OAK RIDGE National Laboratory

Allen-Bradley PLCs and EPICS

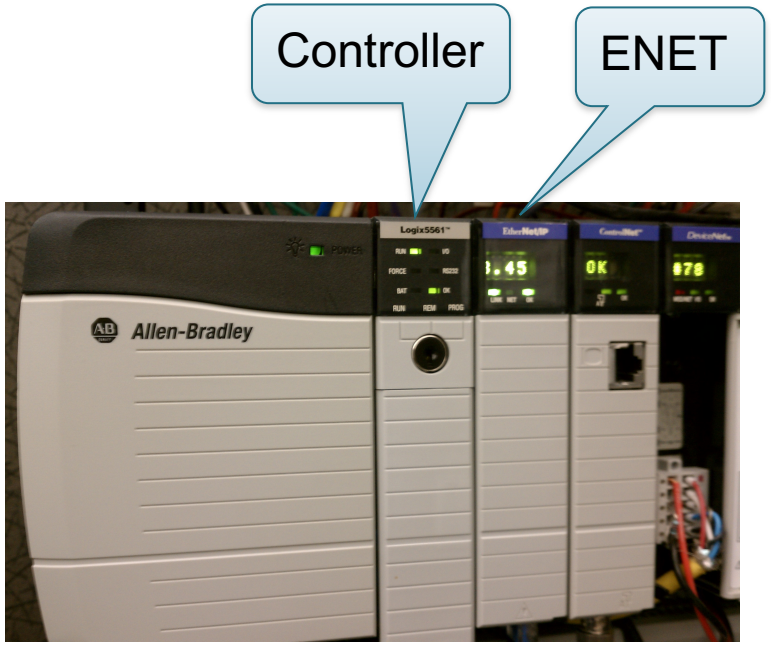
Kay Kasemir

Oct. 2021

ORNL is managed by UT-Battelle, LLC for the US Department of Energy

U.S. DEPARTMENT OF ENERGY

Allen Bradley Control Logix Series



PLC

- Controller
  - Ladder Logic
    or structured text, …
  - Named tags
    Not ‘binary table, offset 62’
- Ethernet Module
  - ‘Industry standard’
  - Combined w/ Controller
    in recent models

Program Tags - MainProgram

Scope: MainProgram Show: Show All Sort: Tag Nam

Tag Name	Alias For	Base Tag	Type
north_tank_mix			BOOL
north_tank_pressure			REAL
north_tank_temp			REAL
+-one_shots			DINT
+-recipe			TANK[3]
+-recipe_number			DINT
replace_bit			BOOL
+-running_hours			COUNTER
+-running_seconds			TIMER
start			BOOL
stop			BOOL

Monitor Tags | Edit Tags

Network Protocol EtherNet/IP

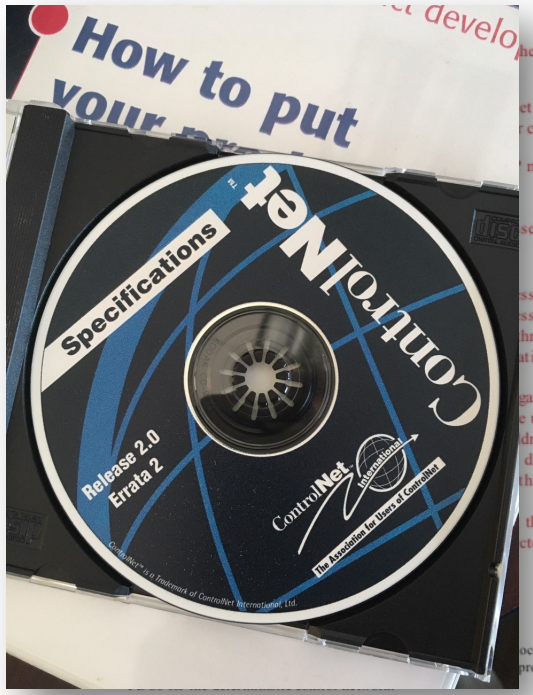
‘Control Net Over Ethernet”

- TCP encapsulation of CIP (Control & Information Protocol)

ODVA Control Net Spec

- How to send message
- Read serial number of ENET module
- How to send/receive CIP via TCP

.. but how to read tags?



he IPv4 Organization Local Scope: 239.192.0.0/14

et of IP multicast addresses to use for multicast
choosing IP multicast addresses:

multicast addresses to use. The actual

ses to use, based on their IP address, according to

sses, based on the host's IP address. If a subnet
ss to obtain the relative host number: host 1 uses
hrough 239.192.1.63, and so on. (Remember that the
ative assignments, as noted above.)

ganization Local Scope, if the subnet mask results
e used. If nosubnet mask is in use, then the HOST
dresses only the low-order 10 bits of the HOST ID
different devices could produce data on the same
his (see below).

the same IP multicast addresses. Consequently,
cted data based on the Connection ID and the IP

ocol is required. This section defines the encapsulation
protocol that may be used for transporting data other than

8.6.1 Encapsulation messages

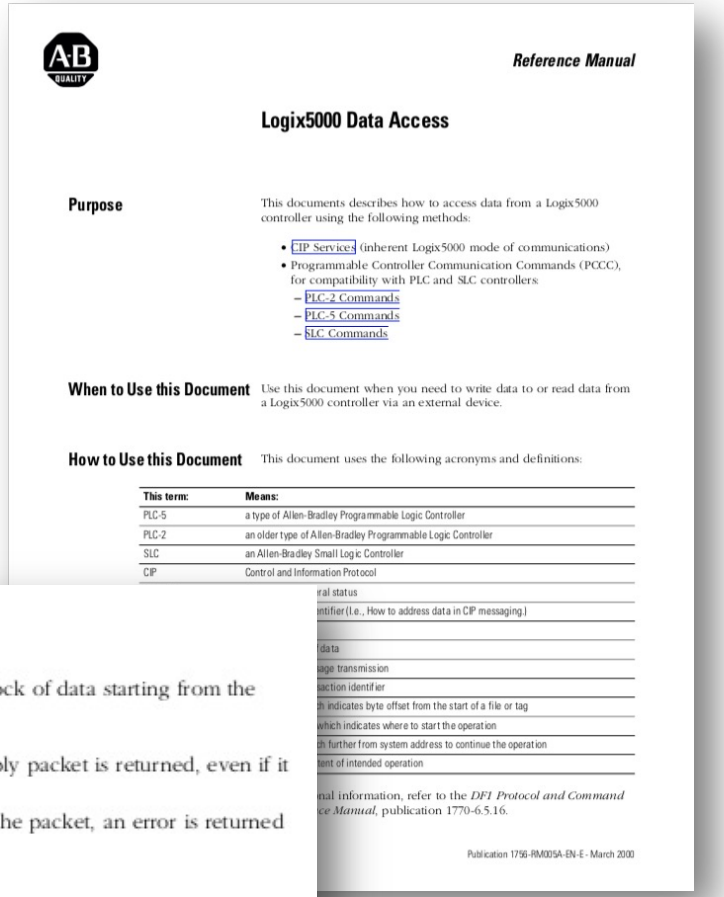
All encapsulation messages shall be composed of a fixed-length header of 24 bytes followed by an optional data portion. The total encapsulation message length (including header) shall be limited to 65535 bytes. The encapsulation message length shall not override length restrictions imposed by the encapsulated protocol. Its structure shall be as shown in Figure 4-73, where the top of the diagram indicates the portion of the message sent first on the wire.

