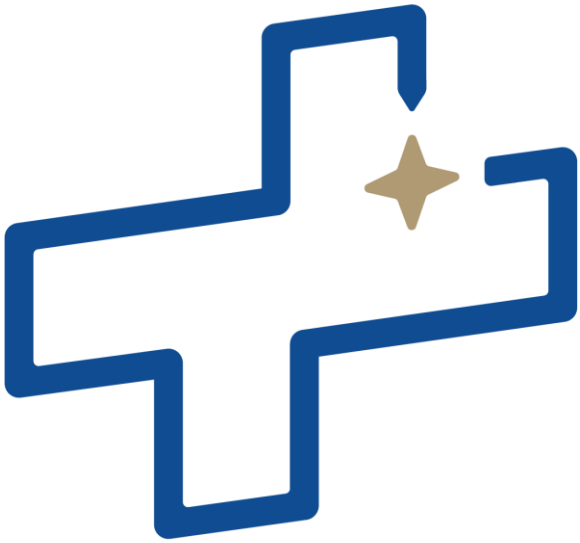

CUBESPACE

CubeObcLib

User Manual

DOCUMENT NUMBER	CS-DEV.UM.COL-01
VERSION	1.01
DATE	15/09/2023
PREPARED BY	O. GRIES
REVIEWED BY	L. Visagie
APPROVED BY	W. Morgan
DISTRIBUTION LIST	External

Revision History

VERSION	AUTHORS	DATE	DESCRIPTION
1.00	O. Gries	02/06/2023	First draft
1.01	O. Gries	15/09/2023	Add option to pack TCTLM structures. Changed context scheme to use void* for user data.

Reference Documents

The following documents are referenced in this document.

[1] CS-DEV.FRM.CA-01 CubeProduct Firmware Reference Manual

[2] CS-DEV.AMNL.BL-01 Bootloader Application Manual

List of Acronyms/Abbreviations

ACP	ADCS Control Program
ADCS	Attitude Determination and Control System
CAN	Controller Area Network
COTS	Commercial Off The Shelf
CSS	Coarse Sun Sensor
CVCM	Collected Volatile Condensable Materials
DUT	Device Under Test
EDAC	Error Detection and Correction
EHS	Earth Horizon Sensor
EM	Engineering Model
EMC	Electromagnetic Compatibility
EMI	Electromagnetic Interference
ESD	Electrostatic Discharge
FDIR	Fault Detection, Isolation, and Recovery
FM	Flight Model
FSS	Fine Sun Sensor
GID	Global Identification
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GYR	Gyroscope
I2C	Inter-Integrated Circuit
ID	Identification
LTDN	Local Time of Descending Node
LEO	Low Earth Orbit
MCU	Microcontroller Unit
MEMS	Microelectromechanical System
MTM	Magnetometer
MTQ	Magnetorquer
NDA	Non-Disclosure Agreement
OBC	On-board Computer

PCB	Printed Circuit Board
RTC	Real-Time Clock
RWA	Reaction Wheel Assembly
RW	Reaction Wheel
SBC	Satellite Body Coordinate
SOFIA	Software Framework for Integrated ADCS
SPI	Serial Peripheral Interface
SRAM	Static Random-Access Memory
SSP	Sub-Satellite Point
STR	Star Tracker
TC	Telecommand
TCTLM	Telecommand and Telemetry (protocol)
TID	Total Ionizing Dose
TLM	Telemetry
TML	Total Mass Loss
UART	Universal Asynchronous Receiver/Transmitter

Table of Contents

Table of Tables

1 Introduction

CubeObcLib is a bare-metal library, written in C, to facilitate the integration of CubeSpace products to a client OBC.

The library may be compiled and linked to the OBC code and used as is, or the user may use the library as a guide on implementing their own code.

The library provides, but is not limited to, the following functionality:

- Communication protocol handling (CAN, UART, etc..).
- Packing and parsing of all TCTLM, for all CubeProduct API's.
- Wrapper functions that perform sequences of operations as needed by the firmware implementations. The user should explore the library for such available functions. Some examples are:
    - Bulk Data Transfer (see [1])
    - Firmware binary file uploads to bootloader (using Bulk Data Transfer)
    - Event log download from CubeComputer (using Bulk Data Transfer)
    - Telemetry log download from CubeComputer (using Bulk Data Transfer)
    - Image log download from CubeComputer (using Bulk Data Transfer)

1.1 Folder Structure

libcubeobc/			
	api/		
		docs/	HTML files for each API definition formatted into tables for easy reading. These files are generated from the API xml files.
		src/	C source and header files for each API definition. These files define functions for each TCTLM message. The functions handle packing/parsing the data as per the definition of each message. These functions call the master comms handler to also perform the transaction. These files are compiled into the library.
		xml/	API definitions in xml format. These files are the source of all API information.
	examples/		
	include/		
	src/		

		arch/	Architecture specific functions required by the library.
		drivers/	Hardware drivers for comms peripherals. Note that these modules are not called by the library and serve only to aid in running the examples.
		interfaces/	Communications interfaces (CAN, UART, etc..). These are “interfaces” to the OBC code, via weak definitions that the OBC should override, to link the library to hardware peripherals.
	Makefile		
	systemversion		The system version tied to the software bundle.
	version		Version of the library code itself.

2 Build System

The library uses make as the build system.

