PAUL SCHERRER INSTITUT

PSI

SLS

Loadable driver modules

Building drivers for multiple releases of EPICS

Loading drivers dynamically from startup script

What's the problem?

- Some facilities use more than one EPICS release
  - Historical reasons ("Never touch a running system.")
  - Third party code (no sources available)
- Drivers should be built for all EPICS releases in parallel
  - Custom record types and other code as well
  - The same (latest/best) driver version should be available
- But EPICS build facility is related to one EPICS release
  - $(TOP)/config/RELEASE file (in R3.13)
- Duplicating source code is error-prone

How to solve the problem?

- Generic version independent makefile recursively calls EPICS build facility for all installed releases
- Change name of temporary build directory
  - old: O.<*arch*>
  - new: O.<*release*>_<*arch*>
- Generic makefile can do even more:
  - find out what to compile/install
  - build a loadable library from it
  - install driver library and dbd with version handling

The SLS driver.makefile

- Detects source files in directory
  - All *.c, *.cc, *.st, *.stt in source directory
  - But files can also be listed manually
  - Driver module library is built for each *release/arch*
- Detects *.dbd files
  - Files can also be listed manually
  - Files are expanded and combined to a single dbd file
- Detects driver version number from CVS tags
  - Modules are installed with version numbers

Driver module version handling

- Generate version number from CVS tag (e.g. *driver*_1_2_3)
- Install all files (lib, dbd, include) with version numbers
  - *driver*Lib-1.2.3, *driver*-1.2.3.dbd, *driver*-1.2.3.h
- Set symbolic links to highest versions
  - *driver*Lib -> *driver*Lib-1.2.3
  - *driver*Lib-1 -> *driver*Lib-1.2.3
  - *driver*Lib-1.2 -> *driver*Lib-1.2.3
- Install test version *driver*Lib-test if any source file is untagged
  - No symbolic links for test versions

Why loading drivers dynamically?

- We install projects on more than one IOC and more than one project on an IOC
  - Projects require drivers
  - Projects should be independent
  - Each project has its own startup script
- We don’t want monolithic library with *all* drivers included
  - Difficult to maintain (rebuilding, distributing version)
  - How to handle test versions?
  - At least on vxWorks, it is easy to load a module with `ld`

Loading driver modules

- Just `ld < driverLib` is not enough
  - Also load module dbd file (`driver.dbd`) if it exists
  - R3.14: register driver, device support, etc. to iocsh
  - Prevent duplicate loading (of different versions)
- New command `require "`*`driver`*`" [, "`*`version`*`"]`
  - Checks if driver module has already been loaded
    - YES: Check version number (mismatch: abort startup script)
    - NO: Load library with `ld`
      load dbd file with `dbLoadDatabase`
      R3.14: call *`driver`*`_registerRecordDeviceDriver()`

Non-vxWorks implementation (R3.14)

- Build driver module as LOADABLE_LIBRARY including *driver*_registerRecordDeviceDriver.cpp
- Linux (any Unix?):
  - driver module is built as shared library lib*driver*.so
  - `ld()` loads library with `dlopen()`
  - `require()` finds register function with `dlsym()`
- Windows (not yet implemented):
  - driver module is build as *driver*.dll
  - `ld()` loads library with `LoadLibrary()`
  - `require()` finds register function with `GetProcAddress()`

Unsolved problems

- Windows
  - I have no clue how DLLs in Windows work
  - R3.14.7 base does not compile properly with cygwin gcc
  - No loadable library support for cygwin
  - I could not yet build loadable libraries with Visual Studio compiler

END