NOTE: The encapsulation data portion of the message is required only for certain commands.

Encapsulation Header (24 bytes)
Encapsulation Data (max size: 65511 bytes)

Allen Bradley Network Protocol

- 1756-RM005A-EN-E.pdf
  - Read tag
  - Write tag
  - Combine commands



Reference Manual

Logix5000 Data Access

Purpose

This documents describes how to access data from a Logix5000 controller using the following methods:

- CIP Services (inherent Logix5000 mode of communications)
- Programmable Controller Communication Commands (PCCC), for compatibility with PLC and SLC controllers:
  - PLC-2 Commands
  - PLC-5 Commands
  - SLC Commands

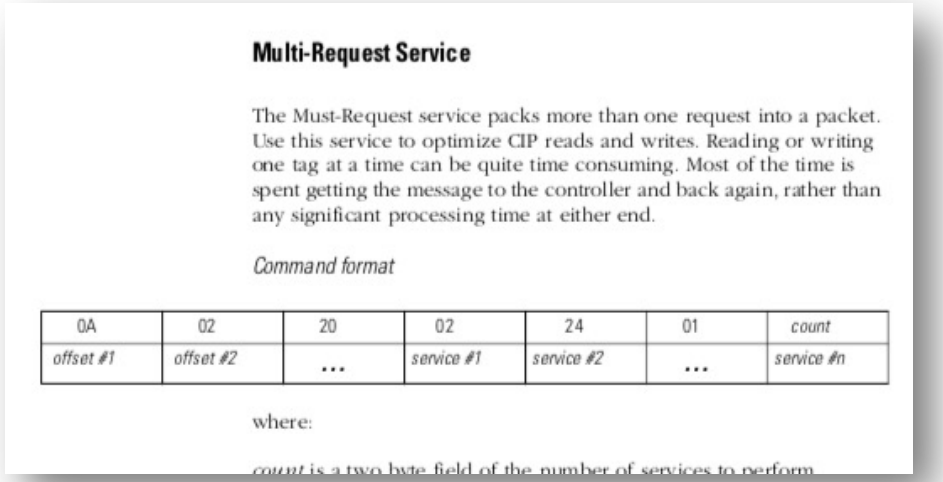
When to Use this Document

Use this document when you need to write data to or read data from a Logix5000 controller via an external device.

How to Use this Document

This document uses the following acronyms and definitions:

This term:	Means:
PLC-5	a type of Allen-Bradley Programmable Logic Controller
PLC-2	an older type of Allen-Bradley Programmable Logic Controller
SLC	an Allen-Bradley Small Logic Controller
CIP	Control and Information Protocol



Multi-Request Service

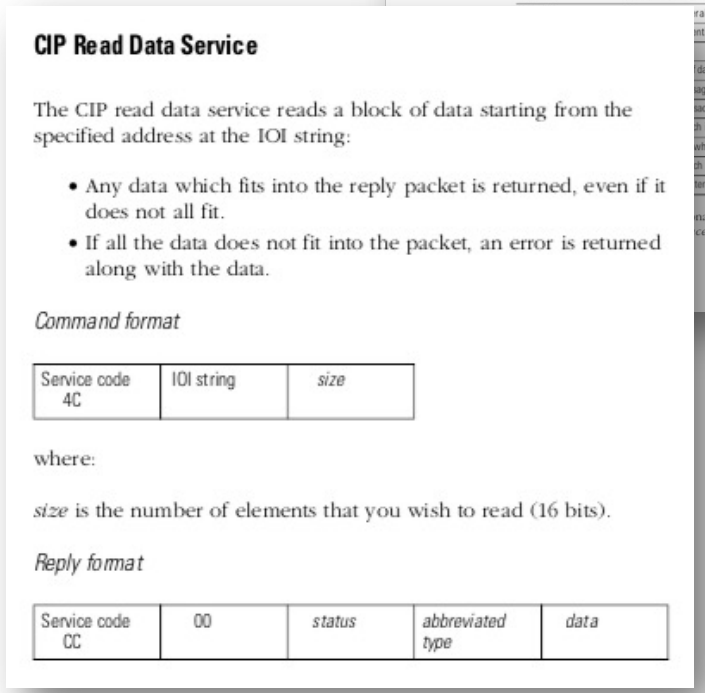
The Must-Request service packs more than one request into a packet. Use this service to optimize CIP reads and writes. Reading or writing one tag at a time can be quite time consuming. Most of the time is spent getting the message to the controller and back again, rather than any significant processing time at either end.

Command format

0A	02	20	02	24	01	*count*
offset #1	*offset #2*	...	*service #1*	*service #2*	...	*service #n*

where:

count is a two byte field of the number of services to perform



CIP Read Data Service

The CIP read data service reads a block of data starting from the specified address at the IOI string:

- Any data which fits into the reply packet is returned, even if it does not all fit.
- If all the data does not fit into the packet, an error is returned along with the data.

Command format

Service code 4C	IOI string	*size*

where:

size is the number of elements that you wish to read (16 bits).

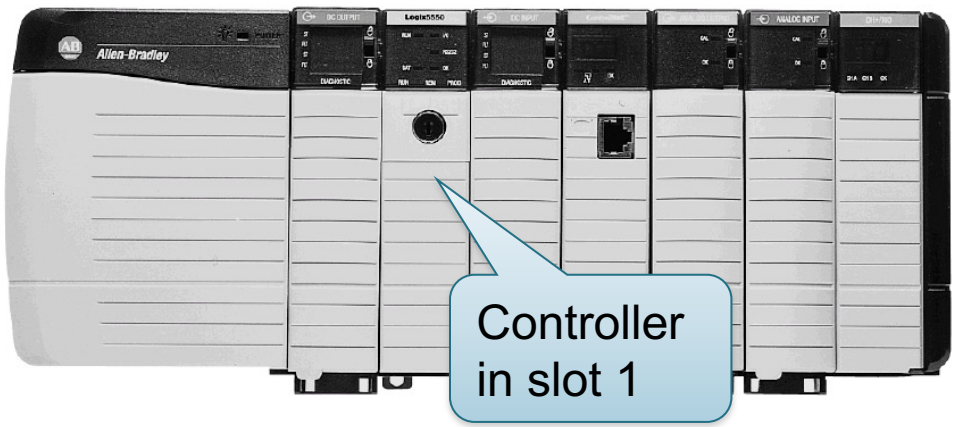
Reply format

Service code CC	00	*status*	*abbreviated type*	*data*

EPICS "Ether IP" Support

- Developed ~2000 for SNS
  - Key to its operation: Vacuum, cryo, water, …
  - Now used by many EPICS sites
- Convenient
  - Compared to serial, ControlNet, …:
    No special interface hardware nor wiring
  - Can read/write any controller tag on the PLC,
    no "publish" required
  - "Fast enough"
- Supports AllenBradley ControlLogix, CompactLogix

