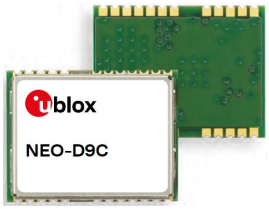
u-blox D9 QZS 1.01

u-blox D9 QZSS correction service receiver

Interface description



Abstract
This document describes the interface (version 26.00) of the NEO-D9C QZSS correction service receiver.

www.u-blox.com

UBX-21031777 - R02
C1-Public

Document information

Title	**u-blox D9 QZS 1.01**	
Subtitle	u-blox D9 QZSS correction service receiver	
Document type	Interface description	
Document number	UBX-21031777	
Revision and date	R02	26-Feb-2024
Disclosure restriction	C1-Public	

Contents

1 General information

1.1 Document overview

This document describes the interface of the u-blox D9 QZSS correction service receiver. The interface consists of the following parts:

- UBX protocol
- Configuration interface

☞ Some of the features described here may not be available in the receiver, and some may require specific configurations to be enabled. See the applicable data sheet for availability of the features and the integration manual for instructions for enabling them.

☞ Previous versions of u-blox receiver documentation combined general receiver description and interface specification. In the current documentation the receiver description is included in the integration manual.

See also Related documents.

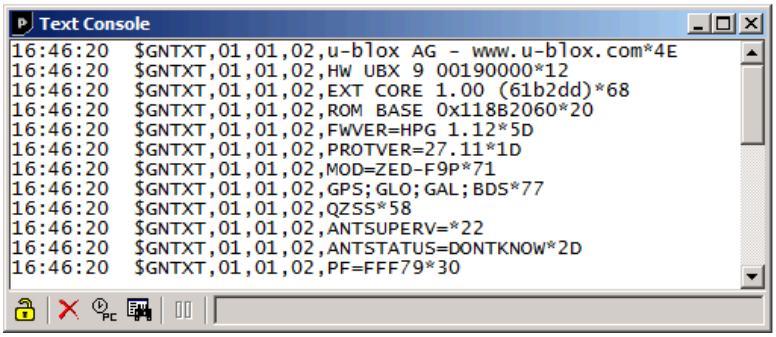
1.2 Firmware and protocol versions

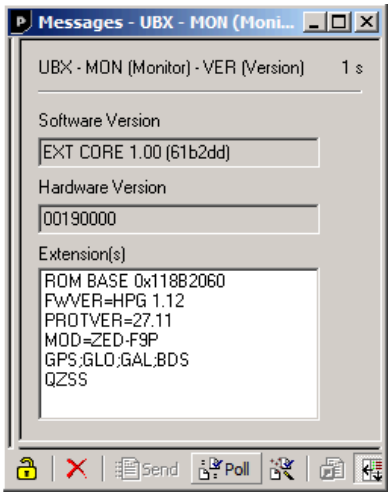
u-blox receivers execute firmware from internal ROM or load an external image and execute it from internal code-RAM.

- If the product does not have internal code-RAM, the firmware runs from the ROM.
- If the product has internal code-RAM but an external image is not available, the firmware runs from the ROM. Some products have only limited ROM and enter boot mode with no GNSS function if an external image is not available.
- If the external firmware image is stored in a flash memory, it is loaded into the code-RAM before execution.
- In some products, the firmware image can be stored in the host system and loaded into the code-RAM from there.

The location and the version of the currently running firmware can be found in the boot screen and in the UBX-MON-VER message. If the firmware has been loaded from the flash memory or from the host processor, it is indicated by text "EXT". Running from the internal ROM is indicated by text "ROM". When the receiver is started, the boot screen is output automatically in UBX-INF-NOTICE or NMEA-Standard-TXT messages if configured using CFG-INFMSG. The UBX-MON-VER message can be polled using the UBX polling mechanism.

The following u-center screenshots show an example of boot information:





The following information is available (✓) from the boot screen (**B**) and the UBX-MON-VER message (**M**):

B	M	Example	Information
✓		`u-blox AG - www.u-blox.com`	Start of the boot screen.
✓		`HW UBX 10 00000000`	Hardware version of the u-blox receiver.
	✓	`00000000`	
✓	✓	`ROM SPG 5.10 (000000)`	Firmware version and revision identifier.
✓	✓	`ROM BASE 0x118B2060`	Revision of the underlying boot loader firmware in ROM.
✓	✓	`FWVER=SPG 5.10`	Product firmware version, where: • `SPG` = Standard precision GNSS product • `HPG` = High precision GNSS product • `ADR` = Automotive dead reckoning product • `TIM` = Time sync product • `LAP` = Lane accurate positioning product • `HPS` = High precision sensor fusion product • `DBS` = Dual band standard precision • `MDR` = Multi-mode dead reckoning product • `PMP` = L-Band Inmarsat point-to-multipoint receiver • `QZS` = QZSS L6 centimeter level augmentation service (CLAS) message receiver • `DBD` = Dual band dead reckoning product • `LDR` = ROM bootloader, no GNSS functionality
✓	✓	`PROTVER=34.00`	Supported protocol version.
✓	✓	`MOD=EVK-M101`	Module name.
✓	✓	`GPS;GLO;GAL;BDS`	List of supported major GNSS (see GNSS identifiers).
✓	✓	`SBAS;QZSS`	List of supported augmentation systems (see GNSS identifiers).
✓		`ANTSUPERV=AC SD PDoS SR`	Configuration of the antenna supervisor, where: • `AC` = Active antenna control enabled • `SD` = Short circuit detection enabled • `OD` = Open circuit detection enabled • `PDoS` = Short circuit power down logic enabled • `SR` = Automatic recovery from short state enabled
✓		`PF=FFF79`	Product configuration.

B	M	Example	Information
✓		`BD=E01C`	GNSS band configuration.

☞ The "FWVER" product firmware version indicates which firmware is currently running. This is referred to as "firmware version" in this and other documents.

☞ The version and revision numbers should only be used to identify a known firmware version. They are not necessarily numeric nor are they guaranteed to increase with later firmware versions.

☞ All u-blox receivers output the start text, hardware version, and firmware version and revision. Some of the other entries in the boot screen example may be omitted.

The product firmware version and revision relate to the protocol version:

Firmware version	Version and revision identifier	Protocol version
QZS 1.01	EXT CORE 1.00 (d716bf)	26.00

1.3 Receiver configuration

u-blox positioning receivers are fully configurable with UBX protocol messages. The configuration used by the receiver during normal operation is called the "current configuration". The current configuration can be changed during normal operation by sending UBX-CFG-VALSET messages over any I/O port. The receiver changes its current configuration immediately after receiving a configuration message. The receiver always uses the current configuration only.

The current configuration is loaded from permanent configuration hard-coded in the receiver firmware (the defaults) and from non-volatile memory (user configuration) on startup of the receiver. Changes made to the current configuration at run-time will be lost when there is a power cycle, a hardware reset or a (complete) controlled software reset (see Configuration reset behavior).

See Configuration interface for a detailed description of the receiver configuration system, the explanation of the configuration concept and its principles and interfaces.

☞ The configuration interface has changed from earlier u-blox positioning receivers. There is some backwards compatibility provided in UBX-CFG configuration messages. Users are strongly advised to only use the Configuration interface. See also Legacy UBX message fields reference.

☞ See the integration manual for a basic receiver configuration most commonly used.

1.4 Message naming

Message names are written in full with the parts of the name separated by hyphens ("-"). The full message name consists of the protocol name (e.g. *UBX)*, the class name (e.g. *NAV*) and the message name (e.g. *PVT*). For example, the receiver software version information message is referred to as *UBX-MON-VER*.

References to fields of the message add the field name separated by a dot (".), e.g. *UBX-MON-VER.swVersion*.

Some messages use a fourth level of naming, called the message version. One example is the *UBX-MGA-GPS* message for GPS assistance data, which exists in versions for ephemerides (*UBX-MGA-GPS-EPH*) and almanacs (*UBX-MGA-GPS-ALM*).

Names of configuration items are of the form *CFG-GROUP-ITEM*. For example, *CFG-NAVSPG-DYNMODEL* refers to the navigation dynamic platform model the receiver uses. Constants add a fourth level to the item name, such as *CFG-NAVSPG-DYNMODEL-AUTOMOT* for the automotive

platform model. In the context of describing an item's value, only the last part of the constant name can be used (e.g. "set *CFG-NAVSPG-DYNMODEL* to *PORT* for portable applications").

1.5 GNSS, satellite, and signal identifiers

1.5.1 Overview

Many UBX protocol messages contain infomation about specific satellites. Any single satellite can be identified by a `gnssId` field indicating the GNSS the satellite is part of and an `svId` (SV for space vehicle) field indicating the number of the satellite in that system. Usually, the `svId` is the native number associated with the satellite in the specific GNSS. For example, the Galileo SV4 is identified as gnssId 2, svId 4, while the GPS SV4 is gnssId 0, svId 4.

Some legacy UBX protocol messages combine both the satellite number and the GNSS identification into a one-byte (type U1) field. See the single svid mapping in Satellite identifiers to identify the corresponding GNSS and satellite.

GLONASS satellites can be tracked before they have been identified. In UBX messages, the unknown satellites are reported with svId 255. Product-related documentation and u-center use R? to label unidentified GLONASS satellites.

Signal identifiers are used when different signals from the same GNSS satellite need to be distinguished (e.g. in the UBX-NAV-SIG message). A separate `sigId` field identifies the signal. These signal identifiers are only valid when combined with a GNSS identifier (`gnssId` field).

☞ Note that the following sections are a generic overview for different u-blox positioning receivers. A particular product may not support all of the described GNSS identifiers, satellite numbers, signal identifiers or combinations thereof.

1.5.2 GNSS identifiers

Table 1 lists each GNSS along with the GNSS identifier (UBX protocol), and abbreviations used in this document:

GNSS	Abbreviations		UBX gnssId
GPS	GPS	G	0
SBAS	SBAS	S	1
Galileo	GAL	E	2
BeiDou	BDS	B	3
QZSS	QZSS	Q	5
GLONASS	GLO	R	6
NavIC	NavIC	N	7

Table 1: GNSS identifiers

1.5.3 Satellite identifiers

The satellite numbering scheme for the UBX protocol is provided in Table 2.

GNSS	SV Range	gnssId:svId	single svid
GPS	G1-G32	0:1-32	1-32
SBAS	S120-S158	1:120-158	120-158
Galileo	E1-E36	2:1-36	211-246
BeiDou	B1-B5	3:1-5	159-163

GNSS	SV Range	gnssId:svId	single svid
	B6-B37	3:6-37	33-64
	B38-B63	3:38-63	n/a
QZSS	Q1-Q10	5:1-10	193-202
GLONASS	R1-R32	6:1-32	65-96
	R?	6:255	255
NavIC	N1-N7	7:1-7	247-253
	N8-N14	7:8-14	n/a

Table 2: UBX protocol satellite numbering scheme

1.5.4 Signal identifiers

A summary of all the signal identification schemes used in the UBX protocol is provided in the following table. (Only a subset of the signals is supported by each product.)

