ECLiPSe Build-and-Test Setup Guide

Author: Joachim Schimpf, Andrew Cheadle, Kish Shen
Date: 2006-11-24
Version: 5
Relates to: ECLiPSe 6.0
History: Version 1, 2006-06-23, Version 2, 2007-05-08, Version 3, 2007-12-03, Version 4: 2008-09-12, Version 5: 2008-10-29

This document describes how to set up all the tools necessary to compile and test the ECLiPSe Constraint Logic Programming System on different operating systems and hardware architectures.

Table of Contents

Architectures

ECLiPSe can be built for a variety of machine architecture/operating system combinations, for example

Architecture/Operating System	*ECLiPSe architecture ID*
Solaris 5.5-5.9, sparc 32 bit	sparc_sunos5
Solaris 5.10, x86 32-bit	i386_sunos5
Solaris 5.10, x86 64-bit	x86_64_sunos5
Linux, x86 32-bit, many variants	i386_linux
Linux, x86 64-bit (x86-64, AMD64)	x86_64_linux
Windows NT/2000/XP, x86 32-bit	i386_nt
MacOS-X, Power PC	ppc_macosx
MacOS-X, x86 (Intel)	i386_macosx
DEC Alpha, Linux 64-bit	alpha_linux

The ECLiPSe architecture ID is used throughout the documentation and the system, e.g. to label architecture-dependent sub-directories and the like.

ECLiPSe 5.10 is designed to be built on UNIX-like architectures, with the Windows version being cross-built. Earlier Windows versions were built with a native Windows compiler – the corresponding project files are not used in 5.10.

Environment settings

For building ECLiPSe, the following environment variables need to be set:

- ARCH – the ECLiPSe architecture id, see above
- CVSROOT and CVS_RSH – location and shell for accessing the source repository
- ECLIPSETOOLS – location of software tools, see below
- ECLIPSETHIRDPARTY – location of support software, see below
- JAVA_HOME – location of Java development system, used to find include files when building the Java/ECLiPSe interface
- PATH – might need to be extended to include the $ECLIPSETOOLS/bin and $ECLIPSETOOLS/$ARCH/bin.
- OSTYPE – this is preset for many systems running bash. It is used to determine the platform specific directory include path in some versions of Java's native AWT interface: $JAVA_HOME/include/$OSTYPE, to link in the file jawt_md.h while building the Java interface. Unfortunately, both the exact path for the file, and the value of OSTYPE, may vary in different versions of Javas and shells, so OSTYPE may need to be explicitly changed to the value needed.

For running ECLiPSe tests:

- ARCH – see above
- JRE_HOME – location of Java runtime environment, used to find the java executable (in $JRE_HOME/bin) and jar files (in $JRE_HOME/lib)

Software Tools required

If one wants to build ECLiPSe for several architectures, then it is makes sense to choose one *primary architecture*, which has all the tools installed, and make the others *secondary architectures.* The primary architecture is then used to build architecture-independent components of the system (e.g. the documentation, precompiled ECLiPSe files), while architecture-dependent components must be built on each architecture (except for those that can be cross-built, see below). The most convenient choice for the primary architecture is currently i386_linux, due to the easy availability of the tools.

The following tools are needed for **all** build-architectures:

Tool	***Remarks***	***Recommended Architecture***
sh	Bourne shell sufficient, bash useful	All
gmake	gnu-make features required	All
gcc	recommended version 3.3-3.4, 4.1, 4.2 (*not* 4.3 – generates slow code)	All
binutils	needed for ar(1)	All
tar	gnu tar (Solaris tar has no z option)	All

The following are in principle only needed on a single architecture. Autoconf is not needed for a pure build, but only when porting to new architectures.

Tool	***Remarks***	***Recommended Architecture***
cvs	To access the ECLiPSe source repository	any
autoconf	only needed to change configure scripts	any
m4	gnu m4, for autoconf	any

The next group of tools is used for building the documentation, and is therefore only required on one architecture:

Tool	***Remarks***	***Recommended Architecture***
latex2e	Standard Linux package	i386_linux
ghostscript	version 8.53 at least	i386_linux
hevea	for latex-to-html conversion	i386_linux
ocaml	for building *hevea*	i386_linux

The Windows architecture i386_nt is cross-compiled. This has been done and tested only on the i386_linux host architecture. The following tools are needed:

Tool	***Remarks***	***Recommended Architecture***
mingw32	compiler tool chain for i386_nt cross-build	i386_linux
NSIS	Windows installer	i386_linux

A few other tools should be considered for convenience:

Tool	***Remarks***	***Recommended Architecture***
tkcvs or similar	GUI for CVS and SVN	Any

Installation Instructions for Tools

All tools can of course be installed by a system administrator in the standard locations of the local setup. This makes sense in particular for standard components like gcc.

- For Linux, these can be installed from binary packages, eg. in rpm format as super user
- For Solaris, many tools are available at sunfreeware.com. They are installed using pkgadd as super user.
- For Mac OS X, the tools are included in a CD that comes with the computer, and can also be downloaded from AppleDeveloper Connection (ADC) website connect.apple.com.

If this is not possible, tools can also be installed in a private location by a restricted user. In the following, we assume that everything is installed from sources, and goes below the directory $PREFIX, and that we a using a bash shell. The following two locations should then be added to the user's PATH:

```
$PREFIX/bin:$PREFIX/$ARCH/bin
```

where $ARCH is the ECLiPSe architecture ID explained above.