https://controlssoftware.sns.ornl.gov/etherip/

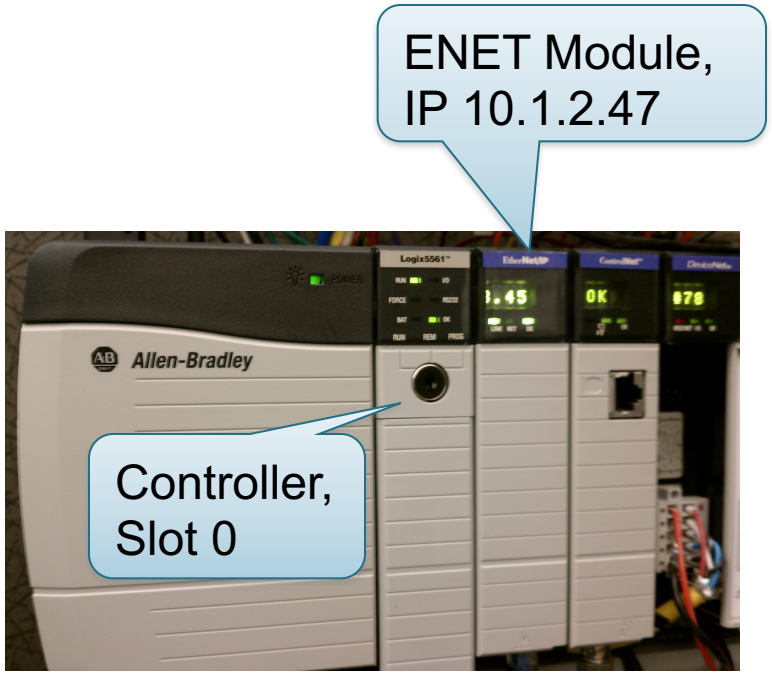


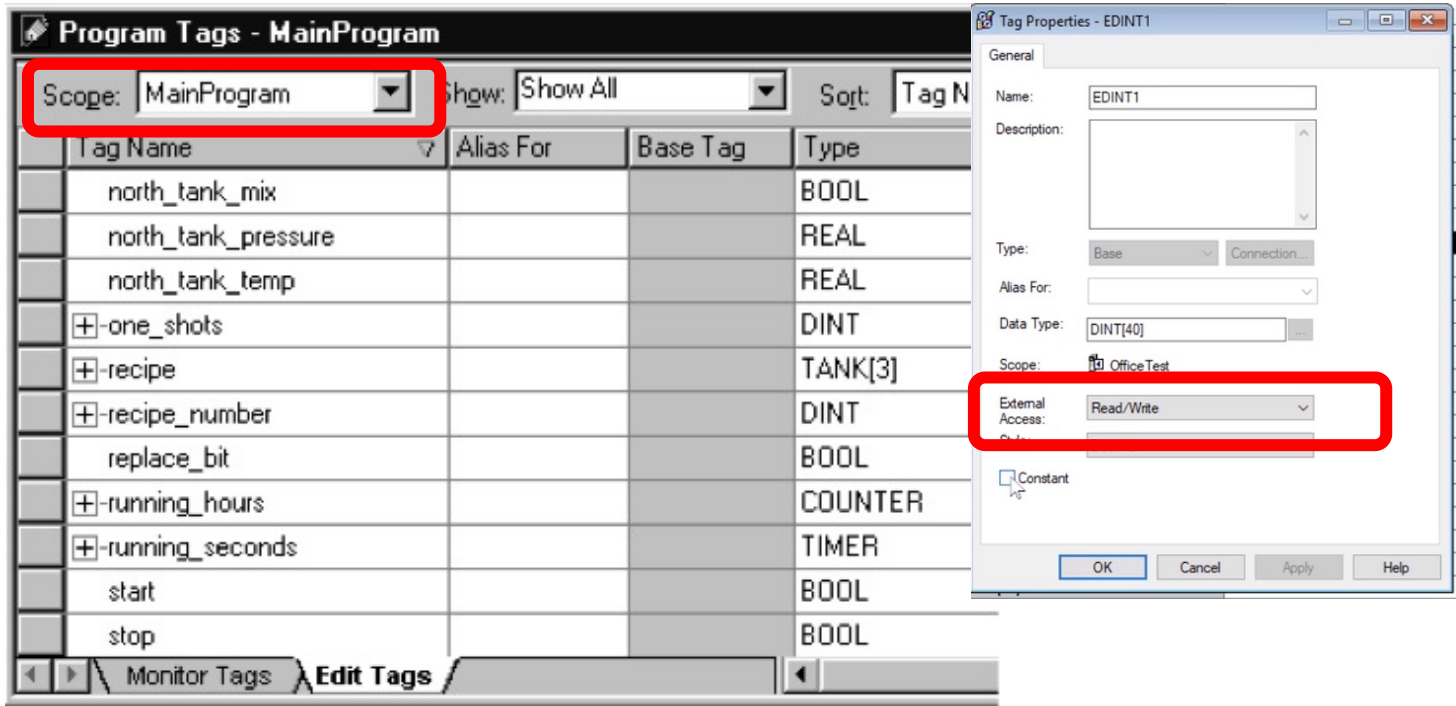
PLC Requirements

- ENET Module
  - .. Or new CPU with built-in ENET
  - – Know the IP address
  - – Know the CPU slot number

    One PLC could have multiple controllers!
- Tags
  - – Know tag names
  - – "Controller" level, not! private to program
  - – No need to "produce"

    but new versions may require allowing "External Access"



