to build the target. Following sections will deal with them and their interaction with others makefiles.

### 6.4.1 Configuring project type

The `PROJ_MOD_APP` variable can be used to specify the type of target that must be build. Table 6 shows available values.

| Name | Meaning |
|---|---|
| `gnssapp` | The target is a default GNSS application. It will inherit all configurations from `app_binimg_defs.mk` makefile, available in `build` subfolder. |
| `binary` | The target is a standard ST binary. It will inherit all configurations from `app_binimg_defs.mk` and `app_gnssapp_defs.mk` makefiles, available in `build` subfolder. |

**Table 6: Values for PROJ_MOD_APP variable.**

Options in configuration makefile included should not be modified.

The `PROJ_MOD_EXTRA` variable can be used to add a postfix to the target name to differentiate it from similar targets.

### 6.4.2 Configuring package and memory regions

A target is commonly identified by a triple: package, load region and NVM region.

Package can be configured using `PROJ_PACKAGE` variable as showed in Table 7.

| Name | Meaning |
|---|---|
| `SAL` | The target will run on a SAL package |
| `SOC` | The target will run on a SoC package |

**Table 7: Values for PROJ_PACKAGE variable.**

Load region can be configured using `PROJ_LOAD_REGION` as showed in Table 8.

| Name | Meaning |
|---|---|
| `SQI` | Target will run from SQI |
| `NOR` | Target will run from NOR |
| `RAM` | Target will run from RAM |

**Table 8: Values for PROJ_LOAD_REGION variable.**

NVM (Non Volatile Memory) region can be configured using `PROJ_NVM_REGION` as showed in Table 9.

| Name | Meaning |
|---|---|
| `SQI` | NVM will be stored in SQI |
| `NOR` | NVM will be stored in NOR |
| `RAM` | NVM will be stored in RAM |

**Table 9: Values for PROJ_NVM_REGION variable.**

The memory addresses used for each setting are pre-defined in another configuration makefile, namely `core_defs.mk` in `build` subfolder. This file defines default values for each setting. The variables used to specify load and NVM regions are shown in Table 10.

| Variable name | Variable meaning |
|---|---|
| `LR_CODE_BASE` | The start address of the load region used for the code. The `_FWUPG` variant is used if FWUPG is selected as optional feature. |
| `LR_CODE_SIZE` | The total size of the load region used for the code. The `_FWUPG` variant is used if FWUPG is selected as optional feature. |
| `NVM_BASE` | The base address in execution region of the NVM (non-volatile memory) region used by GNSS libraries to store non-volatile data. |
| `NVM_SIZE` | This defines the size of the NVM region used by GNSS libraries to store non-volatile data. |

**Table 10: Variables to configure load and NVM regions.**

Target configuration makefile can also be used to redefine some other memory regions: data TCM and OS heap.

Data TCM region of ARM946 can be configured using `DATA_TCM_START` and `DATA_TCM_SIZE`. This setting will also affect the instruction TCM of ARM946, as the total memory available for both is 256kb. These areas, as tightly coupled to ARM core, has the fastest access speed and can be used for code and data dealing with speed critical processes.

Table 11 shows the definition of `DATA_TCM_START` and `DATA_TCM_SIZE` variables. Table 12 and Table 13 show available values for `DATA_TCM_SIZE` variable for Teseo2 and Teseo3 families.

| Variable name | Variable meaning |
|---|---|
| `DATA_TCM_START` | The start address of DTCM area. Typically default value (`0x100000`) should not be changed to ensure that DTCM region stays nearby ITCM region. |
| `DATA_TCM_SIZE` | The total size of DTCM. This variable also affects size of ITCM. Refer to Table 12 and Table 13 for allowed values. |

**Table 11: Variables to configure DTCM region.**

| DATA_TCM_SIZE | ITCM | DTCM |
|---|---|---|
| `0x2c000` | 80kB | 176kB |
| `0x28000` | 96kB | 160kB |
| `0x24000` | 112kB | 144kB |
| `0x20000` | 128kB | 128kB |
| `0x1c000` | 144kB | 112kB |
| `0x18000` | 160kB | 96kB |
| `0x14000` | 176kB | 80kB |
| `0x10000` | 192kB | 64kB |

**Table 12: Values allowed for `DATA_TCM_SIZE` for Teseo2 family.**

| DATA_TCM_SIZE | ITCM | DTCM |
|---|---|---|
| `0x3c000` | 16kB | 240kB |
| `0x38000` | 32kB | 224kB |
| `0x34000` | 48kB | 208kB |
| `0x30000` | 64kB | 192kB |
| `0x2c000` | 80kB | 176kB |
| `0x28000` | 96kB | 160kB |
| `0x24000` | 112kB | 144kB |
| `0x20000` | 128kB | 128kB |

**Table 13: Values allowed for `DATA_TCM_SIZE` for Teseo3 family.**

OS heap is the region that operating system will use for dynamic memory allocation. Typically, it would refer to a region in external RAM. If the size of this region is set to zero, then DTCM will be used as heap regions. Otherwise, DTCM will be available to allocate different information.

