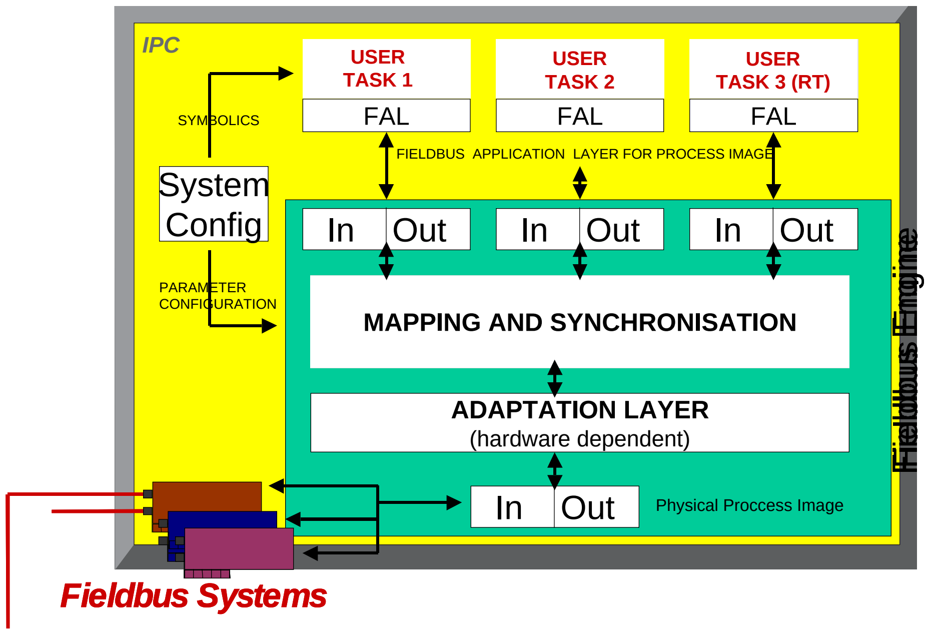
Generic Fieldbus Application Program Interface for Windows

Dipl.-Ing. Martin Rostan, Beckhoff Industrie Elektronik, Nürnberg
Dipl.-Ing. Gerd Hoppe, Beckhoff Automation LLC, Minneapolis

The choice of the appropriate fieldbus system for a specific application is not always driven by technical features, cost and availability considerations. System Integrators are often forced to use several fieldbus systems, as many customers demand to provide their favourite bus.

Microsoft Windows operating systems are accepted world wide, and thus provide a common base independent of the underlying fieldbus system. However, there is no common upper interface for the various bus solutions. Each bus requires specific tools and a change of the bus system leads to significant adaptations in the user and control programs. These efforts can be reduced by using a generic fieldbus application program interface for Windows.



The generic Fieldbus Application Program Interface provides an interface with the purpose of data exchange in real-time or non real-time between Windows applications and between those applications and any connected I/O device for Microsoft´s Windows operating systems.

The fieldbus API extends the Windows system to support cyclic and acyclic (packet based) communication to fieldbus devices. While the acyclic communication is very similar to standard network communication, the cyclic part differs: cyclic fieldbus communication is process image based. The applications work in a cyclic loop and read their input process image, calculate their output process image and write it back to the fieldbus.

If several applications share one fieldbus or one application uses more than one fieldbus at the same time or both, a mapping algorithm is needed. This mapping engine. The fieldbus engine provides a private process image for each application and for each fieldbus. This structure allows to develop simple applications and simple fieldbus drivers while using them in complex systems.

Applications who need to communicate to fieldbus devices in an acyclic way (upload/download of parameters etc.) use the ADS (Automation Device Specification) protocol. This protocol enables a wide range of communications and enables remote access from anywhere. At configuration time an ADS address is assigned to cation support. This address is used by the application to identify a specific device.

Features

The fieldbus engine handles two generic types of communications:

- cyclic data exchange
- acyclic data exchange

The cyclic data exchange for real-time access of IO data is usually triggered cyclic due to the nature of most common Fieldbus Systems, however, the data exchange may be triggered event-driven as well. Internal and external trigger sources may be applied to start the data exchange sequence.

The acyclic data exchange uses TCP/IP for networked connections and shared memory for local communications. Therefore it may be executed locally or remotely.

