TROPIC01

ODN_TR01_app_006

Configuration Objects Application Note

Version: 1.2

Abstract

This application note provides necessary information about TROPIC01 Configuration Objects (CO). It describes what the COs are and how they can be used. Then, it describes the usage of the objects in the context of the TROPIC01 SDK (Libtropic). Finally, it provides an overview of how these objects are configured from the factory for each Part Number of the TROPIC01 device.

February 4, 2026

Contents

Glossary

- **MCU** – Micro-Controller Unit
- **OEM** – Original Equipment Manufacturer
- **ECC** – Elliptic Curve Cryptography
- **NVM** – Non-Volatile Memory
- **SDK** – Software Development Kit
- **OTP** – One Time Programmable memory
- **UAP** – User Access Privileges
- **CO** – Configuration Object
- **MAC** – Message Authentication Code
- **GPO** – General-Purpose Output
- **TRNG** – True Random Number Generator
- **R-Memory** – Reversible Memory (AntiFuse OTP)
- **I-Memory** – Irreversible Memory (Flash)
- **R-Config** – Reversible Configuration
- **I-Config** – Irreversible Configuration

1 Introduction

The user can configure TROPIC01 by modifying Configuration Objects (COs) in TROPIC01's non-volatile memories (I-Memory and R-Memory). The Configuration Objects allow the user to enable or disable certain features of TROPIC01 and manage User Access Privileges (UAP). For a detailed description of the features of TROPIC01, refer to **ODD_TR01_datasheet** [1].

The user can:

- Enable or Disable security sensors
- Manage rights to modify the COs
- Manage access privileges for various objects, such as:
  - ***Pairing keys*** (see [1] Section 5.2, Mutual Authentication)
  - ***User Data*** (see [1] Section 7.6, User Data Access)
  - ***ECC Keys*** (see [1] Section 7.7, Elliptic Curve Cryptography)
  - ***MAC And Destroy slots*** (see [1] Section 7.9, PIN Verification)
  - ***Monotonic Counters*** (see [1] Section 7.10, Monotonic Counters)
  - ***TRNG*** (see [1] Section 7.12, Random Number Generation)

and more ...

The configuration objects are defined at 2 levels of abstraction:

1. **Registers** – Encapsulate the configuration of a given feature, e.g. ECDSA Sign, User Data access etc. Each CO Register has 32 bits.
2. **Fields** – Each CO Register is divided into one or more CO Fields. The CO Fields are used for more detailed configuration of given feature, e.g. configuration of particular security sensor, or access privileges to particular range of user slots (ECC Keys, User Data slots etc.).

For an example of CO Register, its Fields and their meaning, see Subsection 3.3

The default value for all COs is 1 unless specified otherwise by the Part Number specific configuration (see Section **??**).

Configuration Objects are located at two locations:

- **I-Config** – **I**rreversible **Config**uration. Stored in I-Memory (OTP).
  Only $1 \rightarrow 0$ transition, therefore modification of a given CO Field in I-Config **cannot** be reverted.
- **R-Config** – **R**eversible **Config**uration. Stored in R-Memory (Flash).
  Both state transitions are possible ($1 \rightarrow 0$, $0 \rightarrow 1$). Therefore modification of a given CO Field in R-Config **can** be reverted.

Both locations share the same memory layout specified in **ODU_TR01_user_api** [2] Section 5, User Configuration Objects.
The final configuration used by TROPIC01 is then given by bit-wise AND of the I-Config and R-Config. If a CO is modified, the new configuration is used after the next power-up.

Why Separate I-Config and R-Config? Having the Configuration Objects in both Reversible and Irreversible forms allows the user to:

- Perform **temporary** configuration using R-Config, which can be reset to default values. This gives the user better flexibility, without the risk of forever damaging the device.
- Also, to perform **permanent** configuration using I-Config. This leads to increased security when implementing Ownership Transfer and User Access Privileges.

2 Use Cases and Applications

The main use case for the Configuration Objects is to implement User Access Privileges. Please see **ODD_TR01_datasheet** [1] Section 7.5, User Access Privileges and Ownership Transfer use case.