General Gnu Software

GNU software is usually installed from sources and has a common installation procedure. The following applies e.g. to tar, binutils, m4, bison, flex, autoconf, gmake. These can be downloaded from any site that carries GNU software, e.g. www.gnu.org.

1. Make sure you have these settings:
```
export ARCH=sparc_sunos5                    or your architecture
export ECLIPSETOOLS=/vol/Eclipse/tools      or your chosen location
PREFIX=/vol/Eclipse/tools
EXECPREFIX=$PREFIX/$ARCH
```
2. Unpack the downloaded sources (usually a .tar.gz file) in $PREFIX/src
3. Cd to the resulting directory, then do:
```
./configure --prefix=$PREFIX –exec-prefix=$EXECPREFIX
make install
```

Windows Crosstools

Install the cross compiler (linux->mingw32msvc) from the binary package at

- http://www.libsdl.org/extras/win32/cross

under $PREFIX/i386_linux/ where it makes a subdirectory cross-tools. Although it is meant to be unpacked under /usr/local, it works without setting any paths etc, just by calling the executables. The include directories get found relative to the binary path name.

Windows NSIS Installer

The NSIS-based ECLiPSe-Windows-installer is cross-built on i386_linux. The NSIS builder must therefore be installed on i386_linux. Obtain the latest source tarball **and** the matching zip release for Windows. Build the POSIX only version of the binary (don't cross-compile it) - use the command:

```
scons SKIPSTUBS=all SKIPPLUGINS=all SKIPUTILS=all SKIPMISC=all PREFIX=…
PREFIX_BIN=… PREFIX_CONF=… PREFIX_DATA=…
```

Unzip the zip release in the location where you wish to install nsis. The 'PREFIX' options above should be set to the correct path for this location. Move the `makensis` binary from the build directory of the source release to the zip release. Ensure the makensis binary is in the binary path.

OCaml (Objective Caml)

This is required for compiling HeVeA, and should be built before building HeVeA. It can be downloaded from:

- http://caml.inria.fr/ocaml/release.en.html

As with HeVeA, this is only required for architectures that you intend to build your documentation on.

1. Set the environment variables as for building the Gnu software
2. Unpack and cd into the ocaml directory, then do:

```
./configure -prefix $PREFIX -bindir $EXECPREFIX/bin
make world
make opt
umask 022
make install
make clean
```

make opt should be done for platforms where ocaml has native code support (e.g. i386_linux)). This will allow hevea to run independently of ocaml.

HeVeA

Translator from LaTeX to HTML. Used to build the html docs from the latex sources. The source can be obtained from

- http://pauillac.inria.fr/hevea

This is only needed on platforms where you intend to build the documentation.

1. First, build ocaml as described above.
2. Unpack the downloaded tarball and cd into the hevea directory
3. Edit the Makefile, changing the entries for PREFIX, LIBDIR, BINDIR, DIR at the start of the file to the following:

```
# Compile using ocamlopt
TARGET=opt
# Use ocamlc, for targets that do not support native code for ocaml:
# TARGET=byte
# Install prefix
PREFIX=$(PREFIX)
# Library directory of hevea
LIBDIR=$(PREFIX)/lib/hevea
# Where to install programms
BINDIR=$(PREFIX)/$(ARCH)/bin
# Install prefix prefix
DESTDIR=
#Where to install hevea.sty
LATEXLIBDIR=$(PREFIX)/lib/hevea
############### End of configuration parameters
SUF=
DIR=$(BINDIR)/
```

4.Copy hevea.sty in the hevea toplevel directory to a place where LaTeX can find it (i.e. into a directory given in the TEXINPUTS environment variable). Alternatively, this can be done by setting LATEXLIBDIR in the Makefile.

5.In the hevea directory, do:

```
make
make install
```

Getting the Sources

The ECLiPSe sources can be obtained either by downloading a package from a web site that carries ECLiPSe distributions, or by accessing the master CVS repository on sourceforge.net. The ECLiPSe project name on sourceforge is **eclipse-clp**.

Source download

Find a web site that carries ECLiPSe distributions (e.g. sourceforge file releases, or eclipse-clp.org). Select a release and download eclipse_src.tgz from the group of common/platform independent files. Unpacking this will create a directory called Eclipse, containing the full sources.

CVS - General

You can use the cvs command (or a GUI for cvs) to get a local copy of the sources. See below for how to specify the repository. For example, to get a read-only copy of the 5.10 release sources with the latest verified patches, call

```
      cvs export -r last_successful_patches_5_10 Eclipse
```

Here, last_successful_patches_5_10 is a cvs *tag*. The kinds of tags that are commonly used in the ECLiPSe cvs are:

- rel_X_Y: a tag marking the point where release X.Y was made.
- patches_X,Y: a branch tag, marking the branch started with release X.Y. All patches to this release are on this branch, while major developments are on the main branch.
- last_successful_patches_X_Y, last_successful_main_branch: tags marking the last version on a branch that was successfully built and tested in the nightly builds. If you want to build a particular release, use such a tag for checking out.
- merge_DATE: marks points on branches where the side branch was merged into the main branch.

The repository contains two modules, Eclipse (the full sources) and Tests (the test suite).

