Marble Developers Guide

Dennis Nienhüser

August 16th, 2014

Contents

Welcome! This guide assists in setting up Marble in development environments and provides a detailed overview of the Marble library. It is directed mostly at

- developers who want to use the Marble library in their project
- packagers who want to generate binary packages for installing Marble and
- contributors who want to help making Marble yet more versatile and polished.

End users might find some useful information also, but are often better served just installing Marble and, if needed, reading the end-user documentation or requesting support.

The Marble project consists of software (LGPL 2.1 license) and data (some free license). The vast majority of its features are contained in the library portions (`libmarblewidget`, `libastro` and `plugins` sum up to 140k SLOC) and hence available to 3rd party applications. The Marble application end-users typically interact with (Marble KDE or Marble Qt) are each just thin wrappers that expose the library features, each less than 3k SLOC. A high-level overview of the existing modules looks like follows:

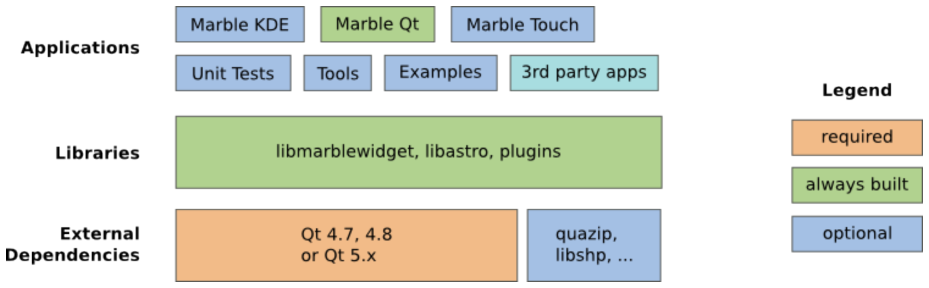


Figure 1: Marble Modules

The remaining parts of the developers guide are organized into three chapters. Building Marble describes how to retrieve the Marble sources, build them, set up a development environment, install Marble and create packages/installers. The Architecture and Concepts looks into the code contained within the library portions: What are the most important classes, how do they interact? Finally we consider how to extend Marble by new plugins, map themes and how to embed it into 3rd party applications in the Extending Marble chapter.

1 Building Marble

Retrieving the source code, compiling and installing it and executing Marble can be accomplished with just a few commands:

```
git clone git://anongit.kde.org/marble ~/marble/sources
mkdir -p ~/marble/build
```

```
cd ~/marble/build
cmake ~/marble/sources
make
sudo make install
```

For day-to-day development, packaging and other scenarios you need more control though. The following sections look into various build aspects.

1.1 Obtaining the Source Code

The Marble source code is maintained in a git repository. The clone URL is

- `git://anongit.kde.org/marble` (read-only/anonymous access)
- `git@git.kde.org:marble` (write access for KDE developers)

There are also web interfaces for quickly browsing the sources at quickgit.kde.org and projects.kde.org. The latter provides alternative access methods (http and tarballs) also. Note that despite the different URLs all point to the same repository.

Retrieving the Marble sources by anonymous access and storing them at `~/marble/sources` then becomes

```
git clone git://anongit.kde.org/marble ~/marble/sources
```

Development happens in feature branches or (most often) directly in the `master` branch. Although the `master` branch contains the latest and greatest features, using it in production environments is not recommended. Stable (release) branches are those starting with `KDE/4.`, e.g. `KDE/4.14`. They are created and maintained according to the KDE Release Schedules. The most recent stable branch can be determined using

```
git branch -r | grep KDE | sort -V
```

which reports said branch on the last line, e.g. `origin/KDE/4.14`. It can then be checked out using the command

```
git checkout -b KDE/4.14 origin/KDE/4.14
```

