Bauble Documentation

Release 3.1.8

Brett Adams

January 19, 2021

Contents

Ghini is a suite of applications for managing botanical specimen collections.

- **ghini.desktop** lets you create and query a database representing objects and events in your plant collection.

- **ghini.web** publishes highlights from your database on the web.

- **ghini.pocket** puts a snapshot of your database in your handheld device.

- **ghini.tour** assists garden visitors with a map and spoken virtual panels.
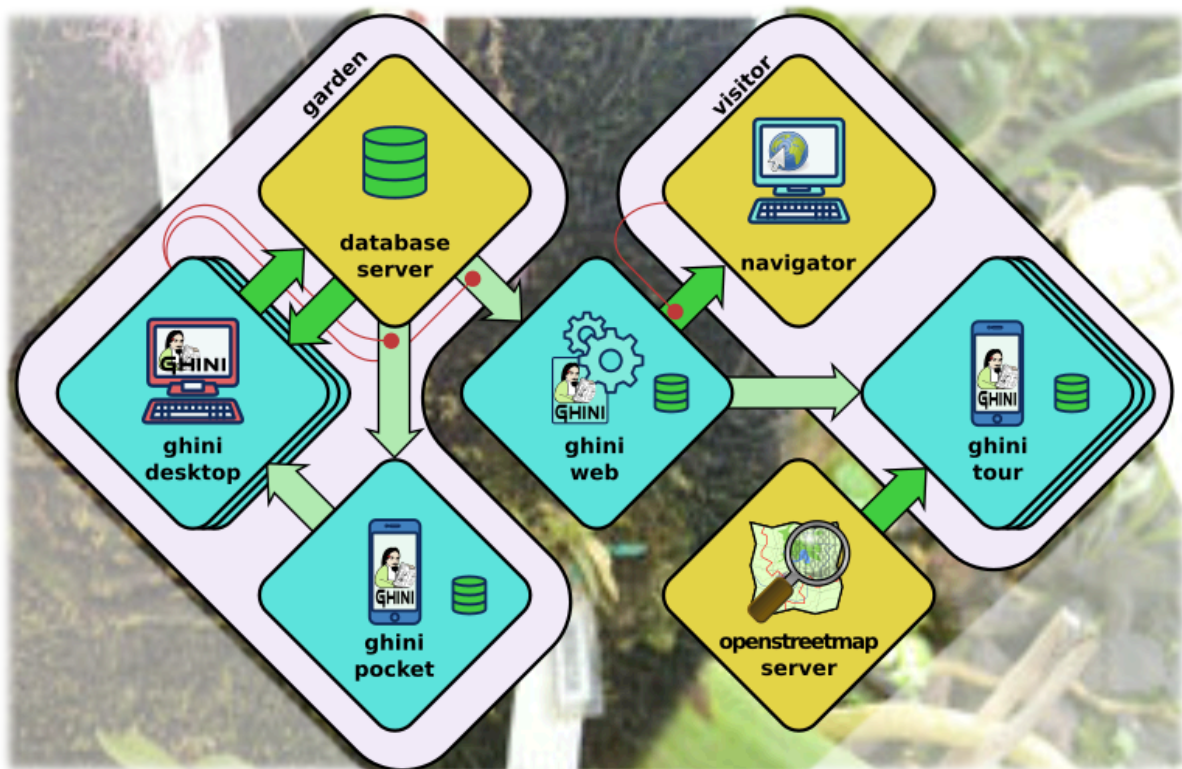
The bulk of this documentation focuses on ghini.desktop. One final chapter presents the rest of the Ghini family: *ghini.pocket*, *ghini.web*, *ghini.tour*, and the *data streams between components*.

All Ghini software is open and free. Our standalone software is released under the GNU Public License. Our client-server software follows the GNU Affero Public License.

Contents

CHAPTER 1

Statements

1.1 Ghini's goals and highlights

Should you use this software? This question is for you to answer. We trust that if you manage a botanic collection, you will find Ghini overly useful and we hope that this page will convince you about it.

This page shows how Ghini makes software meet the needs of a botanic garden.

If you already know, and all you want is to do something practical, just install the software, then check our user-contributed recipes.

1.1.1 Botanic Garden

According to the Wikipedia, »A botanic(al) garden is a garden dedicated to the collection, cultivation and display of a wide range of plants labelled with their botanical names«, and still according to the Wikipedia, »a garden is a planned space, usually outdoors, set aside for the display, cultivation, and enjoyment of plants and other forms of nature.«

So we have in a botanic garden both the physical space, the garden, as its dynamic, the activities to which the garden is dedicated, activities which makes us call the garden a botanic garden.

1.1.2 Botanic Garden Software

At the other end of our reasoning we have the application program Ghini, and again quoting the Wikipedia, »an application program is a computer program designed to perform a group of coordinated functions, tasks, or activities for the benefit of the user«, or, in short, »designed to help people perform an activity«.

Fig. 1: **the physical garden**

Fig. 2: **collection related activities in the garden**

Data and algorithms within Ghini have been designed to represent the physical space and the dynamic of a botanic garden.

Fig. 3: **core structure of Ghini's database**

In the above figure, a simplified view on the database, the highlighted blocks are those relative to objects you definitely need insert in the database.

We distinguish three main sections in the database. Start reading the graph from the right hand side, with the relevant **Taxonomy** information, then step to administering your **Collection**, and finally consider the physical **Garden**.

The central element in Ghini's point of view is the `Accession`. Following its links to other database objects lets us better understand the structure:

Accession links Planting to Species

> An `Accession` represents the action of receiving this specific plant material in the garden. As such, `Accession` is an abstract concept, it links physical living `Plantings` —groups of plants placed each at a `Location` in the garden— to the corresponding `Species`. It is not the same as an acquisition from a source, because in a single acquisition you can access material of more than one species. In other words: a single aquisition can embark multiple accessions. An `Accession` has zero or more `Plantings` associated to it (0..n), and it is at all times connected to exactly 1 `Species`. Each `Planting` belongs to exactly one `Accession`, each `Species` may have multiple `Accessions` relating to it.

> An `Accession` stays in the database even if all of its `Plantings` have been removed, sold, or have died. Identifying the `Species` of an `Accession` consistently connects all its `Plantings` to the `Species`.

Accession at the base of the history of your plants

> `Propagations` and `Contacts` provide plant material for the garden; this information is optional and smaller collectors might prefer to leave this aside. A `Propagation` trial may be unsuccessful, most of the time it will result in exactly one accession, but it may also produce slightly different taxa, so the database allows for zero or more `Accessions` per `Propagation` (0..n). Also a `Contact` may provide zero or more `Accessions` (0..n).

Accession and Verification opinions

> Specialists may formulate their opinion about the `Species` to which an `Accession` belongs, by providing a `Verification`, signing it, and stating the applicable level of confidence.

Accessing your own Propagations

> If an `Accession` was obtained in the garden nursery from a successful `Propagation`, the `Propagation` links the `Accession` and all of its `Plantings` to a single parent `Planting`, the seed or the vegetative parent.

Even after the above explanation, new users generally still ask why they need pass through an `Accession` screen while all they want is to insert a `Plant` in the collection, and again: what

---

is this "accession" thing anyway? Most discussions on the net don't make the concept any clearer. One of our users gave an example which I'm glad to include in Ghini's documentation.

use case

1. At the beginning of 2007 we got five seedlings of *Heliconia longa* (a plant `Species`) from our neighbour (the `Contact` source). Since it was the first acquisition of the year, we named them 2007.0001 (we gave them a single unique `Accession` code, with quantity 5) and we planted them all together at one `Location` as a single `Planting`, also with quantity 5.

2. At the time of writing, nine years later, `Accession` 2007.0001 has 6 distinct `Plantings`, each at a different `Locations` in our garden, obtained vegetatively (asexually) from the original 5 plants. Our only intervention was splitting, moving, and of course writing this information in the database. Total plant quantity is above 40.

3. New `Plantings` obtained by (assisted) sexual `Propagation` come in our database under different `Accession` codes, where our garden is the `Contact` source and where we know which of our `Plantings` is the seed parent.

the above three cases translate into several short usage stories:

1. activate the menu Insert → Accession, verify the existence and correctness of the `Species` *Heliconia longa*, specify the initial quantity of the `Accession`; add its `Planting` at the desired `Location`.

2. edit `Planting` to correct the amount of living plants — repeat this as often as necessary.

3. edit `Planting` to split it at separate `Locations` — this produces a different `Planting` under the same `Accession`.

4. edit `Planting` to add a (seed) `Propagation`.

5. edit `Planting` to update the status of the `Propagation`.

6. activate the menu Insert → Accession to associate an accession to a successful `Propagation` trial; add the `Planting` at the desired `Location`.

In particular the ability to split a `Planting` at several different `Locations` and to keep all uniformly associated to one `Species`, or the possibility to keep information about `Plantings` that have been removed from the collection, help justify the presence of the `Accession` abstraction level.

1.1.3 Hypersimplified view

People using Ghini only sporadically may prefer ignoring the database structure and look at it as two nested sequences of objects, each element of the sequence being necessary to add element at the next level.

In order to get down to an Accession, you will need four levels, as in this example:

A quite complete set of Families and Genera are inserted in your database at the moment Ghini initializes it. So all you need is adding Species and Accessions, in this order.

When placing a physical Plant (relative to an Accession) somewhere in the garden, you need to describe this "somewhere" digitally, as a Location in the garden.
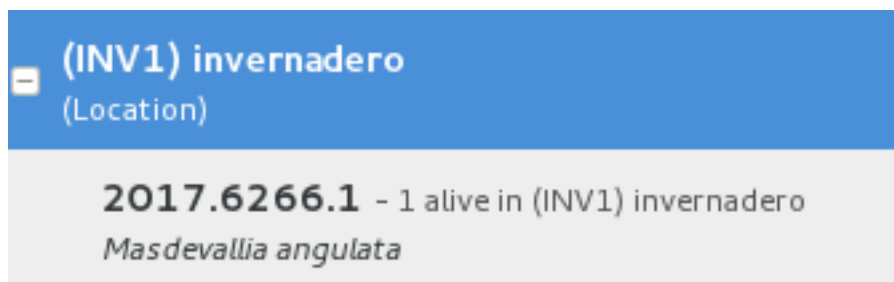
1.1.4 Highlights

not-so-brief list of highlights, meant to whet your appetite.

taxonomic information

When you first start Ghini, and connect to a database, Ghini will initialize the database not only with all tables it needs to run, but it will also populate the taxon tables for ranks family and genus, using the data from the "RBG Kew's Family and Genera list from Vascular Plant Families and Genera compiled by R. K. Brummitt and published by the Royal Botanic Gardens, Kew in 1992". In 2015 we have reviewed the data regarding the Orchidaceae, using "Tropicos, botanical information system at the Missouri Botanical Garden - www.tropicos.org" as a source.

importing data

Ghini will let you import any data you put in an intermediate json format. What you import will complete what you already have in the database. If you need help, you can ask some Ghini professional to help you transform your data into Ghini's intermediate json format.

synonyms

Ghini will allow you define synonyms for species, genera, families. Also this information can be represented in its intermediate json format and be imported in an existing Ghini database.

scientific responsible

Ghini implements the concept of 'accession', intermediate between physical plant (or a group thereof) and abstract taxon. Each accession can associate the same plants to different taxa, if two taxonomists do not agree on the identification: each taxonomist can have their say and do not need overwrite each other's work. All verifications can be found back in the database, with timestamp and signature.

helps off-line identification

Ghini allows you associate pictures to physical plants, this can help recognize the plant in case a sticker is lost, or help taxonomic identification if a taxonomist is not available at all times.

exports and reports

Ghini will let you export a report in whatever textual format you need. It uses a powerful templating engine named 'mako', which will allow you export the data in a selection to whatever format you need. Once installed, a couple of examples are available in the mako subdirectory.

annotate your info

You can associate notes to plants, accessions, species, ... Notes can be categorized and used in searches or reports.

garden or herbarium

Management of plant locations.

database history

All changes in the database is stored in the database, as history log. All changes are 'signed' and time-stamped. Ghini makes it easy to retrieve the list of all changes in the last working day or week, or in any specific period in the past.

simple and powerful search

Ghini allows you search the database using simple keywords, e.g.: the name of the location or a genus name, or you can write more complex queries, which do not reach the complexity of SQL but allow you a decent level of detail localizing your data.

database agnostic

Ghini is not a database management system, so it does not reinvent the wheel. It works storing its data in a SQL database, and it will connect to any database management system which accepts a SQLAlchemy connector. This means any reasonably modern database system and includes MySQL, PostgreSQL, Oracle. It can also work with sqlite, which, for single user purposes is quite sufficient and efficient. If you connect Ghini to a real database system, you can consider making the database part of a LAMP system (Linux-Apache-MySQL-Php) and include your live data on your institution web site.

language agnostic

The program was born in English and all its technical and user documentation is first written in that language. Both technical and user documentation use `gettext`, an advanced tool for semi-automatic translation.

The program has been translated and can be used in various other languages, including Spanish (97%), French (82%), Portuguese (71%), to name some Southern American languages, as well as Ukrainian (100%) and Czech (71%).

Translation of documentation goes a bit slower, with only Ukrainian, Spanish and Italian at more than 50%.

platform agnostic

Installing Ghini on Windows is an easy and linear process, it will not take longer than 10 minutes. Ghini was born on Linux and installing it on ubuntu, fedora or debian is consequently even easier. MacOSX being based on unix, it is possible to successfully run the Linux installation procedure on any recent Apple computer, after a few preparation steps.

easily updated

The installation process will produce an updatable installation, where updating it will take less than one minute. Depending on the amount of feedback we receive, we will produce updates every few days or once in a while.

unit tested

Ghini is continuously and extensively unit tested, something that makes regression of functionality close to impossible. Every update is automatically quality checked, on the Travis

Continuous Integration service. Integration of TravisCI with the github platform will make it difficult for us to release anything which has a single failing unit test.

Most changes and additions we make, come with some extra unit test, which defines the behaviour and will make any undesired change easily visible.

customizable/extensible

Ghini is extensible through plugins and can be customized to suit the needs of the institution.

1.2 Mission & Vision

Here we state who we are, what we think of our work, what you can expect of this project.

1.2.1 Who is behind Ghini

Ghini is a small set of programs, meant to let collection managers manage their collection also digitally.

Ghini was born back in 2004 as Bauble, at the Belize Botanical Garden. It was later adapted to the needs of a few more gardens. Brett Adams, the original programmer, made this software a commons, by releasing it under a GPL license.

After years of stagnation Mario Frasca revived the project, and rebranded it as Ghini in honour of Luca Ghini, founder of the first European botanic garden and herbarium. Mario Frasca started advocating, travelling, distributing, developing, expanding, redefining, documenting it, and it is now Mario Frasca writing this, looking for users, requesting feedback.

Behind Ghini there's not only one developer, but a small but growing global users community.

Translations are provided by volunteers who mostly stay behind the scenes, translating missing terms or sentences, and disappearing again.

To make things clearer when we speak of Ghini, but should—and in this document we will—indicate whether it's Ghini(the software), or Ghini(the people), unless obviously we mean both things.

1.2.2 Mission

Our goal as Ghini Software is to provide free software, of proven quality, and to let anybody install it if they feel like it. We also aim at facilitating access to functional knowledge, in the form of documentation or by laying the contact among users or between users and software professionals.

All our sources, software and documentation, are open and free, and we welcome and stimulate people to use and to contribute. To facilitate community forming, all our platforms can be consulted without registration. Registration is obviously required if you want to contribute.

Ghini welcomes the formation of groups of users, bundling forces to define and finance further development, and we welcome developers contributing software, from any corner in the world, and we stimulate and help them comply with the high quality requirements, before we accept the contributed code in the software sources.

1.2.3 Vision

The Vision serves to indicate the way ahead and projects a future image of what we want our organization to be, in a realistic and attractive way. It serves as motivation because it visualizes the challenge and direction of necessary changes in order to grow and prosper.

- by the year 2020
- reference point
- community
- development
- integration with web portal
- geographic information

CHAPTER 2

Installing Ghini

2.1 Installation

ghini.desktop is a cross-platform program and it will run on unix machines like GNU/Linux and MacOSX, as well as on Windows.

one-liner for hurried users.

> Linux users just download and run the installation script.
>
> Windows users in a real hurry don't the instructions and use a recent Windows installer. You do not miss any functional feature, but you have less chances to contribute to development.
>
> Mac users are never in a hurry, are they?
>
> Please make sure you check the documentation at some later moment.

The installation and subsequent updates for ghini.desktop are based on a single cross-platform installation procedure. This has a few big advantages which you will learn to appreciate as we go.

The installation is based on running a script.

- The GNU/Linux script takes care of everything, from dependecies to installation for users in the `ghini` group.

- The Windows script needs you to first install a couple things.

- On MacOSX we use the same script as on GNU/Linux. Since OSX has no default package manager, we install one and we use it before we start the script.

Following our installation procedure, you will end with Ghini running within a Python virtual environment, all Python dependencies installed locally, non conflicting with any other Python program you may have on your system.

Dependencies that don't fit in a Python virtual environment are: Python, virtualenv, GTK+, and a couple of Gtk libraries. Their installation varies per platform.

If you later choose to remove Ghini, you simply remove the virtual environment, which is a directory, with all of its content, then decide whether to remove the globally installed dependencies.

2.1.1 Installing on GNU/Linux

Open a shell terminal window, and follow the following instructions.

1. Download the devinstall.sh script. Please do not follow the link, instead right-click on it, and choose *Save link as. . . .*

2. Invoke the script from a terminal window, starting at the directory where you downloaded it, like this:

```
bash ./devinstall.sh
```

The script will produce quite some output, which you can safely ignore.

global installation

> When almost ready, the installation script will ask you for your password. This lets it create a `ghini` user group, initialise it to just yourself, make the just created `ghini` script available to the whole `ghini` user group.
>
> If feeling paranoid, you can safely not give your password and interrupt the script there.
>
> Possibly the main advantage of a global installation is being able to find Ghini in the application menus of your graphic environment.

3. You can now start ghini by invoking the `ghini` script:

```
ghini
```

1. You use the same `ghini` script to update ghini to the latest released production patch:

```
~/bin/ghini -u
```

This is what you would do when ghini shows you something like this:

2. Users of the global installation will also type `ghini` to invoke the program, but they will get to a different script, located in `/usr/local/bin`. This globally available `ghini` script cannot be used to update a ghini installation.

3. Again the same `ghini` script lets you install the optional database connectors: option `-p` is for PostgreSQL, option `-m` is for MySQL/MariaDB, but you can also install both at the same time:

```
~/bin/ghini -pm
```

Please beware: you might need solve dependencies. How to do so, depends on which GNU/Linux flavour you are using. Check with your distribution documentation.

4. You can also use the `ghini` script to switch to a different production line. At the moment `1.0` is the stable one, but you can select `3.1` if you want to help us with its development:

```
~/bin/ghini -s 3.1
```

beginner's note

To run a script, first make sure you note down the name of the directory to which you have downloaded the script, then you open a terminal window and in that window you type *bash* followed by a space and the complete name of the script including directory name, and hit on the enter key.

---

technical note

> You can study the script to see what steps if runs for you.
>
> In short it will install dependencies which can't be satisfied in a virtual environment, then it will create a virtual environment named as your selected branch (e.g.: `ghini-3.1`), use git to download the sources to a directory named `~/Local/github/Ghini/ghini.desktop`, and connect this git checkout to the `ghini-3.1` branch (this you can consider a production line), it then builds ghini, downloading all remaining dependencies in the virtual environment, and finally it creates the `ghini` startup script.
>
> If you have `sudo` permissions, it will be placed in `/usr/local/bin`, otherwise in your `~/bin` folder.

Next. . .

Connecting to a database.

2.1.2 Installing on MacOSX

Being macOS a unix environment, most things will work the same as on GNU/Linux (sort of).

First of all, you need things which are an integral part of a unix environment, but which are missing in a off-the-shelf mac:

1. developers tools: xcode. check the wikipedia page for the version supported on your mac.

2. package manager: homebrew (tigerbrew for older OSX versions).

Installation on older macOS.

> Every time we tested, we could only solve all dependencies on the two or three most recent macOS versions. In April 2015 this excluded macOS 10.6 and older. In September 2017 this excluded macOS 10.8 and older. We never had a problem with the lastest macOS.
>
> The problem lies with homebrew and some of the packages we rely on. The message you have to fear looks like this:
>
> ```
> Do not report this issue to Homebrew/brew or Homebrew/core!
> ```
>
> The only solution I can offer is: please update your system.
>
> On the bright side, if at any time in the past you did install ghini.desktop on your older and now unsupported macOS, you will always be able to update ghini.desktop to the latest version.

With the above installed, open a terminal window and run:

```
brew doctor
```

make sure you understand the problems it reports, and correct them. pygtk will need xquartz and brew will not solve the dependency automatically. either install xquartz using brew or the way you prefer:

```
brew install Caskroom/cask/xquartz
```

then install the remaining dependencies:

```
brew install git
brew install pygtk  # takes time and installs all dependencies
```

follow all instructions on how to activate what you have installed.

In particular, make sure you read and understand all reports starting with `If you need to have this software.`

You will need at least the following four lines in your `~/.bash_profile`:

```
export LC_ALL=en_US.UTF-8
export LANG=en_US.UTF-8
export PATH="/usr/local/opt/gettext/bin:$PATH"
export PATH="/usr/local/opt/python/libexec/bin:$PATH"
```

Activate the profile by sourcing it:

```
. ~/.bash_profile
```

Before we can run `devinstall.sh` as on GNU/Linux, we still need installing a couple of python packages, globally. Do this:

```
sudo -H pip install virtualenv lxml
```

The rest is just as on a normal unix machine. Read the above GNU/Linux instructions, follow them, enjoy.

As an optional aesthetical step, consider packaging your `~/bin/ghini` script in a platypus application bundle. The `images` directory contains a 128×128 icon.

Next. . .

*Connecting to a database*.

2.1.3 Installing on Windows

The steps described here instruct you on how to install Git, Gtk, Python, and the python database connectors. With this environment correctly set up, the Ghini installation procedure runs as on GNU/Linux. The concluding steps are again Windows specific.

Note: Ghini has been tested with and is known to work on W-XP, W-7 up to W-10. Although it should work fine on other versions Windows it has not been thoroughly tested.
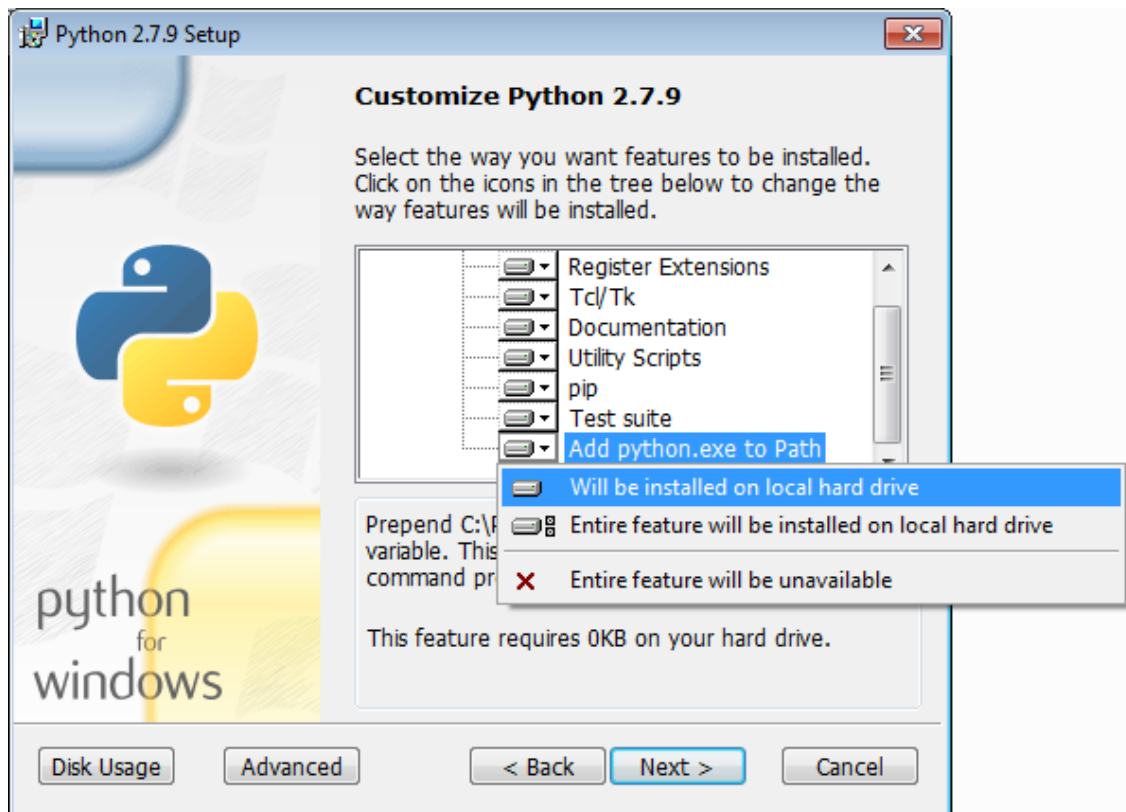

The installation steps on Windows:

1. download and install `git` (comes with a unix-like `sh` and includes `vi`). Grab it from the Git download area.

   all default options are fine, except we need git to be executable from the command prompt:
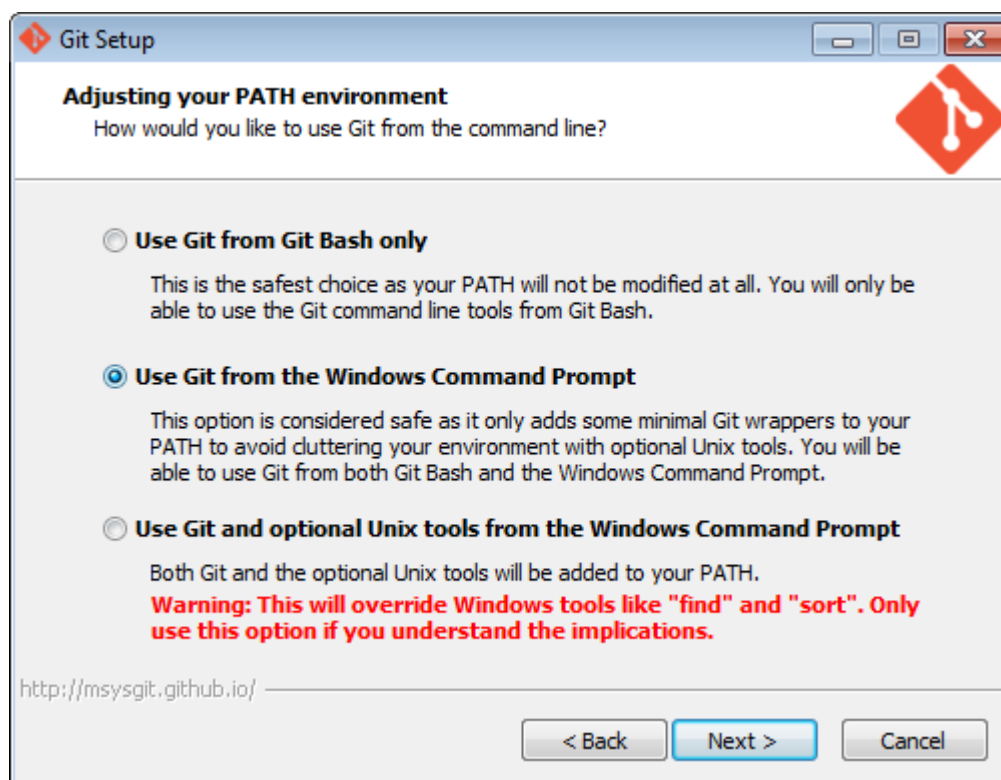
2. download and install Python 2.x (32bit). Grab it from the Python official site.