Table 14 shows the definition of `OS_HEAP_BASE` and `OS_HEAP_SIZE` variables.

| Variable name | Variable meaning |
|---|---|
| `OS_HEAP_BASE` | The start address of a region that will be used as heap for dynamic memory allocation |
| `OS_HEAP_SIZE` | The size of a region that will be used as heap for dynamic memory allocation. If this size equals zero, DTCM will be used as default heap region. |

**Table 14: Variables to configure heap region.**

### 6.4.3 Configuring optional features

Optional features that can be activated/deactivated by specifying them in variable `PROJ_OPT_FEATURES`. This variable is a list of activation flags that can be given in any order. Available flags are reported in Table 15.

| Flag | Feature | Default |
|---|---|---|
| `SBAS` | SBAS (WAAS/EGNOS) | Enabled |
| `SDLOG` | SD card support | Disabled |
| `RTCM` | Differential GPS | Disabled |
| `STAGNSS` | ST AGNSS | Disabled |
| `STBIN` | ST binary protocol | Disabled |
| `DRAW` | DRAW (if available) | Disabled |
| `USB` | USB VCOM support | Disabled |
| `FWUPG` | FW upgrade support | Disabled |

**Table 15: List of flags that can be used in `PROJ_OPT_FEATURES` variable.**

Teseo3 family has some others optional flags that can be used to control the generation of the executable. These are showed in Table 16.

| **Flag** | **Feature** | **Default** |
|---|---|---|
| `TCXO_26` | Set TCXO to 26MHz | Enabled |
| `TCXO_48` | Set TCXO to 48MHz | Disabled |

**Table 16: Teseo3 family specific optional flags.**

*Note: If no `TCXO_xx` is specified, `TCXO_26` is assumed, this is why it is reported as enabled.*

*If RAM load region is selected, `FWUPG` switch will be treated as don't care.*

`FWUPG` option, as explained in Section 5.7, can be used to generate images to be used with FW upgrade tool. In this case the `core_defs.mk` will use proper load region settings.

As Teseo3 family can be fed with either a 26MHz TCXO or 48MHz TCXO, this information must be passed to the build environment to generate a proper image. As reported, `TCXO_26` and `TCXO_48` can be used to get an image that can work with 26MHz TCXO or 48MHz TCXO respectively.

### 6.4.4 Configuring relocation of GNSS code and data

As described in Section 5.6, CPU performance can be improved by relocating to ITCM and DTCM regions portions of code and data from load region. Some defaults subset of code and data can be easily relocated using variable `PROJ_FAST_ENABLE` in target configuration file. This variable is a list of activation flags that can be given in any order. Available flags are reported in Table 17.

| **Flag** | **Feature** | **Default** |
|---|---|---|
| `GNSSLIB` | Improve CPU performance of GNSS library | Disabled |
| `STAGNSS` | Improve CPU performance of ST AGNSS | Disabled |
| `DRAW` | Improve CPU performance of DRAW | Disabled |
| `BIN` | More CPU improvements used for standard ST binaries | Disabled |

**Table 17: List of flags that can be used in `PROJ_FAST_ENABLE` variable.**

The scatter files used for relocating code can be found under `sta8090\gpsapp`. Here are three templates:

- `sta8090_arm9gnssapp_FR_template.scf`: for FreeRTOS with ARM RVCT compiler;
- `sta8090_arm9gnssapp_FR_template.map`: for FreeRTOS with GCC based compilers
- `sta8090_arm9gnssapp_template.scf`: for OS20+ with ARM RVCT compiler;

These scatter files are parametric. Some parameters in it are specified through macros. This means that scatter file is pre-processed with C/C++ pre-processor before being used by the linker.

### 6.4.5 Other configuration variables

Some other variables could be configured from target makefile.

`PROJ_ENTRY_POINT` can be used to specify a custom entry point for the target. By default, `reset_vector` (defined in `crt0` module for OS20+ and `crt0_FR` module for FreeRTOS) is used.

`PROJ_SCFFILE_BASENAME` can be used to specify a custom scatter file template that will be preprocessed and then used for linking. The default scatter files templates can be found in `sta8090/gpsapp` folder: they depend on the combination of operating system (OS20+ or FreeRTOS) and compiler (ARM RVCT or GCC) used.

*Note: These variables should not be used unless strictly needed.*

### 6.4.6 Configuring operating system and toolchain

The target configuration makefile provides variables to configure operating system and toolchain.

*Note: the SDK is provided with a set of ready-to-go projects that realizes most common combination of operating system and toolchain. It is adviced to use desired project and modify this file only if strictly needed.*

Variable `PROJ_MOD_OS` can be used to select which operating system is used for current target. Table 18 shows possible values.

| Name | Meaning |
|---|---|
| `OS20` | Use OS20+ operating system |
| `FREERTOS` | Use FREERTOS operating system |

**Table 18: Values for `PROJ_MOD_OS` variable.**

Variable `PROJ_TC` can be used to select which toolchain will be used for target compilation. Table 19 shows possible values.

| Name | Meaning |
|---|---|
| `rvct` | Use ARM RealView Compiler Toolchain |
| `gae` | Use GCC for ARM Embedded Toolchain |

**Table 19: Values for `PROJ_TC` variable.**

In the case rvct is used, the variant of RVCT to be used can be configured using `PROJ_COMPILER` variable. Table 20 shows possible values.

| Name | Meaning |
|---|---|
| `DS5` | Use RVCT from ARM DS-5.xx |
| `RVDS31` | Use RVCT from RVDS 3.1 |
| `RVDS40` | Use RVCT from RVDS 4.0 |
| `RVDS41` | Use RVCT from RVDS 4.1 |

**Table 20: Values for `PROJ_COMPILER` variable.**

Toolchains are searched in their default installation paths. If the toolchain is installed in a different folder, it can be configured in makefile specific for toolchain.

For ARM toolchains, installation path can be configured by modifying `TC_PATH_INST_DS5` or `TC_PATH_INST_RVDS` in `tc_rvct_defs.mk` configuration file that can be found in `build` subfolder.

For GCC toolchains, installation path can be configured by modifying `TC_PATH_INST_GAE` in `tc_gae_defs.mk` configuration file that can be found in `build` subfolder.

## 6.5 Customizing a project

A project can be customized in different ways. For each project, a default `makefile` is provided. This contains some make variables that can be used to customize the project.

