xspress3epics

Release 3.2

unknown

2023-February-23

CONTENTS

Xspress3Epics provides support for Quantum Detectors Xspress3 series of pulse-counting readout electronics for the EPICS Controls System. The Xspress3 electronics are used with X-ray fluorescence detectors, and provide high perfor-mace in throughput and energy resolution. Both single- and multi-element detector systems are supported. The EPICS driver for Xspress3 is build on top of EPICS areaDetector version 3, and supports all variations of Xspress3, including the "mini" and "X" series.

The driver was originally developed at Diamond Light Source, but has been developed within the epics-modules for many years. Most of the recent development of the driver has been a collaboration between scientists and engineers working at the Advanced Photon Source and Quantum Detectors, and done at xspress3 github page.

Documentation for the low-level API to the Xspress3 system are at *Xspress3 API Documentation*.

CHAPTER

ONE

TABLE OF CONTENTS

1.1 Installation

The Xpsress3 module needs to run on a Linux computer connected to the Xspress3 module. Generally, these have used the Redhat/CentOS Linux distribution, and run when logged into the *xspress3* user account. The instruction methods here may assume that distribution, but it may be possible to build and run the Xspress3 Epics driver on other Linux distribution or other user accounts.

The installation process described here will build a complete and standalone Epics environment under the directory */home/xspress3/epics*. It is certainly possible to install within an existing Epics development environment.

The build_xspress3.py python script is simplest and easiest way to install the Xspress3 driver. Ideally, this driver can be installed with:

```
mkdir /home/xspress3/epics
cd    /home/xspress3/epics
wget  https://raw.githubusercontent.com/epics-modules/xspress3/master/build_xspress3.py
python build_xspress3.py all
```

1.1.1 Required System Packages

The following packages (using the Redhat/Centos names) are required to build and run the Xspress3:

- re2c
- readline, readline-devel
- hdf5-devel
- libtiff-devel
- bzip-devel
- libxml2-devel
- GraphicsMagick-devel

In addition, the *telnet* package is recommended to run the *procServ* process, and *motif-devel* is required if you intend to use the MEDM display screens.

It should be possible to install all of these with:

```
sudo yum install re2c readline-devel hdf5-devel libtiff-devel \
                 bzip-devel libxml2-devel GraphicsMagick-devel \
                 maktelnet motif-devel
```

1.1.2 Required Epics Modules

The following Epics modules are needed to build and run the Xspress3 Input/Output Controller program, with the currently installed versions:

- epics base 7.0.3.1
- asyn 4-38
- seq 2.2.6
- ipac 2.15
- alive R1-1-1
- std R3-6
- autosave R5-10
- iocStats 3.1.16
- busy R1-7-2
- sscan R2-11-3
- calc R3-7-3
- areaDetector R3-9
- adSupport R1-9
- adCore R3-9
- procServ 2.6.0
- medm 3.0.3
- xspress 3-2-6

The build_xspress3.py script will download these versions of these packages from the xspress3 sources web site, hosted at https://millenia.cars.aps.anl.gov/software/xspress3. We expect that newer versions of any of the above packages should also work.

1.1.3 The *build_xspress3.py* script

The build_xspress3.py python script will

1. check the installation of required system packages.
2. download the required Epics modules.
3. configure the require Epics modules, which typically involves writing environmental variables into a file in the `configure` folder for each of the modules.
4. write a master *Build.sh* file.
5. run the master *Build.sh* file to build each module.
6. Download and install

```
#!/usr/bin/env python
#
#  download and build a standalone epics environment for the xspress3
#
```

(continues on next page)

