Documentation EtherCAT Framework 1.2

This document describes the usage and installation of the “EtherCAT Framework“ version 1.2.

1 General Information

The EtherCAT Application Framework is an add-on library for the EtherCAT Acquisition Library. This library uses an object oriented approach with LabVIEW classes to represent the EtherCAT system. EtherCAT slaves are represented by slave objects, which are contained in an EtherCAT system object. This allows easier handling of data, because all information about slave address and variable information, etc. is held in the slave objects. Also the process data is divided into the slave data segments so that each slave object has access to its own process data.

The library also takes care of protection against race conditions. All critical slave data access is done inside a DVR structure. So data can be written to the output process data from any process without extra measures.

The application frameworks builds the basis for driver libraries for specific hardware like Beckhoff terminals. The use of inheritance and dynamic dispatching methods provides a very flexible and effective way to create own slave classes with specific functionality.

It is recommended to get familiar with the EtherCAT Acquisition Library before using the Application Framework.

2 System Requirements

2.1 Operating System

The library can be used on any system supported by the EtherCAT Acquisition Library.

2.2 LabVIEW

The library can be used with LabVIEW 2010 and following versions.

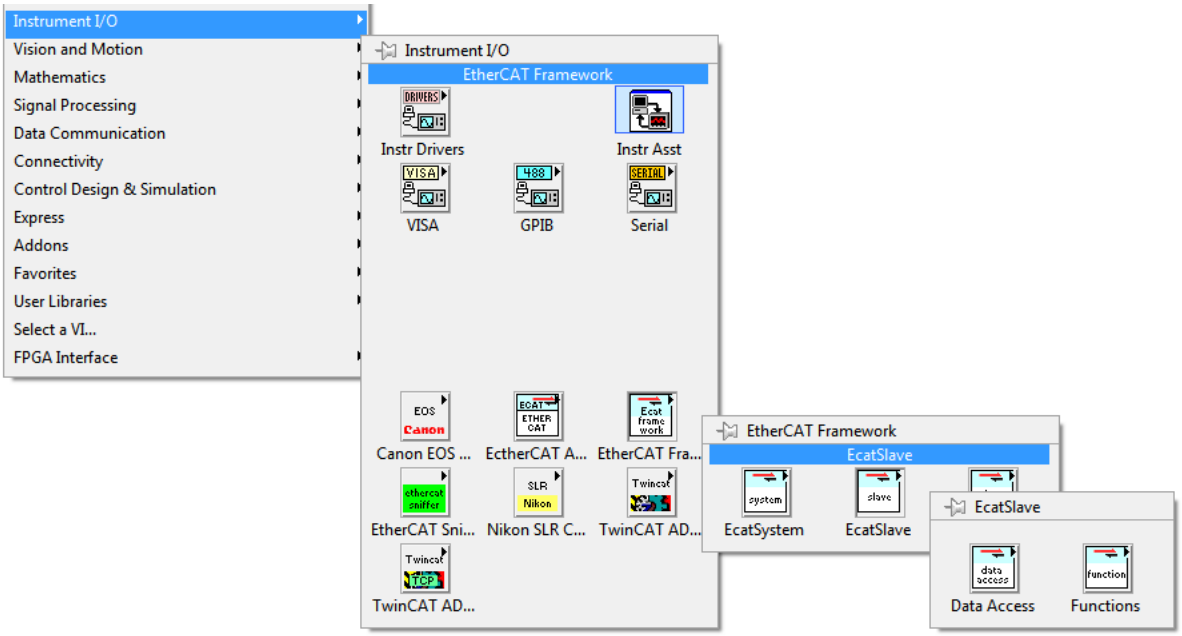
3 Installation

3.1 LabVIEW Library

The installer installs files directly to the LabVIEW folder (..\program files\National Instruments\LabVIEW xx\..)

The library is installed into the instr.lib folder in the subfolder "_Ackermann Automation\EtherCAT Framework".

The functions palette is installed in Instrument Drivers.



3.2 LabVIEW Examples

The examples are installed in the LabVIEW examples folder “..\Program Files\National Instruments\LabVIEW xx\examples\Ackermann Automation\EtherCAT Framework “.

3.3 Documentation

All further documents can be found in:

“..\Program Files\Ackermann Automation\EtherCAT Framework\Docs“

The VI documentation can be found in the help chm files of the VI libraries.

3.4 EtherCAT Acquisition Library

A working installation of the EtherCAT Acquisition library 2.6.3 and following is needed to work with the library.

