Open BEAGLE Compilation HOWTO

Christian Gagné

Current affiliation:
Équipe TAO, INRIA Futurs,
LRI, Bat. 490, Université Paris Sud,
91405 Orsay Cedex, France.

Former affiliation:
Laboratoire de Vision et Systèmes Numériques (LVSN),
Département de Génie Électrique et de Génie Informatique,
Université Laval, Québec (QC), Canada, G1K 7P4.

E-mail: christian.gagne@lri.fr, cgagne@gmail.com

For Open BEAGLE version 3.0.1, document revision 1
Technical report RT-LVSN-2003-02-V301-R1

October 10, 2005

Contents

1 Introduction

This document is on the compilation of the Open BEAGLE[1] C++ framework for evolutionary computations. It first presents how to compile the libraries and examples of the framework on the two supported environments, that is most Unix environments supported by the GNU Autotools [1] (including Linux, Mac OS X, and Cygwin), and Microsoft Visual C++ .NET. The second part of the document explains how to set a project that is linked with the Open BEAGLE libraries, on supported platforms. Finally, known compilation problems and fixes are presented at the end of the document.

2 Building the libraries and examples

The first step toward the use of Open BEAGLE is, obviously, to compile the libraries. Once the libraries compiled, the first contact with the framework can be made by compiling some of the twelve examples provided, and then experimenting with them.

2.1 Building the libraries and examples on Unix

To build Open BEAGLE on Unix, uncompress the distribution, open a shell, `cd` in the root directory of the uncompressed distribution, and type the following commands.

```
./configure
make clean
make
```

This will configure and compile the libraries that compose the framework. Optionally, you can install the framework with the following command.

[1] http://beagle.gel.ulaval.ca

```
make install
```

By default, the compiled libraries are installed in folder `/usr/local/lib`, while the headers are installed in folder `/usr/local/include`. You can specify the installation folder by giving the option `--prefix=`PATH to the configure script.

`configure` recognizes the following options to control how it operates.

- `--prefix=`PATH
  Set the main installation folder (default: `/usr/local`).
- `--disable-shared`
  Disable the build of the shared framework libraries.
- `--disable-static`
  Disable the build of the static framework libraries.
- `--enable-optimization`
  Enable the "optimization" mode by setting the optimization flags of the compiler and disabling of some useful, but not essential, debugging statements. Use this flag to compile a version of the framework libraries that are linked with EC applications that are fully debugged.
- `--enable-full-debug`
  Enable the "full debug" mode of the code by enabling some validation and logging statements. Use this flag to compile a version of the framework for special debugging requirements.
- `--cache-file=`FILE
  Use and save the test results in file FILE instead of `./config.cache`. Set FILE to `/dev/null` to disable caching, for debugging configure.
- `--help`
  Print a summary of the options to `configure`, and exit.
- `--quiet`
  `--silent`
  `-q`
  Do not print messages saying which checks are being made. To suppress all normal output, redirect it to `/dev/null` (any error messages will still be shown).
- `--version`
  Print the version of autoconf used to generate the `configure` script, and exit.

`configure` also accepts some other, not widely useful, options.

You can also compile the libraries with KDevelop[2]. In fact, KDevelop acts only as a graphical wrappers over the `configure` script and the makefiles. KDevelop is a part of the KDE project and is available for most modern Unix flavor. To compile the Open BEAGLE libraries, open file `beagle.kdevelop` in Open BEAGLE main directory (*Project* → *Open Project...*). Then you should execute the `configure` script (*Build* → *Run Configure*) and compile the libraries (*Build* → *Build Project*).

[2] http://kdevelop.org

Once the libraries compiled, the examples can now be compiled. Individual compilation setups have been made for each of the twelve examples. These examples are in directories `examples/GA`, `examples/GP` and `examples/Coev`. The building process is very similar to the one of the building process of the libraries. To do so type the following commands.

```
./configure
make clean
make
```

Then, the binary is in directory with the name of the example, i.e. for the `maxfct` example the binary is in the directory `examples/GA/maxfct/maxfct`.

The `configure` script of the examples recognizes the following options to control how it operates.