The management of the communication and data exchange relations as well as fieldbus network configurations, is handled through a GUI configuration tool, the System Manager, with Drag & Drop features for system setup and maintenance, and a central database holding related information for all connected networks. The system is extended by providing an OPC server as integral part to use OPC within its limitations as IO interface, but more to make the central tag database available to software applications to achieve cross-connectivity. The tag database architecture supports to use distributed databases.

Components

- Fieldbus Engine
- Application Interface
- ADS (Automation Device Specification)
- Configuration Tool

Fieldbus Engine

The fieldbus application program interface provides an abstraction layer to individual I/O networks and allows to establish communication to most different implementations by the use of abstraction and a modular driver model. It provides access in real-time to

- one application to one or several Fieldbus Systems at the same time
- one or several applications to one fieldbus at the same time
- or both of the above
- application to application real-time data exchange and messaging services

Real-time data exchange is guaranteed by a process image based communication system - the fieldbus cyclic services - with capabilities to map data between applications and IO systems, operated by the fieldbus engine. This real-time data exchange may be utilized between software components or from software component to IO subsystems, either networked, memory mapped, or otherwise accessible.

FAPI provides acyclic communication support, and a protocol superset (ADS) abstracts between the application interface and individual protocol implementations, guaranteeing the interoperability of application-to-device and application-to-application configurations.

Process Images and Fieldbus Cyclic Services

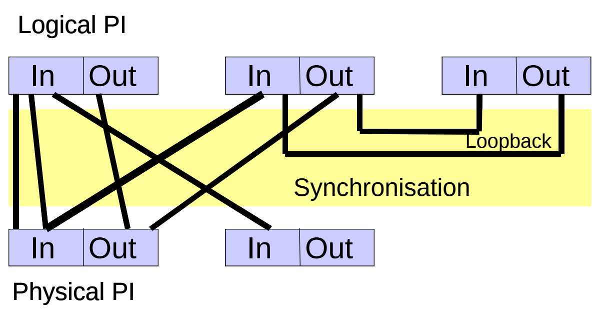
Process images keep a "snapshot" of signal status of a complete device or IO network in a buffer, usually separated for input and output signals. Real-time tasks operate either cyclic or event driven and - at start - read the status of the input signals, then execute logic statements (the program) and terminate after writing results for outputs into the output process image buffer. The size of process images is usually in the range of a few hundred bytes to a few kilobytes, adding a very small amount to the system footprint. Other than in streamed data applications, e.g. video, process images keep track of the status of few signals in real-time, overwriting obsolete old information within a quite small data buffer with no further need to store the history of this data.

To maintain consistent data exchange in a multi-application to multi-network environment, buffered process images are kept and data exchange by the fieldbus engine - the fieldbus cyclic services - is executed at the priorities given by the underlying real-time subsystem (RTSS). The fieldbus images, handles them consistently buffered, interacts in real-time and deterministically.

To abstract the data entity handled between two related process images (task to task or task to IO device) completely from implementations, the Fieldbus engine copies data entities from a single bit to complex structures. Data annotation conversions, e.g. Intel - Motorola are supported internally, giving convenience to users.

Process Images and Mappings

The fieldbus engine is the central mapping machine executing all cyclic communication. Each participant of cyclic communication (user applications, fieldbus cards) is represented in the fieldbus engine with a process image. These participants have access to the fieldbus engine through their assigned process image. The duty of the fieldbus engine is the mapping of data between these process image in the correct timing, consistency and priority.



The fieldbus engine holds a process image object for each participant of cyclic communication. It also holds mapping objects, instantiated for each relation between process images. If one participant exchanges data (inputs or outputs) with another participant; one mapping object represents the relationship between their two process images. A process image object may have several related mapping objects, each to generate a data exchange to another process image.

Both objects (process image and mapping objects) are allocated in advance to meet real-time performance requirements.

Process Image Types

There are two different kinds of process image objects: Master process images and slave process images.

Master process images are used for participants, which act on their own thread, e.g. user applications like a PLC running in a cycle loop and exchanging their inputs and outputs at the beginning and the end of their cycle.

Slave process images are used for participants, which are triggered by other threads (represented by a master process image), e.g. fieldbus card drivers. Slave process images are triggered synchronously in the context of a master thread. This enables for example a PLC to start a fieldbus cycle synchronously with it's own cycle.