   When installing Python, do put Python in the PATH:

3. download `pygtk` from the official source. (this requires 32bit python). be sure you download the "all in one" version.

   Make a complete install, selecting everything:
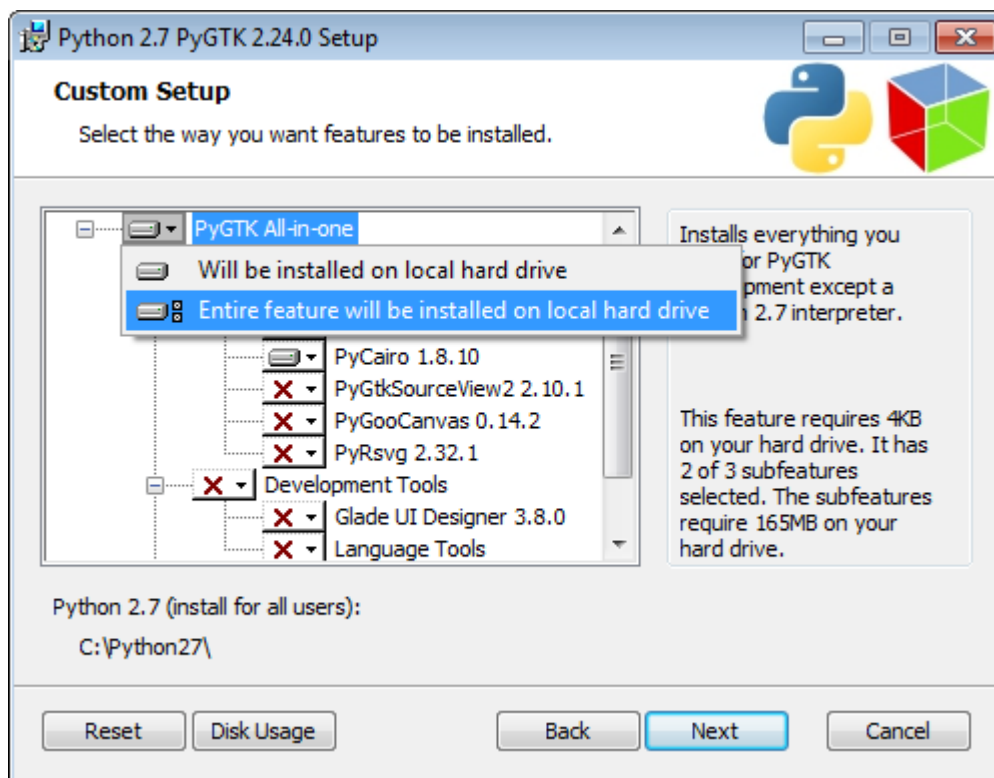
4. (Possibly necessary, maybe superfluous) install lxml, you can grab this from the pypi

archives

Remember you need the 32 bit version, for Python 2.7.

5. (definitely optional) download and install a database connector other than `sqlite3`.

If you plan using PostgreSQL, the best Windows binary library for Python is psycopg and is Made in Italy.

6. **REBOOT**

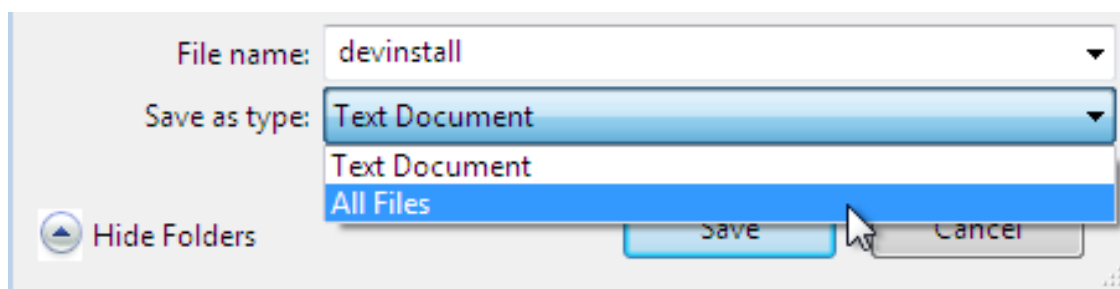hey, this is Windows, you need to reboot for changes to take effect!

7. We're done with the dependecies, now we can download and run the batch file:

devinstall.bat

Please don't just follow the above link. Instead: right click, save link as. . .
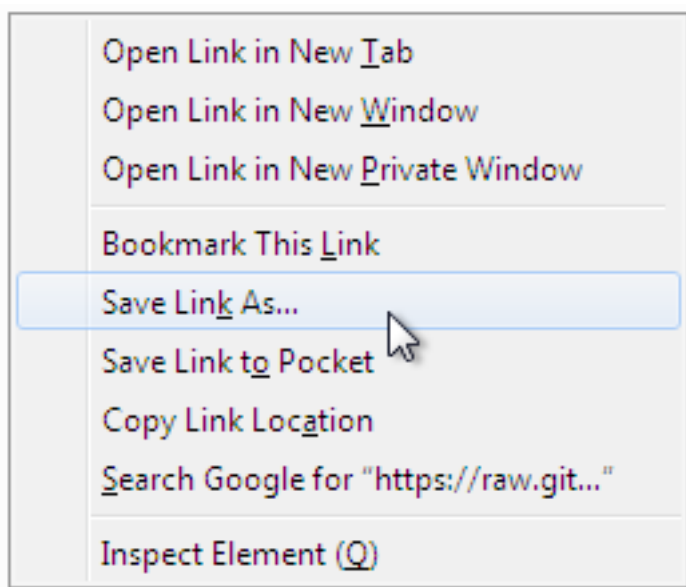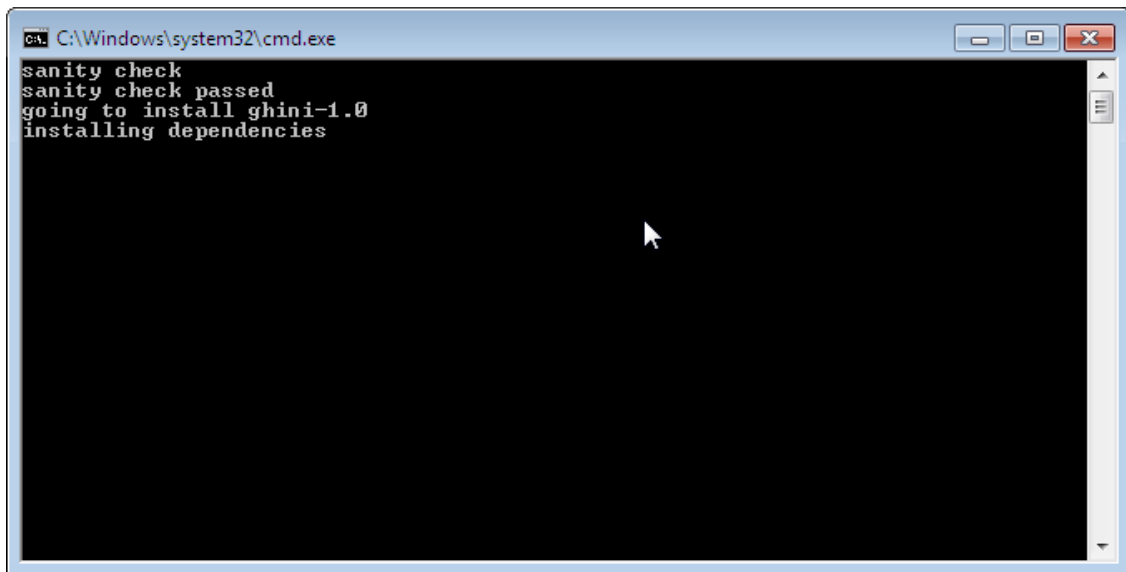

Also make sure you don't let Windows convert the script to a text document.

Now **Open** the script to run it. Please note: in the below image, we have saved the file twice, once letting Windows convert it to a text document, and again as a Windows Batch File. Opening the batch file will run the script. Opening the text document will show you the code of the batch file, which isn't going to lead us anywhere.
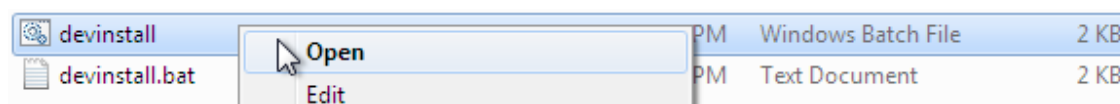
If you installed everything as described here, the first thing you should see when you start the installation script is a window like this, and your computer will be busy during a couple of minutes, showing you what it is doing.


Running `devinstall.bat` will pull the `ghini.desktop` repository from github to your home directory, under `Local\github\Ghini`, checkout the `ghini-3.1` production line, create a virtual environment and install ghini into it.

You can also run `devinstall.bat` passing it as argument the numerical part of the production line you want to follow.

This is the last installation step that depends, heavily, on a working internet connection.

The operation can take several minutes to complete, depending on the speed of your internet connection.

8. the last installation step creates the Ghini group and shortcuts in the Windows Start Menu, for all users. To do so, you need run a script with administrative rights. The script is called `devinstall-finalize.bat`, it is right in your HOME folder, and has been created at the previous step.
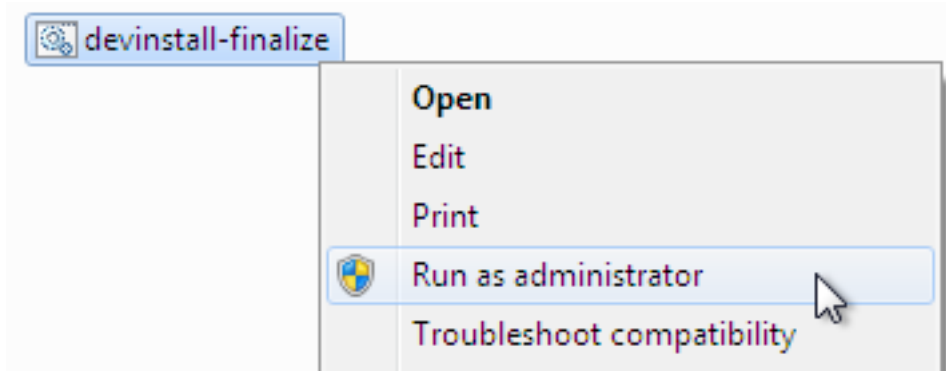
Right-click on it, select run as administrator, confirm you want it to make changes to your computer. These changes are in the Start Menu only: create the Ghini group, place the Ghini shortcut.

9. download the batch file, it will help you staying up-to-date:

   ghini-update.bat

   If you are on a recent Ghini installation, each time you start the program, Ghini will check on the development site and alert you of any newer ghini release within your chosen production line.

   Any time you want to update your installation, just run the `ghini-update.bat` script, it will hardly take one minute.

   How to save a batch file, and how to run it: check the the quite detailed instructions given for `devinstall.bat`.

If you would like to generate and print PDF reports using Ghini's default report generator then you will need to download and install Apache FOP. After extracting the FOP archive you will need to include the directory you extracted to in your PATH.

Next...

Connecting to a database.

2.1.4 Installing on Android

`ghini.desktop` is a desktop program, obviously you don't install it on a handheld device, but we do offer the option, for your Android phone or tablet, to install `ghini.pocket`.

`ghini.pocket` is a small data viewer, it comes handy if you want to have a quick idea of a plant species, its source, and date it entered the garden, just by scanning a plant label.

Installation is as easy as it can be: just look for it in Google Play, and install it.

Export the data from `ghini.desktop` to pocket format, copy it to your device, enjoy.

User's Guide

3.1 Initial Configuration

After a successful installation, more complex organizations will need configure their database, and configure Ghini according to their database configuration. This page focuses on this task. If you don't know what this is about, please do read the part relative to SQLite.

3.1.1 Should you SQLite?

Is this the first time you use Ghini, are you going to work in a stand-alone setting, you have not the faintest idea how to manage a database management system? If you answered yes to any of the previous, you probably better stick with SQLite, the easy, fast, zero-administration file-based database.

With SQLite, you do not need any preparation and you can continue with *connecting*.

On the other hand, if you want to connect more than one bauble workstation to the same database, or if you want to make your data available for other clients, as could be a web server in a LAMP setting, you should consider keeping your database in a database management system like PostgreSQL or MySQL/MariaDB, both supported by Ghini.

When connecting to a database server as one of the above, you have to manually do the following: Create at least one user; Create your database; Give at least one user full permissions on your database; If you plan having more database users: Give one of your users the `CREATEROLE` privilege; Consider the user with the `CREATEROLE` privilege as a super-user, not meant to handle data directly; Keep your super-user credentials in a very safe place.

When this is done, Ghini will be able to proceed, creating the tables and importing the default data set. The process is database-dependent and it falls beyond the scope of this manual.

If you already got the chills or sick at your stomach, no need to worry, just stick with SQLite, you do not miss on features nor performance.

Some more hints if you need PostgreSQL

Start simple, don't do all at the same time. Review the online manual, or download and study the offline version.

As said above, create a database, a user, make this user the owner of the database, decide whether you're going to need multiple users, and preferably reserve a user for database and normal user creation. This super-user should be your only user with CREATEROLE privilege.

All normal users will need all privileges on all tables and sequences, something you can do from the *Tools→Users* menu. If you have any difficulty, please open an issue about it.

Connect using the psql interactive terminal. Create a ~/.pgpass file (read more about it in the manual), tweak your pg_hba.conf and postgresql.conf files, until you can connect using the command:

```
psql <mydb> --username <myuser> --no-password --host
↪<mydbhost>
```

With the above setup, connecting from ghini will be an obvious task.

3.1.2 Connecting to a database

When you start Ghini the first thing that comes up is the connection dialog.

Quite obviously, if this is the first time you start Ghini, you have no connections yet and Ghini will alert you about it.

This alert will show at first activation and also in the future if your connections list becomes empty. As it says: click on **Add** to create your first connection.

Just insert a name for your connection, something meaningful you associate with the collection to be represented in the database (for example: "my home garden"), and click on **OK**. You will be back to the previous screen, but your connection name will be selected and the Connection Details will have expanded.

specify the connection details

If you do not know what to do here, Ghini will help you stay safe. Activate the **Use default locations** check box and create your first connection by clicking on **Connect**.

You may safely skip the remainder of this section for the time being and continue reading to the following section.

fine-tune the connection details

By default Ghini uses the file-based SQLite database. During the installation process you had the choice (and you still have after installation), to add database connectors other than the default SQLite.

In this example, Ghini can connect to SQLite, PostgreSQL and MySQL, but no connector is available for Oracle or MS SQL Server.
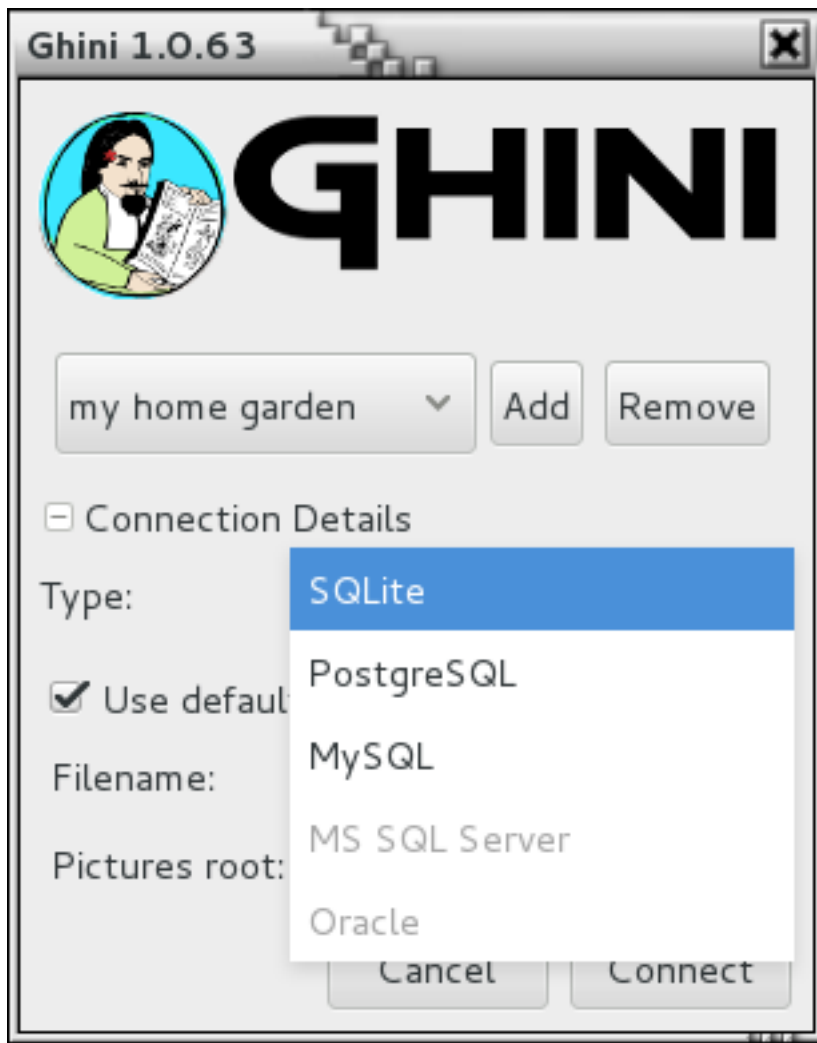
If you use SQLite, all you really need specify is the connection name. If you let Ghini use the default filename then Ghini creates a database file with the same name as the connection and `.db` extension, and a pictures folder with the same name and no extension, both in `~/.bauble` on Linux/MacOSX or in `AppData\Roaming\Bauble` on Windows.

Still with SQLite, you might have received or downloaded a bauble database, and you want to connect to it. In this case you do not let Ghini use the default filename, but you browse in your computer to the location where you saved the Ghini SQLite database file.

If you use a different database connector, the dialog box will look different and it will offer you the option to fine tune all parameters needed to connect to the database of your choice.

If you are connecting to an existing database you can continue to *Editing and Inserting Data* and subsequently searching-in-ghini, otherwise read on to the following section on initializing a database for Ghini.

If you plan to associate pictures to plants, specify also the *pictures root* folder. The meaning of this is explained in further detail at *Pictures* in *Editing and Inserting Data*.

A sample SQLite database

　　Indeed we have a sample database, from our pilot garden "El Cuchubo", in Mom-

pox, Colombia. We have both a zipped sample database for ghini-1.0 and the same equally zipped sample database for ghini-3.1.

Both zip files contain a SQLite database file, named `cuchubo.db`, but they are **not** the same file: one is to be used with ghini-1.0, one for the newer ghini-3.1. You cannot mix versions.

Download the one corresponding to the ghini version you installed, unzip it to the location of your choice, then start Ghini, create a connection named possibly `cuchubo`, or `sample`, and edit the Connection Details. Keep the connection type at the default SQLite, but instead of using the default locations, make sure that Filename points to your unpacked `cuchubo.db` file.

3.1.3 Initialize a database

First time you open a connection to a database which had never been seen by Ghini before, Ghini will first display an alert:
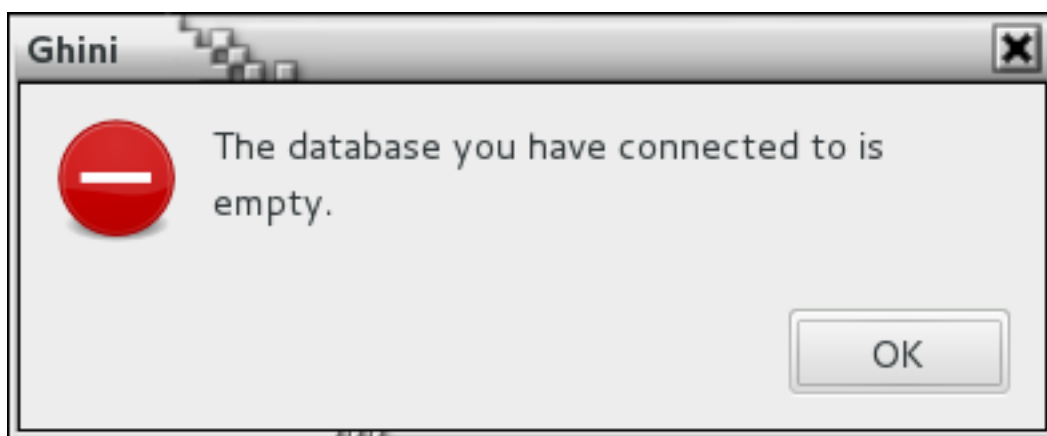
immediately followed by a question:
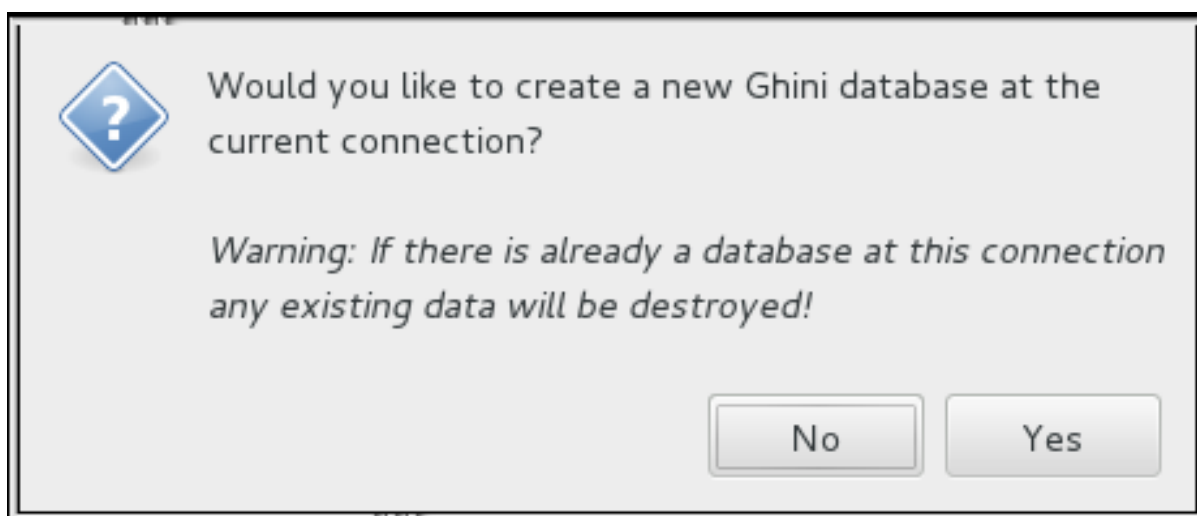
Be careful when manually specifying the connection parameters: the values you have entered may refer to an existing database, not intended for use with Ghini. By letting Ghini initialize a

database, the database will be emptied and all of its content be lost.

If you are sure you want to create a database at this connection then select "Yes". Ghini will then start creating the database tables and importing the default data. This can take a minute or two so while all of the default data is imported into the database so be patient.

Once your database has been created, configured, initialized, you are ready to start *Editing and Inserting Data* and subsequently *Searching in Ghini*.