- `--enable-optimization`
  Enable the "optimization" mode by setting the optimization flags of the compiler. Use this flag to compile a version of the examples linked with an optimized version of the framework libraries.
- `--with-pacc-util=`DIR
  Specify installation path of PACC::Util (default: `../../../../PACC`).
- `--with-pacc-util-include=`DIR
  Specify exact dir for PACC::Util headers (default: `../../../../PACC`).
- `--with-pacc-util-libdir=`DIR
  Specify exact dir for PACC::Util libraries (default: `../../../../PACC/Util`).
- `--with-pacc-xml=`DIR
  Specify installation path of PACC::XML (default: `../../../../PACC`).
- `--with-pacc-xml-include=`DIR
  Specify exact dir for PACC::XML headers (default: `../../../../PACC`).
- `--with-pacc-xml-libdir=`DIR
  Specify exact dir for PACC::XML libraries (default: `../../../../PACC/XML`).
- `--with-pacc-math=`DIR
  Specify installation path of PACC::Math (default: `../../../../PACC`).
- `--with-pacc-math-include=`DIR
  Specify exact dir for PACC::Math headers (default: `../../../../PACC`).
- `--with-pacc-math-libdir=`DIR
  Specify exact dir for PACC::Math libraries (default: `../../../../PACC/Math`).
- `--with-beagle=`DIR
  Specify the installation path of Open BEAGLE (default: `../../../../beagle`).
- `--with-beagle-include=`DIR
  Specify exact installation directory for Open BEAGLE headers (default: `../../../../beagle/include`).

- `--with-beagle-GA-include=`DIR
  Specify exact installation directory for Open BEAGLE GA headers (default: `../../../../beagle/GA/include`).
- `--with-beagle-GP-include=`DIR
  Specify exact installation directory for Open BEAGLE GP headers (default: `../../../../beagle/GP/include`).
- `--with-beagle-libdir=`DIR
  Specify exact installation directory for Open BEAGLE base library (default: `../../../../beagle/src`).
- `--with-beagle-GA-libdir=`DIR
  Specify exact installation directory for Open BEAGLE GA library (default: `../../../../beagle/GA/src`).
- `--with-beagle-GP-libdir=`DIR
  Specify exact installation directory for Open BEAGLE GP library (default: `../../../../beagle/GP/src`).
- `--cache-file=`FILE
  Use and save the test results in file FILE instead of `./config.cache`. Set FILE to `/dev/null` to disable caching, for debugging `configure`.
- `--help`
  Print a summary of the options to `configure`, and exit.
- `--quiet`
  `--silent`
  `-q`
  Do not print messages saying which checks are being made. To suppress all normal output, redirect it to `/dev/null` (any error messages will still be shown).
- `--version`
  Print the version of autoconf used to generate the `configure` script, and exit.

As with the libraries, `configure` also accepts some other, not widely useful, options.

You can alternatively build your example using the KDevelop environment. As with the libraries, the KDevelop project file of the examples is configured to directly use the underlying autoconf / automake scripts and makefiles. To compile the Open BEAGLE examples with KDevelop, open the appropriate `.kdevelop` file in the example main directory (*Project → Open Project...*), execute the `configure` script (*Build → Run Configure*), and start the building process (*Build → Build Project*). You can run (*Build → Execute Program*) and even debug (*Debug → Start*) the examples in KDevelop.

Once some examples built, you can start experimenting with them. Take a look to the respective `README` file to get the usage of the binaries, and the details of the examples.

2.2 Building the libraries and examples with Microsoft Visual C++ .NET

Here is the step-by-step process to compile the libraries on Microsoft Visual C++ .NET 2003 (English version):

1. Open folder `beagle/include/beagle` with the Windows file manager. Copy (or rename) file `config.hpp.msvcpp` as file `config.hpp` (in the same folder).
2. Open the solution file `beagle.sln` in Microsoft Visual C++ .NET, This file is in the folder `MSVCPP` to the Open BEAGLE main directory.
3. Choose the desired compilation configuration in menu *Project* → *Properties*. Set the active configuration, between *Debug* and *Release*. I advice you to use the *Debug* configuration until your application is well debugged and you really need performance.
4. Compile the libraries with menu *Build* → *Build Solution*.

Actually, the solution file is configured only to compiled static libraries (`.lib` files). You must reconfigure the projects manually to build shared libraries (`.dll` files). I let the advanced users doing this by themselves.

By default, the examples are compiled along with the libraries. Individual solution files have also been made for each of the twelve examples. These examples are in directories `examples\GA` and `examples\GP`. Here is generic step-by-step directives on how to build an example.

1. Open the `.sln` file is are in folder `MSVCPP` of the example main directory.
2. Set the compilation configuration (menu *Project* → *Properties*) to the same profile used to compile the libraries.
3. Compile the libraries with menu *Build* → *Build Solution*.

Once the example compiled, you can execute the examples with menu *Debug* → *Start Without Debugging* or debug it with menu *Debug* → *Start*.

3 Building a new project

This section presents different ways to configure compilation projects of EC applications that use Open BEAGLE. The two first sections presents two different ways to proceed under Unix, that is with the GNU Autotools and KDevelop. The last section presents the configuration step necessary to develop a new Open BEAGLE application with Microsoft Visual C++ .NET on Windows.