Mapping Types

There are two different kinds of mapping objects: Asynchronous mappings and synchronous mappings:

Asynchronous mappings are used between two master process images. Master process images are acting on their own thread and cannot be synchronized. To guarantee consistent data exchange without blocking of one thread, the asynchronous mapping uses a three buffer mechanism.

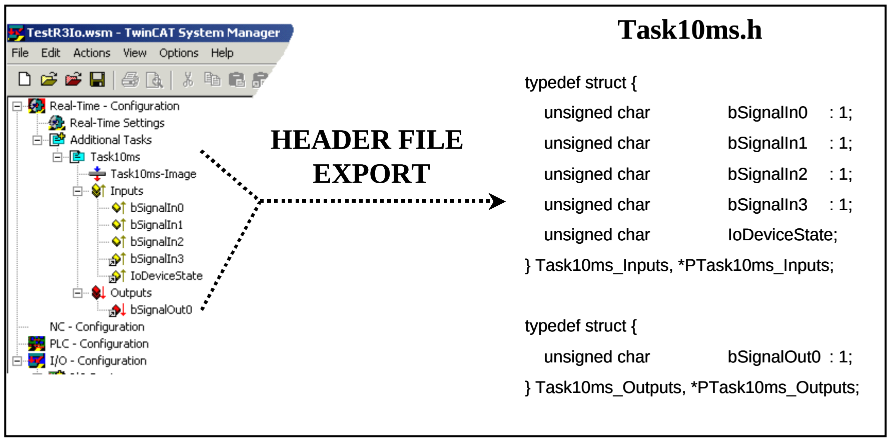
Synchronous mappings are used between a master process image and a slave process image. The mapping does not use any buffers because the master process image can access the slave process image synchronously. In this context the slave cycle can be started by the slave application or by the corresponding fieldbus card driver.

Real-time Issues

Real-time access to I/O devices should be carried out with no further delay through protocol layers or context switches. Usually, a real-time subsystem is driving a set of tasks with real-time handling purpose at certain cycle or response-to-event times. The FCS process images are objects to those tasks, administered in a sophisticated way to avoid context switches to offer direct access to I/O information within algorithm from the local task images into the physical card process image buffers. As process images have usually small data sizes of some kilobytes, the execution time of the fieldbus engine adds only a few percent to the task runtime, performing a duty otherwise executed by the real-time task itself directly. The fieldbus engine works deterministically for a given configuration, the described concept has proven applicability down to cycle times of 50 µs in critical industrial applications on Pentium II class computers.

Interface to User Applications

User applications that want to participate in the cyclic data exchange need an interface to the fieldbus engine. The fieldbus engine implements some function calls through that a user application can access its process image object in the engine. In these calls, the fieldbus engine copies the inputs and outputs to their destination process images (through the related mapping objects) and starts related update cycles, if existing.



```
typedef struct {
    unsigned char          bSignalIn0    : 1;
    unsigned char          bSignalIn1    : 1;
    unsigned char          bSignalIn2    : 1;
    unsigned char          bSignalIn3    : 1;
    unsigned char          IoDeviceState;
} Task10ms_Inputs, *PTask10ms_Inputs;

typedef struct {
    unsigned char          bSignalOut0  : 1;
} Task10ms_Outputs, *PTask10ms_Outputs;
```

Application Interface

The approach provides a software application to software application interface, providing a set of control and data items as an interface superset to existing implementations. Together with the cyclic services it offers an abstracted, deterministic software component interface for synchronous and asynchronous communication. Software components may work local or distributed over network without any change, interact in real-time or non real-

Depending on their physical location (locally in real-time; networked at the speed of the utilized network) software components may directly access FCS in real-time or cross-connect to FCS over ADS and have remote - non real-time - data exchange. A basic set of control and data access methods is included as a superset to define software to software applications.

If an ADS application requests data from another ADS application on the same computer, then the ADS router transfers the data directly through shared memory. Each ADS port has it's own message queue in a memory mapped object, so that ADS can efficiently copy the data from one message queue to the other.