1.2 Source Code Organization

Let's have a quick look at the directory tree of the Marble sources, shortened to the most important directories:

```
src/
  apps/
    marble-ui/                Code shared between Marble KDE and Marble Qt
    marble-qt/                Marble application (Qt variant)
    marble-kde                Marble application (KDE variant)
    marble-mobile/            Marble application (Maemo variant)
    marble-touch/             Marble application (MeeGo variant)
  lib/
    marble/                   The heart of Marble
    astro/                    Astronomical calculations
  plugins/
    designer/                 Qt Designer integration
    declarative/              Qt Declarative support
    positionprovider/         Location providers (e.g. GPS)
    render/                   Float items and other display related
    runner/                   Parsing, searching, routing, reverse geocoding
  bindings/
    python/                   Python bindings (only compile in release branches)
tests/                        Unit tests
tools/                        Tools for e.g. data conversion
examples/                     Programming and data examples
  cpp/                        Using libmarblewidget from C++
  qml/                        Using libmarblewidget from QML
  python/                     Using libmarblewidget from Python
data/
  maps/                       Map themes
```

Remember the high-level overview of the Marble modules? `src/lib/marble`, `src/lib/astro` and `src/plugins` correspond to the libraries, while `src/apps`, `tests`, `tools` and `examples` reflect the applications.

1.3 CMake Build System

Marble uses CMake, a cross-platform open source build system. Building Marble is quite straightforward in the most simple case and comes down to

```
cmake /path/to/marble/sources
make
make install
```

However there are a lot of options to tweak things as needed. This section attempts to list them.

1.3.1 General Options

A couple of CMake generic and Marble specific options are commonly used to control the build process. The most important ones are

Flag	Default	Description
`CMAKE_BUILD_TYPE`		`Debug`, `Release` or `RelWithDebInfo`
`CMAKE_INSTALL_PREFIX`	`/usr/local`	`/usr`, `/home/$user/marble/export`, ...
`TILES_AT_COMPILETIME`	`ON`	Create the data for the Atlas map at compile time?
`BUILD_MARBLE_APPS`	`ON`	Build Marble applications?
`BUILD_MARBLE_TESTS`	`OFF`	Build unit tests?
`BUILD_MARBLE_TOOLS`	`OFF`	Build tools for data conversion etc.?
`BUILD_MARBLE_EXAMPLES`	`OFF`	Build Marble C++ example applications?
`WITH_DESIGNER_PLUGIN`	`ON`	Build the Qt Designer plugins?
`BUILD_HARMATTAN_ZOOMINTERCEPTOR`	`OFF`	Use volume keys for zooming on Maemo?
`BUILD_INHIBIT_SCREENSAVER_PLUGIN`	`OFF`	Inhibit screensaver during navigation on Maemo?

You can change the default values of the options using CMake specific tools like `cmake-gui`. Alternatively just pass them on the command line along the lines of `cmake -DBUILD_MARBLE_TESTS=ON /path/to/marble/sources`.

1.3.2 Controlling Dependencies

The only required dependency is Qt. In order for CMake to find Qt, make sure that `qmake` can be executed from `$PATH`. If you have multiple versions of Qt installed, check the output of `qmake -v` to determine which version will be used. Note that `qmake` alone does not mean Qt is fully installed on your system. You need an installation of Qt that includes development files (headers and optional modules like webkit).

Other libraries can optionally be installed and will be automatically detected by CMake and used. Most prominently kdelibs is among them. For historic reasons Marble treats kdelibs as a required dependency *unless you tell it to do otherwise*: If you do not want to use the Marble KDE application, pass the `QTONLY=ON` parameter to CMake and Marble will happily compile without KDE.

Dependency	Type	Description
Qt	Required	the famous cross-platform application framework
KDE development platform	Optional	an integrated set of technologies to build applications quickly and e
quazip	Optional	reading and displaying .kmz files
libshp	Optional	reading and displaying .shp files
libgps	Optional	position information via gpsd
libwlocate	Optional	Position information based on neighboring WLAN networks
QtLocation	Optional	position information via QtMobility QtLocation
liblocation	Optional	(Maemo) position information via liblocation.

If you encounter problems during the `cmake` run, check its output carefully. CMake reports which dependencies were found or not, and prints an extensive summary of its configuration.