3.2 Searching in Ghini

Searching allows you to view, browse and create reports from your data. You can perform searches by either entering the queries in the main search entry or by using the Query Builder to create the queries for you. The results of Ghini searches are listed in the main window.

3.2.1 Search Strategies

Ghini offers four distinct search strategies:

- by value — in all domains;
- by expression — in a few implicit fields in one explicit domain;
- by query — in one domain;
- by binomial name — only searches the Species domain.

All search strategies —with the notable exception of the binomial name search— are case insensitive.

Search by Value

Search by value is the simplest way to search. You enter one or more strings and see what matches. The result includes objects of any type (domain) where one or more of its fields contain one or more of the search strings.

You don't specify the search domain, all are included, nor do you indicate which fields you want to match, this is implicit in the search domain.

The following table helps you understand the results and guides you in formulating your searches.

search domain overview		
name and shorthands	field	result type
family, fam	epithet (family)	`Family`
genus, gen	epithet (genus)	`Genus`
species, sp	epithet (sp) ✗	`Species`
vernacular, common, vern	name	`Species`
geography, geo	name	`Geography`
accession, acc	code	`Accession`
planting, plant	code ✗	`Plant`
location, loc	code, name	`Location`
contact, person, org, source	name	`Contact`
collection, col, coll	locale	`Collection`
tag, tags	name	`Tag`

Examples of searching by value would be: Maxillaria, Acanth, 2008.1234, 2003.2.1, indica.

Unless explicitly quoted, spaces separate search strings. For example if you search for `Block 10` then Ghini will search for the strings Block and 10 and return all the results that match either of these strings. If you want to search for Block 10 as one whole string then you should quote the string like `"Block 10"`.

✗ Composite Primary Keys

A **species** epithet means little without the corresponding genus, likewise a **planting** code is unique only within the accession to which it belongs. In database theory terminology, epithet and code are not sufficient to form a **primary key** for respectively species and planting. These domains need a **composite** primary key.

Search by value lets you look for **plantings** by their complete planting code, which includes the accession code. Taken together, Accession code and Planting code do provide a **composite primary key** for plantings. For **species**, we have introduced the binomial search, described below.

Search by Expression

Searching with expression gives you a little more control over what you are searching for. You narrow the search down to a specific domain, the software defines which fields to search within the domain you specified.

An expression is built as `<domain> <operator> <value>`. For example the search: `gen=Maxillaria` would return all the genera that match the name Maxillaria. In this case the domain is `gen`, the operator is = and the value is `Maxillaria`.

The above search domain overview table tells you the names of the search domains, and, per search domain, which fields are searched.

The search string `loc like block%` would return all the Locations for which name or code start with "block". In this case the domain is `loc` (a shorthand for `location`), the operator

is `like` (this comes from SQL and allows for "fuzzy" searching), the value is `block%`, the implicitly matched fields are `name` and `code`. The percent sign is used as a wild card so if you search for `block%` then it searches for all values that start with `block`. If you search for `%10` it searches for all values that end in `10`. The string `%ck%10` would search for all value that contain `ck` and end in `10`.

When a query takes ages to complete

You give a query, it takes time to compute, the result contains unreasonably many entries. This happens when you intend to use a strategy, but your strings do not form a valid expression. In this case Ghini falls back to *search by value*. For example the search string `gen lik maxillaria` will search for the strings `gen`, `lik`, and `maxillaria`, returning all that match at least one of the three criteria.

Binomial search

You can also perform a search in the database if you know the species, just by placing a few initial letters of genus and species epithets in the search engine, correctly capitalized, i.e.: **Genus epithet** with one leading capital letter, **Species epithet** all lowercase.

This way you can perform the search `So ha`.

These would be the initials for Solanum hayesii, or Solanum havanense.

Binomial search comes to compensate the limited usefulness of the above search by expression when trying to look for a species.

It is the correct capitalization **Xxxx xxxx** that informs the software of your intention to perform a binomial search. The software's second guess will be a search by value, which will possibly result in far more matches than you had expected.

The similar request `so ha` will return, in a fresh install, over 3000 objects, starting at Family "Acalyp(**ha**)ceae", ending at Geography "Western (**So**)uth America".

Search by Query

Queries allow the most control over searching. With queries you can search across relations, specific columns, combine search criteria using boolean operators like `and`, `or`, `not` (and their shorthands `&&`, `||`, `!`), enclose them in parentheses, and more.

Please contact the authors if you want more information, or if you volunteer to document this more thoroughly. In the meanwhile you may start familiarizing yourself with the core structure of Ghini's database.

Fig. 1: **core structure of Ghini's database**

A few examples:

- plantings of family Fabaceae in location Block 10:

```
plant WHERE accession.species.genus.family.epithet=Fabaceae AND␣
→location.description="Block 10"
```

- locations that contain no plants:

```
location WHERE plants = Empty
```

- accessions associated to a species of known binomial name (e.g.: Mangifera indica):

```
accession WHERE species.genus.epithet=Mangifera AND species.
→epithet=indica
```

- accessions we propagated in the year 2016:

```
accession WHERE plants.propagations._created BETWEEN␣
→|datetime|2016,1,1| AND |datetime|2017,1,1|
```

- accessions we modified in the last three days:

```
accession WHERE _last_updated>|datetime|-3|
```

Searching with queries requires some knowledge of a little syntax and an idea of the extensive Ghini database table structure. Both you acquire with practice, and with the help of the Query Builder.

3.2.2 The Query Builder

Ghini offers a Query Builder, that helps you build complex search queries through a point and click interface. To open the Query Builder click the [icon] icon to the left of the search entry or select *Tools→Query Builder* from the menu.

A window will show up, which will lead you through all steps necessary to construct a correct query that is understood by Ghini's Query Search Strategy.
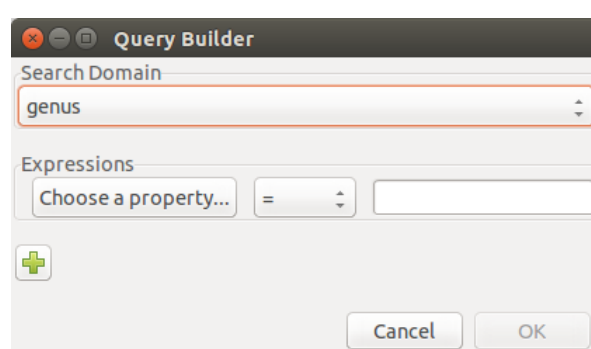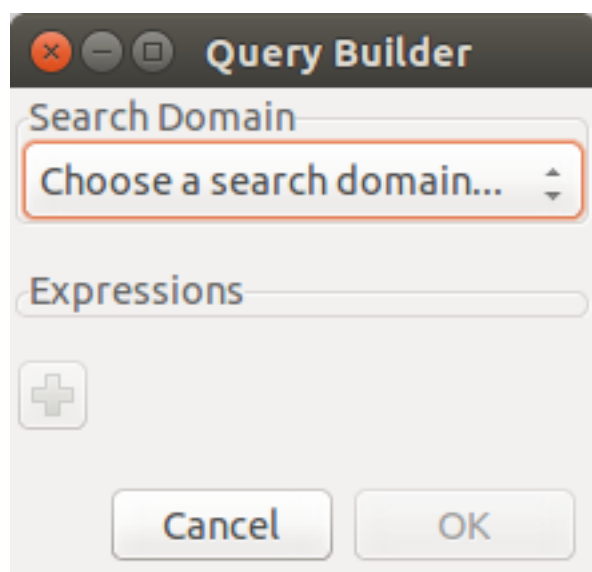
First of all you indicate the search domain, this will allow the Query Builder complete its graphical user interface, then you add as many logical clauses as you need, connecting them with a `and` or `or` binary operator.

Each clause is formed of three parts: a property that can be reached from the starting search domain, a comparison operator that you select from the drop-down list, a value that you can either type or select from the list of valid values for the field.

Add as many search properties as you need, by clicking on the plus sign. Select and/or next to the property name to choose how the clauses will be combined in the search query.

When you are done building your query click OK to perform the search.

At this point the Query Builder writes the query in the search entry, and executes it. You may now edit the string as if you had typed it yourself. Notice how the left hand side values are interpreted by the query builder and enclosed in single quotes if recognized as strings, left alone if they look like numbers or the two reserved words `None` and `Empty`. You may edit the query and insert quotes if you need them, eg if you need to literally look for the string `Empty`.

`None` is the value of an empty field. It is not the same as the zero lenght string `''` nor the numeric `0` nor the boolean `False` nor the set `Empty`, it indicates that the field has no value at all.

`Empty` is the empty set. Being it a set, it can be matched against sets (eg: plants of an accession, or accessions of a species), not against elements (eg: quantity of a plant or description of a location). However, the Query Builder does not let you choose a left hand side value stopping at a set, it expects you to select a field. Choose just any field: at the moment of producing the query, when the Query Builder meets a clause with right hand side value the literal string `Empty`, it will drop the field name and let you compare the set on the left with `Empty` on the right.

We have no literals `False` and `True`. These are typed values, and the Query Builder does not know how to produce them. Instead of `False` type `0`, and instead of `True` type `1`.

3.2.3 Query Grammar

For those who don't fear a bit of formal precision, the following BNF code gives you a rather precise idea of the grammar implemented by the Query Search Strategy. Some grammatical categories are informally defined; any missing ones are left to your fertile imagination; literals are included in single quotes; the grammar is mostly case insensitive, unless otherwise stated:

```
query ::= domain 'WHERE' expression


domain ::= #( one of our search domains )
expression ::= signed_clause
             | signed_clause 'AND' expression
             | signed_clause 'OR' expression
             ;
signed_clause ::= clause
```

(continues on next page)

```
                      | 'NOT' clause  #( not available in Query Builder)
                   ;
clause ::= field_name binop value  #( available in Query Builder)
           | field_name set_binop value_list
           | aggregated binop value
           | field_name 'BETWEEN' value 'AND' value
           | '(' expression ')'
         ;
field_name ::= #( path to reach a database field or connected table
 ↪)
aggregated ::= aggregating_func '(' field_name ')'
aggregating_func ::= 'SUM'
                   | 'MIN'
                   | 'MAX'
                   | 'COUNT'
                   ;
value ::= typed_value
          | numeric_value
          | none_token
          | empty_token
          | string_value
        ;
typed_value ::= '|' type_name '|' value_list '|'
numeric_value ::== #( just a number )
none_token ::= 'None'    #( case sensitive )
empty_token ::= 'Empty'  #( case sensitive )
string_value = quoted_string | unquoted_string

type_name ::= 'datetime' | 'bool' ;  #( only ones for the time
 ↪being )
quoted_string ::= '"' unquoted_string '"'
unquoted_string ::=  #( alphanumeric and more )

value_list ::= value ',' value_list
             | value
           ;
binop ::= '='
          | '=='
          | '!='
          | '<>'
          | '<'
          | '<='
          | '>'
          | '>='
          | 'LIKE'
          | 'CONTAINS'
        ;
set_binop ::= 'IN'
```

Please be aware that Ghini's Query language is quite a bit more complex than what the Query

Builder can produce: Queries you can build with the Query Builder form a proper subset of the queries recognized by the software:

```
query ::= domain 'WHERE' expression

domain ::= #( one of our search domains )
expression ::= clause
             | clause 'AND' expression
             | clause 'OR' expression
             ;
clause ::= field_name binop value
         ;
field_name ::= #( path to reach a database field or connected table
↪)
value ::= numeric_value
        | string_value
        ;
numeric_value ::== #( just a number )
string_value = quoted_string | unquoted_string ;

quoted_string ::= '"' unquoted_string '"'
unquoted_string ::=  #( alphanumeric and more )

binop ::= '='
        | '=='
        | '!='
        | '<>'
        | '<'
        | '<='
        | '>'
        | '>='
        | 'LIKE'
        | 'CONTAINS'
        ;
```

3.3 Editing and Inserting Data

The main way that we add or change information in Ghini is by using the editors. Each basic type of data has its own editor. For example there is a Family editor, a Genus editor, an Accession editor, etc.

To create a new record click on the *Insert* menu on the menubar and then select the type of record your would like to create. This opens a new blank editor for the type.

To edit an existing record in the database right click on an item in the search results and select *Edit* from the popup menu. This opens an editor that allows you to change the values on the record that you selected.

Most types also have children which you can add by right clicking on the parent and selecting "Add ???..." on the context menu. For example, a Family has Genus children: you can add a

Genus to a Family by right clicking on a Family and selecting "Add genus".

3.3.1 Notes

Almost all of the editors in Ghini have a *Notes* tab which should work the same regardless of which editor you are using.

If you enter a web address in a note then the link shows up in the Links box when the item your are editing is selected in the search results.

You can browse the notes for an item in the database using the Notes box at the bottom of the screen. The Notes box is desensitized if the selected item does not have any notes.

3.3.2 Family

The Family editor allows you to add or change a botanical family.

The *Family* field on the editor lets you change the epithet of the family. The Family field is required.

The *Qualifier* field lets you change the family qualifier. The value can either be *sensu lato*, *sensu stricto*, or nothing.

Synonyms allow you to add other families that are synonyms with the family you are currently editing. To add a new synonyms type in a family name in the entry. You must select a family name from the list of completions. Once you have selcted a family name that you want to add as a synonym click on the Add button next to the synonym list and the software adds the selected synonym to the list. To remove a synonym, select the synonym from the list and click on the Remove button.

To cancel your changes without saving then click on the *Cancel* button.

To save the family you are working on then click *OK*.

To save the family you are working on and add a genus to it then click on the *Add Genera* button.

To add another family when you are finished editing the current one click on the *Next* button on the bottom. This saves the current family and opens a new blank family editor.

3.3.3 Genus

The Genus editor allows you to add or change a botanical genus.

The *Family* field on the genus editor allows you to choose the family for the genus. When you begin type a family name it will show a list of families to choose from. The family name must already exist in the database before you can set it as the family for the genus.

The *Genus* field allows you to set the genus for this entry.

The *Author* field allows you to set the name or abbreviation of the author(s) for the genus.

Synonyms allow you to add other genera that are synonyms with the genus you are currently editing. To add a new synonyms type in a genus name in the entry. You must select a genus name from the list of completions. Once you have selcted a genus name that you want to add as a synonym click on the Add button next to the synonym list and it will add the selected synonym to the list. To remove a synonym select the synonym from the list and click on the Remove button.

To cancel your changes without saving then click on the *Cancel* button.

To save the genus you are working on then click *OK*.

To save the genus you are working on and add a species to it then click on the *Add Species* button.

To add another genus when you are finished editing the current one click on the *Next* button on the bottom. This will save the current genus and open a new blank genus editor.

3.3.4 Species/Taxon

For historical reasons called a *species*, but by this we mean a *taxon* at rank *species* or lower. It represents a unique name in the database. The species editor allows you to construct the name as well as associate metadata with the taxon such as its distribution, synonyms and other information.

The *Infraspecific parts* in the species editor allows you to specify the *taxon* further than at *species* rank.

To cancel your changes without saving then click on the *Cancel* button.

To save the species you are working on then click *OK*.

To save the species you are working on and add an accession to it then click on the *Add Accession* button.

To add another species when you are finished editing the current one click on the *Next* button on the bottom. This will save the current species and open a new blank species editor.

3.3.5 Accessions

The Accession editor allows us to add an accession to a species. In Ghini an accession represents a group of plants or clones that are of the same taxon, are of the same propagule type (or treatment), were received from the same source, were received at the same time.

Choose the Taxon name, add one if you forgot to do that in advance.

You may note uncertainty in identification by adding an identification qualifier, at the proper rank, so you can for example have a plant initially identified as *Iris* cf. *florentina* by choosing *Iris florentina* in the taxon name, identification qualifier 'cf.', qualified rank 'species'.

Type the Accession ID, preferably also the Quantity received.

Accession Source

The source of the accessions lets you add more information about where this accession came from. Select a Contact from the drop-down list, or choose "Garden Propagation", which is placed as a default first item in the list of contacts.

A Garden Propagation is the result of successful Propagation.

When accessing material from a Garden Propagation, you would initially leave the first tab alone (General) and start from the second tab (Source). Select as Contact "Garden Propagation", indicate which plant is the parent plant and choose among the still not completely accessed propagations the one you intend to add as an accession in your database.

Once you select a propagation, the software will set several fields in the General tab, which you can now review. The Taxon (maybe you managed to obtain something slightly different than the parent plant). The Initial quantity (in case not all plants go in the same accession). The Type of Material, inferred from the propagation type.

3.3.6 Plant

A `Plant` in the Ghini database describes an individual plant in your collection. A plant belongs to an accession, and it has a specific location.

Creating multiple plants

You can create multiple Plants by using ranges in the code entry. This is only allowed when creating new plants and it is not possible when editing existing Plants in the database.

For example the range, 3-5 will create plant with code 3,4,5. The range 1,4-7,25 will create plants with codes 1,4,5,6,7,25.

When you enter the range in the plant code entry the entry will turn blue to indicate that you are now creating multiple plants. Any fields that are set while in this mode will be copied to all the plants that are created.

Pictures

Just as almost all objects in the Ghini database can have *Notes* associated to them, Plants and Species can also have *Pictures*: next to the tab for Notes, the Plant and the Species editors contain an extra tab called "Pictures". You can associate as many pictures as you might need to a plant and to a species object.

When you associate a picture to an object, the file is copied in the *pictures* folder, and a miniature (500x500) is generated and copied in the *thumbnails* folder inside of the pictures folder.

As of Ghini-1.0.62, Pictures are not kept in the database. To ensure pictures are available on all terminals where you have installed and configured Ghini, you can use a network drive, or a file sharing service like Tresorit or Dropbox.

Remember that you have configured the pictures root folder when you specified the details of your database connection. Again, you should make sure that the pictures root folder is shared with your file sharing service of choice.

When a Plant or a Species in the current selection is highlighted, its pictures are displayed in the pictures pane, the pane left of the information pane. When an Accession in the selection is highlighted, any picture associated to the plants in the highlighted accession are displayed in the pictures pane.

In Ghini-3.1, pictures are special notes, with category "<picture>", and text the path to the file, relative to the pictures root folder. In the Notes tab, Picture notes will show as normal notes, and you can edit them without limitations.

A Plant is a physical object, so you associate to it pictures taken of that individual plant, taken at any relevant development stage of the plant, possibly helping its identification.

Species are abstract objects, so you would associate to it pictures showing the characteristic elements of the species, so it makes sense to associate a flora illustration to it. You can also do that by reference: go to the Notes tab, add a note and specify as category "<picture>", then in the text field you type the URL for the illustration of your choice.

3.3.7 Locations

The Location editor

danger zone

The location editor contains an initially hidden section named *danger zone*. The widgets contained in this section allow the user to merge the current location into a different location, letting the user correct spelling mistakes or implement policy changes.

3.4 Dealing with Propagations

Ghini offers the possibility to associate Propagations trials to Plants and to document their treatments and results. Treatments are integral parts of the description of a Propagation trial. If a Propagation trial is successful, Ghini lets you associate it to a new Accession. You can only associate one Accession to a Propagation Trial.

Here we describe how you use this part of the interface.

3.4.1 Creating a Propagation

A Propagation (trial) is obtained from a Plant. Ghini reflects this in its interface: you select a plant, open the Plant Editor on it, activate the Propagation Tab, click on Add.

When you do the above, you get a Propagation Editor window. Ghini does not consider Propagation trials as independent entities. As a result, Ghini treats the Propagation Editor as a special editor window, which you can only reach from the Plant Editor.

For a new Propagation, you select the type of propagation (this becomes an immutable property of the propagation) then insert the data describing it.

You will be able to edit the propagation data via the same path: select a plant, open the Plant Editor, identify the propagation you want to edit, click on the corresponding Edit button. You will be able to edit all properties of an existing Propagation trial, except its type.

In the case of a seed propagation trial, you have a pollen parent, and a seed parent. You should always associate the Propagation trial to the seed parent.

Note: In Ghini-1.0 you specify the pollen parent plant in the "Notes" field, while Ghini-1.1 has a (relation) field for it. According to ITF2, there might be cases in seed propagation trials where it is not known which Plant plays which role. Again, in Ghini-1.0 you should use a note to indicate whether this is the case, Ghini-1.1 has a (boolean) field indicating whether this is the case.

3.4.2 Using a Propagation

A Propagation trial may be successful and result in a new Accession.

Ghini helps you reflect this in the database: create a new Accession, immediately switch to the Source tab and select "Garden Propagation" in the (admittedly somewhat misleading) Contact field.

Start typing the plant number and a list of matching plants with propagation trials will appear for you to select from.

Select the plant, and the list of accessed and unaccessed propagation trials will appear in the lower half of the window.

Select a still unaccessed propagation trial from the list and click on Ok to complete the operation.

Using the data from the Propagation trial, Ghini completes some of the fields in the General tab: Taxon name, Type of material, and possibly Provenance. You will be able to edit these fields, but please note that the software will not prevent introducing conceptual inconsistencies in your database.

You can associate a Propagation trial to only one Accession.

3.5 Tagging

Tagging is an easy way to give context to an object or create a collection of object that you want to recall later.

The power in this tagging action is that you can share this selection with colleagues, who can act on it, without the need to redo all your collecting work.

For example if you need to print accession labels of otherwise unrelated plants, you can group the objects by tagging them with the string "relabel". You or one of your colleagues can then select "relabel" from the tags menu, the search view will show all the objects you tagged, and performing a report will act on the tagged objects.

Tagging acts on the active selection, that is the items in the search results which you have selected.

Please remember: you can select all result rows by pressing `Ctrl-A`, you can deselect everything by pressing `Ctrl-Shift-A`, you can toggle tagging of a single row by `Ctrl-Mouse click` on it.

Once you have an active selection, tagging can be done in two ways:

3.5.1 dialog box tagging

Press `Ctrl-T` or select *Tag→Tag Selection* from the menu, this activates a window where you can create new tags and apply any existing tag to the selection.

The tag window is composed of three parts:

1. The upper part mentions the list of objects in your active selection. This is the list of object of which you are editing the tags;

2. The middle part has a list of all available tags, with a checkbox that you can activate for applying the tag to or removing the tag from the selection;

3. The lower part only holds a link to new tag creation, and the Ok button for closing the dialog box.

If, when opening the tag dialog box, the active selection holds multiple items, then only the tags that are common to all the selected items will have a check next to it. Tags that only apply to a proper subset of the active selection will show with an 'undecided' status. Tags that don't apply to any object in the active selection will show blank.

The most recently created tag, or the most recently selected tag becomes the active tag, and it shows with a check next to it in the tags menu.

3.5.2 windowless tagging

Once you have an active tag, pressing `Ctrl-Y` applies the active tag to all objects in the active selection. `Ctrl-Shift-Y` removes the active tag from all objects in the active selection.

3.6 Generating reports

A database without exporting facilities is of little use. Ghini lets you export your data in table format (files you can open in a spreadsheet program), as labels (to be printed or engraved), as

html pages, pdf or postscript documents, and the list could go on. Table format files we also call 'flat files', and the most common example would be CSV files.

3.6.1 The Report Tools

Ghini has two Report Tools, one based on templates, and a quick solution for flat file production. The `Template` based tool is as flexible as a programming language can be; the `Quick CSV` tool shares much of its user interface with the query builder.

You activate the Report Tools from the main menu: *Tools→Report*, then choose either `From Template`, or `Quick CSV`. Both Report Tools act on the current result in the results view, so you first select something, then start the Report Tool.
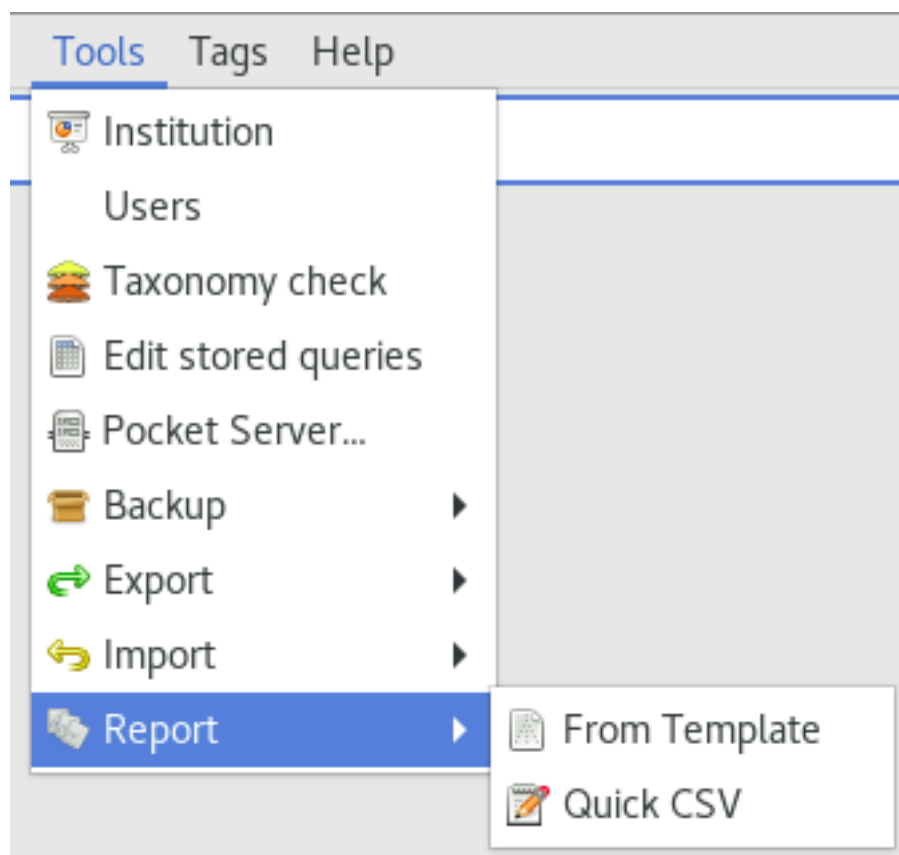
Selecting the whole collection.