Automation Device Specification - ADS

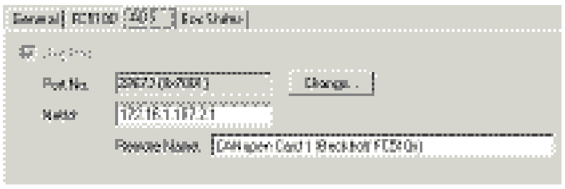
Other than real-time communication to Input / Output devices; many existing devices support parameterization or up/download of programs, web pages or other information. ADS handles this streamed data through asynchronous communication support.

ADS is similar to TCP/IP, and addresses all devices through IP-type addresses. For future application-to-device and application-to-application communication, raw ADS protocol builds an efficient standard.

ADS allows to asynchronously access process image information of the FCS system through asynchronous slave process images: this opens efficiently remotely networked access at the transmission capabilities of the underlying network. ADS provides timestamps for messages for synchronization.

ADS Communication Principles

ADS Communication is basically an asynchronous message transfer between a client application and a server application. The communication is connectionless; the communication endpoints are identified through an IP-type address and an ADS


General | FC5100 | ADS | Box States

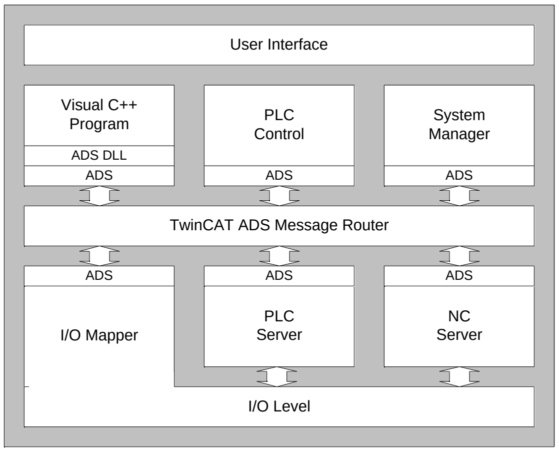
Port No.: [illegible] Change...

NetId: 172.16.1.187.2.1

Remote Name: CANopen Card 1 (Beckhoff FC510x)

All communications are handled through a simple routing mechanism. The ADS API provides methods, which are used for acyclic data transfer and server configuration. Most of the methods need a response through the server for a valid communication sequence, however, an unconfirmed notification message to the client application is supported for servers as well. Within ADS, two possible communication paths are available:

- Communication between two applications on a local system. In this case all requests are marshaled through the ADS router with use of shared memory objects fast and efficiently to the demands of real-time applications.
- Communication between remote partner applications. In this case, a TCP/IP type connection to the remote system has to be established by the local ADS router and the ADS packet will be send to the remote ADS router. On the remote system the server application is called on the same way as in the local case. In a typical communication sequence, the client initiates a ADS request and the server calls the response asynchronously.



Communication methods between ADS Clients and Servers are

- synchronous read / write
- asynchronous read / write
- notification on change (with minimum cycle)

For optimization purposes, a ADS client has the ability to register a notification at the server, those notifications can be configured to be sent by the server in case of data change (Notify on change) or in a cyclic way. This optimization can be used to save bandwidth in a network environment.

For simple data access, an application can call the read, write or read/write methods. The data is addressed through a pair of indices; “Index Group” and “Index Offset”. These indices are server/device dependent, standard/generic device profiles for devices like drives and I/O-modules may be defined, e.g. by OPC, OMAC, PLC-open, or other industry-wide operating interest groups. The data is transferred in a raw byte stream in Little Endian format.

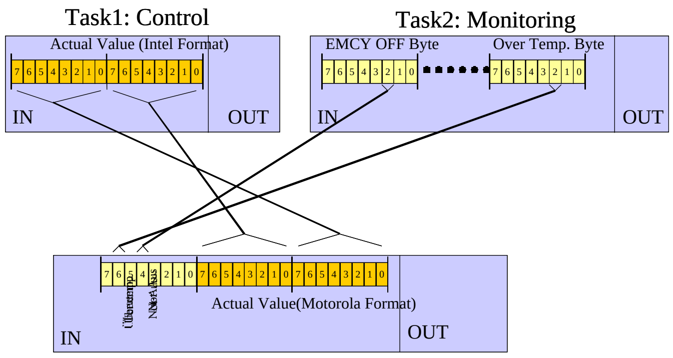
Configuration

Configuration is important before operating Fieldbus Systems: these have no self-configuring features, some even do not allow configuration while operating. Fieldbus interfaces and devices need quite complex configuration data: these include general, fieldbus dependent and device specific information. Also, the flexibility of the fieldbus engine requires application specific configuration data.

