United Electronic Industries

The High-Performance Alternative

UEIDAQ EPICS Support Module

—

User Manual

Experimental Physics & Industrial Control System (EPICS) node client/server software for UEIDAQ PowerDNA Cube and PowerDNR RACKtangle with analog input and output, digital input and output and counter/timer support

Release 4.5

April 2012

PN Man-DNx-EPICS-0412

Contacting United Electronic Industries

Mailing Address:

27 Renmar Avenue
Walpole, MA 02081
U.S.A.

For a list of our distributors and partners in the US and around the world, please see http://www.ueidaq.com/partners/

Support:

Telephone: (508) 921-4600
Fax: (508) 668-2350

Also see the FAQs and online “Live Help” feature on our web site.

Internet Support:

Support: support@ueidaq.com
Web-Site: www.ueidaq.com
FTP Site: ftp://ftp.ueidaq.com

Product Disclaimer:

WARNING!

DO NOT USE PRODUCTS SOLD BY UNITED ELECTRONIC INDUSTRIES, INC. AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS.

Products sold by United Electronic Industries, Inc. are not authorized for use as critical components in life support devices or systems. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness. Any attempt to purchase any United Electronic Industries, Inc. product for that purpose is null and void and United Electronic Industries Inc. accepts no liability whatsoever in contract, tort, or otherwise whether or not resulting from our or our employees' negligence or failure to detect an improper purchase.

Specifications in this document are subject to change without notice. Check with UEI for current status.

UEIDAQ EPICS Support Module

Contents

1. Introduction

1.1. EPICS

The Experimental Physics and Industrial Control System (EPICS) is a software environment used to develop and implement distributed control systems to operate devices such as particle accelerators, telescopes and other large experiments. EPICS also provides SCADA capabilities. The tool is designed to help develop systems which often feature large numbers of networked computers providing control and feedback.

EPICS uses client/server and publish/subscribe techniques to communicate between the various computers. One set of computers (the servers or input/output controllers), collect experiment and control data in real-time using the measurement instruments attached to it. This information is given to another set of computers (the clients) using the Channel Access (CA) network protocol. CA is a high bandwidth networking protocol, which is well suited to soft real-time applications such as scientific experiments.

1.1.1. IOC

EPICS interfaces to the real world with IOCs (Input Output Controllers). These are either stock-standard PCs or VME standard embedded system processors that manage a variety of "plug and play" modules (GPIB, RS-232, IP Carrier etc.) which interface to control system instruments (oscilloscopes, network analyzers) and devices (motors, thermocouples, switches, etc.). Some instruments also can come with EPICS already embedded within them, like certain Oscilloscopes.

The IOC holds and runs a database of 'records' which represent either devices or aspects of the devices to be controlled. IOC software used for hard-real-time normally use RTEMS or VxWorks. Soft real-time IOC software runs on Linux or MS-Windows based machines.