SourceForge CVS – pserver read-only access

There are two ways to access the sourceforge CVS server. The simpler one is anonymous pserver-access, which is read-only, but less secure and might not work from behind a firewall. Set and export

```
      CVSROOT=:pserver:anonymous:@eclipse-clp.cvs.sourceforge.net:/cvsroot/
eclipse-clp
```

Then use cvs commands as above. Note that you cannot commit any changes!

SourceForge CVS – ssh developer access

First, register with the eclispse-clp sourceforge project:

1. You need to have a SourceForge user name: go to http://sourceforge.net and create an account, say *jsmith*
2. You must be registered as a developer with the eclipse-clp project. Contact one of the project admins (jschimpf, kish_shen, andy_cheadle, etc) and ask to be added as a developer.

Generate an ssh key pair for access to sourceforge (in the following, replace *john* with the user name on your home machine, and *jsmith* with your sourceforge user name):

```
    % ssh-keygen -t dsa -C "john@home"
    Generating public/private dsa key pair.
    Enter file in which to save the key (/homes/john/.ssh/id_dsa):
/homes/john/.ssh/id_dsa_sf
    Enter passphrase (empty for no passphrase):
    Enter same passphrase again:
    Your identification has been saved in /homes/john/.ssh/id_dsa_sf.
    Your public key has been saved in /homes/john/.ssh/id_dsa_sf.pub.
    The key fingerprint is: …
```

Submit the contents of `.ssh/id_dsa_sf.pub` to sourceforge via the Account Maintenance page (Log In, Account Options, Edit SSH Keys for Shell/CVS). You need to wait a while for this to become effective, an hour or so.
If you want to avoid having to type your passphrase all the time, start an ssh-agent on your home machine, and add the key you generated above:

```
    ssh-add /homes/john/.ssh/id_dsa_sf
```

Just to check whether password-less ssh is now working, try to log in to the SourceForge shell server (you should be logged in without password or passphrase prompt):

```
    ssh jsmith@shell.sf.net
```

For Cvs access set:

```
    CVS_RSH=ssh
    CVSROOT=:ext:jsmith@eclipse-clp.cvs.sourceforge.net:/cvsroot/eclipse-clp
    export CSV_RSH CVSROOT
```

You can then use cvs and tkcvs as above. E.g. to check out the latest development sources:

```
    cvs checkout Eclipse
```

Third-Party Software Components

As opposed to the tools above, the following are components that ECLiPSe *includes* or *interfaces to*. They must be available at ECLiPSe build time to provide include files or libraries to link against. However, a very basic core of ECLiPSe can be built without any of these components.

Note on cross-compiling for i386_nt If you are cross-compiling ECLiPSe for Windows, you will probably need to cross-compile the third-party components as well. In this case, the configure for each component needs to know it is configuring for a cross-compile. A CONFIG_SITE file needs to be defined, specifying where the various tools are. For example, if your mingw cross-tool is installed in <ECLIPSETOOLS>/i386_linux/cross-tools::

```
CC=$ECLIPSETOOLS/i386_linux/cross-tools/bin/i386-mingw32msvc-gcc

CXX=$ECLIPSETOOLS/i386_linux/cross-tools/bin/i386-mingw32msvc-c++

LD=$ECLIPSETOOLS/i386_linux/cross-tools/bin/i386-mingw32msvc-ld

AR=$ECLIPSETOOLS/i386_linux/cross-tools/bin/i386-mingw32msvc-ar

AS=$ECLIPSETOOLS/i386_linux/cross-tools/bin/i386-mingw32msvc-as

NM=$ECLIPSETOOLS/i386_linux/cross-tools/bin/i386-mingw32msvc-nm

STRIP=$ECLIPSETOOLS/i386_linux/cross-tools/bin/i386-mingw32msvc-strip

RANLIB=$ECLIPSETOOLS/i386_linux/cross-tools/bin/i386-mingw32msvc-ranlib

DLLTOOL=$ECLIPSETOOLS/i386_linux/cross-tools/bin/i386-mingw32msvc-dlltool

OBJDUMP=$ECLIPSETOOLS/i386_linux/cross-tools/bin/i386-mingw32msvc-objdump

RESCOMP=$ECLIPSETOOLS/i386_linux/cross-tools/bin/i386-mingw32msvc-windres

OS_INCLUDES="-isystem $ECLIPSETOOLS/i386_linux/cross-tools/i386-mingw32msvc/include"

ac_cv_func_select='yes'

ac_cv_func_gethostname='yes'

ac_cv_func_getpagesize='yes'
```

you can then set CONFIG_SITE environment variable to the file with the above information, and then run configure with the options

```
configure –build=i586-pc-linux-gnu –host=i386-mingw32msvc …
```

Gmp

This is the Gnu Multi-precision Package (www.swox.com/gmp) and is used to implement ECLiPSe's bignum and rational arithmetic. ECLiPSe can be built without it, but then integers are limited to 32 or 64 bits, and the rational number type is not available. ECLiPSe 5.10 requires GMP 4.1. The configure script can find this either in a standard location, or under $ECLIPSETHIRDPARTY/gmp4.1.

GMP 4.2 can also be used with ECLiPSe. This have been tested on Intel Mac OS X. However, some of the conversion from rationals to floats gives very slightly different results from 4.1 (e.g. float(1_10) gives 0.09999999.. instead of 0.1).

