Subject: Autotool support by Upp
Posted by fallingdutch on Thu, 22 Mar 2007 08:50:22 GMT
View Forum Message <> Reply to Message

I would like to add a way to create Makefile.am's for autotools in Upp so the source release will be more portable.

I thought of a Makefile per Package:
So eg Core gets a Makefile.am and CtrlLib etc
which will output a Library eg noinst_LIBRARIS=Core.a
Core_a_SOURCES= ....
Then the Packages just will have to use the libraries like bin_PROGRAMMS=theide
theide_SOURCES=...
theide_LDADD=../Core/Core.a

For me it is still a problem how to deal with .icpp files.
Mirek introduced them to be files that always have to be recompiled (mostly files including language stuff or database definitions etc) so there should be a way to compile them *every* time and include the objects into the source.

Information/Code/Suggestions etc are very welcome,

Bas

Subject: Re: Autotool support by Upp
Posted by ebojd on Thu, 22 Mar 2007 13:01:22 GMT
View Forum Message <> Reply to Message

fallingdutch wrote on Thu, 22 March 2007 03:50For me it is still a problem how to deal with .icpp files.
Mirek introduced them to be files that always have to be recompiled (mostly files including language stuff or database definitions etc) so there should be a way to compile them *every* time and include the objects into the source.

Information/Code/Suggestions etc are very welcome

There should be several ways of dealing with this. Probably the easiest is to define a .icpp.o translation SUFFIX, and a clean_icpp_ofile method which removes the .o files every time (mind you this is strictly off the top of my head and not tested -- your milage may vary):

ICPP_SRC = bla.icpp woof.icpp

ICPP_OBJ = $(addsuffix .o, $(basename $(ICPP_SRC)))

```
.SUFFIXES: .c .cc .icpp

.icpp.o:
	$(CXX) $(CXXFLAGS) $(UPPFLAGS) -c $<

all: theide

theide: clean_icpp_ofile $(ICPP_OBJS) $(OBJS) ...
	....

clean_icpp_ofile:
	-$(RM) $(ICPP_OBJS)
```

The intent here, if I got this right, is to remove all .o files associated with the .icpp files, then rebuild them.

This is probably not the most elegant mway to get the job done. I seem to remember that there was a way to force compilation using the .FORCE. rule <see http://theoryx5.uwinnipeg.ca/gnu/make/make_19.html> for example). I do not know how portable this approach would be though.

In addition to this you will need to carefully address properly building the dependencies. There is a program called makedepend that will do this, but there is also a method using the -M compiler switch IIRC (but I have not used that one yet).

Hope that helps, more later

EBo --

Subject: Re: Autotool support by Upp
Posted by mirek on Thu, 22 Mar 2007 16:07:03 GMT
View Forum Message <> Reply to Message

fallingdutch wrote on Thu, 22 March 2007 04:50For me it is still a problem how to deal with .icpp files.
Mirek introduced them to be files that always have to be recompiled (mostly files including language stuff or database definitions etc) so there should be a way to compile them *every* time and include the objects into the source.

This is not quite correct.

.icpp files are guaranteed to be included in the final executable.

The problem to solve is that build process tries to minimize the size of executable by not including stuff that is not referenced. First level of this optimization is to link only those objects from library that are referenced. Well, most accurate definition is that in release mode, package are build into .lib (or .a) files first, so that unused .obj (.o) are elimintaed.

So far so good, but quite common trick in U++ is to register modules using global constructors (usually using INITBLOCK macro). E.g. image decoders register themselves so that "LoadAnyFile" can use all possible decoders to load image files. But adding global constructor unfortunately does not make .obj referenced, therefore linker is still free to eliminate it, effectivily eliminating the whole image decoder.

We have struggled with this issue for some time and then decided to tweak build process to solve this problem - .icpp files are simply not put into .lib, but are always linked as .obj. This ensures they are never eliminated from the target.

But no, you do not need to recompile them every time. You just need to not put them into library and link .obj directly.

Mirek