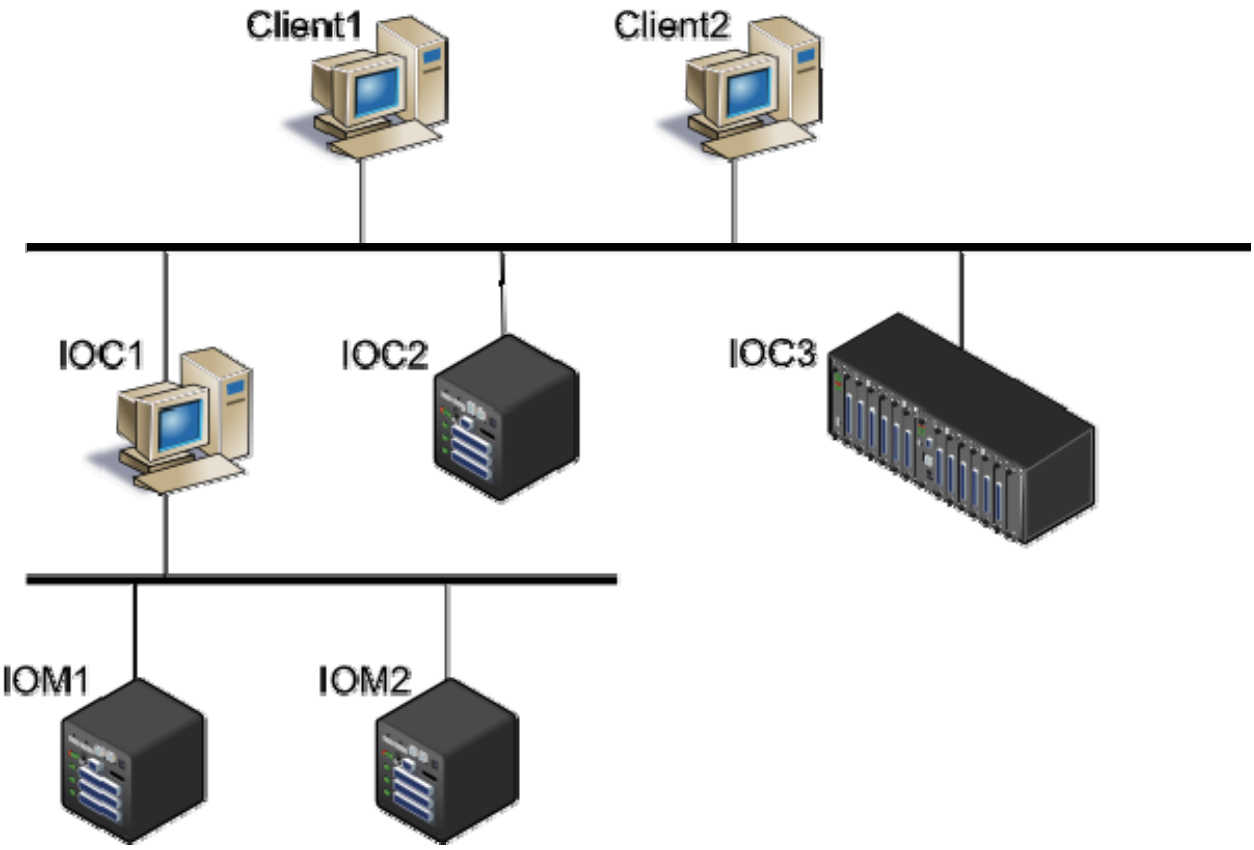
1.1.2. Clients

Other computers on the network can interact with the IOC via the concept of channels. Channel names are typically in the form EQUIPMENT:SIGNALNAME (e.g. ACCELERATOR_RING:TEMP_PROBE_4, although they can be much less verbose to save time).

Any software which can speak the CA protocol can get and put values of records. For example on the EPICS website there are several extension packages which allow CA support in things like MATLAB, LabVIEW, Perl, Python, Tcl, ActiveX, etc.

1.2. UEIDAQ support module

The UEIDAQ support module allows an IOC application to communicate with PowerDNA I/O cubes or PowerDNR I/O racks.



The IO modules (IOM) are available in cube or rack form-factor. Each IOM can be configured to run as a slave controlled by a host or as a standalone PAC (programmable automation controller).

- Slave IOM: the IOC application runs on the Linux/Windows host (PC, VME processor, etc...) and controls slave IOMs via ethernet.
- UEIPAC: the IOC application runs standalone under Linux on the IOM.

Each IOM contains up to 6 (on cubes) or 12 (on racks) I/O layers.

The UEIDAQ support module can only control AI,AO, DIO or CT layers, however you can also use Serial (RS-232/485) layers through the serial server like any other serial port.

Each channel on each AI, AO, DIO or CT I/O layers is associated with an EPICS record on the IOC. Those records can be read from or written to using Process variables.

The UEIDAQ support module comes with a test IOC application, you can also add the UEIDAQ support module as a library to an existing IOC application.

1.3. Theory of operation

The UEIDAQ support module configures the I/O layers in DMAP mode.