Intel Mac OS X specific notes:

Note this applies only to Intel and not PPC Macs. Unfortunately, Intel Mac OS X is not officially supported by GMP. In order to compile GMP with some assembler support for the critical routines, the source have to be modified:

Untar GMP, and go into the gmp source directory, and do the following:

```
cd mpn/x86
rm *dive_1* */*dive_1* */*/*dive_1*
rm */*mode1o* */*/*mode1o*
```

this can then be followed by the normal configure and make.

The .dylib version of the library may also need to be manually generated, as the build process apparently does not produce this automatically.

Java

Java must be installed on a machine in order to build ECLiPSe's Java interface, the JRE must be installed in order to run ECLiPSe with the Java interface. The version should be at least 1.4.

- The configure script tries to find Java in $ECLIPSETHIRDPARTY/java/$ARCH/java or otherwise in a number of standard locations. To specify a different location, either set the JAVA_HOME environment variable, or give a --with-java=<location> argument to the configure script.
- Set JRE_HOME environment variable before testing ECLiPSe. On Mac OS X, this should be set to the same path as JAVA_HOME.

Tcl/Tk

Tcl/Tk (www.tcl.tk) must be installed to build the Tcl/Tk interface. The recommended version is 8.5. The configure script looks for the installation under $ECLIPSETHIRDPARTY/tcltk<version>/$ARCH

with subdirectories include/ (for tcl.h etc) and lib/ (for shared libs and tcl subdirectory). These are accessed for building the tcl interface, and by the PACK script to create the tcltk-package for the distribution.

For Mac OS X, two Tcl/Tk interfaces can be built: an X11 version like other Unix/X11 systems, and a version using the native Aqua Graphical User Interface. X11 is provided with Mac OS X since 10.3, although it may not be installed on the system by default. It can be installed from the Mac OS X System CDs.

Tcl/Tk 8.4 or later should be built for both X11 and Aqua to allow both versions of the interface to be built. The Aqua version is provided with recent versions of Mac OS X, so you may not need to build it, but it can be built in the macosx subdirectory of both Tcl and Tk source, while the X11 can be built in the unix subdirectory – you may need to specify Mac OS X specific flags, and to build without framework, and also the “--with-x11” flag during configure. The X11 wish should have an alias xwish to distinguish it from the Aqua wish (this is what the ECLiPSe RUNME script looks for).

COIN-OR OSI

The eplex library can be interfaced to the open-sourced Open Solver Interface (OSI) of the COIN-OR project (www.coin-or.org), which provides access to various Mathematical Programming solvers, including open-sourced solvers from the COIN-OR project.

To build eplex for OSI, you need to first compile OSI, and the solvers you intend to use via OSI. For solvers from the COIN-OR project, both OSI and the solver can be downloaded as one source tree. :

Currently, the sources are configured to build OSI eplex using COIN-OR's CLP/CBC solvers, and SYMPHONY/CLP solvers. The files required for the eplex build should be placed in

$ECLIPSETHIRDPARTY/coin<version><OSIBUILD>/$ARCH

with the object files in lib subdirectory, and header files in include subdirectory.

<version> and <OSIBUILD> are configure script variables defined in ECLiPSe's toplevel configure script. <version> defines the solver(s) that are being interfaced to, and can be supplied by the user as part of the '--with_osi' option when running configure, By default, the script will look for clpcbc (for the CLP/CBC solvers) and symclp (for the SYMPHONY/CLP solvers).

<OSIBUILD> is used to specify a `build' version in the directory name. This is used mainly to allow different versions of ECLiPSe to be built with different versions of the COIN-OR projects, for example, CLP with (or without) AMD support.

The configure script will look in

$ECLIPSETHIRDPARTY/coin<version><OSIBUILD>/$ARCH/lib

for a file with the name $version to determine if eplex using the solver(s) defined by <version> should be built or not, so this file needs to be added to the directory.

The different COIN-OR solvers share many components, but recent versions of the COIN-OR projects no longer share a single source tree for these components, because these components may no longer be completely compatible between the different projects. This is why configure looks in distinct directories when building eplex for the different solvers, and it is strongly recommended that the different projects be built from their own source directories -- this may present problems if different versions of the shared components (in the form of dynamic load libraries in the third-party directories) are required for the different projects, and so it is best to build the static versions of the libraries and link these into the eplex library code.

The COIN-OR solvers can be built from a single source tree, with all the component COIN-OR projects required, with the exception that some (optional) third-party components, such as AMD, which needs to be built separately. For CBC/SYMPHONY, the CLP solver will be the default linear solver used, and will be downloaded with the rest of source-tree. .

Download the source for the COIN-OR solver you want to build for from the project page for the project -- follow the links from www.coin-or.org. For CBC, the trunk branch (instead of the the default stable branch) is required by eplex, as features not found in the stable branch is used by eplex. To obtain this, you may need to download directly from the version-controlled source tree directly. Currently, Subversion (SVN) is the version control tool used, and to download the source, you need to have Subversion installed on your system, and issue the following command:

svn co https://projects.coin-or.org/svn/Cbc/trunk <CBC>

where <CBC> is your local name for the directory where the CBC source tree will be downloaded to. Note that this will download all the required COIN-OR projects, including CLP and OSI. [See later for building CLP with AMD support]

For ease of handling builds for multiple architectures, we suggest you create a subdirectory in the CBC/SYMPPHONY toplevel directory, and configure and build from there. We also suggest generating the static .a libraries, as there are less problems linking these files. So do the following to build the solver from source::

```
cd <COIN>
mkdir i386_linux
cd i386_linux
../configure --enable-static
make
make install
```

where COIN is the toplevel directory of your COIN source, and i386_linux is the name of the platform you want to build for.