### 6.5.1 Customizing toolchain options

The default `makefile` can be used to add more options to toolchain, like macros, search paths, specific optimization and so on. Table xxx show which variables can be used for this purpose.

| Variable | Meaning |
|---|---|
| `EXT_ASMDEFS` | List of custom assembler defines (in format `name[=value]`) to be passed to C compiler |
| `EXT_ASMINCDIRS` | List of folders to be added to assembler include search path |
| `EXT_ASMOPTS` | List of custom assembler options to be passed to assembler (toolchain dependent) |
| `EXT_CDEFS` | List of custom defines (in format `name[=value]`) to be passed to C compiler |
| `EXT_CINCDIRS` | List of folders to be added to C compiler include search path |
| `EXT_COPTS` | List of custom C compiler options to be passed to C compiler (toolchain dependent) |
| `EXT_LIBDIRS` | List of folders to be added to libraries search path |
| `EXT_LIBOPTS` | List of custom options to be passed to linker (toolchain dependent) |

**Table 21: Variables in project `makefile` used to customize toolchain.**

### 6.5.2 Adding sources

Sources can be added in different ways. Default `makefile` contains some variables that can be used to add extra source file to the project. They are showed in Table 22.

| Variable | Meaning |
|---|---|
| `EXT_CSOURCES` | List of extra C sources that will be built in Thumb mode |
| `EXT_CARMSOURCES` | List of extra C sources that will be built in ARM mode |
| `EXT_ASMSOURCES` | List of extra assembly sources |

**Table 22: Variables in project `makefile` used to add sources.**

File files.mk can also be used to add/remove source files from compilation. Table 23 Shows variables usable and their usage.

| Variable | Meaning |
|---|---|
| `CSOURCES` | List of C sources that will be built in Thumb mode |
| `CARMSOURCES` | List of C sources that will be built in ARM mode |
| `ASMSOURCES` | List of assembly sources |

**Table 23: Variables that can be configured in `files.mk` file.**

### 6.5.3 Adding libraries

Libraries can be added the same way sources can. Default `makefile` contains variables to specify custom libraries to link to the executable. Variable `EXT_LIBS` can be used to specify a list of libraries to be linked to the executable.

### 6.5.4 Customizing GNSS library

Different aspects of GNSS library can be configured using the FW configuration tool. A custom script file can be added to the project root directory to modify these options after linking. This file must be called `fwcfg.txt` and must be filled following instructions in FW configuration manual.

A default `fwcfg_soc_default.txt`, in `build` subfolder, is always applied to ALL configuration of ALL projects. It can be used to spread a configuration to all of them. If some options are specified in both custom `fwcfg.txt` and default `fwcfg_soc_default.txt`, the ones in custom `fwcfg.txt` will have priority.

The default `fwcfg_soc_default.txt` will apply these changes to ALL configurations:

- The CPU speed is set to 192f0 MHz.

*Note: These files are not used by binary related projects. For them, there are specific configuration files located in folder* `build\binimgsupp`*.*

## 6.6 Building a configuration

### 6.6.1 Building using toolbar

*Note: These rules apply to both ARM RVCT, GCC for ARM embedded.*

These chapter does not pretend to be an Eclipse manual, but to provide some information about prebuilt configurations for Teseo products SDK.

Select target to build by clicking button on toolbar.

*Note: This button is active only if a project is selected.*

The format of target name is:

```
<package> <code-region>(<nvm-region>)
```

where:

- `package` specifies the Teseo products package (SoC[2], SAL[3]) on which the code will run.
- `code-region` specifies where the code is allocated when loaded on board. Note that some portion of code is relocated at run-time. This is specified in scatter files.
- `nvm-region` specifies the region where the GNSS non-volatile data will be stored.

### 6.6.2 Building using makefile targets

A target can also be built using makefile targets. They are available in **Build targets** pane on the right side of Eclipse window.

*Note:* *In latest Eclipse Neon, build targets are available also as subfolder of the project.*

Available targets are described in Table 24.

| Target | Meaning |
|---|---|
| `all` | Build applying all changes |
| `clean` | Delete binary and images (this would not clear scripts and support files) |
| `distclean` | Delete everything related to selected target |
| `rebuild` | This is the equivalent of `make clean` + `make all` |
| `regenerate` | This is the equivalent of `make distclean` + `make all` |

**Table 24: Target available for each configuration.**

### 6.6.3 Building from command line

Makefiles can also be used from command line. In this way there is no need to open Eclipse, everything can be controlled from command line.

To build using makefiles from command line, open a command line prompt in specific project folder. Then you can use make executable provided with SDK (available in `build\tools` folder) to start building process.

On command line, it is mandatory to specify `PROJ_CFG` the target that must be built. For example, to build SAL SQI(SQI) target of `gnssapp_os20_rvct` project, here are the steps:

[2] For Teseo2: STA8088EXG and all its derivatives. For Teseo3: STA8090EXG and all its derivatives.

[3] For Teseo2: STA8088FG/GA and all their derivatives. For Teseo3: STA8089FG/GA, STA8090FG/GA and all their derivatives.

```
> cd sta8090_gnssapp\build\eclipse\prj\gnssapp_os20_rvct
> ..\..\..\tools\make.exe PROJ_CFG=sal_sqi_sqi all
```

All makefile targets available from Eclipse IDE are available on command line.

## 6.7 Programming an executable

Once a target has been built, it must be flashed to the device. This can be done with tools communicating using UART or using the debugger through JTAG port.

*Note:* *Currently only Lauterbach scripts supports flashing through JTAG.*

Programming through UART can be done using some tools provided in SDK package:

- **XLoader**: this uses XLoader protocol to flash image over UART. This protocol will interacts directly with Teseo products boot ROM code and needs interaction with the board to move boot pins.

- **FWupgrade**: this uses FWUpgrade protocol to flash image over UART. This protocol needs a running NMEA stream to be used. Once started, it will send a NMEA message to the board to stop execution and restart the secondary bootloader that will be interfaced by the tool to send the new image. In this case, no interaction is required with the board.