3.1 Building a new project with the GNU Autotools

The GNU Autotools, that is autoconf, automake and libtool, are very powerful script-oriented tools to manage portable projects on most modern Unix flavors. Although I found this set of tools very interesting, I would not advice newcomers to use them for their projects, as the learning curve is very steep. I would rather advice newcomers to use KDevelop, that is explained in section 3.2. The more experimented users, not familiar with the GNU Autotools but that are motivated to use them, should start by reading the excellent book from Vaughman *et al.* [1] and the official manuals of autoconf [2], automake [3] and libtool [4].

It is not in my intention to give a step-by-step explanations on how to set-up a compilation project with the Autotools. I would rather give indications on how I set up my compilation projects in order to give some

tips on how to do it right at the first try. The Open BEAGLE have been configured as stand-alone projects configured with the Autotools. Let's take a look to the `configure.ac` file of the symbolic regression example (symbreg).

```
dnl Process this file with autoconf to produce a configure script.
AC_INIT([GP symbolic regression example],[3.0.0],[cgagne@gmail.com],[symbreg])
AC_CONFIG_SRCDIR([symbreg/SymbRegEvalOp.cpp])
AC_CONFIG_AUX_DIR([config])
AM_INIT_AUTOMAKE
AM_CONFIG_HEADER([symbreg/config.hpp:symbreg/config.hpp.in])

dnl Checks for programs.
AC_PROG_AWK
AC_PROG_CXX
AC_PROG_CXXCPP
AC_PROG_LN_S
AC_PROG_LIBTOOL

dnl Set language.
AC_LANG([C++])

dnl Checks for libraries.
AC_HEADER_STDC

dnl Check for zlib support.
CHECK_ZLIB

dnl Initialize PACC libraries with the given paths.
PACC_UTIL_INIT([../../../../PACC], [../../../../PACC], [../../../../PACC/Util])
PACC_XML_INIT([../../../../PACC], [../../../../PACC], [../../../../PACC/XML])
PACC_MATH_INIT([../../../../PACC], [../../../../PACC], [../../../../PACC/Math])

dnl Initialize Open BEAGLE with the given paths.
OB_BEAGLE_INIT([../../../../beagle],
               [../../../../beagle/include],
               [../../../../beagle/src],
               [../../../../beagle/GA/include],
               [../../../../beagle/GA/src],
               [../../../../beagle/GP/include],
               [../../../../beagle/GP/src],
               [../../../../beagle/Coev/include],
               [../../../../beagle/Coev/src])

dnl Create makefiles.
AC_CONFIG_FILES([Makefile symbreg/Makefile])
AC_OUTPUT
```

This file includes all the necessary settings for a plain C++ project with autoconf, with the addition of a call

to the `PACC_UTIL_INIT`, `PACC_XML_INIT`, `PACC_MATH_INIT`, and `OB_BEAGLE_INIT` macros. This macro sets all the compilation variable to the appropriate paths in order to find the headers and the libraries of PACC and Open BEAGLE at the compilation and link time. These macros are in file `acinclude.m4` of the projects. You can recuperate this macro for your projects if you want.

The macro `CHECK_ZLIB` is not standard in autoconf. It was taken from the autoconf macro archive[3] and copied into file `acinclude.m4`. This macro is pretty useful to dynamically locate installation directory of zlib[4], used to gzip/gunzip Open BEAGLE files. It also adds some nice command-line arguments to the `configure` script, in order to set manually where zlib are installed.

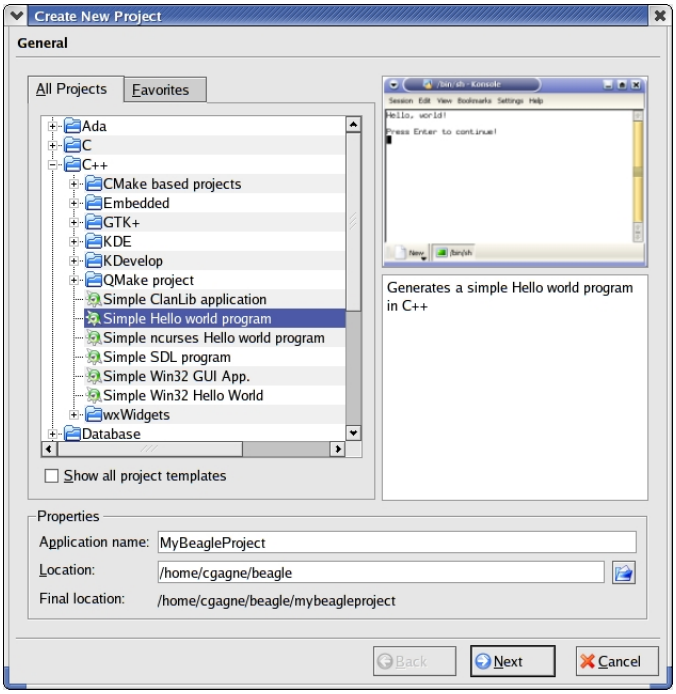
To generate the configure scripts and makefile, I use a script named `bootstrap` that call the appropriate Autotools commands. Execute it as needed when you modify the `configure.ac` and `Makefile.am` files. Take note that the projects are set-up with a directory named `config` in which several scripts goes. This is to keep the main directory as clean as possible. I also add a KDevelop project file (`.kdevelop` file) in the main directory. These projects are configured to directly use the underlying Autotools settings without modifying them. For the rest of the configuration of an Open BEAGLE project with the Autotools, take a look to the content of an example, they will give you as much details as you need.

