CSX Software Test Protocol

COLLABORATORS

	TITLE : CSX Software Test Protocol		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Frederick Ollinger	March 6, 2015	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2015.03.06	Initial Version	kmc

Contents

Chapter 1

Getting Started With Autotools

1.1 Initial configure.ac

In the top level of your project, create the simplest `configure.ac`. This file will eventually be turned into our `configure` file by autoconf.

```
    AC_INIT([libabc], [1.0]) # BOILERPLATE BEGIN: name of lib and the version
    AC_OUTPUT   # BOILERPLATE END
```

Note: this is the simplest example. But it does nothing. We can transform it to a do nothing `configure` and test it:

```
    autoconf
    ./configure
```

1.2 Initial Makefile.in

Next, let's make our initial `Makefile.in`. Again, we'll make a useless, do nothing file at first so we can getting a clearer picture of all the pieces:

```
    package        = @PACKAGE_NAME@
    SOURCES        = $(wildcard src/*.c)
    srcdir         = @srcdir@
```

Now edit `configure.ac` and add the following:

```
    AC_CONFIG_FILES([Makefile]) # create top level Makefile from Makefile.in
```

Thus, our new configure.ac looks like this:

```
    AC_INIT([libmtd], [1.0]) # BOILERPLATE BEGIN: name of lib and the version
    AC_CONFIG_FILES([Makefile]) # create top level Makefile from Makefile.in
    AC_OUTPUT   # BOILERPLATE END
```

1.3 Basic Functionality

If it's a C or C++ library, we want to autodetect compiler or allow the user to specify it. Add the following to `configure.ac`

```
    AC_C_INLINE # Find C compiler (gcc)
```

Now add the compiler to the Makefile.in:

```
    CC   = @CC@
```

Don't forget to rerun:

```
    autoconf
    ./configure
```

From now on, it will be assumed that each time you change `configure.ac` and want a new `Makefile`, you'll do the above.

If we have C++, we need add this as well to `configure.ac`

```
    AC_LANG_CPLUSPLUS
```

Instead of CC, we use CXX in our `Makefile.in`

```
    CXX = @CXX@
```

Now we need to compile the code. Add to `Makefile.in`

```
    .c.o:
        $(CC) $(AM_CFLAGS) -c $< -o $@
```

Note we need to get the AM_FLAGS. Add this to `Makefile.in`

```
    LIBDIR      = libs
    INCLUDEDIR = include
    CFLAGS      = @CFLAGS@ # user generated do not overwrite CFLAGS use AM_CFLAGS
    LDFLAGS     = @LDFLAGS@
    AM_CFLAGS   = -Wall -I$(INCLUDEDIR) -fPIC -fno-strict-aliasing -O2 -g -D_GNU_SOURCE $( ↩
        CFLAGS)
```

As the comment says, we don't want to add to CFLAGS because these will be overwritten. This is a good thing as it gives flexibility.

```
    LIBDIR = libs
    SOURCES = $(wildcard *.c)
    OBJS = $(SOURCES:%.c=%.o)

    TARGET_LINK = $(package).so
    TARGET = $(TARGET_LINK).1.0.0

    all: $(TARGET)

    $(TARGET): $(OBJS)
        mkdir -p $(LIBDIR)
        $(CC) $(OBJS) $(AM_LDFLAGS) $(AM_CFLAGS) -o $(LIBDIR)/$(TARGET)
```

1.4 Autotools Summary

By now, we should have a working system that actually builds our library. Here's the `Makefile.in`:

```
LIBDIR            = libs
INCLUDEDIR        = include

package         = @PACKAGE_NAME@
srcdir          = @srcdir@
CC              = @CC@
CFLAGS          = @CFLAGS@  # user generated do not overwrite CFLAGS use AM_CFLAGS
```

```
LDFLAGS         = @LDFLAGS@ # user generated do not overwrite LDFLAGS use AM_LDFLAGS. In  ↩
    this example none are needed.

AM_CFLAGS = -Wall -I$(INCLUDEDIR) -fPIC -fno-strict-aliasing -O2 -g -D_GNU_SOURCE $(CFLAGS)
AM_LDFLAGS = -shared -Wl,-soname,$(TARGET_LINK) -lpthread -ldl $(LDFLAGS)

SOURCES         = $(wildcard src/*.c)
OBJS            = $(SOURCES:%.c=%.o)

TARGET_LINK = $(package).so
TARGET = $(TARGET_LINK).1.0.0

all: $(TARGET)

$(TARGET): $(OBJS)
    mkdir -p $(LIBDIR)
    $(CC) $(OBJS) $(AM_LDFLAGS) $(AM_CFLAGS) -o $(LIBDIR)/$(TARGET)

.c.o:
    $(CC) $(AM_CFLAGS) -c $< -o $@
```