The following use case serves just as an example to illustrate how the COs may be used. Assume the following roles:

- **Tropic Square** – The device manufacturer. The owner of S_{H0} Pairing Key.
- **OEM** – The customer of Tropic Square who integrates TROPIC01 into their product. The owner of S_{H0}, S_{H1} and S_{H2} Pairing Keys.
- **End Customer** – The customer of the OEM who uses the OEM's product. Possible owner of the S_{H2} and S_{H3} Pairing Keys.

The OEM wants to provide the End Customer a system where the End Customer can:

- Verify the system is genuinely from the OEM (Identity Attestation).
- Sign a message with ECDSA or EdDSA with an ECC key that is guaranteed to be from the OEM.
- Store and generate their own ECC keys and sign messages with ECDSA/EdDSA.
- The End Customer can further restrict the privileges for any other possible users (e.g. implement admin and common user privileges)

Therefore the OEM performs the following actions:

1. Uses the S_{H0PRIV} to establish the first Secure Channel Session and populates the Pairing Key slots 1 and 2 with $S_{H1,2PUB}$. Then the OEM invalidates the S_{H0PUB}, so Tropic Square permanently loses the ownership of the device.
2. The OEM stores its ECC Key to slot 0, and restricts $S_{H0,2,3}$ from modifying them permanently in I-Config.
3. The OEM restricts $S_{H0,3}$ to ever access the ECC Keys in slots [0,7].
4. The OEM can set a recommended configuration (e.g. security sensors configuration) in R-Config.
5. The OEM then sends the private part of S_{H2} (S_{H2PRIV}) to the End User.

The End User then can:

1. Use the S_{H2} to establish a Secure Channel Session, and use the classic public-key challenge-response authentication to verify the OEM is indeed the owner of the ECC Key stored in the slot 0, and therefore the product is genuine OEM product.
2. The End User then stores their own ECC Key to e.g. slot 8, and restricts $S_{H0,1,3}$ from accessing it, ensuring that even the OEM cannot access the End User's ECC Keys.
3. The End User can also populate the last Pairing Key slot with S_{H3} and further restrict the privileges of S_{H3}. The owner of S_{H2} would then play the role of admin, and the future owner of only S_{H3} would play the role of common user with restricted privileges.
4. If the End User does not accept the recommended configuration from the OEM, they can change it in the R-Config (e.g. disable some security sensors in order to lower the power consumption).

3 Configuration Objects in Detail

Each CO Register can be categorized as:

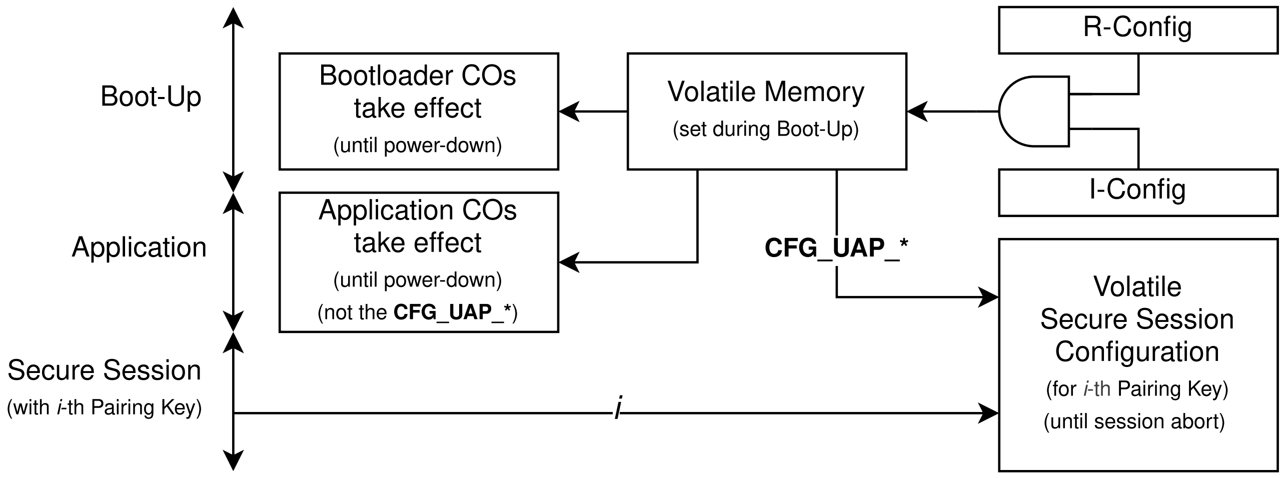
- **Bootloader** – Configuration used by bootloader. This configuration stays active also for the application (e.g. security sensors configuration).
- **Application** – Configuration used by the application.