3.2 Building a new project with KDevelop

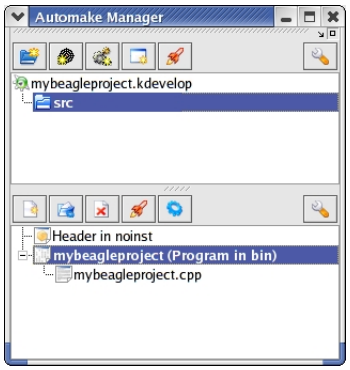
KDevelop is a nice Integrated Development Environment (IDE) for Unix-based systems. I use it for most of my developments tasks. The following text will explain the simplest way to set up a KDevelop project of a Open BEAGLE application. These explanations have been tested on KDevelop 3.4.2 under Fedora Core 3 (English localization).

First, let's create a new C++ terminal project (menu *Project* → *New Project...*). Select *C++* → *Simple Hello world program* project in the application wizard.

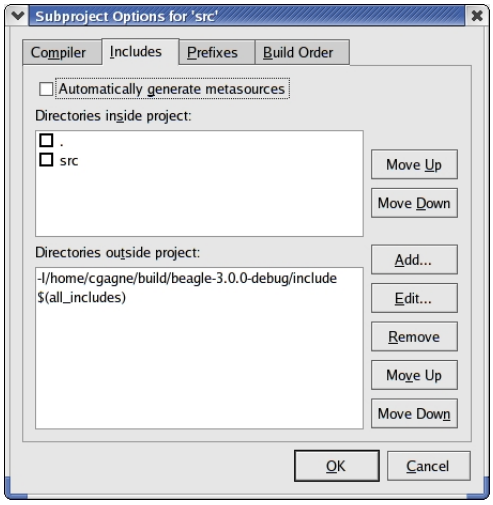
[3] http://www.gnu.org/software/ac-archive
[4] http://www.gzip.org/zlib



Set the other options as needed and then generate the project and exit the application wizard. You have now an empty C++ terminal project. The project must now be set in order to add the paths to the PACC and Open BEAGLE headers when compiling, and to give the paths to the libraries to the linker. To do so, open the *Automake Manager* window, which is usually activated by clicking on a button at the right of the KDevelop main window.

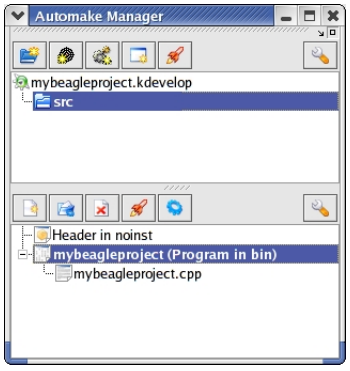


In the *Automake Manager* window, click on the `src` directory and them click on the *Option* button, which is represented by a wrench icon at the **upper right** of the window. Then go into the *Includes* tab and add the directory where the PACC and Open BEAGLE includes can be found using the *Add...* button.

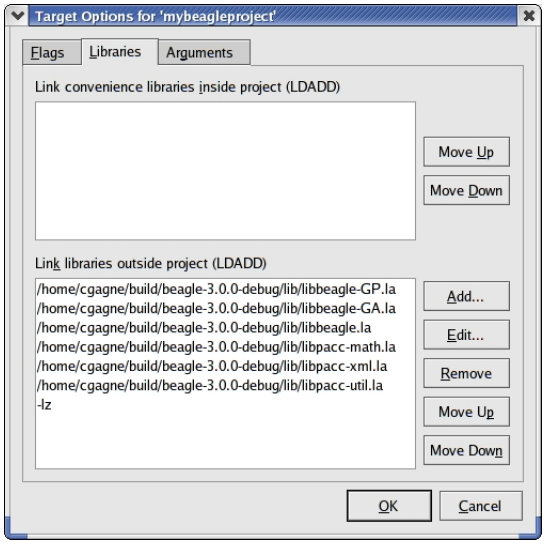


In my case, I installed PACC and Open BEAGLE in the directory `/home/cgagne/build/beagle-3.0.0-debug`, so the headers have been installed in directory `include`. Change `/home/cgagne/build/beagle-3.0.0-debug` for the installation prefix you used when you compiled and installed the Open BEAGLE libraries.

