Installing ODB with `build2`

The latest ODB compiler and runtime libraries (`libodb*`) can now be built and installed on all the major platforms (Linux, Windows, and Mac OS) from source packages available in the `build2` cppget.org repository. See the current build status for the ODB compiler and ODB runtime libraries.

This installation method uses the `build2` package manager (`bpkg`) which has a uniform interface across all the platforms and compilers. It significantly simplifies the initial build as well as the ongoing maintenance (upgrades/downgrades, etc). Note also the your own projects don't have to use `build2` to benefit from this installation method. But if you would like to learn more, see The `build2` Toolchain Introduction.

1 Linux

The overall plan is as follows: first install the `build2` toolchain then use it to build and install the ODB compiler (with system GCC) and runtime libraries (with GCC or Clang).

Note also that while most things are by default installed into `/usr/local`, with `build2` everything can be cleanly uninstalled at any moment as long as you keep the build configuration that was used for the installation.

1.1 Installing `build2`

Build the latest `build2` toolchain by following the UNIX (Linux, Mac OS, FreeBSD) installation instructions.

1.2 Building ODB Compiler

First make sure the GCC plugin headers are installed. For Debian/Ubuntu, this package is called `gcc-X-plugin-dev` (where *X* is the GCC major version), for example:

```
$ gcc --version
gcc X.Y.Z

$ sudo apt-get install gcc-X-plugin-dev
```

For RedHat/Fedora, this package is called `gcc-plugin-devel`, for example (replace `dnf` with `yum` for older distributions):

```
$ sudo dnf install gcc-plugin-devel
```

Next create the build configuration (replace *X* with the GCC major version):

```
$ mkdir odb-build
$ cd odb-build

$ bpkg create -d odb-gcc-X cc     \
  config.cxx=g++                  \
  config.cc.coptions=-O3          \
  config.bin.rpath=/usr/local/lib \
  config.install.root=/usr/local  \
  config.install.sudo=sudo

$ cd odb-gcc-X
```

To build:

```
$ bpkg build odb@https://pkg.cppget.org/1/beta
```

To test:

```
$ bpkg test odb
```

To install:

```
$ bpkg install odb

$ which odb
$ odb --version
```

To upgrade:

```
$ bpkg fetch
$ bpkg status odb

$ bpkg uninstall odb
$ bpkg build --upgrade --recursive odb
$ bpkg install odb
```

See Package Consumption for more information on these `bpkg` commands.

1.3 Building ODB Runtime Libraries

While you can build the ODB runtime libraries (`libodb*`) in the same build configuration as the ODB compiler above, it will be easier to manage (install/uninstall, upgrade, etc) if they are in a separate build configuration.

Create the build configuration (replace *X* with the GCC major version):

```
$ cd odb-build
```

```
$ bpkg create -d gcc-X cc         \
  config.cxx=g++                  \
  config.cc.coptions=-O3          \
  config.install.root=/usr/local  \
  config.install.sudo=sudo
```

Next decide which database runtime libraries you need and perform the following steps (for `libodb-oracle` and `libodb-mssql` see their respective `INSTALL` files for additional requirements):

```
$ cd gcc-X
$ bpkg add https://pkg.cppget.org/1/beta
$ bpkg fetch
$ bpkg build libodb
$ bpkg build libodb-sqlite
$ bpkg build libodb-pgsql
$ bpkg build libodb-mysql
$ bpkg install --all --recursive
```

Note that by default the underlying database libraries (`libsqlite3`, `libmysqlclient`, and `libpq`) will be built from source packages. If you would prefer to use the system-installed versions, adjust the corresponding `build` commands as follows:

```
$ bpkg build libodb-sqlite ?sys:libsqlite3
$ bpkg build libodb-pgsql  ?sys:libpq
$ bpkg build libodb-mysql  ?sys:libmysqlclient
```

To upgrade:

```
$ cd gcc-X
$ bpkg fetch
$ bpkg status libodb libodb-...
$ bpkg uninstall --all --recursive
$ bpkg build --upgrade --recursive
$ bpkg install --all --recursive
```

Note also that you can create as many builds of the runtime libraries as you need: different compilers, debug/release, 32/64-bit, etc. Simply create another build configuration with the desired settings and repeat the above steps. For such development builds you will also probably want to adjust the installation location to somewhere private. For, example, this is how we can create a debug build that uses Clang with the `libc++` runtime and install it into `~/install/odb` (replace *X* with the Clang major version):

```
$ bpkg create -d clang-X-libc++-debug cc \
  config.cxx=clang++                     \
  config.cc.coptions=-g                  \
  config.cxx.coptions=-stdlib=libc++     \
  config.install.root=~/install/odb
```

After the installation the ODB runtime headers will be in `~/install/odb/include` and libraries in `~/install/odb/lib`. To use this build simply add the corresponding `-I` and `-L` options to your project's build configuration.