(continued from previous page)

```
#  to use this script:
#    mkdir /home/xspress3/epics
#    cd    /home/xspress3/epics
#    wget  https://raw.githubusercontent.com/epics-modules/xspress3/master/build_
↪xspress3.py
#    python build_xspress3.py all

#  See INSTALL_Standalone.md  for details.
#######

# Note: centos7 defaults with python2, so this needs to work
# with both Py2 and Py3
from __future__ import print_function

import os
import sys
import subprocess as sp
import shutil
import time
from argparse import ArgumentParser

modules = {'base': 'base-7.0.3.1.tar.gz',
           'asyn': 'asyn4-38.tar.gz',
           'sncseq': 'seq-2.2.6.tar.gz',
           'ipac': 'ipac-2.15.tar.gz',
           'alive': 'alive-R1-1-1.tar.gz',
           'std': 'std-R3-6.tar.gz',
           'autosave': 'autosave-R5-10.tar.gz',
           'iocStats': 'iocStats-3.1.16.tar.gz',
           'busy': 'busy-R1-7-2.tar.gz',
           'sscan': 'sscan-R2-11-3.tar.gz',
           'calc': 'calc-R3-7-3.tar.gz',
           'areaDetector': 'areaDetector-R3-9.tar.gz',
           'xspress3': 'xspress3-2-6.tar.gz'}

ad_modules = {'ADSupport': 'adSupport-R1-9.tar.gz',
              'ADCore': 'adCore-R3-9.tar.gz'}

other_sources = ('procServ-2.6.0.tar.gz', 'bin.tar.gz', 'medm_ext.tar.gz')

SOURCES_URL = 'https://millenia.cars.aps.anl.gov/software/xspress3/sources'

LARCH_URL   = 'https://millenia.cars.aps.anl.gov/xraylarch/downloads/'
LARCH_FNAME = 'xraylarch-2022-04-Linux-x86_64.sh'

######################################################################
required_tools = {'re2c': '/bin/re2c', 'rpcgen': '/bin/rpcgen',
                  'readline': '/lib64/libreadline.so.6',
                  'readline-devel': '/usr/lib64/libreadline.so',
                  'hdf5-devel': '/usr/include/hdf5.h',
                  'libtiff-devel': '/usr/include/tiff.h',
                  'bzip-devel': '/usr/include/bzlib.h',
```

(continues on next page)

(continued from previous page)

```
                  'libxml2-devel': '/usr/lib64/libxml2.so',
                  'GraphicsMagick-devel ':  '/usr/lib64/libGraphicsMagick.so'}

recommended_tools = {'telnet': '/bin/telnet',
                     'motif-devel': '/usr/lib64/libXm.so'}

HELP_MESSAGE = '''build_xspress3.py: build Epics Xspress3 module

options:
   -h, --help   shows this message and exit
   -n, --nelem  number of detector elements to configure for [4]

arguments:
    extract     download (if needed) and extract source files only
    configure   configure sources only
    build       compile sources only (must be configured first)
    xrfapp      install and configure XRF display app

    all         extract, configure, and compile sources, install xrfapp
    clean       remove all downloaded source files.
    realclean   completely remove all source files and built components

examples:

   To build everything for 4 element detector:

   > python build_xspress3.py n -4 all

'''

COPY_DMFILES = '''
find . -name "*.adl" | sed \'s|^|cp |g\' | sed \'s|$| adls/.|g\' > tmp_copy_dmfiles
find . -name "*.ui"  | sed \'s|^|cp |g\' | sed \'s|$| uis/.|g\'  >> tmp_copy_dmfiles
find . -name "*.opi" | sed \'s|^|cp |g\' | sed \'s|$| opis/.|g\' >> tmp_copy_dmfiles

sh tmp_copy_dmfiles
rm tmp_copy_dmfiles

'''

HOME_DIR = os.environ.get('HOME', '/home/xspress3')

START_IOC = '''#!/bin/bash
TOPDIR={topdir:s}

IOCNAME=ioc_{nelem:d}Channel
SCRIPT=st.cmd

APPNAME=$TOPDIR/xspress3/iocs/xspress3IOC/bin/linux-x86_64/xspress3App
IOCDIR=$TOPDIR/xspress3/iocs/xspress3IOC/iocBoot/$IOCNAME

source $TOPDIR/bin/bash_profile.sh
```