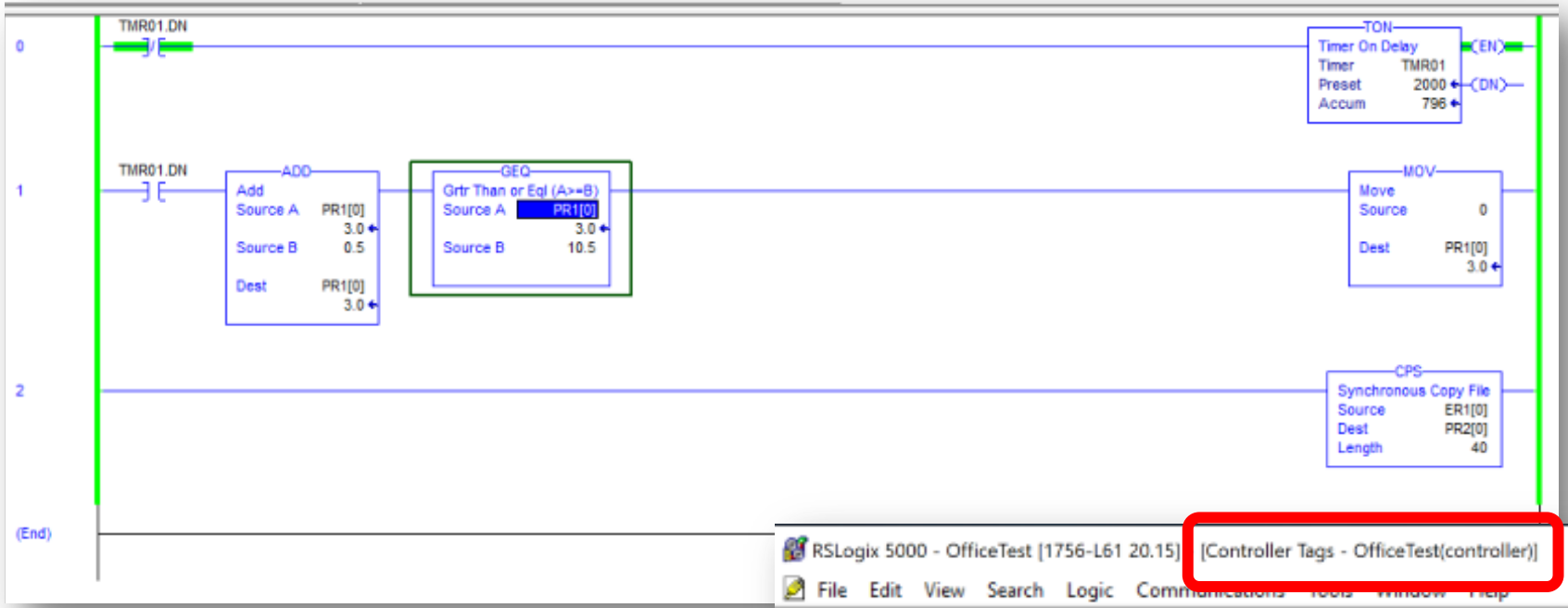


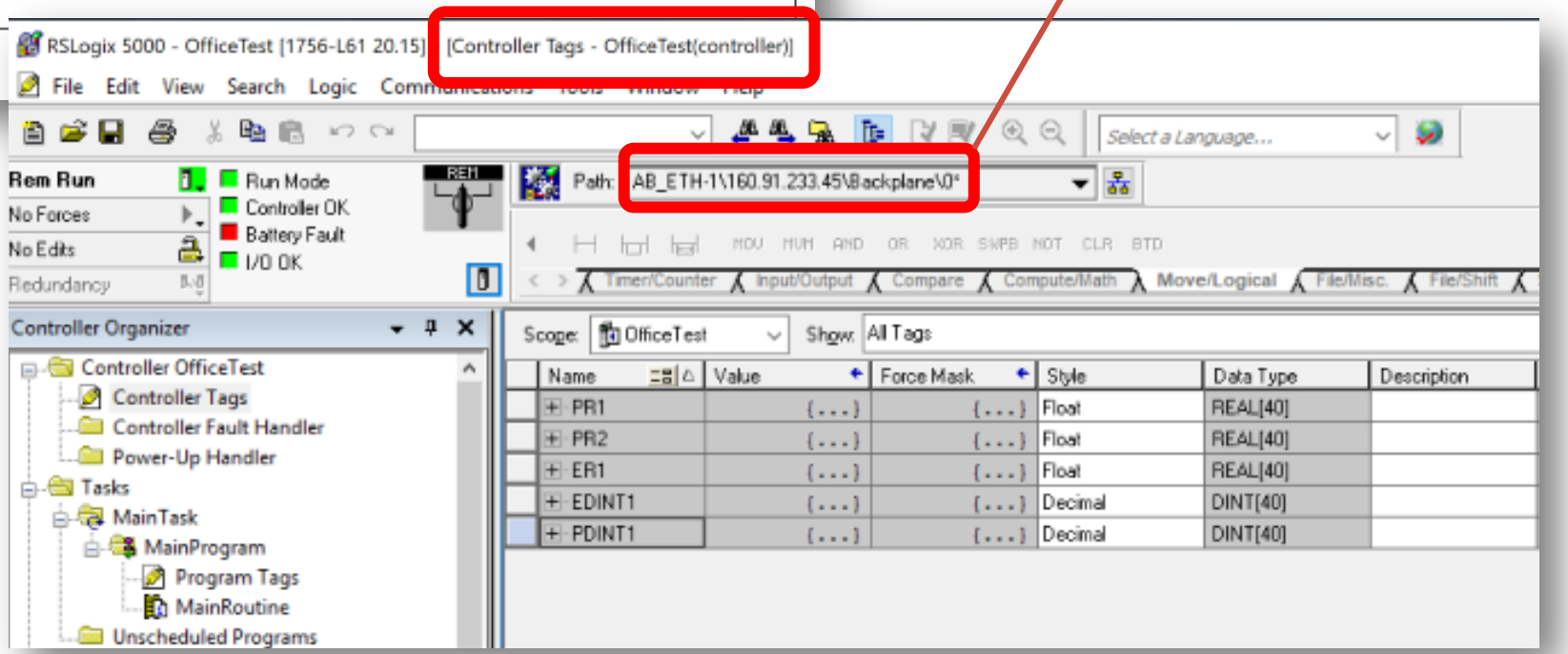
OAK RIDGE National Laboratory | SPALLATION NEUTRON SOURCE

Demo Setup

These examples assume that a PLC with tags as shown in the following slides is accessible on the network.

Example PLC





ENET IP: 160.91.233.45
Controller in slot 0

Basic Protocol Test

```
$ ether_ip_test -h

Usage: ether_ip_test <flags> [tag]

Options:

    -v verbosity

    -i ip  (as 123.456.789.001 or DNS name)

    -p port

    -s PLC slot in ControlLogix crate (default: 0)

    -t timeout (ms)

    -a array size

    -w <double value to write>
```

Protocol Tests

```
# Read changing number (call a few times to see changes)
ether_ip_test -i 160.91.233.45 PR1


# Same, but whole array (compare time to single element read)
ether_ip_test -i 160.91.233.45 -a 40 PR1


# Try reading too much
ether_ip_test -i 160.91.233.45 -a 41 PR1


# Show protocol details
ether_ip_test -v 10 -i 160.91.233.45 PR1


# Write array elements, then read whole array
ether_ip_test -i 160.91.233.45 -w 40 ER1[1]
ether_ip_test -i 160.91.233.45 -w 42 ER1[2]
ether_ip_test -i 160.91.233.45 -a 40 ER1
```

EPICS Records

```
# Run like this:
#  eipIoc -p my_plc=160.91.233.45 -d plc.db


record(ai, "number1")
{
  field(DTYP, "EtherIP")
  field(INP,  "@my_plc PR1[0]")
  field(SCAN, "1 second")
}
```

In IOC console:
- help
- dbl
- dbpr number1
- drvEtherIP_dump
- drvEtherIP_report 10
- EIP_verbosity 10

In other terminal:
- camonitor number1

See also VM in /ics/examples/15_plc

Starting the IOC: Development

Type this on one line:

```
eipIoc

  -p my_plc=10.1.2.47

  -m "P=MyPrefix,N=2"

  -d example.db
```

