Genealogical Tree

Find all the descendant on any level of ancestry

0.0.7_064b2b7

Fri Jul 17 2015 16:09:30 on user@ubuntu

VERSION 0.0.7 Git:

064b2b7 (HEAD, origin/develop, origin/HEAD, develop) First clean up, pending github readme images

VERSION CORE 0.0.2 Git:

d1f6b6c (HEAD, origin/develop, origin/HEAD, develop) First clean up, pending github readme images

VERSION BASE 0.0.3 Git:

eab971d (HEAD, origin/develop, origin/HEAD, develop) First clean up, pending github readme images

VERSION UI 0.0.1 Git:

4ed96e1 (HEAD, origin/master, origin/develop, origin/HEAD, develop) Merge branch 'master' of github.com:xue2sheng/GenealogicalTreeUI

VERSION PERSIST 0.0.1 Git:

c114f7c (HEAD, origin/master, origin/develop, origin/HEAD, develop) Initial commit

Contents

Chapter 1

Genealogical Tree

1.1 Summary

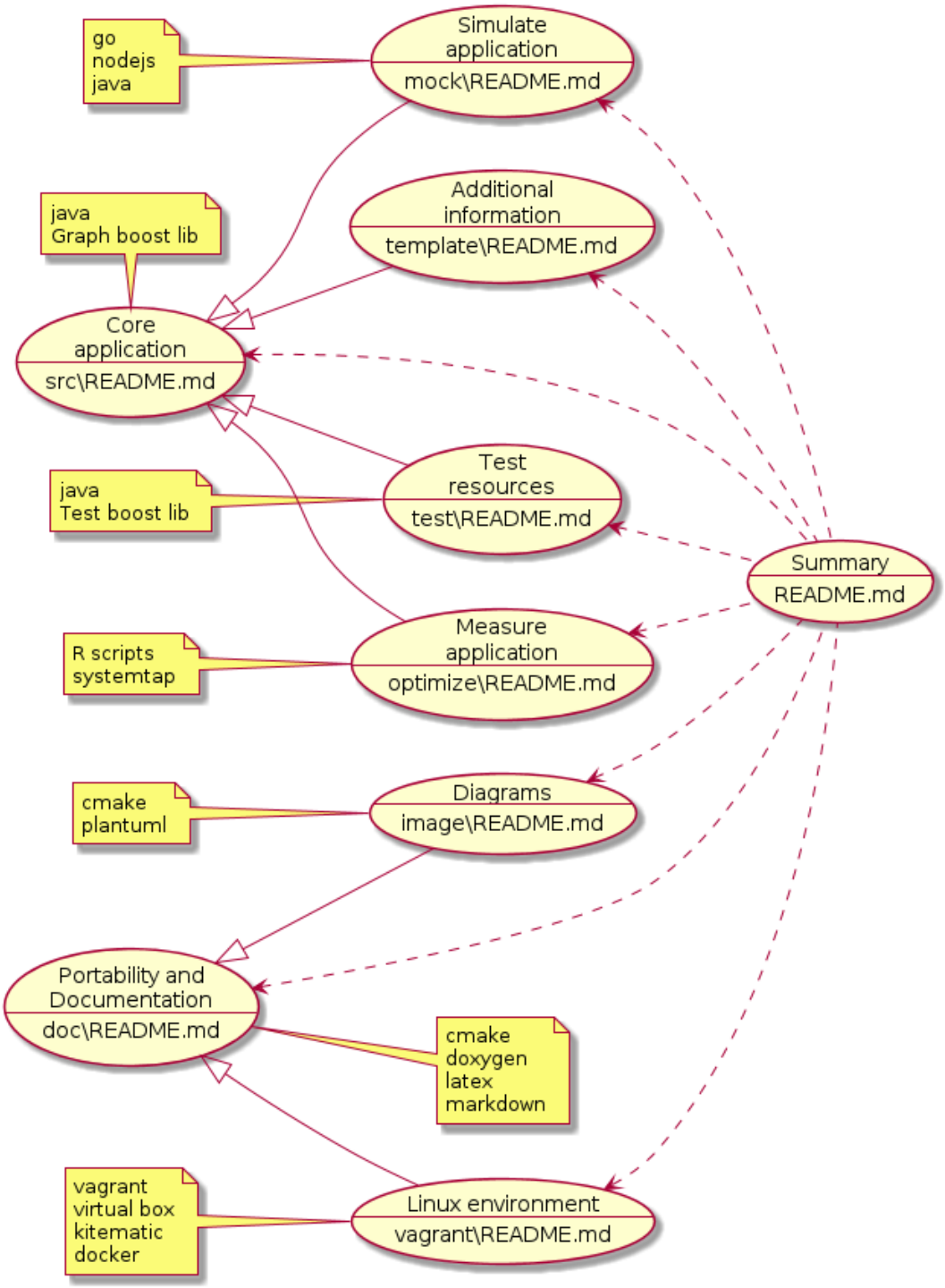
Program should be able to **find all the descendant with name Bob for all the ascendants with name Will on any level of ancestry**. In order to present the capabilities of your app:

- implement the application to optimize the initialization time.
- application should have built in data about genealogical tree of people living in particular country.
- please generate a representative data that has sample people an relationships between them. Use all varieties of names (can be also generated) but also put two test names (Bob and Will) and connect them in different relationships.
- the application should posses tests that are checking possible edge cases and ensure the stability of the application.
- the designed data structure should ensure optimized search time on following fields: name, last name, date of birth and location.

1.2 Approach

Instead of starting directly with the problem core, don't test thoroughly edge cases, leaping into too early optimization, don't document your results/decisions/mistakes and ending with an app that only run partially on your development environment, the **aproach** will be the opposite one.

1. Ensure a minimum of portability on different environments.
2. Configure linux environment on your Windows or MacOSX box if needed.
3. Generate diagrams from codex and documentation to be able to track down all the changes.
4. Use templates to gather external information to document as much automatically as possible.
5. Write tests to cover your app and let you optimize knowing you're not breaking previous development.
6. Measure your application in order to compare improvements/regressions during the optimization stage.
7. Simulate your deployment infrastructure to hunt down integration issues as soon as possible.
8. Solve the core problem in the most simple and maintainable way at our disposal.



No doubt this approach is an overkill for a pet project but it's way more realistic for big, long ones.

Chapter 2

Portability and documentation

A Modern C++ GNU compiler, g++ 4.9.2 or above, and a recent cmake, 3.1 or above, are the minimum. As well a valid *boost* library is supposed to be installed.

Mock servers are basically a bunch of **nodejs** and **go** scripts, so those languages are needed if you plan to execute or modify them. The recommended way to install **nodejs** and **go** is using `nvm` and `gvm` if possible.

2.1 Containers

Some `Dockerfiles` are provided to relieve the burden of installing. For example, getting a `documentation server`. One option to manage those *Docker* containers on *OSX* and *Windows* might be `Kitematic`

As well you can go directly for `Linux Containers` or mix both containers technologies. Visit `learning tools` for further instructions and related `vagrant` files.

2.1.1 NGINX server

A very simple **Dockerfile** is provided in the *doc* folder following `nginx example`

```
cd <project git folder>/doc
docker build -t nginx/doc .
docker run --name nginx_doc -d -p 8080:80 nginx/doc
```

Note: If you happen not to work on **Linux**, you should first install some *Linux Virtual Machine* as **docker server**. One option is to let **Kitematic** deal with that detail and use its *Docker CLI*.

2.2 Platforms

Several platforms were tested to some extent:

2.2.1 DEB Linux Type

Regarding to documentation, *doxygen*, *latex*, *graphviz* and *plantuml.jar* are needed. For example, if you work with **Xubuntu** 15.04 or its **Docker** equivalent, the following commands might do the trick for you:

```
sudo apt-get -y install git build-essential libboost-all-dev
sudo apt-get -y install doxygen doxygen-latex openjdk-8-jdk graphviz
sudo add-apt-repository -y ppa:george-edison55/cmake-3.x
sudo apt-get -y update
sudo apt-get -y install cmake
sudo apt-get -y upgrade
```

```
sudo mkdir /opt/plantuml && sudo chmod a+wr /opt/plantuml
wget http://sourceforge.net/projects/plantuml/files/plantuml.jar/download \
     -O /opt/plantuml/plantuml.jar
```

If you want to use latest *compiler*, you can use an extra repository:

```
sudo add-apt-repository ppa:ubuntu-toolchan-r/test
sudo apt-get update
sudo apt-get install gcc-5 g++-5
```

But then you might want to compile newer *boost* libraries with that compiler.

