Frameworks and Components

Brian Barrett

Overview

- Already seen the MCA overview
- Time to put it to use
  - Adding a new framework
  - Adding a new component
- Covers build system, naming conventions, and required source code

Creating a Framework

- Choose a name
- Create `<project>/mca/<name>/`
- Create `<project>/mca/<name>/base/`
- Define interface in `<project>/mca/<name>/<name>.h`
- Create functions for framework to open/initialize/close components
- Re-run autogen.sh

Create Some Directories

- Create basic directory structure under `<project>/mca/`
  - `<name>`
  - `<name>/base/`
- Add `Makefile.am` file in `<name>/`, probably copying from another framework
- Library name must be specified as `libmca_<framework_name>.la`

Framework Header

- Framework header:
  - `<project>/mca/<name>/<name>.h`
- Need to define two structures
  - `mca_<name>_base_module_t`
  - `mca_<name>_base_component_t`
- Quick example…

Autogen, autogen, autogen

- After adding a framework (or component), run autogen.sh
  - Adds the framework to the list of frameworks to be configured
  - Adds all the Makefiles to the list that need to be created by configure
- Failure to run autogen.sh will result in framework being (silently) ignored

Creating a Component

- Choose a name
- Create directory:
  `<project>/mca/<framework>/<component>`
- Provide build system information
- Provide component structure of known name
- Run autogen.sh

Component Build System

- Three choices for how to configure component
  - **no-configure**: component always built
  - **configure.m4**: component provides macro run in main configure script
  - **configure.stub**: component will have its own configure script, run from main configure
- First two preferred -- depends on what you are building

Configure.params

- Build system information for component
- Provides options on component building
  - `PARAM_INIT_FILE`: A file (relative to top of component directory) that should always exit. A sanity check for the build system.
  - `PARAM_CONFIG_FILES`: Space delimitated list of files that should be added to AC_CONFIG_FILES.
  - `PARAM_CONFIG_HEADER_FILE`: Configuration header file (configure.stub only)
  - `PARAM_AUX_DIR`: Where to store configure-related files (configure.stub only)
  - `PARAM_WANT_COMPILE_EXTERNAL`: unused

No-configure Components

- Easiest way to add a component
- Component must be applicable everywhere
- Must provide PARAM_INIT_FILE and PARAM_CONFIG_FILES options in configure.params

Configure.m4 Components

- Most complex way to add components
- Recommended when no-configure not appropriate (due to configure speed)
- Provide configure.m4 file that contains macro to configure component
- Provide configure.params giving PARAM_INIT_FILE and PARAM_CONFIG_FILE

Simple Example

```
# MCA_btl_tcp_CONFIG([action-if-found],
#                    [action-if-not-found])
# -----------------------------------------------------------
AC_DEFUN([MCA_btl_tcp_CONFIG],[
    # check for sockaddr_in (a good sign we have TCP)
    AC_CHECK_TYPES([struct sockaddr_in],
                   [$1],
                   [$2],
                   [AC_INCLUDES_DEFAULT
#ifdef HAVE_NETINET_IN_H
#include <netinet/in.h>
#endif])
])dnl
```

Macro Restrictions

- Evaluate $1 if component can build, $2 otherwise
- Only call AC_MSG_ERROR if user requested something not available
- Macro evaluated in context of top-level configure -- take care with variables
- Use AC_SUBST and AM_flags options to set compiler / linker options

More Complex Example

```
AC_DEFUN([MCA_btl_portals_CONFIG],[
  # save compiler flags so that we don't alter
  # them for later components.
  btl_portals_save_CPPFLAGS="$CPPFLAGS"

  # check for Portals libraries, setting
  # btl_portals_CPPFLAGS

  # substitute in the things needed to build Portals
  AC_SUBST([btl_portals_CPPFLAGS])

  # reset the flags for the next test
  CPPFLAGS="$btl_portals_save_CPPFLAGS"])dnl
```

More Complex Example

```
AM_CPPFLAGS = $(btl_portals_CPPFLAGS)

portals_SOURCES =  blah

…

noinst_LTLIBRARIES = $(component_noinst)
libmca_btl_portals_la_SOURCES = $(portals_SOURCES)
nodist_libmca_btl_portals_la_SOURCES = \
   $(portals_nodist_SOURCES)
libmca_btl_portals_la_LIBADD = $(btl_portals_LIBS)
libmca_btl_portals_la_LDFLAGS = -module -avoid-version \
   $(btl_portals_LDFLAGS)
```

Configure.stub components

- Rarely used anymore
- Slightly less complexity than configure.m4
  - Less care with variables
  - Can more easily modify CFLAGS, LIBS, etc
- Much slower to configure
  - Must re-check C compiler (and all other tests already run)
- More care required for “make dist”

OMPI_CHECK_PACKAGE Helper Macro

- Reduce burden of checking package headers and libraries
- Deals with <prefix>/lib and <prefix>/lib64 issue “almost rationally”
- Will not add -I or -L arguments if not needed (Deal with `/usr/include` breaking compilers issue)
- When things break, one place to fix

Are We Done Yet?

- Sounds pretty complicated (and it is)
- However, copying from another component is recommended path
  - Just make sure to change all the variable names…
- When in doubt, Brian and Jeff can usually help out