'softIoc' with EtherIP built in

`-p "name=ip,slot"`
Define PLC name for "@my_plc" in INP/OUT links,
IP address, optional CPU slot

`-m "macro=value"`
Define database macros

`-d path/to/some.db`
Load database

EPICS Records

Input Records (ai, bi, mbbi, mbbiDirect, stringin)

```
field(DTYP, "EtherIP")
field(INP,  "@my_plc name_of_tag")
field(SCAN, "1 second”)
```

Output Records (ao, bo, mbbo, mbboDirect)

```
field(DTYP, "EtherIP")
field(OUT,  "@my_plc name_of_tag")
```

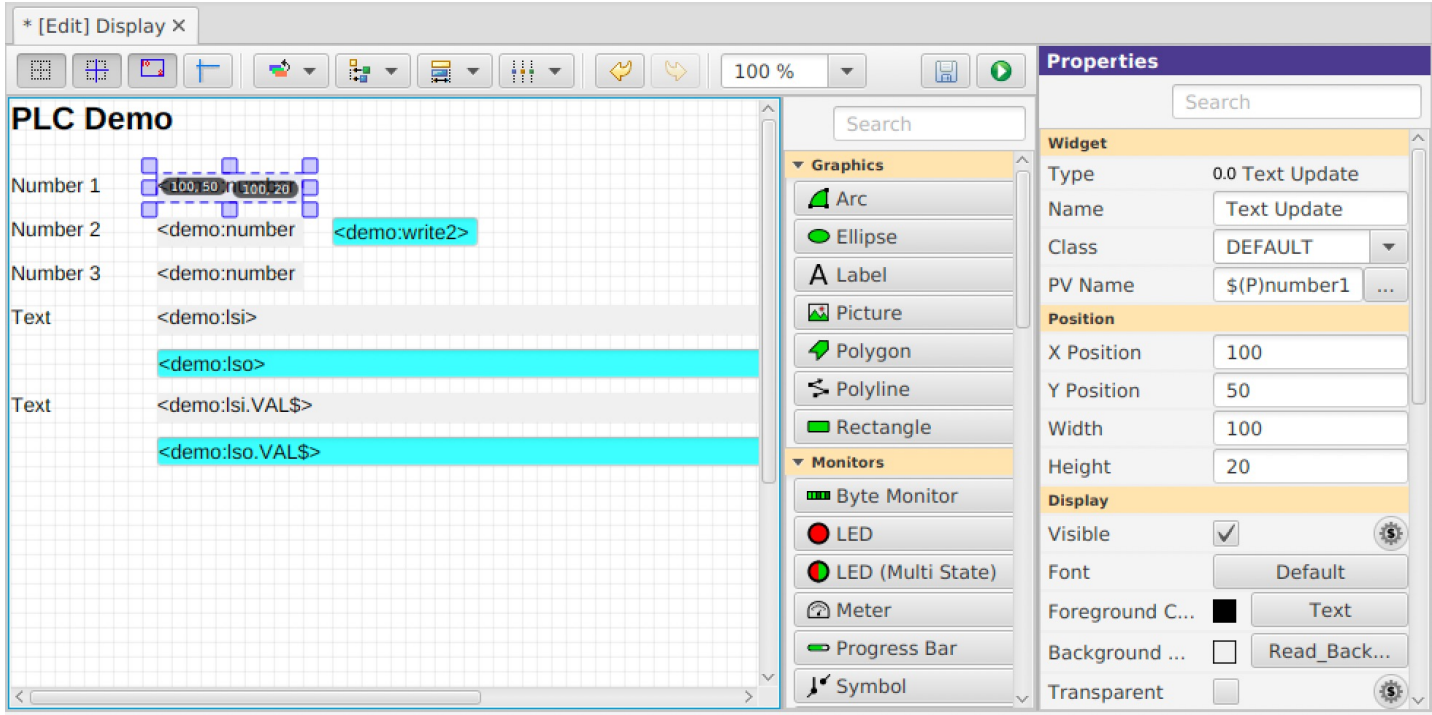
➔Need to be familiar with EPICS Records, https://epics.anl.gov/base/R7-0/4-docs/RecordReference.html

Tools on IOC

- `drvEtherIP_dump`
  - Dumps all tags and their value
- `drvEtherIP_report` *`level`*
  - Dumps info about each plc, period list, tag
  - Level 0..10
- Set records' `TPRO` field
  - Show when tag is read/written by that record
- `EIP_verbosity(7)`, `EIP_verbosity(10)`
  - Show warnings or details of protocol

Display (CS-Studio)

- Start ‘css’
- Use menu ‘Applications’, ‘Display’, ‘New Display’, enter file name
- Select ‘Text Update’ from ‘Monitors’
- Enter ‘PV Name’
- Click ‘Run’ button 



➔ Does the number show up as expected?

Driver Combines Tags used by Records

plc2.db

Records

1. INP "@plc1 arrayA[2]", SCAN "2 second"
2. INP "@plc1 arrayA[3]", SCAN "2 second"
3. INP "@plc1 arrayB[13]", SCAN "1 second"
4. INP "@plc2 arrayA[2]", SCAN "2 second"

Driver

1. "plc1"
   - Handle "arrayA[0-3]" every 2 seconds
   - Handle "arrayB[0-13]" every 1 second
2. "plc2"
   - Handle "arrayA[0-2]" every 2 seconds

Scan Flag “ S <seconds>”

Records

1. INP “@plc array[2]”, SCAN “2 second”
2. INP “@plc array[3]”, SCAN “1 second”
3. INP “@plc array[13]”, SCAN “I/O Intr”
4. OUT “@plc array[5]”, SCAN “Passive”

Driver

Uses the “fastest” SCAN of any record related to a tag.

Records 2 & 3 should add “@plc array[..] S 1” to provide suggested scan period, otherwise warning

`“cannot decode SCAN field, no scan flag given”`

PLC Data Type ⇔ Record Type

REAL, INT, DINT	⇔	ai, ao (REAL? VAL, else RVAL)
INT, DINT, BOOL[]	⇔	bi, bo, mbbi, mbbo, mbbiDirect, mbboDirect
BOOL	⇔	bi, bo
STRING	⇔	stringin, stringout

For arrays, access one element:

```
field(INP,"@my_plc my_array[42]")
```

For structures, access one element:

```
field(INP,"@my_plc gadget.ps.voltage")
```

Bitfiddly Stuff

plc3.db

Internally, DINT and BOOL[32] look similar.

Driver assumes bit for binary records.

