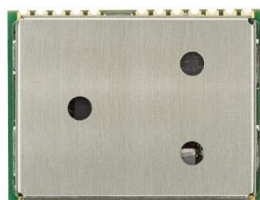


Android GNSS Driver v18.11

u-blox GNSS receiver integration

Application Note



Abstract

This document describes the steps required to integrate a u-blox GNSS receiver into an Android reference design board

Document Information

Title	Android GNSS Driver v18.11	
Subtitle	u-blox GNSS receiver integration	
Document type	Application Note	
Document number	UBX-18064207	
Revision and date	R01	23-Nov-2018
Disclosure Restriction	Confidential	

CONFIDENTIAL

u-blox or third parties may hold intellectual property rights in the products, names, logos and designs included in this document. Copying, reproduction, modification or disclosure to third parties of this document or any part thereof is only permitted with the express written permission of u-blox.

The information contained herein is provided "as is" and u-blox assumes no liability for its use. No warranty, either express or implied, is given, including but not limited to, with respect to the accuracy, correctness, reliability and fitness for a particular purpose of the information. This document may be revised by u-blox at any time without notice. For the most recent documents, visit www.u-blox.com.

Copyright © u-blox AG.

Contents

Document Information	2
Contents	3
1 Introduction	5
1.1 Scope	5
1.1.1 Demonstration platform	5
2 Android	6
2.1 Overview	6
2.2 Android Versions	7
3 u-blox Android GNSS Driver	8
3.1 Overview	8
3.2 GNSS Driver Source Files	9
3.3 GNSS Driver Interfaces	9
3.3.1 HIDL Abstraction Layer	9
3.3.2 Supported Interfaces	9
3.4 Control	10
3.5 Parser / Database	10
3.6 SUPL Client	10
3.7 A-GNSS (Multiple GNSS Assistance)	10
3.8 UDP Server (for debugging with u-center (PC))	11
3.9 TCP Server (for debugging)	11
4 Evaluation software	12
4.1 u-center for Android	12
5 Internet Services	13
5.1.1 u-blox Multiple GNSS Assistance (MGA) Online	13
5.1.2 u-blox Multiple GNSS Assistance (MGA) Offline	13
6 GNSS Integration Steps	14
6.1 Preparation of the environment	14
6.2 Building and Running the GNSS driver	14
6.2.1 Building the driver	14
6.2.2 Deployment of the driver	14
6.2.3 Android.mk Build Options	16
6.2.4 Important u-blox.conf Options	16
6.3 Accessing the Receiver from Android	16
6.3.1 File System Permissions and Ownership	16
6.3.2 SELinux configuration	17
6.4 Configuration files	17
6.4.1 General	17
6.4.2 gps.conf	17
6.4.3 u-blox.conf	18
6.5 GNSS Receiver Power	24
6.6 Source creation time	25
6.7 Leap second handling	25

6.8 Hardware information	25
6.8.1 u-blox Receiver Series	25
6.8.2 Further Information.....	25
Related documents	26
Revision history	26
Contact.....	27

CONFIDENTIAL

1 Introduction

1.1 Scope

This application note provides a comprehensive reference for customers integrating a u-blox GNSS receiver into an Android-based device. To speed up GNSS integration into customer's portable devices, u-blox has developed a specific GNSS driver that can be placed into the Android software stack.

A possible GNSS receiver integration solution is presented, in order to support customers during the prototyping phase and thus reduce time to market.

1.1.1 Demonstration platform

For demonstration purposes, HiKey620 has been chosen as a reference board (shown in Figure 1). In the basic setup, the u-blox evaluation kits can be connected by USB to the device. More information about this reference board can be found on the LeMaker website¹.



Figure 1: The HiKey620 development board

¹ <http://www.lemaker.org/product-hikey-specification.html>

2 Android

2.1 Overview

Android is an operating system expressly designed for mobile devices such as mobile phones and tablets. The operating system is based on the Linux kernel and organized in a software stack on top of the kernel, including drivers, application framework and applications.

The Android standard architecture is shown in Figure 2, where the four main layers are highlighted:

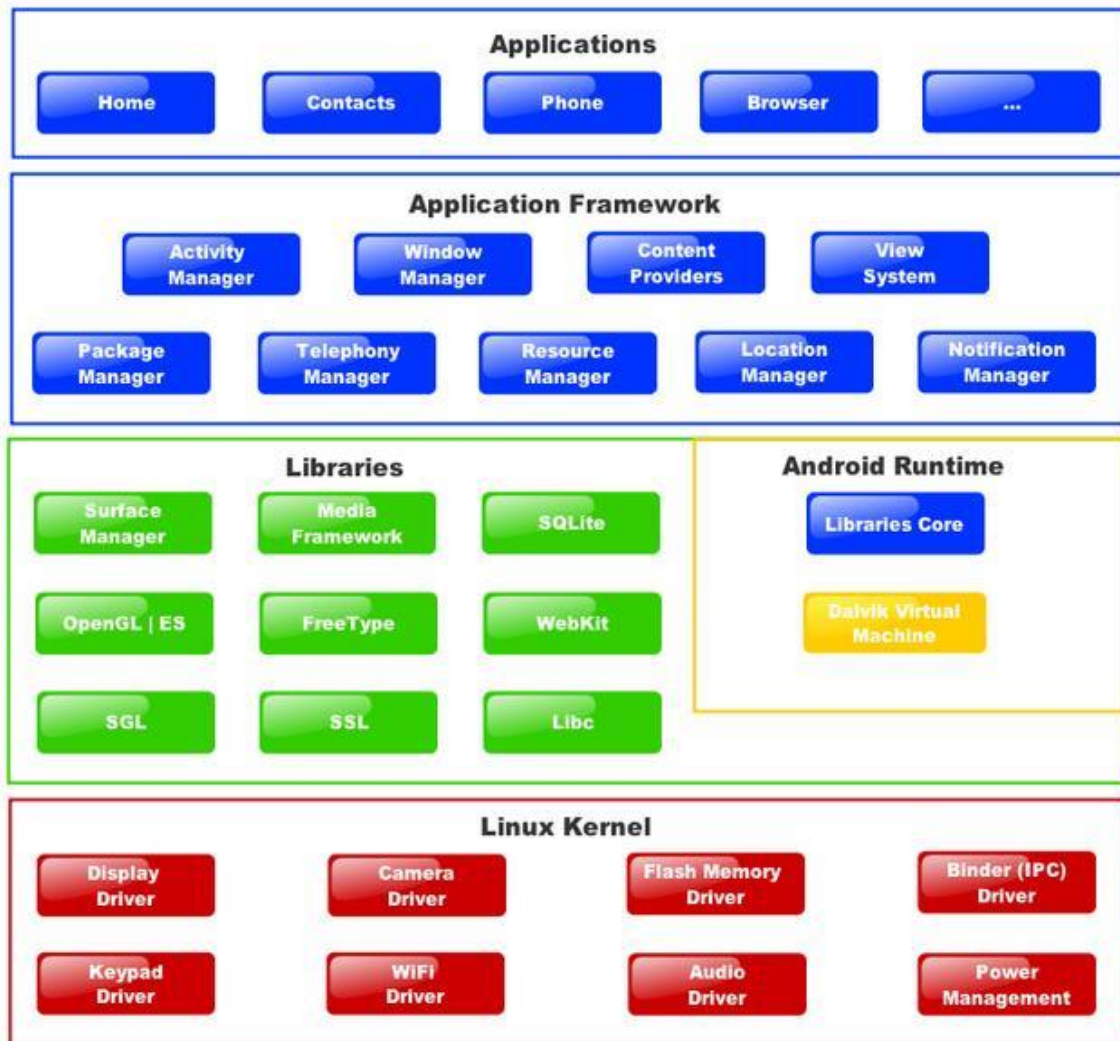


Figure 2: Android standard architecture

As summarized in Figure 2, the architecture is composed of the following layers:

- Application Layer: hosts the java-written applications used by the end-user (e.g. SMS program, calendar, maps, browser, contacts)
- Application Framework: just below the application layer, it allows the developer to access the same APIs used by the core applications and allows reuse of components
- Libraries: a set of C/C++ Libraries used by various components of the Android system
- Android runtime: contains libraries providing most of the functionalities available in the Java programming language and a Dalvik VM
- Linux Kernel: is the abstraction layer between the hardware and the rest of the software stack. Android relies on the Linux Kernel for core system services (e.g. memory management, process management)
- The u-blox GNSS driver is implemented in the form of a library that sits underneath and used by the application framework. It is **not** a Linux kernel mode driver.

2.2 Android Versions

Several versions of the Android operating system (OS) have already been released, a summary of which can be found here: <http://developer.android.com/resources/dashboard/platform-versions.html>

The driver currently supports Android versions 7.0, 8.0, 8.1, 9.0 and is fully compliant with the Android 9.0 CDD. The receiver integration described in this document is based on Android version 9.0. For older Android versions (below version 7.0), previous driver releases can be used. Please contact u-blox technical support for further information.

3 u-blox Android GNSS Driver

3.1 Overview

The diagram in Figure 3 shows an overview of the u-blox Android GNSS driver and the aiding services it connects to.

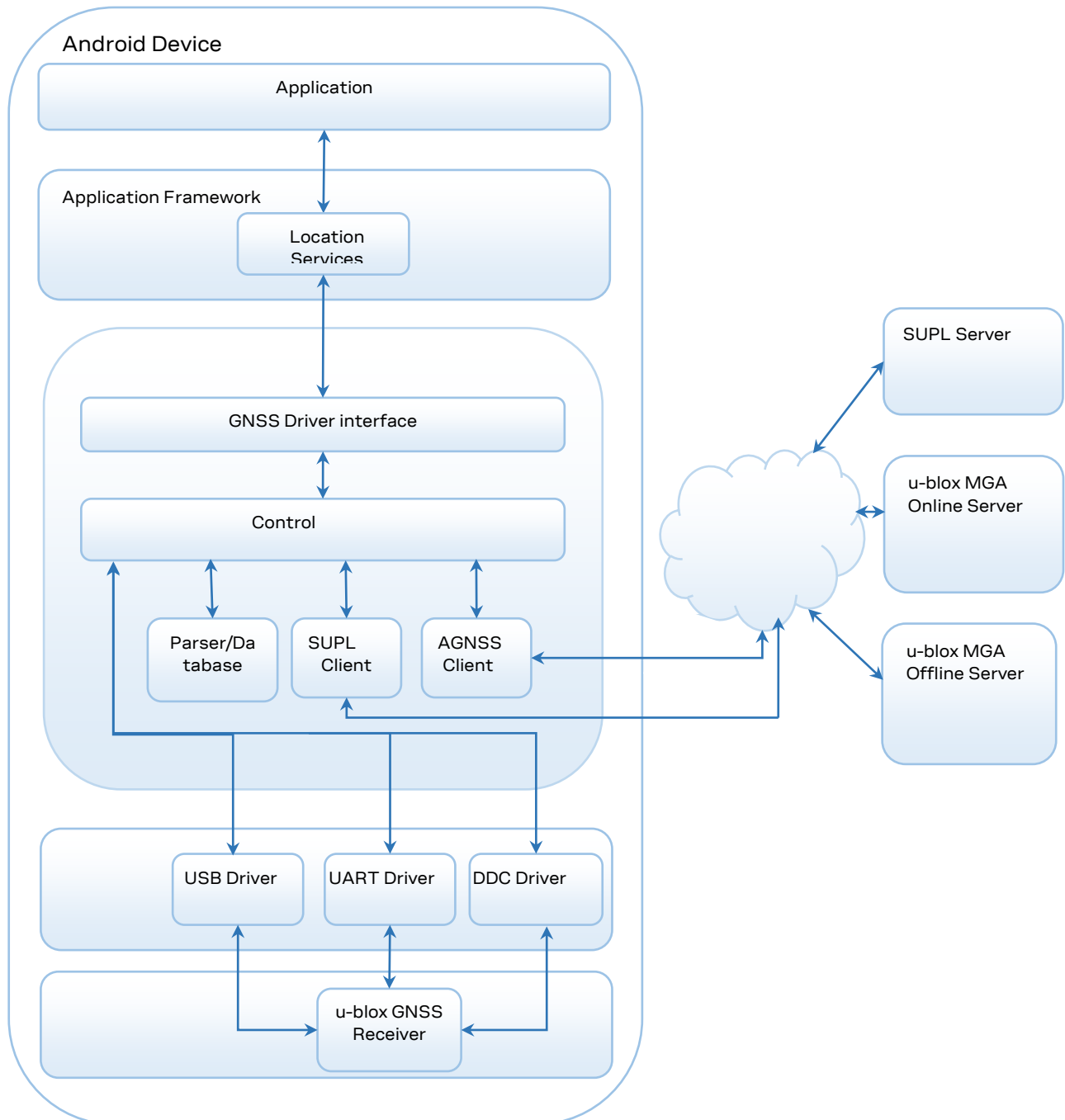


Figure 3: u-blox Android GNSS Driver Overview

When compiled, the u-blox GNSS driver takes the form of a Linux shared object library that is loaded by the Android framework when the framework itself is loaded. This typically occurs when the Android

device is booted. To make the system reload the Android GNSS Driver, the device needs to be restarted.

3.2 GNSS Driver Source Files

Refer to the release notes for a complete list of all supplied files including their headers.