> If you want to produce a report regarding your whole collection, you can do it from at least two points of view: you may want all the `Species`, or you may want all the `Locations`.

> A handy shortcut to get all your species in the selection, go to the home screen, then click on the `Families:   in use` cell.

> If your focus is more on the garden location than on taxonomy and accessions, you would click on the `Locations:   total` cell.

The *Quick CSV* report tool can act on the whole collection, regardless the content of the results view.

Template Based Reports

Activate the *Tools→Report→From Template* and you get the following dialog box (fields may be filled in differently):
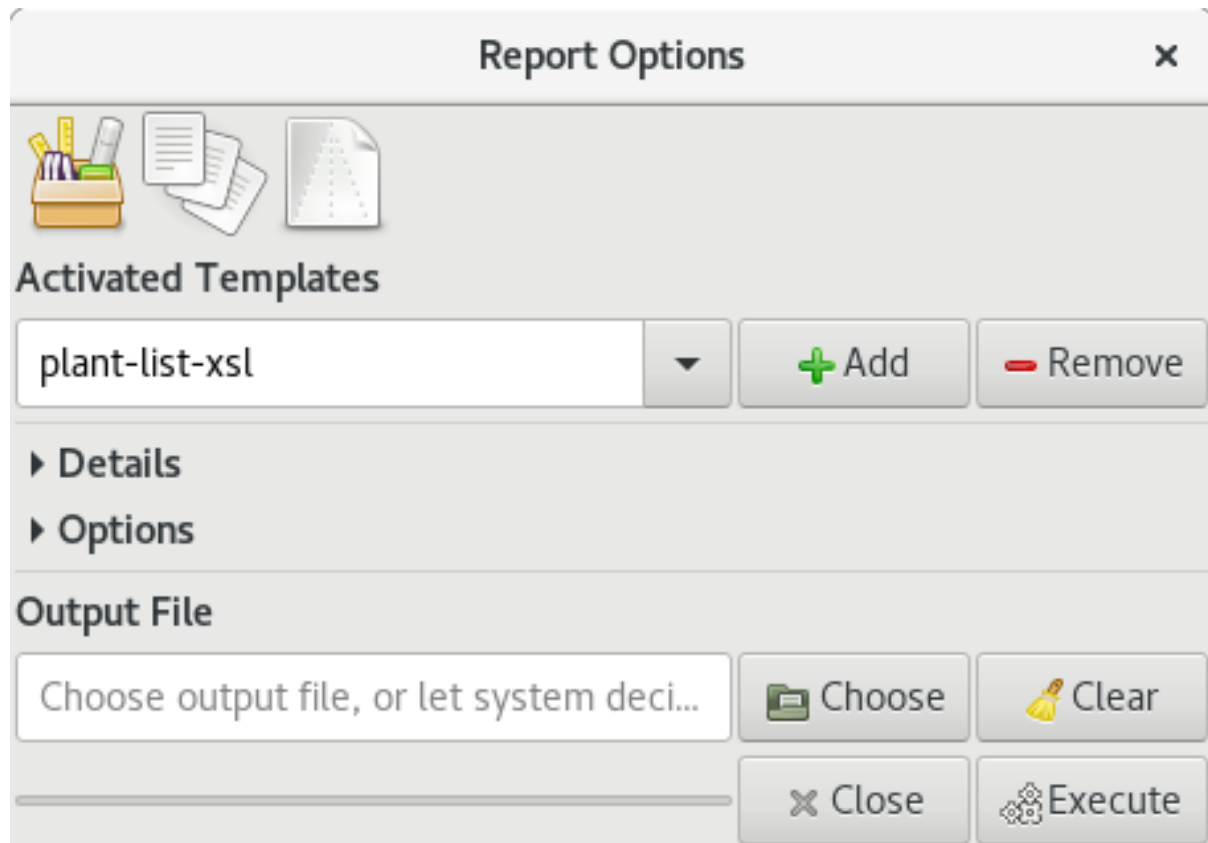
You have already selected your objects in the main window, you now here select a report template, then press on Execute to produce your report. Ghini will open the report in the corresponding application, if installed and enabled.

Ghini comes with a few sample package-templates, and it lets you install your own user-templates.

---

package-templates vs. user-templates

Package-templates are integral part of the installation and are overwritten every time you update your installation. User-templates are part of your custom data, just like your ghini configuration, your sqlite databases and your plant pictures. User-templates are persistent as long as you stay in the same production line (say, ghini-3.1).

To install a new template as a user-template, click on Add, choose the file that contains your template, confirm. ghini.desktop will copy it to your ghini

---
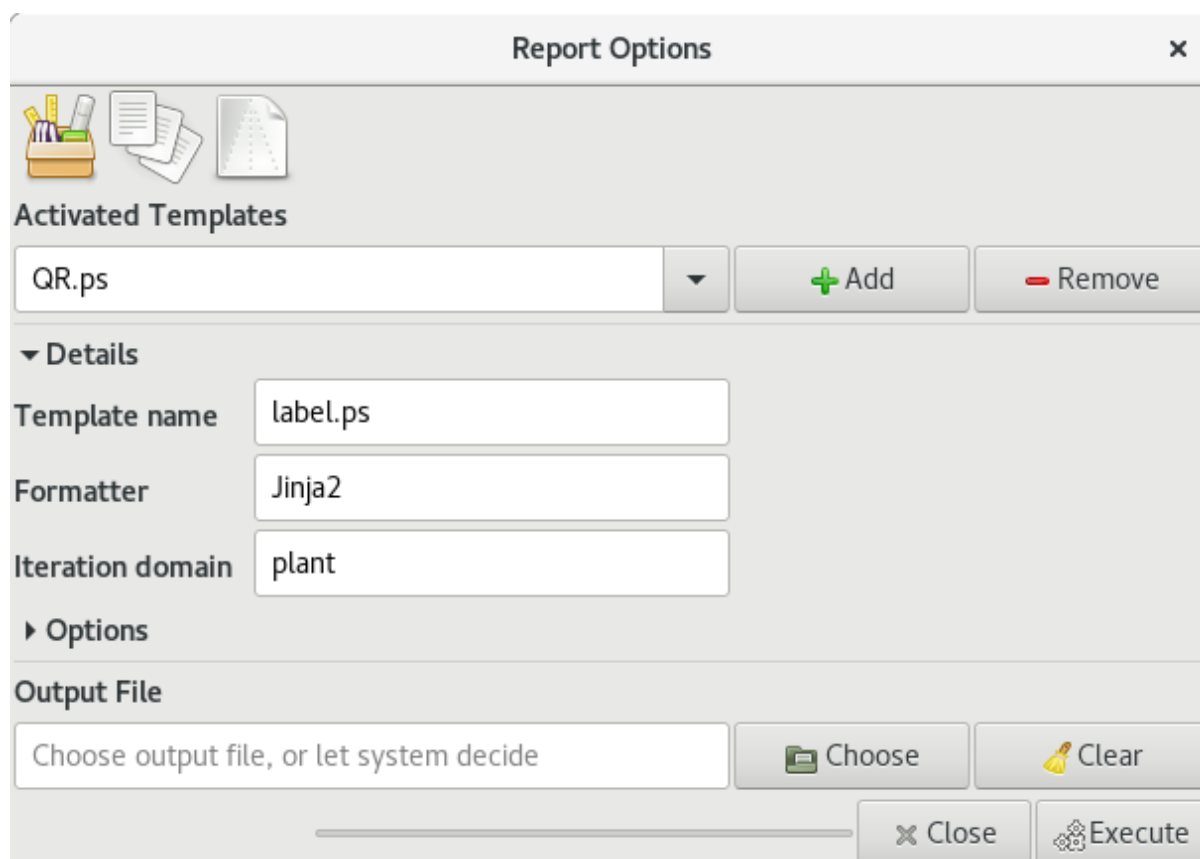
user directory.

You can delete installed user-templates, or disable package-templates: select the template name in the pull down list, click on `Remove`. Removing a user-templates deletes its file from your custom user data. Removing a package-template only disables it for the current user. You can enable all disabled package-templates by opening the `Add` dialog box, and clicking on the `Reset package-templates` button

A user can overrule a package-template, by installing it again as a user-template. Installed user-templates are just text files, you can edit them with any decent text editor. This allows you to alter package-templates, and to keep your changes across updates. You can use your preferred file handling tool to copy or rename your user-templates.

Choosing a template implies the choice of a template language. `ghini.desktop` supports three template languages: Jinja2, Mako and XSL. There is only one formatting engine handling the Jinja2 template language, and the same goes for Mako. During installation you indicated which of the several XSL formatting engines you wanted to have, if any.

Expand the `Details` section to see some information about the selected formatter template.

The formatter engine combines selection and template to produce a report. Each formatter template indicates its iteration domain, that is what kind of collection objects you focus on in your report. In the above example, we are using Jinja2 to report about individual plants, producing —per plant— a postscript label with a QR code.

Expand the `Options` section to see what extra parameters your selected template may require or expects.
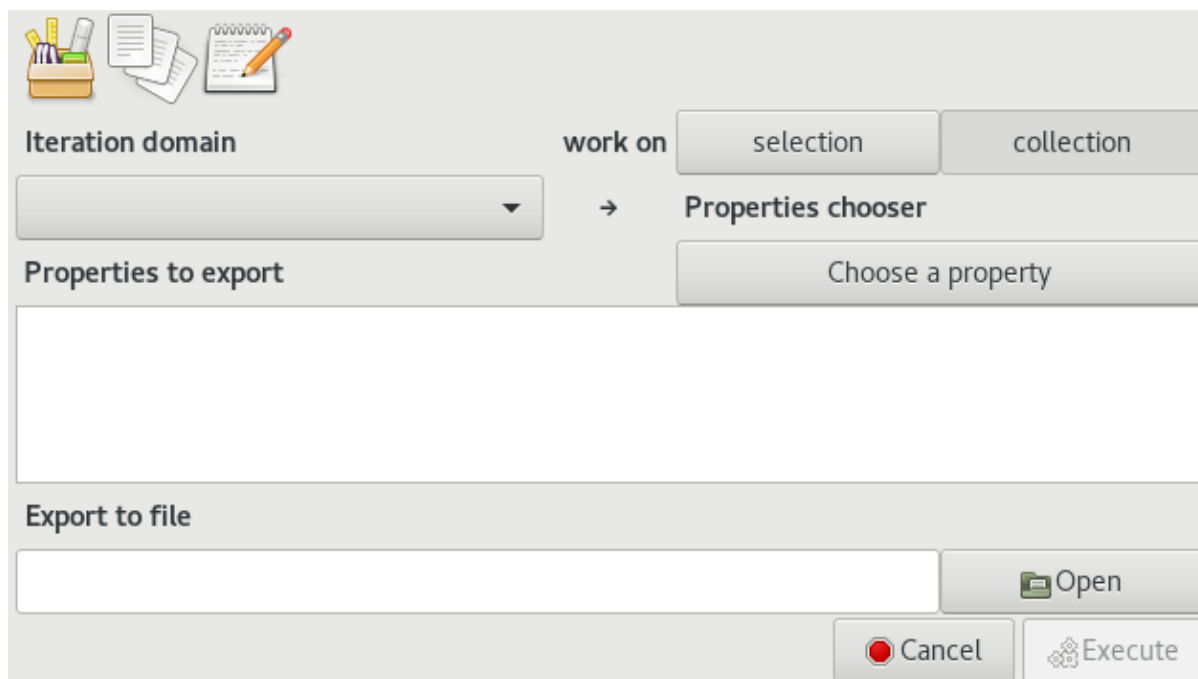

In the above example, the plant-labelling formatter lets you override the selection, and produce a set of labels in your preferred format, for a given range of accession codes.

In general: choose the report you need, specify parameters if required, and produce the report. Ghini will open the report in the associated application.

This is as far as generic information can go. Formatter templates can be very specific, and vary broadly from each other, most of them are small pieces of software themselves.

Template-less Reports (Quick CSV)

Activate the *Tools→Report→Quick CSV* and you get the following dialog box:

Start from the top and work your way to the bottom:

- decide whether you work on the selection or the whole collection,

- choose the iteration domain,

- select the properties to include in the report,

- drag and drop them in the list to get them in the correct order,

- choose the destination file,

- execute.

Ghini will open the report, a CSV file, in your preferred spreadsheet program.

Do you really need any further documentation? If anything isn't clear then please ask.

3.6.2 Technical information

The remainder of this page provides technical information and links regarding the formatter engines, and gives hints on writing report templates. Writing templates comes very close to writing a computer program, and that's beyond the scope of this manual, but we have hints that will definitely be useful to the interested reader.

Working with Templates Languages

Common information

Creating reports with Mako and Jinja2 technically equivalent to creatint a web page from a template. Both Mako and Jinja2 are indeed mostly used for dynamic creation of static web pages. This should be easy to do, for anyone with a little programming experience.

Ghini instructs the template generator to use the same file extension as the template, stripping the trailing `.mako` / `.jj2` part. The remaining template extension indicates the type of output produced by the template. The trailing `.mako` / `.jj2` helps ghini decide which template generator to use, and prevents you from mistaking a template for an output file.

For example, to generate an HTML page from a Mako template, you would name the template something like `report.html.mako`. Similarly, you would name a template `report.csv.jj2` if it uses Jinja2 to generate a comma separated value file.

A template must declare its iteration domain, that is, on which type of objects it reports. The iteration domain is declared in a comment line, something like this (for Mako):

```
## DOMAIN <name>
```

or this (for Jinja2):

```
{# DOMAIN <name> #}
```

Here `<name>` is one of `Species`, `Accession`, `Plant`, `Location`, or `raw`. The keyword `DOMAIN` needs be all upper case, but the declaration is further case insensitive.

The role of the `DOMAIN` declaration is to instruct ghini about the data that the template needs to receive: when rendering a template, ghini starts by building a raw list containing all top-level objects in the current result. If the declared iteration domain is `raw`, ghini will pass this raw list to the template. If the declared iteration domain is a ghini class, ghini will then build a list of all objects in the iteration domain, associated to the raw list. E.g.: if you select a couple of locations, and the report declares `DOMAIN Accession`, ghini will collect all accessions having plants in one of the selected locations.

These objects are available to the template as elements of the list `values`.

A template declaring `DOMAIN raw` needs more programming logic. On the other hand, a small, well-thought set of such templates can reduce the amount of template names that your users need to handle.

A template may require extra options, that can the user will define at run time. These are described in comment lines, like in this Mako example:

```
## OPTION accession_first: (type: integer, default:
'', tooltip: 'start of range.')
```

The Jinja2 equivalent of the above is:

```
{# OPTION accession_first: (type: integer, default:
'', tooltip: 'start of range.') #}
```

As you can see from the example, an option has a name and the three compulsory fields `type`, `default`, `tooltip`. `type` must be the python name of a type, valid at runtime, and initializable from the default value, and from the text inserted by the user at runtime. Built-in examples would be `str`, `int`, `float`, `bool`. If the user provided value is invalid for the type, or if the user provides no value, the `default` value will be used. `tooltip` is shown when the user places the mouse cursor over the text, without clicking.

Options are available to the template, collected in the dictionary `options`.

`ghini.desktop` provides several Postscript and SVG production functions, in the two namespaces `PS` and `SVG`, respectively.

The SVG namespace defines the `add_text`, `add_QR` and `add_code39` functions. The PS namespace defines the `add_text`, `add_QR`, and `add_image` functions. Please refer to sources and examples for their usage. There is nothing preventing us from defining an `add_code39` function also in the PS namespace, while defining the `add_image` for the SVG namespace is more of a challenge, because of the raster/vectorial incompatibility.

Working with Jinja2

Jinja2 is a mainstream, powerful and well documented template language. Please refer to Jinja2 online documentation for information regarding how to write templates.

A good and comprehensive example for Jinja2 within `ghini.desktop` is the `tortuosa.ps.jj2` template. It shows how to write a template that inherits from a base template, how to define a template domain, how to include pictures, how to use the `PS` and `SVG` namespaces and the `enumerate` function, included by default to the environment accessible from your Jinja2 templates.

The `tortuosa.ps.jj2` is not precisely a beginner's case. Please check Mako examples for simpler templates.

Working with Mako

The Mako report formatter uses the Mako template language for generating reports. The Mako templating system is included in all `ghini.desktop` installation.

Mako is less mainstream than Jinja2, and arguably less good documented, but also quite more flexible than Jinja2. They are very similar to each other so most concepts apply to both. If you're doing something rather simple, start with Jinja2. If you stumble against Jinja2 limitations, try Mako. If you don't understand how Mako works, spend a couple of hours on Jinja2 documentation, then go back to Mako.

More information about Mako and its language can be found at makotemplates.org.

An bare bones sample Mako template is the further uninteresting `example.csv.mako` template. A more useful example is the `bgci-upload.csv.mako` template. Just as simple, but it accomplishes a useful goal.

A very comprehensive example for Mako within `ghini.desktop` is the `accession-label-qr.ps.mako` example. It shows how to write a template that inherits from a base templates, how to define a template domain. The `accession-label-qr.ps.mako` example also shows how to import pictures, how to import functions from the available Python environment, and how to use the `PS` and `SVG` namespaces, included by default to the environment accessible from your Mako templates.

Working with XSL Stylesheets

Not up-to-date, help required.

> The current maintainer was never hired to work at this part of the software, and he leaves this feature as he found it, both code and user documentation, trying not to break it. Please contribute, with text, screenshots, examples, if you know how to produce reports with XSL. The below notes date back to the old Bauble 1.0.11 docs. And, oh well, you may state that they say close to nothing.

The XSL report formatter requires an XSL to PDF renderer to convert the data to a PDF file. Apache FOP is a free and open-source XSL->PDF renderer and is recommended.

Installing Apache FOP on GNULinux

If using Linux, Apache FOP should be installable using your package manager. On Debian/Ubuntu it is installable as `fop` in Synaptic or using the following command:

```
apt-get install fop
```

Installing Apache FOP on Windows

You have two options for installing FOP on Windows. The easiest way is to download the prebuilt ApacheFOP-0.95-1-setup.exe installer.

Alternatively you can download the archive. After extracting the archive you must add the directory you extracted the archive to to your PATH environment variable.

3.7 Importing and Exporting Data

Ghini offers several formats for exchanging data with other systems, while additional formats may be provided in the form of plug-ins.

For two formats (CSV and JSON) Ghini offers both import and export. CSV aims at saving and restoring whole tables. JSON is mostly meant for selections, for non-destructive updating existing data, for combining selections from different databases.

A special case of import-export is handling `ghini.pocket` snapshots, and accepting collected updates from `ghini.pocket` instances.

There is some support for exporting in Access for Biological Collections Data (ABCD) format, while export in BGCI format is provided by the Report plug-in in the form of a Mako Template.

There is also limited support for exporting to an XML format that more or less reflects exactly the tables and row of the database.

Exporting ABCD and XML will not be covered here.

3.7.1 Backup as Flat-files / CSV

This section is about exporting the complete database as CSV files, one file per table. These files can be later used to restore your database, to initialize an other Ghini instance, as a base for migrating your data to a newer Ghini version.

Please note that we also have a QuickCSV reporter, able to produce a single CSV file, where the user decides, on-the-fly, which fields have to be included in the report. Here we talk about creating and (partially) restoring databases.

Creating a backup

To export the Ghini data to CSV select *Tools→Backup→Create* from the menu.

This tool will ask you to select a directory to export the CSV data. All of the tables in Ghini will be exported to files in the format `table_name.txt` if `TableName` is the name of the table where the data was exported from.

Restoring a backup

In general it is best to only import CSV files into Ghini that were previously exported from Ghini. It is possible to import any CSV file but that is more advanced that this doc will cover.

To import CSV files into Ghini select *Tools→Backup→Restore* from the menu.

After clicking `OK` on the dialog that ask if you are sure you know what you're doing a file chooser will open. In the file chooser select the files you want to import.

Restoring a backup deletes all previous content from restored tables.

3.7.2 JSON - JavaScript Object Notation

This feature has being developed when a user asked for it. Their request was quite simple, but we preferred anyway to lay down the bases for a more generic solution. At the moment, everything we ever needed is in place. Even so, this feature is still quite far from complete, so if it does not satisfy your needs, please contact the maintainers.

Exporting to JSON
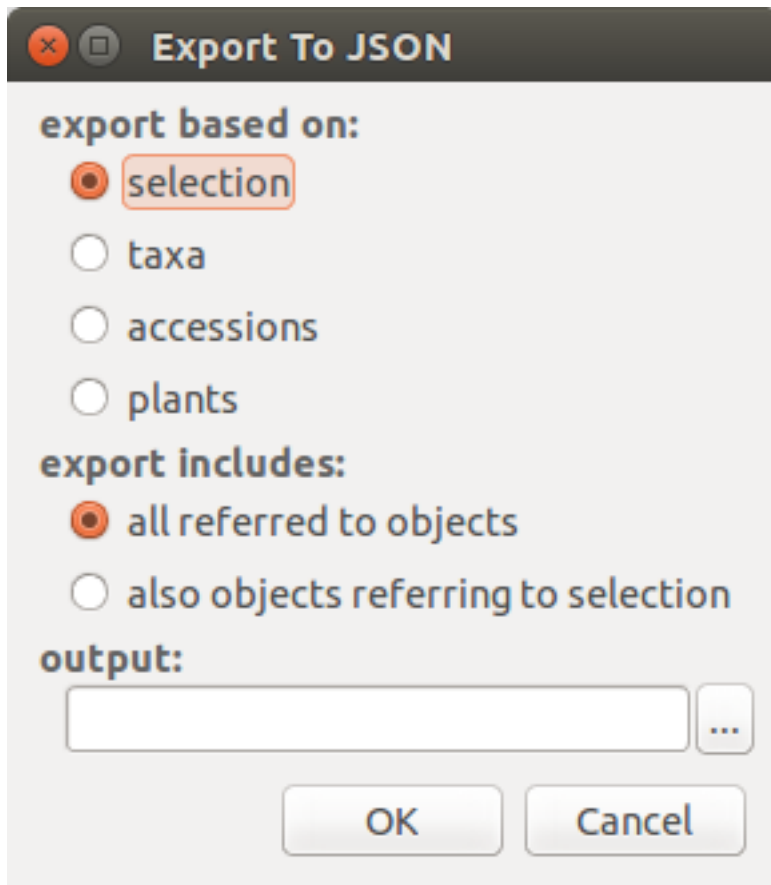
when you activate this export tool, you are given the choice to specify what to export. You can use the current selection to limit the span of the export, or you can start at the complete content of a domain, to be chosen among Species, Accession, Plant.

Exporting *Species* will only export the complete taxonomic information in your database. *Accession* will export all your accessions plus all the taxonomic information it refers to: unreferred to taxa will not be exported. *Plant* will export all living plants (some accession might not be included), all referred to locations and taxa.

Importing from JSON

This is *the* way to import data into an existing database, without destroying previous content. A typical example of this functionality would be importing your digital collection into a fresh, just initialized Ghini database. Converting a database into bauble json interchange format is beyond the scope of this manual, please contact one of the authors if you need any further help.

Using the Ghini json interchange format, you can import data which you have exported from a different Ghini installation.

3.7.3 Importing a Pictures Collection

We can consider a collection of plant pictures as a particular form of botanical database, in which each picture is clearly associated with one specific plant.
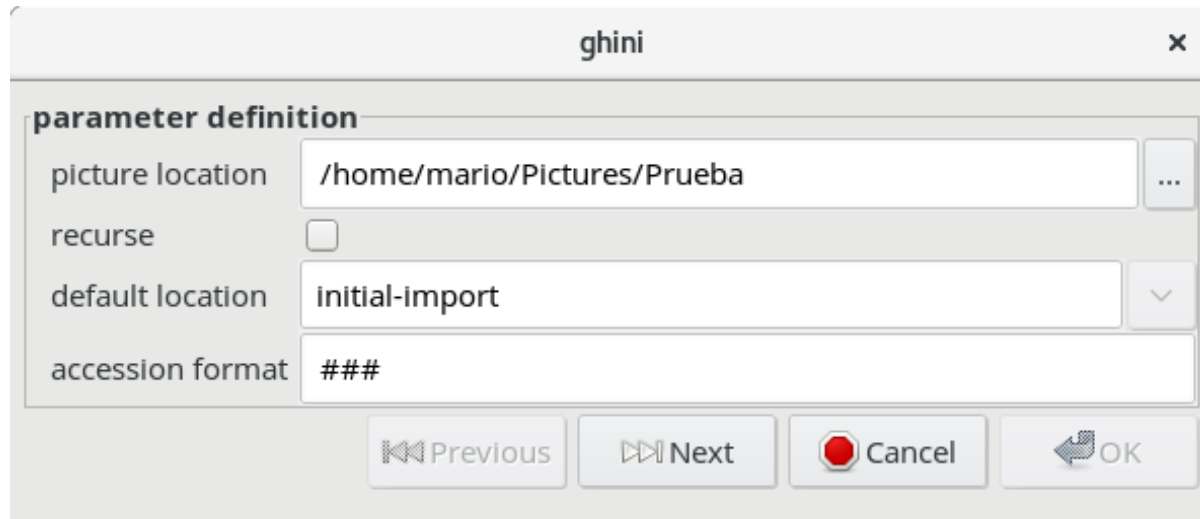
Even without using a photo collection software, you can associate pictures to accessions by following one and the same clear rule when naming picture files.

