** This file is adapted from libcurl and not yet fully rewritten for c-ares! **

```
                                  ___       __ _ _ __ ___  ___
                                 / __| ___ / _` | '__/ _ \/ __|
                                | (_  |___| (_| | | |  __/\__ \
                                 \___|     \__,_|_|  \___||___/

                                        How To Compile
```

Installing Binary Packages

Lots of people download binary distributions of c-ares. This document does not describe how to install c-ares using such a binary package. This document describes how to compile, build and install c-ares from source code.

Building from Git

If you get your code off a Git repository rather than an official release tarball, see the GIT-INFO file in the root directory for specific instructions on how to proceed.

In particular, if not using CMake you will need to run `./buildconf` (Unix) or `buildconf.bat` (Windows) to generate build files, and for the former you will need a local installation of Autotools. If using CMake the steps are the same for both Git and official release tarballs.

AutoTools Build

General Information, works on most Unix Platforms (Linux, FreeBSD, etc)

A normal Unix installation is made in three or four steps (after you've unpacked the source archive):

```
./configure
make
make install
```

You probably need to be root when doing the last command.

If you have checked out the sources from the git repository, read the GIT-INFO on how to proceed.

Get a full listing of all available configure options by invoking it like:

```
./configure --help
```

If you want to install c-ares in a different file hierarchy than /usr/local, you need to specify that already when running configure:

```
./configure --prefix=/path/to/c-ares/tree
```

If you happen to have write permission in that directory, you can do `make install` without being root. An example of this would be to make a local install in your own home directory:

```
./configure --prefix=$HOME
make
make install
```

More Options

To force configure to use the standard cc compiler if both cc and gcc are present, run configure like

```
CC=cc ./configure
# or
env CC=cc ./configure
```

To force a static library compile, disable the shared library creation by running configure like:

```
./configure --disable-shared
```

If you're a c-ares developer and use gcc, you might want to enable more debug options with the `--enable-debug` option.

Special Cases

Some versions of uClibc require configuring with `CPPFLAGS=-D_GNU_SOURCE=1` to get correct large file support.

The Open Watcom C compiler on Linux requires configuring with the variables:

```
./configure CC=owcc AR="$WATCOM/binl/wlib" AR_FLAGS=-q \
    RANLIB=/bin/true STRIP="$WATCOM/binl/wstrip" CFLAGS=-Wextra
```

CROSS COMPILE

(This section was graciously brought to us by Jim Duey, with additions by Dan Fandrich)

Download and unpack the c-ares package.

`cd` to the new directory. (e.g. `cd c-ares-1.7.6`)

Set environment variables to point to the cross-compile toolchain and call configure with any options you need. Be sure and specify the `--host` and `--build` parameters at configuration time. The following script is an example of cross-compiling for the IBM 405GP PowerPC processor using the toolchain from MonteVista for Hardhat Linux.

```
#! /bin/sh

export PATH=$PATH:/opt/hardhat/devkit/ppc/405/bin
export CPPFLAGS="-I/opt/hardhat/devkit/ppc/405/target/usr/include"
export AR=ppc_405-ar
export AS=ppc_405-as
export LD=ppc_405-ld
export RANLIB=ppc_405-ranlib
export CC=ppc_405-gcc
export NM=ppc_405-nm

./configure --target=powerpc-hardhat-linux \
     --host=powerpc-hardhat-linux \
```

```
    --build=i586-pc-linux-gnu \
    --prefix=/opt/hardhat/devkit/ppc/405/target/usr/local \
    --exec-prefix=/usr/local
```

You may also need to provide a parameter like `--with-random=/dev/urandom` to configure as it cannot detect the presence of a random number generating device for a target system. The `--prefix` parameter specifies where c-ares will be installed. If `configure` completes successfully, do `make` and `make install` as usual.

In some cases, you may be able to simplify the above commands to as little as:

```
./configure --host=ARCH-OS
```

Cygwin (Windows)

Almost identical to the unix installation. Run the configure script in the c-ares root with `sh configure`. Make sure you have the sh executable in `/bin/` or you'll see the configure fail toward the end.

Run `make`

QNX

(This section was graciously brought to us by David Bentham)

As QNX is targeted for resource constrained environments, the QNX headers set conservative limits. This includes the `FD_SETSIZE` macro, set by default to 32. Socket descriptors returned within the c-ares library may exceed this, resulting in memory faults/SIGSEGV crashes when passed into `select(..)` calls using `fd_set` macros.

A good all-round solution to this is to override the default when building c-ares, by overriding `CFLAGS` during configure, example:

```
# configure CFLAGS='-DFD_SETSIZE=64 -g -O2'
```

RISC OS

The library can be cross-compiled using gccsdk as follows:

```
CC=riscos-gcc AR=riscos-ar RANLIB='riscos-ar -s' ./configure \
     --host=arm-riscos-aof --without-random --disable-shared
make
```

where `riscos-gcc` and `riscos-ar` are links to the gccsdk tools. You can then link your program with `c-ares/lib/.libs/libcares.a`.