3.3 GNSS Driver Interfaces

All communication between the framework and the GNSS driver takes place through a number of interfaces, which the driver has to implement via jump tables to functions. For Android versions prior to 8.0, these interfaces are defined in the framework's `gps.h` header file. Since Android 8.0 this is defined according to the "HAL Interface Definition Language" (HIDL)² that has been introduced as part of the effort to improve Android's architecture with Project Treble³. With the provided make files, this driver will automatically be compiled as a "Binderized HAL" for Android 8.0 and newer, and with the legacy `gps.h` interface for versions older than that.

3.3.1 HIDL Abstraction Layer

To maintain compatibility with Android 7.0 as well as being compatible with the new HIDL interface of Android 8.0 and newer an abstraction layer has been introduced to the Android GNSS Driver. This layer is located in the folder `hal/hidl/` of the provided sample code and is conditionally compiled depending on the Android version for which it is compiled. Changes to these files are generally not required, unless customer specific modifications to the standard Android framework interfaces are introduced.

3.3.2 Supported Interfaces

During initialization of an interface, the HIDL abstraction layer (respectively the framework in Android versions prior to 8.0) can supply a jump table of pointers to allow the driver to call back to the framework for supplying data or requesting services.

GPS Interface

- Implemented in `hal/ubx_moduleIf.cpp` & `hal/ubx_moduleIf.h`
- This is the main interface between the Android framework and the GNSS library. It implements communication of location, time, status, NMEA data and other important information between the driver and framework.

XTRA Interface

- Implemented in `hal/ubx_xtraIf.cpp` & `hal/ubx_xtraIf.h`
- This interface is intended for supplying a GNSS driver with proprietary assistance data. The u-blox GNSS driver uses this interface for implementing its use of the AssistNow Offline assistance data mechanism. By adjusting `u-blox.conf` and `gps.conf` for AssistNow Offline usage the framework is able to download aiding data and pass its contents to the GNSS driver via this interface.

A-GPS Interface

- Implemented in `hal/ubx_agpsIf.cpp` & `hal/ubx_agpsIf.h`
- This is the interface for Assisted GNSS using the SUPL Client. SUPL server information is stored in the configuration file `gps.conf`, which is read by the framework. This information is then passed to the GNSS driver via this interface, allowing the SUPL client to connect to the right SUPL server.
- Only really made use of by the SUPL client.

² <https://source.android.com/devices/architecture/hidl/>

³ <https://source.android.com/devices/architecture/treble>

RIL Interface

- Implemented in `hal/ubx_rilif.cpp` & `hal/ubx_rilif.h`
- Handles communication from the framework regarding information and activity to do with the mobile phone radio. Receiving a Network Initiated message to start a SUPL session is handled via this interface.
- Only really made use of by the SUPL client.

NI Interface

- Implemented in `hal/ubx_nilf.cpp` & `hal/ubx_nilf.h`
- Handles user interaction and authorization arising from network initiated SUPL sessions.
- Only really made use of by the SUPL client.

3.4 Control

The control module is implemented by the rest of the files in the “hal” folder. These implement the main operation of the driver.

- Reading data from the receiver and sending to the parser
- Handling commands from the framework
- Controlling the receiver
- Overall driver state handling
- UDP debug server
- TCP debug server
- `u-blox.conf` handling
- Main background control thread

3.5 Parser / Database

The parser module (implemented with all files in the “parser” folder) analyzes the sequence of bytes coming from the receiver (either via UART, USB or DDC), which will consist of both standardized NMEA messages, and u-blox proprietary UBX messages.

From these messages, the parser will extract GNSS properties (e.g. latitude, longitude, altitude, satellite positions and signal strengths, etc.) which are then passed up to the Android framework. These will eventually be passed on to any applications using the framework’s location services.

3.6 SUPL Client

The SUPL client module allows the driver to interact with a SUPL server.

- Implemented with all files in the “supl” and “asn1” folder/sub-folder
- Supports the following SUPL modes of operation:
 - Set Initiated, MS-Based (SI-MSB)
 - Set Initiated, MS-Assist (SI-MSA)
 - Network Initiated, MS-Based (NI-MSB)
 - Network Initiated, MS-Assist (NI-MSA)
- SUPL/RRLP encoding/decoding performed by C generated code from ASN.1 definitions.

3.7 A-GNSS (Multiple GNSS Assistance)

The AGNSS module implements another GNSS assistance mechanism, proprietary to u-blox, called Multiple GNSS Assistance (MGA). It is totally independent of the Android framework, being completely self-contained in the driver. MGA can be used by adding settings to the `u-blox.conf` file (see later).


3.8 UDP Server (for debugging with u-center (PC))

The u-blox GNSS driver contains a UDP server which broadcasts any received NMEA & UBX messages to any UDP client connected to the Android device. Also, if a UDP connected client sends a message to the Android device, the GNSS driver will pass this on, unchanged, to the receiver.

The functionality is implemented in the files “hal/ubx_udpServer.cpp”.

A port number can be specified at compile time in hal/Android.mk and is typically port 46434. By default this feature is disabled.

Using this feature, it is possible to connect the full version of u-center, running on a PC, to the GNSS receiver connected to the Android device under test/development.

 Note that enabling the UDP server in the driver opens a port to the external world and is not recommended for production!

3.9 TCP Server (for debugging)

The u-blox GNSS driver also contains a TCP server which broadcasts all NMEA & UBX messages from the receiver to all authenticated TCP clients connected to the Android device on that TCP port. A TCP connected client can also send UBX messages to the receiver through the GNSS driver if they are in a list of allowed UBX messages and the client got authenticated before. By default only one TCP client is allowed to be connected to the server at any time.

The list of allowed UBX messages to be sent to the receiver by a client is defined at compile time in an array named CTcpServer::_allowed[] consisting of UBX_MSG_TYPE structs in the file “hal/ubx_tcpServer.cpp” The message and class ID of every UBX message that is intended to be allowed to be sent to the receiver by a TCP client must be stored in this array. If you enable the TCP server, at least one message must be contained in the array. By default, the only message allowed is the UBX-MON-VER message.

To increase security, the TCP server only listens to localhost, and you cannot connect externally to it. This reduces the risk of opening an unwanted security hole.

The TCP server will start in “listen-only” mode and waits for a passphrase from the client before outputting anything to it or accepting any messages. This passphrase can be configured at compile time in an array of unsigned characters named CTcpServer::_passPhrase[] in the file “hal/ubx_tcpServer.cpp”. By default the passphrase contains several entries of zeroes. This, as any passphrase only consisting of zeroes, will lead to the TCP server being disabled. It is recommended to use a passphrase of at least 16 random bytes for the passphrase.

To have the TCP server compiled in, uncomment line 79 (-DTCP_SERVER_PORT=42434) in the file “hal/Android.mk” which will enable the TCP server on port 42434.

Please note that the passphrase functionality does not introduce cryptographic-level security and introduces only a minimum level of security against a skilled attacker. Enabling the TCP server in the driver opens a local TCP port and is not recommended for production!

