



... the far end of Ethernet

- Direct access to remote I/Os over a Windows®-DLL
- Use of the Express-I/O functions on ethernode®:
 - IOInfo
 - IOCfg
 - IOIn
 - IOOut
 - IOVecIn
 - IOVecOut
- One order can address several ports at the same time

A website is nice if you want to look into an engine or an unit, but for remote control you need functions which can be installed into your program surroundings. *nodeAccess*™ offers this possibility with a DLL which makes in- and outputports on a far-off ethernode® or a TSM-CPU directly manipulable. Therewith you save local programming on remote controlled units.

Express-I/O makes it easy for the programmer to access process I/Os like 24V plugs or temperature sensors without the need for think over portaddresses and bitmanipulation. Already for many years it has been possible to access Express-I/O ports directly over the BITBUS fieldbus even from a master. Also from afar you only have to name at which type of bus the in-/output is located (e.g. at the processor pins, at the I²C-bus or at the TSM-bus), on which module and at which clamp.

nodeAccess™ uses the infrastructure that is built up by BITBUS and calls the BITBUS "BAPI" which is called "BAPITCP" in the TCP/IP version. So you install the included BAPITCP first and connects with BapiTcpCfg a ethernode or a TSM-ARMCPU with the desired IP-address with a BITBUS unit name. Up to 16 units are possible which are usually consecutively named with BBUS0, BBUS1 and so on. For reasons of compatibility you have to rename the in \windows\system32 installed DLL "bapitcp.dll" as "bapint.dll".

Even if the access goes now over Ethernet, we have hold the appellation because the simple master-slave schema is easily comprehensible. For *nodeAccess*™ addressed ethernode is a BITBUS master (which can actually be true, see [ETH-BIT](#))

The access with *nodeAccess*™ can be explained with an example where a digital input at clamp 1 is requested:

```
#include <windows.h>
#include <stdio.h>
#include "nodeacc.h"
int main( int argc, char *argv[], char *envp[])
{
    XPIO_IO_Item in;
    XPIO_IO_Item *in_p;

    const char *BitbusDevice = "BBUS0";
    unsigned BitbusNode = 255;
```

```
if (vio)
{
    vio->hdl = xp_io_h;
    vio->num_values = num;
    XPIOVecIn(vio);
    ldata = vio->value;
    chan = 0;
    while (num-->0) {
        printf("CH[%d] = %ld\n",chan++,*ldata++);
    }
    free(vio);
}
```

Order codes:

nodeAccess™ is available for free on all modules that support Express-I/O and a network.

* all prices in EUR ex works (+VAT/MwSt inside Germany)

```

unsigned bus = BUS_TYPE_CPU;
unsigned module = CPU_DIN;
unsigned chan = 0;

/*
 * Get a handle for digital input.
 */
in_p = &in;
in_p->hdl = XPIOGetHandle(BitbusDevice, BitbusNode,
bus, module, chan);
in_p->value = (-1);
if (in_p->hdl) {
    /*
     * Send In to get and display the values.
     */
    XPIOIn(1, &in_p);
    printf("IN %d = %d\n", chan, in_p->value);
}
return 0;
}

```

At first a Bitbusdevice is named "BBUS0".
 With BitbusNode = 255 the "Master" is addressed (With "3" you would address a Bitbus slave with the *****knotennummer**** 3 which is connected to the ethernode or the TSM-ARMCPU).
 bus = BUS_TYPE_CPU means that the I/Os are addressed which are connected to the CPU itself, not to an enlargement module (e.g. an analog module at the TSM-ARMCPU).
 module = CPU_DIN is called the module inside the bus (at an I/O enlargement module the address according to the ****Drehschalter**** would be declared here, e.g. module = 7).
 chan = 0 Finally the clamp 1 chooses ****=wählt aus?***** (which is traditionally addressed as "0" in the programming language).

After that the actual program fetches a reference to the input-port with XPIOGetHandle and the parameters above. If this works out XPIOIn is called and returns with the input's state in in_p->value. Instead of in_p you can use also a variable-name which enables a associativity to the application, so e.g. motor3turns.

Now the access runs simply over the function calls XPIOIn and XPIOOut. For reasons of efficiency several ports can be packed into one order, so only one TCP/IP message is en-route in the Ethernet.

In their development surroundings there is to be declared the *nodeAccess*[™]-DLL. This is done differently depending on language and tools, but it follows the instructions for the integration of external libraries. In addition to the example above and the belonging visual C++-projections there is also an example on the CD for a command line program wherewith you can access I/Os on ethernode directly. In its compiled form it is useful as a setup help for the test of new installations' I/Os.

nodeAccess Command Line Tool