2 Windows

The overall plan is as follows: first install the `build2` toolchain (which includes GCC build with plugins enabled) then use it to build and install the ODB compiler (with GCC) and runtime libraries (with MSVC).

Note that the ODB compiler is installed into the `build2` installation directory (`C:\build2`). Currently this is the only supported arrangement due to the complexity of finding plugins, compilers, etc. Note also that you should be able to run the ODB compiler without having `C:\build2\bin` in your `PATH` as well as move/rename `C:\build2` or zip and copy it to another machine (in other words, `C:\build2` is a self-sufficient ODB compiler distribution).

2.1 Installing `build2`

Build the latest `build2` toolchain by following the Windows with MinGW installation instructions.

2.2 Building ODB Compiler

Open the command prompt, then run:

```
> set "PATH=C:\build2\bin;%PATH%"

> C:
> md \odb-build
> cd \odb-build
```

Create the build configuration:

```
> bpkg create -d odb-mingw cc^
  config.cxx=g++^
  config.cc.coptions=-O2^
  config.install.root=C:\build2

> cd odb-mingw
```

To build:

```
> bpkg build odb@https://pkg.cppget.org/1/beta
```

To test:

```
> bpkg test odb
```

To install:

```
> bpkg install odb

> where odb
> odb --version
```

To upgrade:

```
> bpkg fetch
> bpkg status odb

> bpkg uninstall odb
> bpkg build --upgrade --recursive odb
> bpkg install odb
```

See Package Consumption for more information on these `bpkg` commands.

2.3 Building ODB Runtime Libraries

Start a suitable Visual Studio command prompt (for example, `x86` or `x64`), then run:

```
> set "PATH=C:\build2\bin;%PATH%"

> C:
> cd \odb-build
```

Create the Release and Debug build configurations:

```
> bpkg create -d msvc-release cc^
  config.cxx=cl^
  "config.cc.coptions=/O2 /MD"^
  config.install.root=C:\odb\release

> bpkg create -d msvc-debug cc^
  config.cxx=cl^
  "config.cc.coptions=/Od /MDd /Zi"^
  config.cc.loptions=/DEBUG^
  config.install.root=C:\odb\debug
```

If you prefer to include the debug information into the libraries instead of having it in separate `.pdb` files, use this Debug configuration instead:

```
> bpkg create -d msvc-debug cc^
  config.cxx=cl^
  "config.cc.coptions=/Od /MDd /Z7"^
  config.install.root=C:\odb\debug
```

Next, for each build configuration perform the following set of steps (pick the database runtime libraries that you need; for `libodb-oracle` see below). Here replace *XXX* either with `debug` or `release`:

```
> cd msvc-XXX
> bpkg add https://pkg.cppget.org/1/beta
> bpkg fetch
> bpkg build libodb
> bpkg build libodb-sqlite
> bpkg build libodb-pgsql
> bpkg build libodb-mysql
> bpkg build libodb-mssql
> bpkg install --all --recursive
```

The result (both shared and static libraries) will be in the `C:\odb\XXX\` directories with headers in the `include\` subdirectory, static libraries and DLL import stubs (`.lib`) in `lib\` and DLLs in `bin\`.

If you need both 32 and 64-bit builds, then repeat the above steps for the other set of configurations (and from the corresponding Visual Studio command prompt). In this case you may also want to include the width into your build/install directory names (for example, `msvc-32-debug` and `C:\odb\32\debug`, etc).