To enable and use the TCP server, the following steps have to be performed:

1. Enter all allowed UBX messages (CTcpServer::_allowed) in hal/ubx_tcpServer.cpp
2. Enter the passphrase (CTcpServer::_passPhrase) in hal/ubx_tcpServer.cpp
3. Enable the compiler define (-DTCP_SERVER_PORT=42434) in hal/Android.mk
4. Connect with a TCP client to the server and send the correct passphrase once. Afterwards it will be possible to send allowed and valid UBX messages to the receiver and receive all UBX and NMEA messages output by the receiver.

4 Evaluation software

4.1 u-center for Android

At the top level of the software stack lie the applications. u-blox has released a reduced version of its u-center application to run on an Android device for evaluating the GNSS receiver performance and for visualizing the location data and GNSS status published by the Android framework.

Additionally, this application allows recording of log files for subsequent analysis and debugging using the standard u-center evaluation tool.

The u-center for Android application can be freely downloaded from Google Play:

<https://play.google.com/store/apps/details?id=com.ublox.ucenter>

The user guide can be found here:



<https://www.u-blox.com/en/product/u-center-android>

5 Internet Services

5.1.1 u-blox Multiple GNSS Assistance (MGA) Online

u-blox provides a “Multiple GNSS Assistance Online” service, which allows ephemeris data to be downloaded from a server and transferred to the GNSS receiver.

The u-blox GNSS driver, which supports MGA Online, needs the following information:

-  The server address needs to be placed in the `u-blox.conf` file
-  The customer specific token needs to be placed in the `u-blox.conf` file

5.1.2 u-blox Multiple GNSS Assistance (MGA) Offline

u-blox provides a “Multiple GNSS Assistance Offline” service that combines Almanac with AlmanacPlus data in order to calculate a position fix in just a few seconds, even when ephemeris data has expired. AlmanacPlus is a differential Almanac correction that can be used to predict the gap between true satellite orbit and Almanac data.

To use MGA Offline only requires a file transfer from the server to the client. The u-blox GNSS driver supports MGA Offline, and only requires the server address URL to be placed in the `gps.conf` file.

6 GNSS Integration Steps

This section provides the developer with the information needed for a proper GNSS integration within an Android device. It is assumed that the reader has some Linux experience.

6.1 Preparation of the environment

Before building the driver, it is necessary to install the Android source code, and set up the Android build environment. The relevant documentation to do this is available here:

<http://source.android.com/source/initializing.html>

Each vendor of the available Android supported boards will typically have a board support package. The instructions for the BSP must be followed. For the HiKey620 this information can be found on the official Android website⁴.

To set up the environment, start a shell/terminal window and then do the following:

```
cd <android root>
source build/envsetup.sh
lunch
```

The u-blox GNSS driver source code needs to be inserted into the correct place in the Android source code.

```
<android root>/hardware/
```

6.2 Building and Running the GNSS driver

6.2.1 Building the driver

The driver can be built in one of two ways

- With SUPL support
- Without SUPL support

To build the u-blox GNSS driver **with** SUPL support(note - check build options):

```
cd <android root>/hardware/<u-blox release>
SUPL_ENABLED=1 mm -B
```

To build the u-blox GPS driver **without** SUPL support:

```
cd <android root>/hardware/<u-blox release>
mm -B
```

There is also a build option for enabling or disabling SUPL support in “Android.mk” file (see section 6.2.3).

6.2.2 Deployment of the driver

Dependent of the Android version in use, the deployment of the GNSS driver in system is different.

⁴ <https://source.android.com/source/devices>

6.2.2.1 In Android 8 and newer

Due to the introduction of binderized HAL's and SELinux policy applied (see section 6.3.2), the build artifacts that need to be copied to the target device fundamentally differ in Android 8 and newer. For the HiKey620, the following file transfers from build system to target device need to be performed:

- `<android_root>/out/target/product/hikey/system/bin/hw/android.hardware.gnss@1.0-service-ubx` → `<target_rootfs_folder>/system/vendor/bin/hw/`
- `<android_root>/out/target/product/hikey/system/etc/init/android.hardware.gnss@1.0-service-ubx.rc` → `<target_rootfs_folder>/system/vendor/etc/init/`
- `<android_root>/out/target/product/hikey/system/vendor/lib64/hw/android.hardware.gnss@1.0-impl-ubx.so` → `<target_rootfs_folder>/system/vendor/lib64/hw/`
- `<android_root>/out/target/product/hikey/system/vendor/lib/hw/android.hardware.gnss@1.0-impl-ubx.so` → `<target_rootfs_folder>/system/vendor/lib/hw/`

The configuration files `gps.conf` and `u-blox.conf` (see section 6.4), need to be copied to the "system/vendor/etc" folder in the target root file system. i.e.

```
<target_rootfs_folder>/system/vendor/etc
```

Then reboot the device. After reboot, the binderized GNSS driver will be loaded and the configuration files will be read.

Run the u-center for Android application to see if the receiver is working.

 In SELinux enforce mode, the whole system image needs to be built and the target device needs to be re-flashed with the new built system image, in order to make the proper SELinux settings in effect for the GNSS driver.

6.2.2.2 Versions older than Android 8

For Android versions older than 8, a library is generated at the end of the compiling process and its location displayed. For the HiKey620, this is

```
<android_root>/out/target/product/hikey/obj/lib/gps.default.so
```

This library needs to be copied to the "system/lib64/hw" folder in the target root file system of the device (respectively "system/lib/hw" on 32-bit architectures).

```
<target_rootfs_folder>/system/lib64/hw/
```

The configuration files `gps.conf` and `u-blox.conf` (see section 6.4), need to be copied to the "system/etc" folder in the target root file system. i.e.

```
<target_rootfs_folder>/system/etc
```

Then reboot the device. After reboot, the GNSS driver will be loaded and the configuration files will be read.

Run the u-center for Android application to see if the receiver is working.

6.2.2.2.1 Perl script for automating upload of legacy interface driver

In the `<android_root>/hardware/<u-blox_release>` folder there is a Perl script called "rebirth.pl". By using the Android SDK utility "adb", `rebirth.pl` does the following:

- Copies the configuration files to the target's /system/etc folder
- Copies `gps.default.so` to the target's /system/lib64/hw/ folder
- Reboot the device

To be able to use `rebirth.pl`, ADB needs to be able to contact the board. Thus the HiKey620 board must be connected by USB to the host from which the data will be uploaded.

 Note that the HiKey620 does not allow operating as a USB host and USB device at the same time.

Alternatively it is possible to use “adb connect <ip_of_the_board>” to make a connection to a HiKey620 board that is located in the same network and has its ADB server configured to listen to a TCP port. How to configure this in a permanent setup is out of scope of this application note and needs modifications to the default binary image delivered with the HiKey620

6.2.3 Android.mk Build Options

SUPL

The Android make file does not compile SUPL client code by default.