	UBX Protocol	
Signal	**gnssId**	**sigId**
GPS L1C/A[1]	0	0
GPS L2 CL	0	3
GPS L2 CM	0	4
GPS L5 I	0	6
GPS L5 Q	0	7
SBAS L1C/A[1]	1	0
Galileo E1 C[1]	2	0
Galileo E1 B[1]	2	1
Galileo E5 aI	2	3
Galileo E5 aQ	2	4
Galileo E5 bI	2	5
Galileo E5 bQ	2	6
BeiDou B1I D1[1]	3	0
BeiDou B1I D2[1]	3	1
BeiDou B2I D1	3	2
BeiDou B2I D2	3	3
BeiDou B1 Cp (pilot)	3	5
BeiDou B1 Cd (data)	3	6
BeiDou B2 ap (pilot)	3	7
BeiDou B2 ad (data)	3	8
QZSS L1C/A[1]	5	0
QZSS L1S	5	1
QZSS L2 CM	5	4
QZSS L2 CL	5	5
QZSS L5 I	5	8
QZSS L5 Q	5	9
GLONASS L1 OF[1]	6	0

[1] UBX messages that do not have an explicit `sigId` field contain information about the subset of signals marked.

	UBX Protocol	
Signal	**gnssId**	**sigId**
GLONASS L2 OF	6	2
NavIC L5 A[1]	7	0

Table 3: Signal identifiers

1.6 Message types

The following message types are defined:

Message type	Description
Input	Messages that are input to the receiver and never output. E.g. UBX-MGA-GPS-EPH.
Output	Messages that are output by the receiver in no particular interval and never input. E.g. UBX-ACK-ACK.
Input/output	Messages that can be output by or input to the receiver. E.g. UBX-MGA-DBD-DATA0.
Periodic	Messages that are output in regular intervals but cannot be polled. E.g. UBX-NAV-EOE.
Periodic/polled	Messages that are output in regular intervals and can be polled. E.g. UBX-NAV-PVT.
Command	Messages that are a command to the receiver. Similar to type *Input* these are input-only. E.g. UBX-CFG-RST.
Get	Output-only configuration or command messages. E.g. UBX-CFG-DAT.
Set	Input-only configuration or command messages. E.g. UBX-CFG-VALDEL.
Get/set	Input/output configuration or command messages. E.g. UBX-CFG-NAVX5.
Polled	Non-periodic messages that can only be polled. E.g. UBX-MON-VER.
Poll request	Poll request. E.g. UBX-MGA-DBD-POLL.

2 UBX protocol

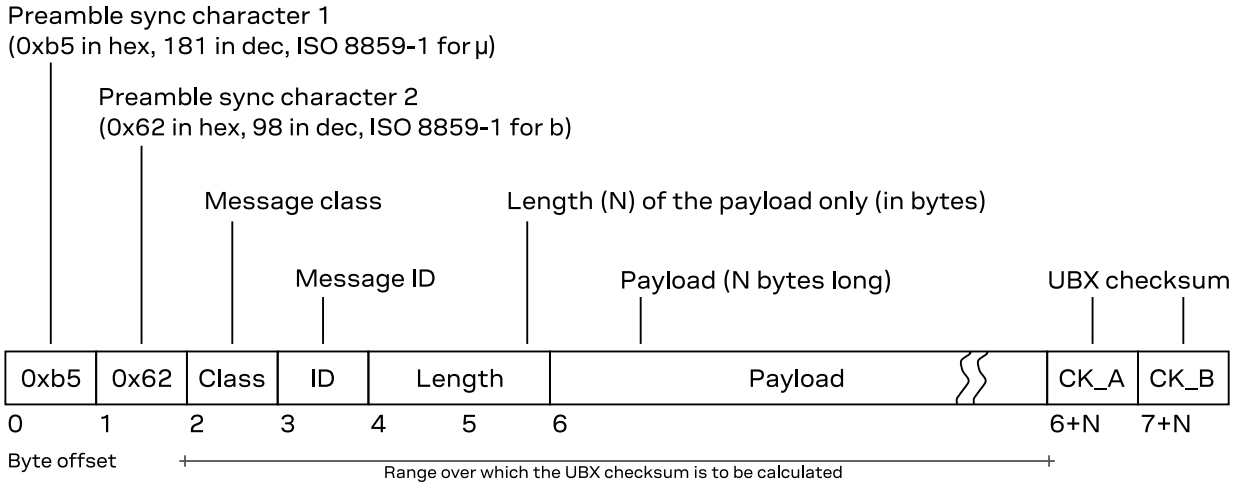
2.1 UBX protocol key features

u-blox receivers support a u-blox-proprietary protocol to communicate with a host computer. This protocol has the following key features:

- Compact – uses 8-bit binary data
- Checksum protected – uses a low-overhead checksum algorithm
- Modular – uses a two-stage message identifier (Class and Message ID)

2.2 UBX frame structure

The structure of a basic UBX frame is shown in the following diagram.



- Every *frame* starts with a 2-byte *preamble* consisting of two synchronization characters: `0xb5` and `0x62`.
- A 1-byte *message class* field follows. A class is a group of messages that are related to each other.
- A 1-byte *message ID* field defines the message that is to follow.
- A 2-byte *length* field follows. The length is defined as being that of the payload only. It does not include the preamble, message class, message ID, length, or UBX checksum fields. The number format of the length field is an unsigned little-endian 16-bit integer (a "U2" in UBX data types).
- The *payload* field contains a variable number (= *length*) of bytes.
- The two 1-byte *CK_A* and *CK_B* fields hold a 16-bit checksum whose calculation is defined in UBX checksum section. This concludes the frame.

2.3 UBX payload definition rules

This section contains the rules and guidelines for UBX message payloads. See also UBX message example.

2.3.1 UBX structure packing

Values are placed in such an order that structure packing is not a problem. This means that two-byte values shall start on offsets that are a multiple of two; four-byte values shall start at a multiple of four; and so on.

2.3.2 UBX reserved elements

Some messages contain reserved fields or bits to allow for future expansion. The contents of these elements should be ignored in output messages and must be set to zero in input messages. Where a message is output and subsequently returned to the receiver as an input message, reserved elements can either be explicitly set to zero or left with whatever value they were output with.

For fields in a bitfield the same rules apply. Note that bits not described are automatically reserved and are not explicitly stated (see UBX message example).

2.3.3 UBX undefined values

The description of some fields provide specific meanings for specific values. For example, the field `gnssId` appears in many UBX messages and uses 0 to indicate GPS, 1 for SBAS and so on (see GNSS identifiers for details); however it is usually stored in a byte with far more possible values than the handful currently defined. All such undefined values are reserved for future expansion and therefore should not be used.

2.3.4 UBX conditional values

Some UBX messages use validity flag fields to indicate whether the values of some value fields are valid. For example, the UBX-NAV-PVT message has the `validDate` and `validTime` fields that indicate whether the date (`year`, `month` and `day` fields), and, respectively, the time (`hour`, `min` and `sec` fields) are valid. This means that these value fields will only contain meaningful data if the corresponding flag field is set (has the value 1).

2.3.5 UBX data types

The following data types (number formats) are defined.

Name	Type	Size (Bytes)	Range	Resolution
U1	unsigned 8-bit integer	1	$0...2^8-1$	1
I1	signed 8-bit integer, two's complement	1	$-2^7...2^7-1$	1
X1	8-bit bitfield	1	n/a	n/a
U2	unsigned little-endian 16-bit integer	2	$0...2^{16}-1$	1
I2	signed little-endian 16-bit integer, two's complement	2	$-2^{15}...2^{15}-1$	1
X2	16-bit little-endian bitfield	2	n/a	n/a
U4	unsigned little-endian 32-bit integer	4	$0...2^{32}-1$	1
I4	signed little-endian 32-bit integer, two's complement	4	$-2^{31}...2^{31}-1$	1
X4	32-bit little-endian bitfield	4	n/a	n/a

Name	Type	Size (Bytes)	Range	Resolution
R4	IEEE 754 single (32-bit) precision	4	$-2^{127}...2^{127}$	~ value·2^{-24}
R8	IEEE 754 double (64-bit) precision	8	$-2^{1023}...2^{1023}$	~ value·2^{-53}
CH	ASCII / ISO 8859-1 char (8-bit)	1	n/a	n/a
$U_{:n}$	unsigned bitfield value of *n* bits width	var.	variable	variable
$I_{:n}$	signed (two's complement) bitfield value of *n* bits width	var.	variable	variable
$S_{:n}$	signed bitfield value of *n* bits width, in sign (most significant bit) and magnitude (remaining bits) notation	var.	variable	variable

2.3.6 UBX fields scale and unit

Fields in UBX messages can have a unit defined. Whenever possible, SI units and symbols are used (e.g. "m" for meters, "s" for seconds). For civil (UTC) time representation units of years (y), months (month), days (d), hours (h), minutes (min) and seconds (s) are used.

Fields in UBX messages can have a scale factor defined. Unity (factor 1) is assumed if no scale is specified. For integer type fields this is often combined with a unit. When a scale is combined with a unit, the scale represents the smallest storage unit. For example, if meters (m) are expressed (stored) in centimeters the scale would be 0.01 (or 1e-2). This is equivalent of specifying a unit of centimeters (cm) and no scale.

The description of some integer values (e.g. U2, I4 or I8) indicates a fixed-point format (e.g. [UU.FF], [IIIII.FFF] or [IIIIIII.FFFFFFFF]). The fixed-point value can be retrieved from the integer value by first casting it to appropriate type (e.g. as a floating-point number) and then scaling it with the indicated scaling factor.

2.3.7 UBX repeated fields

There are two types of repetitions in UBX messages. The first type specifies that a single field is repeated a constant number of times. This repetition is defined in the type of the field. For example, the UBX message example can specify a field `data` of type *U1[5]*. In this case the `data` field should be interpreted as an array of five U1 values.

The second type of repetition in messages is referred to as *repeated groups*, which groups one or more fields into a block of payload data. There are several types of repetition:

- The number of repetitions of *variable-by-field group* is indicated by another, earlier field in the same message. The number of repetitions can be zero or more, depending on the value of the referenced field.
- A *constant group* has a constant number of repetitions.
- An *optional group* is repeated zero or one times, depending on the available payload data. That is, the fields are present in the message only if the payload of the message is large enough to cover the whole group of fields.
- The number of repetitions of a *variable-by-size* group is given by the available payload size. The group will repeat until there is not enough payload data left to cover the whole group of fields another time.

Note that only some combinations of repeated groups of fields are possible in a single message. See also UBX payload decoding.

2.3.8 UBX payload decoding

UBX message payloads are designed so that the data (fields) can be extracted by a single pass through the payload from start to end. Fixed-size messages are the trivial case where the offset of all fields is unambiguously defined. Variable-size messages have variable number of repetitions of one or multiple groups of fields. For groups where the number of repetitions is given by the value of another field, that field can always be found at a fixed offset in the message payload before the respective group of fields. Groups whose number of repetitions depend on the payload size can only be the last group of fields in a message and only one such group may exist in a message. See also UBX repeated fields.

2.4 UBX checksum

The checksum is calculated over the message, starting and including the class field up until, but excluding, the checksum fields (see the figure UBX frame structure).

The checksum algorithm used is the 8-bit Fletcher algorithm, which is used in the TCP standard RFC 1145). This algorithm works as follows:

- `Buffer[N]` is an array of bytes that contains the data over which the checksum is to be calculated.
- The two `CK_A` and `CK_A` values are 8-bit unsigned integers, only! If implementing with larger-sized integer values, make sure to mask both `CK_A` and `CK_B` with the value `0xff` after both operations in the loop.
- After the loop, the two *U1* values contain the checksum, transmitted after the message payload, which concludes the frame.

```
CK_A = 0, CK_B = 0
For (I = 0; I < N; I++)
{
    CK_A = CK_A + Buffer[I]
    CK_B = CK_B + CK_A
}
```

2.5 UBX message flow

There are certain features associated with the messages being sent back and forth:

2.5.1 UBX acknowledgement

When messages from the class CFG are sent to the receiver, the receiver will send an "acknowledge" (UBX-ACK-ACK) or a "not acknowledge" (UBX-ACK-NAK) message back to the sender, depending on whether or not the message was processed correctly.