4 Programming Benefits

4.1 Race Conditions

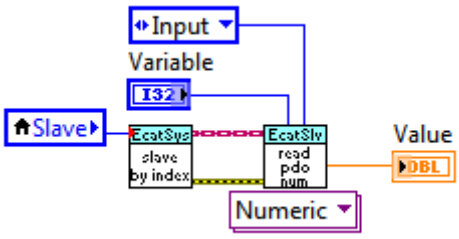
The library takes care of protection against race conditions for the slave data, which can occur when parallel processes access the same data resource. In the case of EtherCAT this is mainly the output process data array. All slave output data ends up in a byte array, that is sent on the bus. When for example one process takes care of digital outputs of a system and one process controls a analog waveform output, both processes need to write their data in the output process data array.
All critical slave data access is done inside a DVR structure, which is blocking access to the data while executed. So data can be written to the output process data from any process without extra measures.

4.2 Performance

The received process data coming from the EtherCAT Bus in buffered mode are 2D arrays for input and output data. The array rows hold the data of one cycle. With a bus cycle rate of 10 kHz this means each second 10000 rows of data are received. Each slave occupies a specific sub segment in the 2D array. To interpret the process data of a slave this data segment has to be read from the main 2D process data array.
Because LabVIEW works by Value, passing around big arrays of data in the code is not the best considering performance. Therefore the incoming process data is handled in a c dll to chop the main process data array in the smaller segments of the slave data. Each slave only works with its own smaller data arrays.

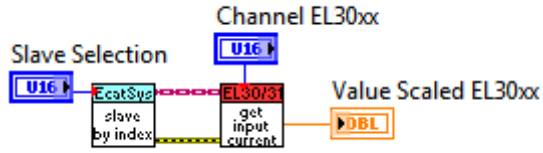
4.3 Data Indexing

In the process data each slave occupies a certain space defined by offset and length for their variables. If something in the slave topology or PDO configuration is changed, the offsets change. Therefore the offsets are handled inside the slave objects and the offsets are automatically adjusted.



4.4 Easy Data Access

All information about slave address and variable information is held in the slave objects. Using the slave objects in programming, the data access or slave function execution becomes much easier.
The data objects are updated to the recent data internally. So a slave object always contains fresh data.



4.5 Driver Libraries

The application frameworks builds the basis for driver libraries for specific hardware like Beckhoff terminals. The use of inheritance and dynamic dispatching methods provides a very flexible and effective way to create own slave classes with specific functionality.

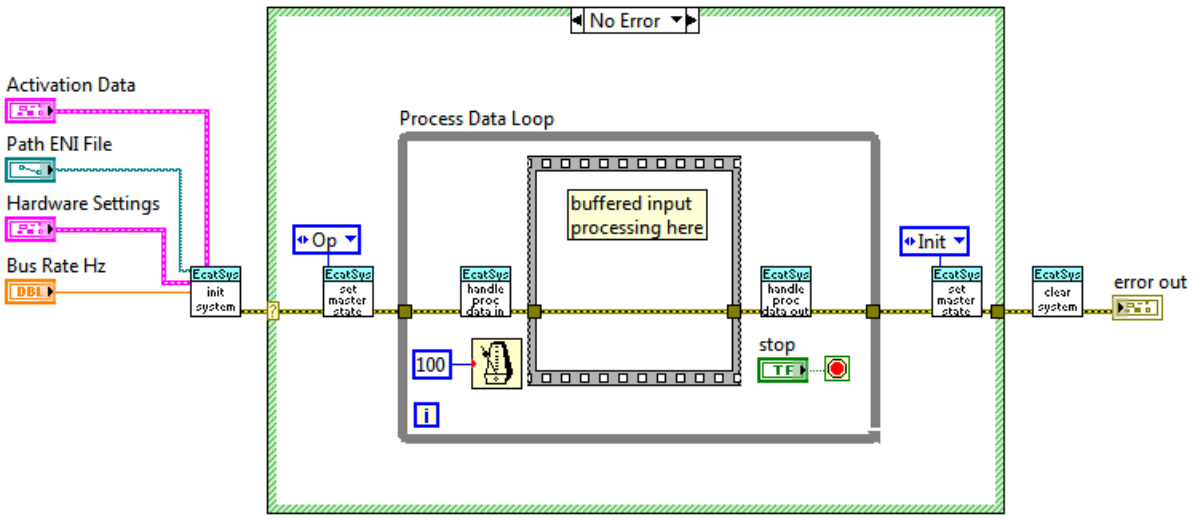
5 Basic Programming

This chapter describes the principals of programming with the library. The detailed VI information can be found in the help file.

5.1 EcatSystem

The EtherCAT System class contains the master configuration and holds a collection of EtherCAT slave objects. It is also used to control the bus.