2.1 Build Targets

Table 1: Build Targets

TARGET	DESCRIPTION
Default	**The default build target is "libcubeobc".**
libcubeobc	**Compiles the library to output "libcubeobc.a" shared library.**
libcubeobc-examples	**Compiles all examples to <example>.elf. Requires libsocketcan and termios.**
libcubeobc-clean	**Cleans the build directory.**

2.2 Build Options

Table 2: Build Options

OPTION	VALUE	DEFAULT	DESCRIPTION
BUILD_DIR	**<path>**	**libcubeobc/build**	**Specifies the build directory.**
CC	**<name>**	**gcc**	**Compiler**
DEBUG	**0/1**	**0**	**0 - DEBUG build, no optimization** **1 - RELEASE build, -o1 optimization**
ARCH	**<name>**	**posix**	**Specifies the target architecture. This determines which source files are compiled.**
SOCKETCAN	**0/1**	**0**	**0 - No socketcan** **1 - Use socketcan (required for examples)**
LIBSOCKETCAN_SH	**<path>**	**/usr/local/lib/libsocketcan.a**	**Path to libsocketcan shared library. Required to use socketcan.**
TERMIOS	**0/1**	**0**	**0 - No termios** **1 - Use termios (required for examples)**
VERBOSE	**0/1**	**0**	**0 - Quiet build** **1 - Print commands**
USE_TCTLM_PACKED	**0/1**	**0**	**0 - TCTLM structures are not packed** **1 - TCTLM structures are packed**

2.3 Build Examples

$ make libcubeobc-examples SOCKETCAN=1 TERMIOS=1

3 Library Usage

The library is bare-metal, which dramatically reduces complexity. The flow of data is straight forward. All function calls are blocking and only return once the operation is complete.

All public library functions are exposed via the “cubeObc.h” header file, excluding the API source files which need to be included explicitly for each API definition the user wants to make use of.

Callbacks to the OBC are achieved using weak definitions instead of function pointers. The OBC code must override the weak definitions that are required for the desired functionality.

Memory usage is static. However, the library does require frame buffer allocation for certain functionality. Callbacks are defined to request frame buffers from the OBC. It is left to the OBC on how to handle allocating the frame buffers, however, this can be done statically as well.

3.1 Initialization

Before the library can be used it must be initialized with configuration parameters.

See cubeObc_init.c.

3.2 Endpoints

An endpoint defines where data is routed, as well as other information regarding communications handling.

Routing is determined by the selected peripheral type (CAN, UART, etc..), and the address (if bus transport).

3.3 General Data Flow

When the OBC wants to send a telecommand or request telemetry directly using the API source files, the data flow is as follows:

```
OBC <-> API function <-> Master Comms <-> Interface Callbacks <-> OBC Peripheral
```

1. OBC defines endpoint and calls an API function for the desired telecommand or telemetry request.
2. API function packs/parses data to/from buffer. Calls master comms to transfer data.
3. Master comms formats the data for the communications protocol. Calls interfaces transmit/receive.
4. OBC transmits/receives data on endpoint peripheral.
5. Result is propagated back up to OBC caller.

When the OBC wants to perform a sequence of operations implemented by the library, the data flow is as follows:

```
OBC <-> Public Function <-> API function(s) <-> Master Comms <-> Interface Callbacks <-> OBC Peripheral
```

1. OBC defines endpoint and calls a public function for desired operation.

2. The public function will call all necessary API functions to send commands and request telemetry from the ADCS, as required by the operation sequence.
3. API function packs/parses data to/from buffer. Calls master comms to transfer data.
4. Master comms formats the data for the communications protocol. Calls interfaces transmit/receive.
5. OBC transmits/receives data on endpoint peripheral.
6. Result is propagated back up to OBC caller.

3.4 Communications Protocols

Communication protocol handling is done in cubeObc_tctlmCommsMasterSvc.c.

For documentation on the communications protocols, refer to [1].

3.5 WEAK Overrides

3.5.1 Communications Endpoint

Overrides for communications endpoints (CAN, UART, etc..) are defined in the interface files located under libcubeobc/src/interfaces/.

Note that the receive (rx) implementation in the OBC must be non-blocking. i.e. if there is no data available on the interface, it should return immediately regardless. A timeout is implemented internally, and if no data is received in time, an error will be propagated up to the caller.

3.5.2 Frame Buffer Allocation

Frame buffer allocation is required when performing Bulk Data Transfers. The data being transferred either needs to be read from OBC storage and uploaded or downloaded and stored by the OBC.

The library uses a request/commit scheme for handling the frame buffers. The library will request a frame buffer of a certain size, then make use of it, and then commit it to signal that frame buffer has been used.

The library does not keep track of the buffer index. This is left to the OBC. i.e. When the library commits a frame buffer, the OBC must update the buffer index so that when the next frame buffer is requested, it is in fact the next frame of data. The frame buffers are used as follows for uploading and downloading data:

- While uploading a file to the ADCS: If a frame buffer is requested, the OBC reads the requested frame size from the file in storage and gives the populated frame to the library. The library sends the frame and when done commits the frame so that the OBC can update the buffer index.
- While downloading data from the ADCS: If a frame buffer is requested, the OBC must simply allocate some memory of the requested size and give the buffer to the library. The library then downloads a frame into this buffer and when done commits the frame buffer so that the OBC can write it to storage and update the buffer index.

The relevant weak definitions are defined in cubeObc_bulkDataTransfer.c.

Request frame buffer: cubeObC_bulkDataTransfer_getFrameBuffer()

Commit frame buffer: cubeObC_bulkDataTransfer_commitFrameBuffer()

3.5.3 User Data

Weak definitions may define a “userData” argument. This is an arbitrary void* that is specified by the OBC and can be used to pass data to the callback. E.g., For Bulk Data Transfer, this can be used to pass the file to target in storage (upload/download).

Note that the OBC does not set the “userData” in the callback. If there is a public function defined (not weak) with a “userData” argument, the OBC specifies the “userData” when calling the function, which is then propagated to the callback(s).

In cases where the library is used to perform a sequence of operations at a higher level, the “userData” may be set by the library.