Some messages from other classes also use the same acknowledgement mechanism.

2.5.2 UBX polling mechanism

The UBX protocol is designed so that messages can be polled by sending the message required to the receiver but without a payload (or with just a single parameter that identifies the poll request). The receiver then responds with the same message with the payload populated.

2.6 GNSS, satellite, and signal numbering

See GNSS, satellite, and signal identifiers for details on how GNSS, satellites and signals are numbered in the UBX protocol.

2.7 UBX message example

This is an example of the definition of UBX messages as shown in the following sections.

Message ❶	**UBX-DEMO-EXAMPLE** **Example demo message**				
Type ❷	Periodic/polled				
Comment ❸	This is a comment that describes the use of the demo example message. There can be references to other sections in the documentation (such as: UBX protocol). ☞ Note that there can be important remarks here.				
Message ❹	*Header*	*Class ID*	*Length (bytes)*	*Payload*	*Checksum*
Structure	0xb5 0x62	0x01 0x07	16 + numRepeat*4	*see below*	CK_A CK_B

Payload description: ❺

Byte offset	*Type*	*Name*	*Scale*	*Unit*	*Description*
0	U4	`aField`	-	-	a field that contains an unsigned integer with no particular scale or unit
4	I4	`anotherField`	1e-2	m	a field that contains a length in meters (m) with a scale of 1e-2 (= 0.01), i.e. a length in centimeters
8	X2	`bitfield` ❻	-	-	this field contains flags or values smaller than one byte, whose definition follows below (bits not described are reserved)
bit 0	U:1	`aFieldValid`	-	-	the first bit in `bitfield` indicates whether the `aField` is valid or not (see UBX conditional values)
bit 1	U:1	`someFlag`	-	-	the second bit is a flag (1 = true, 0 = false)
bits 5...2	U:4	`aBitFieldValue`	-	-	a 4-bits value (range: 0...15)
10	U1[5] ❼	`reserved0`	-	-	a reserved field, whose value shall be ignored (in output messages) or set to 0 (in input messages)
15	U1	`numRepeat`	-	-	number of repetitions in the group of fields below
Start of repeated group (`numRepeat` times) ❽					
16 + n*4	I2	`someValue`	-	-	a signed value in a repeated group of fields
18 + n*4	U2	`anotherValue`	-	-	another value in a repeated group of fields
End of repeated group (`numRepeat` times)					

❶ The first line shows the message name (see Message naming). The second line shows a short description of the message.

❷ The message type (see Message types).

❸ This section contains comments that describe the message. Often links to other related sections in the documentation or other related messages are found here.

❹ The message structure gives the parameters for the UBX frame structure, notably the message class and message ID values and the payload length. For many messages the payload length is a fixed number (of bytes). Messages that contain repeated blocks of information (fields) have a variable payload (see UBX repeated fields).

❺ The message payload definition is given as a list of fields and their parameters. Each field starts at a specified offset (in bytes) in the payload (see also UBX structure packing), is of a specific type (see UBX data types), has a unique name (within the message), and a description. Optionally, fields can have a scale and/or a unit (see UBX fields scale and unit).

❻ Bitfields ("X" types) are broken down into smaller parts. Each part can be one or more bits wide. Values that are two or more bits wide can be unsigned or one of two signed value representation (see UBX data types). Note that the ten unused bits 15...6 are not explicitly stated as UBX reserved elements.

❼ Fields can be arrays of values of the same type (see UBX repeated fields).

❽ Groups of fields can be repeated in the payload. The number of repetitions can be given by another field in the message (this example), a constant number, zero or one times (known as "optional group"), or derived from the remaining payload size (labeled as "repeated N times"). See also UBX repeated fields and UBX payload decoding.

2.8 UBX messages overview

Message	*Class/ID*	*Description (Type)*
UBX-ACK – Acknowledgement and negative acknowledgement messages		
UBX-ACK-ACK	0x05 0x01	• Message acknowledged (Output)
UBX-ACK-NAK	0x05 0x00	• Message not acknowledged (Output)
UBX-CFG – Configuration and command messages		
UBX-CFG-VALDEL	0x06 0x8c	• Delete configuration item values (Set) • Delete configuration item values (with transaction) (Set)
UBX-CFG-VALGET	0x06 0x8b	• Get configuration items (Poll request) • Configuration items (Polled)
UBX-CFG-VALSET	0x06 0x8a	• Set configuration item values (Set) • Set configuration item values (with transaction) (Set)
UBX-MON – Monitoring messages		
UBX-MON-VER	0x0a 0x04	• Poll receiver and software version (Poll request) • Receiver and software version (Polled)
UBX-NAV – Navigation solution messages		
UBX-NAV-PVT	0x01 0x07	• Navigation position velocity time solution (Periodic/polled)
UBX-NAV-TIMEGPS	0x01 0x20	• GPS time solution (Periodic/polled)
UBX-RXM – Receiver manager messages		
UBX-RXM-QZSSL6	0x02 0x73	• QZSS L6 message (Periodic)
UBX-RXM-SFRBX	0x02 0x13	• Broadcast navigation data subframe (Output)

2.9 UBX-ACK (0x05)

The messages in the UBX-ACK class are used to indicate acknowledgement or rejection (i.e. negative acknowledgement) of input messages, such as UBX-CFG messages.

2.9.1 UBX-ACK-ACK (0x05 0x01)

2.9.1.1 Message acknowledged

Message	**UBX-ACK-ACK** **Message acknowledged**					
Type	Output					
Comment	Output upon processing of an input message. A UBX-ACK-ACK is sent as soon as possible but at least within one second.					
Message structure	*Header*	*Class*	*ID*	*Length (Bytes)*	*Payload*	*Checksum*
	0xb5 0x62	0x05	0x01	2	*see below*	CK_A CK_B

Payload description:

Byte offset	*Type*	*Name*	*Scale*	*Unit*	*Description*
0	U1	`clsID`	-	-	Class ID of the Acknowledged Message
1	U1	`msgID`	-	-	Message ID of the Acknowledged Message

2.9.2 UBX-ACK-NAK (0x05 0x00)

2.9.2.1 Message not acknowledged

Message	**UBX-ACK-NAK** **Message not acknowledged**					
Type	Output					
Comment	Output upon processing of an input message. A UBX-ACK-NAK is sent as soon as possible but at least within one second.					
Message structure	*Header*	*Class*	*ID*	*Length (Bytes)*	*Payload*	*Checksum*
	0xb5 0x62	0x05	0x00	2	*see below*	CK_A CK_B

Payload description:

Byte offset	*Type*	*Name*	*Scale*	*Unit*	*Description*
0	U1	`clsID`	-	-	Class ID of the Not-Acknowledged Message
1	U1	`msgID`	-	-	Message ID of the Not-Acknowledged Message

2.10 UBX-CFG (0x06)

The messages in the UBX-CFG class are used to configure the receiver and poll current configuration values as well as for sending commands to the receiver. Unless stated otherwise, any message in this class sent to the receiver is either acknowledged (by a UBX-ACK-ACK message) if processed successfully or rejected (with a UBX-ACK-NAK message) if processed unsuccessfully.

2.10.1 UBX-CFG-VALDEL (0x06 0x8c)

2.10.1.1 Delete configuration item values

Message	**UBX-CFG-VALDEL** **Delete configuration item values**
Type	Set
Comment	Overview: • This message can be used to delete saved configuration to effectively revert the item values to defaults. • This message can delete saved configuration from the flash configuration layer and the BBR configuration layer. The changes will not be effective until these layers are loaded into the RAM layer. • This message is limited to containing a maximum of 64 keys up for deletion; i.e. N is a maximum of 64. • This message can be used multiple times and every time the result will be applied immediately. To send this message multiple times with the result being applied at the end, see version 1 of UBX-CFG-VALDEL that supports transactions.

- This message does not check if the resulting configuration is valid.
- See Receiver configuration for details.

This message returns a UBX-ACK-NAK and no configuration is applied:

- if any key is unknown to the receiver FW
- if the layer's bitfield does not specify a layer to delete a value from.

Notes:

- If a key is sent multiple times within the same message, the value is effectively deleted only once.
- Attempting to delete items that have not been set before, or that have already been deleted, is considered a valid request.
- The provided keys can be complete key values (group and item specifiers) or wild-card specifications. A complete key value constitutes a deletion request for one key-value pair. A key value with a valid group specifier and 0xffff in the item part of the key value (bits 0-15) constitutes a deletion request for all items in the specified group. A key with a value of 0xfff in the group part of the key value (bits 16-27) is a deletion request for all items known to the receiver in all groups.

Message structure	*Header*	*Class*	*ID*	*Length (Bytes)*	*Payload*	*Checksum*
	0xb5 0x62	0x06	0x8c	4 + [0..n]·4	*see below*	CK_A CK_B

Payload description:

Byte offset	*Type*	*Name*	*Scale*	*Unit*	*Description*
0	U1	`version`	-	-	Message version (0x00 for this version)
1	X1	`layers`	-	-	The layers where the configuration should be deleted from
bit 1	U:1	`bbr`	-	-	Delete configuration from the BBR layer
bit 2	U:1	`flash`	-	-	Delete configuration from the Flash layer
2	U1[2]	`reserved0`	-	-	Reserved
Start of repeated group (N times)					
4 + n·4	U4	`keys`	-	-	Configuration key IDs of the configuration items to be deleted
End of repeated group (N times)					

2.10.1.2 Delete configuration item values (with transaction)

Message	**UBX-CFG-VALDEL** **Delete configuration item values (with transaction)**
Type	Set
Comment	Overview: • This message can be used to delete saved configuration to effectively revert them to defaults. • This message can delete saved configuration from the flash configuration layer and the BBR configuration layer. The changes will not be effective until these layers are loaded into the RAM layer. • This message is limited to containing a maximum of 64 keys up for deletion; i.e. N is a maximum of 64. • This message can be used multiple times with the result being managed within a transaction. • This message does not check if the resulting configuration is valid. • See Receiver configuration for details. • See version 0 of UBX-CFG-VALDEL for simplified version of this message. This message returns a UBX-ACK-NAK, cancels any started transaction, and no configuration is applied: • if any key within a transaction is unknown to the receiver FW • if an invalid transaction state transition is requested • if the layer's bitfield changes within a transaction • if the layer's bitfield does not specify a layer to delete a value from. Notes: • Any request for another UBX-CFG- message type (including UBX-CFG-VALSET and UBX-CFG-VALGET) will cancel any started transaction, and no configuration is applied. • This message can be sent with no keys to delete for the purposes of managing the transaction state transition. • If a key is sent multiple times within the same message or within the same transaction, the value is effectively deleted only once.

- Attempting to delete items that have not been set before, or that have already been deleted, is considered a valid request.
- The provided keys can be complete key values (group and item specifiers) or wild-card specifications. A complete key value constitutes a deletion request for one key-value pair. A key value with a valid group specifier and 0xffff in the item part of the key value (bits 0-15) constitutes a deletion request for all items in the specified group. A key with a value of 0xfff in the group part of the key value (bits 16-27) is a deletion request for all items known to the receiver in all groups.