For example, `2018.0020.1 (4) Epidendrum.jpg` would be the name of the fourth picture for plant number 1 within accession 2018.0020, identified to rank genus as an Epidendrum.

The *Tools→Import→Pictures* functionality here described is meant for importing an ordered collection of plant pictures either to initialize a ghini database, or for periodically adding to it.

Use *Tools→Import→Pictures* to activate this import tool. Import goes in several steps: parameter definition; data revision and confirmation; the import step proper; finally review the import log. At the first two steps you can confirm the data and go to the next step by clicking on the **|next|** button, or you can go back to the previous step by clicking on the **|prev|** button. Once the import is done and you're reviewing the log, you can only either confirm —or abort— the whole transaction.

In the "parameter definition" pane you: select the directory from which you intend to import pictures; indicate whether to import pictures recursively; select or create a location which will be used as default location for new plants; inform the tool about the rule you've been following when naming picture files.
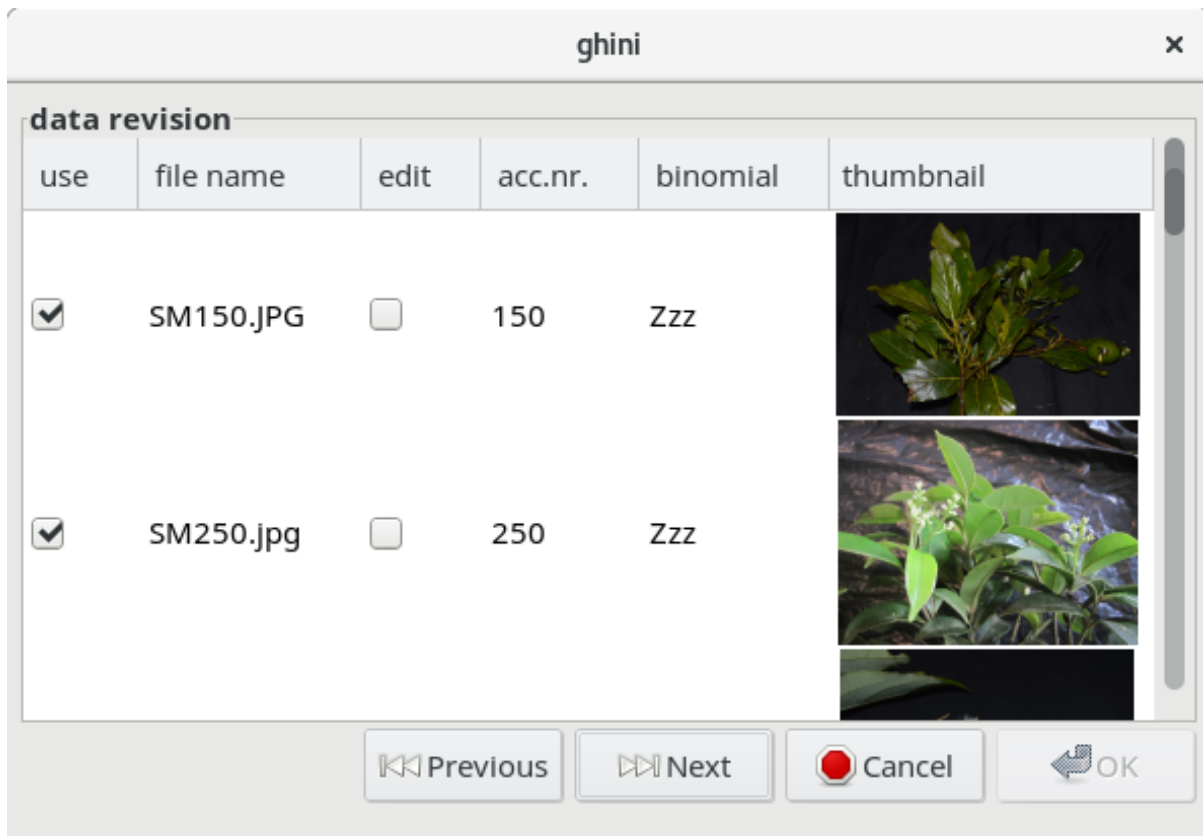

In the "data revision" pane you are shown a table with as many rows as the pictures you are importing. Each row holds as much information as the tool managed to extract from the picture name. You can review the information, correct or confirm, and indicate whether or not the row should be imported.

In the final "commit or rollback" pane you read the logs relative to your data import, and decide whether to keep them (commit them to the database), or undo them (rollback the transaction).
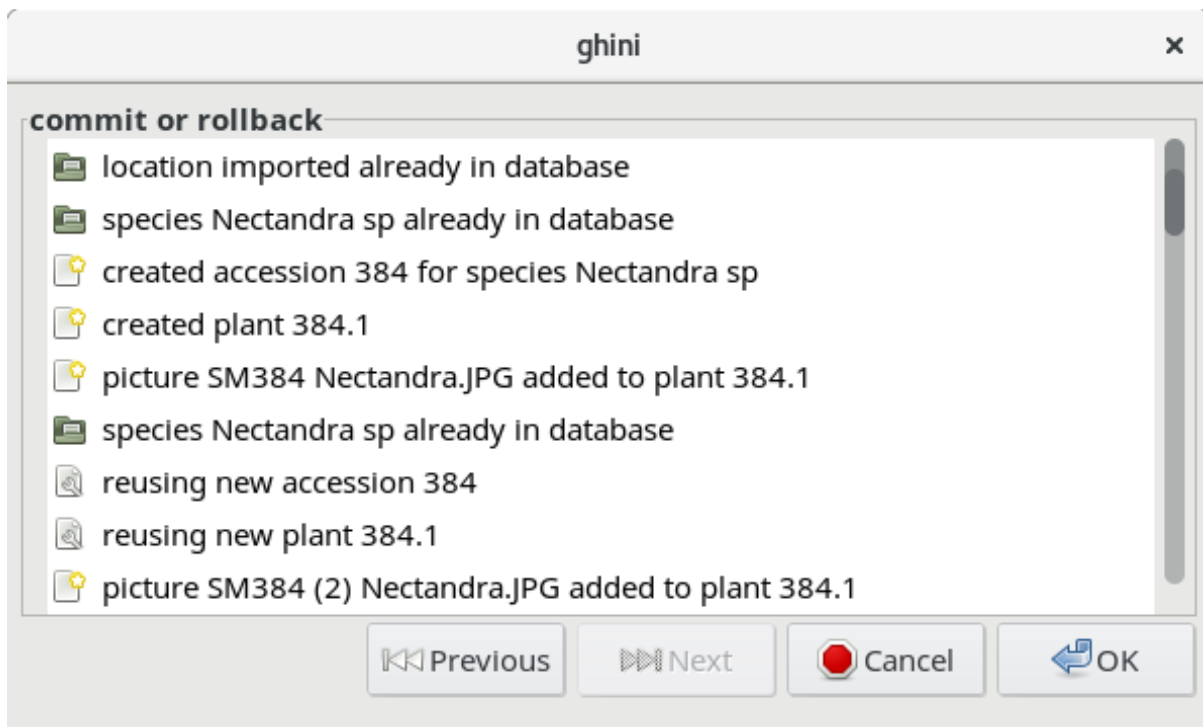
When the Picture Collection importer creates or updates objects, it also sets a Note that you can use for selecting the objects involved in the import, and for reviewing if needed.

3.7.4 Importing from a Generic Database

This functionality is the object of issue #127, for which we have no generic solution yet.

If you're interested in importing data from some flat file (e.g.: Excel spreadsheet) or from any database, contact the developers.

3.8 Managing Users

Note: The Ghini users plugin is only available on PostgreSQL based databases.

The Ghini Users Plugin will allow you to create and manage the permissions of users for your Ghini database.

You must log in to your database as a user with `CREATEROLE` privilege in order to manage other users.

3.8.1 Creating Users

To create a new user...

3.8.2 Permissions

Ghini allows read, write and execute permissions.

Cookbook

4.1 Contributed recipes collection

This page presents lists of use cases. If you're looking for straight, practical information, you are at the right place. If you prefer a thorough presentation of the software and database structure, check the section software for botanical gardens

All material here has been contributed by gardens using the software and sharing their experiences back to the user community.

The authors of the software wish to thank all dearly.

4.1.1 Quito Botanical Garden

At the JBQ, Quito Botanical Garden, we have adopted the Ghini software in April 2015. Since that time, we have accumulated experience with the program, and we are ourselves in need to document it, in order to secure the knowledge to the institution. We are happy to share it.

Technical

- We work on GNU/Linux, a platform that many users don't master, and our database is inside of a remote database management system. This implies steps that are not obvious to the casual end user.

How to start a program

To start a program given its name, hit the key next to Alt, or click on
, then start typing the name of the program, in our case "Ghini" or just
click on the program symbol , appearing near the left margin of your
display.

Database server

We chose for a centralised PostgreSQL database server. This way we are
protected from concurrent conflicting changes, and all changes are simultane-
ously available on all ghini clients. We did need to outsource database server
management.

adding a new user

Ghini keeps track of the user performing all sort of edits to the database, and
at the garden, apart from the stable users, we have all sorts of temporary users
writing to the database, that we decided we would let Ghini help us keep track
of database events.

Since we work using PostgreSQL, the users that Ghini stores in the database
history are the database users, not the system users.

Each user knows their own password, and only knows that one. Our super-
user, responsible for the database content, also has the `bauble` fictional user
password, which we only only use to create other users.

We do not use account names like `voluntario`, because such accounts do
not help us associate the name to the person.

— adding a new system user (linux/osx)

Adding a system user is not strictly necessary, as ghini does not use it in
the logs, however, adding a system user allows for separation of preferences,
configured connections, search history. On some of our systems we have a
single shared account with several configured connections, on other systems
we have one account per user.

On systems with one account per user, our users have a single configured
connection, and we hold the database password in the `/home/<account>/`
`.pgpass` file. This file is only readable for the `<account>` owner.

On systems with a shared account, the user must select their own connection
and type the corresponding password.

These are the steps to add system users:

```
sudo -k; sudo adduser test
sudo adduser test adm; sudo adduser test sudo
sudo adduser test sambashare; sudo adduser test ghini
```

— adding a new database user

Ghini has a very minimal interface to user management, it only works with postgresql and it very much lacks maintainance. We have opened issues that will allow us use it, for the time being we use the `create-role.sh` script:

```
#!/bin/bash
USER=$1
PASSWD=$2
shift 2
cat <<EOF | psql bauble -U bauble $@
create role $USER with login password '$PASSWD';
alter role $USER with login password '$PASSWD';
grant all privileges on all tables in schema public to
 →$USER;
grant all privileges on all sequences in schema public
 →to $USER;
grant all privileges on all functions in schema public
 →to $USER;
EOF
```

The redundant `alter role` following the `create role` lets us apply the same script also for correcting existing accounts.

Our ghini database is called `bauble`, and `bauble` is also the name of our database super user, the only user with `CREATEROLE` privilege.

For example, the following invocation would create the user `willem` with password `orange`, on the `bauble` database hosted at 192.168.5.6:

```
./create-role.sh willem orange -h 192.168.5.6
```

• Understanding when to update

Updating the system

Ubuntu updates are a lot lighter and easier than with Windows. So whenever the system suggests an update, we let it do that. Generally, there's no need to wait during the update nor to reboot after it's done.

Updating ghini

The first window presented by Ghini looks like this. Normally, you don't need do anything in this window, just press enter and get into the main program screen.

Occasionally, at the top of the screen an information text will appear, telling you that a newer version is available on-line.

The update procedure is simple, and it depends on the operating system you use, we're not explaining here again.

It is generally a good idea updating the software. If in doubt, contact the author, or write to the group.

• understanding ghini initial screen

Complete screen

At the moment of writing, our initial screen looked like this:

Apart from the main application menu, Ghini offers three special interface sections with information and tools to explore the database.

Numeric overview

The table in the right half of the screen presents a summary of all the registered plants can be observed. Each entry printed in bold is a link to the query selecting the corresponding objects.

Stored queries

The lower half of the right hand side contains a set of stored queries. While you can edit them to your liking, our hints include selecting those accessions that have not been identified at rank species. And one for the database history.

Query and action buttons

At the top of this screen you can find the field in which you would enter your searches.

– With the button, in the form of a house, you can return from your searches to the main screen.

– With the button, in the form of an arrow, you can return to your last search.

– With the button, in the form of a gear, you can start the "Query Builder", which helps you compose complex searches in a simple, graphical way.

• We often have volunteers who only work at the garden for a very short time. It was with them in mind that we have developed a hyper-simplified view on the ghini database structure.

Details

The two figures here show all that our temporary collaborators need to know.

• At times, the program gives error messages. **DON'T PANIC** , retry, or report to the developers.

Network problems

In order to work, the program needs a stable network connection to the database server. It can happen: you start the program, and it can't connect to our database server. You would then get a rather explicit but very badly typeset error message.
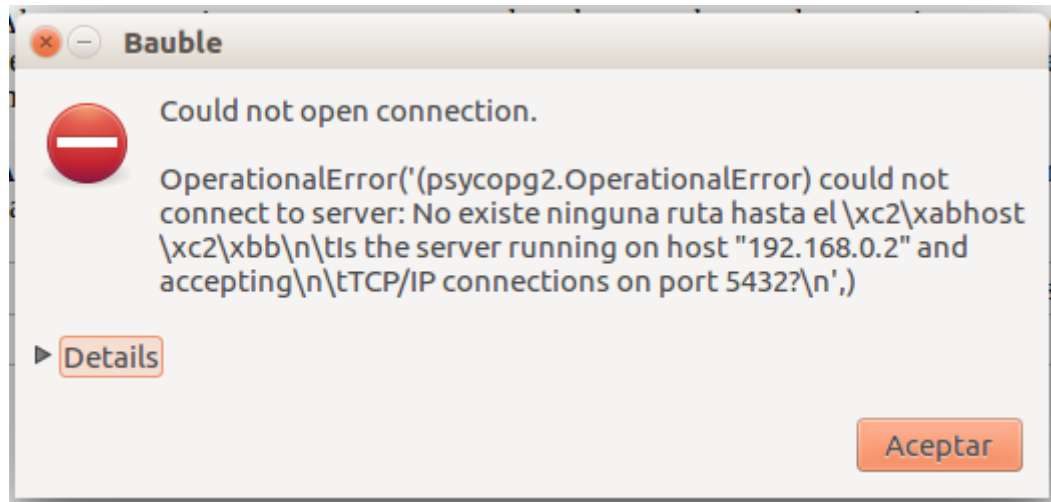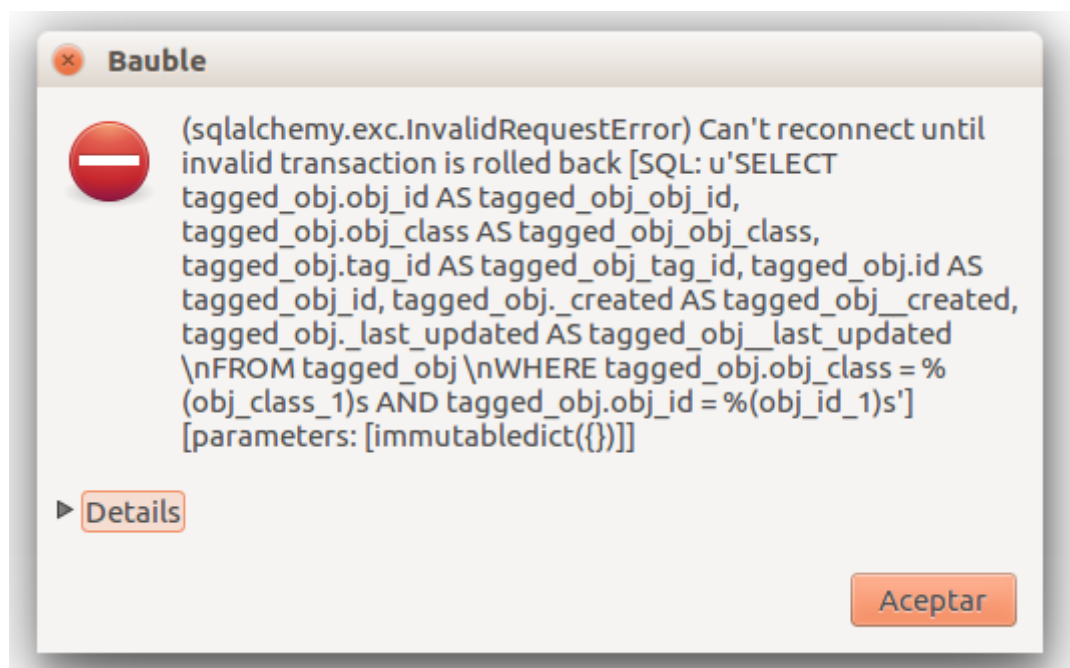

Just ignore it and try again.

Search fails with error

Sometimes and without any apparent cause, a search will not run successfully, and a window with an error message will be displayed. In this case you only have to try to perform the same search again.

An example of such an error message:

Search does not return something I just inserted

Accession codes starting with zero and composed of just numbers, as for example `016489` are considered by the software as numbers, so if you don't enclose the search string in quotes, any leading 0 will be stripped and the value will not be found.

Try again, but enclose your search string in single or double quotes.

Number on the label	corresponding search
16489	'016489'

Please note: when you look for a Plant code, not an Accession, the leading zero becomes optional, so in the above example it's maybe easier to type `16489.1`.

- A serious situation happened once, and we absolutely want to prevent it from happening again: a user deleted a genus, with everything that was below it, species and accessions, and synonyms.

Solving it with user permissions

We propose to have different connection profiles, associated to different database users, each user with all needed permissions.

Full permission (BD-JBQ) Only qualified personnel get this kind of access.

Insert and update (BD-JBQ-limitado) We use this one for those users who come help us for a limited time, and who did not get a complete introduction to database concepts. It is meant to prevent costly mistakes.

Read only (BD-JBQ-lectura) it can be shared with anyone visiting the garden

You select the connection at start-up, and the software asks you for the password corresponding to the connection you selected.
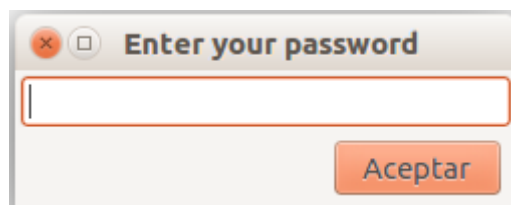
If you want to review the details of the connection, click on the next to 'Connection Details', it will change to , and the connection window will be displayed as one of the following:
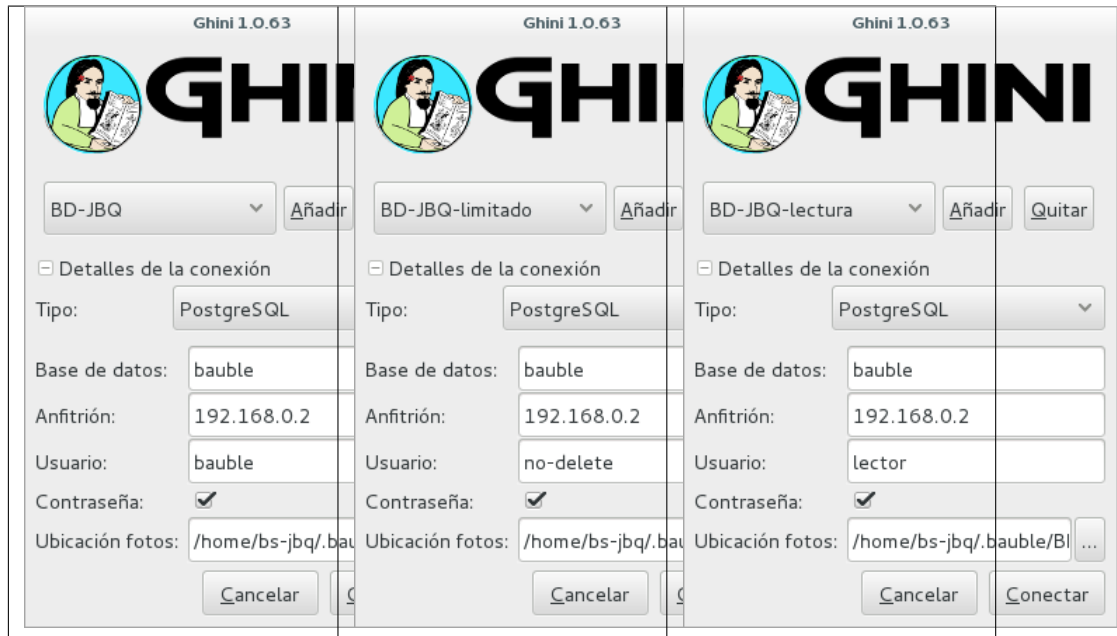
As you can see, we are connecting to the same database server, each connection uses the same database on the server, but with different user.

Thinking further about it

On the other hand, we are questioning if it is at all appropriate, letting any user delete something at such high level as a family, or a genus, or, for that matters, of anything connected to accessions in the collection.

The ghini way to question the software features, is by opening a corresponding issue.

• When contacting the developers, they will definitely ask for technical information, or at least to see a screen-shot. Help them help you.

Taking a screen-shot

On Linux there are three ways to create a screen-shot, all involve hitting the 'PrtSc' key. The most practical one is possibly hitting the 'PrtSc' key in combination with Ctrl and Shift. This will start an interactive screen copy tool. You select a rectangle and the area is copied in the clipboard. Paste it in the email you're writing, or in the chat line where the developers are trying to help you.

Where are the logs

Ghini continuously saves a very informative log file, in the `~/.bauble/ bauble.log` file. Don't bother opening it, just send it over. It contains

loads of technical information.

Continuous unmanned alerting

An other option is to activate the sentry handler. It will notify our sentry server of any serious situations in the software. If you registered, the developers will know how to contact you if necessary.

To the healthy paranoid: we're not monitoring what you're doing, we're monitoring how our software works. You can always opt out.

You activate the Sentry handler in the `:prefs` page: look for the row with name `bauble.use_sentry_handler`, if the value is not what you wish, double click on the line and it will change to the other value.

Taxonomy

- Introduction

Orchidaceae taxonomic complexity

At the JBQ, we work most of all with orchids, family Orchidaceae, one of the largest plant families, with no less than 850 genera, organized —according to Dressler— in approximately 70 subtribes, 22 tribes, 5 subfamilies. How we represent this information is not obvious and needs be explained.

The taxonomy of the Orchidaceae family is continuously being reviewed. Genera get added, refused, reorganized, recognized as synonyms, some taxonomists prefer grouping species or genera in a new way, others split them again and differently, botanists of different nationalities may have different views on the matter. All this sounds very complex and specialistic, but it's part of our daily routine, and it can all be stored in our Ghini database.

- Identifying at rank Genus, or Family

At rank genus

Ghini-3.1 prescribes that an accession is identified at rank species, in all cases. The current maintainer acknowledges that this is a mistake, coming from the early Bauble days, and which Ghini-3.1 has in common with other botanic software. Until this is fixed, we rely on established practices.

If an accession is identified at rank genus, we add a fictive species in that genus, we don't specify its species epithet (we don't know that) and we add an unranked epithet in the infraspecific information section, like this:

When displayed in a search result, it shows like this:

Editor de especies de plantas

| Nombre de la especie | Información adicional | Notas | Fotos |

Vanda sp

Género * | Vanda

Autor/a

Híbrido ☐

Cultivar Grupo

Especie ⬤

Cualificación

Información infraespecífica

Rango	Epíteto	Autor/a
	sp	

➕

| Cancelar | Aceptar | Añadir accesiones | ⇨ Siguiente |

At rank family

If an accession is only identified at rank family, we need a fictive genus, to which we can add the fictive species. Since our garden is primarily focusing on Orchidaceae, we use the very short name **Zzz** for the fictive genus within the family, like this:

The current maintainer suggests to use the prefix **Zzz-** and behind the prefix to write the family name, possibly removing the trailing **e**. Removal of the trailing **e** is useful in order not to get results that include genus names when you as for stuff ending in **aceae**.

Apart from the aforementioned **Zzz** genus in the Orchidaceae family, we follow this suggested practice, so for example our collection would include *Zzz-cactacea* or *Zzz-bromeliacea*.

Remember: our **Zzz** genus is a fictive genus in the **Orchidaceae** family, do not use it as unspecified genus in other families.

- Identifying at a rank that is not allowed by the software (eg: Subtribe, or Subfamily)

At rank subtribe

We sometimes can't identify a taxon at rank genus, but we do manage to be more precise than just "it's an orchid". Quite often we are able to indicate the

subtribe, this is useful when you want to produce hybrids.

The software does not let us store ranks which are intermediate between family and genus, so we need to invent something, and this is what we do:

We insert a fictive genus, naming it as the subtribe, prefixing it with 'Zzx-', like in this example:

This Zzx-Laeliinae is some genus in the Laeliinae subtribe.