ODD_TR01_datasheet [1] Section 7.5 specifies User Access Privileges (UAP) use case. For that purpose, all **CFG_UAP_*** CO Registers are categorized as:

- **Configuration** – Access privileges to manage:
  - I/R-Config
  - Pairing Keys
- **Functionality** – Access privileges to use:
  - User Data
  - Elliptic Curve Cryptography
  - MAC and Destroy PIN Verification
  - Monotonic Counters
  - Random Number Generation

3.1 How and When do the COs Take Effect?

The following figure explains how and when the configuration is applied.



3.2 List of Configuration Objects

Following table shows a list of all CO Registers. For detailed description of the COs at Field level, please refer to **ODU_TR01_user_api** [2] Section 5, User Configuration Objects.

CO Register Name	Description
Bootloader	
CFG_START_UP	Start-up tests and maintenance restart
CFG_SENSORS	Security sensors enablement
CFG_DEBUG	FW Logging
Application	
CFG_GPO	Configure GPO functionality
CFG_SLEEP_MODE	Reaction on ***Sleep_req*** L2 Request
Application – User Access Privileges – Configuration	
CFG_UAP_PAIRING_KEY_WRITE	***Pairing_Key_Write*** L3 Command access privileges
CFG_UAP_PAIRING_KEY_READ	***Pairing_Key_Read*** L3 Command access privileges
CFG_UAP_PAIRING_KEY_INVALIDATE	***Pairing_Key_Invalidate*** L3 Command access privileges
CFG_UAP_R_CONFIG_WRITE_ERASE	***R_Config_Write*** and ***R_Config_Erase*** L3 Command access privileges
CFG_UAP_R_CONFIG_READ	***R_Config_Read*** L3 Command access privileges
CFG_UAP_I_CONFIG_WRITE	***I_Config_Write*** L3 Command access privileges
CFG_UAP_I_CONFIG_READ	***I_Config_Read*** L3 Command access privileges
Application – User Access Privileges – Functional	
CFG_UAP_PING	***Ping*** L3 Command access privileges
CFG_UAP_R_MEM_DATA_WRITE	***R_Mem_Data_Write*** L3 Command access privileges
CFG_UAP_R_MEM_DATA_READ	***R_Mem_Data_Read*** L3 Command access privileges
CFG_UAP_R_MEM_DATA_ERASE	***R_Mem_Data_Erase*** L3 Command access privileges
CFG_UAP_RANDOM_VALUE_GET	***Random_Value_Get*** L3 Command access privileges
CFG_UAP_ECC_KEY_GENERATE	***ECC_Key_Generate*** L3 Command access privileges
CFG_UAP_ECC_KEY_STORE	***ECC_Key_Store*** L3 Command access privileges

CO Register Name	Description
CFG_UAP_ECC_KEY_READ	***ECC_Key_Read*** L3 Command access privileges
CFG_UAP_ECC_KEY_ERASE	***ECC_Key_Erase*** L3 Command access privileges
CFG_UAP_ECDSA_SIGN	***ECDSA_Sign*** L3 Command access privileges
CFG_UAP_EDDSA_SIGN	***EDDSA_Sign*** L3 Command access privileges
CFG_UAP_MCOUNTER_INIT	***MCounter_Init*** L3 Command access privileges
CFG_UAP_MCOUNTER_GET	***MCounter_Get*** L3 Command access privileges
CFG_UAP_MCOUNTER_UPDATE	***MCounter_Update*** L3 Command access privileges
CFG_UAP_MAC_AND_DESTROY	***MAC_And_Destroy*** L3 Command access privileges

! Warning !

R-Config and I-Config support up to 128 CO Registers each. Currently, not all this space is used, and is reserved for future use. However, TROPIC01 will not refuse to access the unused location. We strongly advise the user not to write to these locations, as it might take an unintended effect once Tropic Square releases an update, possibly introducing some new Configuration Objects.

3.3 CO Register Example

Here we provide an example of one CO Register, how it is divided into CO Fields, and how to set these CO Fields to achieve desired configuration.