2.2.2 RPM Linux type

Another typical Linux platform is **CentOS**. Their *gcc* and *cmake* are very conservative, even for *CentOS 7*, so compile newer ones from **source code** might be a possibility:

CMake:

Download *tar.gz* with latest version and *untar* its source code

```
cd <building directory>
./bootstrap --prefix=/opt/cmake --mandir=/opt/cmake/man --docdir=/opt/cmake/doc
make
sudo make install
```

Compiler:

It's going to take long so try to use all the cores you got

```
mkdir ~/sourceInstallations
cd ~/sourceInstallations
svn co svn://gcc.gnu.org/svn/gcc/tags/gcc_5_1_0_release/
cd gcc_5_1_0_release/
./contrib/download_prerequisites
cd ~/sourceInstallations
mkdir gcc_5_1_0_release_build/
cd gcc_5_1_0_release_build/
../gcc_5_1_0_release/configure --enable-lenguages=c,c++ --prefix=/opt/gcc \
                               --program-suffix=-5 --disable-multilib
make -j <number of cores>
sudo make install

cd /usr/bin
sudo ln -s /opt/gcc/bin/gcc-5 gcc-5
sudo ln -s /opt/gcc/bin/g++-5 g++-5
```

Boost:

A long compilation that needs to be told where to get the proper toolset.

```
cd <folder with source code>
cp tools/build/example/user-config.jam .
```

Don't forget to edit *user-config.jam* to point to **g++-5**, i.e., *using gcc : 5 : /opt/gcc/bin/g++-5*

```
using gcc : 5 : /opt/gcc/bin/g++-5
:
<dll-path>/opt/gcc/lib64:/opt/gcc/boost/lib
<harcode-dll-paths>true
<cxxflags>-std=c++14
<cxxflags>-Wl,-rpath=/opt/gcc/lib64:/opt/gcc/boost/lib
<linkflags>-rpath=/opt/gcc/lib64:/opt/gcc/boost/lib
;
```

Note: Not all the targets might be created. If some of the missing ones are required for your apps, try to hack *boost* compilation scripts.

```
./bootstrap.sh --prefix=/opt/gcc/boost --with-toolset=gcc
sudo ./b2 -j8 install toolset=gcc-5 --prefix=/opt/gcc/boost \
          threading=multi define=BOOST_SYSTEM_NO_DEPRECATED stage release
```

Note: Take into account when compile with 'g++-5' on one Linux platform that got its own previous compiler version, you should let know to the **linker** where to get 'g++-5' libraries. Try to avoid **LD_LIBRARY_PATH** and use instead **RPATH**:

```
g++-5 -pthread --std=c++14 -Wl,-rpath=/opt/gcc/lib64 <rest of options>
```

In case of your linking against *boost* generates too many *auto_ptr* deprecated warnings:

```
g++-5 -pthread -std=c++14 -Wno-deprecated -Wl,-rpath=/opt/gcc/lib64:/opt/gcc/boost/lib \
      -I/opt/gcc/boost/include -L/opt/gcc/boost/lib -lboost_program_options <rest of options.
```

Hint: If you generate those *cmake*, *gcc* and *boost* on one machine and then copy them onto another, remember that there is **soft links** involved.

2.2.3 OSX type

In order to use *GNU* compiler instead of *XCode* **clang** one, there are several options. The one followed for this project was Homebrew.

Note: If you happen to work with *OSX* and *Homebrew*, don't forget to compile **boost** with the previous **gcc** compiler, not with the default *clang* one:

```
brew install gcc
brew install boost --cc=gcc-5
```

2.2.4 Windows type

As well there are several options to get your *GNU* chaintool ready on windows instead of *Visual Studio*. For example, Git and MinGW.