For packaging Marble a system-wide install prefix makes most sense. During development a local installation is usually better though: It does not need root privileges to run `make install`.

- Make sure there is no system-wide installation of Marble e.g. from distribution packages. A system-wide installation can typically be detected by checking whether `/usr/lib/libmarblewidget.so` or `/usr/local/lib/libmarblewidget.so` exists, though other installation paths are in use also.
- You *must* run `make install`; a simple `make` is not enough to execute Marble later on. The reason for this is the installation of required data files.
- For a local installation, set `WITH_DESIGNER_PLUGIN=OFF`
- For a local installation, the environment variables `LD_LIBRARY_PATH` and `PATH` need to be adjusted to include the `lib/` and `bin/` directory of the installation directory, respectively.

1.3.3 Troubleshooting

- Marble starts, but shows a black map only. *You did not run `make install`, or moved the installation directory afterwards.*
- Marble does not start, the shell reports `marble: command not found`. *Call marble or marble-qt with an absolute path, or adjust the $PATH environment variable to include the `bin/` directory of your installation.*
- Marble does not start, the shell reports `error while loading shared libraries: libmarblewidget.so.19: cannot open`

`shared object file: No such file or directory.` *Setup the LD_LIBRARY_PATH environment variable to include the* `lib/` *directory of your installation.*

- Marble crashes during startup with an assertion talking about a hash containing some name. *Different versions of libmarblewidget are loaded at the same time. This can happen when an older installation of Marble is still around and old plugins linking against the old installation are trying to be loaded. Remove the old installation.*

In a similar fashion we can compare the directories in the sources to the files that will be installed after compilation and running `make install` (described further below):

Sources	Installation
`src/apps/marble-kde`	`bin/marble`
`src/apps/marble-qt`	`bin/marble-qt`
`src/apps/marble-mobile`	`bin/marble-mobile`
`src/apps/marble-touch`	`bin/marble-touch`
`tests`	not installed, run `make test` in the build folder
`tools`	not installed, execute directly from the build folder
`examples/cpp/$name`	`share/apps/marble/examples/$name`
`src/lib/marble/*.cpp`	`lib/libmarblewidget.so`
`src/lib/astro/*.cpp`	`lib/libastro.so/`
`src/plugins/*/$name/*.cpp`	`lib/kde4/plugins/marble/$name.so`
`src/lib/marble/*.h`	`include/marble/`
`src/lib/astro/*.h`	`include/astro/`
`data/`	`share/apps/marble/data`

1.3.4 Classic Makefiles (Linux)

1.3.5 Ninja Makefiles (Linux)

1.3.6 Visual Studio Solution (MS Windows)

1.3.7 MinGW (MS Windows)

1.3.8 Bundles (MacOSX)

1.4 Development Environment

1.4.1 QtCreator

1.4.2 Visual Studio

1.5 Installation and Packaging

1.5.1 System-wide Installation (Linux)

1.5.2 Local Installation (Linux)

1.5.3 NSIS Installer (MS Windows)

2 Architecture and Concepts

2.1 MarbleWidget

`MarbleWidget` is a `QWidget` that shows a map and associated information. For many use cases it is the main entry point into the `marblewidget` library. Many properties and methods are available to customize the appearance of the widget for your needs. By default `MarbleWidget` is interactive, allowing users to change which parts of the planet are displayed.

2.1.1 Minimalistic Code Example

```
#include <QApplication>
#include <marble/MarbleWidget.h>

int main(int argc, char** argv)
{
    QApplication app(argc, argv);
    Marble::MarbleWidget *mapWidget = new Marble::MarbleWidget;
    mapWidget->setMapThemeId("earth/srtm/srtm.dgml");
    mapWidget->show();
    return app.exec();
}
```

Thanks to using sane default values for all properties a basic `MarbleWidget` is easy to setup. The only thing that needs to be set is the `mapThemeId` property as explained in the Map Themes section. The code above, once compiled and executed, results in an application that looks like this:

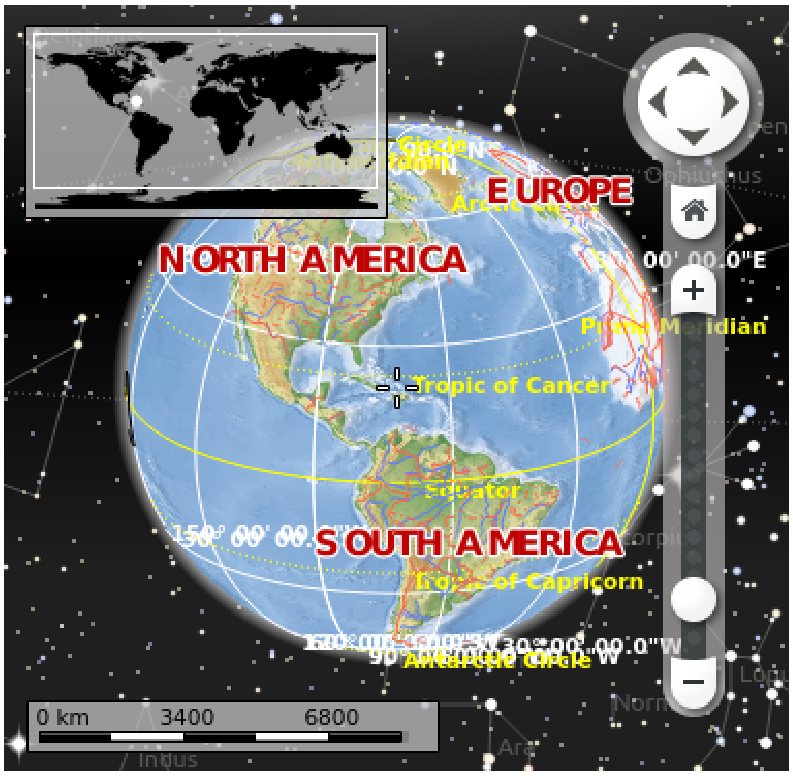


Figure 2: MarbleWidget Screenshot

2.1.2 Most Common Properties and Methods

While `MarbleWidget` can easily be created, it also provides a versatile configuration. Among the commonly changed properties and methods are:

- Choosing a planet to display and how it is rendered with the help of the `mapThemeId` property. See the Planets and Map Themes section for details.

- Deciding whether the map is shown as a globe or as some variant of a flat map through the `projection` property. Most commonly `Marble::Spherical` is used to render a globe and `Marble::Mercator` for a “flat” map.
- Adjusting the part of the map that is shown by the `zoom`, `longitude` and `latitude` parameters or one of their convenience methods. See the Viewport and Projections section for details.
- Changing which elements are shown and how they are displayed by the various variants of `show*` properties as well as the `floatItems` and `renderPlugins` methods.
- Fine-tuning how the user can interact with `MarbleWidget` using the `popupMenu` and `inputHandler` methods, or disabling input completely using `setInputEnabled(false)`.
- Adding your own rendering in a custom map layer using the `addLayer` method. The Map Layers section covers this in detail.
- Managing data related properties by accessing properties and methods of `MarbleModel` via the `model` property. See MarbleModel for details.

2.1.3 Qt Designer Integration

‘MarbleWidget’ can be used from within Qt Designer as described in this section.

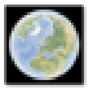
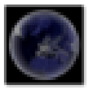
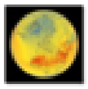
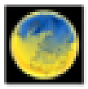
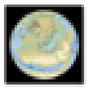
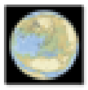
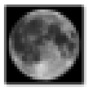
2.2 MarbleModel

2.3 Planets and Map Themes

While most importantly used to display maps of planet Earth, Marble can render maps of other planets or moons just as fine. All it takes is a DGML file which assembles the associated image and/or vector data and provides meta data like a name or the radius of the planet. Such a DGML file can be loaded using `MarbleWidget::setMapThemeId`; the associated map theme is loaded as well as the corresponding planet.