Message structure	*Header*	*Class*	*ID*	*Length (Bytes)*	*Payload*	*Checksum*
	0xb5 0x62	0x06	0x8c	4 + [0..n]·4	*see below*	CK_A CK_B

Payload description:

Byte offset	*Type*	*Name*	*Scale*	*Unit*	*Description*
0	U1	`version`	-	-	Message version (0x01 for this version)
1	X1	`layers`	-	-	The layers where the configuration should be deleted from
bit 1	U:1	`bbr`	-	-	Delete configuration from the BBR layer
bit 2	U:1	`flash`	-	-	Delete configuration from the Flash layer
2	X1	`transaction`	-	-	Transaction action to be applied:
bits 1...0	U:2	`action`	-	-	Transaction action to be applied: • 0 = Transactionless UBX-CFG-VALDEL: In the next UBX-CFG-VALDEL, it can be either 0 or 1. If a transaction has not yet been started, the incoming configuration is applied. If a transaction has already been started, cancels any started transaction and the incoming configuration is applied. • 1 = (Re)Start deletion transaction: In the next UBX-CFG-VALDEL, it can be either 0, 1, 2 or 3. If a transaction has not yet been started, a transaction will be started. If a transaction has already been started, restarts the transaction, effectively removing all previous non-applied UBX-CFG-VALDEL messages. • 2 = Deletion transaction ongoing: In the next UBX-CFG-VALDEL, it can be either 0, 1, 2 or 3. • 3 = Apply and end a deletion transaction: In the next UBX-CFG-VALDEL, it can be either 0 or 1.
3	U1	`reserved0`	-	-	Reserved
Start of repeated group (N times)					
4 + n·4	U4	`keys`	-	-	Configuration key IDs of the configuration items to be deleted
End of repeated group (N times)					

2.10.2 UBX-CFG-VALGET (0x06 0x8b)

2.10.2.1 Get configuration items

Message	**UBX-CFG-VALGET** **Get configuration items**
Type	Poll request
Comment	Overview: • This message is used to get configuration values by providing a list of configuration key IDs, which identify the configuration items to retrieve. • This message can specify the configuration layer where the values of the specified configuration items are retrieved from. • This message is limited to containing a maximum of 64 key IDs. • See Receiver configuration for details.

This message returns a UBX-ACK-NAK:

- if any key is unknown to the receiver FW
- if the layer field specifies an invalid layer to get the value from
- if the keys array specifies more than 64 key IDs.

Notes:

- If a value is requested multiple times within the same poll request, then the reply will contain it multiple times.
- The provided keys can be complete key values (group and item specifiers) or wild-card specifications. A complete key value will constitute a request for one key-value pair. A key value that has a valid group specifier and 0xffff in the item part of the key value (bits 0-15) constitutes a request for all items in the specified group. A key with a value of 0xfff in the group part of the key value (bits 16-27) is a request for all items known to the receiver in all groups.
- The response message is limited to containing a maximum of 64 key-value pairs. If there are wild-card specifications then there may be more than 64 possible responses. In order to handle this, the 'position' field can specify that the response message should skip this number of key-value pairs before it starts constructing the message. This allows a large set of values to be retrieved 64 at a time. If the response contains less than 64 key-value pairs then all values have been reported, otherwise there may be more to read.
- It is not possible to retrieve configuration values for the same configuration item from multiple configuration layers. Separate poll requests must be made for each desired layer.

Message structure	*Header*	*Class*	*ID*	*Length (Bytes)*	*Payload*	*Checksum*
	0xb5 0x62	0x06	0x8b	4 + [0..n]·4	*see below*	CK_A CK_B

Payload description:

Byte offset	*Type*	*Name*	*Scale*	*Unit*	*Description*
0	U1	`version`	-	-	Message version (0x00 for this version)
1	U1	`layer`	-	-	The layer from which the configuration items should be retrieved: • 0 - RAM layer • 1 - BBR layer • 2 - Flash layer • 7 - Default layer
2	U2	`position`	-	-	Skip this many key values before constructing output message
Start of repeated group (N times)					
4 + n·4	U4	`keys`	-	-	Configuration key IDs of the configuration items to be retrieved
End of repeated group (N times)					

2.10.2.2 Configuration items

Message	**UBX-CFG-VALGET** **Configuration items**
Type	Polled
Comment	This message is output by the receiver to return requested configuration data (key and value pairs). See Receiver configuration for details.

Message structure	*Header*	*Class*	*ID*	*Length (Bytes)*	*Payload*	*Checksum*
	0xb5 0x62	0x06	0x8b	4 + [0..n]	*see below*	CK_A CK_B

Payload description:

Byte offset	*Type*	*Name*	*Scale*	*Unit*	*Description*
0	U1	`version`	-	-	Message version (0x01 for this version)

1	U1	layer	-	-	The layer from which the configuration item was retrieved: • 0 - RAM layer • 1 - BBR • 2 - Flash • 7 - Default
2	U2	position	-	-	Number of configuration items skipped in the result set before constructing this message (mirrors the equivalent field in the request message)
Start of repeated group (N times)					
4 + n	U1	cfgData	-	-	Configuration data (key and value pairs)
End of repeated group (N times)					

2.10.3 UBX-CFG-VALSET (0x06 0x8a)

2.10.3.1 Set configuration item values

Message	**UBX-CFG-VALSET** **Set configuration item values**
Type	Set
Comment	Overview: • This message is used to set a configuration by providing configuration data (a list of key and value pairs), which identify the configuration items to change, and their new values. • This message is limited to containing a maximum of 64 key-value pairs. • This message can be used multiple times and every time the result will be applied immediately. To send this message multiple times with the result being applied at the end, see version 1 of UBX-CFG-VALSET that supports transactions. • See Receiver configuration for details. This message returns a UBX-ACK-NAK and no configuration is applied: • if any key is unknown to the receiver FW • if the layer's bitfield does not specify a layer to save a value to • if the requested configuration is not valid. The validity of a configuration is checked only if the message requests to apply the configuration to the RAM configuration layer. Notes: • If a key is sent multiple times within the same message, then the value eventually being applied is the last sent.

Message structure	*Header*	*Class*	*ID*	*Length (Bytes)*	*Payload*	*Checksum*
	0xb5 0x62	0x06	0x8a	4 + [0..n]	*see below*	CK_A CK_B

Payload description:

Byte offset	*Type*	*Name*	*Scale*	*Unit*	*Description*
0	U1	version	-	-	Message version (0x00 for this version)
1	X1	layers	-	-	The layers where the configuration should be applied
bit 0	U:1	ram	-	-	Update configuration in the RAM layer
bit 1	U:1	bbr	-	-	Update configuration in the BBR layer
bit 2	U:1	flash	-	-	Update configuration in the Flash layer
2	U1[2]	reserved0	-	-	Reserved
Start of repeated group (N times)					
4 + n	U1	cfgData	-	-	Configuration data (key and value pairs)
End of repeated group (N times)					

2.10.3.2 Set configuration item values (with transaction)

Message	**UBX-CFG-VALSET** **Set configuration item values (with transaction)**
Type	Set
Comment	Overview: • This message is used to set a configuration by providing configuration data (a list of key and value pairs), which identify the configuration items to change, and their new values. • This message is limited to containing a maximum of 64 key-value pairs. • This message can be used multiple times with the result being managed within a transaction. Within a transaction there is no limit on the number key-value pairs; a transaction is effectively limited to the number of known keys. • See Receiver configuration for details. • See version 0 of UBX-CFG-VALSET for simplified version of this message. This message returns a UBX-ACK-NAK, cancels any started transaction, and no configuration is applied: • if any key within a transaction is unknown to the receiver FW • if an invalid transaction state transition is requested • if the layer's bitfield changes within a transaction • if the layer's bitfield does not specify a layer to save a value to This message returns a UBX-ACK-NAK, and no configuration is applied: • if the requested configuration is not valid. While in a transaction context, only the last message that requests to apply the transaction returns a UBX-ACK-NAK. The validity of a configuration is checked only if the message requests to apply the configuration to the RAM configuration layer. This also applies to a transactionless request. Notes: • Any request for another UBX-CFG-message type (including UBX-CFG-VALDEL and UBX-CFG-VALGET) will cancel any started transaction, and no configuration is applied. • This message can be sent with no key/values to set for the purposes of managing the transaction state transition. • If a key is sent multiple times within the same message or within the same transaction, then the value eventually being applied is the last sent.

Message structure	*Header*	*Class*	*ID*	*Length (Bytes)*	*Payload*	*Checksum*
	0xb5 0x62	0x06	0x8a	4 + [0..n]	*see below*	CK_A CK_B

Payload description:

Byte offset	*Type*	*Name*	*Scale*	*Unit*	*Description*
0	U1	`version`	-	-	Message version (0x01 for this version)
1	X1	`layers`	-	-	The layers where the configuration should be applied
bit 0	$U_{:1}$	`ram`	-	-	Update configuration in the RAM layer
bit 1	$U_{:1}$	`bbr`	-	-	Update configuration in the BBR layer
bit 2	$U_{:1}$	`flash`	-	-	Update configuration in the Flash layer
2	U1	`transaction`	-	-	Transaction action to be applied
bits 1...0	$U_{:2}$	`action`	-	-	Transaction action to be applied: • 0 = Transactionless UBX-CFG-VALSET: In the next UBX-CFG-VALSET, it can be either 0 or 1. If a transaction has not yet been started, the incoming configuration is applied (if valid). If a transaction has already been started, cancels any started transaction and the incoming configuration is applied (if valid). • 1 = (Re)Start set transaction: In the next UBX-CFG-VALSET, it can be either 0, 1, 2 or 3. If a transaction has not yet been started, a transaction will be started. If a transaction has already been started, restarts the transaction, effectively removing all previous non-applied UBX-CFG-VALSET messages.

					• 2 = Set transaction ongoing: In the next UBX-CFG-VALSET, it can be either 0, 1, 2 or 3. • 3 = Apply and end a set transaction: In the next UBX-CFG-VALSET, it can be either 0 or 1.
3	U1	`reserved0`	-	-	Reserved
Start of repeated group (N times)					
4 + n	U1	`cfgData`	-	-	Configuration data (key and value pairs)
End of repeated group (N times)					

2.11 UBX-MON (0x0a)

The messages in the UBX-MON class are used to report the receiver status, such as hardware status or I/O subsystem statistics.

2.11.1 UBX-MON-VER (0x0a 0x04)

2.11.1.1 Poll receiver and software version

Message	**UBX-MON-VER** **Poll receiver and software version**					
Type	Poll request					
Comment						
Message structure	*Header*	*Class*	*ID*	*Length (Bytes)*	*Payload*	*Checksum*
	0xb5 0x62	0x0a	0x04	0	*see below*	CK_A CK_B
Payload	This message has no payload.					

2.11.1.2 Receiver and software version

Message	**UBX-MON-VER** **Receiver and software version**					
Type	Polled					
Comment						
Message structure	*Header*	*Class*	*ID*	*Length (Bytes)*	*Payload*	*Checksum*
	0xb5 0x62	0x0a	0x04	40 + [0..n]·30	*see below*	CK_A CK_B

Payload description:

Byte offset	*Type*	*Name*	*Scale*	*Unit*	*Description*
0	CH[30]	`swVersion`	-	-	Nul-terminated software version string.
30	CH[10]	`hwVersion`	-	-	Nul-terminated hardware version string
Start of repeated group (N times)					
40 + n·30	CH[30]	`extension`	-	-	Extended software information strings. A series of nul-terminated strings. Each extension field is 30 characters long and contains varying software information. Not all extension fields may appear. Examples of reported information: the software version string of the underlying ROM (when the receiver's firmware is running from flash), the firmware version, the supported protocol version, the module identifier, the flash information structure (FIS) file information, the supported major GNSS, the supported augmentation systems. See Firmware and protocol versions for details.