Here's the `configure.ac`:

```
AC_INIT([libmtd], [1.0]) # BOILERPLATE BEGIN: name of lib and the version
AC_C_INLINE # Find C compiler (gcc)
AC_CONFIG_FILES([Makefile]) # create top level Makefile from Makefile.in
AC_OUTPUT   # BOILERPLATE END
```

Chapter 2

Adding Targets

Now that we have a basic format, we can add more targets.

2.1 Clean Targets

Let's add some clean targets to the `Makefile.in`

```
clean:
    rm -rf $(OBJS) libs *.pc

distclean: clean
    rm -rf autom4te.cache config.log config.status Makefile build ltmain.sh m4 aclocal. ↩
        m4

cleanall: distclean
    rm -f configure
```

The first, clean, target will clearly clean out the object files, the finished library as well as the pkgconfig file which we'll get to. This is a normal Makefile clean.

The second target, distclean, gets rid of all the junk that autoconf creates. This will restore the directory to the state in which it was checked into verion control. It will get on to the unconfigured state.

The last one gets rid of the configure in case one wanted to go back to just the human generated files. This is for maintainers, only. (I didn't call it maintainer-clean because it's too much typing.

2.2 Install Target

No good if we don't get the files where they need to be

```

HEADERS = $(INCLUDEDIR)/$(package).h

install:
    mkdir -p $(DESTDIR)$(libdir)
    mkdir -p $(DESTDIR)$(includedir)/$(package)

    $(VERBOSE) install -m 755 $(LIBDIR)/* $(DESTDIR)$(libdir)
    $(VERBOSE) install -m 644 $(HEADERS) $(DESTDIR)$(includedir)/$(package)
```

2.3 Adding Versioning Information

This is going to come up later so we might as well add it now to the `configure.ac`:

```
    m4_define([package_version_major],[1])
    m4_define([package_version_minor],[0])
    m4_define([package_version_micro],[0])
    m4_define([api_version],[package_version_major])
    AC_SUBST([PACKAGE_VERSION_MAJOR],package_version_major)
    AC_SUBST([PACKAGE_VERSION_MINOR],package_version_minor)
    AC_SUBST([PACKAGE_VERSION_MICRO],package_version_micro)
    AC_SUBST([API_VERSION],api_version)
```

Note that our api version is the same as our major version. This is by design.

For those who are keeping up, our `configure.ac` is now:

```
AC_INIT([libmtd], [1.0]) # BOILERPLATE BEGIN: name of lib and the version
m4_define([package_version_major],[1])
m4_define([package_version_minor],[0])
m4_define([package_version_micro],[0])
m4_define([api_version],[1])
AC_SUBST([PACKAGE_VERSION_MAJOR],package_version_major)
AC_SUBST([PACKAGE_VERSION_MINOR],package_version_minor)
AC_SUBST([PACKAGE_VERSION_MICRO],package_version_micro)
AC_SUBST([API_VERSION],api_version)
AC_C_INLINE # Find C compiler (gcc)
AC_CONFIG_FILES([Makefile]) # create top level Makefile from Makefile.in
AC_OUTPUT   # BOILERPLATE END
```

Chapter 3

Pkg-config

Pkg-config is a system where shared libraries will let us know all the information we need in order to build applications against them. Autotools can help us make them by dynamically filling in information.

3.1 The pkg-config Template File

First we need a basic pkg-config file: `libabc.pc.in`. The extention .in does imply that this is a file which will be used as a template to autogenerate our `libabc.pc`.

3.2 The pkg-config Template File

The first line is boilerplate stuff to get started. It ensures that the machinery for pkg-config is part of the build system

```
PKG_PROG_PKG_CONFIG # PKG-CONFIG BOILERPLATE
```

PKG_INSTALLDIR allows the user user to override the ordinary path for pkgconfig.

```
PKG_INSTALLDIR
```

```
AC_CONFIG_FILES([$PACKAGE_NAME-$API_VERSION.pc:$PACKAGE_NAME.pc.in ],[], [API_VERSION='  ↩
    $API_VERSION'])
```