

# **CSX Software Test Protocol**

<b>COLLABORATORS</b>
----------------------

	<i>TITLE :</i> CSX Software Test Protocol		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Frederick Ollinger	March 6, 2015	

<b>REVISION HISTORY</b>
-------------------------

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2015.03.06	Initial Version	kmc

# Contents

<b>1</b>	<b>Getting Started With Autotools</b>	<b>1</b>
1.1	Initial configure.ac . . . . .	1
1.2	Initial Makefile.in . . . . .	1
1.3	Basic Functionality . . . . .	1
1.4	Autotools Summary . . . . .	2
<b>2</b>	<b>Adding Targets</b>	<b>4</b>
2.1	Clean Targets . . . . .	4
2.2	Install Target . . . . .	4
<b>3</b>	<b>Pkg-config</b>	<b>5</b>

## 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`.

```
1 AC_INIT([libabc], [1.0]) # BOILERPLATE BEGIN: name of lib and the version
2 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:

```
1 autoconf
2 ./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:

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

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

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

Thus, our new `configure.ac` looks like this:

```
1 AC_INIT([libmtd], [1.0]) # BOILERPLATE BEGIN: name of lib and the version
2 AC_CONFIG_FILES([Makefile]) # create top level Makefile from Makefile.in
3 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`

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

Now add the compiler to the Makefile.in:

```
1 CC = @CC@
```

Don't forget to rerun:

```
1 autoconf
2 ./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`

```
1 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`

```
1  clean:
2      rm -rf $(OBJDIR) libs *.pc
3
4  distclean: clean
5      rm -rf autom4te.cache config.log config.status Makefile build ltmain.sh m4 aclocal. ↵
6      m4
7
8  cleanall: distclean
9      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 version 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

```
1  install:
2      mkdir -p $(DESTDIR)$(libdir)
3      mkdir -p $(DESTDIR)$(includedir)/$(package)
4
5      $(VERBOSE) install -m 755 $(LIBDIR)/* $(DESTDIR)$(libdir)
6      $(VERBOSE) install -m 644 $(HEADERS) $(DESTDIR)$(includedir)/$(package)
```

## Chapter 3

# Pkg-config

What of it?

---