DMAP stands for Data Mapping, a snapshot (aka DMAP) of the input and output data is kept on both the IOM and the host.

In this mode, timing is controlled by the host which periodically synchronizes its DMAP with the DMAP on the IOM.

When requested by the host, data from/to all layers in the IOM is input/output as a snapshot and all such data is transmitted to the host (or other device) in a single packet. In other words, each snapshot of all layers is transmitted in a single network round trip. (Note that if the data does not fit in a single packet, multiple packets will be used.)

This mode is ideal for stimuli/response control and other real-time applications. DMap is the most efficient mode for point by point input/output on AI, AO, DIO, and CT layers.

The UEIDAQ support module runs a periodic task to refresh the DMAP at a rate specified in the IOC startup script. The maximum DMAP rate depends on the number of channels configured. A fully loaded slave rack (12 I/O layers with 24 channels each = 288 channels) can run up to 2kHz, a fully loaded UEIPAC rack can run up to 5kHz.

You can specify whether you want the EPICS records associated with each channel to be processed synchronously with the DMAP or at a slower rate.

2. Installation and Build

We assume that you already have EPICS base installed on your Linux or Windows PC

Place the UEIDAQ Support distribution file into this directory.

Execute the following commands:

```
cd support
tar xvfz ueidaq-epics-support-<release>.tgz
cd ueidaq-<release>
```

Where `<release>` is the release number of the UEIDAQ support.

Edit the **configure/RELEASE** file and set the paths to your installation of EPICS base.

Execute make in the top level directory.

3. Configuration

3.1. Startup script

The IP address of the IOM(s), the location of each I/O layer and the configuration for each channel is specified with a driver API. Those API functions are called in the IOC startup script:

3.1.1. void drvUeiDaqInit()

Initializes the driver internal data structures. This API function must be called first.

3.1.2. int drvUeiDaqAddChannels(const char* ipAddress, const char* channelOptions)

- **ipAddress**: IP address of the IOM where the channels are located
- **channelOptions**: contains the list of channels, the device where they are located and common configurations. format: D=<device id> CL=<channel list> <options>
    - **D**: device id, on cubes top-most device's id is 0, on racks leftmost device's id is 0
    - **CL**: channel type (AI/AO/DI/DO/CI/CO) followed by comma separated channel list. ex: "AI0,1,2,3", "DO1"
      DI and DO channels represent multiple bits. For example the DIO-403 comes with two channels of 24 bits each.
    - **options**: configuration option common to all channels specified by CL
        - AI options:
            - inputmode: DIFF (default) or RSE
            - gain: index of selected gain in available gain list, 0 (default) is first gain (usually gain of 1x)
            - measurementtype: V for volt, TC for thermocouple
        - TC options:
            - tctype: E/J/K/S/R/T/B/N
            - tempscale: C for celsius, F for fahrenheit or K for Kelvin
            - cjctype: "builtin" or "constant"
            - cjcconst: constant value for CJC temperature
        - CI options:
            - measurement mode: COUNT/QUAD/PERIOD/PULSEWIDTH
            - source: INTERNAL/EXTERNAL
            - gate: INTERNAL/EXTERNAL
            - inputinverted: 0/1
        - CO options:
            - output mode: PULSE/TRAIN
            - source: INTERNAL/EXTERNAL
            - period: duration of each pulse cycle in us
            - dutycycle: dutycycle of PWM valid range is [0.0 1.0]

Call this API function as many times as required to configure channels located on the same I/O layer with common configuration.

3.1.3. void drvUeiDaqVerbosity(int level)

Sets the verbosity level which controls the number of messages printed by drvUeiDaq on the IOC shell. It goes from 0 for no messages at all to 10 for the maximum number of messages. 7 is a good middle ground value.

3.1.4. void drvUeiDaqRefreshRate(double rate)

Sets the rate at which the IOC refreshes the inputs and outputs.