Now click on your project in the lower half of the *Automake Manager*, in my case *mybeagleproject (Program in bin)*, and then click on the *Option* button represented by a wrench icon at the **center right** of the window.



Then go into the *Libraries* tab and then add the absolute path where the the PACC and Open BEAGLE librairies can be found using the *Add...* button.

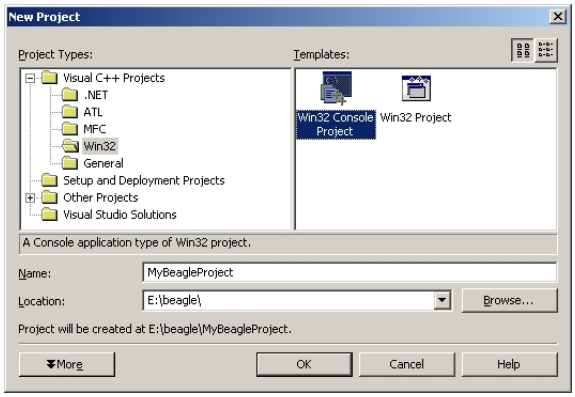


It is important to enumerate the libraries from the most dependent to the least dependent one. The libraries are linked to the binary following the order they are given in the option window. In our case, the dependencies are (from most dependent to the least): `libbeagle-GP.la`, `libbeagle-GA.la`, `libbeagle.la`, `libpacc-math.la`, `libpacc-xml.la`, and `libpacc-util.la`. You must also inform the linker to use zlib by giving the flag `-lz`. Don't forget that the installation prefix used here is `/home/cgagne/build/beagle-3.0.0-debug` and is probably different on your system.

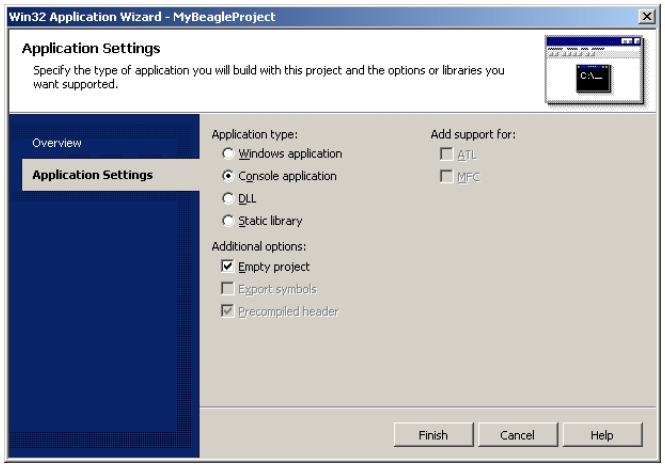
Then, your KDevelop project is ready to be compiled. If you have problem with your new setup, force clean-up of the project configure with menu *Build* → *DistClean*, followed by menu *Build* → *Run automake & friends*, menu *Build* → *Run configure*, and menu *Build* → *Build Project*.

3.3 Building a new project with Microsoft Visual C++ .NET

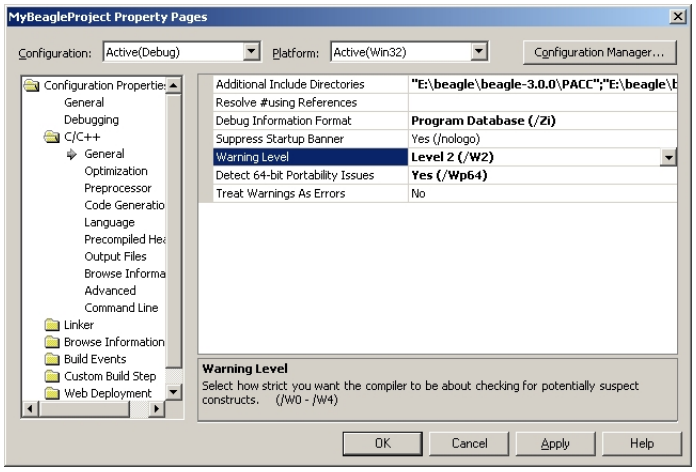
Now, let's discuss on a to setup a new Open BEAGLE project with Microsoft Visual C++ .NET. First, make a new project (menu *File* → *New* → *Project...*). Select *Visual C++ Projects* → *Win32 Project* → *Win32 Console Project*, give the location and the name of the project, and click on *OK*.



