OAK RIDGE National Laboratory

‘makeBaseApp’ IOC

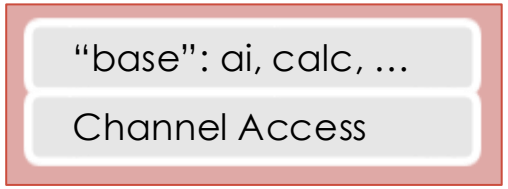
Kay Kasemir

July 2026

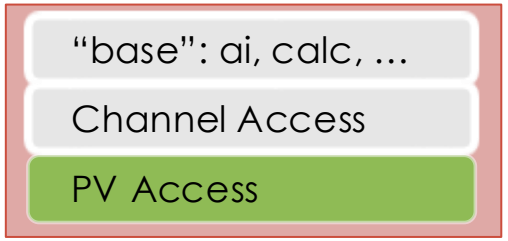
ORNL is managed by UT-Battelle, LLC for the US Department of Energy

U.S. DEPARTMENT OF ENERGY

IOC binaries

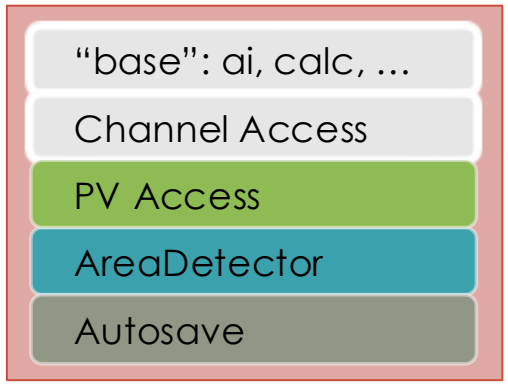


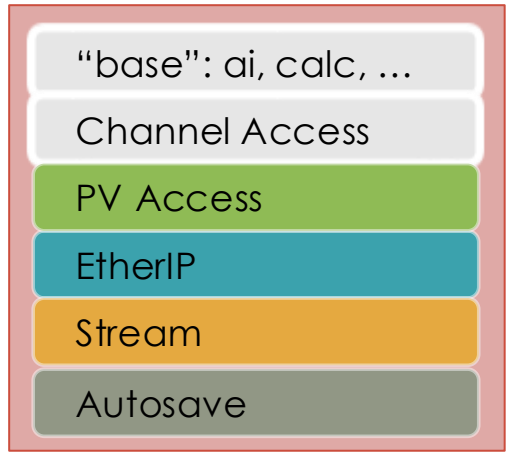
- softIoc
  - Executes *.db files with ai, calc, …
- softIocPVA, softIocPVX
  - Adds PVAccess server



- eipIoc
  - Adds EtherIP

What if I want ‘myCustomIoc’ that includes PVA, EtherIP, Autosave, Stream Device, …?





‘Soft IOC’, ‘Real IOC’, ‘Host’, ‘Target’

- Host-based, aka “soft” IOC
  - Runs on same type of host (Linux, Mac, Windows) on which it’s compiled
  - IOC is just another program on the host
  - May run many IOCs on the same host
  - Examples: `softIoc`, `softIocPVA`, all IOCs in this training session
- Target IOC
  - Cross-compiled from e.g. Linux to VxWorks, RTEMS, embedded Linux
  - Runs on VME, µTCA, embedded CPU
  - IOC is the primary, maybe only program running on the target
  - IOCs on VxWorks, talking to VME hardware, were the original IOCs

A lot of EPICS code can be used on both

- Records
- Device support for networked I/O

‘makeBaseApp.pl’

Creates skeleton for custom IOC

- Directory structure
- Makefiles
- Examples: *.db, *.st, driver/device/record *.c
- IOC startup file

Two extremes

- makeBaseApp.pl –t example
  - Get most everything; you delete what’s not needed
- makeBaseApp.pl –t ioc
  - Just dirs & Makefiles; you add what’s needed

EPICS Build Facility

Is outstanding

- Builds on Linux, Mac, Windows,
  for Linux, FreeBSD, OS X, Windows, vxWorks, RTEMS, x86, x86_64, ppc, arm, …
- AppDevGuide
- Served us for decades across many changes of OSs, compilers, …

Is aggravating

- Why is it not a Visual C++, Kdeveloper, VSCode, … project? What about CMake, GNU automake, … ?
- What's the name of that option again?
- What's causing this error now?