If you also want to build `libodb-oracle`, then you will need to first install the OCI library (see `libodb-oracle\INSTALL` for various ways to obtain it) and then specify the location of its headers and libraries when creating the build configurations (you can also add the necessary `/I` and `/LIBPATH` values to an existing configuration by editing its `msvc-XXX\build\config.build` file). For example:

```
> bpkg create ...^
  config.cc.poptions=/IC:\oracle\oci\include^
```

```
  config.cc.loptions=/LIBPATH:C:\oracle\oci\lib\msvc^
```

To upgrade:

```
> cd msvc-XXX
> bpkg fetch
> bpkg status libodb libodb-...
> bpkg uninstall --all --recursive
> bpkg build --upgrade --recursive
> bpkg install --all --recursive
```

3 Mac OS

The overall plan is as follows: first install the `build2` toolchain then use it to build and install the ODB compiler with Homebrew GCC. Finally, build and install the ODB runtime libraries with Clang (and/or GCC).

As a first step, make sure you have Apple Clang:

```
$ clang++ --version
```

If it's not present, install it as part of the Command Line Tools:

```
$ xcode-select --install
```

Note also that while most things are by default installed into `/usr/local`, with `build2` everything can be cleanly uninstalled at any moment as long as you keep the build configuration that was used for the installation.

3.1 Installing `build2`

Build the latest `build2` toolchain by following the UNIX (Linux, Mac OS, FreeBSD) installation instructions.

3.2 Building ODB Compiler

First install GCC from Homebrew (here *X* is the GCC major version):

```
$ brew update
$ brew info gcc
$ brew install gcc

$ which g++-X
```

Next create the build configuration (replace *X* with the GCC major version):

```
$ mkdir odb-build
$ cd odb-build

$ bpkg create -d odb-gcc-X cc     \
  config.cxx=g++-X                \
  config.cc.coptions=-O3          \
  config.bin.rpath=/usr/local/lib \
  config.install.root=/usr/local

$ cd odb-gcc-X
```

To build:

```
$ bpkg build odb@https://pkg.cppget.org/1/beta
```

To test:

```
$ bpkg test odb
```

To install:

```
$ bpkg install odb

$ which odb
$ odb --version
```

To upgrade:

```
$ bpkg fetch
$ bpkg status odb

$ bpkg uninstall odb
$ bpkg build --upgrade --recursive odb
$ bpkg install odb
```

See Package Consumption for more information on these `bpkg` commands.

3.3 Building ODB Runtime Libraries

The following steps show how to build the ODB runtime libraries (`libodb*`) with Apple Clang. However, you can instead (or in addition, see below) build them with Homebrew GCC.

Create the build configuration (replace *X* with the Clang major version):

```
$ cd odb-build

$ bpkg create -d clang-X cc       \
  config.cxx=clang++              \
  config.cc.coptions=-O3          \
  config.install.root=/usr/local
```

Next decide which database runtime libraries you need and perform the following steps (for `libodb-oracle` and `libodb-mssql` see their respective `INSTALL` files for additional requirements):

```
$ cd clang
$ bpkg add https://pkg.cppget.org/1/beta
$ bpkg fetch
$ bpkg build libodb
$ bpkg build libodb-sqlite
$ bpkg build libodb-pgsql
$ bpkg build libodb-mysql
$ bpkg install --all --recursive
```

Note that by default the underlying database libraries (`libsqlite3`, `libmysqlclient`, and `libpq`) will be built from source packages. If you would prefer to use the system-installed versions, adjust the corresponding `build` commands as follows:

```
$ bpkg build libodb-sqlite ?sys:libsqlite3
$ bpkg build libodb-pgsql  ?sys:libpq
$ bpkg build libodb-mysql  ?sys:libmysqlclient
```

To upgrade:

```
$ cd clang-X
$ bpkg fetch
$ bpkg status libodb libodb-...
$ bpkg uninstall --all --recursive
$ bpkg build --upgrade --recursive
$ bpkg install --all --recursive
```

Note also that you can create as many builds of the runtime libraries as you need: different compilers, debug/release, 32/64-bit, etc. Simply create another build configuration with the desired settings and repeat the above steps. For such development builds you will also probably want to adjust the installation location to somewhere private. For, example, this is how we can create a debug build and install it into `~/install/odb`:

```
$ bpkg create -d clang-X-debug cc   \
  config.cxx=clang++                \
```

```
  config.cc.coptions=-g             \
  config.install.root=~/install/odb
```

After the installation the ODB runtime headers will be in `~/install/odb/include` and libraries in `~/install/odb/lib`. To use this build simply add the corresponding `-I` and `-L` options to your project's build configuration.