Another option might be following MSYS2 instruction:

```
pacman -Syu
pacman -S mingw-w64-x86_64-toolchain mingw-w64-x86_64-pkgconf make
pacman -S git mingw-w64-x86_64-cmake-git
```

2.3 Working with binaries & documentation

Usual commands at the **core** folder:

```
mkdir build
cd build
cmake ..
make
make doc
```

Optionally you can invoke *make install* to install binaries or *make install_doc* / *make show* to install / preview documentation.

Note: If you happen to work with *OSX* and Homebrew, don't forget to invoke *cmake* pointing to the **GNU** compiler:

```
cmake -DCMAKE_CXX_COMPILER=/usr/local/bin/g++-5 ..
```

Note: If you happen to work with *Windows* and Git/MinGW, don't forget to invoke *cmake* pointing to the **GNU** generator:

```
cmake -G "MSYS Makefiles" ..
```

As well a script, called **show** or something similar, will be created in your *home* directory as a shortcut for generating & viewing documentation. Don't hesitate to use it as a *template* for your specific environment.

2.4 Generate only documentation

Similar commands to the previous ones, just the compiler is not required at **root** folder:

```
mkdir build
cd build
make doc
```

Note: If you happen to work with *Windows* and Git/MinGW, don't forget to invoke *cmake* pointing to the **GNU** generator:

```
cmake -G "MSYS Makefiles" ..
```

Note: If your **make** utility is not installed in the default place, define *CMAKE_BUILD_TOOL*

```
cmake -G "MSYS Makefiles" -DCMAKE_BUILD_TOOL=<your location>  ..
```

As well, if you installed the documentation utility with **make show**, you're supposed to able to recreate and view that documentation PDF though usual *ssh* connection with enabled X11:

```
ssh -X <user>@<location> "./show"
```

Note: By default **make install_doc** or **make show** copy the documentation *PDF* with the default project name in your **home** directory. You can define that target file with:

```
cmake <rest of options> -DDOC_PDF=<your path & name, ending in .pdf> ..
```

2.5 IDE hints

Apart from the omnipresent **vim**, a couple of *IDE* were used:

2.5.1 Atom

Basically for **nodejs**, **go** and **markdown**

To use Atom don't forget to install plugins for *markdown* and *html* previews. As well for *running make* files and edit *cmake* files.

Note: Two *Vim* plugins, *vim-mode* and *ex-mode*, might be downloaded if you're accustomed to *vim*.

Note: If you happen to be only interested on generation documentation from *cmake* generated *make* files, configure *doc* task at **make-runner** plugin when *build/Makefile* is selected.

2.5.2 NetBeans

Basically for **java**, **c++**∗∗ **and** ∗∗**markdown**

To use NetBeans don't forget to configure a *cmake* project with *custom* **build** folder. Add at that moment any extra customization in the command line used by *cmake* instruction. For example:

- -DCMAKE_CXX_COMPILER=g++-5 for **OSX**
- -G "MSYS Makefiles" for **Windows**

Note: If you happen to use *jVi* plugin on *OSX*, don't forget to use "-lc" instead of just "-c" for its /bin/bash flag.

Note: In order to launch *images* and *documentation* generation from **IDE**, add *image* and *doc* tasks when *build/*↩ *Makefiles* is selected.

2.6 Development details

In order to generate binaries & documentation, the following versions were used:

2.6.1 Code

Pay attention to *cmake* and *gcc* versions. A minimum is required to work on several O.S. using modern C++. Feel free to locally hack **CMakeLists.txt** to meet your needs.

Linux (Xubuntu 15.04)

- **cmake** *3.2.2*
- **gcc** *4.9.2*
- **boost** *1.55*

OSX (Yosemite 10.10.4)

- **cmake** *3.2.2*
- **gcc** *5.1*
- **boost** *1.58*

Windows (Win7 x64)

- **cmake** *3.3.0*
- **gcc** *5.1*
- **boost** *1.58*

2.6.2 Documentation

Environment variables to locate PlantUML *jar* and default *PDF* viewer can be defined to overwrite default values. See **CMakeLists.txt** for further information on your platform.