*Note:* *Details about using these tools is provided in specific documentation provided with the SDK.*

Programming can be done also from Eclipse IDE (or command line) through makefile targets. The target that can be used to program the device are showed in Table 25.

| Target | Meaning |
|---|---|
| `program` | Flash the target image using XLoader protocol |
| `upgrade` | Flash the target image using FW Upgrade protocol |

**Table 25: Makefile targets to program a device.**

The parameters used by this system can be controlled using a specific makefile available in project folder, `prgdbg.mk`. Table 26 shows the description of configurable parameters.

| Variable | Description |
|---|---|
| `PROJ_BOARD_COM` | COM port used to run the protocol |
| `PROJ_BOARD_COMSPEED` | Baudrate of used COM |
| `PROJ_BOARD_ERASENVM` | Set to `on` to erase NVM during flashing |
| `PROJ_BOARD_UPGRECOVERY` | Set to `on` to run in recovery mode (available only for `upgrade` target) |

**Table 26: Variables to configure device programming.**

## 6.8 Debugging SDK

Teseo products SDK can be debugged with any system that supports ARM axf format or GNU ELF format. The SDK provides scripts that can be used with Lauterbach Trace32® and scripts to debug using Segger Ozone.

### 6.8.1 Debugging with Lauterbach Trace32

Start the Trace32® software, then select `Run batchfile...` from `File` menu and select file `build\scripts\sta8090\trc_setup.cmm`. A new menu appears named as Teseo family, where a bunch of options are available, as shown in Figure 9.

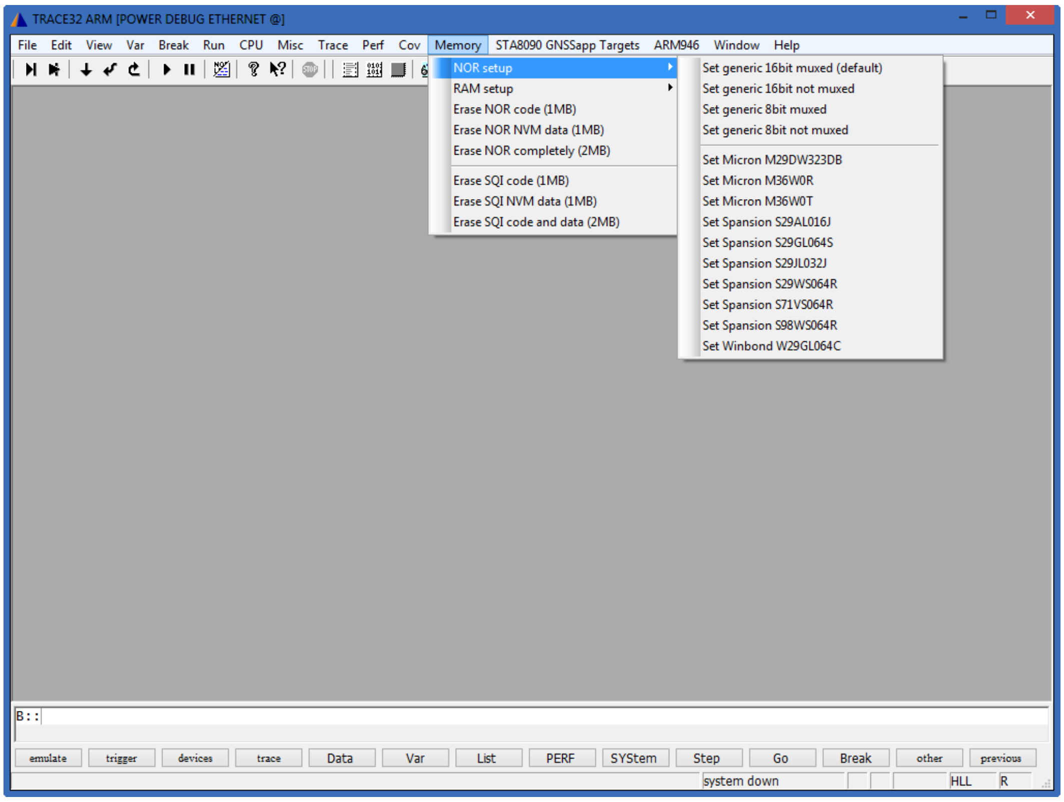

TRACE32 ARM [POWER DEBUG ETHERNET @]
File Edit View Var Break Run CPU Misc Trace Perf Cov Memory STA8090 GNSSapp Targets ARM946 Window Help
NOR setup
RAM setup
Erase NOR code (1MB)
Erase NOR NVM data (1MB)
Erase NOR completely (2MB)
Erase SQI code (1MB)
Erase SQI NVM data (1MB)
Erase SQI code and data (2MB)
Set generic 16bit muxed (default)
Set generic 16bit not muxed
Set generic 8bit muxed
Set generic 8bit not muxed
Set Micron M29DW323DB
Set Micron M36W0R
Set Micron M36W0T
Set Spansion S29AL016J
Set Spansion S29GL064S
Set Spansion S29JL032J
Set Spansion S29WS064R
Set Spansion S71VS064R
Set Spansion S98WS064R
Set Winbond W29GL064C
B::
emulate trigger devices trace Data Var List PERF SYStem Step Go Break other previous
system down
HLL
R

**Figure 9: Trace32 window showing customized menus**

Figure 9 shows a set of menu items. The opened custom menu can be used to control memory setup of the system.

In case of parallel NOR flashes, there is a set of available configurations that can be used to setup configuration of used board.