The configure script may try to look for a FORTRAN compiler and this may cause configure to fail if it cannot find the compiler. A FORTRAN compiler is not needed for building CBC/SYMPHONY, rerun configure with a 'F77=unavailable' flag if this occurs:

```
../configure --enable-static F77=unavailable
```

Important note for users of gcc 3: gcc version 3 is apparently unable to correctly compile recent (around 2008 onwards) Coin libraries with the default settings, because of bugs in the optimiser: the generated library will seg fault with even the simplest MIP problem with CBC/CLP, and can also return incorrect results. To compile COIN correctly with gcc 3, additional flags should be supplied with configure:

```
OPT_CXXFLAGS="-O2 -fno-omit-frame-pointer -momit-leaf-frame-pointer –DNDEBUG"
```

This should not be needed for gcc 4 (verified for 4.1+). Note that the COIN code built with gcc 4 will not load on a system with only a libstdc++.so from gcc 3 (in fact this seems to be even the case between different versions of gcc 4, e.g. compiled with 4.2 and loading in 4.1).

After building the Cbc and/or SYMPHONY projects, the library files in lib directory and the include files in include/coin directory should be copied to $ECLIPSETHIRDPARTY/coin/$ARCH/{lib,include/coin} so that ECLiPSe can find them when building eplex.

The clpcbc eplex requires the compilation of CbcBranchUser.cpp and CbcCompareUser.cpp from Cbc's example directory to be compiled and copied to the thirdparty directories:

```
cd Cbc/example

make CbcBranchUser.o
make CbcCompareUser.o
cp *.o $ECLIPSETHIRDPARTY/coin/$ARCH/lib
cp *User.hpp $THIRDPARTY/coin/$ARCH/include
```

Building eplex with other solvers available through COIN-OR's OSI interface should be straightforward, as long as all the functionality goes through OSI. This requires adding a few lines to coinplex.cpp in the icparc_solvers directory, and adding a new rule to the Makefile. The file coinplex.cpp has the code for building eplex for GNU's GLPK solver as an example for how the modification can be done. However, to fully exploit the solver's capabilities, you may need to add specialised code for the solver, as is done (extensively) for CBC/CLP.

x86-64 Linux specific note: The compiler flag –fPIC is needed for some platforms (e.g. x86_64_linux) to generate code that can be relocated (needed when the COIN libraries are linked to the ECLiPSe eplex library). However, specifying that you want the static libraries during configure for COIN may disable this. In such cases, you may need to supply the flag explicitly during configure:

../configure –enable-static ADD_CXXFLAGS=”-fPIC” ADD_CFLAGS=”-fPIC”

Intel Mac OS X specific note: there may be some problems linking the static COIN libraries, due to the use of assert macros: this uses vsprint() and eprintf(), which are missing with the standard linking commands. Normally, assert macros are nops if compiled with the –DNDEBUG flag, but several source files in Clp/src explicitly undef NDEBUG.

The easiest way to work around this is to use the dynamic .dylib libraries instead, but the COIN OSI source can be modified by removing the undefs of NDEBUG in the source files in Clp/src, Cbc/src, Osi/src/OsiClp , Cgl/src.

Cross-compiling for Windows note: recent versions of CoinUtils source (needed by both Cbc and SYMPHONY) does not compile `out-of-the-box' if cross-compiling for Windows with MinGW. This is because the code has a macro for conditional compiling for Windows that fail with the cross-compiler. This test needs to be changed so that the cross-compiling is detected correctly. In the file `<Coin>/CoinUtils/src/CoinTime.hpp`, change the line

```
#if defined(_MSC_VER)
```

To

```
#if defined(_MSC_VER) || defined(_WIN32)
```

the `defined(_MSC_VER)` that should be changed is the one for the definition of `CoinGetTimeOfDay().`

Compiling CLP with AMD support

The CLP/CBC solver code in eplex supports the use of the Interior Point/Barrier solver of CLP from ECLiPSe 5.11 onwards. This solver requires code to support Cholesky factorisation; and the default code provided with CLP is not very efficient. External code, such as University of Florida's Approximate Minimum Degree (AMD) ordering code, are much more efficient, and our tests have shown that the barrier solver can be more than 2 order of magnitudes faster with the AMD code than without.

With recent versions of CLP, the following steps are needed to compile CLP with AMD support:

- The AMD library has to be compiled. This code is available from http://www.cise.ufl.edu/research/sparse/amd
- The file <Coin-Cbc>/Clp/src/Makefile.in needs to be modified:
    - Add `ClpCholeskyUfl.hpp` to the files specified in `includecoin_HEADERS:`

```
includecoin_HEADERS = \
        ../inc/config_clp.h \
        Clp_C_Interface.h \
        ClpCholeskyUfl.hpp \
        ClpCholeskyBase.hpp \
```

    - Add `-DUFL_BARRIER` to the `DEFS` definition:

      ```
      DEFS = @DEFS@ -DUFL_BARRIER
      ```

    - Modify ADDLIBS to include the libamd.a generated by compiling the AMD library, e.g.:

      ```
      ADDLIBS = @ADDLIBS@ -L<AMD> -lamd
      ```

      where <AMD> is the path to where the libamd.a file for the platform is.

