Quick and dirty porting of autoconf apps

Introduction

With the release of the OpenC plugin, it has become far easier to take an existing open source project to Symbian OS. Here's how to take an open source project developed using autotools, and turn it in to a Symbian project fast.

I'm going to be using libxml2 [1] as the example project, but these steps generally work for any project.

Task one: Build a bld.inf file for the project

The first task when porting software to Symbian OS is to write a bld.inf file. For libraries like libxml2, this will export all the appropriate headers to the \epo32\include tree, together with a customised config.h.

Start by opening the appropriate Makefile.am, and locate the list of headers that are exported. In libxml2, this is:

```
xmlinc_HEADERS = \
  SAX.h \
  entities.h \
  encoding.h \
  parser.h \

/* ... etc ... */
```

Copy and paste this in to your bld.inf file, under the heading PRJ_EXPORTS. Rewrite each line to export it to an appropriate subdirectory of \epoc32\include. In this instance, we'll be exporting to \epoc32\include\libxml as shown below:

```
PRJ_EXPORTS
..\include\libxml\SAX.h libxml\SAX.h
..\include\libxml\entities.h libxml\entities.h
..\include\libxml\encoding.h libxml\encoding.h
..\include\libxml\parser.h libxml\parser.h

/* ... etc ... */
```

Now, create a copy of config.h.in called config.h.symbian, and add an export to your project exports to ex as so:

```
..\config.h.symbian libxml\config.h
```

Now your exports are complete. From the command line, type bldmake bldfiles followed by abld export to export the headers.

Task two: Building an MMP file

Create an empty MMP file for your project, and add a boilerplate header to it along these lines:

```
TARGET libxml2.dll
TARGETTYPE dll
UID 0x1000008d 0xDEADBEEF
VENDORID 0
```

Where 0xDEADBEEF would be replaced by a UID you've procured from Symbian Signed. Now, again find in Makefile.am the list of project sources, for libxml2, this is:

```
libxml2_la_SOURCES = SAX.c entities.c encoding.c error.c parserInternals.c  \
```

```
  parser.c tree.c hash.c list.c xmlIO.c xmlmemory.c uri.c  \
/* ... etc ... */
```

Copy and paste this is to your MMP file, and munge each one in to a SOURCE line, as shown here:

```
SOURCEPATH      ..\
SOURCE SAX.c
SOURCE entities.c
SOURCE encoding.c
SOURCE error.c
SOURCE parserInternals.c
SOURCE parser.c
SOURCE tree.c

/* ... etc ... */
```

Now, add some system include paths. Minimally you'll want the stdapis directory, and the folder to which you exported all project headers. This is shown below:

```
SYSTEMINCLUDE   \epoc32\include
SYSTEMINCLUDE   \epoc32\include\stdapis
SYSTEMINCLUDE \epoc32\include\libxml
```

Then, add a libraries line to link the project against lib.c, and assign it some capabilities and you're done.

For some projects it may be necessary to define the macro HAVE_CONFIG_H to include your config.h. This isn't the case with libxml, but to do that include this line in your MMP file:

```
MACRO HAVE_CONFIG_H
```

Task three: Customise config.h.symbian

Now import your completed MMP file in to your favourite IDE. Next we need to customise config.h to the Symbian platform. For every entry in config.h, you need to decide if it does or does not apply to Symbian. For some entries, this is easy.

For example, things like:

```
#undef HAVE_FCNTL_H
```

We know Symbian has this header, so change it to:

```
#define HAVE_FCNTL_H 1
```

Others aren't so simple, they control the behaviour of the application. If you aren't sure what a config.h define does, just search for it in the code and read the context in which it is used, and then decide if it should be on or off.

Once you are finished, remember to export the headers again to update your config.h in \epoc32\include.

Task four: compile and fix

Now, compile the project. If it fails to compile, then fix the compile errors. Obviously, this varies from project to project, so there can't be any specific advice for this. Libxml2 compiles flawlessly.

Task five: Run and debug

Now run the application, or an application that uses the library. Again, you may encounter errors but this guide can't advise you on the specifics of fixing them.

Done!

And that's that. You should now have a working Symbian executable or library. From start to finish, libxml2 took about half an hour to port.

Hope this helps. --Cdavies-nokia 14:35, 27 June 2007 (UTC)