If SUPL client is required, find the following lines in “hal/Android.mk” and set SUPL_ENABLED to 1.

```
#SUPL is disabled by default
SUPL_ENABLED ?= 0
```

UDP

To add the UDP server for the debugging build, uncomment the following lines in hal/Android.mk

```
LOCAL_CFLAGS += \
    -DUDP_SERVER_PORT=46434
```

TCP

To enable the TCP server for the debugging build, uncomment the following lines in hal/Android.mk

```
LOCAL_CFLAGS += \
    -DTCP_SERVER_PORT=42434
```

6.2.4 Important u-blox.conf Options

Although most of the options in the supplied u-blox.conf file can be left unchanged, there are three that need to be set correctly in order for the GNSS driver to work with the connected receiver. They are:

- BAUDRATE_DEF
- SERIAL_DEVICE
- RECEIVER_GENERATION

BAUDRATE_DEF must be set to the receiver’s default baud rate on power up.

SERIAL_DEVICE must be set to the correct Linux hardware device receiving NMEA/UBX messages from the receiver. It is also important to make sure that the device specified by the SERIAL_DEVICE option has user read/write permissions.

RECEIVER_GENERATION must be set to the correct generation of the used receiver. This will define, beside others, which aiding method is used.

6.3 Accessing the Receiver from Android

As the driver needs to be able to communicate with the receiver, it must have the corresponding permissions to access the device file configured as SERIAL_DEVICE in u-blox.conf. For all Android versions, the file system permissions and ownership must be configured accordingly (see chapter 6.3.1). Additionally the SELinux configuration might need to be changed as well (see chapter 6.3.2).

6.3.1 File System Permissions and Ownership

On some systems, the Linux command tools “chmod” and “chown” can be used to permanently change the permissions and ownership. Then the framework can access these devices. On some devices (especially when using USB) it might be necessary to edit a file named ueventd.<device_name>.rc or

similar on the Android image to change the default permissions and ownership of the device file configured as SERIAL_DEVICE.

6.3.2 SELinux configuration

From Android 5.0 forwards, a new security mechanism named SELinux is enforced, as described on the official Android website⁵. An Android implementation with this security mechanism might prevent the GNSS driver from accessing system resources, such as a serial device, configuration files, and TCP/UDP sockets. The security mechanism needs to be correctly configured, to allow for the driver to access these resources.

The configuration for SELinux is platform specific and can usually be found in `<android_root>/device/<platform>/sepolicy` in the AOSP tree. In Android versions older than Android 8, as the Android GNSS driver is loaded as a library by the process `system_server`, the corresponding file `system_server.te` might need adjustments to set the correct permissions for the driver. In Android 8, a dedicated type enforcement file is needed for the GNSS driver. Furthermore, it might be necessary to adjust the file `file_contexts` as well. Especially if the device file through which the communication with the receiver will be established is not recognized as part of the `tty_device` group. The u-blox Android GNSS driver release package provides a reference of SELinux settings for Android 8 based on Hikey620. The referred files are under “`gps/sepolicy/`”. To take SELinux policy in effect in Android 8 and newer systems, the whole system image needs to be built with the following change in the device “`BoardConfig.mk`” file,

```
BOARD_SEPOLICY_DIRS += hardware/u-blox/gps/sepolicy
```

To evaluate the driver without such a configuration, you can disable the security mechanism temporary for testing, by issuing the command “`setenforce 0`” in the Android shell.

6.4 Configuration files

6.4.1 General

The GNSS driver uses two configuration files (in `/<android_root>/hardware/<u-blox release>`).

- `gps.conf`
- `u-blox.conf`

In order for their settings to be used, both these files need to be present in the “`\system\etc`” or “`\system\vendor\etc`” folder on the target device when the Android framework is started (either when the device is rebooted, or when the “`zygote`” process is killed and automatically restarted).

The “`rebirth.pl`” Perl script can be used to copy the configuration and GNSS driver files to their correct folders on the target, and then kill `zygote`.

The `gps.conf` file is read by the framework and any settings relevant to the GNSS driver are passed to it via the driver interface.

The `u-blox.conf` file is proprietary to u-blox and is read solely by the u-blox GNSS driver.

Any line starting with “`#`” is treated as a comment.

6.4.2 gps.conf

Example:

```
### AssistNow Offline (XTRA) Link
# For being able to configure the AssistNow Offline feature (MGA or Legacy)
```

⁵ <http://source.android.com/devices/tech/security/selinux>

```
# Two of the following XTRA_SERVER_* tokens have to be uncommented
# and the following parts of the provided URL replaced:
# 1) <token>: The token received from the u-blox for accessing the services.
# 2) <gnss>: The comma separated list of enabled GNSS in the receiver
# (e.g.: gps,glo). Make sure to insert no spaces into the URL!
# Refer to the u-blox.conf configuration file for actually
# activating AssistNow Offline and additional configuration options
# as well as enabling and configuring AssistNow Online (MGA or Legacy)

# For AssistNow MGA Offline
#XTRA_SERVER_1=http://offline-live1.services.u-
    blox.com/GetOfflineData.ashx?token=<token>;gnss=<gnss>;period=5;resolution=1
#XTRA_SERVER_2=http://offline-live2.services.u-
    blox.com/GetOfflineData.ashx?token=<token>;gnss=<gnss>;period=5;resolution=1

# For AssistNow Legacy Offline
#XTRA_SERVER_1=http://offline-live1.services.u-
    blox.com/GetOfflineData.ashx?token=<token>;format=aid;days=14
#XTRA_SERVER_2=http://offline-live2.services.u-
    blox.com/GetOfflineData.ashx?token=<token>;format=aid;days=14

### AGPS SETTINGS ###
# Secure User Plane Location (AGPS-SUPL) - TLS connection
SUPL_HOST=supl.google.com
SUPL_PORT=7275

# Secure User Plane Location (AGPS-SUPL) - non-TLS connection
#SUPL_HOST=supl.google.com
#SUPL_PORT=7276
```

Setting Descriptions

XTRA_SERVER_1 and XTRA_SERVER_2

- The framework can download a file from the u-blox MGA server and pass it to the GNSS driver. The server, the customer specific token and the GNSSs are specified with this setting.
- The validity of orbital data can vary from 1 to 14 days depending on the data file downloaded. u-blox receivers from generation 8 onwards support the MGA service and the driver must be instructed to use the configuration section “For AssistNow MGA Offline”. For all other generations, please use the legacy service and the configuration section “For AssistNow Legacy Offline”.
- To activate a periodic update of aiding information from this source, the AGNSS_STRATEGY setting in u-blox.conf must be set to include AssistNow Offline as aiding solution.

SUPL_HOST

- The name of the SUPL server to use.