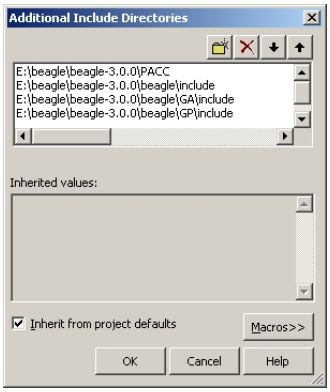
Then, click on the *Application Settings*, set *application type* to *Console application*, and select the *Empty project* option in *additional options*.



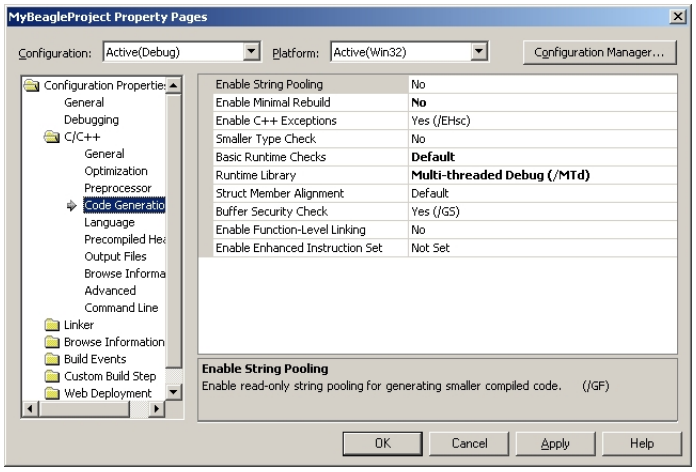
Click on *Finish* and a new Win32 console application project is generated. The project must be configured properly to find the Open BEAGLE headers. Go in menu *Project* → *Properties*, tab *C/C++* → *General*. Click on the field of *Additional Include Directories* and add the directories in which the PACC and Open BEAGLE headers are.



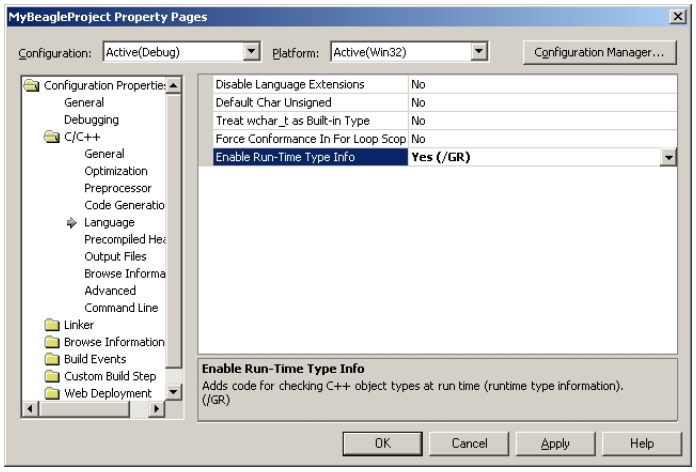
This can be easily done by clicking in the field, then clicking on the three dots, and finally adding the file with a dialog. In my case, the PACC and Open BEAGLE main installation directory is `E:\beagle\beagle-3.0.0`.



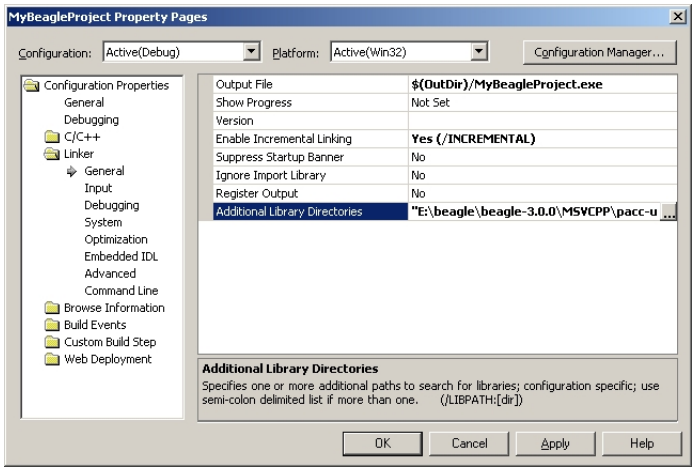
Open BEAGLE supports co-evolution based on a multi-threading model. Even if your application does not use co-evolution, you must select the multi-threaded runtime library for your application in tab *Code Generation*.