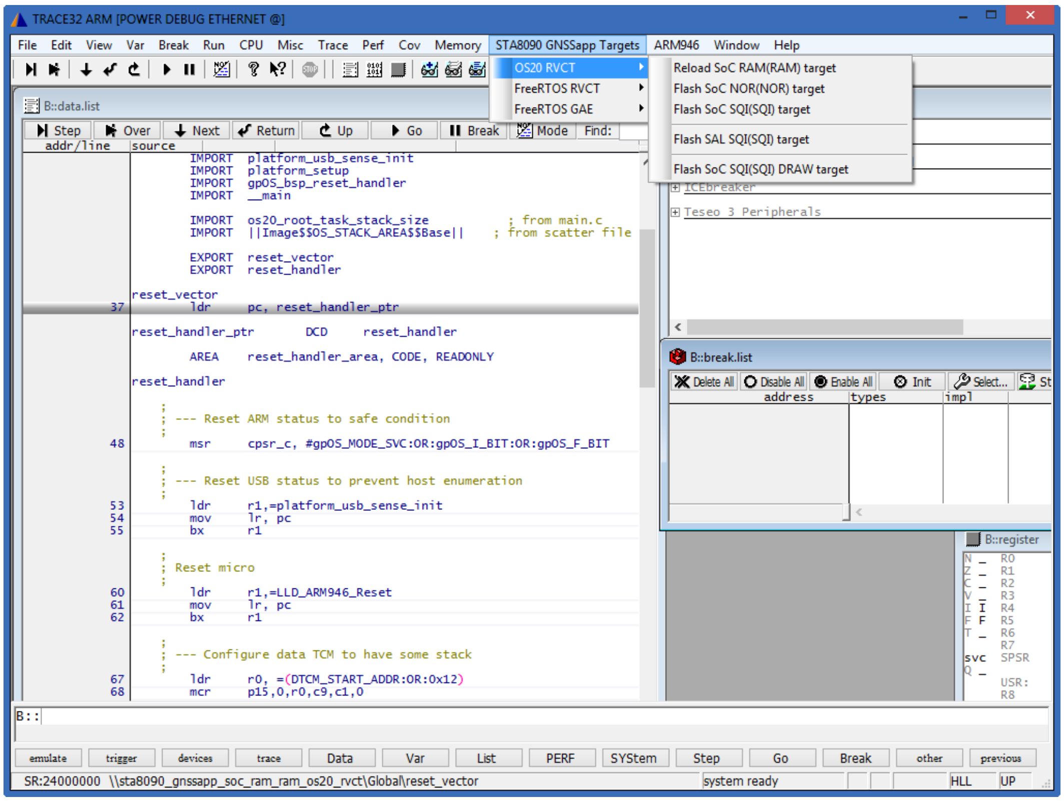
*Note: More configurations can be added by customizing cmm script files.*

There are also menu items to erase regions of NOR flash. The once provided by SDK are referring to default memory partitioning of SDK images.

In case of RAM memories, you can select through a set of available configurations.

In case of SQI flashes, the memory is recognized by the scripts itself, so there are only menu items to erase specific regions.

As showed in Figure 10, the second custom menu contains menu items to program and start images built with SDK. Menu items refer to the naming used in Eclipse configurations.


TRACE32 ARM [POWER DEBUG ETHERNET @]
File Edit View Var Break Run CPU Misc Trace Perf Cov Memory STA8090 GNSSapp Targets ARM946 Window Help
OS20 RVCT
FreeRTOS RVCT
FreeRTOS GAE
Reload SoC RAM(RAM) target
Flash SoC NOR(NOR) target
Flash SoC SQI(SQI) target
Flash SAL SQI(SQI) target
Flash SoC SQI(SQI) DRAW target
B::data.list
Step Over Next Return Up Go Break Mode Find:
addr/line source
ICEbreaker
Teseo 3 Peripherals
B::break.list
Delete All Disable All Enable All Init Select...
address types impl
B::register
emulate trigger devices trace Data Var List PERF SYStem Step Go Break other previous
system ready
HLL UP

```
        IMPORT  platform_usb_sense_init
        IMPORT  platform_setup
        IMPORT  gpOS_bsp_reset_handler
        IMPORT  __main

        IMPORT  os20_root_task_stack_size          ; from main.c
        IMPORT  ||Image$$OS_STACK_AREA$$Base||    ; from scatter file

        EXPORT  reset_vector
        EXPORT  reset_handler

reset_vector
        ldr     pc, reset_handler_ptr

reset_handler_ptr       DCD     reset_handler

        AREA    reset_handler_area, CODE, READONLY

reset_handler

    ;
    ; --- Reset ARM status to safe condition
    ;
        msr     cpsr_c, #gpOS_MODE_SVC:OR:gpOS_I_BIT:OR:gpOS_F_BIT

    ;
    ; --- Reset USB status to prevent host enumeration
    ;
        ldr     r1,=platform_usb_sense_init
        mov     lr, pc
        bx      r1

    ;
    ; Reset micro
    ;
        ldr     r1,=LLD_ARM946_Reset
        mov     lr, pc
        bx      r1

    ;
    ; --- Configure data TCM to have some stack
    ;
        ldr     r0, =(DTCM_START_ADDR:OR:0x12)
        mcr     p15,0,r0,c9,c1,0
```

```
B::
SR:24000000  \\sta8090_gnssapp_soc_ram_ram_os20_rvct\Global\reset_vector
```

**Figure 10: Trace32 window showing targets and typical debug session**

In target image submenus there are two sets of options:

- Reload <target>: the specified target is loaded into external RAM;
- Flash <target> the specified target is flashed into NOR or SQI as selected.