End of repeated group (N times)

2.12 UBX-NAV (0x01)

The messages in the UBX-NAV class are used to output navigation results and data, such as position, altitude and velocity in a number of formats, and status flags and accuracy estimate figures, or satellite and signal information. The messages are generated with the configured navigation rate.

2.12.1 UBX-NAV-PVT (0x01 0x07)

2.12.1.1 Navigation position velocity time solution

Message	**UBX-NAV-PVT** **Navigation position velocity time solution**					
Type	Periodic/polled					
Comment	This message combines position, velocity and time solution, including accuracy figures. Note that during a leap second there may be more or less than 60 seconds in a minute. See description of leap seconds in the integration manual for details.					
Message structure	*Header*	*Class*	*ID*	*Length (Bytes)*	*Payload*	*Checksum*
	0xb5 0x62	0x01	0x07	92	*see below*	CK_A CK_B

Payload description:

Byte offset	*Type*	*Name*	*Scale*	*Unit*	*Description*
0	U4	`iTOW`	-	ms	GPS time of week of the navigation epoch. See section iTOW timestamps in the integration manual for details.
4	U2	`year`	-	y	Year (UTC)
6	U1	`month`	-	month	Month, range 1..12 (UTC)
7	U1	`day`	-	d	Day of month, range 1..31 (UTC)
8	U1	`hour`	-	h	Hour of day, range 0..23 (UTC)
9	U1	`min`	-	min	Minute of hour, range 0..59 (UTC)
10	U1	`sec`	-	s	Seconds of minute, range 0..60 (UTC)
11	X1	`valid`	-	-	Validity flags
bit 0	$U_{:1}$	`validDate`	-	-	1 = valid UTC Date (see section Time validity in the integration manual for details)
bit 1	$U_{:1}$	`validTime`	-	-	1 = valid UTC time of day (see section Time validity in the integration manual for details)
bit 2	$U_{:1}$	`fullyResolved`	-	-	1 = UTC time of day has been fully resolved (no seconds uncertainty). Cannot be used to check if time is completely solved.
bit 3	$U_{:1}$	`validMag`	-	-	1 = valid magnetic declination
12	U4	`tAcc`	-	ns	Time accuracy estimate (UTC)
16	I4	`nano`	-	ns	Fraction of second, range -1e9 .. 1e9 (UTC)

20		U1	fixType	-	-	GNSSfix Type: • 0 = no fix • 1 = dead reckoning only • 2 = 2D-fix • 3 = 3D-fix • 4 = GNSS + dead reckoning combined • 5 = time only fix
21		X1	flags	-	-	Fix status flags
	bit 0	U:1	gnssFixOK	-	-	1 = valid fix (i.e within DOP & accuracy masks)
	bit 1	U:1	diffSoln	-	-	1 = differential corrections were applied
	bits 4...2	U:3	psmState	-	-	Power save mode state (see Power management section in the integration manual for details. • 0 = PSM is not active • 1 = Enabled (an intermediate state before Acquisition state • 2 = Acquisition • 3 = Tracking • 4 = Power Optimized Tracking • 5 = Inactive
	bit 5	U:1	headVehValid	-	-	1 = heading of vehicle is valid, only set if the receiver is in sensor fusion mode
	bits 7...6	U:2	carrSoln	-	-	Carrier phase range solution status: • 0 = no carrier phase range solution • 1 = carrier phase range solution with floating ambiguities • 2 = carrier phase range solution with fixed ambiguities (not supported for protocol versions less than 20.00)
22		X1	flags2	-	-	Additional flags
	bit 5	U:1	confirmedAvai	-	-	1 = information about UTC Date and Time of Day validity confirmation is available (see section Time validity in the integration manual for details) This flag is only supported in Protocol Versions 19.00, 19.10, 20.10, 20.20, 20.30, 22.00, 23.00, 23.01, 27 and 28.
	bit 6	U:1	confirmedDate	-	-	1 = UTC Date validity could be confirmed (see section Time validity in the integration manual for details)
	bit 7	U:1	confirmedTime	-	-	1 = UTC Time of Day could be confirmed (see section Time validity in the integration manual for details)
23		U1	numSV	-	-	Number of satellites used in Nav Solution
24		I4	lon	1e-7	deg	Longitude
28		I4	lat	1e-7	deg	Latitude
32		I4	height	-	mm	Height above ellipsoid
36		I4	hMSL	-	mm	Height above mean sea level
40		U4	hAcc	-	mm	Horizontal accuracy estimate
44		U4	vAcc	-	mm	Vertical accuracy estimate
48		I4	velN	-	mm/s	NED north velocity
52		I4	velE	-	mm/s	NED east velocity
56		I4	velD	-	mm/s	NED down velocity
60		I4	gSpeed	-	mm/s	Ground Speed (2-D)

64	I4	headMot	1e-5	deg	Heading of motion (2-D)
68	U4	sAcc	-	mm/s	Speed accuracy estimate
72	U4	headAcc	1e-5	deg	Heading accuracy estimate (both motion and vehicle)
76	U2	pDOP	0.01	-	Position DOP
78	X2	flags3	-	-	Additional flags
bit 0	U:1	invalidLlh	-	-	1 = Invalid lon, lat, height and hMSL (applicable to heading products only)
bits 4...1	U:4	lastCorrectionAge	-	-	Age of the most recently received differential correction: • 0 = Not available • 1 = Age between 0 and 1 second • 2 = Age between 1 (inclusive) and 2 seconds • 3 = Age between 2 (inclusive) and 5 seconds • 4 = Age between 5 (inclusive) and 10 seconds • 5 = Age between 10 (inclusive) and 15 seconds • 6 = Age between 15 (inclusive) and 20 seconds • 7 = Age between 20 (inclusive) and 30 seconds • 8 = Age between 30 (inclusive) and 45 seconds • 9 = Age between 45 (inclusive) and 60 seconds • 10 = Age between 60 (inclusive) and 90 seconds • 11 = Age between 90 (inclusive) and 120 seconds • >=12 = Age greater or equal than 120 seconds
bit 13	U:1	authTime	-	-	Flag that indicates if the output time has been validated against an external trusted time source • 0 = Time is not authenticated • 1 = Time is authenticated
bit 14	U:1	nmaFixStatus	-	-	Indicates that the PVT fix has been verified with the NMA data • 0 = Not Verified • 1 = Verified
80	U1[4]	reserved0	-	-	Reserved
84	I4	headVeh	1e-5	deg	Heading of vehicle (2-D), this is only valid when headVehValid is set, otherwise the output is set to the heading of motion
88	I2	magDec	1e-2	deg	Magnetic declination. Only supported in ADR 4.10 and later.
90	U2	magAcc	1e-2	deg	Magnetic declination accuracy. Only supported in ADR 4.10 and later.

2.12.2 UBX-NAV-TIMEGPS (0x01 0x20)

2.12.2.1 GPS time solution

Message	**UBX-NAV-TIMEGPS** **GPS time solution**					
Type	Periodic/polled					
Comment	This message reports the precise GPS time of the most recent navigation solution including validity flags and an accuracy estimate.					
Message structure	*Header*	*Class*	*ID*	*Length (Bytes)*	*Payload*	*Checksum*
	0xb5 0x62	0x01	0x20	16	*see below*	CK_A CK_B

Payload description:

Byte offset	*Type*	*Name*	*Scale*	*Unit*	*Description*

0	U4	iTOW	-	ms	GPS time of week of the navigation epoch. See section iTOW timestamps in the integration manual for details.
4	I4	fTOW	-	ns	Fractional part of iTOW (range: +/-500000). The precise GPS time of week in seconds is: (iTOW * 1e-3) + (fTOW * 1e-9)
8	I2	week	-	-	GPS week number of the navigation epoch
10	I1	leapS	-	s	GPS leap seconds (GPS-UTC)
11	X1	valid	-	-	Validity Flags
bit 0	U:1	towValid	-	-	1 = Valid GPS time of week (iTOW & fTOW, (see section Time validity in the integration manual for details)
bit 1	U:1	weekValid	-	-	1 = Valid GPS week number (see section Time validity in the integration manual for details)
bit 2	U:1	leapSValid	-	-	1 = Valid GPS leap seconds
12	U4	tAcc	-	ns	Time Accuracy Estimate

2.13 UBX-RXM (0x02)

The messages in the UBX-RXM class are used to output status and result data from the receiver manager as well as sending commands to the receiver manager.

2.13.1 UBX-RXM-QZSSL6 (0x02 0x73)

2.13.1.1 QZSS L6 message

Message	**UBX-RXM-QZSSL6** **QZSS L6 message**					
Type	Periodic					
Comment	Output of a received QZSS L6 message, which is defined in 'Quasi Zenith Satellite System Interface Specification Centimeter Level Augmentation Service (IS-QZSS-L6-001)'.					
Message structure	*Header*	*Class*	*ID*	*Length (Bytes)*	*Payload*	*Checksum*
	0xb5 0x62	0x02	0x73	264	*see below*	CK_A CK_B

Payload description:

Byte offset	*Type*	*Name*	*Scale*	*Unit*	*Description*
0	U1	version	-	-	Message version (0x01 for this version)
1	U1	svId	-	-	Satellite identifier (see Satellite Numbering)
2	U2	cno	2^-8	dBHz	Mean C/N0
4	U4	timeTag	-	ms	Local time tag corresponding to the beginning of a received QZSS L6 message
8	U1	groupDelay	-	ns	L6 group delay w.r.t. L2 on channel
9	U1	bitErrCorr	-	-	Number of bit errors corrected by Reed-Solomon decoder
10	X2	chInfo	-	-	Information about receiver channel associated with a received QZSS L6 message
bits 9...8	U:2	chn	-	-	Receiver channel (0, 1)
bit 10	U:1	msgName	-	-	Message name, 0=L6D, 1=L6E
bits 13...12	U:2	errStatus	-	-	Error status of the received QZSS L6 message: 0=unknown, 1=error-free, 2=erroneous

bits 15...14	U:2	chName	-	-	Channel name, 0=channel A, 1=channel B
12	U1[2]	reserved0	-	-	Reserved
14	U1[250]	msgBytes	-	-	Bytes in a QZSS L6 message

2.13.2 UBX-RXM-SFRBX (0x02 0x13)

2.13.2.1 Broadcast navigation data subframe

Message	**UBX-RXM-SFRBX** **Broadcast navigation data subframe**					
Type	Output					
Comment	This message reports a complete subframe of broadcast navigation data decoded from a single signal. The number of data words reported in each message depends on the nature of the signal.					
Message structure	*Header*	*Class*	*ID*	*Length (Bytes)*	*Payload*	*Checksum*
	0xb5 0x62	0x02	0x13	8 + numWords·4	*see below*	CK_A CK_B

Payload description:

Byte offset	*Type*	*Name*	*Scale*	*Unit*	*Description*
0	U1	gnssId	-	-	GNSS identifier (see Satellite Numbering)
1	U1	svId	-	-	Satellite identifier (see Satellite Numbering)
2	U1	sigId	-	-	Signal identifier (see Signal Identifiers)
3	U1	freqId	-	-	Only used for GLONASS: This is the frequency slot + 7 (range from 0 to 13)
4	U1	numWords	-	-	The number of data words contained in this message (up to 10, for currently supported signals)
5	U1	chn	-	-	The tracking channel number the message was received on
6	U1	version	-	-	Message version, (0x02 for this version)
7	U1	reserved0	-	-	Reserved
Start of repeated group (numWords times)					
8 + n·4	U4	dwrd	-	-	The data words
End of repeated group (numWords times)					

3 Configuration interface

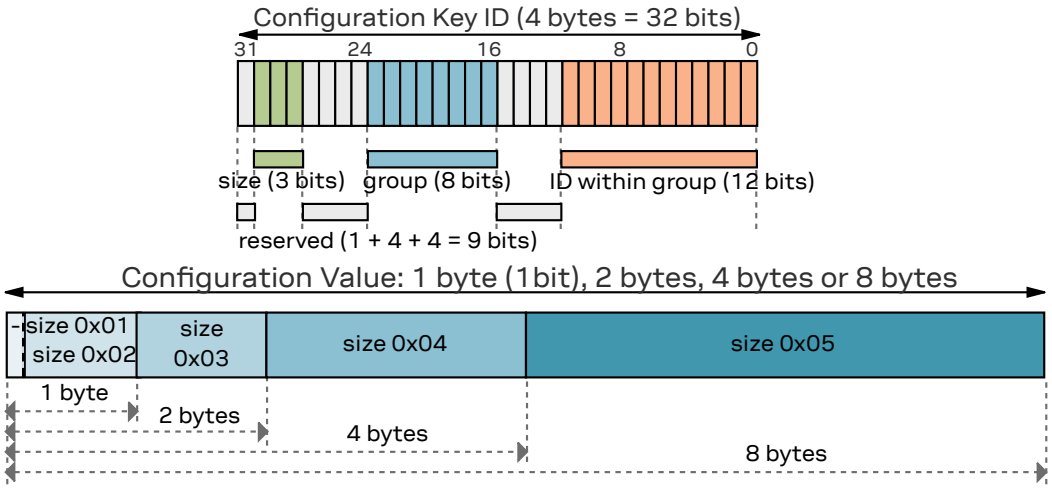
This chapter describes the receiver configuration interface.