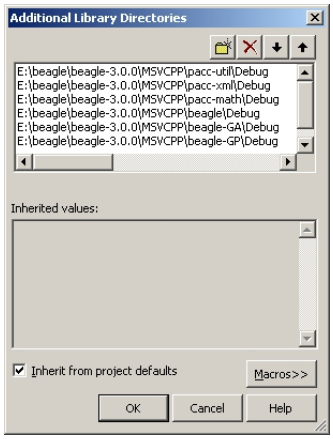
By default, Open BEAGLE makes use of Run-Time Type Information (RTTI). You must enable this feature for your project by going in tab *C/C++* → *Language*.



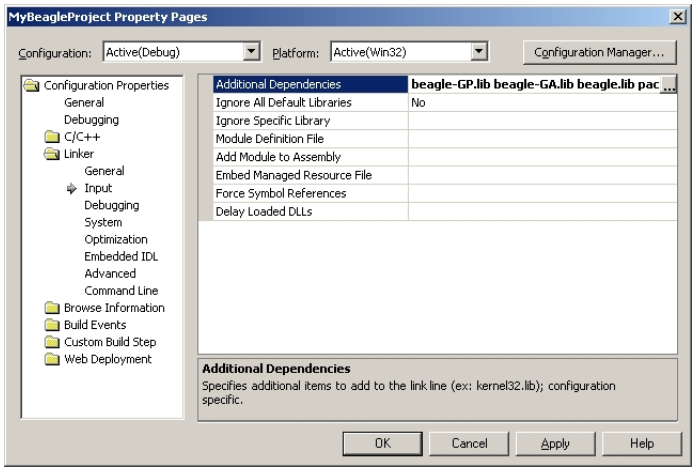
The project must also be linked with the PACC and Open BEAGLE libraries. Add the path to the libraries in the tab *Linker* → *General*, field *Additional Library Directories*



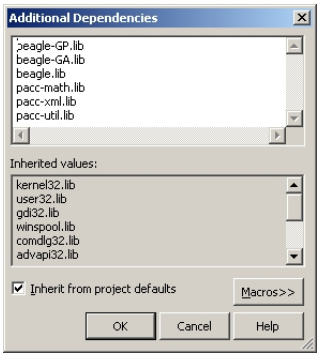
As with the includes, a friendly dialog allow you to select the directories with some click. To enter this dialog, click in the *Additional Library Directories* entry field and then click on the three dots.



And finally, the project must be set in order to be linked with the three PACC libraries (`pacc-util.lib`, `pacc-xml.lib`, and `pacc-math.lib`) and the three Open BEAGLE libraries (`beagle.lib`, `beagle-GA.lib`, and `beagle-GP.lib`). Go in the *Linker* → *Input* tab and add them to the *Additional Dependencies* field.



Another friendly dialog is also available to type the name of these six libraries.



Then, the Microsoft Visual .NET project is ready for developments. Enjoy!

4 Specific problems and solutions

4.1 `stringstream` into gcc $\leq$ 2.95 and egcs

`stringstream` are not available with default installations of the compilers gcc, version 2.95 and below, and egcs. This problem is addressed by the gcc FAQ at http://gcc.gnu.org/faq.html#2.95sstream. This fix has enabled the building of the Open BEAGLE framework with gcc 2.95.3 and egcs 1.1.2.

4.2 Problems with compiling the libraries and/or the examples on Unix

If you have problems with the compilation of the libraries and/or the examples on Unix, the first thing to try is to regenerate the configure script and the makefiles. First be sure that you have autoconf, automake, and libtool installed on your workstation. Then, execute the script `bootstrap` in the Open BEAGLE main directory in order to regenerate everything. It may be necessary to execute the script several time. When everything seems to be correctly regenerated, erase file `config.cache` in main directory, and repeat the compilation process of section 2.1. This may solve most of the problems that occurred during the building process on Unix.

4.3 Build a fully optimized binary under Unix

To build a fully optimized binary, several arguments must be given to the `configure` scripts. The first one is the `--enable-optimization`, which set the `BEAGLE_NDEBUG` preprocessor flag. Then some compiler-specific optimization flags must be given using the `CPPFLAGS` and `CXXFLAGS` variables. For gcc, a good combination of optimization flags for the `CXXFLAGS` variable is `-march=i686`[5], `-ffast-math` and `-O3`. The command-lines to compile with full optimization would then be:

```
./configure --enable-optimization CPPFLAGS=-DNDEBUG CXXFLAGS='-march=i686 -ffast-math -O3'
make clean
make
```

These command-lines must be used to both (re-)compile the Open BEAGLE libraries and your application code. A good way to proceed is to install the Open BEAGLE headers and libraries in a specific path different from the usual one used for developments, by assigning a different value with the `--prefix` argument.

In KDevelop, you can set the variables by going in the menu *Project* → *Project Options* → *Compiler Options*, tab *C++*, and put in the field *Compiler Flags (CXXFLAGS)* the compiler-specific values.