(continues on next page)

(continued from previous page)

```
cd $IOCDIR
$APPNAME $SCRIPT
'''

RUN_MEDM = '''#!/bin/bash
{topdir:s}/bin/medm -x -macro "P=XSP3_{nelem:d}Chan" {topdir:s}/adls/xspress3_{nelem:d}
↪chan.adl
'''

BUILD_PROCSERV = '''
cd procServ-2.6.0 && sh ./configure && make -j8 && cp procServ ../bin/. && cd ..
'''

BUILD_MEDM = '''
cd extensions/src/medm  && make -j8 && cd ../../.. && cp extensions/bin/linux-x86_64/
↪medm bin/.
'''

AD_RELEASE_LOCAL = '''
ADSUPPORT=$(AREA_DETECTOR)/ADSupport
-include $(TOP)/configure/RELEASE.local.$(EPICS_HOST_ARCH)
'''

AD_RELEASE_LIBS_LOCAL = '''
SUPPORT={topdir:s}
EPICS_BASE={topdir:s}/base
ASYN=$(SUPPORT)/asyn
AREA_DETECTOR=$(SUPPORT)/areaDetector
ADSUPPORT=$(AREA_DETECTOR)/ADSupport
ADCORE=$(AREA_DETECTOR)/ADCore
-include $(AREA_DETECTOR)/configure/RELEASE_LIBS.local.$(EPICS_HOST_ARCH)
'''

AD_RELEASE_PRODS_LOCAL = '''
SUPPORT={topdir:s}
EPICS_BASE={topdir:s}/base
ASYN=$(SUPPORT)/asyn
AREA_DETECTOR=$(SUPPORT)/areaDetector
ADSUPPORT=$(AREA_DETECTOR)/ADSupport
ADCORE=$(AREA_DETECTOR)/ADCore
AUTOSAVE=$(SUPPORT)/autosave
BUSY=$(SUPPORT)/busy
CALC=$(SUPPORT)/calc
SNCSEQ=$(SUPPORT)/sncseq
SSCAN=$(SUPPORT)/sscan
DEVIOCSTATS=$(SUPPORT)/iocStats
-include $(AREA_DETECTOR)/configure/RELEASE_PRODS.local.$(EPICS_HOST_ARCH)
'''

AD_CONFIG_LOCAL = '''
WITH_BOOST     = NO
```

(continues on next page)

(continued from previous page)

```
BOOST_EXTERNAL = YES
WITH_PVA   = YES
WITH_QSRV = YES
WITH_BLOSC      = YES
BLOSC_EXTERNAL = NO
WITH_BITSHUFFLE      = YES
BITSHUFFLE_EXTERNAL = NO
WITH_GRAPHICSMAGICK      = YES
GRAPHICSMAGICK_EXTERNAL = NO
GRAPHICSMAGICK_PREFIX_SYMBOLS = YES
WITH_HDF5      = YES
HDF5_EXTERNAL = NO
WITH_JPEG      = YES
JPEG_EXTERNAL = NO
WITH_NETCDF      = YES
NETCDF_EXTERNAL = NO
WITH_NEXUS      = YES
NEXUS_EXTERNAL = NO
WITH_OPENCV      = NO
OPENCV_EXTERNAL = YES
WITH_SZIP      = YES
SZIP_EXTERNAL = NO
WITH_TIFF      = YES
TIFF_EXTERNAL = NO
XML2_EXTERNAL = NO
WITH_ZLIB      = YES
ZLIB_EXTERNAL = NO
ARAVIS_INCLUDE  = /usr/local/include/aravis-0.8
GLIB_INCLUDE = /usr/include/glib-2.0 /usr/lib64/glib-2.0/include
glib-2.0_DIR = /usr/lib64
'''

PROFILE = '''#!/bin/bash
# startup script to use epics environment for xspress3

export EPICS_HOST_ARCH=linux-x86_64
export EPICS_CA_MAX_ARRAY_BYTES=768000

export EPICS_BASE='{topdir:s}/base'
export EPICS_EXTENSIONS='{topdir:s}/extensions'
export EPICS_DISPLAY_PATH='{topdir:s}/adls'

# turn this off to make sure PVs can be seen by other
# machines on your network
unset  EPICS_CAS_INTF_ADDR_LIST

# setting EPICS Address list, there are 2 options:
# a) use automatic address lookup.
#    works well for  machines with 1 NIC (xpsress3 minis?)
export EPICS_CA_AUTO_ADDR_LIST=YES
unset  EPICS_CA_ADDR_LIST
```