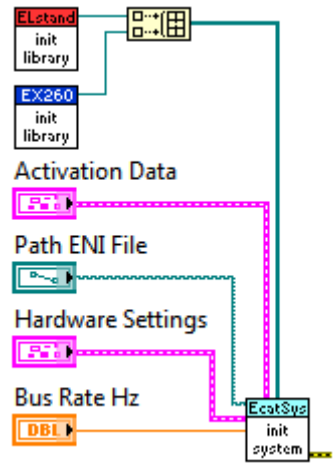
This class is designed as a singleton class with by reference access. This means there is only one EtherCAT system object. All methods use a by reference mechanism (DVR) to access the object data.



The basic structure for running the EtherCAT bus is:
1. Starting the master,
2. Setting the bus state into Operational
3. Doing process data handling in a loop
4. Setting the bus state to Init
5. closing the master

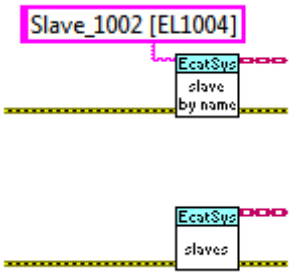
Starting the master

The InitializeSystem function contains all steps to start the EtherCAT master. It generates a slave object collection using the information from the ENI file. If slave driver libraries are used with the framework, these libraries have to be registered in the InitializeSystem function. This is done with the LibraryInit functions. Only then slave objects of the library type are generated.



Slave objects

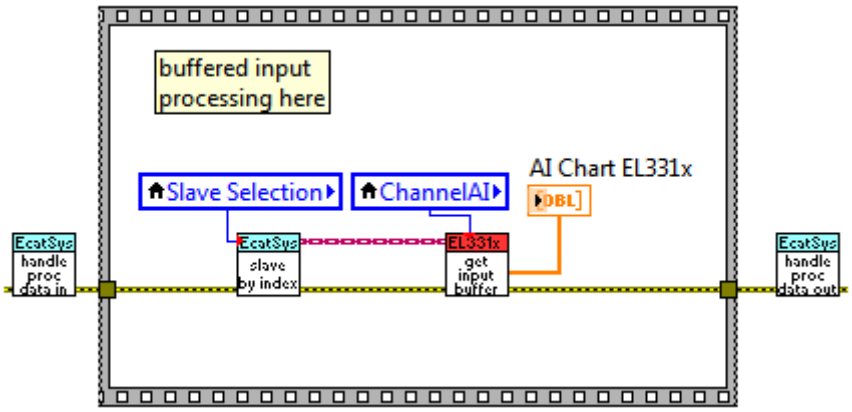
The slave objects are kept in an array within the EcatSystem object. They can be received as complete array or by using the slave name from the ENI file or the slave index.



Process data handling

The process data handling consists of handling input data and handling output data.

The function handleProcessDataIn collects the received process data and distributes it to the slave objects. This block of data is only valid until the next call of handleProcessDataIn. This means continuous buffered data cannot be acquired in parallel to the process data loop. It has to be placed between handleProcessDataIn and handleProcessDataOut function calls.

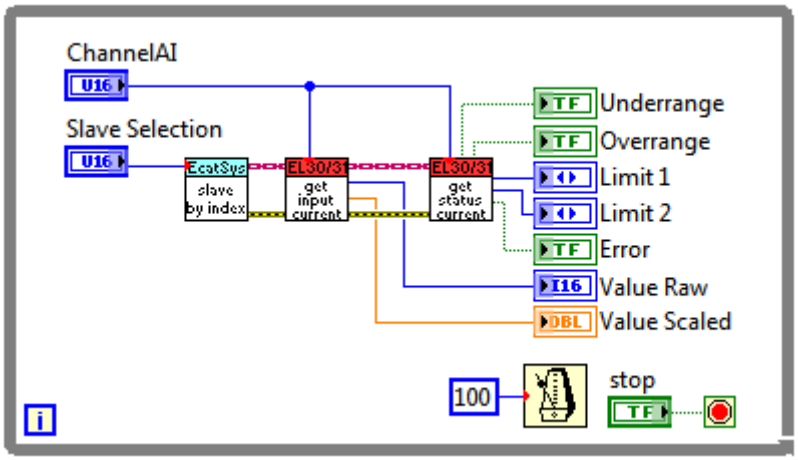


The function handleProcessDataOut writes the output data of the slave objects to the output process data and sends it to the bus.
Output data can be written to the slave objects in parallel to the process data loop. The data access is secured against race conditions.

5.2 EcatSlave

The EtherCAT Slave class contains the slave data. It contains the slave identification as well as the process data and scaling parameters.

The PDO functions internally update the slave object to the current data.



6 Support

For support contact:

Ackermann Automation GmbH
Kelsterbacher Strasse 15-19
60528 Frankfurt am Main
www.ackermann-automation.de

Tel.: +49(0)69-40562742
Fax: +49(0)69-40562816
info@ackermann-automation.de