Argonne NATIONAL LABORATORY

EPICS Programming

Andrew Johnson, AES-SSG

Outline

- Why program on top of EPICS
- Build system features
  - Assume a basic understanding of Unix make
- Facilities available in libCom

Why program on top of EPICS?

- Community standard
  - EPICS collaborators know and understand the EPICS layout
- Code portability across many Operating Systems
  - Making C & C++ code portable is not always easy
  - EPICS APIs work the same on all targets
  - Support for Linux, Mac, Windows (MS, Cygwin, MinGW), Solaris, VxWorks, RTEMS etc.
- Build portability across Operating Systems
  - Compiling code portably is not trivial
  - EPICS Makefiles work the same on all hosts
  - Support for Linux, Mac, Windows (MS, Cygwin, MinGW), Solaris

EPICS Build System

- Advanced set of build rules for Makefiles in source tree
  - Requires GNU Make version 3.81 or later
  - Never seen a similar set of build rules
    - Autotools, imake, Premake and Qmake all generate Makefiles
    - Cmake generates Makefiles or Visual Studio project files
- Designed for multiple target architecture builds
  - Other build systems don't integrate that functionality
- Build rules expect a specific application layout
  - <top>/configure directory
    - RELEASE file(s)
    - Several CONFIG* and RULES* files
  - Makefiles must have specific content
    - Set TOP variable
    - Include specific CONFIG and RULES files

Makefiles

- Four different kinds, related to the role
  - Top-level
    - For descending into subdirectories (supports extra build targets)
  - Structural
    - For descending into subdirectories
  - Constructional
    - For building software
  - Startup
    - For iocBoot/ioc directories
- Differences
  - Which configure/RULES* file they include
  - Which variables those rules examine to control what they do

Top-level Makefile

- <top>/Makefile

```
TOP = .
include $(TOP)/configure/CONFIG
DIRS = list of subdirectories
    Also set *_DEPEND_DIRS here
include $(TOP)/configure/RULES_TOP
```

- Add any extra target rules after the 2nd include line
- DIRS variable lists all subdirectories to recursively build in
- *_DEPEND_DIRS variables control the build order of subdirectories
  - Example:

    ```
    test_DEPEND_DIRS = configure src
    ```

  - The test subdirectory will be built after the configure and src directories
  - Setting *_DEPEND_DIRS variables is important for parallel builds
    - Running ‘make -j’ will build in all subdirectories simultaneously otherwise

Structural Makefiles

- Makefiles for descending into subdirectories only

```
TOP = ..                    Adjust path as appropriate
include $(TOP)/configure/CONFIG
DIRS = list of subdirectories
    Also set *_DEPEND_DIRS here
include $(TOP)/configure/RULES_DIRS
```

- Very similar to top-level Makefile
  - Set DIRS and *_DEPEND_DIRS variables as before
  - Includes RULES_DIRS instead of RULES_TOP

- Examples:
  - <top>/exampleApp
  - <top>/iocBoot

Startup Makefiles

- Makefiles for iocBoot/ioc directories

```
TOP = ../..
include $(TOP)/configure/CONFIG
ARCH = ioc target architecture
TARGETS = additional files to build
include $(TOP)/configure/RULES.ioc
```

- The ARCH setting controls the content of the generated TARGETS
- Known TARGETS are
  - cdCommands Intended for VxWorks only
  - envPaths For other Operating Systems
  - dllPath.bat For Windows architectures
  - relPaths.sh For Cygwin
- Rules for extra targets can be added after the 2nd include line

Constructional Makefiles

- Makefiles for compiling software

```
TOP = ../..          Adjust path as appropriate
include $(TOP)/configure/CONFIG
    Set variables here
include $(TOP)/configure/RULES
    Add extra rules and dependencies here
```

- Constructional Makefiles must be named ‘Makefile’, never ‘GNUmakefile’ or ‘makefile’

- Many variables are available to control what gets built
  - See Chapter 4 of the Application Developers’ Guide for a full list

What gets built and/or installed

- Controlled by a set of variables naming the final products
  - INC — C/C++ Header files (.h)
  - LIBRARY — Static or shared object libraries (lib.a lib.so .dll)
  - LOADABLE_LIBRARY — Shared object libraries (lib.so .dll lib.dylib)
  - PROD — Executable programs (.exe)
  - TESTPROD — Executable programs, not installed
  - OBJS — Object files (.o)
  - SCRIPTS — Interpreted scripts
  - DBD — Database definition files (.dbd)
  - DBDINC — Record type and menu database definition files
  - DB — Database instance files (.db .vdb)
  - TARGETS — Other build targets, may need build rule
    - In many cases the name you use should not include the prefix/suffix
- Named objects are copied into the appropriate install directory as they get built, e.g. <top>/include, <top>/lib/<arch>, <top>/bin/<arch>, <top>/dbd, <top>/db