‘demo’ based on ‘example’ skeleton

```
rm -rf /ics/examples/mine
mkdir -p /ics/examples/mine
cd /ics/examples/mine


# Create IOC application of type ‘example’,
# using ‘demo’ in the generated names
makeBaseApp.pl -t example demo


# Create IOC startup settings of type ‘example’,
# call it ‘demo’ because it’s for the app of that name
makeBaseApp.pl -t example -i demo
# When prompted, use the previously created ‘demo’
# application as the one that the IOC should load

# Compile everything
make

# Start IOC
cd iocBoot/iocdemo
chmod +x st.cmd
./st.cmd
```

Repeat after changes

Directory Layout: Key Files

```
# makeBaseApp.pl -t example demo
configure/RELEASE
configure/CONFIG_SITE
demoApp/Db/*.db
demoApp/Db/*.substitutions
demoApp/Db/Makefile
demoApp/src/Makefile

# makeBaseApp.pl -t example -i demo
iocBoot/iocdemo/Makefile
iocBoot/iocdemo/st.cmd
```

To study the skeleton, check files before the first ‘make’ or after a ‘make distclean’

configure/RELEASE

- Defines the path to EPICS base and other modules

```
EPICS_BASE=/ics/tools/base
SNCSEQ =/ics/tools/sequencer
AUTOSAVE =/ics/tools/autosave
```

- Since about version 3.15, includes ../RELEASE.local

```
basedir/RELEASE.local: Lists all the modules

basedir/top1/configure/RELEASE   - includes ../../RELEASE.local
basedir/top1/abcApp/             - uses EPICS base etc.
basedir/top1/iocBoot/            - IOC bootups

basedir/top2/configure/RELEASE   - includes ../../RELEASE.local
basedir/top2/xyzApp/             - uses EPICS base etc.
basedir/top2/iocBoot/            - IOC bootups
```

demoApp

- xyzApp/Db **Database files**
- xyzApp/src *Main.cpp,
  Sequences,
  custom device support,
  Makefile that lists required *.dbd and libs

Start over: ‘demo’ based on ‘ioc’ skeleton

```
rm -rf /ics/examples/mine
mkdir -p /ics/examples/mine
cd /ics/examples/mine


# Create IOC application of type ‘ioc’,
# using ‘demo’ in the generated names
makeBaseApp.pl -t ioc demo


# Create IOC startup settings of type ‘ioc’,
# call it ‘demo’ because it’s for the app of that name
makeBaseApp.pl -t ioc -i demo
# When prompted, use the previously created ‘demo’
# application as the one that the IOC should load

# Compile everything
make

# Start IOC
cd iocBoot/iocdemo
chmod +x st.cmd
./st.cmd
```

Repeat after changes

➔ **Empty IOC!**

How To Add Database Files

1. Create `demoApp/Db/another.db`

   For simple database, can test via
   ```
   softIoc -m "macro=value" -d another.db
   ```

2. Add to `demoApp/Db/Makefile`:

   ```
   DB += another.db
   ```

3. `make`

   Now it's under `db/another.db`

4. Add to `iocBoot/iocdemo/st.cmd`

   ```
   dbLoadRecords "db/another.db", "macro=value"
   ```

5. (Re-)start the IOC

Directory Layout: Generated Files

```
**/O.Common
**/O.linux-x86_64
**/O.*
db/*
dbd/*
include/*
lib/*
bin/*
```

Beware of difference:

- xyzApp/Db/*
  - Database 'Sources'. Edit these!
- db/*
  - 'Installed' databases, may have macros replaced. Will be overwritten by next 'make'!

Plain Database files, Macros, Templates

```
# demoApp/Db/plain.db
record(calc, "noise")
{
    field(SCAN, "1 second")
    field(CALC, "RNDM*10")
}
```

‘make’

```
# db/plain.db
record(calc, "noise")
{
    field(SCAN, "1 second")
    field(CALC, "RNDM*10")
}
```

```
# demoApp/Db/Makefile
…
DB += plain.db
```

```
# iocBoot/iocdemo/st.cmd
…
dbLoadRecords("../../db/plain.db")
```

```
# IOC shell
epics> dbl
noise
```

Open 3 terminals:

1) For running ‘make’:
   `cd /ics/examples/05_template`
2) For editing database files:
   `cd /ics/examples/05_template/demoApp/Db`
3) For running ./st.cmd
   `cd /ics/examples/05_template/iocBoot/iocdemo`

Plain Database files, Macros, Templates

```
# demoApp/Db/macro.db
record(calc, "noise$(NUM)")
{
    field(SCAN, "1 second")
    field(CALC, "RNDM*$(LIMIT)")
}
```

‘make’

```
# db/macro.db
record(calc, "noise$(NUM)")
{
    field(SCAN, "1 second")
    field(CALC, "RNDM*$(LIMIT)")
}
```

```
# demoApp/Db/Makefile
…
DB += macro.db
```

```
# iocBoot/iocdemo/st.cmd
…
dbLoadRecords("../../db/macro.db","NUM=1,LIMIT=10")
dbLoadRecords("../../db/macro.db","NUM=2,LIMIT=100")
```

```
# IOC shell
epics> dbl
noise1
noise2
epics> dbgf noise2.CALC
DBF_STRING:          "RNDM*100"
```