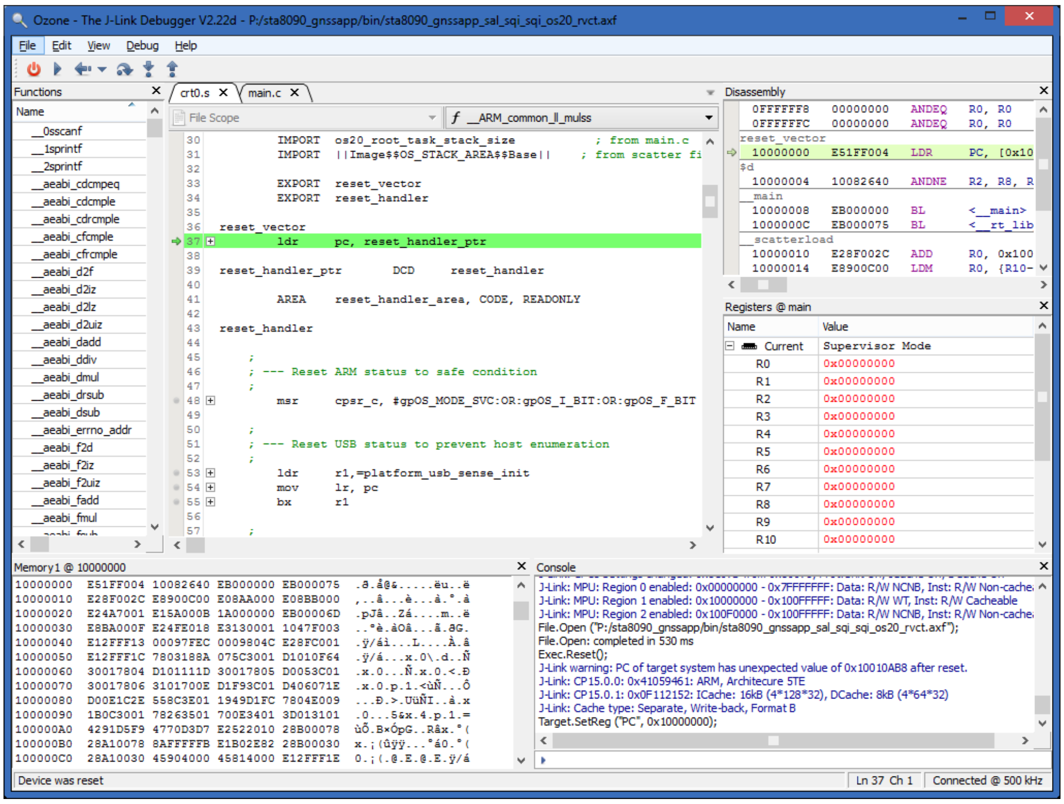
*Note: It is important to configure properly parallel NOR flash or RAM before using a related target program.*

In Figure 10 it is showed a typical debug session.

## 6.8.2 Debugging with Segger Ozone

When a target is built, a proper Segger Ozone script file is generated starting from a template, available in `build\scripts\sta8090` folder. There is a specific makefile target to start a debug session with Ozone: `debug`.

By running `debug` makefile target, a window of Segger Ozone will be opened and the generated script will be loaded to connect to the board, as showed in Figure 11.



```
        IMPORT  os20_root_task_stack_size         ; from main.c
        IMPORT  ||Image$$OS_STACK_AREA$$Base||    ; from scatter fi

        EXPORT  reset_vector
        EXPORT  reset_handler

reset_vector
        ldr     pc, reset_handler_ptr

reset_handler_ptr       DCD     reset_handler

        AREA    reset_handler_area, CODE, READONLY

reset_handler

    ;
    ; --- Reset ARM status to safe condition
    ;
        msr     cpsr_c, #gpOS_MODE_SVC:OR:gpOS_I_BIT:OR:gpOS_F_BIT

    ;
    ; --- Reset USB status to prevent host enumeration
    ;
        ldr     r1,=platform_usb_sense_init
        mov     lr, pc
        bx      r1

    ;
```

```
  0FFFFFF8   00000000   ANDEQ   R0, R0
  0FFFFFFC   00000000   ANDEQ   R0, R0
reset_vector
  10000000   E51FF004   LDR     PC, [0x10
$d
  10000004   10082640   ANDNE   R2, R8, R
__main
  10000008   EB000000   BL      <__main>
  1000000C   EB000075   BL      <__rt_lib
__scatterload
  10000010   E28F002C   ADD     R0, 0x100
  10000014   E8900C00   LDM     R0, {R10-
```

| Name | Value |
|---|---|
| Current | Supervisor Mode |
| R0 | 0x00000000 |
| R1 | 0x00000000 |
| R2 | 0x00000000 |
| R3 | 0x00000000 |
| R4 | 0x00000000 |
| R5 | 0x00000000 |
| R6 | 0x00000000 |
| R7 | 0x00000000 |
| R8 | 0x00000000 |
| R9 | 0x00000000 |
| R10 | 0x00000000 |

```
10000000  E51FF004 10082640 EB000000 EB000075  .ð.å@&.....ëu..ë
10000010  E28F002C E8900C00 E08AA000 E08BB000  ,..â...è...à.°.à
10000020  E24A7001 E15A000B 1A000000 EB00006D  .pJâ..Zá....m..ë
10000030  E8BA000F E24FE018 E3130001 1047F003  ..ºè.àOâ...ã.ðG.
10000040  E12FFF13 00097FEC 0009804C E28FC001  .ÿ/áì...L....À.â
10000050  E12FFF1C 7803188A 075C3001 D1010F64  .ÿ/á...x.0\.d..Ñ
10000060  30017804 D101111D 30017805 D0053C01  .x.0...Ñ.x.0.<.Ð
10000070  30017806 3101700E D1F93C01 D406071E  .x.0.p.1.<ùÑ...Ô
10000080  D00E1C2E 558C3E01 1949D1FC 7804E009  ...Ð.>.UüÑI..à.x
10000090  1B0C3001 78263501 700E3401 3D013101  .0...5&x.4.p.1.=
100000A0  4291D5F9 4770D3D7 E2522010 28B00078  ùÕ.B×ÓpG..Râx.°(
100000B0  28A10078 8AFFFFFB E1B02E82 28B00030  x.¡(ûÿÿ...°á0.°(
100000C0  28A10030 45904000 45814000 E12FFF1E  0.¡(.@.E.@.E.ÿ/á
```

```
J-Link: MPU: Region 0 enabled: 0x00000000 - 0x7FFFFFFF: Data: R/W NCNB, Inst: R/W Non-cache
J-Link: MPU: Region 1 enabled: 0x10000000 - 0x100FFFFF: Data: R/W WT, Inst: R/W Cacheable
J-Link: MPU: Region 2 enabled: 0x100F0000 - 0x100FFFFF: Data: R/W NCNB, Inst: R/W Non-cache
File.Open ("P:/sta8090_gnssapp/bin/sta8090_gnssapp_sal_sqi_sqi_os20_rvct.axf");
File.Open: completed in 530 ms
Exec.Reset();
J-Link warning: PC of target system has unexpected value of 0x10010AB8 after reset.
J-Link: CP15.0.0: 0x41059461: ARM, Architecure 5TE
J-Link: CP15.0.1: 0x0F112152: ICache: 16kB (4*128*32), DCache: 8kB (4*64*32)
J-Link: Cache type: Separate, Write-back, Format B
Target.SetReg ("PC", 0x10000000);
```