Limiting builds

- Limit build target to (all) host architectures by using these variables:
  - PROD_HOST, TESTPROD_HOST, LIBRARY_HOST, LOADABLE_LIBRARY_HOST, OBJS_HOST, SCRIPTS_HOST
- Limit build to (all) IOC architectures by using these variables:
  - PROD_IOC, TESTPROD_IOC, LIBRARY_IOC, OBJS_IOC, SCRIPTS_IOC
- Limit build to OS-specific architectures by using these variables:
  - PROD_<osclass>, TESTPROD_<osclass>, LIBRARY_<osclass>, LOADABLE_LIBRARY_<osclass>
  - <osclass> may be Linux, vxWorks, WIN32, Darwin, RTEMS or solaris
  - Example, build library only for embedded targets:

```
LIBRARY_vxWorks = myDev
LIBRARY_RTEMS = myDev
```

Naming Source Files

- If a Makefile only creates one target (library, executable etc), you can add the names of all source files to the SRCS variable:

```
SRCS = myDev.c myDrv.c
```

- If a Makefile only creates one library, you can add the names of all library source files to the LIB_SRCS variable:

```
LIBRARY = myDev
LIB_SRCS += myDev.c myDrv.c
```

- If a Makefile only creates one executable (PROD), you can add the names of all its sources to the PROD_SRCS variable:

```
PROD = myIoc
PROD_SRCS += myMain.c mySeq.st
```

- However it's usually best to use the <name>_SRCS variable:

```
LIBRARY = myLib
myLib_SRCS = parser.c scanner.cpp process.cpp
PROD = myTool
myTool_SRCS += tool.c
```

OS-Specific Source Files

- You can append _<osclass> to the source variable names to limit which OS the code gets built on
  - SRCS_<osclass>
  - LIB_SRCS_<osclass>
  - PROD_SRCS_<osclass>
  - <name>_SRCS_<osclass>

- When setting _<osclass> variables the relevant _DEFAULT variable is used for all OS's that don't have an _<osclass> version
- Example:

```
LIBRARY = myDev
LIB_SRCS = myDev.c
LIB_SRCS_vxWorks = devVx.c
LIB_SRCS_RTEMS   = devRtems.c
LIB_SRCS_DEFAULT = devPosix.c   # Linux, Darwin, Solaris
LIB_SRCS_WIN32   = -nil-
```

Source File Locations

- Normally source files appear in the same directory as the Makefile
- Make can be told to search nearby directories for source files
  - SRC_DIRS += <dir>
    - Where <dir> is the relative path from the O.<arch> build directory to the directory containing the source files
- Multiple OS-specific implementations of code can also be used
- Place source files in one or more of these subdirectories
  - os/<osclass> OS-specific versions
  - os/posix Posix-based OS's (Linux, Unix, Darwin, RTEMS)
  - os/default Last-chance generic version
- The same source filename should be used for all versions

C & C++ Compiler Flags

- Many ways to add flags to the compiler command-line, e.g.

USR_CFLAGS	All C compiles
USR_CXXFLAGS	All C++ compiles
USR_CPPFLAGS	C Preprocessor flags
USR_CFLAGS_<osclass>	All C compiles for <osclass>
USR_CXXFLAGS_<osclass>	All C++ compiles for <osclass>
USR_CFLAGS_<arch>	All C compiles for <arch>
USR_CXXFLAGS_<arch>	All C++ compiles for <arch>
<name>_CFLAGS	Compiling <name>.c
<name>_CFLAGS_<osclass>	Compiling <name>.c for <osclass>
<name>_CFLAGS_<arch>	Compiling <name>.c for <arch>

- Include file search directories have their own variables
  - USR_INCLUDES, USR_INCLUDES_<osclass>, <name>_INCLUDES, <name>_INCLUDES_<osclass>, <name>_INCLUDES_<arch>
  - A '`-I`' flag is required before each directory named in the INCLUDES

Linking with Libraries

- When building an executable, you specify the list of application libraries to be linked with in a LIBS variable
- Leave off any 'lib' prefix and '.a', '.so' or '.dll' suffix in library names
- If all executables built by a Makefile need a common set of libraries, name them in the PROD_LIBS variable:

```
PROD_LIBS = ca Com
```