Plain Database files, Macros, Templates

```
# demoApp/Db/macro.db
record(calc, "noise$(NUM)")
{
    field(SCAN, "1 second")
    field(CALC, "RNDM*$(LIMIT)")
}
```

"Template"

'make'

```
# db/macros.db
record(calc, "noise1")
{
    field(SCAN, "1 second")
    field(CALC, "RNDM*10")
}
record(calc, "noise2")…
record(calc, "noise3")…
record(calc, "noise4")…
```

Expanded Template

```
# demoApp/Db/macros.substitutions
file "macro.db"
{
    { NUM="1", LIMIT="10"  }
    { NUM="2", LIMIT="100" }
}

file "macro.db"
{
    pattern { NUM, LIMIT }
            { 3,   300   }
            { 4,   400   }
}
```

Original syntax

Alternate syntax

```
# iocBoot/iocdemo/st.cmd
…
dbLoadRecords("../../db/macros.db")
```

```
# demoApp/Db/Makefile
…
# macros.db will be created
# from macros.substitutions!
DB += macros.db
```

```
# IOC shell
epics> dbl
noise1
noise2
noise3
noise4
epics> dbgf noise4.CALC
DBF_STRING:          "RNDM*400"
```

Plain Database files, Macros, Templates?

- Plain database files?
  - Add at least some naming macro like
    `record(xx, "$(P):Motor")`
    to allow running with `P=Test` or `P=ActualGadget`

- Macros in *.db
  - Whenever there's more than one instance of something, macros make sense
  - Perfectly fine to load them
    with `dbLoadRecords("file_with_macro.db","P=ABC,NUM=1")`

- Template/Substitutions?
  - Some prefer `*.substitutions`, expanded by 'make', over expanding macros in `dbLoadRecords` because file on disk then holds all the expanded record names.
    `fgrep noise1 db/*` will find that record in expanded file

*.dbd: Database Descriptions

IOC record types, device support, … are extensible

- Implement new record type, new device support: Write C/C++ code for certain interfaces, compile.
- Then 'register' your addition with core IOC code: *.dbd file

Internals:
VxWorks RTOS, the original IOC target, had runtime loader and symbol table.
RTEMS, .. don't necessarily offer this.
EPICS build facility generates IOC startup source code from *.dbd file.

How To Add Support Modules (Device, …)

Example: 'Autosave'

1. Define path in `configure/RELEASE` or better in `../RELEASE.local`

```
AUTOSAVE=/ics/tools/autosave
```

2. Add binary and DBD info to `xyzApp/src/Makefile`:

```
YourProduct_DBD += asSupport.dbd
YourProduct_LIBS += autosave
```

3. Use the support module in the IOC startup file:

```
cd ${AUTOSAVE}
dbLoadRecords "db/save_restoreStatus.db", "P=demo"
set_requestfile_path("$(TOP)/iocBoot/$(IOC)/autosave")
create_monitor_set(...)
```

Details on how to use a support module depend on the specific one, including names of provided *.dbd, binary, *.db, IOC commands

Site specifics

- Most EPICS sites use a makeBaseApp layout at the core
  - Local expert understands the details
  - Everybody else uses existing setups to add/modify database files
- Many sites wrap with local tool to
  - Create new IOC setups
  - Start/stop IOCs as ‘procServ’ Linux services with console access & log

```
[ky9@sts-icsdev2 ~]$ ics-iocs -h
usage: ics-iocs [-h] [--all] [--quiet] [--force]
                {start,stop,restart,status,info,console,log,register,unregister,list}
                [ioc]

IOCs management utility

positional arguments:
  {start,stop,restart,status,info,console,log,register,unregister,list}
                        Action to execute
  ioc                   IOC name

optional arguments:
  -h, --help            show this help message and exit
  --all, -a             Apply action for all IOCs (works for start, stop,
                        restart, status, register, unregister, info, list).
  --quiet, -q           Supress log output
  --force, -f           Force attempt at action if IOC not found in XML file.
```

```
$ ioc create --engineer "Ozzy Osbourne”
   --location "8600 J-105”
   --host "ics-srv-softioc1”
   -m "autosave=R5-11, StreamDevice=2.8.26”
   ics-ioc-xxx`
```

Summary

makeBaseApp.pl provides the commonly used IOC skeleton

You need some minimal understanding

- Work in `xxxApp/Db/`, not `db/`
- `make`
- IOC runs `iocBoot/iocXXX/st.cmd`