Register name:		CFG_UAP_ECC_KEY_READ		
Address offset:		0x138		
Field	**Type**	**Reset value**	**Bits**	**Description**
READ_ECCKEY_SLOT_0_7	RW W1C	0xFF	7:0	Access privileges of the ***ECC_Key_Read*** L3 Command packet to ECC Key slots 0-7.
READ_ECCKEY_SLOT_8_15	RW W1C	0xFF	15:8	Access privileges of the ***ECC_Key_Read*** L3 Command packet to ECC Key slots 8-15.

READ_ECCKEY_SLOT_16_23	RW W1C	0xFF	23:16	Access privileges of the ***ECC_Key_Read*** L3 Command packet to ECC Key slots 16-23.
READ_ECCKEY_SLOT_24_31	RW W1C	0xFF	31:24	Access privileges of the ***ECC_Key_Read*** L3 Command packet to ECC Key slots 24-31.

Here, each CO Field is 8 bits. The READ_ECCKEY_SLOT_<A>_<B> Field controls what Pairing Keys can be used to read the ECC Public Key from the ECC Key slots in range [A, B]. Bit *i* of each Field controls the privileges for the *i*-th Pairing Key (also referred to as S_{Hi}).

- READ_ECCKEY_SLOT_0_7[*i*] = 1 ⇒ Owner of S_{Hi} **is** authorized to read the ECC Keys from slots [0, 7].
- READ_ECCKEY_SLOT_0_7[*i*] = 0 ⇒ Owner of S_{Hi} **is not** authorized to read the ECC Keys from slots [0, 7]. TROPIC01 responds with **UNAUTHORIZED** in the L3 Result packet.

Let's say you want to:

- Allow only the owner of S_{H0} to be able to read the ECC Key slots [0,7].
- Allow only the owner of S_{H1} or S_{H2} to be able to read the ECC Key slots [8,31].
- Forbid the owner of S_{H3} to read the ECC Key slots whatsoever.

Then the final value of the **CFG_UAP_ECDSA_SIGN** will be:

Field	**7-4 (Not used)**	**3 (S_{H3})**	**2 (S_{H2})**	**1 (S_{H1})**	**0 (S_{H0})**
READ_ECCKEY_SLOT_0_7	1111	0	0	0	1
READ_ECCKEY_SLOT_8_15	1111	0	1	1	0
READ_ECCKEY_SLOT_16_23	1111	0	1	1	0
READ_ECCKEY_SLOT_24_31	1111	0	1	1	0

4 TROPIC01 SDK Usage Examples

This section provides a guide how to use the TROPIC01 SDK (Libtropic [4]) to work with the Configuration Objects. Refer to Libtropic documentation [4] for a quick guide on how to start with TROPIC01 and Libtropic. We also recommend refering to the Libtropic API Reference for details about the used functions, types and other (part of the Libtropic documentation).
The TROPIC01 SDK provides the following set of functions to work with the Configuration Objects:

Function Name	Description
Core Functions	
`lt_r_config_write`	Write one CO Register in R-Config
`lt_r_config_read`	Read one CO Register in R-Config
`lt_r_config_erase`	Erase whole R-Config (set all COs to 1)
`lt_i_config_write`	Set one CO Field in I-Config to 0
`lt_i_config_read`	Read one CO Register in I-Config
Helper Functions (added by defining the `LT_HELPERS` configuration macro)	
`lt_write_whole_R_config`	Write whole R-Config using `lt_config_t` structure
`lt_read_whole_R_config`	Read whole R-Config to `lt_config_t` structure
`lt_write_whole_I_config`	Write whole I-Config using `lt_config_t` structure
`lt_read_whole_I_config`	Read whole I-Config to `lt_config_t` structure

Addresses of all CO Registers are given by the `lt_config_obj_addr_t` enum. All values inside the `lt_config_obj_addr_t` have names with the following format:

`TR01_CFG_<CO Register Name>_ADDR`

When using Libtropic to work with the Configuration Objects, make sure you include the following headers:

```
#include "libtropic.h"        // Core API function declarations
#include "libtropic_common.h" // Common macros, types, etc.
```

4.1 Error Handling

All the mentioned functions return an `lt_ret_t` return code. See the possible values in the Libtropic documentation [4], specifically the API Reference. In general, the functions may “fail” in following cases:

- The Secure Session is not active or is not authorized to read or modify the CO Register.