- All libraries are linked against the list in the LIB_LIBS variable
- Named products and libraries are linked against a list or libraries named in the <name>_LIBS variable:

```
myTool_LIBS = myLib ca Com
```

- System libraries must be listed in similar SYS_LIBS variables
  - PROD_SYS_LIBS, PROD_SYS_LIBS_<osclass>, PROD_SYS_LIBS_DEFAULT, LIB_SYS_LIBS, LIB_SYS_LIBS_<osclass>, LIB_SYS_LIBS_DEFAULT, <name>_SYS_LIBS, <name>_SYS_LIBS_<osclass>, <name>_SYS_LIBS_DEFAULT

Library Locations

- Libraries provided by other EPICS modules that are listed in the configure/RELEASE file will normally be found automatically
  - The build system automatically searches those lib/<arch> directories as well as the <top>/lib/<arch> directory
- If a library is located elsewhere, the Makefile must specify where
  - Set the variable <name>_DIR to the absolute path of the library
  - For example:

```
LIBS += usb
usb_DIR = /opt/local/lib
```

- If it's from a non-EPICS package, use <top>/configure/CONFIG_SITE to set the path to that package
  - Don't make users have to edit Makefiles to be able to build the code

Conditionals in Makefiles

- Use GNU Makefile conditionals to adjust the build
  - The configure/CONFIG file includes the <top>/configure/RELEASE and <top>/configure/CONFIG_SITE file(s)
  - Use configure/RELEASE variables if build depends on whether optional modules are available or not

```
ifdef SNCSEQ
    Lines for builds with sequencer here
else
    Lines for builds without sequencer here
endif
```

  - Use variables in CONFIG_SITE to let user enable/disable features

```
ifeq ($(BUILD_IOCS),YES)
    Lines for building IOCs here
else
    Lines for building without IOCs here
endif
```

libCom — General Purpose Facilities Library

- The library has 2 main purposes
  - Provide a common Operating System API across all supported OS's
  - Implement additional general purpose facilities for use by the IOC, Channel Access, and other programs

- base/src/libCom contains 159 C/C++ header files (3.14.12.5)
- Don't have time to cover or even mention all of them here
- The main facilities are discussed in these IOC Application Developers' Guide sections:
  - 10. IOC Error Logging
  - 16.3 Task Watchdog
  - 18. IOC Shell
  - 19. libCom
  - 20. libCom OSI libraries
  - 21. Registry

libCom Highlights for C code

- Multi-threading and inter-thread communication
  - epicsThread, epicsMutex, epicsEvent
  - epicsRingBytes & epicsRingPointer, epicsMessageQueue
- Process communication and string conversions
  - epicsStdio, epicsStdlib, epicsString
  - osiSock
  - errlog & logClient
  - macLib
- Data types and structures
  - epicsTypes, ellLib, gpHash
- Mathematics
  - Calc engine, epicsMath, epicsEndian
- Shared libraries
  - shareLib.h and epicsExport.h

Multi-threading

- epicsThread.h provides a generic threading API
  - Thread creation (name, priority, stack size, function, argument)
    - If supported, OS thread priorities mapped to range low=0 .. high=99
    - Stack sizes are OS & architecture dependent: Small, Medium, Large
  - Thread operations supported:
    - Sleep (time delay), suspend, resume, get name, get id, sleep quantum, show
    - No API to remotely kill a thread, routine must return for thread to exit
  - C++ wrapper class
- Thread private variables
  - Variable operations
    - Create, destroy, get, set
- Thread once API
  - Guarantees execution of initialization function only once
    - Parallel attempts to execute the initialization function by other threads will delay them until the function has returned within the first thread

Mutual Exclusion and Event Signaling

- epicsMutex.h
  - Mutual exclusion semaphore
  - Supports recursive locking
  - Priority inheritance and deletion safety where available from OS
  - Mutex operations:
    - Create, destroy, lock, unlock, try-lock, show
  - C++ wrapper class
- 3.15: epicsSpin.h
  - Spin-lock semaphore, based on epicsMutex C API
- epicsEvent.h
  - Binary semaphore
  - Event operations:
    - Create, destroy, signal, wait, try-wait, wait with timeout, show
  - C++ wrapper class

Circular Message Buffer

- epicsRingBytes.h
  - Fixed size circular buffer
  - Supports variable length messages
  - Caller must implement locking if needed
    - If single writer thread, no locking is needed on put
    - If single reader thread, no locking is needed on get
    - Base 3.15 provides an optional internal spin-lock
  - Buffer operations:
    - Create, delete, put, get, flush
    - 3.15: Create-locked
  - Status queries
    - Size, is full, is empty, used bytes, free bytes