"some_tag[40]"

a) ai, ao:
   Element 40 of a REAL, INT, DINT array

b) bi, bo, mbb*:
   Bit 40 of a BOOL array,
   or
   bit 40 of a DINT array, i.e. bit 8 of DINT[1]

   To force [i] to be the i^{th} array element, add 'Bit' flag:
   "some_tag[1] B 8"

Reading, Writing, Scanning

Records

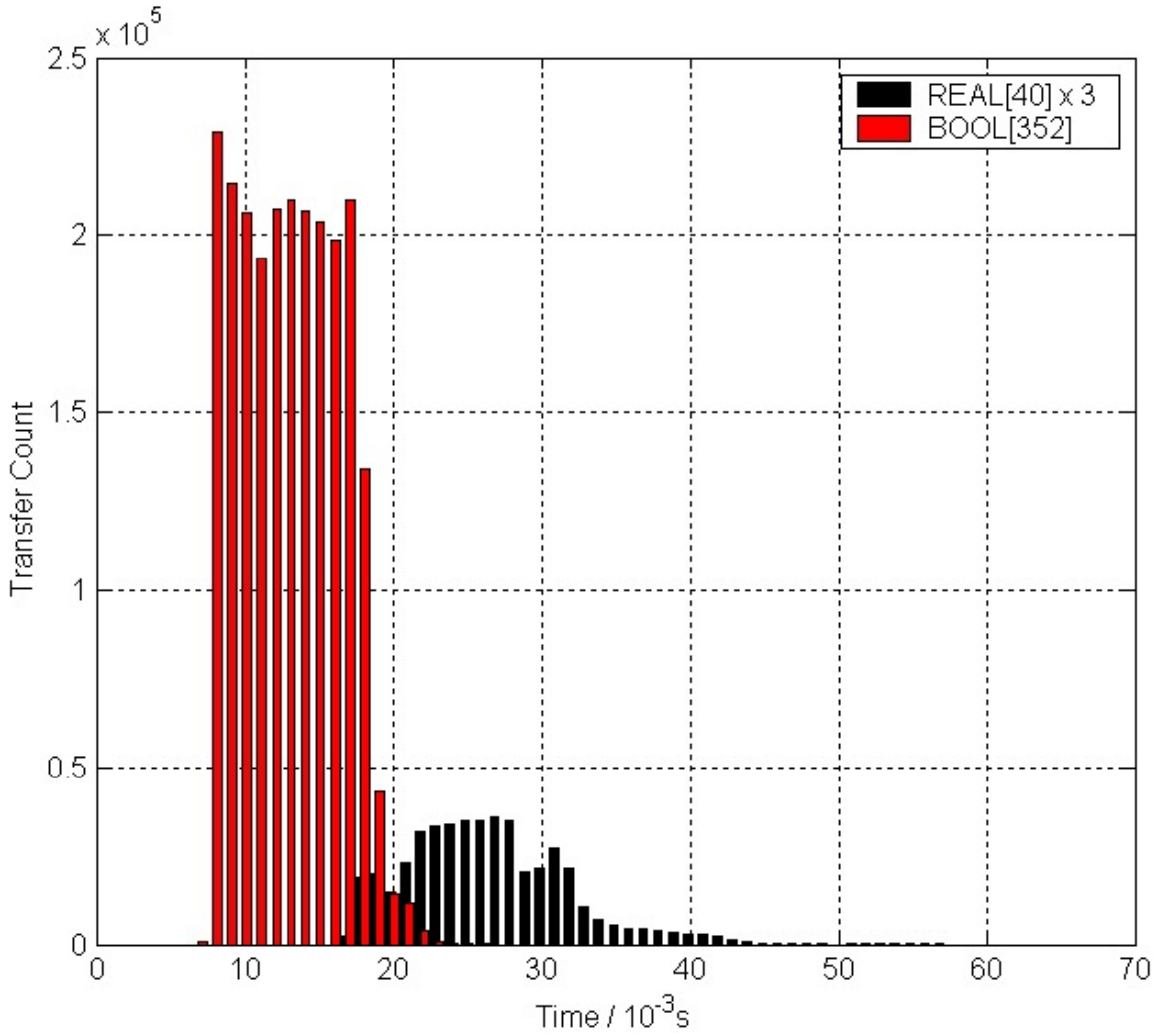
1. INP "@plc array[2]", SCAN "2 second"
2. INP "@plc array[3]", SCAN "1 second"
3. INP "@plc array[13]", SCAN "I/O Intr"
4. OUT "@plc array[5]", SCAN "Passive"

Driver

"array[0-13]" read every 1 second

1. Reads most recent value[2] every 2 seconds
2. .. [3] every 1 second, data may be 0.999 sec old
3. Processes whenever value [13] changes
4. Reads (though *output!*). Checks if value [5] changes. When processed, updates value [5], marks whole tag for writing. Driver will then once *write* instead of read.

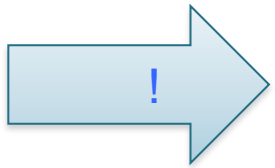
Transfer Time Histogram



352 BI @ 10Hz, 3 x REAL[40] @ 2 Hz

Arrays, Aliases

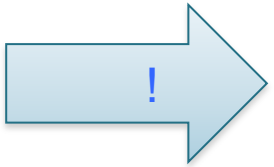
- Arrays are MUCH more effective
  - Read scalar REAL tag: 10..30ms
  - Read REAL[40]: 10..30ms
  - Read BOOL[352]: 10..30ms
  - Read 352 separate BOOL tags: ~352 * (10..30) ms
- Records with INP or OUT that access
  - INP="@plc my_array[2]"
  - INP="@plc my_array[39]"
  - → Driver reads my_array[0..39]



1. **Arrange data in array tags**
2. **Configure INP/OUT to use array elements**
3. **May use alias tags within PLC logic, or designated read-inputs, write-outputs tasks at start/end of your ladder logic**

Write vs. Read Arrays

- "Output" records read
  - Just in case somebody else (PanelView, other IOC) changes tag
- Writing one or more array elements means writing whole array
  - Gets confusing when both PLC and IOC update elements in same array

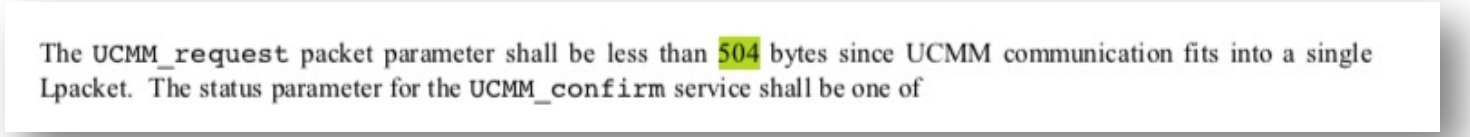


1. **Arrange data in array tags**
2. **Use separate arrays for 'IOC reads' and 'IOC writes'**
3. **Configure INP/OUT to use array elements**
4. **May use alias tags within PLC logic**

The Elusive Buffer Limit

Driver combines several reads and writes for tags into one network transfer until either the request or the response reaches `EIP_buffer_limit`

How large?
About 500 bytes, details fuzzy


The UCMM_request packet parameter shall be less than 504 bytes since UCMM communication fits into a single Lpacket. The status parameter for the UCMM_confirm service shall be one of

Set too large

→ PLC error "Buffer too small, partial data only"

Set too small

→ Inefficient, can't combine many tags

Set way too small

→Can't read a single REAL[40] array, driver error
"Tag 'xyz' exceeds buffer limit of .. Bytes"

Don't mess with the default EIP_buffer_limit.