Linux

- **doxygen** *1.8.9.1*
- **latex**/**pdfTeX** *2.6-1.40.15*
- **graphviz**/**dot** *2.38.0*
- **java**/**plantuml** *1.8.0_45/8026*

OSX

- **doxygen** *1.8.9.1*
- **latex**/**pdfTeX** *2.6-1.40.15*
- **graphviz**/**dot** *2.38.0*
- **java**/**plantuml** *1.8.0_40/8026*

Windows

- **doxygen** *1.8.9.1*
- **latex**/**pdfTeX** *2.9.5496-1.40.15*
- **graphviz**/**dot** *2.38.0*
- **java**/**plantuml** *1.8.0_45/8026*

Chapter 3

Linux environment on Win/OSX boxes

If it's not available a common Linux box for your development, it's possible to install a `Virtual Box` `Vagrant` box on your system.

The official `Ubuntu imagen`, modified to install the minumum required software for this project, can be located at **vagrant** folder.

`Docker` containers need as well at least a Linux machine as server. One dead easy way to get it, it's just install `Kitematic`.

Note: Try to *clone* this `GIT` project into a folder shared by Kitematic-installed *Docker* virtual box in order to **share** information among your native O.S., your vagrant box and your

Note: Avoid using paths with blanks in their names, even if your Windows or MacOSX is prone to. Those names are **not** *linux* friendly.

3.1 Approach

Supposed your shared working directory is *c:/Users/Public* and you've cloned using the default project name for its folder:

```
cd c:\Users\Public\GenealogicalTree\vagrant
vagrant up
```

There are sometimes issues with shared folders or similar, specially when the original vbox image is older than current installed Virtual Box or if your connection is not good enough.

In this case follow **Basic Linux Machine** and **Default Configuration** instructions. As you must be *root*, the following commands could be useful:

```
vagrant ssh
<inside your vbox>
sudo su
```

Or just login onto as root, password *vagrant*, through usual VBox IDE.

3.1.1 Basic Linux Machine

At VirtualBox create a Ubuntu 64 box, as BaseLinux, and installed previous ISO with the memory, processors and disk space you consider proper. Take into account that machine might be a kind of "Docker server", so make it sense go for a version without explicit desktop.

Don't forget to choose one near mirror for your packages because they might be updated several times during this installation.

Don't define a very complex password for your root and user profile. For example, root/vagrant and vagrant/vagrant might do the trick. Later on you can change them if needed. In the same way, define the hostname of that machine as 'localhost'

As well share some folder to copy scripts into and save the pain of typing so many lines. For that, you should install Guest Additions, adding an optical drive to your virtual machine if needed, probably from command line (as root):

```
apt-get update
apt-get install dkms sudo build-essential xserver-xorg xserver-xorg-core
mkdir /cdrom
mount /dev/cdrom /cdrom
cd /cdrom
./VBoxLinuxAdditions.run
reboot
```

3.1.2 Default configuration

Taking from granted you got already an upgraded Ubuntu box with Guest Additions installed and sharing as "vagrant" folder, now we can copy the following script to configure users properly base.sh and provision.sh (as root). Don't forget to check out that your sharing your *c:/Users/Public* foldera as **shared** on your VBox settings:

```
hostname localhost
mkdir /shared
usermod -aG vboxsf root
mount -t vboxsf  shared /shared
cp /shared/GenealogicalTree/vagrant/base.sh .
chmod a+x ./base.sh
cp /shared/GenealogicalTree/vagrant/provision.sh .
chmod a+x ./provision.sh
cp /shared/GenealogicalTree/vagrant/extra.sh .
chmod a+x ./extra.sh
./base.sh
./provision.sh
su - vagrant
sudo bash /root/extra.sh
su - tecnotree
sudo bash /root/extra.sh
reboot
```

3.1.3 Possible shortcuts for PowerShell

If you happen to work with PowerShell, there are some alias for your **$PROFILE** file:

- Resume Linux box: **resume-linux**

```
function Call-Linux-Resume {
  $current = pwd
  cd "C:\\Users\\Public\\GenealogicalTree\\vagrant"
  vagrant resume
  cd $current
}
Set-Alias resume-linux Call-Linux-Resume
```

- Suspend Linux box: **suspend-linux**

```
function Call-Linux-Suspend {
  $current = pwd
  cd "C:\\Users\\Public\\GenealogicalTree\\vagrant"
  vagrant suspend
  cd $current
}
Set-Alias suspend-linux Call-Linux-Suspend
```

- SSH onto Linux box: **linux**

```
function Call-Linux {
  $current = pwd
```

```
  cd "C:\\Users\\Public\\GenealogicalTree\\vagrant"
  vagrant ssh
  cd $current
}
Set-Alias linux Call-Linux
```

3.2 MacOSX

Supposed your shared working directory is ∗/Users/Shared∗, you can follow previous Windows steps.