**Figure 11: Ozone window**

The path to Segger Ozone executable can be configured by modifying makefile `debug_ozn.mk` in `build\eclipse\prj\build` folder. Here, the makefile variable `DBG_OZONE_PATH_INST` must point to the installation folder of Segger Ozone.

# 7 Appendix A: Teseo3 RAM and CPU free for customer application

Here is the current status for Teseo3 using default SDK configuration for SAL SQI (SQI) compilation target.

Details of the configuration:

- CPU running at ~200 MHz
- No external RAM
- Following build options ON: SBAS
- NMEA output on UART
- Debug log disabled

For CPU measurement, a Labsat scenario with 2 constellations (GPS+Beidou) and 21 satellites (10 GPS, 11 Beidou, 2 SBAS) is used and the measures are handled during tracking phase (data collection starting from 12mn50s) with 1 Hz fix rate.

## 7.1 RAM and CPU free for default configuration

Using `PROJ_FAST_ENABLE` option, `GNSSLIB` is set to allow to move some code in ITCM to improve CPU usage.

| | **GNSSLIB fast disabled** | | **GNSSLIB fast enabled** | |
|---|---|---|---|---|
| | **RAM free** | **CPU free** | **RAM free** | **CPU free** |
| Using OS20 OS and RVCT compiler: | 127 kBytes | 59% | 78 kBytes | 77% |
| Using Free RTOS and GCC compiler: | 113 kBytes | 42% | 66 kBytes | 71% |
| Using Free RTOS and RVCT compiler: | 120 kBytes | 56% | 70 kBytes | 76% |

**Table 27: RAM and CPU free for default configuration**

## 7.2 RAM and CPU cost of the build options

Values given below have to be taken into account on top of the default configuration.

### 7.2.1 RAM cost

| **Module** | **OS20/RVCT** | **FreeRTOS/GCC** | **FreeRTOS/RVCT** |
|---|---|---|---|
| RTCM | 5kB | 5kB | 5kB |

| | | | |
|---|---|---|---|
| STAGNSS | 13kB | 16kB | 13kB |
| DR | 20kB | 20kB | 20kB |

**Table 28: RAM cost of build options**

Note: RTCM with an extra port is 5kB. Mapped on Debug, it is only 2.3kB.

### 7.2.2 CPU cost

| Module | OS20/RVCT | FreeRTOS/GCC |
|---|---|---|
| RTCM | N/A | N/A |
| STAGNSS | 1% | 1% |
| DR | 7% | 12% |

**Table 29: CPU cost of build options**

Notes:

- DR and STAGNSS CPU costs have been evaluated with `GNSSLIB` option.
- RTCM CPU cost not evaluated because no existing setup in place.

## 7.3 Hints to increase free RAM amount

On top of RAM free in Table 16, the 32kbytes of the backup RAM memory can be used to map additional data (see Section 8).

If the targeted application is booting from backup RAM, it will limit this area, the backup RAM has to be properly organised (For example, 8kB for boot code and 24kB for data). So, in total, considering Free RTOS GCC default configuration, a maximum amount of 145kB is available without `GNSSLIB` option and 98kB available with `GNSSLIB` option.

ITCM/DTCM size, as described in Section 5.6 can only be modified by blocks of 16kB. So it is possible to increase the free DTCM amount by reducing the allocated ITCM but only taking into account this constraint. For example, if there is 10kB of free ITCM, then it is not possible to increase the total allocated DTCM amount whereas it would have been possible if there was 18kBytes of free ITCM.

# 8 Appendix B: Teseo3 additional embedded RAM

The Teseo3 family provides 256kB of tightly coupled memory as described in Section 5.6.

Two other areas can be used to get additional memory:

- Additional 2kB ("SRAM2") can be configured to contain read/write or read only data. The memory is located at address `0x30100000`.
- Additional 31kB from Backup RAM ("SRAM") can be also used for application as long as it is not used by SDK. This memory is located at address `0x40000000`.

To store data in SRAM (or SRAM2), add "SRAM_DATA" (or respectively "SRAM2_DATA") in front of the concerned variable declaration.

Examples:

```
tUInt data0 = 0; // No specific assigned location

SRAM_DATA tUInt data1 = 0; // data1 is stored in SRAM with initial
value set to 0.

SRAM2_DATA tUInt data2 = 8; //data2 is stored in SRAM2 with initial
value set to 8;
```

*Note:* *To enable SRAM2 usage, the following lines must be added in `platform_setup_mpu()` function, just after the local variables declaration. `platform_setup_mpu()` is located in `sta8090\platforms\soc\platform.c` or in `sta8090\platforms\sal\platform.c`, whether SOC or SAL target is used.*

```
/* Enable SRAM2 Full availability */
LLD_PRCC_SetROMPatchMode(LLD_PRCC_ROMPATCHMODE_FULL_AVAIL);
```

# 9 Appendix C: NVM size management

Flash size on Teseo products is 2MB and is, by default, equally split between binary and NVM (each region is 1MB wide).

NVM size is defined as described in 6.4.2 section and can be modified on purpose (in case of an amount of code superior to 1MB which would require to decrease NVM size for instance).