- Copy amd.h (from <AMD>/Include) and UFconfig.h (from the UFconfig package used by UFL AMD -- <AMD>/../UFconfig) to <Coin-Cbc>/Clp/src.
- Configure and build Cbc/Clp. Copy the resulting libraries, *including* libamd.a to the third-party directory for use with ECLiPSe as described above. The configure script for ECLiPSe determines if eplex_cbcclp code should be compiled with AMD by checking for libamd.a in the Coin-Cbc third-party directory.

CPLEX

Needed to build an eplex-library that interfaces to the ILOG Cplex solver (www.ilog.com). Unpack the original directory structure in $ECLIPSETHIRDPARTY/cplex<version>, which has architecture-dependent subdirectories. These are named differently from the ECLiPSe conventions, and also seem to change in different versions of CPLEX, select one appropriate directory for the platform you want to build, and rename the directory according to ECLiPSe convention, e.g. for i386_linux platform and CPLEX 11.0, rename the Linux directory in cplex110/lib/(x86_rhel4.0_3.4) to i386_linux.

For Windows, CPLEX is installed by an installer, which installs (by default) the CPLEX files into C:\ILOG\CPLEEX<version>. Some of these files/subdirectory need to be copied to $ECLIPSETHIRDPARTY/cplex<version>:

CPLEX	ECLiPSETHIRDPARTY/cplex<version>
bin\<cplexplatform>\cplex<version>.dll	bin/i386_nt/cplex<version>.dll
include	include
lib\<platform>\<type>*.lib	lib/i386_nt/*.lib

To build the eplex library with a version of CPLEX, the parameters for that version must be defined in the file eplex_params.h in icparc_solvers directory of ECLiPSe. See that file for instructions to add new parameters to the file if it is not already defined. A new rule to build the new version would also need to be added to the Makefile.

Xpress-MP

Needed to build an eplex-library that interfaces to Dash Optimization's XPRESS-MP solver (www.dashoptimization.com). Set up under $ECLIPSETHIRDPARTY/xosl<version> a structure with ARCH subdirectories for the different architectures and unpack the Xpress-MP versions there.

To build the eplex library with a version of XPRESS, the parameters for that version must be defined in the file eplex_params.h in icparc_solvers directory of ECLiPSe. See that file for instructions to add new parameters to the file if it is not already defined. A new rule to build the new version would also need to be added to the Makefile.

Gap

Gap is a free system for computational algebra (www.gap-system.org). ECLiPSe has an interface to it, and it is used for symmetry breaking. Gap is not needed for building Eclipse, only for running tests of the symmetry library. To make this work, the **gap** executable must be in the PATH. Unpack in $EXECPREFIX/lib and do

```
configure
make
cp bin/gap.sh ../../bin/gap
```

FlexLM

Needed to build an interface to the FlexLM licence manager. The configure script will look in $ECLIPSETHIRDPARTY/flexlm<version>/$ARCH for the corresponding include and library files. The last version known to work is 7.2.

Grappa

This is a Java library, required for building ECLiPSe's visualisation tools. The configure script will look in $ECLIPSETHIRDPARTY/grappa<version> to find the jar file. The version required is 1.2. Can be found via www.graphviz.org .

Graphviz

Graphviz 1.10 (www.graphviz.org) is required to run the ECLiPSe Visualisation tools. It is not strictly required for building those tools, since it is sufficient that the graphviz executables are installed on the machine where ECLiPSe finally runs. However, if a graphviz distribution is provided in $ECLIPSETHIRDPARTY/graphviz<version>/$ARCH during build, then the contents of this directory will be bundled with the ECLiPSe distribution.

MySQL

MySQL 5.0 is needed to build lib(dbi), an ECLiPSe interface to the MySQL database. The configure script will try to find include and library files in $ECLIPSETHIRDPARTY/mysql<version>/$ARCH/{include/lib}.

They can be downloaded from http://dev/mysql.com/downloads .

For cross-compiling to Windows, a libmysql.dll.a has to be created from libmysql.dll. See the

Makefile.in in Oci subdirectory of the ECLiPSe source for instructions on how to do this.

Manual Build

To build ECLiPSe from source

- Check out the module *Eclipse* from the repository or get it from a source distribution (see Getting The Sources), and *cd* there.
- Make sure you have the necessary tools installed and in your PATH.
- Set up a directory of third party components if you want any of the related functionality. Set ECLIPSETHIRDPARTY to point to it.
- Set ARCH to the ECLiPSe architecture id you want to build for.
- configure and make.

You should have the following environment settings:

```
export ARCH=<architecture to build for>
```

this is the ECLiPSe architecture name as explained earlier in this document.

```
export ECLIPSETHIRDPARTY=/vol/Eclipse/thirdparty
```

this points to the location of third-party components that ECLiPSe builds interfaces for, e.g. gmp, cplex, xpress-mp, flexlm, graphviz, grappa. Configure will detect the presence of components there and build interfaces if possible.