(continues on next page)

(continued from previous page)

```
# b) turn off automatic address lookup, explicitly set address
#    list to the broadcast address (or localhost).  This gives
#    fewer warning messages for machines with multiple NICs,
#    such as non-mini xspress3 units.
export EPICS_CA_AUTO_ADDR_LIST=NO
# export EPICS_CA_ADDR_LIST=localhost
export EPICS_CA_ADDR_LIST={broadcast:s}

# include bin folders in PATH
PATH=/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin:/usr/lib64/qt-3.3/bin
export PATH=$PATH:$EPICS_BASE/bin/linux-x86_64:$EPICS_EXTENSIONS/bin/linux-x86_64:
↪{topdir:s}/bin

# include bin folders in LD_LIBRARY_PATH
LD_LIBRARY_PATH={homedir:s}/software/boost/stage/lib:/usr/local/lib
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$EPICS_BASE/lib/linux-x86_64:$EPICS_EXTENSIONS/
↪lib/linux-x86_64
'''


XRFAPP_SCRIPT = '''#!{homedir:s}/xraylarch/bin/python
# Epics XRF Display App
from larch.epics import EpicsXRFApp

# Note: 'det_type' determines the layout of the buttons for the
#       individual detector elements. Known layouts include:
#      ME-4    Hitachi ME-4
#      ME-7    Hitachi ME-7
#      SXD-7   Canberra SXD-7
#      other   all other detector type  (display 4 per row)

app = EpicsXRFApp(nmca={nelem:d},
                  det_type='other',
                  environ_file='{topdir:s}/bin/Xspress3.env',
                  prefix='XSP3_{nelem:d}Chan:',
                  ioc_type='Xspress3')
app.MainLoop()
'''

def remove_link_or_dir(name):
    if os.path.islink(name):
        os.unlink(name)
    elif os.path.isdir(name):
        shutil.rmtree(name)

def write_bash_profile(nelem=4):
    # grep /sbin/ifconfig output for broadcast
    broadcast = ''
    for line in sp.check_output(['/usr/sbin/ifconfig']).splitlines():
        if (sys.version_info.major == 3):
            line = line.decode('utf-8')
        if 'inet' in line and 'broadcast' in line:
```

(continues on next page)

