UDO - Universal Device Objects

First Published:	2023-04-30
Publish Site:	github.com/nvitya/udo
Original Author:	Viktor Nagy
License Type	MIT (Open Source)
Actual Version, date:	1.0, 2023-04-30

Basics

The UDO is a simple, universal, binary master-slave data exchange system.
The slave data selected by a 16-bit address and an optional 32-bit offset.
The offset is a data byte offset by default, but the implementation can vary from object to object, like using as a sub-object identifier.

Targeted communication mediums:

- UART serial communication
- USB (serial communication emulation)
- IP (UDP)

Common (Mandatory) Device Objects

Obj. (hex)	**Type**	**R/W**	**Function**
Test Objects, Generic device Information (Mandatory)			
0000	u32	R	Communication Test Object, fix 0x66CCAA55
0001	u32	R	Maximal data (payload) length: 64 - 1024. The smaller devices might not support the maximal 1024 byte payload.
0002	blob	RW	Blob Test Object, data size = 262144 bytes It returns 16384 * 32-bit integer numbers in ascending order beginning with 0. The OFFSET handled as byte offset here. Its main purpose to test the device large data transfers. By writing an error code will be responded if the written data does not corresponds the proper incrementing numbers.
0003	str[32]	R	Device SW (Firmware) Identification Useful for pre-configuration check.
0004	u32	R	Device SW version
Device Generic Setup (Mandatory)			
0010	u8	RW	Configuration Status and Control

			Read and Write: 0 = CONFIG mode: outputs should be safe, configuration objects are writeable 1 = RUN mode: outputs activated, configuration objects are read only Write only: 2 = restart device (keeps the actual CONFIG/RUN status) Writing value 1 saves the configuration into the internal non-volatile storage. When the device starts loads the configuration and goes into RUN mode then. Writing value 0 does not clears the previous configuration, and when the device restarts it might start in RUN mode again. Beware, that too frequent configuration save might wear out the Flash memory used for storage in the devices. Generic device in the manufactured state or without valid configuration must be CONFIG mode.
0011	str[32]	RW	Device ID Used as the USB Device ID too. Recommended to assign a unique ID for different configurations and / or different pieces.
0012	u16	RW	USB Vendor ID
0013	u16	RW	USB Product ID
0014	str[32]	RW	Manufacturer Name
0015	str[32]	RW	Serial Number Used as the USB Device Serial number too.

Serial Request Format

Bytes	**Segment ID**	**Description**
1	SYNC	sync byte, always = 0x55
1	OFFSLEN, MLEN, LEN, RW	Read/Write and length information bit0..1: OFFSLEN: offset length 0=0, 1=1, 2=2, 3=4 bit2..3: MLEN: metada length, 0=0, 1=1 byte, 2=2, 3=4 bit4..6: LEN: read or write length, if LEN = 7 then EXTLEN follows 0=0, 1=1, 2=2, 3=4, 4=8, 5=16, 6=INVALID, 7=EXTLEN bit7: RW: 0 = read, 1=write
0 | 2	EXTLEN	16-bit extended length, present only if LEN = 7
2	INDEX	16-bit Object Index
0 - 4	OFFSET	0 - 32 bit offset, if OFFSLEN > 0
0 - 4	METADATA	Optional metadata,present only if MLEN > 0

		Can be useful for extra request parameters
0 - 1024	WDATA	Write data. Present only if RW=1, length determined by LEN or EXTLEN (when LEN = 15).
1	CRC	The CRC calculated for all the previous bytes (including the SYNC byte). The CRC8 generator polinom is 0x07.

Serial Response Format

Bytes	**Segment ID**	**Description**
1	SYNC	sync byte, always = 0x55
1	OFFSLEN, MLEN, LEN, RW	Read/Write and length information bit0..1: OFFSLEN: offset length 0=0, 1=1, 2=2, 3=4 bit2..3: MLEN: metada length, 0=0, 1=1 byte, 2=2, 3=4 bit4..6: LEN: read or write length, if LEN = 7 then EXTLEN follows 0=0, 1=1, 2=2, 3=4, 4=8, 5=16, 6=ERROR CODE, 7=EXTLEN bit7: RW: 0 = read, 1=write
0 | 2	EXTLEN	16-bit extended length, present only if LEN = 7
2	INDEX	16-bit Object Index
0 - 4	OFFSET	0 - 32 bit offset (mirrored from the request), if OFFSLEN > 0
0 - 4	METADATA	Optional metadata (mirrorred from request), present only if MLEN > 0.
0 | 2	ECODE	16-bit error code, present only if ERR = 1
0 - 1024	RDATA	Read data. Present only if RW=0 (read), and ERR = 0. Length determined by LEN or EXTLEN (when LEN = 15).
1	CRC	The CRC calculated for all the previous bytes (including the SYNC byte). The CRC8 with generator polinom is 0x07.

IP Request Format

Bytes	**Segment ID**	**Description**
4	RQID	Request ID
2	LEN, MLEN, RW	Data Length + CMD bit0..10: LEN (0..1024), LEN > 1024 = INVALID bit11..12: reserved bit13..14: MLEN: metada length, 0=0, 1=1 byte, 2=2, 3=4 bit15: RW: 0 = read, 1 = write
2	INDEX	16-bit Object Index
4	OFFSET	0 - 32 bit offset

4	METADATA	Metadata, meaningful only when MLEN > 0 Can be useful for extra request parameters 4 bytes will be used when MLEN=1, or MLEN=2
0 - 1024	WDATA	Write data. Present only if RW=1, length determined by LEN

IP Response Format

Bytes	**Segment ID**	**Description**
4	RQID	Request ID
2	LEN, MLEN, RW	Data Length + CMD bit0..10: LEN: if 0..1024 response data length, 0x7FF = ERROR CODE bit11..12: reserved bit13..14: MLEN: metada length, 0=0, 1=1 byte, 2=2, 3=4 bit15: RW: 0 = read, 1 = write
2	INDEX	16-bit Object Index
4	OFFSET	0 - 32 bit offset
4	METADATA	Metadata, meaningful only when MLEN > 0 Can be useful for extra request parameters 4 bytes will be used when MLEN=1, or MLEN=2
0 | 2	ECODE	16-bit error code, present only if LEN=0x7FF
0 - 1024	RDATA	Write data. Present only if RW=1, length determined by LEN (if LEN <= 1024)

CAN

The CAN compatibility is to be defined.
Ideally it should work trough CANopen SDO requests, but this way it is impossible to send properly the read length, 32-bit offset or metadata.