Simple build from toplevel directory (e.g. i386_linux, sparc_sunos5):

```
CONFIG_SITE=config.$ARCH ./configure
make -f Makefile.$ARCH
```

To build on multi-architecture machine (e.g. 64 bit build on 32/64 bit solaris), the configure scripts needs to be told what the target architecture is:

```
export ARCH=x64_64_sunos5
CONFIG_SITE=config.$ARCH ./configure --build=x86_64-pc-solaris2.9
make -f Makefile.$ARCH
```

Similar for a cross build, e.g. for Windows:

```
export ARCH=i386_nt
CONFIG_SITE=config.$ARCH ./configure --host=i386-mingw32msvc
make -f Makefile.$ARCH
```

The documentation is a separate build target, and is created by invoking

```
make -f Makefile.$ARCH install_documents
```

ECLiPSe consists of several subsystems in subdirectories of their own. Each of them has its own Makefile or Makefile.$ARCH in the subdirectory and can in principle be built separately, but they are all configured via the toplevel configure script.

The results of the build go into new subdirectories *lib, lib_xxx, include, doc* and *bin*. These directories comprise the binary ECLiPSe distribution.

Configuration options

The configure script accepts the following options to control inclusion of third-party components / interfaces:

`--with-osi[=versions]`

> Build ECLiPSe interface to COIN-OR Osi. You can optionally specify a version list, e.g. "clpcbc sym". Default is to build all available versions.

`--with-cplex[=versions]`

> Build ECLiPSe interface to ILOG Cplex. You can optionally specify a version list, e.g. "81 90". Default is to build all available versions.

`--with-xpress[=versions]`

> Build ECLiPSe interface to Dash Xpress-MP. You can optionally specify a version list, e.g. "1427 1525". Default is to build all available versions.

`--with-java[=dir]`

> Build ECLiPSe/Java interface.If dir is given, this is taken as the Java installation to use, otherwise the value of a JAVA_HOME environment variable is used, otherwise a number of standard locations is searched for a Java installation. Default:yes

`--with-flexlm[=version]`

> Build ECLiPSe interface to FlexLM. You can optionally specify a version preference list. Default is to build first available.

`--with-graphviz[=versions]`

> Copy graphviz executables into the installation. You can optionally specify a version list, e.g. "1.10". Default is to use the first available version.

`--with-mysql[=versions]`

> Build ECLiPSe database interface. You can optionally specify a version list, e.g. "50". Default is to build the first available version.

Note that some of the standard GNU configure options, such as `'--bindir',` are not supported in the ECLiPSe configure script for historical reasons, even though they are listed when configure is run with '`--help'` option.

Adding Contributions to ECLiPSe

To add contributions to ECLiPSe, a directory for the source needs to be selected. The most likely location for this is the Contrib directory, which is where contributions to ECLiPSe are normally added. When ECLiPSe is built, the files are copied from their source locations to various library directories, as determined by the various Makefile in each source directory. Files in Contrib directory are copied to the lib_public directory, as specified by the Makefile. The platform specific Makefiles (e.g. Makefile.i386_linux) is generated from Makefile.in during configure, and Makefile.in needs to be modified so that the new files can be treated correctly. Generally, ECLiPSe source (.ecl) or compiled (.eco) files are copied to platform independent directory, whereas platform specific files (e.g. objective files, compiled from languages such as C). are placed into platform specific directories within these library directories.

The ECLiPSe binary distributions are produced by copying the files in the library directories after building ECLiPSe. This is done by the PACK script in ECLiPSe's top-level directory; this script may need to be modified so that the new contributions are correctly copied, but all the files in the lib_public directory will be copied and should not require modification of the PACK script.

Manual Test

To run the test suite

- Check out the module *Tests* from the repository and *cd* there
- Invoke *./test_eclipse <name_of_eclipse_executable>* to run all tests
- Optionally use e.g. *–test-flags <flags>* to exclude certain tests (*no_eplex, no_tcl, no_gap*) or *–only <subdir>* to run only test in one subdirectory

Test failures will be printed, a full log goes into *testlog.$ARCH*. Further details can be found in the *README* file.

Automated Build-and-Test

Automated build-and-test is controlled by the shell script *BUILD_ROTD*. Copy this script from the top level directory of a checked-out Eclipse source tree to a location of your choice. Copy also the configuration file *site_info*. Edit *site_info* to reflect the machine names and path names at your site. Build-and-test is then performed by running the *BUILD_ROTD* script. Call *BUILD_ROTD -?* to get documentation about the available options.

The script supports builds for multiple architectures on multiple machines. In release 5.9,

- the file system where the build is performed must be shared between all Unix machines involved
- Window versions are cross-built on a Linux machine, and Windows test machines can be remote with a separate file system
- The main build machine (the one where the BUILD_ROTD script is run) must be able to perform password-less *ssh* to all other machines involved, including Windows. For that purpose, the Windows machines should have *cygwin* installed, including *ssh*, and *rsync* (but note that *cygwin* is only needed for the test system, not for installing or running ECLiPSe)

Architecture-specific Hints

Mac OS X on Power PC/Intel

The PPC and Intel Mac OS X are treated as separate platforms. For Power PC Macs, ARCH is set to ppc_macosx, and for Intel Macs, ARCH is set to i386_macosx.

Mac OS X has some differences with other Unix. For the build, you may need to set DYLD_LIBRARY_PATH (instead of the more standard LD_LIBRARY_PATH) so that ECLiPSe can find the dynamic libraries.

There are also some subtle differences in how files are linked. In particular, standard C++ and gcc libraries may need to be linked in, especially if static .a files are used. The exact libraries to link in depends on the version of Mac OS X.