3.1.5. void drvUeiDaqHelp()

Prints drvUeiDaq** APIs help

3.1.6. void drvUeiDaqDump()

Dumps the latest acquired values on all input channels and the latest values written to output channels

3.1.7. void drvUeiDaqReport(int level)

Prints a report of various internal variables as well as a list of the channels with detailed configuration. This command is useful to detect syntax errors in the channel option string.

3.1.8. Example

Edit the start-up script **iocBoot/iocueidaq/st.cmd** to configure your channels.

The example below configures 8 channels on AI layer 1 of IOM "192.168.100.2". The first four channels with a gain 0 and the next four channels with gain 1 It also configures four AO channels on AO layer 0.

```
#!../../bin/linux-x86/ueidaq
envPaths

cd ${TOP}

dbLoadDatabase("dbd/ueidaq.dbd")
ueidaq_registerRecordDeviceDriver(pdbbase)

## Load record instances
dbLoadTemplate("db/iom.substitutions")
drvUeiDaqInit()
```

```
drvUeiDaqAddChannels("192.168.100.2", "device=1 cl=AI0,1,2,3
inputmode=diff gain=0 measurementtype=V")
drvUeiDaqAddChannels("192.168.100.2", "device=1 cl=AI4,5,6,7
inputmode=diff gain=1 measurementtype=V")
drvUeiDaqAddChannels("192.168.100.2", "device=0
cl=AO0,1,2,3")
drvUeiDaqVerbosity(7)
drvUeiDaqRefreshRate(100.0)

cd ${TOP}/iocBoot/${IOC}
iocInit
```

3.2. Record database

To inform EPICS of this new driver/device, a DBD file is used.

ueidaqSupport.dbd looks like this:

```
device(ai, INST_IO, devAiUeiDaq, "UeiDaq")
device(bi, INST_IO, devBiUeiDaq, "UeiDaq")
device(mbbi, INST_IO, devMbbiUeiDaq, "UeiDaq")
device(mbbiDirect, INST_IO, devMbbiDirectUeiDaq, "UeiDaq")
device(ao, INST_IO, devAoUeiDaq, "UeiDaq")
device(bo, INST_IO, devBoUeiDaq, "UeiDaq")
device(mbbo, INST_IO, devMbboUeiDaq, "UeiDaq")
device(mbboDirect, INST_IO, devMbboDirectUeiDaq, "UeiDaq")
driver(drvUeiDaq)
registrar(drvUeiDaqRegister)
```

ueidaq.dbd merges **base.dbd** and **ueidaqSupport.dbd** and is automatically created at build time.

You can load **ueidaq.dbd** directly via dbLoadDatabase() in the startup script or you can add the following lines in your application DBD file:

```
include "base.dbd"
include "ueidaqSupport.dbd"
```

3.2.1. EPICS Record Generic fields

3.2.1.1. DTYP: Device Type field

Must be "UeiDaq" as defined in DBD file

```
field(DTYP, "UeiDaq")
```

3.2.1.2. SCAN: Scan field

- **I/O Intr**: The driver causes the record to be processed as soon as a new value is received
- **Passive**: Output records are often passive. They are only processed when the record is accessed via ChannelAccess from an operator screen where someone entered a new value for this record
- **X.Y second**: The record can be processed less often than the main

```
field(SCAN, "I/O Intr")
```

3.2.1.3. INP, OUT: Input/Output Link field

The INP field for input records or the OUT field for output records has to match the following pattern:

@<IOM IP address> D<Device> C<Channel list index> B<first bit of interest>

- IOM IP address: The IP address of the IOM where the channel is located
- Device: The index of the device where the channel is located
- Channel list index: The index of the channel in the device's channel list
- First bit of interest: The bit of interest for bi or bo record.

```
field(INP," @192.168.100.2 D0 C3 B2")
```

3.2.2. ai, Analog Input record fields

```
record (ai, "$(RECORDNAME)") {
 field (DTYP, "UeiDaq")
 field (INP, "@$(IOMIP) D0 C0")
 field (SCAN, "I/O Intr")
}
```

Read voltage or temperature measured on the input channels. Measurement is stored in the **VAL** field.