But using Homebrew for development tools and your native *docker* client, you could almost do it with just installing *Kitematic* for a *docker* server.

Chapter 4

Generate diagrams from code or documentation

Diagrams related directly to the current code are a key part of any kind of technical documentation. As well being able to track down partial changes **inside** of those images along the code itself right out off the bat it's a huge improvement.

4.1 Way of working

Note: Use different names for any image you create

Generated images are saved at *image* folder, regardless from where defined at the project. So when you refer to them from **markdown** README.md files, you should user *relative paths*.

But **doxygen** *latex* documentation goes directly to that folder, so use just the name of the image.

4.2 Low level considerations

The tool is `PlantUML` and the format usually used is **PNG**. The reason behind was that **metadata** information is embedded into those photos and it might be checked out images before even generating them. This way you can save a huge deal of time at big projects and avoid stesssing too much your *GIT* repositories with binaries. Not free meals of course, more time to check all out.

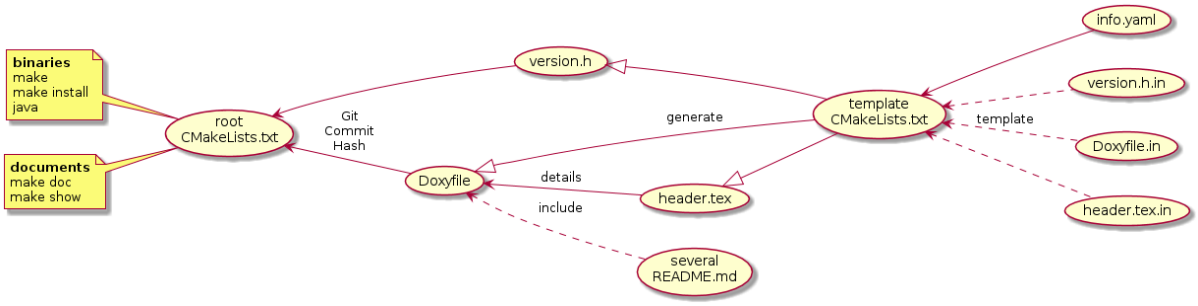
As well **SVG** format will be used by *Presentation* as *Sozi*. Being pure text are more *friendly* to *GIT* but being able to contain **code**, *GitHub* and other repositories prevent them from being renderized for your README.md files.

Chapter 5

Templates to gather external information

Global **info.yaml** file is processed to obtain some human friendly *digits* as **VERSION**.

Besides, extra basic external information to be included is **GIT COMMIT HASH**. This way *code* and *documentation* are related by this piece of information.



As well information on the machine where **cmake** was invoked is collected.

Note: Take into account that **the last commit** information will be processed; if there are new changes not yet committed, they will be included anyway. So in order to generate *final official* documentation first commit all your changes, generate the documentation and, if needed, commit that generated document.

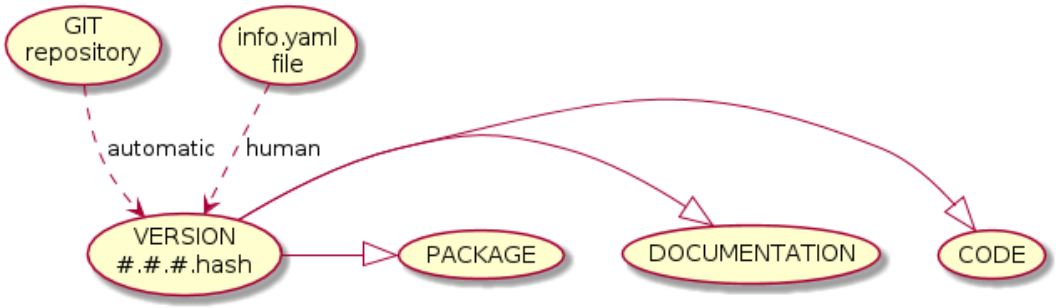
Note: On Windows system it's supposed that *awk* is available at command line. For example, usual **GIT** installation provides a *GIT Bash* environment.