Keep arrays to REAL/DINT/INT[40] and BOOL[400]. Driver will then combine several of those in one transfer.

Elementary “ E” Flag

Records

1. INP “@plc array[190]”
2. OUT “@plc array[191]”

Driver

1. Reads array[0..191], but we may only use [190], [191]
   ➔
   “@plc array[190] E“ causes transfer of just that array element
2. Writes all of [0..191] even if just 191 was changed, *writing* elements which we otherwise wanted to *read*
   a) Use separate arrays for “writing” and “reading”
   b) Add “ E” to write just that element, not whole array

Summary

- Arrange PLC data in arrays
  - Alias tags to document PLC logic
  - Separate arrays for reading, writing
  - Use about 40 elements per array (400 for BOOL)
- Point records' INP/OUT to those array tags
- Create display (CS-Studio, EDM, …)
- Done

Starting the IOC: Production Setup

Add driver to ‘makeBaseApp’ type IOC:

configure/RELEASE:

```
ETHER_IP=/path/to/share/ether_ip
```

yourApp/src/Makefile:

```
your_DBD += ether_ip.dbd
your_LIBS += ether_ip
```

iocBoot/iocYours/st.cmd:

```
drvEtherIP_init()
drvEtherIP_define_PLC("my_plc", "10.1.2.47", 0)
dbLoadRecords(…
```

Assume you have…

- 3 vacuum PLCs, basically the same ladder logic and tags, but for different sectors of the machine

Sector 1 Vacuum PLC

Sector 2 Vacuum PLC

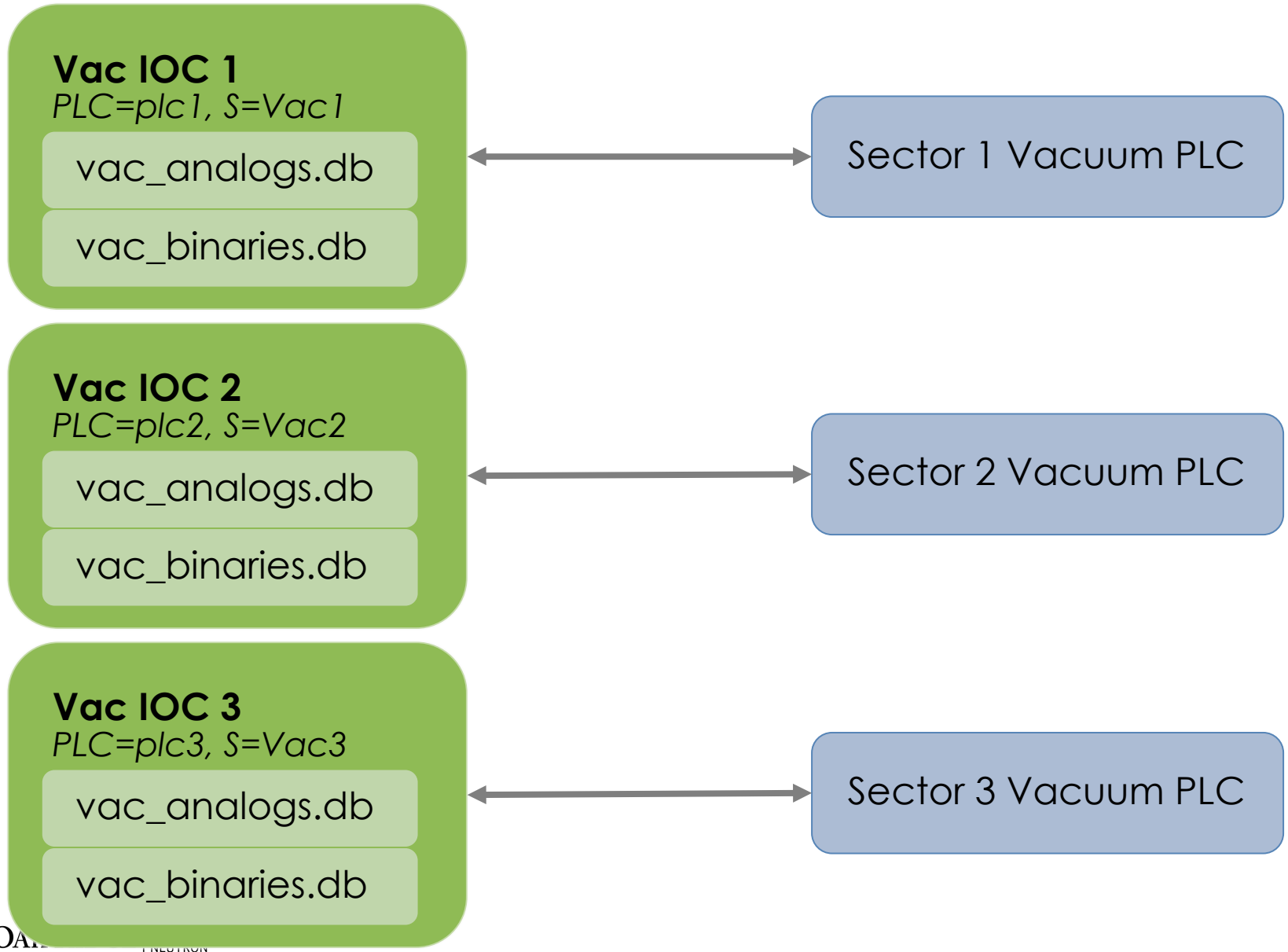
Sector 3 Vacuum PLC

- Database split for convenience into two files
  - Using macros "$(PLC)" to select PLC and "$(S)" for start of record names