10.2: -lgcc (and/or –lstdc++ if linking static libraries generated from C++)

10.3: -lcc_dynamic (and/or –lstdc++) **UNTESTED – information from web

10.4 –L/usr/lib/gcc/i686*/* -lgcc (and/or –lstdc++)

-L/usr/lib/gcc/powerpc*/* -lgcc (and/or –lstdc++)

Where */* are version specific and may vary with different versions of 10.4 – check your copy for the exact path.

Currently, the configure files are configured to compile on 10.4. It may need to be changed if you are compiling with earlier versions of Mac OS X.

Solaris 10 on Intel x86 and x86_64

ECLiPSe can be built on the free Solaris 10 system enhanced with open-source tools. On an AMD 64-bit machine, both 32-bit and 64-bit versions can be built (the corresponding ECLiPSe architecture names are i386_sunos5 and x86_64_sunos5). This flexibility requires some care to make sure that the correct tool executables and libraries get picked up. We recommend the following setup:

- Create $ECLIPSETOOLS and make sure its bin directories are in the path before the /usr/sfw and other system directories

- Get and install Gnu tar as gtar. Create a symbolic link $ECLIPSETOOLS/bin/tar to gtar to make sure it is preferred over Solaris tar (which cannot handle long pathnames and does not have a compress option).
- Get the gcc compiler package from sunfreeware.com (tested version is 3.4) and install in standard location /usr/sfw. This uses the gcc compiler, the gas assembler and the native Solaris linker.
- Create a script $ECLIPSETOOLS/i386_sunos5/bin/gcc
  ```
  #! /bin/sh
  exec /usr/sfw/bin/gcc -m32 “$@”
  ```
  and a script $ECLIPSETOOLS/ x86_64_sunos5/bin/gcc
  ```
  #! /bin/sh
  exec /usr/sfw/bin/gcc -m64 “$@”
  ```
  This ensures that gcc creates code corresponding to the setting of $ARCH
- LD_LIBRARY_PATH may have to be set to /usr/sfw/lib/64:/usr/sfw/lib:...

For the third-party components:

- Install Sun Java, both the 32 bit version and the 64 bit extension, e.g. as /usr/java. Then create a directory $ECLIPSETOOLS/ x86_64_sunos5/jre containing symbolic links
  ```
  bin -> /usr/java/bin/amd64
   lib -> /usr/java/lib
  ```
  and for 64-bit builds set JRE_HOME=$ECLIPSETOOLS/ x86_64_sunos5/jre. For 32-bit builds JRE_HOME can be set simply to /usr/java/jre
- Build separate 32 and 64 bit versions for tcl/tk 8.4 from source
- For gap and graphviz, the 32 bit versions can be used with both 32 and 64-bit ECLiPSe.
- For building COIN projects, add ‘-m64 –fPIC’ to CFLAGS and ‘-m64’ to LDFLAGS

Windows

From ECLiPSe 6.0 onwards, the binary will not execute on Windows version that is older than NT 4.0 (e.g. the original Windows NT, Windows 95). It is unlikely that there are many systems that still use these old version of Windows, but ECLiPSe can be compiled to run on such systems – this requires telling the compilation that the target is older than Windows NT 4.0, by defining the _WIN32_WINNT to be less than 0x400. For cross-compiling, this macro is set in the toplevel configure.

Pitfalls

Here is a list of minor problems that have occured:

- On some machines, mktemp command does not understand –t option. Install new Gnu mktemp
- When manually checking out a main branch from CVS, some files will be missing (these are files that were once added on a branch). This is a cvs bug that can be worked around by calling: cvs update –D now; cvs update –A
- If a non-GNU tar is used to untar the archive, some of the longer file/path names can become corrupted (for example, in the Java source files in the Visualisation directory). If this happens, please use GNU tar if possible, or rename the files manually if not.
- If using Cygwin on Windows, make sure your Cygwin is configured to use Unix-style new lines. Otherwise the various scripts will not run correctly and produce many errors.

We have experienced problems with the Windows Java Interface (ec_java.dll) when it is cross-built on Linux with IBM-Java. Use Sun Java.

Using the SourceForge Compile Farm

SourceForge have withdrawn the Compile Farm service in Feb 2007, so this is no longer available.

Hints for Porting to New Platforms

Porting to other Unix or Unix-like platforms should be relatively straightforward, if tools needed to build ECLiPSe are available for the platform.

Define a new name for ARCH to represent the platform – this should be in the form of <machine type>_<ostype>. This should be added to the configure/configure.ac files at the toplevel directory, along with any platform specific information that cannot be determined by the configure script.

A new file Platform_<ARCH>.java need to be added and Platform.java modified in

JavaInterface/src/com/parctechnologies/eclipse

to return the new platform type to the JavaInterface (see the other files to see what needs to be done).

eclipse_arch.tcl in lib_tcl needs to be modified to return the platform type for the Tcl interface.

If the Windowing system is not X11, more changes may be needed in tkeclipse.tcl (also in lib_tcl).

Some assembler code is used in Shm/mutex.c to implement locks, but this is not needed for the normal ECLiPSe, so the dummy locks can be used.

Some processor dependent code for controlling ECLiPSe arithmetic behaviour in the files sepia/include/rounding_control.h, sepia/src/bip_arith.c, sepia/src/emu.c
may need to be modified for new processor types.