- The address of the specified CO register is not aligned to a multiple of 4, or the address is out of range.

4.2 R-Config How to

The default value for all COs in the R-Config is 1 unless specified otherwise by the Part Number specific configuration (see Section **??**). The 'R' in R-Config stands for 'Reversible', so if a given CO Field is set from 1 to 0, it **can** be set back to 1 again.

! Warning !

R-Config lives in one Flash memory sector. Due to the nature of Flash memory, if a CO Field shall be set from 0 back to 1, the whole R-Config must be erased (set to 1) first, and then written back with the modification. Otherwise the state of R-Config is undefined!

We strongly recommend to use the `lt_config_t` structure provided by Libtropic as a mirror of the R-Config, and any modification should be performed as follows:

1. Read the whole R-Config into `lt_config_t` using the:
   (a) function `lt_r_config_read` in a `for` loop, if you do not want to use Libtropic helper functions, or the
   (b) helper function `lt_read_whole_R_config`.
2. Prepare the desired state of R-Config in the `lt_config_t` structure.
3. Erase the whole R-Config using the `lt_r_config_erase` function.
4. Write the whole R-Config using the:
   (a) function `lt_r_config_write` in a `for` loop, if you do not want to use Libtropic helper functions, or the
   (b) helper function `lt_write_whole_R_config` and your edited instance of the `lt_config_t` structure.

See how to work with the whole R-Config in Section 4.2.4.
The examples that are about to follow assume the following:

1. An instance of `lt_handle_t`, refered to as `handle`, has already been initialized.
2. Secure Channel Session has already been established, so R-Config can be modified.

3. The headers `libtropic.h` and `libtropic_common.h` have already been included (as mentioned in Section 4). The examples will not include them to save space and will include only headers for additional functionality, not directly needed to work with COs.

4. The permissions to read, write or erase R-Config or I-Config have not been restricted.

For hints on how to do these steps, refer to the *Tutorials* → *Model* → *HW Wallet* example tutorial in the Libtropic documentation [4].

4.2.1 How to Write Single CO Register in R-Config

To write single CO Register in R-Config, use the `lt_r_config_write` function:

```
/**
 * @brief Writes configuration object specified by `addr`
 *
 * @param h           Device's handle
 * @param addr        Address of a config object
 * @param obj         Content to be written
 *
 * @retval            LT_OK Function executed successfully
 * @retval            other Function did not execute successfully, you might use
    lt_ret_verbose() to get verbose encoding of returned value
 */
lt_ret_t lt_r_config_write(lt_handle_t *h, const enum lt_config_obj_addr_t addr, const
    uint32_t obj);
```

Use the function as follows. This example sets the **CFG_UAP_ECC_KEY_STORE** CO Register, so:

- only S_{H0} can store key into ECC Key slots 0-7,
- only S_{H1} and S_{H2} can store key into ECC Key slots 8-31.

```
lt_ret_t ret;
uint32_t r_config_cfg_uap_ecc_key_store;

// Prepare the CO Register content using the macros from libtropic_common.h
r_config_cfg_uap_ecc_key_store = (
    // Slots 0-7, only SH0 pairing key has access
    TO_ECC_KEY_SLOT_0_7     (SESSION_SH0_HAS_ACCESS) |
    // Slots 8-31, only SH1 and SH2 pairing keys have access
    TO_ECC_KEY_SLOT_8_15    (SESSION_SH1_HAS_ACCESS | SESSION_SH2_HAS_ACCESS) |
    TO_ECC_KEY_SLOT_16_23   (SESSION_SH1_HAS_ACCESS | SESSION_SH2_HAS_ACCESS) |
    TO_ECC_KEY_SLOT_24_31   (SESSION_SH1_HAS_ACCESS | SESSION_SH2_HAS_ACCESS)
);

// Write the prepared CO Register to the R-Config
ret = lt_r_config_write(&handle, TR01_CFG_UAP_ECC_KEY_STORE_ADDR,
    r_config_cfg_uap_ecc_key_store);

if (ret != LT_OK) {
    // Your error handling
    return -1;
}
```

! Warning !

Use this function carefully. The assumption is that all Fields of the CO Register are 1. This function alone **cannot** be used to set a CO Field from 0 back to 1!