3.1 Configuration database

The configuration database in the receiver's RAM holds the current configuration, which is used by the receiver at run-time. It is constructed on startup of the receiver from several sources of configuration. These sources are called *Configuration Layers*. The current configuration is called the *RAM Layer*. Any configuration in any layer is organized as *Configuration Items*, where each Configuration Item is referenced to by a unique *Configuration Key ID* and holds a single *Configuration Value*.

3.2 Configuration items

The following figure shows the structure of a *Configuration Item*, which consists of a *(Configuration) Key ID* and its *(Configuration) Value*:



A Configuration Key ID is a 32-bit integer value, which is split into the following parts:

- Bit 31: Currently unused. Reserved for future use.
- Bits 30...28: Three bits that indicate the storage size of a Configuration Value (range 0x01-0x05, see below)
- Bits 27...24: Currently unused. Reserved for future use.
- Bits 23...16: Eight bits that define a unique group ID (range 0x01-0xfe)
- Bits 15...12: Currently unused. Reserved for future use.
- Bits 11...0: Twelve bits that define a unique item ID within a group (range 0x001-0xffe)

The entire 32-bit value is the unique Key ID, which uniquely identifies a particular item. The numeric representation of the Key ID uses the lower-case hexadecimal format, such as `0x20c400a1`. An easier, more readable text representation uses the form CFG-*GROUP-ITEM*. This is also referred to as the *(Configuration) Key Name*.

Supported storage size identifiers (bits 30...28 of the Key ID) are:

- 0x01: one bit (the actual storage used is one byte, but only the least significant bit is used)
- 0x02: one byte
- 0x03: two bytes
- 0x04: four bytes

- 0x05: eight bytes

Each Configuration Item is of a certain type, which defines the interpretation of the raw binary data (see also UBX data types):

- U1, U2, U4, U8: unsigned little-endian integers of 8-, 16-, 32- and 64-bit widths
- I1, I2, I4, I8: signed little-endian, two's complement integers of 8-, 16-, 32- and 64-bit widths
- R4, R8: IEEE 754 single (32-bit) and double (64-bit) precision floats
- E1, E2, E4: unsigned little-endian enumeration of 8-, 16-, and 32-bit widths
- X1, X2, X4, X8: unsigned little-endian integers of 8-, 16-, 32- and 64-bit widths for bitfields and other binary data, such as strings
- L: single-bit boolean (true = 1, false = 0), stored as U1

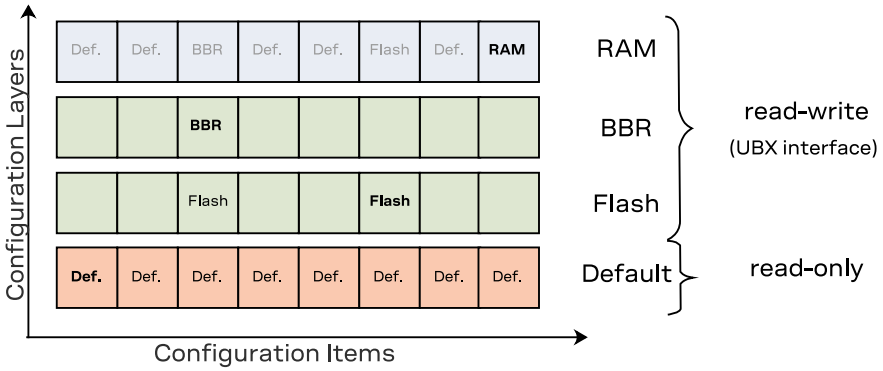
3.3 Configuration layers

The receiver has several *Configuration Layers*. They are separate sources of Configuration Items. Some of the layers are read-only and others are modifiable. Layers are organized in terms of priority. Values in a high-priority layer replace values stored in a low-priority layer. At startup, the receiver reads all configuration layers and stacks up the items to create the *Current Configuration*, which is used by the receiver at run-time.

The following configuration layers are available (in order of priority, highest priority first):

- **RAM**: This layer contains items stored in volatile RAM. This is the Current Configuration. The value of any item can be set by the user at run-time (see UBX protocol interface) and it is effective immediately.
- **BBR**: This layer contains items stored in the battery-backed RAM. The contents in this layer are preserved as long as a battery backup supply is provided during off periods. The value of any item can be set by the user at run-time (see UBX protocol interface) and it becomes effective when the receiver is restarted.
- **Flash**: This layer contains items stored permanently in the external flash memory. This layer is only available if there is a usable external flash memory. The value of any item can be set by the user at run-time (see UBX protocol interface) and it becomes effective when the receiver is restarted.
- **Default:** This layer contains all items known to the running receiver software and their hard-coded default values. Data in this layer is not writable.

The stacking of the configuration items from the different layers (sources) in order to construct the Current Configuration in the RAM Layer is depicted in the following figure. For each defined item, i.e. for each item in the Default Layer, the receiver software goes through the layers above and stacks all the found items on top. Some items may not be present in every layer. The result is the RAM Layer filled with all configuration items given Configuration Values coming from the highest priority layer the corresponding item was present. In the example figure below bold text indicates the source of the value in the Current Configuration (the RAM Layer). Empty boxes mean that the layer can hold the item but that it is not currently stored there. Boxes with text mean that an item is currently stored in the layer.



In the example figure above several items (e.g. the first item) are only set in the Default Layer and hence the default value ends up in Current Configuration in the RAM Layer. The third item is present in the Default, Flash and BBR Layers. The value from the BBR Layer has the highest priority and therefore it ends up in the RAM Layer. On the other hand, the default value of the sixth item is changed by the value in the Flash Layer. The value of the last item is changed in the RAM Layer only, i.e. upon startup the value in the RAM Layer was the value from the Default Layer, but the user has changed the value in the RAM Layer at run-time.

3.4 Configuration interface access

The following sections describe the existing interfaces to access the Configuration Database.

3.4.1 UBX protocol interface

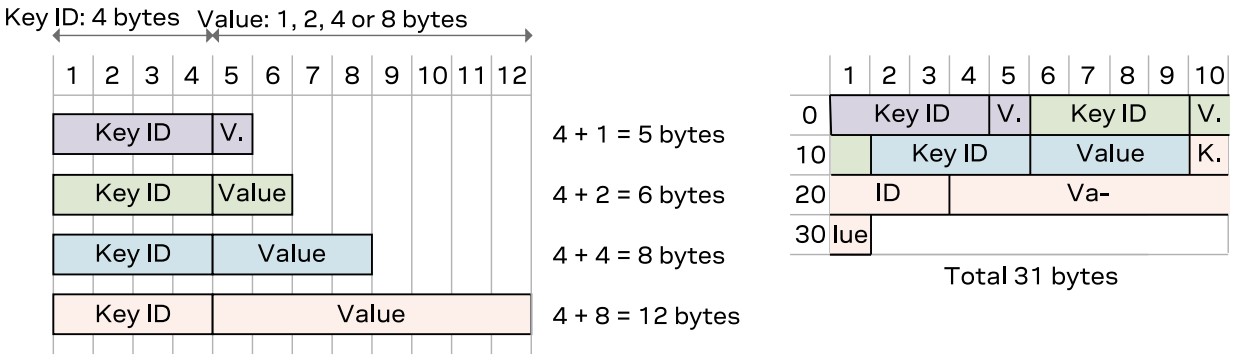
The following UBX protocol messages are available to access the Configuration Database:

- UBX-CFG-VALGET to read configuration items from the database
- UBX-CFG-VALSET to set configuration items in the database
- UBX-CFG-VALDEL to delete configuration items from the database

3.5 Configuration data

Configuration data is the binary representation of a list of Key ID and Value pairs. It is formed by concatenating keys (U4 values) and values (variable type) without any padding. This format is used in the UBX-CFG-VALSET and UBX-CFG-VALGET messages.

The figure below shows an example. The four Items (Key ID - Value pairs) on the left use the four fundamental storage sizes: one byte (L, U1, I1, E1 and X1 types), 2 bytes (U2, I2, E2 and X2 types), four byte (U4, I4, E4, X4 and R4 types) and eight bytes (U8, I8, X8 and R8 types). When concatenated (right) the Key IDs and Values are not aligned and there is no padding.



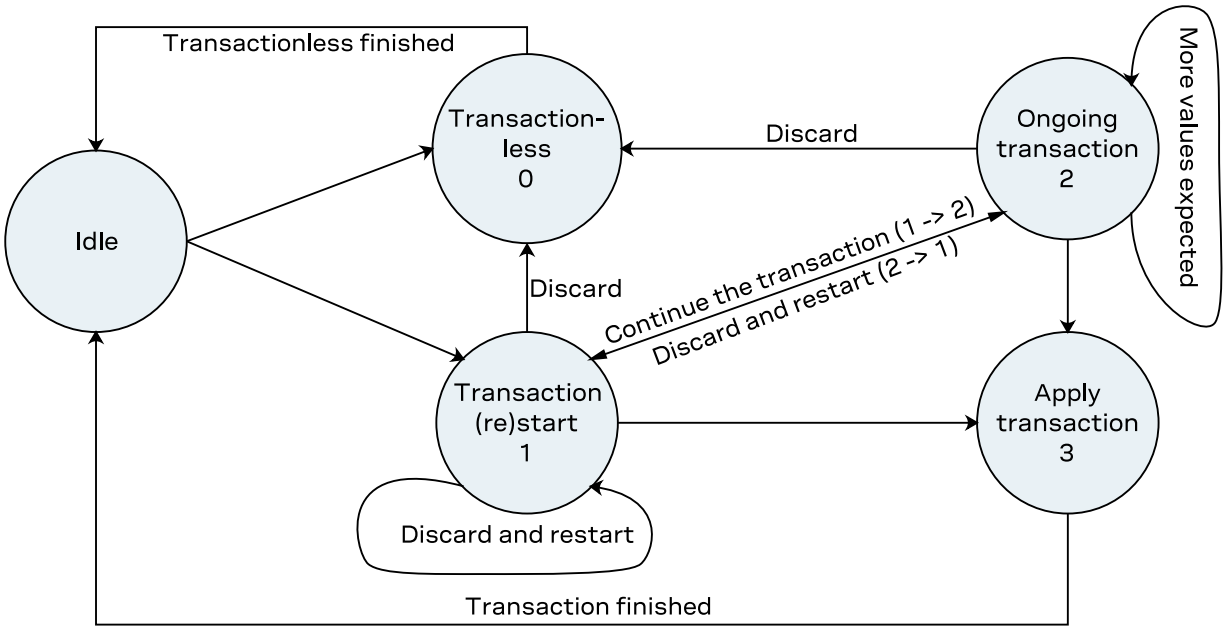
Note that this is an arbitrary example and any number of items of any value storage size can be concatenated the same way.