(continued from previous page)

```
                words = [i.strip() for i in line.split('broadcast')]
                broadcast = words[-1]
                break
    with open ('bin/bash_profile.sh', 'w') as fh:
        fh.write(PROFILE.format(topdir=os.getcwd(),
                                homedir=HOME_DIR, broadcast=broadcast))
    with open ('bin/run_xspress3.sh', 'w') as fh:
        fh.write(START_IOC.format(topdir=os.getcwd(), nelem=nelem))
    with open ('bin/run_medm.sh', 'w') as fh:
        fh.write(RUN_MEDM.format(topdir=os.getcwd(), nelem=nelem))
    os.chmod('bin/bash_profile.sh', 493) # mod 755
    os.chmod('bin/run_xspress3.sh', 493) # mod 755
    os.chmod('bin/run_medm.sh', 493) # mod 755
    print('# wrote bash_profile.sh')

def create_bindir():
    tarball = 'bin.tar.gz'
    tball = os.path.abspath(os.path.join('sources', tarball))
    cwd = os.path.abspath(os.getcwd())
    if not os.path.exists(tball):
        os.chdir('sources')
        print("# fetch %s" % ('%s/%s' % (SOURCES_URL, tarball)))
        o = sp.call(['/bin/wget', '%s/%s' % (SOURCES_URL, tarball)])
        os.chdir(cwd)
    if not os.path.exists('bin'):
        os.mkdir('bin')
    if not os.path.exists(tball):
        print("# download failed! for %s" % tball)
    else:
        tarcmd = ['tar', 'xvzf', tball]
        o = sp.check_output(tarcmd).splitlines()

def unpack_tarball(tarball, shortname=None, sourcedir='sources'):
    tball = os.path.join(sourcedir, tarball)
    cwd = os.path.abspath(os.getcwd())
    if not os.path.exists(sourcedir):
        os.mkdir(sourcedir)
    if not os.path.exists(tball):
        os.chdir(sourcedir)
        print("# fetch %s" % ('%s/%s' % (SOURCES_URL, tarball)))
        o = sp.call(['/bin/wget', '%s/%s' % (SOURCES_URL, tarball)])
        os.chdir(cwd)
    if not os.path.exists(tball):
        print("# download failed! for %s" % tball)
    else:
        tarcmd = ['tar', 'xvzf', tball]
        o = sp.check_output(tarcmd).splitlines()
        first_line = o[0]
        if (sys.version_info.major == 3):
            first_line = first_line.decode('utf-8')
        outdir = first_line.split('/')[0]
        if shortname is not None:
```

(continues on next page)

(continued from previous page)

```
            if os.path.exists(shortname):
                remove_link_or_dir(shortname)
            os.symlink(outdir, shortname)
        print("# downloaded and unpacked source for %s" % outdir)

def extract_sources():
    print('# extracting sources')
    for key, value in modules.items():
        unpack_tarball(value, shortname=key)

    os.chdir('areaDetector')
    for key, value  in ad_modules.items():
        unpack_tarball(value, shortname=key, sourcedir=os.path.join('..', 'sources'))
    os.chdir('..')

    for tarball in other_sources:
        unpack_tarball(tarball)

    create_bindir()


def configure_release():
    subs = {}
    TOP = os.getcwd()
    subs['EPICS_BASE'] = '%s/base' % TOP
    subs['SUPPORT']  = TOP
    epics_mods = list(modules.keys())
    for mod in epics_mods:
        subs[mod.upper()] = '%s/%s'  % (TOP, mod)

    for mod in epics_mods:
        release = os.path.join(TOP, mod, 'configure', 'RELEASE')
        with open(release, 'r') as fh:
            text = fh.readlines()
        savefile = release + '.backup'
        if not os.path.exists(savefile):
            shutil.copy(release, savefile)
        out = []
        for line in text:
            line = line[:-1]
            for key, val in subs.items():
                if line.startswith(key):
                    line = '%s=%s' % (key, val)
            out.append(line)
        out.append('')
        with open(release, 'w') as fh:
            fh.write('\n'.join(out))
    # fix asyn rpc.h include if needed
    if not os.path.exists('/usr/include/rpc/rpc.h'):
        asyn_conffile = os.path.join(TOP, 'asyn', 'configure', 'CONFIG_SITE')
        shutil.copy(asyn_conffile, asyn_conffile + '.backup')
        buff = []
```

(continues on next page)