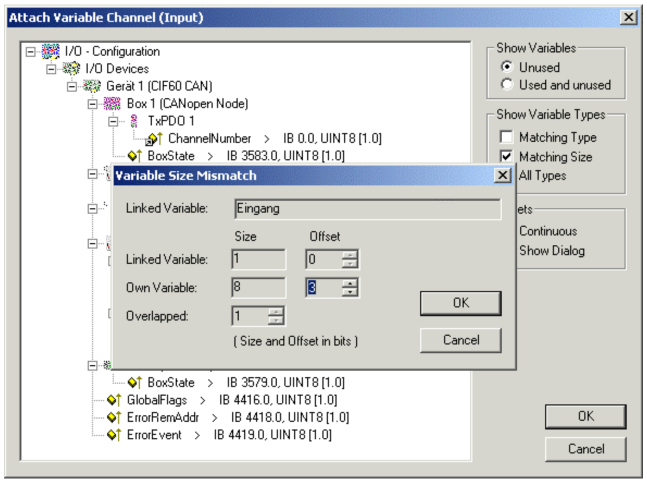
Two major configuration issues may be addressed: The first issue is the configuration of the fieldbus engine and the related process image and mapping objects. The second issue is the configuration of the fieldbus interfaces and fieldbus devices. While the first part is a private configuration of the fieldbus API, the second part is very device- and vendor specific.

Configuration of the Fieldbus Engine

The configuration of the fieldbus engine is variable based: all relations between the participants of the cyclic data exchange are expressed via variables, not by origin inside a memory. Each participant defines its process image through variables. As an example, an application like a PLC defines virtual input and output variables. These variables are used in the internal program puts as variables in their logical process images. These variables may be linked together, assigning a virtual variable of an application to a physical variable of a fieldbus.



The relationship between variables of two different process images (done at configuration time) defines a mapping object, describing all relations between variables of the process images. A process image shares exact one mapping object with every other process image in which linked


Attach Variable Channel (Input)

I/O - Configuration
I/O Devices
Gerät 1 (CIF60 CAN)
Box 1 (CANopen Node)
TxPDO 1
ChannelNumber > IB 0.0, UINT8 [1.0]
BoxState > IB 3583.0, UINT8 [1.0]
BoxState > IB 3579.0, UINT8 [1.0]
GlobalFlags > IB 4416.0, UINT8 [1.0]
ErrorRemAddr > IB 4418.0, UINT8 [1.0]
ErrorEvent > IB 4419.0, UINT8 [1.0]

Show Variables: [✓ Unused] Used and unused
Show Variable Types: Matching Type [✓ Matching Size] All Types
Continuous
Show Dialog

Variable Size Mismatch

Linked Variable: Eingang

	Size	Offset
Linked Variable:	1	0
Own Variable:	8	3
Overlapped:	1	

(Size and Offset in bits)

variables exists. As on a local machine, mapping objects are located in shared memory, to use an efficient data exchange method for time critical applications.

Normally the size and data type of two linked variables has to be unique; however, the configuration tool allows the linking of differently sized variables. If the size of these two variables differs, the user has to specify the necessary shift operations to match the divider of both sizes.

Variable Types

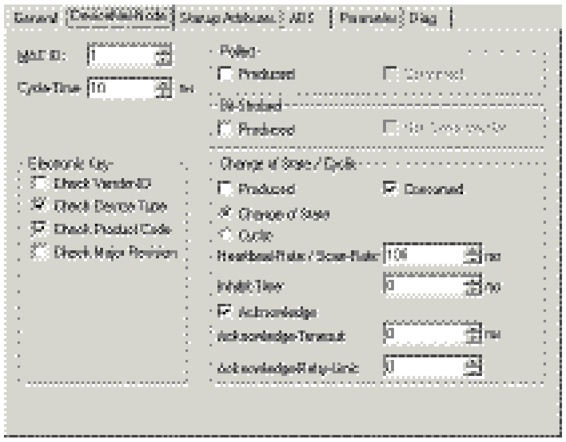
The fieldbus API supports basic variable types such as signed and unsigned 8, 16, 32, and 64 bit integers, 32 and 64 bit floating point values. Arrays and user defined data types (structures) of the basic types are supported, but subject to certain limits. A single bit value is also supported due to the special needs in fieldbus environments.