3.6 Configuration transactions

The configuration interface supports two mechanisms of configuration: the first is a transactionless mechanism where sent configuration changes are applied immediately to the configuration layer(s) requested. The second mechanism is a configuration transaction.

A transaction offers a way of queuing multiple configuration changes. It is particularly useful where different configuration keys depend on each other in such a way that sending one before the other can cause the configuration to be rejected. The queued configuration change requests are stored then checked collectively before being applied to the receiver.

A transaction can have the following states described in the figure below.



When starting a transaction, specify the layer(s) to apply the changes to. This list of configuration layer(s) must be observed throughout the transaction states. Modifying the configuration layer(s) mid-transaction causes the transaction to be aborted and consequently, no queued changes will be applied.

In the start transaction state, the receiver locks the configuration database so that changes from another entity or message cannot be applied. It is possible to send a configuration key-value pairs with the start transaction state. These are queued waiting to be applied.

In the ongoing state, a configuration key and value must be sent. The receiver aborts the transaction and does not apply any changes if this condition is violated. Key-value pairs sent in the ongoing state are queued waiting to be applied.

In the apply state, the receiver collectively checkes the queued changes and applied them to the requested configuration layer(s). Note that any additional key-value pairs sent within the apply state are ignored.

Note that a transaction can only come from a single source, a UBX-CFG-VALSET message or a UBX-CFG-VALDEL message. This means that in any given transaction it is not possible to mix a delete

and a save request. Starting a transaction from a different source aborts the current transaction and the queued changes are not applied.

Refer to UBX-CFG-VALSET and UBX-CFG-VALDEL messages for a detailed description of how to set up a configuration transaction, its limitations and conditions that would cause the transaction to be rejected.

3.7 Configuration reset behavior

The RAM layer is always rebuilt from the layers below when the chip's processor comes out from reset. When using UBX-CFG-RST the processor goes through a reset cycle with these reset types (`resetMode` field):

- `0x00` hardware reset (watchdog) immediately
- `0x01` controlled software reset
- `0x04` hardware reset (watchdog) after shutdown

See section Forcing a receiver reset in the integration manual.

3.8 Configuration overview

Group	Description
CFG-I2C	Configuration of the I2C interface
CFG-INFMSG	Information message configuration
CFG-MSGOUT	Message output configuration
CFG-QZSS	QZSS system configuration
CFG-UART1	Configuration of the UART1 interface
CFG-UART1INPROT	Input protocol configuration of the UART1 interface
CFG-UART1OUTPROT	Output protocol configuration of the UART1 interface
CFG-USB	Configuration of the USB interface
CFG-USBINPROT	Input protocol configuration of the USB interface
CFG-USBOUTPROT	Output protocol configuration of the USB interface

3.9 Configuration reference

3.9.1 CFG-I2C: Configuration of the I2C interface

Settings needed to configure the I2C communication interface.

Configuration item	Key ID	Type	Scale	Unit	Description
CFG-I2C-ADDRESS	0x20510001	U1	-	-	I2C address of the receiver (7 bits)
CFG-I2C-EXTENDEDTIMEOUT	0x10510002	L	-	-	Flag to disable timeouting the interface after 1.5 s
CFG-I2C-ENABLED	0x10510003	L	-	-	Flag to indicate if the I2C interface should be enabled

Table 4: CFG-I2C configuration items

3.9.2 CFG-INFMSG: Information message configuration

Information message configuration for the NMEA and UBX protocols.

Configuration item	Key ID	Type	Scale	Unit	Description
CFG-INFMSG-UBX_UART1	0x20920002	X1	-	-	Information message enable flags for the UBX protocol on the UART1 interface
See Table 6 below for a list of possible constants for this item.					
CFG-INFMSG-UBX_USB	0x20920004	X1	-	-	Information message enable flags for the UBX protocol on the USB interface
See Table 6 below for a list of possible constants for this item.					

Table 5: CFG-INFMSG configuration items

Constant	Value	Description
ERROR	0x01	Enable ERROR information messages
WARNING	0x02	Enable WARNING information messages
NOTICE	0x04	Enable NOTICE information messages
TEST	0x08	Enable TEST information messages
DEBUG	0x10	Enable DEBUG information messages

Table 6: Constants for CFG-INFMSG-UBX_I2C, CFG-INFMSG-UBX_UART1, CFG-INFMSG-UBX_UART2, CFG-INFMSG-UBX_USB, CFG-INFMSG-UBX_SPI, CFG-INFMSG-NMEA_I2C, CFG-INFMSG-NMEA_UART1, CFG-INFMSG-NMEA_UART2, CFG-INFMSG-NMEA_USB, CFG-INFMSG-NMEA_SPI

3.9.3 CFG-MSGOUT: Message output configuration

For each message and port a separate output rate (per second, per epoch) can be configured.

Configuration item	Key ID	Type	Scale	Unit	Description
CFG-MSGOUT-UBX_RXM_QZSSL6_UART1	0x2091033b	U1	-	-	output rate of the UBX-RXM-QZSSL6 message on port UART1
CFG-MSGOUT-UBX_RXM_QZSSL6_USB	0x2091033d	U1	-	-	output rate of the UBX-RXM-QZSSL6 message on port USB

Table 7: CFG-MSGOUT configuration items

3.9.4 CFG-QZSS: QZSS system configuration

Note that enabling and disabling of individual GNSS is done via the CFG-SIGNAL configuration group.

Configuration item	Key ID	Type	Scale	Unit	Description
CFG-QZSS-L6_SVIDA	0x20370020	I1	-	-	QZSS L6 SV Id to be decoded by channel A
-1 = disable channel; 0 = automatic selection; 1, 2, ... = manual satellite selection					
CFG-QZSS-L6_SVIDB	0x20370030	I1	-	-	QZSS L6 SV Id to be decoded by channel B
-1 = disable channel; 0 = automatic selection; 1, 2, ... = manual satellite selection					
CFG-QZSS-L6_MSGA	0x20370050	E1	-	-	QZSS L6 messages to be decoded by channel A
See Table 9 below for a list of possible constants for this item.					
CFG-QZSS-L6_MSGB	0x20370060	E1	-	-	QZSS L6 messages to be decoded by channel B
See Table 10 below for a list of possible constants for this item.					
CFG-QZSS-L6_RSDECODER	0x20370080	E1	-	-	QZSS L6 message Reed-Solomon decoder mode
See Table 11 below for a list of possible constants for this item.					

Table 8: CFG-QZSS configuration items

Constant	Value	Description
L6D	0	L6D messages

Constant	Value	Description
L6E	1	L6E messages

Table 9: Constants for CFG-QZSS-L6_MSGA

Constant	Value	Description
L6D	0	L6D messages
L6E	1	L6E messages

Table 10: Constants for CFG-QZSS-L6_MSGB

Constant	Value	Description
DISABLED	0	Disabled, received messages are output with unknown bit error status
ERRDETECT	1	Error detection, RS-decoder detects bit errors in received messages
ERRCORRECT	2	Error correction, RS-decoder detects and corrects bit errors in received messages

Table 11: Constants for CFG-QZSS-L6_RSDECODER

3.9.5 CFG-UART1: Configuration of the UART1 interface

Settings needed to configure the UART1 communication interface.

Configuration item	Key ID	Type	Scale	Unit	Description
CFG-UART1-BAUDRATE	0x40520001	U4	-	-	The baud rate that should be configured on the UART1
CFG-UART1-STOPBITS	0x20520002	E1	-	-	Number of stopbits that should be used on UART1
See Table 13 below for a list of possible constants for this item.					
CFG-UART1-DATABITS	0x20520003	E1	-	-	Number of databits that should be used on UART1
See Table 14 below for a list of possible constants for this item.					
CFG-UART1-PARITY	0x20520004	E1	-	-	Parity mode that should be used on UART1
See Table 15 below for a list of possible constants for this item.					
CFG-UART1-ENABLED	0x10520005	L	-	-	Flag to indicate if the UART1 should be enabled

Table 12: CFG-UART1 configuration items

Constant	Value	Description
HALF	0	0.5 stopbits
ONE	1	1.0 stopbits
ONEHALF	2	1.5 stopbits
TWO	3	2.0 stopbits

Table 13: Constants for CFG-UART1-STOPBITS

Constant	Value	Description
EIGHT	0	8 databits
SEVEN	1	7 databits

Table 14: Constants for CFG-UART1-DATABITS

Constant	Value	Description
NONE	0	No parity bit
ODD	1	Add an odd parity bit

Constant	Value	Description
EVEN	2	Add an even parity bit

Table 15: Constants for CFG-UART1-PARITY

3.9.6 CFG-UART1INPROT: Input protocol configuration of the UART1 interface

Input protocol enable flags of the UART1 interface.

Configuration item	Key ID	Type	Scale	Unit	Description
CFG-UART1INPROT-UBX	0x10730001	L	-	-	Flag to indicate if UBX should be an input protocol on UART1
CFG-UART1INPROT-RTCM2X	0x10730003	L	-	-	Flag to indicate if RTCM2X should be an input protocol on UART1

Table 16: CFG-UART1INPROT configuration items

3.9.7 CFG-UART1OUTPROT: Output protocol configuration of the UART1 interface

Output protocol enable flags of the UART1 interface.

Configuration item	Key ID	Type	Scale	Unit	Description
CFG-UART1OUTPROT-UBX	0x10740001	L	-	-	Flag to indicate if UBX should be an output protocol on UART1

Table 17: CFG-UART1OUTPROT configuration items

3.9.8 CFG-USB: Configuration of the USB interface

Settings needed to configure the USB communication interface.

Configuration item	Key ID	Type	Scale	Unit	Description
CFG-USB-ENABLED	0x10650001	L	-	-	Flag to indicate if the USB interface should be enabled
CFG-USB-SELFPOW	0x10650002	L	-	-	Self-powered device
CFG-USB-VENDOR_ID	0x3065000a	U2	-	-	Vendor ID
CFG-USB-PRODUCT_ID	0x3065000b	U2	-	-	Vendor ID
CFG-USB-POWER	0x3065000c	U2	-	mA	Power consumption
CFG-USB-VENDOR_STR0	0x5065000d	X8	-	-	Vendor string characters 0-7
CFG-USB-VENDOR_STR1	0x5065000e	X8	-	-	Vendor string characters 8-15
CFG-USB-VENDOR_STR2	0x5065000f	X8	-	-	Vendor string characters 16-23
CFG-USB-VENDOR_STR3	0x50650010	X8	-	-	Vendor string characters 24-31
CFG-USB-PRODUCT_STR0	0x50650011	X8	-	-	Product string characters 0-7
CFG-USB-PRODUCT_STR1	0x50650012	X8	-	-	Product string characters 8-15
CFG-USB-PRODUCT_STR2	0x50650013	X8	-	-	Product string characters 16-23
CFG-USB-PRODUCT_STR3	0x50650014	X8	-	-	Product string characters 24-31
CFG-USB-SERIAL_NO_STR0	0x50650015	X8	-	-	Serial number string characters 0-7
CFG-USB-SERIAL_NO_STR1	0x50650016	X8	-	-	Serial number string characters 8-15
CFG-USB-SERIAL_NO_STR2	0x50650017	X8	-	-	Serial number string characters 16-23

Configuration item	Key ID	Type	Scale	Unit	Description
CFG-USB-SERIAL_NO_STR3	0x50650018	X8	-	-	Serial number string characters 24-31

Table 18: CFG-USB configuration items

3.9.9 CFG-USBINPROT: Input protocol configuration of the USB interface

Input protocol enable flags of the USB interface.