Circular Buffer for Pointers

- epicsRingPointer.h
  - Fixed size circular buffer
  - Supports single pointer messages only
  - Caller must implement locking if needed
    - If single writer thread, no locking is needed on put
    - If single reader thread, no locking is needed on get
    - Base 3.15 provides an optional internal spin-lock
  - C++ wrapper class
  - Buffer operations:
    - Create, delete, push, pop, flush
    - 3.15: Create-locked
  - Status queries
    - Size, is full, is empty, used bytes, free bytes

Message Queue

- epicsMessageQueue.h
  - Fixed size queue
  - Supports variable length messages
  - Designed for use with multiple reader and writer threads
  - C++ wrapper class
  - Queue operations:
    - Create, destroy, send, try-send, send with timeout, receive, try-receive, receive with timeout
  - Status queries:
    - Pending, show

Wrapper for <stdio.h>

- epicsStdio.h (includes stdio.h)
  - epicsSnprintf() & epicsVsnprintf()
    - Implementation or wrapper for C99's snprintf() & vsnprintf() functions
    - Ensure all operating systems behave almost the same
  - Infrastructure for redirecting stdin, stdout, stderr streams
    - Per-thread settings for each stream (mainly for iocsh)
      - epicsGetThreadStdin(), epicsSetThreadStdin(), etc.
    - Include epicsStdioRedirect.h to redefine the identifiers stdin, stdout & stderr and the functions printf(), puts() & putchar()
      - In 3.15 this header merged into epicsStdio.h
  - Miscellaneous file and filename functions
    - Recommend not using these old APIs

Standard library

- epicsStdlib.h (includes stdlib.h)
  - epicsStrtod()
    - Synonym or wrapper for strtod()
    - Ensure all operating systems behave the same, support NaN and Inf strings
  - epicsScanDouble(), epicsScanFloat()
    - sscanf("%f") and sscanf("%lf") guaranteed to support NaN and Inf strings
  - 3.15: Series of epicsParse functions
    - For converting strings to all numeric types
    - Optional units string capture
    - Error checks include value overflow and underflow
    - All functions return a status value (error code)

String Processing

- epicsString.h
  - A compilation of useful string functions:
    - Convert string from raw to C-style escaped
    - Convert string from C-style escaped to raw
    - Print string with unprintable characters escaped
    - Shell glob pattern matching (does this string match that wildcard pattern?)
    - Calculate hash value of strings and memory buffers
  - These replace standard routines that are not available on all operating systems:
    - Case-independent string comparisons (strcasecmp, strncasecmp)
    - Re-entrant string tokenization (strtok_r)
    - String duplication (strdup)

Macro Substitutions

- macLib.h
  - General purpose macro substitution library
  - Supports multiple variable scopes, recursive macros, …
  - Also handles environment variables
  - Operations:
    - Create context, enable/disable warnings, delete context, get macro value, set macro value, push scope, pop scope, parse macro definitions, install parsed definitions, expand string, expand string with environment variables, report context
  - Efficient, reliable, well-tested
  - See macLibREADME file in base/src/libCom/macLib for more details

Network Sockets API

- osiSock.h
  - Provides a unified API for creating and using network sockets
    - No special application code needed for Windows, Solaris etc.
  - Used by Base (CA client and servers, log client & server etc.), Asyn, pvAccess
  - Extremely widely used, reliable
  - Several routines provided for common tasks
    - Query available network interfaces
    - Create socket, bind to address, listen for connections, ioctl, destroy
    - Configure socket for broadcast UDP
    - Convert socket or IP address to/from ASCII (DNS and numeric)
    - How to unfreeze a thread that is blocked reading from a socket
    - Look up socket error message strings

Reporting and Logging Errors

- errMdef.h, errlog.h
  - Provide APIs for various purposes related to error handling and logging
    - Associate and look up strings with error status value
    - Standard for error number prefixes
    - Log and flush error messages
    - Listeners to forward logged errors to a remote server
    - Enable/disable display of logged messages on console
- locLogServer
  - Server application for IOCs and applications to log error messages to
  - Stores messages in a circular file (configurable fixed maximum size)
  - Log file directory rotation also supported
- Details in Chapter 10 of the IOC Application Developers' Guide