SUPL_PORT

- The port number to communicate with the SUPL server.

6.4.3 u-blox.conf

Example:

```
### Serial interface
# Device to be used by the driver
SERIAL_DEVICE /dev/ttyACM0 # USB (Default interface)
#SERIAL_DEVICE /dev/ttymxc4 # UART on Sabre SD (IMX6Q)
#SERIAL_DEVICE /dev/i2c-1 # I2C on Sabre SD (IMX6Q)
#SERIAL_DEVICE /dev/ttyO3 # UART on Panda board
#SERIAL_DEVICE /dev/i2c-4 # I2C on Panda board

# If set to 1, the TX-Ready feature will be used if the
```

```
# specified SERIAL_DEVICE file is a I2C interface. This feature
# requires an unused PIO on the u-blox receiver being connected to a host GPIO that
# can be used as an interrupt-generating GPIO by the Linux Kernel.
# The further configuration options for this feature are defined
# below.
I2C_TX_READY_ENABLED          0

# The number of the PIO on the u-blox receiver which will be used as the
# TX-Ready output, if data is available on the I2C interface.
# Refer to the chapter "Peripheral Input Output (PIO)" in the
# Hardware integration guide of your product for more information
# on the PIO mapping.
I2C_TX_READY_RECV_PIO        6

# The GPIO of the host processor which is connected to the TX-Ready output pin
# of the receiver. The GPIO configured below must be exported to the sysfs,
# be configured as an input and must be able to generate interrupts. Refer
# to the file Documentation/gpio/sysfs.txt of the Linux Kernel used
# in your Android version for more information.
I2C_TX_READY_HOST_GPIO       140

# The baud rate that should be used to communicate with the receiver and at
# which the receiver will output the NMEA messages.
BAUDRATE                     115200

# The default baud rate the receiver is configured to after start-up. The
# HAL interface will configure itself to this baud rate and send a command
# to the receiver to switch to the baud rate assigned to the BAUDRATE keyword
BAUDRATE_DEF                 9600

# The generation of the receiver attached to the host device. The default
# is 7. (E.g. when using a MAX-7Q, the value for this field should be
# set to "7". For a LEO-M8N the value should be set to "8")
# This will, beside other, define if AssistNow Legacy or AssistNow MGA is
# used (if enabled).
RECEIVER_GENERATION          7

### Assistance

# Timeout for stopping the receiver [in seconds]
STOP_TIMEOUT                 10

# Which AssistNow strategies to use (Legacy and MGA)
# 0 - None <- default value
# 1 - AssistNow Autonomous only
# 2 - AssistNow Offline only
# 3 - AssistNow Online only
# 4 - AssistNow Autonomous and AssistNow Online
# 5 - AssistNow Offline and AssistNow Online
AGNSS_STRATEGY               0

# Refer to the configuration file gps.conf to
# configure AssistNow Offline (MGA or Legacy)

# AssistNow Offline data download (via Xtra) interval [In minutes]
AGNSS_OFFLINE_INTERVAL       720

# AssistNow Online data download interval [in minutes]
AGNSS_ONLINE_INTERVAL        120

# AssistNow Online and Offline Configuration (Legacy and MGA)
AGNSS_ONLINE_SERVER1         online-live1.services.u-blox.com
AGNSS_ONLINE_SERVER2         online-live2.services.u-blox.com
#AGNSS_TOKEN                  <placeholder_token>
```

```
# Save aiding data to file system when engine stops
# 0 - Don't save aiding data on the filesystem
# 1 - Save aiding data on the filesystem <- default value
AGNSS_PERSISTENCE 1

# File path and name for aiding information
AGNSS_PERSISTENCE_FILE /data/persistence.agnss

# The time source for aiding the receiver
# It is important to aid the receiver with precise time information. The
# local system can be assumed to be accurate to 10 seconds, if it is going
# to be updated by an RTC, takes leap seconds into account and the user will
# not be able to change the system time to a wrong value. If this is the case,
# it is best to use the system time for aiding.
# If the system time will, however, be lost after startup or the system time
# is not trustworthy for an other reason, it is possible to make the driver
# calculate the time itself based on NTP injects, the time received from
# the online service when downloading data or the receiver when it has a fix. This
# will however only work if there has been such a time injection since the last
# startup. Otherwise the receiver will not be aided until such an update
# occurred. Use the following time for aiding
# 0 - System time <- default value
# 1 - Driver time
AGNSS_TIME_SOURCE 0

### SUPL configuration
# SUPL requests configuration
SUPL_ALMANAC_REQUEST 0
SUPL_UTC_MODEL_REQUEST 0
SUPL_IONOSPHERIC_MODEL_REQUEST 1
SUPL_DGPS_CORRECTIONS_REQUEST 0
SUPL_REF_LOC_REQUEST 1
SUPL_REF_TIME_REQUEST 1
SUPL_AQUISITION_ASSIST_REQUEST 0
SUPL_TIME_INTEGRITY_REQUEST 0
SUPL_NAVIGATIONAL_MODEL_REQUEST 1

# The SUPL_CACERT keyword enables a TLS encryption to the SUPL server.
# If this keyword is missing, the communication will be unencrypted.
# If no value is assigned to this keyword every CA certificate will be
# accepted as valid.
# If a value is assigned to this keyword, this value will be interpreted
# as the path to a ca-certificate file.
# If the ca-certificate in this file does not match the root of the certificate
# chain the connection will be aborted.
#SUPL_CACERT /system/etc/ca-cert-google.pem
SUPL_CACERT

# Uncomment the line below to use the certificate in SUPL test server
#SUPL_CACERT /system/etc/v1_slp_rs_de_cert.pem

# Number of retries for contacting the SUPL server
SUPL_CONNECT_RETRIES 0

# Maximum time to show the dialog for network initiated SUPL session [in seconds]
SUPL_NI_UI_TIMEOUT 10

# Maximum time for sending information to the SUPL server for network
# initiated SUPL session [in seconds]
SUPL_NI_RESPONSE_TIMEOUT 75

### Debugging aids

# Option to send fake mobile phone network information
# leave this as 0 unless you are testing
```

```
SUPL_FAKE_PHONE_CONNECTION      0

# Save decoded RRLP and UPL data to logcat
SUPL_LOG_MESSAGES               0

# Create CMCC compatible A-GPS.LOG and GPS.LOG
SUPL_CMCC_LOGGING               0

# Save decoded RRLP and UPL data to file SUPL-MESSAGE.LOG
SUPL_MSG_TO_FILE                0

#####
```

Setting Descriptions

SERIAL_DEVICE

- Specifies the Linux device to communicate with. This should be the device the GNSS receiver is connected to. Refer to chapter 6.3 to learn about requirements that have to be met in order for the driver to be able to communicate with the receiver.
- Examples are
 - /dev/ttyACM0
 - /dev/ttymx4
 - /dev/i2c-1
 - /dev/tty03
 - /dev/i2c-4

I2C_TX_READY_ENABLED

- If communicating with the receiver over I2C, this setting specifies if the connected receiver is signaling available data through a GPIO. If set to a non-zero value, a GPIO will be checked for a logic high before requesting data from the receiver. This configuration field has no effect for non-I2C interfaces.