3.2.3. ao, Analog Output record fields

```
record (ao, "$(RECORDNAME)") {
 field (DTYP, "UeiDaq")
 field (OUT, "@$(IOMIP) D1 C0")
 field (SCAN, "Passive")
}
```

Write the value contained in the **VAL** field (as Volts) to the output channel.

3.2.4. bi, Binary Input record fields

```
record (bi, "$(RECORDNAME)") {
 field (DTYP, "UeiDaq")
```

```
 field (INP, "@$(IOMIP) D2 C0 B2")
 field (SCAN, "I/O Intr")
}
```

Read a single bit from a DI channel. The bit of interest is specified using the **B** parameter in the **INP** field.

The DI channel is read to **RVAL** and masked with (1<<B). **VAL** is 1 if **RVAL** is not 0

B can vary from 0 to 31. Bit 0 is the least significant bit. In little endian byte order, bit 0 is in the first byte, in big endian byte order it is in the last byte of the DI channel.

3.2.5. bo, Binary Output record fields

```
record (bo, "$(RECORDNAME)") {
 field (DTYP, "UeiDaq")
 field (OUT, "@$(IOMIP) D2 C0 B2")
 field (SCAN, "Passive")
}
```

Writes a single bit to a DO channel. The bit of interest is specified using the **B** parameter in the **OUTP** field.

If **VAL** is not 0, then **RVAL** is set to (1<<B), else **RVAL** is set to 0. Only the referenced bit of the DO channel is changed while all other bits remain untouched. Thus, other output records can write to different bits of the same DO channel.

B can vary from 0 to 31. Bit 0 is the least significant bit. In little endian byte order, bit 0 is in the first byte, in big endian byte order it is in the last byte of the DI channel.

3.2.6. mbbi, mbbiDirect, Multi-Bit Binary Input record fields

```
record (mbbi, "$(RECORDNAME)") {
 field (DTYP, "UeiDaq")
 field (INP, "@$(IOMIP) D2 C0")
 field (SCAN, "I/O Intr")
 field (NOBT, "4")
 field (SHFT, "0")
}
```

Read multiple consecutive bits from a DI channel.

The DI channel is read to **RVAL**, shifted right by **SHFT** bits and masked with **NOBT** bits. **NOBT+SHFT** must not exceed the number of bits of the DI channel.

Bit 0 is the least significant bit. In little endian byte order, bit 0 is in the first byte, in big endian byte order it is in the last byte of the DI channel.

3.2.7. mbbo, mbboDirect, Multi-bit Binary Output record fields

```
record (mbbo, "$(RECORDNAME)") {
 field (DTYP, "UeiDaq")
 field (OUT, "@$(IOMIP) D2 C0")
 field (SCAN, "Passive")
 field (NOBT, "8")
 field (SHFT, "8")
}
```

Write multiple consecutive bits to a DO channel.

RVAL is masked with **NOBT** bits, shifted left by **SHFT** bits and written to the DO channel. **NOBT**+**SHFT** must not exceed the number of bits of the DO channel.

Bit 0 is the least significant bit. In little endian byte order, bit 0 is in the first byte, in big endian byte order it is in the last byte of the DO channel.

Only the referenced **NOBT** bits of the DO channel are changed. All other bits remain untouched. Thus, other output records can write to different bits of the same DO channel.

4. Test

Configure a few devices in the **st.cmd** file as shown in section 3.1.8, modify the **dbLoadRecords** call to load your dbd file.