Before any change, the following rule has to be observed to modify NVM size:

- If ST AGNSS is disabled: minimum NVM size is 128kB*2
- If ST AGNSS is enabled: minimum NVM size is 256kB*2

Then:

- Update `nvm_data_region_size` parameter to match NVM size.
- Update `nvm_data_region_data` to be "2MB – `nvm_data_region_size`"
- Modify the memory descriptor for NVM region.
  For SoC targets, this is located in file `sta8090/platforms/soc/platform.c`, and it is named `platform_mputable_xxx_nvm` (where xxx depends on type of memory and could be NOR or SQI).
  For SAL targets, this is located in file `sta8090/platforms/sal/platform.c`, and it is part of `memory_table[]` array.
  Memory descriptor fields to be changed are:
  - `size`: change to `LLD_ARM946_MEMORYSIZE_xxx`, where xxx must be 256 or 512 depending on disabling/enabling of STAGNSS;
  - `base_addr`: change the address inside `LLD_ARM946_GETBASEADDR(…)` macro that is related to original NVM configuration.

## 9.1 Example for Teseo2 family

In a SOC SQI (SQI) project, NVM size can be reduced from 1MB to 512kB by modifying some parameters in Eclipse project as showed in Table 30.

| | NVM size = 1MB | NVM size = 512 kB |
|---|---|---|
| `nvm_data_region_base` | `0x30100000` | `0x30180000` |
| `nvm_data_region_size` | `0x100000` | `0x80000` |

**Table 30: modifying NVM placement and size for Teseo2 family**

Besides, declaration of `platform_mputable_sqi_nvm[]` must be changed from:

```
#if defined( NVM_SQI_CACHED )
static const LLD_ARM946_MemoryDescTy platform_mputable_sqi_nvm  = {
0, TRUE,  TRUE,  LLD_ARM946_MEMORYSIZE_1MB,
LLD_ARM946_GETBASEADDR( 0x30100000),  0};
#else
static const LLD_ARM946_MemoryDescTy platform_mputable_sqi_nvm  = {
0, FALSE, FALSE, LLD_ARM946_MEMORYSIZE_1MB,
LLD_ARM946_GETBASEADDR( 0x30100000),  0};
#endif
```

to:

```
#if defined( NVM_SQI_CACHED )
static const LLD_ARM946_MemoryDescTy platform_mputable_sqi_nvm  = {
0, TRUE,  TRUE,  LLD_ARM946_MEMORYSIZE_512KB,
LLD_ARM946_GETBASEADDR( 0x30180000),  0};
#else
static const LLD_ARM946_MemoryDescTy platform_mputable_sqi_nvm  = {
0, FALSE, FALSE, LLD_ARM946_MEMORYSIZE_512KB,
LLD_ARM946_GETBASEADDR( 0x30180000),  0};
#endif
```

## 9.2 Example for Teseo3 family

In a SOC SQI (SQI) project, NVM size can be reduced from 1MB to 512kB by modifying some parameters in Eclipse project as showed in Table 31.

| | NVM size = 1MB | NVM size = 512 kB |
|---|---|---|
| `nvm_data_region_base` | `0x10100000` | `0x10180000` |
| `nvm_data_region_size` | `0x100000` | `0x80000` |

**Table 31: modifying NVM placement and size for Teseo3 family**

Besides, declaration of `platform_mputable_sqi_nvm[]` must be changed from:

```
#if defined( NVM_SQI_CACHED )
static const LLD_ARM946_MemoryDescTy platform_mputable_sqi_nvm  = {
0, TRUE,  TRUE,  LLD_ARM946_MEMORYSIZE_1MB,
LLD_ARM946_GETBASEADDR( 0x10100000),  0};
#else
static const LLD_ARM946_MemoryDescTy platform_mputable_sqi_nvm  = {
0, FALSE, FALSE, LLD_ARM946_MEMORYSIZE_1MB,
LLD_ARM946_GETBASEADDR( 0x10100000),  0};
#endif
```

to:

```
#if defined( NVM_SQI_CACHED )
static const LLD_ARM946_MemoryDescTy platform_mputable_sqi_nvm  = {
0, TRUE,  TRUE,  LLD_ARM946_MEMORYSIZE_512KB,
LLD_ARM946_GETBASEADDR( 0x10180000),  0};
#else
static const LLD_ARM946_MemoryDescTy platform_mputable_sqi_nvm  = {
0, FALSE, FALSE, LLD_ARM946_MEMORYSIZE_512KB,
LLD_ARM946_GETBASEADDR( 0x10180000),  0};
#endif
```

# 10 Appendix D: migrating from SDK version 2.2.x or below

## 10.1 IDE Installation

For SDK versions previous to 2.3, a specific version of Eclipse (Kepler SR2) was required. This was basically due to guarantee compatibility with ARM DS-5 CE and GNUARM plugins needed to build software.

From release 2.3 upwards, there is no need to use a specific version of Eclipse. Eclipse is not needed as well, as all targets can be built from command line as described in Section 6.6.3.

*Note: There is no problem in reusing previous setups of Eclipse Kepler SR2 (used for previous SDK versions) with SDK 2.3 upwards.*

## 10.2 Configuring a target

For SDK versions previous to 2.3, most options should be configured from Eclipse window, namely **Build properties** window of each project.

From release 2.3, options are controlled by makefiles, as described in Section 6.4.

## 10.3 Debugging a target with Segger J-Link

For SDK versions previous to 2.3, debugging with Segger J-Link was provided through a specific custom Eclipse plug-in for Teseo products.

From release 2.3, debugging with Segger J-Link can be performed with a Segger tool, Ozone.

*Note: If Eclipse Kepler SR2 is still used and Segger plugin for Teseo products is already installed, it can still be used as debugging tool for Segger J-Link, although ST will not provide any more support to it.*

# 11 Disclaimer