In order to be able to select genera by subtribe, we also add a note to the Zzx-Laeliinae fictive genus as well as for all real genera in that subtribe, note category subtribus, note value the subtribe name.

This allows for queries like:

```
genus where notes.note=Laeliinae
```

We are very much looking forward to seeing that issue-9 solved!

At rank subfamily, tribe

Just as we reserved the prefix Zzx- for subtribe, we reserve the prefixes Zzy- for tribe, Zzw- for subfamily.
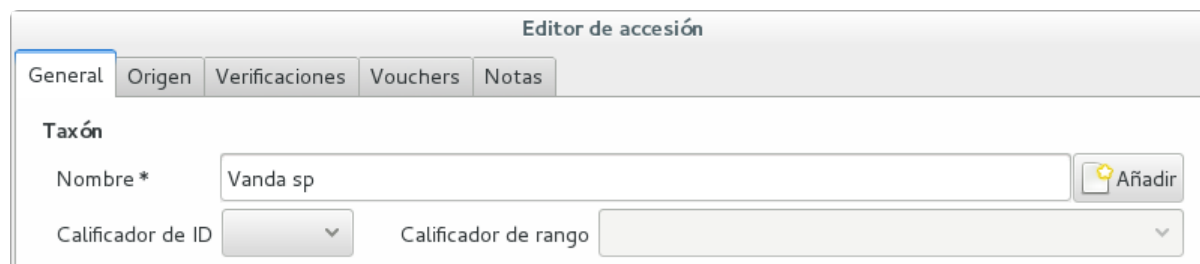
In particular, the subfamily information is relevant, because there are subfamilies within the Orchidaceae family which are not further separated.

- Editing the Accession identification - the Species details

Placeholder species for individual accessions

Scenario one describes the identification of a single accession, which had been associated to a "generic", placeholder species, something like "*Zzz* sp" or "*Vanda* sp";

In this case, when the plant species becomes known, we change the association in the accession, selecting a different species.

We do not edit the species, because there might be totally unrelated accessions connected to the same placeholder species.

Unknown species for multiple accessions

A different case is when we have a whole batch of accessions, all obviously the same species, but we haven't been able to identify it. In this case, we associate the accessions with an incompletely specified species, something like "*Zzz* sp-59", preferably adding the taxonomist's name, who made the association.

A species like "*Vanda* sp-018599" is not a placeholder species, it is a very concrete species, which we haven't yet identified.

In this case, when the species gets identified (and it could even be a species nova), we directly edit the species, so all accessions that refer to it get the change.

- A new plants is relative to a species not yet in our collection.

Last minute species

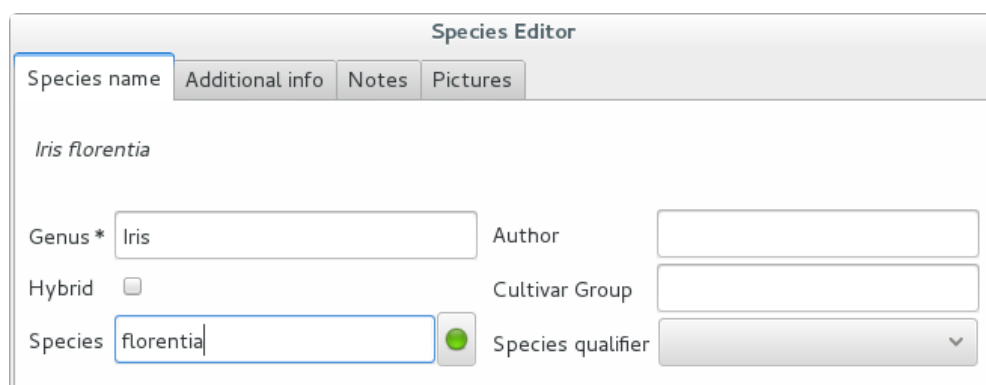We start this from the Accession window and it's very simple, just click on the **+** next to the species name, we get into the Species window.

- Adding a species and using online taxonomic services

Adding a new species — the plant list.

We start the obvious way: type the genus epithet, possibly select it from the completion list, then type the species epithet, or at least your best guess.
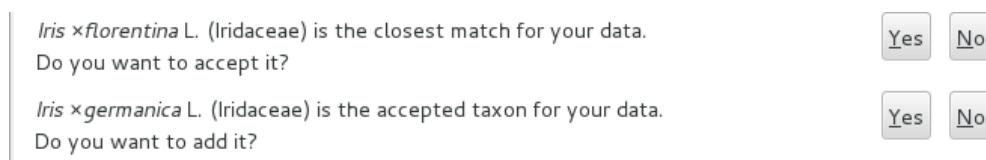
Next to the species epithet field there's a small button, , which connects us to the plant list. Click on it, a message area appears at the top of the window.

Depending on the speed of your internet connection, but also on how close your best guess is to a correct published name, the top area will change to something like this:
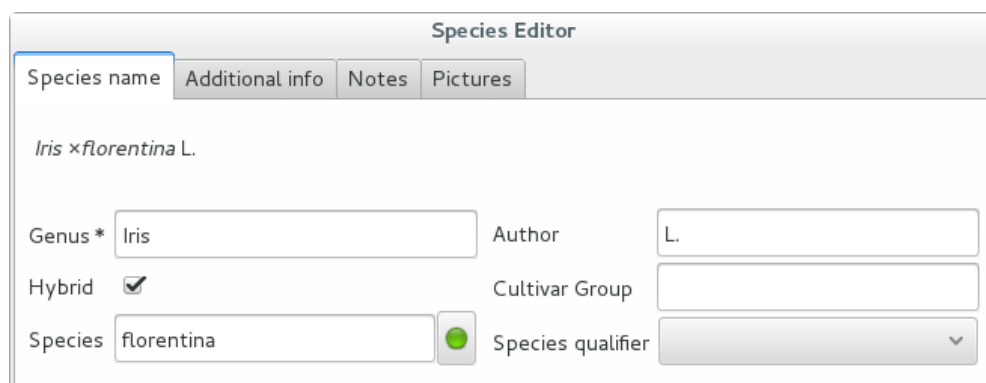
Accept the hint and it will be as if you had typed the data yourself.

Reviewing a whole selection — TNRS.

This is described in the manual, it's extremely useful, don't forget about it.

Let the database fit the garden

- A never-ending task is reviewing what we have in the garden and have it match what we have in the database.

Initial status and variable resources

When we adopted ghini, we imported into it all that was properly described in a filemaker database. That database focused solely on Orchids and even so it was far from complete. In practice, we still meet labeled plants in the garden which have never been inserted in the database.

From time to time, we manage to get resources to review the garden, comparing it to the collection in the database, and the main activity is to insert accession codes to the database, take pictures of the plant in question, and note its location, all tasks that are described in the remainder of this section.

The small Android app ghini.pocket was added to the Ghini family while a Ghini programmer was here in Quito. It helps us take a snapshot of the database in our pocket while walking in the garden, but it also allows for a very swift inventory procedure.

Inventory procedure

We start ghini.pocket, we write down the name of the location where we will be conducting the inventory, for example (INV 1) for greenhouse 1. We enter (type or scan if the plant has bar code or QR code) the accession code and we look it up in ghini.pocket.

A side effect of performing the search is that ghini.pocket writes the date with time, location and the code looked for in a text file that can later be imported into the database.

For a greenhouse with around 1000 plants our estimates suggest you will need two days, working at relaxed pace, from 8:00 am to 5:00 pm.

After having imported the file generated by ghini.pocket, it is easy to reveal which plants are missing. For example: If we did the inventory of the INV3 from 4 to 5 September, this is the corresponding search:

```
plant where location.code = 'INV3' and not notes.note
 ↪like '2017090%'
```

All of these plants can be marked as dead, or lost, according to garden policy.

Visualizing the need of taxonomic attention

Our protocol includes one more detail intended to visually highlight plants that need the attention of a taxonomist.

> A plant that only appears in our data base identified at family level or that
> wasn't yet the database receives a visual signal (e.g.: a wooden or plastic stick,
> for ice cream or french fries), to highlight that it is not identified. In this way
> the taxonomist in charge, when making a tour of the greenhouse can quickly
> spot them and possibly proceed to add their identification in the database.

- Naming convention in garden locations

Details

code	description
CAC-B *x*	Reserved to cactus plants next to the orchids exposition glasshouses.
CRV:	Nepenthaceae exibition
IC-*xx*:	orquidearios de calor en el jardín (1A a 9C son lugares específicos entre del orquideario)
IF-xx:	orquidearios de frío en el jardín (1A a 5I son lugares específicos dentro del orquideario)
INV1:	invernadero 1 (calor)
INV2:	invernadero 2 (frío)
INV3:	invernadero 3 (calor)

- Adding an Accession for a Plant

Obviously we keep increasing our collection, with plants coming from commercial
sources, or collected from the wild, more rarely coming from expeditions to remote areas
of our country, or we receive plants which were illegally collected.

Sometimes we have to add plants to the digital collection, just because we have them
physically, found in the garden, with or without its label, but without their digital coun-
terpart.

Existing plant, found in the garden with its own label

> This activity starts with a plant, which was found at a specific garden location,
> an accession label, and the knowledge that the accession code is not in the
> database.
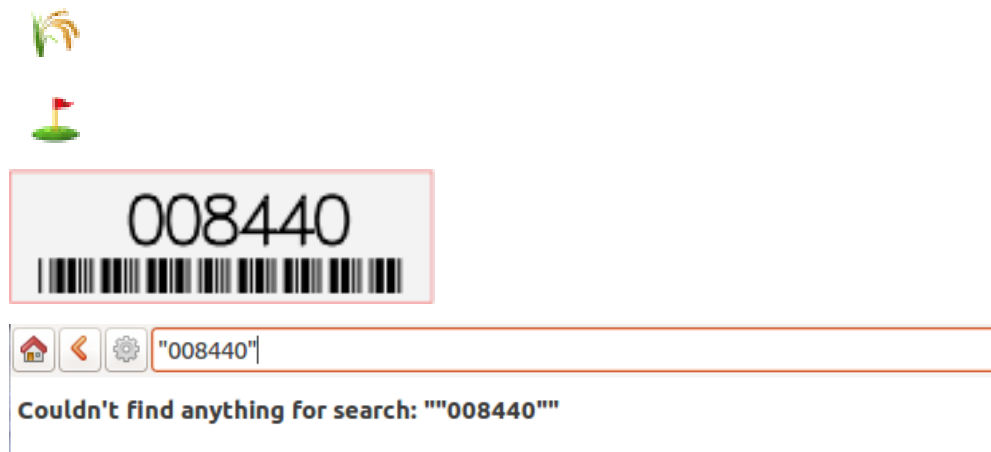
For this example, let's assume we are going to insert this information in the database.

Accession	Species	Location
008440	*Dendrobium* ×'Emma White'	Invernadero 1 (calor)

We go straight into the Accession Editor, start typing the species name in the corresponding field. Luckily, the species was already in the database, otherwise we would use the **Add** button next to the entry field.
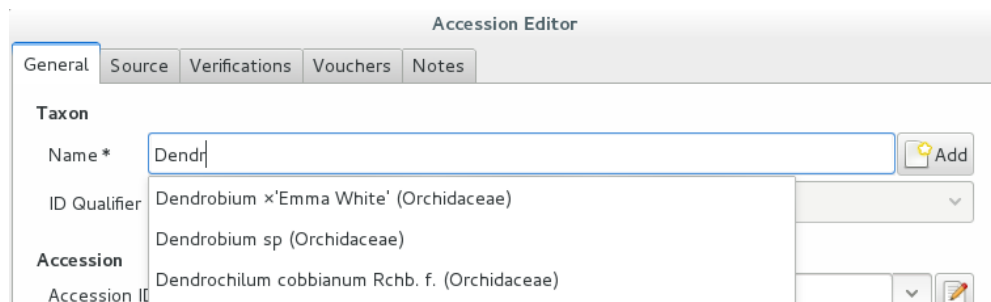
We select the correct species, and we fill in a couple more fields, leaving the rest to the default values:

Accession ID	Type of Material	Quantity	Provenance
008440	Plant	1	Unknown

After this, we continue to the Plant editor, by clicking on **Add Plants**.

We do not fill in the Accession's "**Intended Locations**", because we don't know what was the original intention when the plant was first acquired.

In the Plant Editor, we insert the Quantity and the Location. And we're done.

The plant is now part of the database:

New accession: plant just entering the garden

This activity starts with a new Plant, just acquired from a known Source, a plant label, and an intended Location in the garden.

We mostly do the same as for the case that a plant is found in the garden, there are two differences: (1) we know the source of the plant; (2) acquiring this plant was a planned action, and we intend to place it at a specific location in the garden.

Again, we go straight into the Accession Editor, start typing the species and we either select if from the completion list or we add it on the fly.

Accession ID	Type of Material	Quantity	Source
033724	Plant	1	specified

After this, we continue to the Plant editor, by clicking on **Add Plants**.

In the Plant Editor, we insert the Quantity and the Location.

Please note that the plant may be initially placed in a greenhouse, before it reaches its intended location in the garden.

Existing plant, found in the garden without its label

When this happens, we can't be sure the plant had never been in the collection, so we act as if we were re-labeling the plant. This is discussed in the next section, but we fall back to the case of a new accession.

• When we physically associate a label to a plant, there's always the chance that something happens either to the plant (it may die) or to the label (it may become unreadable), or to the association (they may be separated). We have software-aided protocols for these events.

We find a dead plant

Whenever a plant is found dead, we collect its label and put it in a box next to the main data insertion terminal, the box is marked "dead plants".

Definitely at least once a week, the box is emptied and the database is updated with this information.

Dead plants aren't *removed* from the database, they stay there but get a **quantity** zero. If the cause of death is known, this is also written in the database.

Please once again remember that a **Plant** is not an **Accession** and please remember we do not remove objects from the database, we just add to their history.

Insert the complete plant code (something like `012345.1`, or `2017.0001.3`, and you don't need leading zeros nor quotes), right click on the corresponding row, and click on **edit**. change the quantity to 0, fill in the reason and preferably also the date of change.

If you need add any details about the plant death, please use a **note**, and re-use the note category "death_cause".

Plants with **quantity** zero are shown with a different colour in the results view. This helps distinguish them from live plants.

We find a plant without a label

We can't be sure the plant had ever been in the collection or not. We assume it had, and that its label was lost.

Losing a plant label is unfortunate, but it just sometimes happens. What we do is to put a new label to the plant, and to clearly state that the label is a replacement of an original one.

We then handle the case as if it was a new accession, plus we add a note to the accession, category "label", text "relabeled".

• Keeping track of different sources of plant material

What different sources we can have

In this botanical garden, we receive plants from different types of origin. It could be from expeditions (plants coming from nature, collected with legal permission from MAE - Ecuadorian Environment Ministry), donated plants mostly coming as gifts from collectors or orchid commercialization enterprises, purchased, or confiscated plants (usually coming from MAE raids around the country).

If the plant comes from a wild source

The accession editor offers the option "origin" option. When a plant is traceable to a wild source, we can specified its specific origin. We want to comply

with ITF2, and ghini-3.1 only partly respects that standard. The ITF2 complying options are:

- Wild: Accession of wild source.

- Cultivated: Propagule(s) from a wild source plant.

- Not Wild: Accession not traceable to a wild source.

- Insufficient data

In the case of a donated plant, it is better to put detail information just as a note in the plant accession; in the case of a plant with an unknown origin, we select the Insufficient data option.

Using the source tab in the accession editor

In this section we can create or use a contact, our source of plant material. It could be from an expedition to a collecting place, and in this case we would specify the region and the expedition name, or could be the name of the person or enterprise donating a specific batch of plants.

Once you choose or create the contact information, this section deploys more options, here you can specify the region, where you can choose the country of origin, and a specific location within the region, georeferencing information (including the GPS data), habitat description collector name. For the last one, I recommend also to write the specific date next to the collector name (eg. Luis Baquero 11/10/2016).

Donated, bought or confiscated plants

However useful for expeditions or for donors where the main information is geographic, this source tab is not very practical in our remaining cases: we handle three more categories: confiscated, purchased and donated, for these categories the options available in the source tab do not apply: too much information and not to the point.

In these cases, we add a set of notes, according to the case.

— Donated plants

If the plant was donated by individual, we add the individual among our contacts and specify it as source, then we add the notes:

category	text
source-type	gift
source-detail	Contribución científica al JBQ

— Bought plants

If the plant was bought, we add the previous owner among our contacts and specify it as source, then we add the notes:

category	text
source-type	purchase
source-detail	optional, free text
factura	the invoice number

— Confiscated plants

If the plant was confiscated, we add the previous owner among our contacts and specify it as source, then we add the notes:

category	text
source-type	confiscated
source-detail	possibly, legal details, law number …

- Producing or reproducing labels

Refreshing plant labels

Sometimes we refresh the labels, for example all that is in a greenhouse, or maybe just a set of plants because their labels risk becoming unreadable.
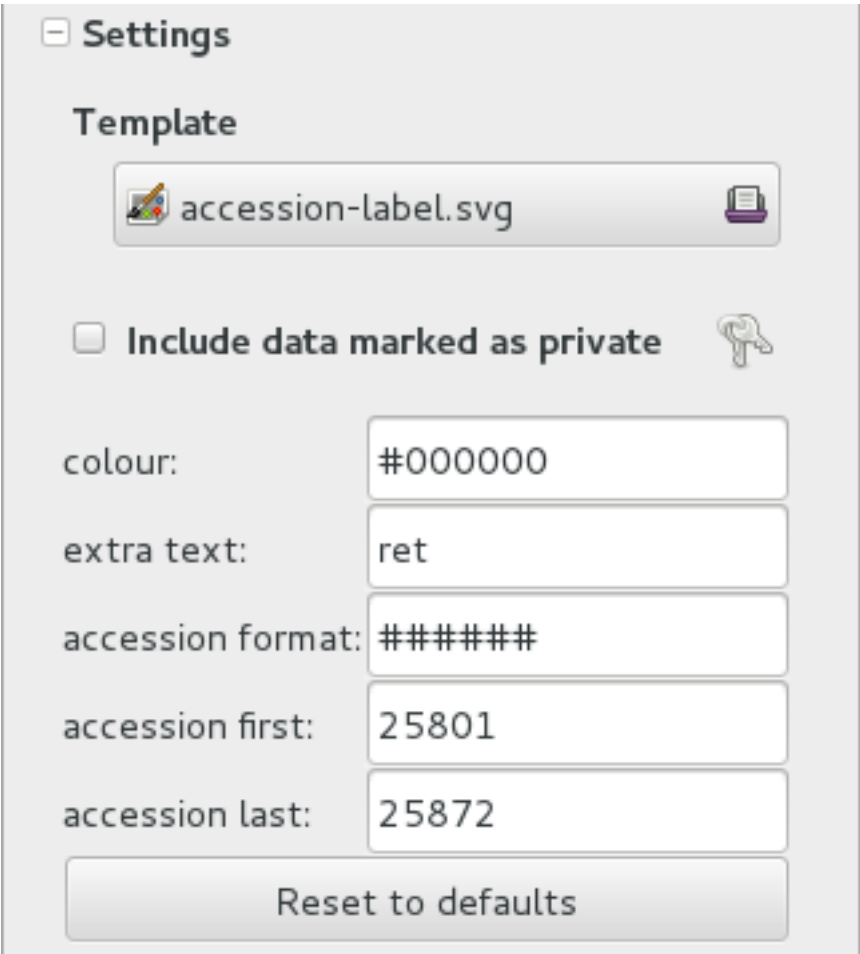
In the first case it's easy selecting all plants in the Location, we just type the location name, or give the search `location like <location name>`.

The second case it's a bit trickier. What we do is to create a temporary **Tag**, and use it to tag all plants that were found in need for a new label.

Given the selection, we start the report tool, using the mako `accession-label.svg` template. We reset its options to default values, and since we're using a simple printer, we set the colour to `black` instead of `blue`, which is meant for engraving.

Preparing labels for non-database plants

To prepare the batch of 72 labels, we use a mako report template, named `accession-label.svg`. This template accepts parameters, this is an example that would produce labels from 025801 all the way to 025872.

Labels come for us in two flavours: (1) either new plants just being acquired by the garden; (2) or plants in the garden, found without a label. We distinguish the two cases by adding a 'ret' extra text for relabeled plants.

We keep two boxes with labels of the two types, ready to be used.

- Our garden has two exposition greenhouses, and several warm and cold greenhouses where we keep the largest part of our collection. Plants are moved to the exposition when flowering and back to the "warehouse" when less interesting for the exposition. For each plant in our collection we need to know its current locations and history of movements.

Planned action

> The action starts by moving the plants around, and collecting the plant code either on paper, or in our mobile app, if we had one.
>
> We then go to the desktop terminal and revise all plants one by one changing their location in the database. It is important that the date of the location change is correctly memorized, because this tells us how long a plant stays in the exposition.
>
> If we had a mobile app, we would just upload the info to the server and we would be done.

Ex-post correction

> While revising the garden, we find a plant at a location that is not what the database says. We update the database information.
>
> For example, the plant belonging to accession "012142", species "*Acineta* sp", was found in "Invernadero 1", while the database says it is in "ICAlm3".
>
> All we do is find the Plant in the database and update its information. We do not change anything in the initial Accession information, just the current Plant information.
>
> We type the accession code in the search entry field, with quotes, hit enter. The search results now shows the accession, and it tells us how many plants belong to it. Click on the squared **+** in the results row, so we now also see a row for the plant belonging to the accession.
>
> Right click on the Plant row, the three options will show: "Edit, Split, Delete", select Edit, you land in the Plant Editor.
>
> Just correct the Location field, and click on OK.
>
> The InfoBox contains information about the last change to the object:

> **▼ Properties**
>
> **ID:** 6021
>
> **Type:** Accession
>
> **Date created:** 2014-11-18 00:11:00
>
> **Last updated:** 2017-03-17 14:03:01

> For plants, even more interesting, it builds a history of changes, list that includes Location changes, or Quantity changes.

> **⊟ Changes**
>
> 28-02-2014: 1 Added to (3b) Back of Pinus
>
> 28-02-2014: 1 Transferred from (4a) limpio to (3b) Back of Pinus
>
> 18-02-2014: 1 Added to (4a) limpio

- As plants enter the flowering stage, we can review their identification directly, or we take pictures of details of the flower, hoping that a visiting specialist could help completing the identification.

Adding pictures

> We are practicing with ODK Collect, a small program running on hand-held android devices. Ghini's use of ODK Collect hasn't yet frozen to a best practice. Do have a look at the corresponding issue on github.

- Regularly, we need producing reports about our collection that the Ecuadorian Environment Ministry (MAE) requires and that justify the very existence of the garden.

Producing reports

> Each year the botanic garden has to submit a report (annual report of management and maintenance of orchids collection) complying to the requirements of the Ecuadorian Ministry of the Environment.

> To this end, we start selecting the plants we have to include in the report. It might be all acquisition in the past year:

```
accession where _created between |datetime|2017,1,1|
↪and |datetime|2018,1,1|
```

> or all plants within a location, or all plants belonging to a species, or just everything (but this will take time):

```
plant where location = 'abc'
plant where accession.species.epithet='muricata' and␣
→accession.species.genus.epithet='Annona'
plant like %
```

Having selected the database objects which we want in the report, we start the report tool, which acts on the selection.

Searching the database

You search the database in order to edit the data further, or because you want to produce a report. Anyway you start with typing something in the search field

and you hope to see your result in the search result view.

Search in order to edit (plant or accession)

When searching in order to edit, you want to be very specific, and select as few objects as possible. The most fine-tuned search is the one based on plant number: you know the code, you get one object.

If your plant is not there, the screen would look like this:
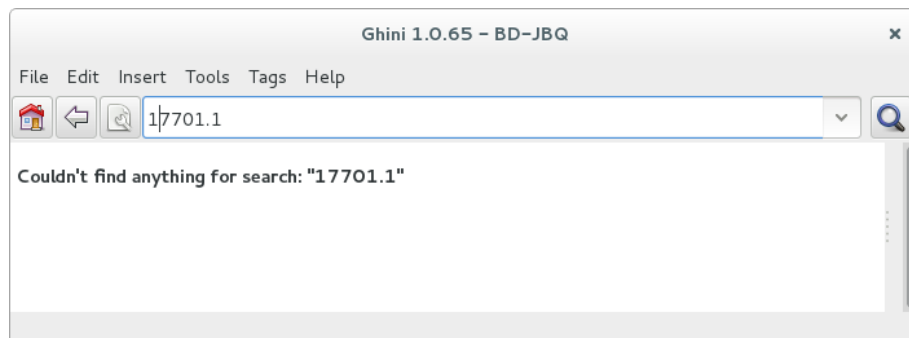
Other example, plant `007701.1` is in the database:
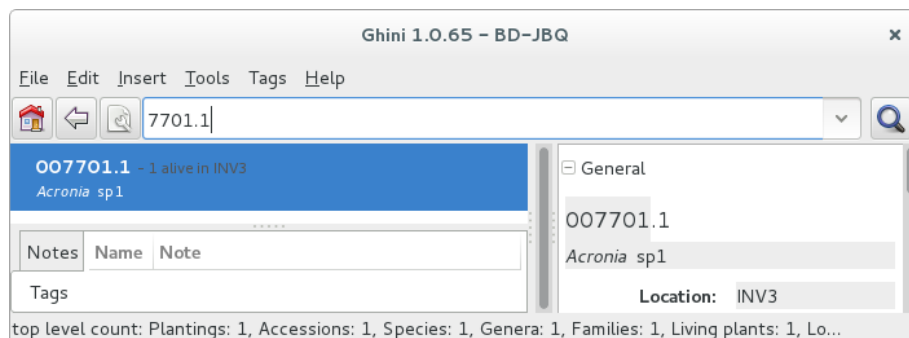
All fields with a darker background in the infobox on the right hand side are hyperlinks to other objects in the database. Clicking on them will

---

either replace the text in the search field and execute the query, or will simply add the object to the results.