4.2.2 How to Read Single CO Register in R-Config

To read single CO Register in R-Config, use the `lt_r_config_read` function:

```
/**
 * @brief Reads configuration object specified by `addr`
 *
 * @param h             Device's handle
 * @param addr          Address of a config object
 * @param obj           Variable to read content into
 *
 * @retval              LT_OK Function executed successfully
 * @retval              other Function did not execute successully, you might use
    lt_ret_verbose() to get verbose encoding of returned value
 *
 */
lt_ret_t lt_r_config_read(lt_handle_t *h, const enum lt_config_obj_addr_t addr, uint32_t *
    obj);
```

Use the function as follows. This example reads the state of **CFG_SENSORS** CO Register to the `r_config_cfg_sensors` variable, and checks if the temperature sensor is enabled in the R-Config.

```c
#include "bits.h"

lt_ret_t ret;
uint32_t r_config_cfg_sensors;

// Read the CO Register from R-Config
ret = lt_r_config_read(&handle, TR01_CFG_SENSORS_ADDR, &r_config_cfg_sensors);

if (ret != LT_OK) {
    // Your error handling
    return -1;
}

// Check the state of the temperature sensor CO Field
// Use the *_MASK macro from tropic01_bootloader_co.h (included by libtropic_common.h)
// Use the FIELD_GET macro from bits.h
uint32_t cfg_temp_sens_dis = FIELD_GET(
    BOOTLOADER_CO_CFG_SENSORS_TEMP_SENS_DIS_MASK,
    r_config_cfg_sensors
);

if (cfg_temp_sens_dis)
    // The temperature sensor is DISABLED in R-Config
else
    // The temperature sensor is ENABLED in R-Config
```

4.2.3 How to Erase the Whole R-Config

To erase the whole R-Config, use the `lt_r_config_erase` function:

```c
/**
 * @brief Erases all configuration objects
 *
 * @param h            Device's handle
 *
 * @retval             LT_OK Function executed successfully
 * @retval             other Function did not execute successfully, you might use
    lt_ret_verbose() to get verbose encoding of returned value
 */
lt_ret_t lt_r_config_erase(lt_handle_t *h);
```

4.2.4 How to Work with the Whole R-Config

We strongly recommend to use the `lt_config_t` structure provided by Libtropic as a mirror of the R-Config, and always work with the R-Config as a whole. Here we give a simplified example of how to use the `lt_config_t` structure together with the helper functions. The used helper functions are just wrappers around the core API functions from the examples above, so you can use those if you do not want to use the helpers.

Read To read the whole R-Config out of TROPIC01 use the `lt_read_whole_R_config` helper function:

```
/**
 * @brief Reads all of the R-Config objects into ‘config‘.
 *
 * @param h            Device’s handle
 * @param config       Struct into which objects are readed
 *
 * @retval             LT_OK Function executed successfully
 * @retval             other Function did not execute successully, you might use
    lt_ret_verbose() to get verbose encoding of returned value
 */
lt_ret_t lt_read_whole_R_config(lt_handle_t *h, struct lt_config_t *config);
```

and use it as follows:

```
#include <stdio.h>

lt_ret_t ret;
lt_config_t r_config;

// Read the R-Config to the r_config structure
ret = lt_read_whole_R_config(&handle, &r_config);

if (ret != LT_OK) {
    // Your error handling
    return -1;
}

// You can print the read R-Config content as follows.
// The cfg_desc_table is defined in libtropic.h.
for (int i = 0; i < LT_CONFIG_OBJ_CNT; i++) {
    printf("%s: 0x%08" PRIX32 "\n", cfg_desc_table[i].desc, r_config.obj[i]);
}
```

Modify Now, the content of the R-Config is in the `r_config` structure. In `r_config`, you can access the read values of CO Registers using the indexes in the `lt_config_obj_idx_t` enum, declared in `libtropic_common.h`.
Following example gets the value of **CFG_SENSORS** CO Register, and disables only the temperature sensor.

```
#include "bits.h"

// Do read-modify-write on the CO Register.
// Use the *_MASK macro from tropic01_bootloader_co.h (included by libtropic_common.h)
// Use the FIELD_SET macro from bits.h
uint32_t r_config_cfg_sensors = r_config.obj[TR01_CFG_SENSORS_IDX];

r_config_cfg_sensors = FIELD_SET(
    r_config_cfg_sensors,
    BOOTLOADER_CO_CFG_SENSORS_TEMP_SENS_DIS_MASK,
    1 // Disable
);

r_config.obj[TR01_CFG_SENSORS_IDX] = r_config_cfg_sensors;
```