Standard Types

- epicsTypes.h
  - EPICS-standard type definitions of various sizes
  - Pre-dates C99, as do some of our supported operating systems
  - Defines these standard types
    - epicsInt8, epicsUInt8, epicsInt16, epicsUInt16, epicsInt32, epicsUInt32, epicsFloat32, epicsFloat64, epicsEnum16
    - 3.15: epicsInt64, epicsUInt64
  - Unfortunately epicsInt8 is always ‘char’, so may be unsigned on some architectures
  - Also defines
    - MAX_STRING_SIZE (40)
    - epicsFalse (0)
    - epicsTrue (1)
    - stringOf(token) – stringifies token

Linked Lists

- ellLib.h
  - Doubly-linked list management routines
  - Intrusive – listed objects must include an ELLNODE, no extra memory needed
  - Operations:
    - Initialize list, object count, first object, last object, next object, previous object, add object, concatenate lists, delete object, extract objects, get first object, insert object, get *n*th object, step *n* objects, find object in list, free all objects, validate list
  - Lists and nodes may be statically initialized
  - Efficient, reliable, well-tested
  - API modeled on VxWorks lstLib

Hash Tables

- gpHash.h
  - General-purpose hash table for fast object lookup by name
  - Number of buckets fixed at initialization time
    - Powers of 2 from 256 to 65536
  - Can store multiple object types in one hash table
    - Pass in a type ID (pointer) to distinguish, included in hash calculation
  - Non-obtrusive, table allocates node objects
  - Thread-safe access, table contains a mutex
  - Operations:
    - Create table, add named object, find by name, delete by name, free table, dump contents

Calc Engine

- postfix.h
  - Expression compiler and evaluation engine
  - Compiles math expressions into a private postfix byte-code format
    - Most standard C operations supported, minor differences in syntax
  - Fast execution of the byte-code with a given set of input values (double)
  - Used by calc & calcout records in Base, transform record, areaDetector plugin
  - Routine to examine input variables used and modified by compiled expression
  - Operations:
    - Compile expression, perform calculation, argument usage, dump byte-code
  - Efficient, reliable, well-tested
  - Detailed documentation in IOC Applications Developers' Guide

Other Headers

- epicsMath.h
  - Includes math.h
  - Defines epicsINF and epicsNAN
  - Ensures finite(), isnan() and isinf() are all defined
- epicsEndian.h
  - Defines 4 numeric macros
    - EPICS_ENDIAN_LITTLE
    - EPICS_ENDIAN_BIG
    - EPICS_BYTE_ORDER
    - EPICS_FLOAT_WORD_ORDER
  - The two _ORDER macros are architecture-specific and should be compared with the first two to determine CPU endianness

Library Exports & Imports

- shareLib.h
  - Defines several macros for marking library symbols, essential for Windows DLLs
    - epicsShareFunc – function to be exported/imported
    - epicsShareClass – class to be exported/imported
    - epicsShareExtern – 'extern' variable declaration
    - epicsShareDef – variable definition
    - epicsShareAPI – function uses __stdcall calling convention on Windows
  - The definitions vary depending whether the macro epicsExportSharedSymbols is defined, and whether the compiler is building a DLL or a static (archive) library

- epicsExport.h
  - Defines epicsExportSharedSymbols, then includes shareLib.h
  - Defines a few other macros for IOC registration
    - epicsExportAddress, epicsExportRegistrar(), epicsRegisterFunction()

Using shareLib.h Properly

- Library header files **should**
  - include shareLib.h and any other header files needed by the declarations in this header, then apply the appropriate epicsShare keywords to decorate the header's declarations
- Library implementations **should**
  1. include all needed headers for code found *outside* the library that this code will be part of. External headers included by the module header file **must** also be included here
  2. #define epicsExportSharedSymbols
  3. include all needed headers for code found *inside* the library this code will be part of
- Implementations **may** include epicsExport.h instead of defining the macro epicsExportSharedSymbols, but it is harder to make it obvious that this should be a hard dividing line that include files should not cross during subsequent edits

Unit Testing

- epicsUnitTest.h
  - Unit test reporting library
  - Generates Test Anything Protocol (TAP) standard output
  - Works with the build system ‘runtests’ and ‘tapfiles’ targets on workstations
  - Built-in test harness functionality when running on embedded operating systems
  - Operations:
    - Plan, Ok, pass, fail, skip, todo, diagnostic, abort, done
- testMain.h
  - Defines a macro MAIN() allowing tests to be built as programs on Workstations and functions on embedded operating systems
- Build system variables TESTPROD, TESTSCRIPTS (3.15: TESTLIBRARY) build programs without installing them; test programs are usually run in their O.<arch> directory
- Add test program names to TESTS to be run by ‘make runtests’ etc.