Typically you do not have to deal with planets directly, however. Instead one of the ready-made map themes can easily be loaded for display in MarbleWidget using its `mapThemeId` property. Map theme IDs follow a `$planet/$theme/$theme.dgml` naming scheme. Check out the `data/maps/` directory in the Marble sources for a list of default map themes. A typical Marble installation provides the following ones:

Preview	Name	ID	Description
	OpenStreetMap	earth/openstreetmap/openstreetmap.dgml	The free Wiki world map

Preview	Name	ID	Description
	Atlas	earth/srtm/srtm.dgml	A vector map in Atlas sty
	Satellite View	earth/bluemarble/bluemarble.dgml	Earth seen from space
	Political Map	earth/political/political.dgml	Highlights governmental b
	Earth at Night	earth/citylights/citylights.dgml	Earth seen from space at
	Plain Map	earth/plain/plain.dgml	A minimalistic vector map
	Historical Map 1689	earth/schagen1689/schagen1689.dgml	More than 400 years old
	Temperature (July)	earth/temp-july/temp-july.dgml	Average temperature in Ju
	Temperature (December)	earth/temp-dec/temp-dec.dgml	Average temperature in D
	Precipitation (July)	earth/precip-july/precip-july.dgml	Average precipitation in J
	Precipitation (December)	earth/precip-dec/precip-dec.dgml	Average precipitation in D
	Moon	moon/clementine/clementine.dgml	Earth's moon recorded by

Further map themes are available at marble.kde.org. The Marble applications provide built-in support to easily explore and install them. You can add support for it as well in your application using the `MapThemeDownloadDialog` class. Other popular maps like Google Maps, Google Satellite, Bing Maps and more can easily be integrated into Marble as well from a technical point of view. From a legal point of view however this is likely problematic and we do not want to promote the usage of maps which we think are not free.

Programmatic access to the list of installed map themes is provided through the `MapThemeManager` class, which provides a `QAbstractItemModel` for convenience.

Marble does know about a dozen different planets by itself. They can be accessed using `PlanetFactory`. For example, `PlanetFactory::construct("earth")` creates an instance of planet Earth. The `Planet` class is a lightweight class that contains a handful properties mostly needed for calculations. For planets yet unknown to Marble you can instantiate a `Planet` directly and set the respective properties. Alternatively new planets can also be instantiated from DGML files. This enables dynamic extension of the list of planets without the need to recompile Marble.

2.4 Rendering in Map Layers

2.5 Viewport and Projections

The `zoom` property has no unit, but its related properties `distance` and `radius` have: `distance` is the distance in `km` between the map surface and the virtual camera (increasing the distance zooms out), `radius` is the radius of the full globe in pixel (increasing the radius zooms in). These three properties are related and changing any of them changes the other two automatically. The `longitude` and `latitude` properties are accompanied by several convenience `centerOn` methods which take points, places or rectangular regions as arguments.

2.6 The Tree Model

2.7 Geographic Scene

2.8 Runners

2.9 Search

2.10 Routing

2.11 Location

2.12 Plugins

2.13 Float Items

2.14 DGML

2.15 KML

2.16 GeoData

2.17 Python Bindings

3 Extending Marble

3.1 Writing Map Themes

3.2 Writing Plugins

3.3 Integrating Marble into Applications

3.4 Using Qt Designer Plugins

When Marble is build with the `WITH_DESIGNER_PLUGIN=TRUE` flag (activated by default), `MarbleWidget` is available from within Qt Designer and can be embedded into `.ui` files just like any other `QWidget`. If `MarbleWidget` does not show up in Qt Designer, check that:

- The boolean cmake option WITH_DESIGNER_PLUGIN is activated
- The designer plugin is installed in a directory that is among Qt Designer's plugin paths

- The running Qt Designer has access to the `marblewidget` library through `LD_LIBRARY_PATH` or a comparable library search path setup.

Please note that `MarbleWidget` will not show up in Qt Creator's Design Pane. You have to use Qt Designer instead.

3.5 Contributing Back

3.5.1 Generating Patches

3.5.2 Reviewboard