Write After you modify all COs in the `r_config` structure to your desired configuration, you can write it back to TROPIC01 using the `lt_write_whole_R_config` helper function:

```
/**
 * @brief Writes the whole R-Config with the passed ‘config‘.
 *
 * @param h           Device’s handle
 * @param config      Array into which objects are read
 *
 * @retval            LT_OK Function executed successfully
 * @retval            other Function did not execute successully, you might use
   lt_ret_verbose() to get verbose encoding of returned value
 */
lt_ret_t lt_write_whole_R_config(lt_handle_t *h, const struct lt_config_t *config);
```

As stated before, to write to CO Registers in R-Config that are already set to 0, you **must** erase it before. Write the contents of the `r_config` structure to TROPIC01 as follows:

```
lt_ret_t ret;

// Erase the R-Config in TROPIC01
ret = lt_r_config_erase(&handle);

if (ret != LT_OK) {
    // Your error handling
    return -1;
}

// Write the r_config structure to the R-Config
ret = lt_write_whole_R_config(&handle, &r_config);

if (ret != LT_OK) {
    // Your error handling
    return -1;
}
```

! Warning !

Even if the R-Config is reversible, incorrect configuration configuration might lead to irreversible consequences. If the user disables modification of both R and I-Config for all Pairing Keys, there is no way to modify the Configuration Objects ever again, and the device will retain the configuration permanently!

4.3 I-Config How to

The default value for all COs in the I-Config is 1 unless specified otherwise by the Part Number specific configuration (see Section **??**). The ’I’ in I-Config stands for ’Irreversible’, so if a given CO Field is set to 0, it **cannot** be set back to 1 ever again.

4.3.1 How to Set Single CO Field bit in I-Config to 0

To set a single CO Field in I-Config to 0, use the `lt_i_config_write` function of Libtropic. Because the modification of CO Field in I-Config is irreversible, the function takes only the bit index (0-31) of the CO Field in the CO Register given by `addr` which should be set to 0.

```
/**
 * @brief Writes configuration object specified by 'addr' to I-Config
 *
 * @param h           Device's handle
 * @param addr        Address of a config object
 * @param bit_index   Index of bit to write from 1 to 0
 *
 * @retval            LT_OK Function executed successfully
 * @retval            other Function did not execute successfully, you might use
    lt_ret_verbose() to get verbose encoding of returned value
 */
lt_ret_t lt_i_config_write(lt_handle_t *h, const enum lt_config_obj_addr_t addr, const
    uint8_t bit_index);
```

Use the function as follows. This example permanently forbids the S_{H2} Pairing Key to use the ECC Key slots [8,15] for ECDSA.

```
lt_ret_t ret;

// Get the position (index) of the bit in the CO Register
// Use the *_POS macro from tropic01_application_co.h (included by libtropic_common.h)
uint8_t s_hi_offset = 2; /* Offset of the SH2 Pairing Key in the CO Field */
uint8_t cfg_bit_idx = APPLICATION_CO_CFG_UAP_ECDSA_SIGN_ECDSA_ECCKEY_SLOT_8_15_POS +
    s_hi_offset;

// Set the bit to 0
ret = lt_i_config_write(&handle, TR01_CFG_UAP_ECDSA_SIGN_ADDR, cfg_bit_idx);

if (ret != LT_OK) {
    // Your error handling
    return -1;
}
```

4.3.2 How to Read Single CO Register in I-Config

Reading a CO Register in I-Config works the same as in case of R-Config. Use the `lt_i_config_read` function of Libtropic:

```
/**
 * @brief Reads configuration object specified by 'addr' from I-Config
 *
 * @param h           Device's handle
 * @param addr        Address of a config object
 * @param obj         Variable to read content into
 *
 * @retval            LT_OK Function executed successfully
 * @retval            other Function did not execute successfully, you might use
    lt_ret_verbose() to get verbose encoding of returned value
 */
lt_ret_t lt_i_config_read(lt_handle_t *h, const enum lt_config_obj_addr_t addr, uint32_t *
    obj);
```

