maxon motor ag
Brünigstrasse 220
CH – 6072 Sachseln
www.maxonmotor.com

**EPOS Command Library:
Data Exchange Rates**

Version: 1.0 (Eng.)
Author : WJ
Date : 2018-06-14

EPOS Command Library:
Expected data exchange rates?

Topic:

- Why is not possible to exchange data every 1 ms by USB (or other interfaces)?
- What data exchange cycle times can be achieved by using the EPOS Command Library?

Remark:

- The term “EPOS” refers to the EPOS, EPOS2, and EPOS4 product lines in the following.
- The term “EposCmd Library” is in use for the “EPOS Command Library” in the following.
  More information about this library and its set of functions is provided by the “EPOS Command Library.pdf” document.

Situation:

There is a Windows or Linux based master system (e.g. PC, Raspberry Pi, BeagleBone) in use and the master program uses the EPOS Command Library (-> EposCmd.dll) to exchange data with EPOS controllers by USB, RS232, or CAN. The execution of a typical EposCmd function call which configures a movement or retrieves some data has been evaluated as 10 – 20 ms which seems to be pretty long although the interface (e.g. USB) in use promises high data transfer rates and maxon states a 1 ms commanding cycle time for the EPOS.

What is the root cause that the expected (e.g. 1 ms) and actually measured time (e.g. 10 … 40 ms) varies so strongly and how can this be improved?

Solution:

Some technical background first:

- The EPOS can process CAN messages typically at a rate of 1 ms.
- A periodic data exchange at a rate of 1 ms (or less than 10 ms) has to be based on some basic requirements concerning the system environment too:
  - Precondition 1:
    A real time bus interface (like CAN or EtherCAT).
  - Precondition 2:
    Data exchange by so-called PDOs (= Process data object).
  - Precondition 3:
    A real time master which is capable to process and trigger the data exchange.

  Finally this means that a real time master must be able to process a so-called synchronized PDO data exchange by CAN or EtherCAT.

- Windows and Linux are no real time operating systems and cannot offer a predictable fixed processing and reaction time which both are preconditions for a fast periodic data exchange.
  - Non real time operating systems like Windows and Linux share the computer's processing power in between lots of different tasks. These are not scheduled on the base of defined time frames or priorities which can be fully configured by an application program or library. It is possible that another prioritized task of the operating system (e.g. hard disk access or graphics update) interrupts or blocks the data processing by the EposCmd library or application program for some milliseconds. The start of the EPOS data exchange or processing and forwarding the data by the application program can be delayed and exceed the pure reaction time of the EPOS strongly.
- PC-based programming environments (e.g. Microsoft Visual Studio, NI LabView) and the resulting application program, as well as the EposCmd library depend on the computing power and prioritization assigned by the operating system.
  - Application programs based on the EposCmd library cannot process data in a fast cyclic manner and there can be no fixed timing or response time specified because this is strongly influenced by the operating system and other processes active at the same time.
- The EposCmd library is designed to be in use by Windows and Linux only, i.e. not for time-critical high performance applications.
  - The EposCmd library does not support PDO based data exchange because this requires a defined timing and reaction time by most applications which is just possible by a real time operating system.
  - The EposCmd library uses a so-called SDO (= Service Data Object) data exchange. It is possible to read or configure any object by a SDO access, i.e. SDOs can be used for all-purpose.
    The SDOs use an acknowledged data exchange, i.e. the master sends information or a request to the EPOS and waits for a reply. A broken communication or mismatching data can be detected by a timeout or an error message by the EPOS. This results in a very reliable data exchange.
    If you compare SDO and PDO data access, the SDO can just access one object each time. If more objects have to be accessed, this has to be processed step by step and there is always the need to wait for the response of one object before the next can be processed. A SDO based data exchange requires more action and time to exchange multiple objects than by PDOs which can hold more than one object and do not acknowledge the data transfer.

What does that mean in practice?

The bottleneck of commanding and data exchange with an EPOS is neither the bus system nor the EPOS itself (which typically replies within 1 ms after receiving the request). The delay is mainly due to the operating system and the execution time of the application program and EposCmd library functions which have to initiate the data exchange, process, and forward the data.

If an application demands for an exhange of data at a rate less than 10 – 20 ms, there will be the need for a real time master (e.g. PLC, NI's cRIO, zub's MACS, …) and a bus communication like CAN or EtherCAT.

Function calls and reaction time by the usage of the EposCmd library:

The different application programming environments (e.g. C++, C#, Basic, LabView) supported by maxon's EposCmd library use a SDO based data exchange which sends a request to the controller and waits until the reply is received. Typically the SDO reply by the EPOS controller internally takes 1 ms up to 2 ms.