Create a dbd file and configure a few records in to access one AI channel and one AO channel

```
record(ai, "uei:ai_test"){
 field(SCAN, "I/O Intr")
 field(DTYP, "UeiDaq")
 field(INP, "@192.168.100.2 D1 C0")
}

record(ao, "uei:ao_test"){
 field(SCAN, "Passive")
 field(DTYP, "UeiDaq")
 field(OUT, "@192.168.100.2 D0 C0")
}
```

Run the ueidaq IOC:

```
cd iocboot/iocueidaq
../../bin/O.linux-x86/ueidaq st.cmd

Starting iocInit
############################################################################
###############
## EPICS R3.14.12.2 $Date: 2012-04-04$
## EPICS Base built Mar 27 2012
############################################################################
###############
drvUeiDaq: Delaying launch of scan task for IOM
'192.168.100.2' until database ready
drvUeiDaq: launch scan task for IOM '192.168.100.2'
iocRun: All initialization complete
epics>
```

Check that the records are there:

```
epics> dbl
uei:ai_test
uei:ao_test
epics>
```

Read current value for ai record:

```
epics> dbpr uei:ai_test
ASG: DESC: DISA: 0 DISP: 0
DISV: 1 NAME: uei:ai_test RVAL: 0 SEVR: NO_ALARM
STAT: NO_ALARM SVAL: 0 TPRO: 0
VAL: -0.002059967956054
epics>
```

On a remote computer where you installed EPICS base. Run **cainfo** to get information about **uei:ai_test** variable

```
C:\work\epics\base\bin\win32-x86>cainfo uei:ai_test
uei:ai_test
State: connected
Host: 192.168.1.18:5064
Access: read, write
Native data type: DBF_DOUBLE
Request type: DBR_DOUBLE
Element count: 1
```

Run **caget** to get the variable's value

```
C:\work\epics\base\bin\win32-x86>caget uei:ai_test
uei:ai_test -0.00114443
```

Run **caput** to write a new value to **uei:ao_test**

```
C:\work\epics\base\bin\win32-x86>caput uei:ao_test 4.0
Old : uei:ao_test 0
New : uei:ao_test 4
```

Run **camonitor** to continuously monitor **uei:ai_test**

```
C:\work\epics\base\bin\win32-x86>camonitor uei:ai_test
uei:ai_test 2012-04-04 16:22:30.388435 2.01854
uei:ai_test 2012-04-04 16:22:30.419439 2.01946
uei:ai_test 2012-04-04 16:22:30.429420 2.01854
uei:ai_test 2012-04-04 16:22:30.459424 2.01762
uei:ai_test 2012-04-04 16:22:30.469413 2.01854
uei:ai_test 2012-04-04 16:22:30.490405 2.01808
uei:ai_test 2012-04-04 16:22:30.500414 2.01717
uei:ai_test 2012-04-04 16:22:30.510407 2.01762
uei:ai_test 2012-04-04 16:22:30.530398 2.01854
uei:ai_test 2012-04-04 16:22:30.540411 2.01717
uei:ai_test 2012-04-04 16:22:30.550418 2.01808
```

5. Adding support module to application

Several files need minor modifications to use UEI support module in an application.

5.1. Add the full path to the UEI support directory to the application configure/RELEASE file:

```
UEIDAQ=xxxx/ueidaq-x.y.z
```

5.2. Add UEIDAQ support to application database definition file

The application database definition file must include the database definition files for the HP3458A instrument and for any needed ASYN drivers. There are two ways that this can be done:

If you are building your application database definition file from an xxxInclude.dbd file you include the additional database definitions in that file:

```
include "base.dbd"
include "ueidaq.dbd"
```

If you are building your application database definition file from the application Makefile your specify the aditional database definitions there:

```
xxx_DBD += base.dbd
xxx_DBD += ueidaq.dbd
```

5.3. Add the UEIDAQ support libraries to the application

You must link the HP3458A support library and the ASYN support library with the application. Add the following lines:

```
xxx_LIBS += ueidaqSupport
```

before the

```
xxx_LIBS += $(EPICS_BASE_IOC_LIBS)
```

in the application Makefile.

5.4. Load the UEIDAQ support database records in the application startup script:

```
cd $(UEIDAQ)
dbLoadRecords("db/ueidaqai.db")
```

or

```
cd $(UEIDAQ)
dbLoadTemplate("db/iom.substitutions")
```