(continued from previous page)

```
        for t in open(asyn_conffile, 'r').readlines():
            t = t[:-1]
            if 'TIRPC' in t:
                t = 'TIRPC=YES'
            buff.append(t)
        buff.append('')
        with open(asyn_conffile, 'w') as fh:
            fh.write('\n'.join(buff))

    # write areaDetector configs
    ad_configs = {'RELEASE.local': AD_RELEASE_LOCAL,
                  'RELEASE_LIBS.local': AD_RELEASE_LIBS_LOCAL,
                  'RELEASE_PRODS.local': AD_RELEASE_PRODS_LOCAL,
                  'CONFIG_SITE.local': AD_CONFIG_LOCAL}
    for fname, config in ad_configs.items():
        fullname = os.path.join(TOP, 'areaDetector', 'configure', fname)
        with open(fullname, 'w') as fh:
            fh.write(config.format(topdir=TOP))


def create_buildscript(nelem=4):
    write_bash_profile(nelem=nelem)
    cwd = os.path.abspath(os.getcwd())
    script = ['#!/bin/bash',
              '# build epics sources script',
              'source ./bin/bash_profile.sh',
              'ROOT=`pwd`']

    bline = 'cd {dirname:s} && make -j8 install && cd ..'
    build_mods = ('base', 'sncseq', 'asyn', 'ipac', 'alive', 'std',
                  'autosave', 'iocStats', 'busy', 'sscan', 'calc',
                  'areaDetector', 'xspress3')

    for mod in build_mods:
        dirname = os.path.join(cwd, mod)
        script.append(bline.format(dirname=mod))

    script.append('# build medm')
    script.append(BUILD_MEDM)
    script.append('# build procServ utility')
    script.append(BUILD_PROCSERV)
    script.append('# copy display files')
    script.append('mkdir -p adls uis opis')
    script.append(COPY_DMFILES)

    script.append('')
    with open('do_build.sh', 'w') as fh:
        fh.write('\n'.join(script))
    time.sleep(1)
    print("# wrote do_build.sh. Now ready to build with 'sh do_build.sh'")

def install_xrfapp(nelem=4):
```

(continues on next page)

(continued from previous page)

```
    cwd = os.path.abspath(os.getcwd())
    if not os.path.exists(os.path.join(HOME_DIR, 'xraylarch')):
        os.chdir('sources')
        o = sp.call(['/bin/wget', '%s/%s' % (LARCH_URL, LARCH_FNAME)])
        o = sp.call(['sh', './%s' % LARCH_FNAME, '-b'])
        os.chdir(cwd)

    with open('bin/run_xrfcontrol.py', 'w') as fh:
        fh.write(XRFAPP_SCRIPT.format(topdir=cwd, homedir=HOME_DIR,
                                      nelem=nelem))
    os.chmod('bin/run_xrfcontrol.py', 493) # mod 755
    # environ file
    buff = []
    template = 'XSP3_{nelem:d}Chan:C{elem:d}SCA:{sca:d}:Value_RBV Chan{elem:d} {label:s}'
    for sca, label in enumerate(('ClockTicks', 'ResetTicks', 'ResetCounts',
                                 'AllEvents', 'AllGood', 'Window1', 'Window2',
                                 'Pileup', 'EventWidth', 'DeadTimeFactor',
                                 'DeadTimePercent')):
        for elem in range(1, nelem+1):
            buff.append(template.format(nelem=nelem, elem=elem, sca=sca, label=label))

    buff.append('')
    with open('bin/Xspress3.env', 'w') as fh:
        fh.write('\n'.join(buff))

def check_dependencies():
    missing_reqs = []
    for name, exe in required_tools.items():
        if not os.path.exists(exe):
            missing_reqs.append(name)
    if len(missing_reqs) > 0:
        print("There are missing dependencies. Install with `sudo yum install`:")
        print("  ".join(missing_reqs))
        sys.exit(0)

    missing_tools = []
    for name, exe in recommended_tools.items():
        if not os.path.exists(exe):
            missing_tools.append(name)
    return missing_tools

def build(nelem=4):
    missing_tools = check_dependencies()
    if not os.path.exists('do_build.sh'):
        create_buildscript(nelem=nelem)
    o = sp.call(['sh', 'do_build.sh'])

    if len(missing_tools) > 0:
        print("There are missing useful tools. Install with `sudo yum install`:")
        print("  ".join(missing_tools))
```