The application program and the EposCmd library creates the communication commands, decodes the reply and finally processes the information. The apparently present delays are caused by the operating system of the master and other programs which are active at the same time and all of these have to share the computing power.

Any single access of information including the processing of the command in advance and information afterwards by the master typically takes 2 – 10 ms. Actually this period of time can strongly vary due to the fact what other processes are active and demand computing power too and which might be prioritized by the operating system. In case of a non real time operating system this can just be influenced slightly and the reaction time cannot be specified by a fixed value.

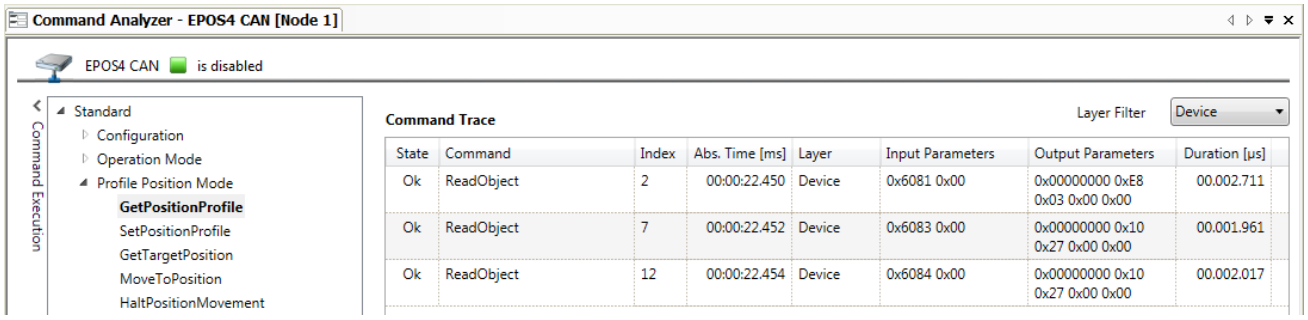
The more complex functions of the EposCmd library are partly based on data exchange of several objects which have to be read and written step by step. In case of "SetPositionProfile()" or "GetPositionProfile()" there is internally the information about the "Profile Velocity", "Profile Acceleration", and "Profile Deceleration" exchanged. Finally this results in a total processing time of 10 – 20 ms or even 40 ms although each single data request is processed within 1 ms by the EPOS internally. The mentioned 40 ms is a empirical value in case of a LabView system environment processing the "SetPositionProfile" VI.

Some hints for evaluation and optimization

Hint 1:
EPOS Studio's "Command Analyzer"

If you want to gather more insights about the timing and data exchange taking part by a EposCmd Library function call, EPOS Studio's "Command Analyzer" tool can provide some detailed information:

- Start EPOS Studio's "Command Analyzer" tool.
- Use the same interface (e.g. RS232, USB) like your application and master later on.
- Use the command tree to select the function call you are interested in,
  e.g. "Profile Position Mode / GetPositionProfile"
- Select the level of information which you are interested by the "Layer Filter" on the top right corner, e.g. …
  - "Device": Lists the EPOS object numbers, parameters, and the timing of each access.
  - "Protocol": Lists the a more detail data content and timing.
  - "Interface": Provides the most detailed information with the complete byte stream.



Hint 2:
Reducing number of data exchange

Just focusing on actually relevant objects can reduce the data exchange and make application programs process faster. Complex EposCmd function calls may read and write a number of objects which actually might not have to be updated based on the concrete application requirements.

The functions "SetObject()" and "GetObject()" can access a single object directly. It may be possible to optimize the required data exchange by this.

Example:
If just the parameter "Profile Velocity" has to updated for the next movement, it is more efficient just to access the corresponding single object by "SetObject()" than using the function "SetPositionProfile()" which updates "Profile Velocity", "Profile Acceleration", and "Profile Deceleration". The information about the naming and object numbers can be found in the "Firmware Specification" document of each EPOS product line.

Conclusion:

- EPOS controllers internally process commands and data requests typically within 1 – 2 ms.
- Windows and Linux based masters have no predictable timing or reaction time and do not provide a prioritized processing of special application or library tasks.
- Each data exchange and processing by a non real time master can typically take 2 – 10 ms.
- Function calls of the EposCmd Library can result in accessing several different objects.
- Processing time of a complex EposCmd Library function call can take 10 – 40 ms.
- The total processing time will be influenced by the master as well as other processes and software which are active at the same time.
- EPOS Studio's "Command Analyzer" provides detailed information about the number and content of the data exchange and the execution time of each step.
- The functions "SetObject()" and "GetObject()" can access information in a more focused way than by complex function calls. Finally this can reduce the number of accessed objects and reduce the total processing time of an application.
- If there is an application's demand for a data exchange rate less than 10 ms, this meant to use a real time master and CAN or EtherCAT.