5.1 Human friendly version

Three digits can be independently updated at *CMakeLists.txt* as human friendly version for both **HARDCODED** and **DYNAMIC** code versions. As well those digits will be used by documentation templates.

5.2 GIT Commit Hash

In order to add the specific **git commit hash** into code & documentation, *templates* are defined in the *template* folder for **Doxyfile**, **header.tex** & **version.h** files.



In order to **speed up** local compilations and let us hardcode our locally generated files, it's possible to instruct *cmake* to use this hardcoded header instead of usual GIT one.

The parameter to pass onto **cmake** is **VERSION_HARDCODED**:

```
cmake <rest of options> -DVERSION_HARDCODED=TRUE ..
```

Chapter 6

Test your application

Taking advantage of *boost* test cases as explained at An Engineer's Guide to Unit Testing.

Chapter 7

Measure your application

Scripts to gather information on performance. Basically *statistical information* on execution time of different approaches.

Chapter 8

Simulate deployment infrastructure

In order to make easy start testing even before all the components are fully developed, some mechanism to integrate **mocks** is required.

A humble `electron` application to supervise different mock services that simulate the real project.

Those mock services might be just `Docker` containers with a bunch of **nodejs** and/or **go** scripts.

Chapter 9

Application Core Code

Source folder for headears & code files directly involved with the core problem.

9.1 Simplications

Some stepts will be taken in order to circle problem to a manageable number of possibilities

9.2 Realistic limits

First of all, we should grasp a rough idea about which range of numbers to consider:

- Most Populous Country: China
  Inhabitants: around 1400000000
- Another populous country, culturally diverse: USA
  Number of first & last names: around 5200 & 152000
- Example of baptism registers: Ireland
  Roman Catholic: around 19th century
- Marriageable age: world
  Some common value: around 20
- Number of locations: India
  Number of villages: around 640000

9.3 Assumptions on numbers

This way we can assume that taking into account around 200 years of sensible information on our ascendants, around 10 generations back in time, we suppose not to deal with more than 4000000000 individuals.

As well, we could consider that our application should only tackle around different 6000 first names or 60000 last names in our given country. Even we can take for granted that there aren't more than 60000 locations, that we might classify them in two levels; one coarse level easy to remember and another fine one more close to small places.

Translate into C++:

- First Name: unsigned short int (uint16_t)

- Last Name: unsigned short int (uint16_t)
- Year of Birth: unsigned char (unit8_t) $<$ 200 years
- Coarse Location of Birth: unsigned short int (unit8_t)
- Month of Birth: unsigned char (uint8_t)
- Day of Birth: unsigned char (unit8_t)
- Fine location of Birth: unsigned short int (uint8_t)
- More information related to a specific subject: extra indexes.

This way we can use the **first 64 bits of information** as a valid **identification** for the individuals and with the advantage of getting the relevant information to debug first: *name and generation*.

Chapter 10

Class Index

10.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 11

File Index

11.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 12

Class Documentation

12.1 struct_index_t Struct Reference

Reuse hash keys as unique id.

`#include <id.h>`

Public Attributes

- char year
  year of birth

12.1.1 Detailed Description

Reuse hash keys as unique id.

The documentation for this struct was generated from the following file:

- core/src/id.h

12.2 union_id_t Union Reference

Let the compiler/memory translate between hash keys and id.

`#include <id.h>`

Public Attributes

- uint64_t id
  id Like a long integer for global hush table
- struct_index_t index
  index Like a structure for different hush tables

12.2.1 Detailed Description

Let the compiler/memory translate between hash keys and id.