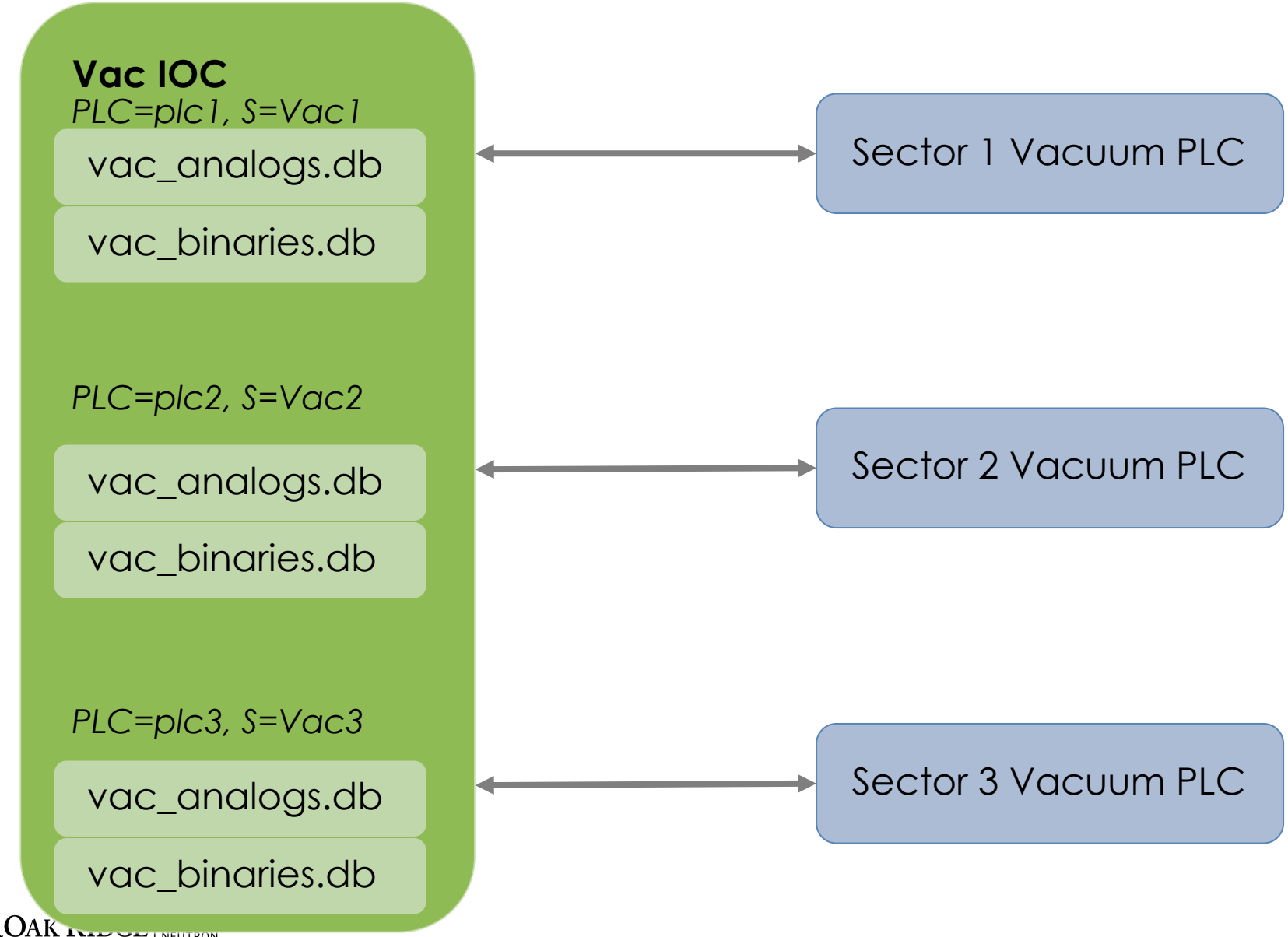
vac_analogs.db

vac_binaries.db

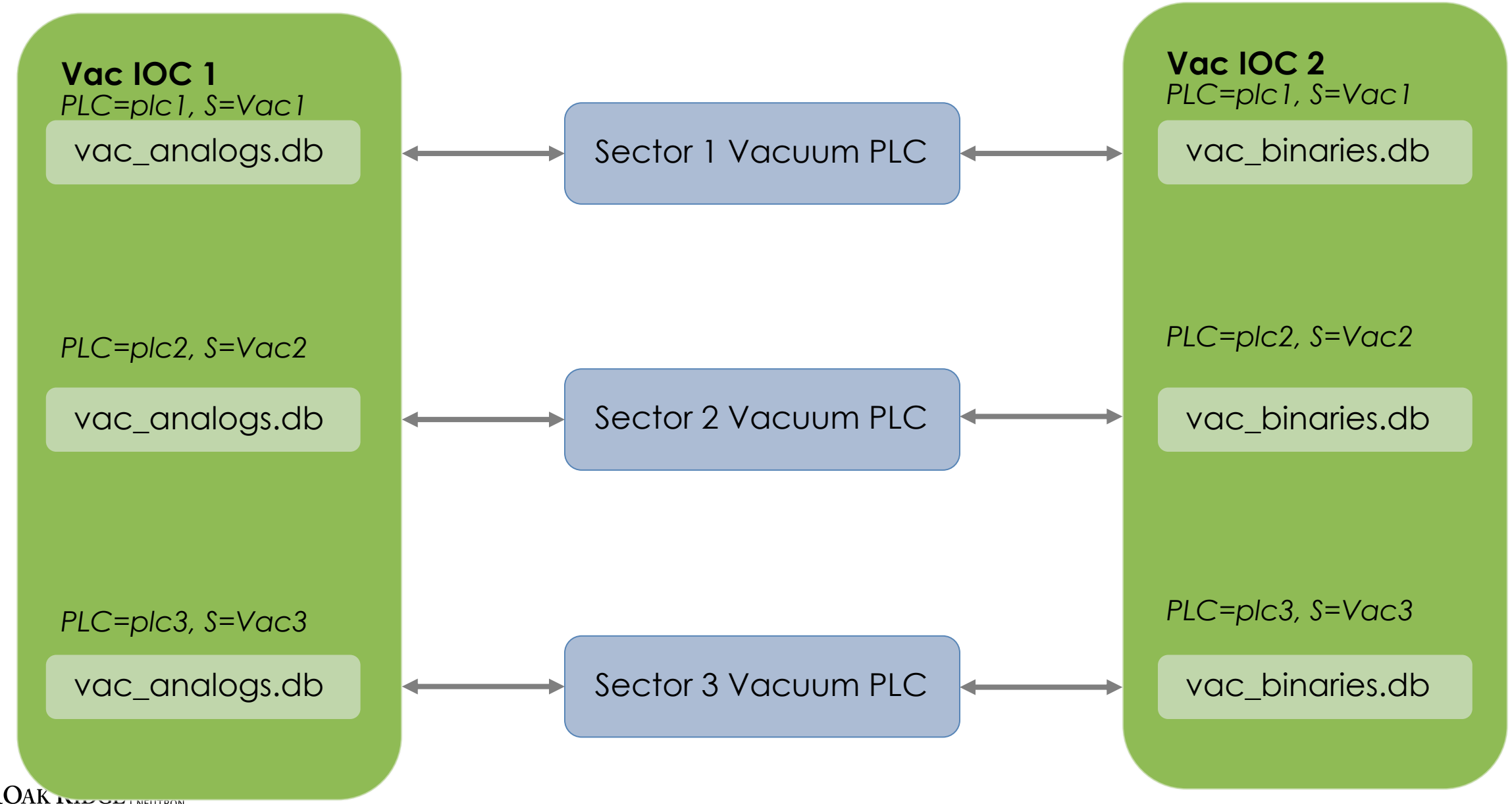
Option 1: One IOC per PLC



Option 2: One IOC for all PLCs



Option 3: One IOC for analogs, one for binaries



	Pro	Con	Verdict
One IOC per PLC	Can reboot IOC for one sector and still operate other sectors' vacuum	More IOCs to maintain. Top-level vacuum display needs to connect to 3 IOCs.	May be necessary if single IOC gets too busy.
One IOC for all PLCs	Fewer IOCs to maintain. Top-level vacuum display can get all data from one IOC.	Reboot or problem of IOC will interrupt all vacuum access.	Simple and good, especially to get started.
One IOC for analogs, another for binaries	??	Each PLC has multiple IOC connections. Vacuum displays need multiple IOC connections.	Don't