Use the function as follows. This example reads the state of **CFG_SENSORS** CO Register to the `i_config_cfg_sensors` variable, and checks if the temperature sensor is enabled in the I-Config.

```
#include "bits.h"

lt_ret_t ret;
uint32_t i_config_cfg_sensors;

// Read the CO Register from I-Config
ret = lt_i_config_read(&handle, TR01_CFG_SENSORS_ADDR, &i_config_cfg_sensors);

if (ret != LT_OK) {
    // Your error handling
    return -1;
}

// Check the state of the temperature sensor CO Field
// Use the *_MASK macro from tropic01_bootloader_co.h (included by libtropic_common.h)
// Use the FIELD_GET macro from bits.h
uint32_t cfg_temp_sens_dis = FIELD_GET(
    BOOTLOADER_CO_CFG_SENSORS_TEMP_SENS_DIS_MASK,
    i_config_cfg_sensors
);

if (cfg_temp_sens_dis)
    // The temperature sensor is DISABLED in I-Config
else
    // The temperature sensor is ENABLED in I-Config
```

4.3.3 How to Work with the Whole I-Config

Unlike with R-Config, we strongly recommend to use the `lt_i_config_write` function to modify a CO Field in the I-Config. Because the modification **cannot** be reverted, the user must be very careful when doing any modification in the I-Config.

Read You can read-out the whole I-Config with the `lt_read_whole_I_config` helper function:

```
/**
 * @brief Reads all of the I-Config objects into ‘config‘.
 *
 * @param h           Device’s handle
 * @param config      Struct into which objects are readed
 *
 * @retval            LT_OK Function executed successfully
 * @retval            other Function did not execute successully, you might use
    lt_ret_verbose() to get verbose encoding of returned value
 */
lt_ret_t lt_read_whole_I_config(lt_handle_t *h, struct lt_config_t *config);
```

and use the function as follows:

```
lt_ret_t ret;
lt_config_t i_config;

// Read the I-Config to the i_config structure
ret = lt_read_whole_I_config(&handle, &i_config);

if (ret != LT_OK) {
    // Your error handling
    return -1;
}

// Access a given CO Field as follows.
// Use the *_MASK macro from tropic01_bootloader_co.h or tropic01_application_co.h (included
     by libtropic_common.h)
// Use the FIELD_GET macro from bits.h
uint32_t cfg_temp_sens_dis = FIELD_GET(
    BOOTLOADER_CO_CFG_SENSORS_TEMP_SENS_DIS_MASK,
    i_config.obj[TR01_CFG_SENSORS_IDX]
);
```

Write You can prepare your desired I-Config configuration into the `lt_config_t` structure and then write it to the I-Config at once with the `lt_write_whole_I_config` function:

```
/**
 * @brief Writes the whole I-Config with the passed `config`.
 * @details Only the zero bits in `config` are written.
 *
 * @param h           Device's handle
 * @param config      Array into which objects are read
 *
 * @retval            LT_OK Function executed successfully
 * @retval            other Function did not execute successully, you might use
    lt_ret_verbose() to get verbose encoding of returned value
 */
lt_ret_t lt_write_whole_I_config(lt_handle_t *h, const struct lt_config_t *config);
```

Unlike the R-Config, there is no erase needed before the write, since the I-Config cannot be set to its initial state (all 1s).

! Warning !

Be really careful when using the `lt_write_whole_I_config` function as you may permanently damage the device with an incorrect configuration. For example, accidentally burning all bits of **CFG_UAP_MAC_AND_DESTROY** CO Register to 0 will disable the entire MAC and Destroy functionality permanently.

Version history

Version	Date	Description
1.0	30.9.2025	Initial public release.
1.1	28.10.2025	Sync all Libtropic parts with release v2.0.1.
1.2	30.1.2026	Sync all Libtropic parts with release v3.1.0.

References

[1] ODD_TR01_datasheet, TROPIC01 Datasheet, Tropic Square

[2] ODU_TR01_user_api, TROPIC01 User API, Tropic Square

[3] OD_TR01_catalog_list, TROPIC01 Catalog list, Tropic Square

[4] TROPIC01 SDK – Libtropic,
GitHub: `https://github.com/tropicsquare/libtropic`
Documentation: `https://tropicsquare.github.io/libtropic/`

Legal Notice