The documentation for this union was generated from the following file:

- core/src/id.h

Chapter 13

File Documentation

13.1 core/src/id.h File Reference

```
#include <cstdint>
#include <iostream>
```

Classes

- struct struct_index_t

  Reuse hash keys as unique id.
- union union_id_t

  Let the compiler/memory translate between hash keys and id.

Functions

- std::ostream & operator<< (std::ostream &os, const union_id_t &u)

  Make it easier to log union_id_t.

Variables

- static constexpr const union_id_t EMPTY_UNION_ID {0}

  To zero union_id_t.

13.1.1 Detailed Description

Define types for id's for our subjects

A first approach of getting packed id & basic information in form of indexes:

- First Name: unsigned short int (uint16_t)
- Last Name: unsigned short int (uint16_t)
- Year of Birth: unsigned char (unit8_t) < 200 years
- Coarse Location of Birth: unsigned short int (unit8_t)
- Month of Birth: unsigned char (uint8_t)

- Day of Birth: unsigned char (unit8_t)

Grouping all that indexes we got a 64 bits identification

13.2 core/src/main.cpp File Reference

```
#include <iostream>
#include "version.h"
#include "id.h"
```

Functions

- int main (int argc, char **argv)

  Main function.

13.2.1 Function Documentation

13.2.1.1 int main (int *argc*, char ** *argv*)

Main function.

Parameters

argc	An integer argument count of the command line arguments
argv	An argument vector of the command line arguments

Returns

an integer 0 upon exit success

13.3 core/src/version.h File Reference

Macros

- #define DEFINE_VERSION_FIRST "0"

  First digit in the version.
- #define DEFINE_VERSION_MIDDLE "0"

  Second digit in the version.
- #define DEFINE_VERSION_LAST "1"

  Third digit in the version.
- #define DEFINE_GIT_DETAILS "f61cf8f (HEAD, origin/master, origin/develop, origin/HEAD, develop) Initial commit"

  git log –oneline –decorate -1
- #define DEFINE_GIT_COMMIT_HASH "f61cf8f"

  git log -1 –format=h
- #define DEFINE_VERSION

  Version Human + Git hash.

Variables

- static const char * VERSION = "VERSION = " DEFINE_VERSION

  static const VERSION to be checked out using 'strings' command
- static const char * GIT_DETAILS = "GIT_DETAILS = " DEFINE_GIT_DETAILS

  static const GIT_DETAILS to be checked out using 'strings' command

13.3.1 Detailed Description

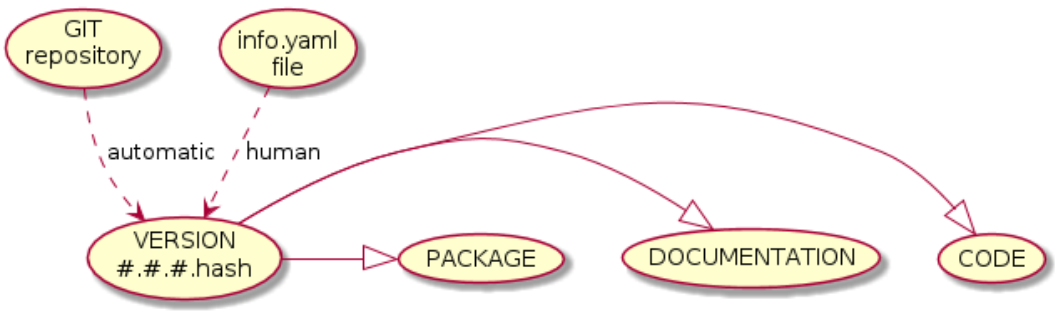
This metadata information might be located through **strings** command

- Linux/Solaris/Mac:

```
strings <binary> | grep VERSION
strings <binary> | grep GIT_DETAILS
```

- Windows (MinGW):

```
strings <binary> | findstr VERSION
strings <binary> | findstr GIT_DETAILS
```



13.3.2 Macro Definition Documentation

13.3.2.1 #define DEFINE_VERSION

Value:

```
DEFINE_VERSION_FIRST "." \
        DEFINE_VERSION_MIDDLE "."
      DEFINE_VERSION_LAST \
    "_" DEFINE_GIT_COMMIT_HASH
```

Version Human + Git hash.

Index