Configuration of Fieldbus Interfaces and Devices

Fieldbus interface cards and fieldbus devices need more configuration parameters than network interface card in general. A fieldbus master device to configure its own interface cards and the external fieldbus devices, which are connected via the fieldbus. The complexity of this configuration depends on the capabilities and features of the external devices.

The major configuration parameters of fieldbus interfaces relate to timing and memory size and offset issues. Also, many interfaces provide notification services for special events or configurations for special modes of fieldbus operation. The configuration functions are accessed through the configuration tool and carried out by the individual driver.

Fieldbus devices connect IO signals to their internal process image, with variation in size, refresh rate, supported services, access method, etc. The configuration of a fieldbus card holds a garbage collection of all connected devices and their features. To support multi-vendor applications, some fieldbus standards have established electronic datasheets to describe devices; however, these are fieldbus - proprietary and do not abstract from a certain fieldbus implementation. Examples are DeviceNet and CANopen EDS files or Profibus GSD. The system manager tool identifies Profibus and DeviceNet devices by their GSD or EDS files and identifies component features.



MAC ID: 1
Cycle-Time: 10 ms

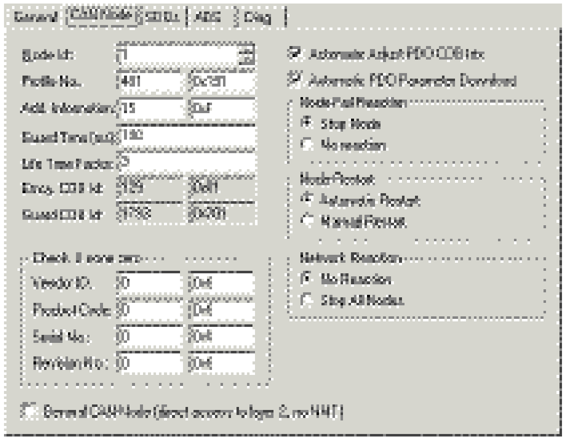
Polled
☐ Produced ☐ Consumed

Bit-Strobed
☐ Produced ☐ [illegible]

Electronic Key
☐ Check Vendor ID
☑ Check Device Type
☑ Check Product Code
☐ Check Major Revision

Change of State / Cyclic
☐ Produced ☑ Consumed
● Change of State
○ Cyclic
Heartbeat-Rate / Scan-Rate: 100 ms
Inhibit Time: 0 ms
☑ Acknowledge
Acknowledge-Timeout: 0 ms
Acknowledge-Retry-Limit: 0

The CANopen Node and PDO configuration of the Beckhoff FC510x CANopen cards can be done manually – this allows to connect devices that do not have Electronic Data Sheets (eds) Files available. Alternatively eds based configuration is possible as soon as CiA-WD306 is in place.



Node Id: 1
Profile No.: 401 0x191
Add. Information: 15 0xF
Guard Time (ms): 100
Life Time Factor: 3
Emcy. COB Id: 129 0x81
Guard COB Id: 1793 0x701

Check if none zero
Vendor ID: 0 0x0
Product Code: 0 0x0
Serial No.: 0 0x0
Revision No.: 0 0x0

☑ Automatic Adjust PDO COB Ids
☑ Automatic PDO Parameter Download

Node Fail Reaction
● Stop Node
○ No reaction

Node Restart
● Automatic Restart
○ Manual Restart

Network Reaction
● No Reaction
○ Stop All Nodes

☐ General CAN Node (direct access to layer 2, no NMT)

The generic fieldbus application program interface concept has proven its benefits: It is implemented within the TwinCAT automation software package and is used in several thousand applications worldwide.

Beckhoff Industrie Elektronik
Eiserstr. 5
D-33415 Verl
Germany
Phone +49 5246 963-0
Fax +49 5246 963-149
Web : www.beckhoff.com