(continues on next page)

(continued from previous page)

```
def clean():
    remove_link_or_dir('sources')

def realclean():
    clean()
    if os.path.exists('do_build.sh'):
        os.unlink('do_build.sh')
    for slink, tarball in modules.items():
        remove_link_or_dir(slink)
        sdir = tarball.replace('.tar.gz', '')
        remove_link_or_dir(sdir)

    for dname in ('do_build.sh', 'adls', 'uis', 'opis', 'procServ',
                  'procServ-2.6.0', 'extensions'):
        remove_link_or_dir(dname)

if __name__ == '__main__':
    parser = ArgumentParser(prog='xspress3_build',
                            description='build xspress3 epics support',
                            add_help=False)
    parser.add_argument('-h', '--help', dest='help', default=False)
    parser.add_argument('-n', '--nelem', dest='nelem', type=int, default=4)
    parser.add_argument('options', nargs='*')

    args = parser.parse_args()
    nelem = args.nelem
    if args.help or len(args.options) == 0:
        print(HELP_MESSAGE)
    else:
        cmd = args.options.pop(0)
        if cmd == 'extract':
            extract_sources()
        elif cmd == 'configure':
            create_buildscript(nelem=nelem)
            configure_release()
        elif cmd == 'xrfapp':
            install_xrfapp(nelem=nelem)
        elif cmd == 'clean':
            clean()
        elif cmd == 'realclean':
            realclean()
        elif cmd == 'build':
            build()
        elif cmd == 'all':
            extract_sources()
            create_buildscript(nelem=nelem)
            configure_release()
            build()
            install_xrfapp(nelem=nelem)
```

1.2 Release Notes

1.3 Setting up and Running an Epics Input/Output Controller

1.4 Running the Epics IOC

The above installation will build all the epics modules, including the Xspress3 application in the current folder. It will also create a ‘bin’ directory in your /home/xspress3/epics’ folder that contains scripts to set your environment and run the Xspress3 IOC:

> bin/bash_profile.sh bash script to set environmental variables bin/procServ useful utility for long-running tasks like an IOC bin/medm simple, minimal Epics display manager bin/run_xspress3.sh bash script to run an XSPRESS3 IOC bin/start_ioc python script to run the Xspress3 using procServ bin/run_xrfcontrol.py python script to view and control the Xpsress3 bin/run_medm.sh bash script to launch medm for your Xspress3

After building you can configure an Xspress3 IOC, starting with one of the defaults in the

> /home/xspress3/epics/xspress3/iocs/xspress3IOC/iocBoot

directory. Note that if you change the prefix or the number of detector elements used, you may need to edit several of the files in the bin/ directory to match your configuration.

Once properly configured, you should be able to run your xspress3 in a long-running procServ process with

> /home/xspress3/epics/bin/start_ioc xspress3

and then view screens for your detector either using an Epics display manager like medm, caqtdm, or css/boy. Display screens for these display managers can be found in the folders

> /home/xspress3/epics/adls screen files for medm /home/xspress3/epics/uis screen files for caqtm /home/xspress3/epics/opis screen files for css/boy

For example, you will be able to run MEDM for your Xspress3 with

> /home/xspress3/epics/bin/run_medm.sh

You will also be able to run a dedicated XRF Control application with

> /home/xspress3/epics/bin/run_xrfcontrol.py

1.5 Xspress3 Display Screens

Because the Xspress3 driver provides a complex set of Epics data structures, there are a fairly large number of simple display screens for interacting with Epics variables to control these electronics. As is customary for Epics modules, these screens were created as `adl` files for the MEDM tool, and have been translated to `ui` screens for caQtDM.

1.6 Counting Modes for the Xspress3

1.7 Installing and using the Optional XRF Display Application