Android

Method using a configure cross-compile (tested with Android NDK r7b):

- prepare the toolchain of the Android NDK for standalone use; this can be done by invoking the script:

  ```
  ./tools/make-standalone-toolchain.sh
  ```

  which creates a usual cross-compile toolchain. Lets assume that you put this toolchain below `/opt` then invoke configure with something like:

```
export PATH=/opt/arm-linux-androideabi-4.4.3/bin:$PATH
./configure --host=arm-linux-androideabi [more configure options]
make
```

- if you want to compile directly from our GIT repo you might run into this issue with older automake stuff:

  ```
  checking host system type...
  Invalid configuration `arm-linux-androideabi':
  system `androideabi' not recognized
  configure: error: /bin/sh ./config.sub arm-linux-androideabi failed
  ```

  this issue can be fixed with using more recent versions of `config.sub` and `config.guess` which can be obtained here: http://git.savannah.gnu.org/gitweb/?p=config.git;a=tree you need to replace your system-own versions which usually can be found in your automake folder: `find /usr -name config.sub`

CMake builds

Current releases of c-ares introduce a CMake v3+ build system that has been tested on most platforms including Windows, Linux, FreeBSD, MacOS, AIX and Solaris.

In the most basic form, building with CMake might look like:

```
cd /path/to/cmake/source
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=/usr/local/cares ..
make
sudo make install
```

Options

Options to CMake are passed on the command line using "-D${OPTION}=${VALUE}". The values defined are all boolean and take values like On, Off, True, False.

- CARES_STATIC - Build the static library (off by default)
- CARES_SHARED - Build the shared library (on by default)
- CARES_INSTALL - Hook in installation, useful to disable if chain building
- CARES_STATIC_PIC - Build the static library as position-independent (off by default)

Ninja

Ninja is the next-generation build system meant for generators like CMake that heavily parallize builds. Its use is very similar to the normal build:

```
cd /path/to/cmake/source
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=/usr/local/cares -G "Ninja"
```

```
..
ninja
sudo ninja install
```

Windows MSVC Command Line

```
cd \path\to\cmake\source
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=C:\cares -G "NMake Makefiles"
..
nmake
nmake install
```

Windows MinGW-w64 Command Line via MSYS

```
cd \path\to\cmake\source
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=C:\cares -G "MSYS Makefiles"
..
make
make install
```

Platform-specific build systems

Win32

Building Windows DLLs and C run-time (CRT) linkage issues

As a general rule, building a DLL with static CRT linkage is highly discouraged, and intermixing CRTs in the same app is something to avoid at any cost.

Reading and comprehension of Microsoft Knowledge Base articles KB94248 and KB140584 is a must for any Windows developer. Especially important is full understanding if you are not going to follow the advice given above.

- KB94248 - How To Use the C Run-Time
- KB140584 - How to link with the correct C Run-Time (CRT) library
- KB190799 - Potential Errors Passing CRT Objects Across DLL Boundaries

If your app is misbehaving in some strange way, or it is suffering from memory corruption, before asking for further help, please try first to rebuild every single library your app uses as well as your app using the debug multithreaded dynamic C runtime.

MingW32

Make sure that MinGW32's bin dir is in the search path, for example:

```
set PATH=c:\mingw32\bin;%PATH%
```

then run 'make -f Makefile.m32' in the root dir.

MSVC 6 caveats

If you use MSVC 6 it is required that you use the February 2003 edition PSDK:
http://www.microsoft.com/msdownload/platformsdk/sdkupdate/psdk-full.htm

MSVC from command line

Run the `vcvars32.bat` file to get a proper environment. The `vcvars32.bat` file is part of the Microsoft development environment and you may find it in `C:\Program Files\Microsoft Visual Studio\vc98\bin` provided that you installed Visual C/C++ 6 in the default directory.

Further details in README.msvc

Important static c-ares usage note

When building an application that uses the static c-ares library, you must add `-DCARES_STATICLIB` to your `CFLAGS`. Otherwise the linker will look for dynamic import symbols.

IBM OS/2

Building under OS/2 is not much different from building under unix. You need:

- emx 0.9d
- GNU make
- GNU patch
- ksh
- GNU bison
- GNU file utilities
- GNU sed
- autoconf 2.13

If during the linking you get an error about `_errno` being an undefined symbol referenced from the text segment, you need to add `-D__ST_MT_ERRNO__` in your definitions.

If you're getting huge binaries, probably your makefiles have the `-g` in `CFLAGS`.

NetWare

To compile `libcares.a` / `libcares.lib` you need:

- either any gcc / nlmconv, or CodeWarrior 7 PDK 4 or later.
- gnu make and awk running on the platform you compile on; native Win32 versions can be downloaded from: http://www.gknw.net/development/prgtools/
- recent Novell LibC SDK available from: http://developer.novell.com/ndk/libc.htm
- or recent Novell CLib SDK available from: http://developer.novell.com/ndk/clib.htm

Set a search path to your compiler, linker and tools; on Linux make sure that the var `OSTYPE` contains the string 'linux'; set the var `NDKBASE` to point to the base of your Novell NDK; and then type `make -f Makefile.netware` from the top source directory;

PORTS

This is a probably incomplete list of known hardware and operating systems that c-ares has been compiled for. If you know a system c-ares compiles and runs on, that isn't listed, please let us know!

```
- Alpha Tru64 v5.0 5.1
- ARM Android 1.5, 2.1, 2.3
- MIPS IRIX 6.2, 6.5
- Power AIX 3.2.5, 4.2, 4.3.1, 4.3.2, 5.1, 5.2
- i386 Linux 1.3, 2.0, 2.2, 2.3, 2.4, 2.6
- i386 Novell NetWare
- i386 Windows 95, 98, ME, NT, 2000, XP, 2003
- x86_64 Linux
```

Useful URLs

- c-ares: https://c-ares.org/
- MingW: http://www.mingw.org/
- MinGW-w64: http://mingw-w64.sourceforge.net/
- OpenWatcom: http://www.openwatcom.org/