Clicking on the accession does the latter.
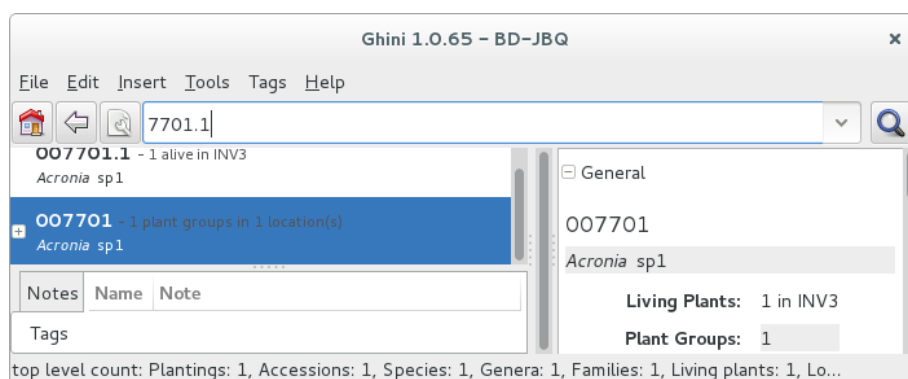
We now have both Plant or Accession in the search result view and we can now edit either or both.

Search in order to report

When searching in order to create a report, you want to be both specific (you don't want to report about irrelevant objects) and broad (you don't want to report about a single object).

Sometimes the report itself suggests the query, as for example: all plants in greenhouse 3; or: all plants belonging to endangered species (we store this information in a note associated to the species); or: all plants added to the collection this year;

```
plant where location.code = INV3
plant where accession.species.notes.note="endangered
↪"
plant where accession._created > |datetime|2017,1,1|
```

Otherwise a flexible way to achieve this is to work with **Tags**.

Using Tags as enhanced searching

Sometimes we have to take the same action on objects of the same type, but we don't manage to quickly think of a search query that would group all that we need and exclude all we do not need.

This is one possible use of **Tags**. We start with a selection, we tag all objects in the selection under a new temporary tag. Let's say we call it "temporary".

We continue searching and adding objects to the temporary tag until the tag identifies all that we need.

Finally from the Tags menu we select the one we just created (in our example this corresponds to the search `tag="temporary"`) and we can invoke the report.

When we're done with a temporary tag, there's no point in leaving it around, so we just delete it.
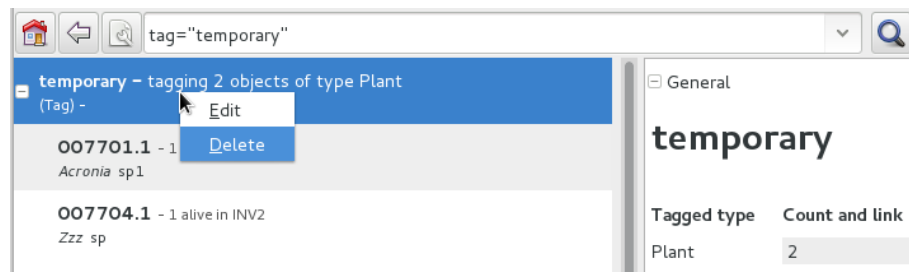
Be aware of the available search strategies

This is nicely documented, "più non dimandare" and read the docs.

4.1.2 using ghini for a seed database

We keep getting involved in groups focusing on endagered plant seeds. They want to note down when seeds come in, but also when they go out to people that order the seed.

In ghini, we keep speaking of ›Plants‹, ›Locations‹, while such user groups focus on ›Seeds‹ and ›Jars‹ and ›Drawers‹ and ›Boxes‹ and ›Envelopes‹. So people wonder whether ghini could be adapted to their use case, or for directions on how to develop their own database.

Does ghini need being adapted for such a seed database?

no it doesn't need any adaptation, it's just that you need to read some of its terms differently.

the taxonomy part is just taxonomy, plant species information, no need to explain that, no way to interpret it otherwise.

›Accessions‹ and ›Plants‹, you know what an ›Accession‹ is, but since you're consistently handling ›Plants‹ still only in seed form, the Wikipedia explanation of an accession sounds like this: it is a seed or group of seeds that are of the same taxon, are of the same propagule type (or treatment), were received from the same source, were received at the same time.

If you hold seeds in jars, or in other sort of containers that is able to hold hundreds of seeds, please make sure that a jar contains seeds of just one accession, as above described: same taxon, same treatment, same source, same time.

Each one of your ›Jars‹ of seeds is in ghini speak a ›Plant‹, and the amount of seeds in the ›Jar‹ is the ›Plant‹ ›quantity‹. An ›Envelope‹ is just the same as a ›Jar‹: a

container of seeds from the same ›Accession‹, just presumably smaller.

A ›Box‹ (where you keep several ›Envelopes‹) or a ›Drawer‹ (where you keep several ›Jars‹) are in ghini speak a ›Location‹.

Since a ›Jar‹ or an ›Envelope‹ contains seeds from an ›Accession‹, you will clearly label it with its ›Accession‹ code (and trailing ›Plant‹ number). You might write the amount of seeds, too, but this would be repeating information from the database, and repeating information introduces an inconsistency risk factor.

How do I handle receiving a batch of seeds?

Note: When we receive seeds, we either collect them ourselves, or we receive it from an other seed collector. We handle receiving them possibly on the spot, or with a small delay. Even when handled together with several other batches of seeds we received, each batch keeps its individuality.

We want to be later able to find back, for example, how many seeds we still have from a specific batch, or when we last received seeds from a specific source.

As long as you put this information in the database, as long as you follow the same convention when doing so, you will be able to write and execute such queries using ghini.

One possibility, the one described here, is based on ›Notes‹. (Ghini does not, as yet, implement the concept "Acquisition". There is an issue related to the Acquisition and Donation objects, but we haven't quite formalized things yet.)

You surely already use codes to identify a batch of seeds entering the seed bank. Just copy this code in a ›Note‹, category 'received', to each ›Accession‹ in the received batch. This will let you select the ›Accessions‹ by the query:

```
accession where notes[category='received'].note='<your code>
↪'
```

Use the 'Source' tab if you think so, it offers space for indicating an external source, or an expedition. When receiving from an external source, you can specify the code internal to their organization. This will be useful when requesting an extra batch.

How do I handle sending seeds?

what you physically do is to grab the desired amount of seeds of the indicated species from a jar, put it in an envelope and send it. what you do from a point of view of the database is exactly the same, but precisely described in a protocol:

- Use the database to identify the ›Jar‹ containing the desired amount of the right seeds.

- remove that amount of seeds from the ›Jar‹ (decrement the quantity),

- put the seeds in an ›Envelope‹ (yes, that's a database object).

- send the envelope (but keep it in the database).

this in short.

When I send seeds, it's not just one bag, how does ghini help me keeping things together?

There's two levels of keeping things together: one is while you're preparing the sending, and then for later reference.

While preparing the sending, we advise you use a temporary ›Tag‹ on the objects being edited.

For later reference, you will have common ›Note‹ texts, to identify received and sent batches.

Can you give a complete example?

Right. Quite fair. Let's see. . .

Say you were requested to deliver 50 seeds of Parnassia palustris, 30 of Gentiana pneumonanthe, 80 of Fritillaria meleagris, and 30 of Hypericum pulchrum.

step 1

The first step is to check the quantities you have in house, and if you do have enough, where you have them. You do this per requested species:

```
accession where species.genus.epithet=Parnassia and species.
→epithet=palustris and sum(plants.quantity)>0
```

Expand in the results pane the ›Accession‹ from which you want to grab the seeds, so you see the corresponding ›Jars‹, highlight one, and tag it with a new ›Tag‹. To do this the first time, go through the steps, just once, of creating a new ›Tag‹. The new tag becomes the active tag, and subsequent tagging will be speedier. I would call the tag ›sending‹, but that's only for ease of exposition and further completely irrelevant.

Repeat the task for Gentiana pneumonanthe, Fritillaria meleagris, Hypericum pulchrum:

```
accession where species.genus.epithet=Gentiana and species.
→epithet=pneumonanthe and sum(plants.quantity)>0
accession where species.genus.epithet=Fritillaria and
→species.epithet=meleagris and sum(plants.quantity)>0
accession where species.genus.epithet=Hypericum and species.
→epithet=pulchrum and sum(plants.quantity)>0
```

Again highilight the accession from which you can grab seeds, and hit Ctrl-Y (this tags the highighted row with the active tag). Don't worry if nothing seems to happen when you hit Ctrl-Y, this is a silent operation.

step 2

Now we prepare to go to the seeds bank, with the envelopes we want to fill.

Select the ›sending‹ ›Tag‹ from the tags menu, this will bring back in the results pane all the tagged ›Plants‹ (›Jars‹ or ›Envelopes‹), and will tell you in which ›Location‹ (›Drawer‹ or ›Box‹) they are to be found. Write this information on each of your physical envelopes. Write also the ›Species‹ name, and the quantity you can provide.

Walk now to your seeds bank and, for each of the envelopes you just prepared, open the ›Location‹, grab the ›Plant‹, extract the correct amount of seeds, put them in your physical envelope.

And back to the database!

step 3

If nobody used your workstation, you still have the Tag in the results pane, and it's expanded so you see all the individual plants you tagged.

One by one, you have to ›split‹ the plant. This is a standard operation that you activate by right-clicking on the plant.

A plant editor window comes in view, in 'split mode'.

Splitting a plant lets you create a database image of the plant group you just physically created, eg: it lets you subtract 30 items from the Gentiana pneumonanthe plant (group number one, that is the one in the jar), and create a new plant group for the same accession. A good practice would be to specify as ›Location‹ for this new plant the 'out box', that is, the envelope is on its way to leave the garden.

Don't forget to delete the temporary 'sending' ›Tag‹.

step 4

Final step, it represents the physical step of sending the envelope, possibly together with several other envelopes, in a single sending, which should have a code.

Just as you did when you received a batch of plants, you work with notes, this time the category is 'sent', and the note text is whatever you normally do to identify a sending. So suppose you're doing a second sending to Pino in 2018, you add the note to each of the newly created envelopes: category 'sent', text: '2018-pino-002'.

When you finally do send the envelopes, these stop being part of your collection. You still want to know that they have existed, but you do not want to count them among the seeds that are available to you.

Bring back all the plants in the sending '2018-pino-002':

```
plant where notes[category='sent'].note = '2018-pino-002'
```

You now need to edit them one by one, mark the ›quantity‹ to zero, and optionally specify the reason of the change, which would be ›given away‹, and the recipient is already specified in the 'sent' ›Note‹.

This last operation could be automated, we're thinking of it, it would become a script, acting on a selection. Stay tuned.

Administration

5.1 Database Administration

If you are using a real DBMS to hold your botanic data, then you need do something about database administration. While database administration is far beyond the scope of this document, we make our users aware of it.

5.1.1 SQLite

SQLite is not what one would consider a real DBMS: each SQLite database is just in one file. Make safety copies and you will be fine. If you don't know where to look for your database files, consider that, per default, bauble puts its data in the `~/.bauble/` directory.

In Windows it is somewhere in your `AppData` directory, most likely in `AppData\Roaming\Bauble`. Do keep in mind that Windows does its best to hide the `AppData` directory structure to normal users.

The fastest way to open it is with the file explorer: type `%APPDATA%` and hit enter.

5.1.2 MySQL

Please refer to the official documentation.

Backing up and restoring databases is described in breadth and depth starting at this page.

5.1.3 PostgreSQL

Please refer to the official documentation. A very thorough discussion of your backup options starts at chapter 24.

5.2 Ghini Configuration

Ghini uses a configuration file to store values across invocations. This file is associated to a user account and every user will have their own configuration file.

To review the content of the Ghini configuration file, type `:prefs` in the text entry area where you normally type your searches, then hit enter.

You normally do not need tweaking the configuration file, but you can do so with a normal text editor program. Ghini configuration file is at the default location for SQLite databases.

5.3 Reporting Errors

Should you notice anything unexpected in Ghini's behaviour, please consider filing an issue on the Ghini development site.

Ghini development site can be accessed via the Help menu.

Ghini Family

6.1 the Ghini family

Let's start by recalling the composition of the Ghini family, as shown in the diagram:
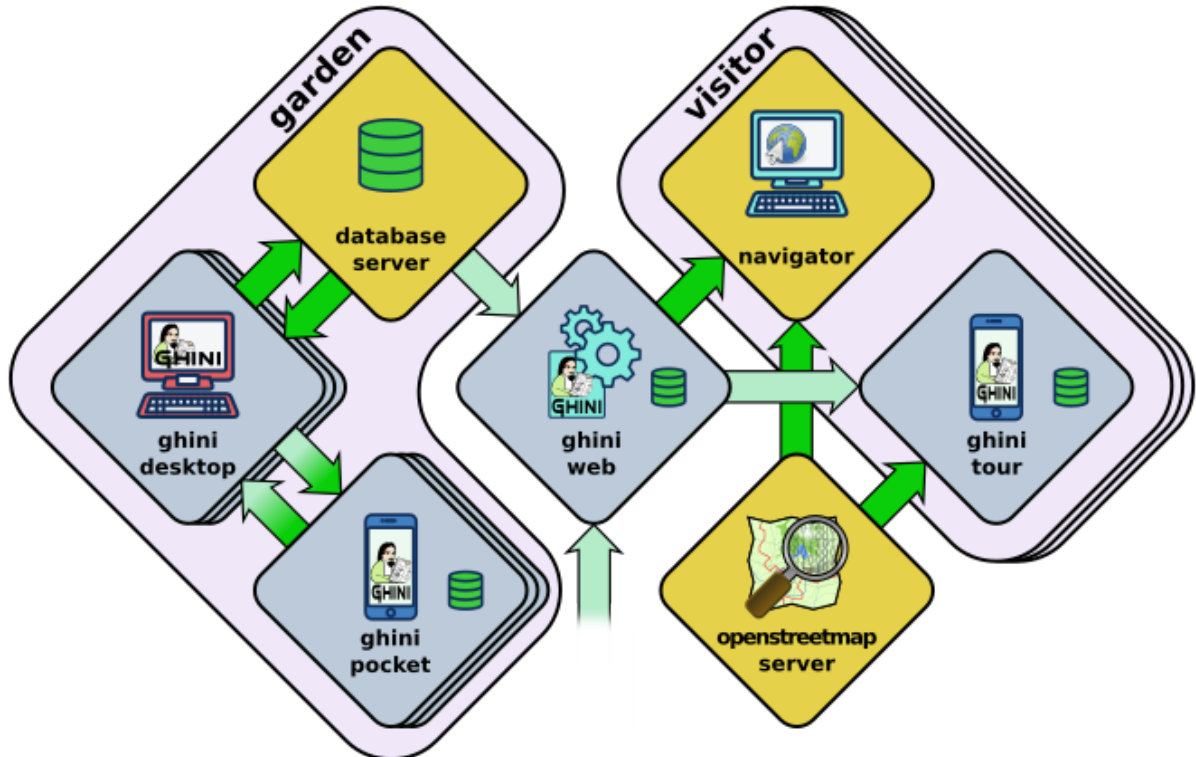
You have learned how to use ghini.desktop, here we introduce the other members of the family, and their interaction.

6.1.1 ghini.pocket

ghini.pocket is an Android app which you can install from the play store. ghini.pocket is definitely the tool you will use most, next to ghini.desktop.

With ghini.pocket you always have the latest snapshot of your database with you.

Type an accession number, or scan its barcode or QR label, and you know:

- the identification of the plant,
- whether it already has pictures,
- when it entered the garden and
- from which source.

Apart as a quick data viewer, you can use ghini.pocket for. . .

data correction

> If by your judgement, some of the information is incorrect, or if the plant is flowering and you want to immediately take a picture and store it in the database, you do not need take notes on paper, nor follow convolute procedures: ghini.pocket lets you write your corrections in a log file, take pictures associated to the plant, and you will import this information straight into the database, with further minimal user intervention.

inventory review

> The initial idea on which we based ghini.pocket is still one of its functionalities: inventory review.
>
> Using ghini.pocket, reviewing the inventory of a greenhouse, in particular if you have QR codes on plant labels, goes as fast as you can walk: simply enter the location code of your greenhouse, reset the log, then one by one scan the plant codes of the plants in the greenhouse. No further data collection action is required.
>
> When you're done, import the log in ghini.desktop. The procedure available in ghini.desktop includes adding unknown but labelled plants in the database, marking as lost/dead all plants that the database reports as alive and present in the inventoried location, but were not found during the inventory.

taxonomic support

As a bonus, ghini.pocket contains a phonetic genus search, and a quite complete database of botanic taxa with rank between order and genus, including tribes, and synonymies.

check further *Managing pocket/desktop interaction*.

6.1.2 ghini.web

ghini.web is a web server, written in nodejs.

Its most visible part runs at http://web.ghini.me and shows as a map of the world, where you browse gardens and search their published collection.

It also serves configuration data to ghini.tour instances.

check further *data streams between components*.

6.1.3 ghini.tour

ghini.tour is an Android app which you can install from the play store.

People visiting your garden will install ghini.tour on their phone or tablet, enjoy having a map of the garden, knowing where they are, and will be able to listen to audio files that you have placed as virtual information panels in strategic spots in your garden.

world view

at startup, you see the world and gardens. select a garden, and enter.

garden view

when viewing at garden level, you see panels. select a panel, and listen.

check further *data streams between components*.

6.1.4 data streams between components

In the diagram showing the composition of the Ghini family, the alert reader noticed how different arrows, corresponding to the different data flows, had different colours: some are deep green, some have a lighter tint, while the two streams connecting ghini.pocket and ghini.desktkop have a colour gradient, from light to deep green.
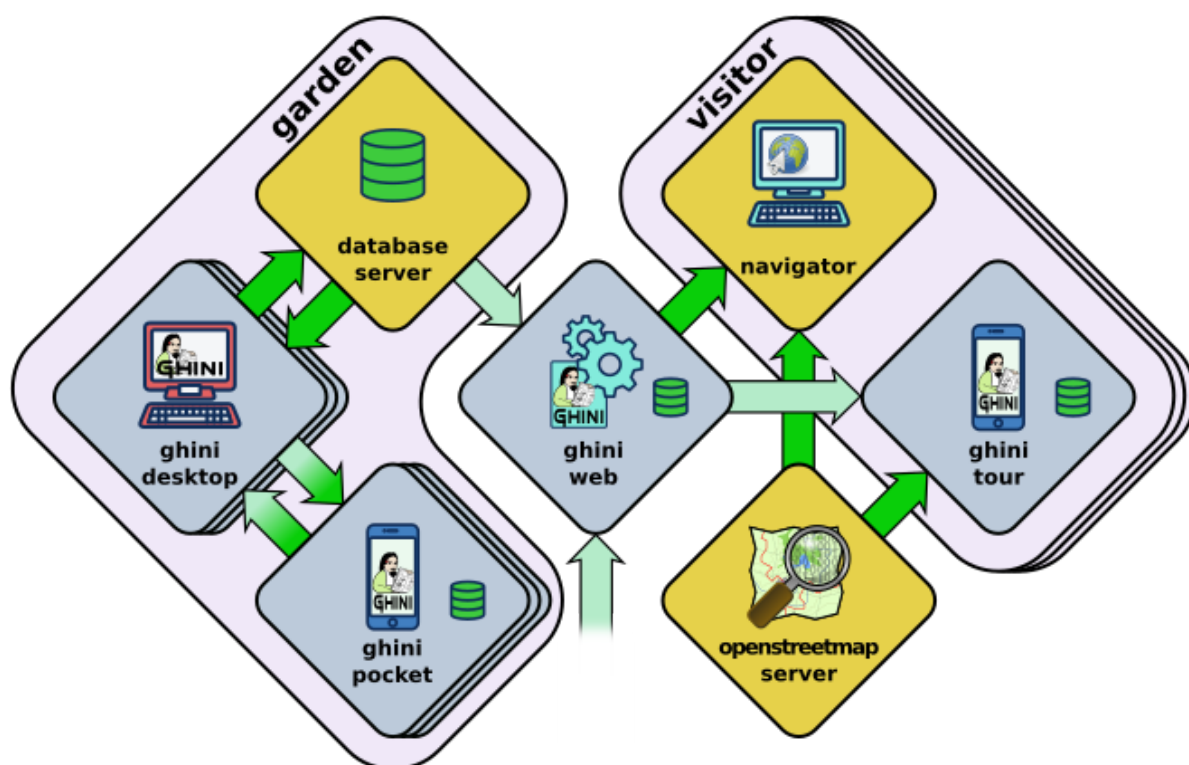
The colour choice for the arrows conveys a meaning: deeper green streams are constant flows of data, representing the core activity of a component, eg: the interaction between ghini.desktop and its database server, or your internet browser and ghini.web.

Lighter green streams are import/export actions, initiated by the user at the command panel of ghini.desktop, or in the ghini.tour settings page.

The interaction between ghini.pocket and ghini.desktop seen from the desktop, it is a stream you initiate when needed and stop as soon as you are done using it, for this reason it is light green at the desktop end. From the point of view of ghini.pocket however, it is an interaction with a data server, and we make this visible by giving the arrows a deep green colour at the pocket end.

This is again the same graph, but now all import data streams have been given an identifier, and where we added thin red lines ending in a thick dot, to indicate the control line from the controlling console to the controlled stream.
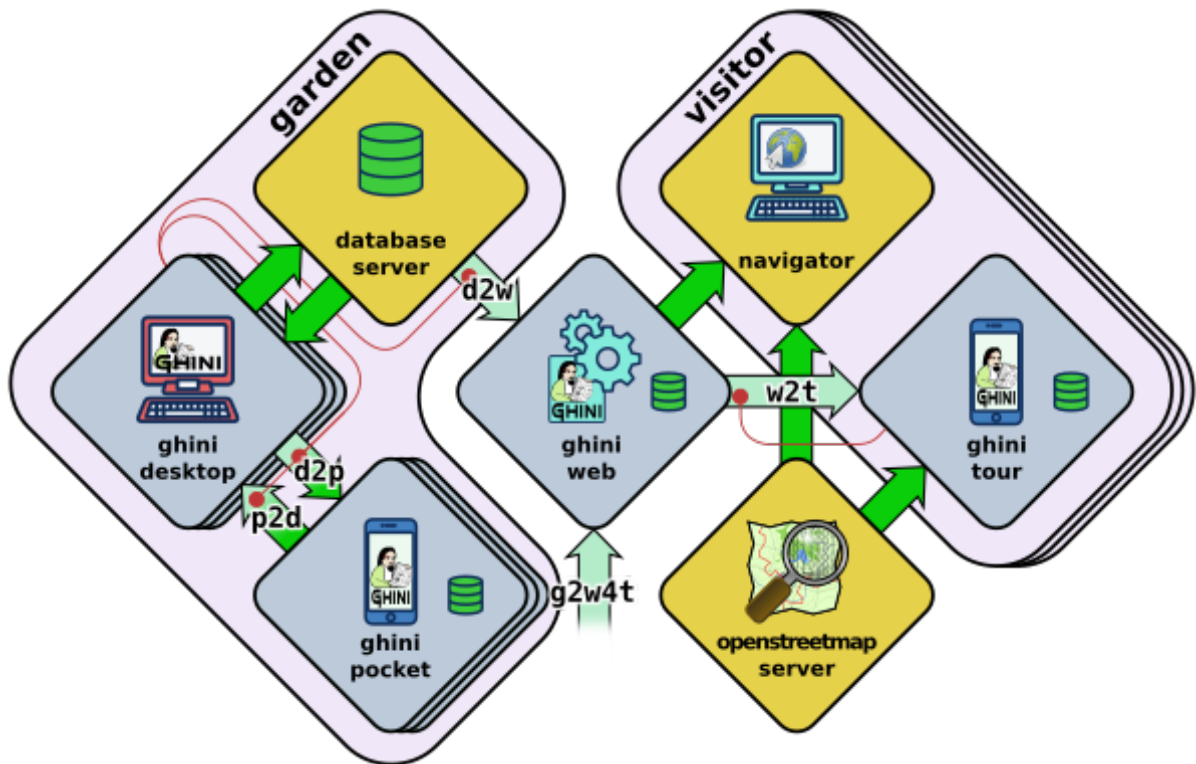
The interaction between ghini.pocket and ghini.desktop is one of the daily tasks of Ghini users, and has an integrated, hopefully user-friendly interface, at both the ghini.pocket and the ghini.desktop ends. This is described in the next section.