Configuration item	Key ID	Type	Scale	Unit	Description
CFG-USBINPROT-UBX	0x10770001	L	-	-	Flag to indicate if UBX should be an input protocol on USB
CFG-USBINPROT-RTCM2X	0x10770003	L	-	-	Flag to indicate if RTCM2X should be an input protocol on USB

Table 19: CFG-USBINPROT configuration items

3.9.10 CFG-USBOUTPROT: Output protocol configuration of the USB interface

Output protocol enable flags of the USB interface.

Configuration item	Key ID	Type	Scale	Unit	Description
CFG-USBOUTPROT-UBX	0x10780001	L	-	-	Flag to indicate if UBX should be an output protocol on USB

Table 20: CFG-USBOUTPROT configuration items

3.10 Legacy UBX message fields reference

The following table lists the legacy UBX message fields and the corresponding configuration item. Note that the mapping from UBX-CFG message fields to configuration items is not necessarily 1:1 and that that some legacy UBX-CFG messages may not be available for certain products.

UBX message and field	Configuration item(s)
UBX-CFG-INF	
UBX-CFG-INF.infMsgMask	CFG-INFMSG-UBX_I2C, CFG-INFMSG-UBX_UART1, CFG-INFMSG-UBX_UART2, CFG-INFMSG-UBX_USB, CFG-INFMSG-UBX_SPI, CFG-INFMSG-NMEA_I2C, CFG-INFMSG-NMEA_UART1, CFG-INFMSG-NMEA_UART2, CFG-INFMSG-NMEA_USB, CFG-INFMSG-NMEA_SPI
UBX-CFG-INF.protocolID	CFG-INFMSG-UBX_UART1, CFG-INFMSG-UBX_UART2, CFG-INFMSG-UBX_USB, CFG-INFMSG-UBX_SPI, CFG-INFMSG-NMEA_I2C, CFG-INFMSG-NMEA_UART1, CFG-INFMSG-NMEA_UART2, CFG-INFMSG-NMEA_USB, CFG-INFMSG-NMEA_SPI
UBX-CFG-PRT	
UBX-CFG-PRT.extendedTxTimeout	CFG-I2C-EXTENDEDTIMEOUT
UBX-CFG-PRT.inProtoMask	CFG-I2C-ENABLED
UBX-CFG-PRT.outProtoMask	CFG-I2C-ENABLED
UBX-CFG-PRT.slaveAddr	CFG-I2C-ADDRESS
UBX-CFG-PRT.baudRate	CFG-UART1-BAUDRATE, CFG-UART2-BAUDRATE
UBX-CFG-PRT.charLen	CFG-UART1-DATABITS, CFG-UART2-DATABITS
UBX-CFG-PRT.inProtoMask	CFG-UART1-ENABLED, CFG-UART2-ENABLED
UBX-CFG-PRT.inRtcm	CFG-UART1INPROT-RTCM2X, CFG-UART2INPROT-RTCM2X
UBX-CFG-PRT.inUbx	CFG-UART1INPROT-UBX, CFG-UART2INPROT-UBX
UBX-CFG-PRT.nStopBits	CFG-UART1-STOPBITS, CFG-UART2-STOPBITS
UBX-CFG-PRT.outProtoMask	CFG-UART1-ENABLED, CFG-UART2-ENABLED

UBX message and field	Configuration item(s)
UBX-CFG-PRT.outUbx	CFG-UART1OUTPROT-UBX, CFG-UART2OUTPROT-UBX
UBX-CFG-PRT.parity	CFG-UART1-PARITY, CFG-UART2-PARITY
UBX-CFG-PRT.inProtoMask	CFG-USB-ENABLED
UBX-CFG-PRT.inRtcm	CFG-USBINPROT-RTCM2X
UBX-CFG-PRT.inUbx	CFG-USBINPROT-UBX
UBX-CFG-PRT.outProtoMask	CFG-USB-ENABLED
UBX-CFG-PRT.outUbx	CFG-USBOUTPROT-UBX
UBX-CFG-USB	
UBX-CFG-USB.powerConsumption	CFG-USB-POWER
UBX-CFG-USB.powerMode	CFG-USB-SELFPOW
UBX-CFG-USB.productID	CFG-USB-PRODUCT_ID
UBX-CFG-USB.productString	CFG-USB-PRODUCT_STR0, CFG-USB-PRODUCT_STR1, CFG-USB-PRODUCT_STR2, CFG-USB-PRODUCT_STR3
UBX-CFG-USB.serialNumber	CFG-USB-SERIAL_NO_STR0, CFG-USB-SERIAL_NO_STR1, CFG-USB-SERIAL_NO_STR2, CFG-USB-SERIAL_NO_STR3
UBX-CFG-USB.vendorID	CFG-USB-VENDOR_ID
UBX-CFG-USB.vendorString	CFG-USB-VENDOR_STR0, CFG-USB-VENDOR_STR1, CFG-USB-VENDOR_STR2, CFG-USB-VENDOR_STR3

Table 21: Legacy UBX message fields and the corresponding configuration items

Configuration defaults

The following tables contain the configuration defaults for the firmware. Some of these values may be changed in production. Refer to the integration manual for product-specific details.

Configuration item	Key ID	Type	Scale	Unit	Default value
CFG-I2C-ADDRESS	0x20510001	U1	-	-	132
CFG-I2C-EXTENDEDTIMEOUT	0x10510002	L	-	-	0 (false)
CFG-I2C-ENABLED	0x10510003	L	-	-	1 (true)

Table 22: CFG-I2C configuration defaults

Configuration item	Key ID	Type	Scale	Unit	Default value
CFG-INFMSG-UBX_UART1	0x20920002	X1	-	-	0x00
CFG-INFMSG-UBX_USB	0x20920004	X1	-	-	0x00

Table 23: CFG-INFMSG configuration defaults

Configuration item	Key ID	Type	Scale	Unit	Default value
CFG-MSGOUT-UBX_RXM_QZSSL6_UART1	0x2091033b	U1	-	-	1
CFG-MSGOUT-UBX_RXM_QZSSL6_USB	0x2091033d	U1	-	-	1

Table 24: CFG-MSGOUT configuration defaults

Configuration item	Key ID	Type	Scale	Unit	Default value
CFG-QZSS-L6_SVIDA	0x20370020	I1	-	-	0
CFG-QZSS-L6_SVIDB	0x20370030	I1	-	-	0
CFG-QZSS-L6_MSGA	0x20370050	E1	-	-	0 (L6D)
CFG-QZSS-L6_MSGB	0x20370060	E1	-	-	0 (L6D)
CFG-QZSS-L6_RSDECODER	0x20370080	E1	-	-	2 (ERRCORRECT)

Table 25: CFG-QZSS configuration defaults

Configuration item	Key ID	Type	Scale	Unit	Default value
CFG-UART1-BAUDRATE	0x40520001	U4	-	-	9600
CFG-UART1-STOPBITS	0x20520002	E1	-	-	1 (ONE)
CFG-UART1-DATABITS	0x20520003	E1	-	-	0 (EIGHT)
CFG-UART1-PARITY	0x20520004	E1	-	-	0 (NONE)
CFG-UART1-ENABLED	0x10520005	L	-	-	1 (true)

Table 26: CFG-UART1 configuration defaults

Configuration item	Key ID	Type	Scale	Unit	Default value
CFG-UART1INPROT-UBX	0x10730001	L	-	-	1 (true)
CFG-UART1INPROT-RTCM2X	0x10730003	L	-	-	1 (true)

Table 27: CFG-UART1INPROT configuration defaults

Configuration item	Key ID	Type	Scale	Unit	Default value
CFG-UART1OUTPROT-UBX	0x10740001	L	-	-	1 (true)

Table 28: CFG-UART1OUTPROT configuration defaults

Configuration item	Key ID	Type	Scale	Unit	Default value
CFG-USB-ENABLED	0x10650001	L	-	-	1 (true)

Configuration item	Key ID	Type	Scale	Unit	Default value
CFG-USB-SELFPOW	0x10650002	L	-	-	1 (true)
CFG-USB-VENDOR_ID	0x3065000a	U2	-	-	5446
CFG-USB-PRODUCT_ID	0x3065000b	U2	-	-	425
CFG-USB-POWER	0x3065000c	U2	-	mA	0
CFG-USB-VENDOR_STR0	0x5065000d	X8	-	-	0x4120786f6c622d75 ("u-blox A")
CFG-USB-VENDOR_STR1	0x5065000e	X8	-	-	0x2e777777202d2047 ("G - www.")
CFG-USB-VENDOR_STR2	0x5065000f	X8	-	-	0x632e786f6c622d75 ("u-blox.c")
CFG-USB-VENDOR_STR3	0x50650010	X8	-	-	0x0000000000006d6f ("om\0\0\0\0\0")
CFG-USB-PRODUCT_STR0	0x50650011	X8	-	-	0x4720786f6c622d75 ("u-blox G")
CFG-USB-PRODUCT_STR1	0x50650012	X8	-	-	0x656365722053534e ("NSS rece")
CFG-USB-PRODUCT_STR2	0x50650013	X8	-	-	0x0000000072657669 ("iver\0\0\0\0")
CFG-USB-PRODUCT_STR3	0x50650014	X8	-	-	0x0000000000000000
CFG-USB-SERIAL_NO_STR0	0x50650015	X8	-	-	0x0000000000000000
CFG-USB-SERIAL_NO_STR1	0x50650016	X8	-	-	0x0000000000000000
CFG-USB-SERIAL_NO_STR2	0x50650017	X8	-	-	0x0000000000000000
CFG-USB-SERIAL_NO_STR3	0x50650018	X8	-	-	0x0000000000000000

Table 29: CFG-USB configuration defaults

Configuration item	Key ID	Type	Scale	Unit	Default value
CFG-USBINPROT-UBX	0x10770001	L	-	-	1 (true)
CFG-USBINPROT-RTCM2X	0x10770003	L	-	-	1 (true)

Table 30: CFG-USBINPROT configuration defaults

Configuration item	Key ID	Type	Scale	Unit	Default value
CFG-USBOUTPROT-UBX	0x10780001	L	-	-	1 (true)

Table 31: CFG-USBOUTPROT configuration defaults

Related documents

[1] NEO-D9C-00B Data sheet C2-Restricted, UBX-17053092
NEO-D9C-00A Data sheet C2-Restricted, UBX-20057098

[2] NEO-D9C integration manual, UBX-21031631

☞ For regular updates to u-blox documentation and to receive product change notifications please register on our homepage (https://www.u-blox.com).

Revision history

Revision	Date	Status / Comments
R01	23-Sep-2021	QZS 1.01 release
R02	26-Feb-2024	Maintenance release

Contact

u-blox AG

Address: Zürcherstrasse 68
8800 Thalwil
Switzerland

For further support and contact information, visit us at www.u-blox.com/support.