I2C_TX_READY_RECV_PIO

- The number of the PIO on the receiver which will be used as the TX-Ready output, if data is available on the I2C interface. Refer to the chapter "Peripheral Input Output (PIO)" in the Hardware integration guide of your product for more information on the PIO mapping. This configuration field has no effect for non-I2C interfaces or if I2C_TX_READY_ENABLED is set to 0.

I2C_TX_READY_HOST_GPIO

- The GPIO of the host processor which is connected to the TX-Ready output of the receiver. The GPIO configured below must be exported to the sysfs, be configured as an input and must be able to generate interrupts. Refer to the file Documentation/gpio/sysfs.txt of the Linux Kernel used in your Android version for more information. The number of the PIO on the receiver which will be used as the TX-Ready output, if data is available on the I2C interface. Refer to the chapter "Peripheral Input Output (PIO)" in the Hardware integration guide of your product for more information on the PIO mapping. This configuration field has no effect for non-I2C interfaces or if I2C_TX_READY_ENABLED is set to 0.

BAUDRATE

- If communicating with the receiver over UART, this setting specifies the baud rate for the receiver to use. This configuration field has no effect for non-UART interfaces.

BAUDRATE_DEF

- If communicating with the receiver over UART, this setting specifies the baud rate to use when the receiver is powered up (receiver default baud rate). This configuration field has no effect for non-UART interfaces.

RECEIVER_GENERATION

- Specifies the generation of the connected receiver. One of its uses is to define the aiding messages used to communicate with the receiver (AssistNow Legacy for receiver generation < 8 and MGA for receiver generation ≥ 8) if aiding is enabled in the receiver. This keyword must be set to the correct value, otherwise the driver will not work correctly.

STOP_TIMEOUT

- When the framework instructs the driver to stop the GNSS receiver, the driver interrogates the receiver for various pieces of information which may help reduce the time in getting a first fix the next time the receiver is switched on.
- This setting specifies the time in seconds to wait for the receiver to complete these requests, before shutting down.

AGNSS_STRATEGY

- This defines the AssistNow strategy to be used. The following options are possible:
 - 0 = No AssistNow enabled
 - 1 = AssistNow Autonomous only
 - 2 = AssistNow Offline only
 - 3 = AssistNow Online only
 - 4 = AssistNow Autonomous and AssistNow Online
 - 5 = AssistNow Offline and AssistNow Online
- Depending on the setting of RECEIVER_GENERATION, the correct type of messages is used. For the options relying on AssistNow Offline, the XTRA_SERVER_1 in gps.conf must be set to a valid value.

AGNSS_OFFLINE_INTERVAL

- Specifies the AssistNow Offline data download (via Xtra) interval in minutes.

AGNSS_ONLINE_INTERVAL

- Specifies the AssistNow Online data download interval in minutes.

AGNSS_ONLINE_SERVER1 and AGNSS_ONLINE_SERVER2

- Specifies the address of the AssistNow Online and Offline servers.
- When the request to the first server fails, the driver will automatically try to connect to the second address.

AGNSS_TOKEN

- Customer specific authentication token to access the MGA service.

AGNSS_PERSISTENCE

- Set to 1 to save aiding data on the file system.
- Set to 0 to not save aiding data on the file system.

AGNSS_PERSISTENCE_FILE

- Specifies the path and file name for saving the aiding information.

AGNSS_TIME_SOURCE

- Specifies the time source for aiding the receiver.
- Set to 1 to use the driver time.
- Set to 0 to use the system time.

SUPL_ALMANAC_REQUEST

- Set to 1 to request almanac assistance data during a SUPL session.
- Set to 0 if almanac not required.

SUPL_UTC_MODEL_REQUEST

- Set to 1 to request UTC model data during a SUPL session.
- Set to 0 if UTC model not required.

SUPL_IONOSPHERIC_MODEL_REQUEST

- Set to 1 to request ionospheric data during a SUPL session.
- Set to 0 if ionospheric data not required.

SUPL_DGPS_CORRECTIONS_REQUEST

- Set to 1 to request DGPS correction data during a SUPL session.
- Set to 0 if DGPS correction data not required.

SUPL_REF_LOC_REQUEST

- Set to 1 to request approximate location during a SUPL session.
- Set to 0 if location not required.

SUPL_REF_TIME_REQUEST

- Set to 1 to request a reference time during a SUPL session.
- Set to 0 if reference time not required.

SUPL_AQUISITION_ASSIST_REQUEST

- Set to 1 to request acquisition assistance data during a SUPL session.
- Set to 0 if acquisition assistance is not required.

SUPL_TIME_INTEGRITY_REQUEST

- Set to 1 to request time integrity data during a SUPL session.
- Set to 0 if time integrity not required.

SUPL_NAVIGATIONAL_MODEL_REQUEST

- Set to 1 to request ephemeris data during a SUPL session.
- Set to 0 if ephemeris not required.

SUPL_CACERT

- If communication with a SUPL server is to take place over a secure socket, then this setting specifies the certificate to use for encryption.
- If this keyword is missing, the communication will be unencrypted.
- If no value is assigned to this keyword, every CA certificate will be accepted as valid.
- Takes the form of a path and file name.

SUPL_CONNECT_RETRIES

- Set the number of times the driver should try to contact the SUPL server in case of errors.

SUPL_NI_UI_TIMEOUT

- During some types of network initiated SUPL sessions, a dialog box is displayed to the user asking for permission to find the current location. This setting specifies the maximum time (in seconds) this dialog should be displayed for, before being removed and a default action taken.

SUPL_NI_RESPONSE_TIMEOUT

- This setting specifies the maximum time (in seconds) the driver is allowed to wait before sending the SUPL server either pseudo ranges or position.

SUPL_FAKE_PHONE_CONNECTION

- If set to 1, the driver will use fake mobile phone MNC, MCC etc values. Use in testing if no mobile phone radio is present.

- Set to 0 if the mobile radio is present. The driver obtains real MNC, MCC etc from framework. Use 0 in final product.

SUPL_LOG_MESSAGES

- Set to 1 to enable logging of incoming and outgoing SUPL/RRLP messages in the Android logcat file.
- Set to 0 to disable.

SUPL_CMCC_LOGGING

- Set to 1 to enable logging of China Mobile style GPS & Assistance logging.
- Logs written to files A-GPS.LOG and GPS.LOG in /mnt/sdcard (if permissions allow) or /data (if not).
- Set to 0 to disable.