[5] This suppose that you are using a Pentium II or better. On other architectures, choose the appropriate flag supported by gcc.

4.4 Building a multi-threaded co-evolution application under Unix

When you want to build an Open BEAGLE multi-threaded co-evolutionary application, you must enable your application to use POSIX threads. On Linux this is done by giving the `-pthread` flag to the compiler and the linker. You must also take care to link on both the `beagle-Coev` and `pacc-threading` libraries provided by Open BEAGLE.

In KDevelop, you can set these compilation flags by going in the menu *Project* → *Project Options* → *Compiler Options*, tab *C++*, and put in the field *C++ Compiler Flags (CXXFLAGS)* `-pthread` option, and then going in tab *General* and put in field *Linker flags (LDFLAGS)* the same `-pthread` option.

If you use a custom autoconf project, you can automatically detect how to set up your application to use POSIX thread, by using the macro `ACX_PTHREAD` from the autoconf macro archive[6]. To get an example on how to use this macro, consults the `configure.ac` and `acinclude.m4` files of the `coev_symbreg` and `ipd` Open BEAGLE co-evolutionary examples.

4.5 Activating "optimization" or "full debug" mode on Microsoft Visual C++ .NET

To activate the "optimization" or "full debug" mode on Microsoft Visual C++ .NET as development environment, open file `beagle/include/beagle/config.hpp` in your favorite editor. To activate the "optimization" mode, change the line

```
/* #undef BEAGLE_NDEBUG */
```

by the following one

```
#define BEAGLE_NDEBUG
```

For the activation of the "full debug" mode, change the line

```
/* #undef BEAGLE_FULL_DEBUG */
```

by the following one

```
#define BEAGLE_FULL_DEBUG
```

4.6 Compiling with Microsoft Visual C++ .NET version 2002

Solution and project files for MS Visual C++ .NET are now generated using version 2003, in opposition to version 2002 which was used previously. Forward compatibility is not a practice at Microsoft, so you probably need to pass to 2003 to use the solution and project files provided. But, it is still possible to regenerate these files for MS Visual C++ .NET 2002. Sorry for the problems that this might cause to Windows users, but the Visual Studio .NET installation in my laptop is now 2003.

[6] http://www.gnu.org/software/ac-archive

4.7 Enable support for compressed files on Microsoft Visual C++ .NET

Support for gzipped files on Microsoft Visual C++ .NET is disabled by default. To enable it, you must first install the zlib[7] library. The *zlib compiled DLL version 1.2.1* installation package, available at the moment of writing this on zlib's Web site, has been tested successfully to work with Open BEAGLE. Once zlib installed, you must modify file `beagle/include/beagle/config.hpp` by uncommenting the following lines.

```
#ifndef BEAGLE_HAVE_LIBZ
#define BEAGLE_HAVE_LIBZ 1
#endif
```

Defining the preprocessor flag `BEAGLE_HAVE_LIBZ` enables support for gzipped file in Open BEAGLE. Then, you should modify the Open BEAGLE libraries projects to properly include zlib headers. And finally, you should modify Open BEAGLE examples and your application projects in order to include zlib headers and to link on zlib library.

4.8 Pre-compiled headers with gcc $\geq$ 3.4

Advanced users might be interested to know that **experimental** support for pre-compiled headers (available only with gcc $\geq$ 3.4) is now provided in the Open BEAGLE build system. To enable pre-compiled headers, only add a call to `make pch` during the compilation of Open BEAGLE, that is:

```
./configure
make clean
make pch
make
```

This will generate `.gch` files for the four Open BEAGLE libraries. It has been observed that enabling pre-compiled headers reduced by about 40% the compilation time on my laptop. But unfortunately, using pre-compiled header might cause some silly compilation problems, for example if you try to include headers that are present in several distinct pre-compiled headers. In some circumstances, gcc might refuse to complete the compilation of your program. In such cases, re-compile your application without pre-compiled headers!

References

[1] Gary V. Vaughan, David MacKenzie, Tom Tromey and Ben Elliston, *GNU Autoconf, Automake, and Libtool*, New Riders Press, October 2000, Available on-line at http://sources.redhat.com/autobook.

[2] David MacKenzie, Ben Elliston and Akim Demaille, *Autoconf*, Free Software Foundation, December 2002, Available on-line at http://www.gnu.org/manual/autoconf.

[3] David MacKenzie and Tom Tromey, *GNU Automake*, Free Software Foundation, December 2002, Available on-line at http://www.gnu.org/manual/automake.

[7] http://www.gzip.org/zlib

[4] Gordon Matzigkeit, Alexandre Oliva, Thomas Tanner, and Gary V. Vaughan, *GNU Libtool*, Free Software Foundation, September 2001, Available on-line at http://www.gnu.org/manual/libtool.