SUPL_MSG_TO_FILE

- Set to 1 to enable saving decoded RRLP and UPL data to file SUPL-MESSAGE.LOG
- Set to 0 to disable

6.5 GNSS Receiver Power

When the Android framework informs the driver that no applications are using GNSS, the driver switches off the GNSS, which puts the receiver into a low power state (just a few mA). However if the receiver is not powered via the USB bus and the customer needs to reduce the power consumption further, the driver provides software hooks to allow a GPIO controlled power supply to be switched on and off.

Two functions within the file `ubxgpsstate.cpp` are provided for customers to add code to switch on or off the receiver's external power supply, if it has one. These functions are shown below.

```

/*****
 * POWER
 *****/

//! Switch the power on to the receiver.
bool CUBxGpsState::powerOn(void)
{
    // Place any code to handle switching power to the device ON
    // HERE
    // This is optional

    UBX_LOG(LCAT_VERBOSE, "placeholder for power ON function");

    return false;    // Return true if power was successfully turned on
}

//! Switch the power off to the receiver.
void CUBxGpsState::powerOff(void)
{
    // Place any code to handle switching power to the device OFF
    // HERE
    // This is optional

    UBX_LOG(LCAT_VERBOSE, "placeholder for power OFF function");
}

```

A possible implementation of this software controlled power supply is shown in Figure 4.

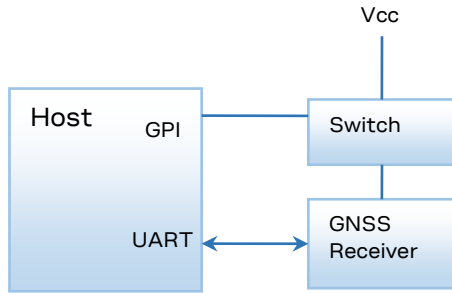


Figure 4 Scheme of software switch-off implementation

6.6 Source creation time

The GPS time provided by the RRLP protocol in the SUPL client is restricted to a maximum value of 1024 weeks. This will result in roll-over approximately 20-year intervals starting 1980. To be able to get valid time aiding for the full period before a roll-over occurs, it is required to use a starting point as reference. The value used for this is the “Source creation time” `SRC_CREAT_TIME_SEC` defined in `agnss/helper/helperTypes.h`, which is also used to verify provided dates for validity. It is crucial to update this value if the source code is deployed a significant amount of time after the official release of the software to guarantee correct time-aiding for the whole period. This date must however not be set to a time in the future.

6.7 Leap second handling

The GPS time provided by the RRLP protocol in the SUPL client does not contain leap second information. To keep the time aiding information precise, it is necessary to update the `LEAP_SEC[]` array in `agnss/helper/helperTypes.h` upon every introduction of an additional leap second. All entries of this array consist of a Unix timestamp of the second before the leap second was introduced.

6.8 Hardware information

6.8.1 u-blox Receiver Series

The integration above uses a u-blox M8 GNSS receiver, although u-blox 7 and 6 receivers are compatible with the driver.

6.8.2 Further Information

Describing the hardware integration is beyond the scope of this document. The necessary documentation is listed in the “Related documents” section at the end of this document.

For hardware integration considerations see one of the following hardware integration manual:

- MAX-7 / NEO-7 Hardware Integration Manual [2]
- MAX-M8 Hardware Integration Manual [3]
- NEO-M8 Hardware Integration Manual [4]


For firmware considerations see one of the following receiver descriptions and protocol specifications:

- u-blox 7 Receiver Description including Protocol Specification [5]
- u-blox 8 / u-blox M8 Receiver Description Including Protocol Specification [6]

For an overview about the latest GNSS solutions for Android by u-blox, refer to the publicly available *GNSS Solution for Android Product Summary* [7].

Related documents

- [1] u-center Android User Guide, Docu. No. UBX-15017010
- [2] MAX-7 / NEO-7 Hardware Integration Manual, Docu. No. UBX-13003704
- [3] MAX-M8 Hardware Integration Manual, Docu. No. UBX-13004876
- [4] NEO-M8 Hardware Integration Manual, Docu. No. UBX-13003557
- [5] u-blox 7 Receiver Description and Protocol Specification V14, Docu. No. GPS.G7-SW-12001
- [6] u-blox 8 / u-blox M8 Receiver Description / Protocol Specification, Docu. No. UBX-13003221
- [7] u-blox Android GNSS Solution Product Summary, Docu. No. UBX-15016573

 For regular updates to u-blox documentation and to receive product change notifications, register on our homepage (www.u-blox.com).

Revision history

Revision	Date	Name	Status / Comments
R01	15-Nov-2018	yzha	Content Updated.

Contact

For complete contact information, visit us at www.u-blox.com.

u-blox Offices

North, Central and South America

u-blox America, Inc.

Phone: +1 703 483 3180
E-mail: info_us@u-blox.com

Regional Office West Coast:

Phone: +1 408 573 3640
E-mail: info_us@u-blox.com

Technical Support:

Phone: +1 703 483 3185
E-mail: support@u-blox.com

Headquarters

Europe, Middle East, Africa

u-blox AG

Phone: +41 44 722 74 44
E-mail: info@u-blox.com
Support: support@u-blox.com

Asia, Australia, Pacific

u-blox Singapore Pte. Ltd.

Phone: +65 6734 3811
E-mail: info_ap@u-blox.com
Support: support_ap@u-blox.com

Regional Office Australia:

Phone: +61 2 8448 2016
E-mail: info_au@u-blox.com
Support: support_ap@u-blox.com

Regional Office China (Beijing):

Phone: +86 10 68 133 545
E-mail: info_cn@u-blox.com
Support: support_cn@u-blox.com

Regional Office China (Chongqing):

Phone: +86 23 6815 1588
E-mail: info_cn@u-blox.com
Support: support_cn@u-blox.com

Regional Office China (Shanghai):

Phone: +86 21 6090 4832
E-mail: info_cn@u-blox.com
Support: support_cn@u-blox.com

Regional Office China (Shenzhen):

Phone: +86 755 8627 1083
E-mail: info_cn@u-blox.com
Support: support_cn@u-blox.com

Regional Office India:

Phone: +91 80 405 092 00
E-mail: info_in@u-blox.com
Support: support_in@u-blox.com

Regional Office Japan (Osaka):

Phone: +81 6 6941 3660
E-mail: info_jp@u-blox.com
Support: support_jp@u-blox.com

Regional Office Japan (Tokyo):

Phone: +81 3 5775 3850
E-mail: info_jp@u-blox.com
Support: support_jp@u-blox.com

Regional Office Korea:

Phone: +82 2 542 0861
E-mail: info_kr@u-blox.com
Support: support_kr@u-blox.com

Regional Office Taiwan:

Phone: +886 2 2657 1090
E-mail: info_tw@u-blox.com
Support: support_tw@u-blox.com