All import/export actions relating to ghini.web are still rather low level actions, so their description needs be of a rather technical type. These are handled in the remainig expandable sections, aimed mainly at database managers and possibly software developers.

d2w: send a selection of your garden data to ghini.web

Offer a selection of your garden data to a central ghini.web site, so online virtual visitors can browse it. This includes plant identification and their geographic location.

content of this flow:

- garden: coords, name, zoom level (for initial view)

- plants: coords, identification, zoom level (for visibility)

- species: binomial, phonetic approximation

There is a `stream_d2w.py` script for this, living in the `scripts` directory. It should really be integrated in ghini.desktop, some day.

g2w4t: add geographic non-botanic data to ghini.web

- Write geographic information about non-botanic data (ie: point of interest within the garden, required by ghini.tour) in the central ghini.web site.

content of this flow:

- – virtual panels: coords, title, audio file

- – photos: coords, title, picture

virtual panels don't necessarily have an associated photo, photos don't necessarily have an associated audio file.

---

w2t: importing locations and POIs from ghini.web to tour

content of this flow:

- Garden (coords, name, zoom level)

- Points of Interest (coords, title, audio file, photo)

6.1.5 Managing pocket/desktop interaction

Interaction between `ghini.pocket` and `ghini.desktop` is handled following a client server scheme. When you need to export a new database snapshot to pocket, or import the pocket log into the main database, you put `ghini.desktop` in server mode, so that it awaits requests from registered `ghini.pocket` clients.

`ghini.desktop` server mode

Select *Tools→Pocket Server..* to activate server mode on `ghini.desktop`. The Pocket Server console looks roughly like this, where it is creating a "snapshot":
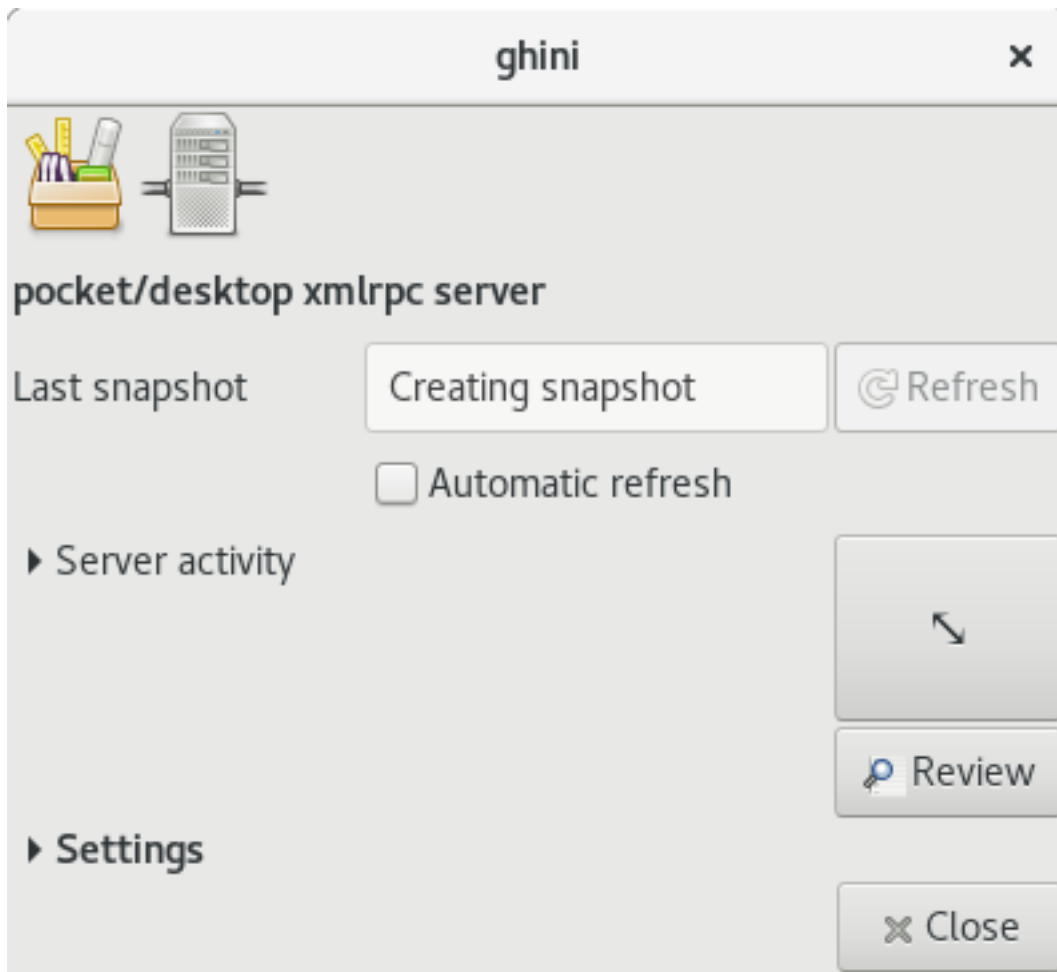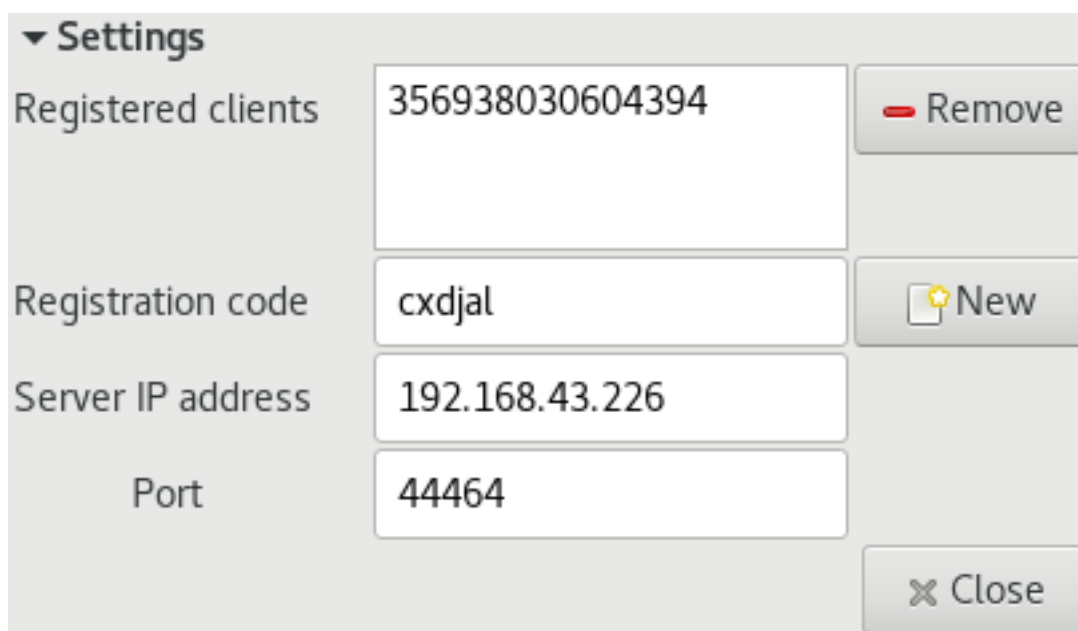
While in server mode, `ghini.desktop` is not available for other uses, and `ghini.pocket` clients will be able to register, request database snapshots, or send collected updates.

`ghini.pocket` works with a reduced database snapshot, containing only the most significant elements, under a simplified schema. `ghini.desktop` produces such a snapshot when it enters server mode, automatically and in the background. A typical Pocket Server session consists of: validate your identity, push updates from `ghini.pocket` clients, refresh your snapshot on `ghini.desktop`, pull the new snapshot to each of your clients.

Production of a snapshot is generally fast, surely for collections below the ten thousand plants, so you may prefer to keep the automatic refresh option enabled: a refresh will be triggered after receiving data from pocket clients and ghini.desktop will compute it while it is still receiving pictures. You will most likely not notice any delay. With automatic refresh enabled, the sequence reduces to: validate your identity, push updates, pull the new snapshot.

`ghini.desktop` and `ghini.pocket` need to be connected to the same (local) network. The server GUI includes an informative entry for the server IP address, this you obviously don't edit, and an entry for the port number, which defaults to GHINI/44464. Make sure that every configuration on your clients matches the settings on the server, i.e.: please copy your Server IP address, as shown by `ghini.desktop`, to all your `ghini.pocket` clients, and make sure that your clients are talking to the port on which the server will be listening.

I understand beans, what should I do?

Right, you're a botanist, with little interest in learning network management, quite fair.

Here are my three guesses for getting ghini.desktop (running on your computer) and ghini.pocket (running on your Android phone) to see each other:

1) Easiest case: you have a local network at home, to which you connect both your phone and your computer. Just make sure both your phone and computer are connected, and read further. In most cases, it will just work.

   Beware: there's routers that will not let you talk from one system to the other, or that leave only a few ports open. If your router has a 'guest' and a 'home' SSID, do use the 'home' signal for both systems. But it all depends, and you should check with your router. If you are using your home router and you still can't quite connect pocket to desktop, try the next option.

2) Next easiest: configure a WiFi hotspot on your Android phone, and connect your computer to it.

   Make sure your ghini.desktop computer has no other active network connection. You might want to disable mobile data on the phone, to avoid your computer to consume bandwith and make you incur in costs. Without mobile data, Android might warn you that the hotspot will not be functional. Ignore that: you're using the hotspot for local connections, not for getting on internet.

3) Still easy enough: connect your Android phone to your computer using a USB cable, and activate USB tethering on the phone.

   Again, make sure your computer has no other active network connection, and again consider disabling mobile data on the phone.

Any of the above three will put your two devices on the same network segment, and enable communication. Minimal terminology you might still check: IP Address; IP Port; Network Segment.

Check your phone IP address in *Settings → About device → Status*, no more to make sure that both addresses, desktop and pocket, belong to the same segment, roughly true if the first three of those four numbers are equal.

Port Number, GHINI/44464? What is that?

44464 is a port number and a port, think of it like of a virtual plug that a server program can enable and listen to, and to which client programs may attach and speak into. This port number, 44464 makes some sense because it's like GHINI on a phone keyboard. There is nothing special with this 44464, you can choose just about any number from 1025 to 65535, as long as it's not already in use by other programs and it's the same on both ghini.desktop and ghini.pocket.

localhost and your local network

Is your ghini.desktop showing the `127.0.0.1` address, that means that it is not connected to any network. That address is the numeric form of `localhost`, and only useful for local connections. You will **not** be able to connect pocket to desktop if this is what desktop reports as server IP address.

Your local IP address is something starting with `192.168.` followed by two more numbers between 0 and 255, or maybe `10.` followed by three more such numbers, or still `172.` followed by a number in the range 16 to 31, and again two more numbers between 0 and 255.

Anything else is something else, not your local IP address, and will not work.

ghini.desktop holds a list of registered ghini.pocket clients. (Check the below section on the client user interface, for how to register a phone.) The first time you activate ghini.pocket, it will create a persistent custom phone code, high chance it is a unique code, and this code will be used by both pocket and desktop to identify your phones. The chance that two of your 40 phones share the same code is smaller than a 2cm meteorite to strike your house within the next 5 minutes. Maybe not precisely this small, but definitely very small. When ghini.desktop receives a valid registration request for a phone code, the code is added to the list of registered clients and associated to the remote user name.

The client registration is persistent, and is used as a basic identity check for all client-server interaction. If you need to use the same phone with a different user name, you must overrule the existing registration, and to do so, you need the security code as shown in the desktop server settings.

Start the server on ghini.desktop by clicking on that thick square push button with the diagonal arrow: the button will stay pushed, the arrow slowly spinning, while the server is active. Decide if you prefer to enable automatic refresh, then move your attention focus to your

`ghini.pocket` client. It is from the `ghini.pocket` clients that you handle the communication. After accepting updates from your `ghini.pocket` clients, refresh the snapshot and update it on all your clients.
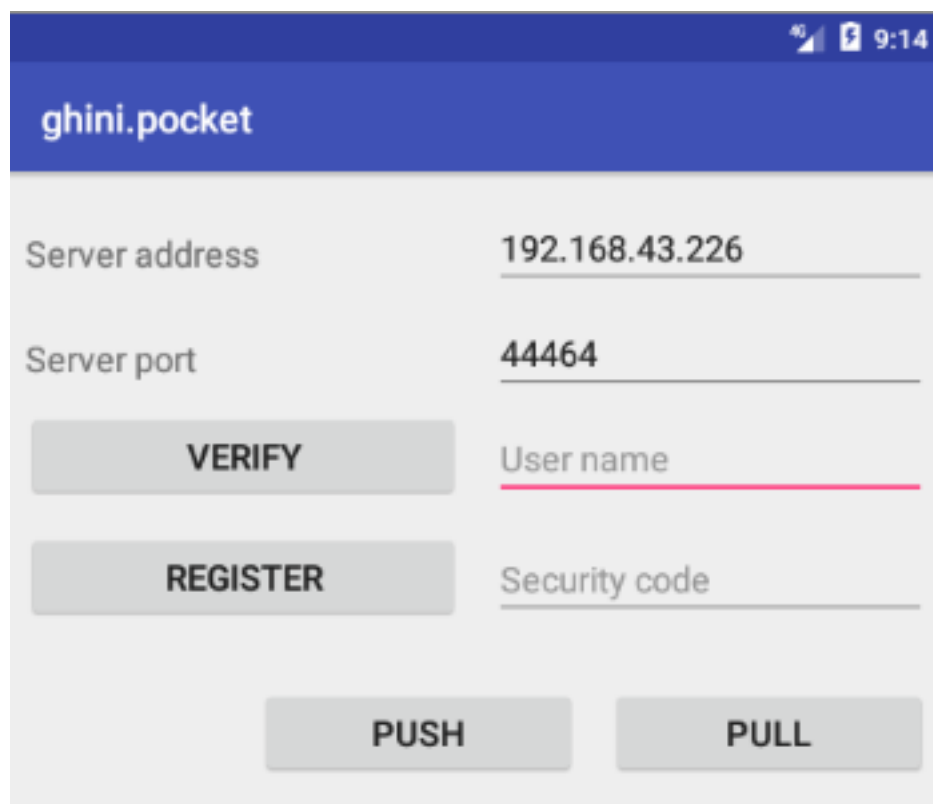
When done, stop the server, review the logs, close the Pocket Server window.

`ghini.pocket` user interface

With the pocket server running on ghini.desktop, grab now your Android phone with the latest ghini.pocket version.

`ghini.pocket` options menu has a `connect to g.desktop` item. Use it to activate the "desktop-client" window, which implements all interaction with the `ghini.desktop` server.

The "desktop-client" window contains data fields you have to edit in order to gain access to the server, and buttons for server communication. The communication buttons are not enabled unless you validate (that is: register, or verify if already registered) your identity.

The above is `ghini.pocket`'s "desktop-client" on my virtual phone. 192.168.43.226 is my laptop's IP address in my local network. Please replace that with your own computer address as previously explained.

`ghini.pocket` implements a very basic authentication check, trusting that your local network is secure. In fact the main goal of authentication between pocket and desktop is to make sure that you know which user is going to be credited with the edits you are supplying from pocket.

`verify` to check the communication parameters and the registered user name for your phone. Enter the server IP address, edit if necessary the communication port, and verify whether your

phone is already registered. If your phone id matches an already registered IMEI number, the User Name widget will show you the currently registered user name, and the bottom communication buttons will be enabled. If no user is registered for your phone, a notification will briefly flash on your phone, asking you to please register.

`register`, to associate your new user name to your phone IMEI code. Enter both a User Name, a Security Code, and press on `register`. This informs the desktop server that your phone, and through it all the information you push from your phone to desktop, is associated to your user name. Registration is permanent, so if you had already previously registered your phone with your name, you don't need registering again. You do need to register if you want to assign the same phone to a different user.

`pull` to refresh the `ghini.pocket` database with the snapshot from the server. This also resets the log, which gets anyway overruled by the new snapshot. Since this is a potentially destructive operation, you need to confirm you really mean it.

`push` to send your collected information: inventory log, corrections, pictures. In particular if you are sending pictures, this operation will take time. Please don't be surprised if copying 20 high resolution pictures, over your high speed local network connection, ghini is making you wait a couple of minutes: it's the sum that makes the total.

Exposed API

This is a technical reference section, you may safely ignore it if you aren't sure what it is about.

`ghini.desktop` runs an XML-RPC server, exposing the following API1. All functions but `verify` and `get_snapshot` return 0 on success; `verify` and `get_snapshot` return a string on success; for all functions, if the result is numeric, it is an error code.

---

verify(client_id)

> Return the user name associated to the client. The result is either a non-empty string, or a numeric error code.

---

register(client_id, user_name, security_code)

> Register the client on the server, associating it to the given user_name, given that the provided security_code matches the expected one.
>
> Overwrite any previous registration of the same client.
>
> Return 0 if successful, otherwise a numeric error code.

---

get_snapshot(client_id)

> Return the current `pocket.db` snapshot of the database.
>
> If client is not registered, return a numeric error code.

---

put_change(client_id, content, baseline)

Update the ghini database with the content of the collected pocket client logs.

Content is a complete log file, to be handled as a block.

Baseline is a timestamp, expressed as seconds since the Unix Epoch.

Each change line applies to a single plant or accession. Change lines are composed of change atoms, and atoms come in two flavours: additions (plant photos, accession verification) and overwrites (location name, location coordinates, quantity). Photos need be sent separately, one per request. Addition-atoms are always performed, while overwrite-atoms are approved, or rejected, based on their timestamp, the baseline value, and the last change of the affected object. In particular, an overwrite-atom is rejected if the affected object had been changed since the baseline, except of course if the change was part of this put_change action.

If client is not registered, return a numeric error code.

Returns a list of numeric values, each corresponding to a log line, indicating whether the line was successfully applied, or (partially) rejected.

put_picture(client_id, name, base64)

Add a picture to the collection. These are sent after the textual data has been updated. There is no check whether or not the picture is indeed referred to in the database.

If a picture by the same name already exists, the action fails with a numeric error code.

If client is not registered, return a numeric error code.

Ghini Development

7.1 Developer's Manual

7.1.1 Helping Ghini development

Installing Ghini always includes downloading the sources, connected to the github repository. This is so because in our eyes, every user is always potentially also a developer.

If you want to contribute to Ghini, you can do so in quite a few different ways:

- Use the software, note the things you don't like, open an issue for each of them. A developer will react sooner than you can imagine.

- If you have an idea of what you miss in the software but can't quite formalize it into separate issues, you could consider hiring a professional. This is the best way to make sure that something happens quickly on Ghini. Do make sure the developer opens issues and publishes their contribution on github.

- Translate! Any help with translations will be welcome, so please do! you can do this without installing anything on your computer, just using the on-line translation service offered by http://hosted.weblate.org/

- fork the respository, choose an issue, solve it, open a pull request. See the *bug solving workflow* below.

If you haven't yet installed Ghini, and want to have a look at its code history, you can open our github project page and see all that has been going on around Ghini since its inception as Bauble, back in the year 2004.

7.1.2 Software source, versions, branches

If you want a particular version of Ghini, we release and maintain versions as branches. You should `git checkout` the branch corresponding to the version of your choice.

production line

Branch names for Ghini stable (production) versions are of the form `ghini-x.y` (eg: ghini-3.1); branch names where Ghini testing versions are published are of the form `ghini-x.y-dev` (eg: ghini-3.1-dev).

7.1.3 Development Workflow

Our workflow is to continuously commit to the testing branch, to often push them to github, to let travis-ci and coveralls.io check the quality of the pushed testing branches, finally, from time to time, to merge the testing branch into the corresponding release.

When working at larger issues, which seem to take longer than a couple of days, I might open a branch associated to the issue. I don't do this very often.

larger issues

When facing a single larger issue, create a branch tag at the tip of a main development line (e.g.: `ghini-3.1-dev`), and follow the workflow described at

https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging

in short:

```
git up
git checkout -b issue-xxxx
git push origin issue-xxxx
```

Work on the new temporary branch. When ready, go to github, merge the branch with the main development line from which you branched, solve conflicts where necessary, delete the temporary branch.

When ready for publication, merge the development line into the corresponding production line.

7.1.4 Updating the set of translatable strings

From time to time, during the process of updating the software, you will be adding or modifying strings in the python sources, in the documentation, in the glade sources. Most of our strings are translatable, and are offered to weblate for people to contribute, in the form of several `.po` files.

A `po` is mostly composed of pairs of text portions, original and translation, and is specific to a target language. When a translator adds a translation on weblate, this reaches our repository on github. When a programmer adds a string to the software, this reaches weblate as "to be translated".

Weblate hosts the Ghini project. Within this project we have components, each of which corresponds to a branch of a repository on github. Each component accepts translations in several languages.

component	repository	branch
Desktop 1.0	ghini.desktop	ghini-1.0-dev
Desktop 3.1	ghini.desktop	ghini-3.1-dev
Documentation 1.0	ghini.desktop-docs.i18n	ghini-1.0-dev
Documentation 3.1	ghini.desktop-docs.i18n	ghini-3.1-dev
Web 1.2	ghini.web	master
Pocket	ghini.pocket	master

To update the `po` files relative to the *software*, you set the working directory to the root of your checkout of *ghini.desktop*, and you run the script:

```
./scripts/i18n.sh
```

To update the `po` files relative to the *documentation*, you set the working directory to the root of your checkout of *ghini.desktop-docs.i18n*, and you run the script:

```
./doc/runme.sh
```

When you run either of the above mentioned scripts, chances are you need to commit *all* `po` files in the project. You may want to review the changes before committing them to the repository. This is most important when you perform a marginal correction to a string, like removing a typo.

Something that happens: running into a conflict. Solving conflicts is not difficult once you know how to do that. First of all, add weblate as remote:

```
git remote add weblate-doc10 https://hosted.weblate.org/git/ghini/
↪documentation-10/
```

Then make sure we are in the correct repository, on the correct branch, update the remote, merge with it:

```
git checkout ghini-3.1-dev
git remote update
git merge weblate-doc10/ghini-3.1-dev
```

Our documentation on readthedocs has an original English version, and several translations. We just follow the description of localisation, there's nothing that we invented ourselves here.

Readthedocs checks the project's *Language* setting, and invokes `sphinx-intl` to produce the formatted documentation in the target language. With the default configuration —which we

did not alter— `sphinx-intl` expects one `po` file per source document, named as the source document, and that they all reside in the directory `local/$(LANG)/LC_MESSAGES/`.

On the other hand, Weblate (and ourselves) prefers a single `po` file per language, and keeps them all in the same `/po` directory, just as we do for the software project: `/po/$(LANG).po`.

In order not to repeat information, and to let both systems work their natural way, we have two sets of symbolic links (git honors them).

To summarise: when a file in the documentation is updated, the `runme.sh` script will:

1. copy the `rst` files from the software to the documentation;

2. create a new `pot` file for each of the documentation files;

3. merge all `pot` files into one `doc.pot`;

4. use the updated `doc.pot` to update all `doc.po` files (one per language);

5. create all symbolic links:

    a. those expected by `sphinx-intl` in `/local/$(LANG)/LC_MESSAGES/`

    b. those used by weblate in `/po/$(LANG).po`

We could definitely write the above in a Makefile, or even better include it in `/doc/Makefile`. Who knows, maybe we will do that.

7.1.5 Producing the docs locally

The above description is about how we help external sites produce our documentation so that it is online for all to see. But what if you want to have the documentation locally, for example if you want to edit and review before pushing your commits to the cloud?

In order to run sphinx locally, you need to install it **within** the same virtual environment as ghini, and to install it there, you need to have a sphinx version whose dependencies don not conflict with ghini.desktop's dependecies.

What we do to keep this in order?

We state this extra dependency in the `setup.py` file, as an `extras_require` entry. Create and activate the virtual environment, then run `easy_install ghini.desktop[docs]`. This gets you the sphinx version as declared in the `setup.py` file.

If all you want is the html documentation built locally, run `./setup.py install docs`. For more options, enter the `doc` directory and run `make`.

7.1.6 Which way do the translated strings reach our users?

A new translator asked the question, adding: »Is this an automated process from Weblate –> GIT –> Ghini Desktop installed on users computers, or does this require manual steps?

The aswer is that the whole interaction is quite complex, and it depends on the component.

When it comes to `ghini.desktop`, when you do a global installation, you don't know which language your users will set up their environment, and a user can change the language configuration any time. So what we do is to install the software in English together with a translation table from English to whatever else, in particular Italian or Hungarian. Then the GUI libraries (Android or GTK) know where to look for the translation strings. These translation tables are generated during the installation or upgrade process, based on the strings you see on Weblate.

Before any of the above gets activated, the path followed by your translations is as you describe: Weblate pushes the strings to github, directly into the development line *ghini-3.1-dev*; I see them, if I understand at least the structure of that language I review them, maybe I look them up in wikipedia or get them translated back to Italian, Spanish or English by some automatic translation service; sometimes I need to solve conflicts arising because of changed context, not too often fortunately; from time to time I publish the development line *ghini-3.1-dev* to the production line *ghini-3.1*, and this is the moment when the new translations finally make it to the distributed software.

users will notice a *new version available* warning and can decide to ignore it, or to update.

For `ghini.pocket`, there is no notification to end users, since we're not yet using the google app store.

For `ghini.web`, we haven't yet defined how to distribute it.

For ghini's documentation, it's completely automatic, and all is handled by readthedocs.org.

7.1.7 Adding missing unit tests

If you are interested contributing to development of Ghini, a good way to do so would be by helping us finding and writing the missing unit tests.

A well tested function is one whose behaviour you cannot change without breaking at least one unit test.

We all agree that in theory theory and practice match perfectly and that one first writes the tests, then implements the function. In practice, however, practice does not match theory and we have been writing tests after writing and even publishing the functions.

This section describes the process of adding unit tests for `bauble.plugins.plants.family.remove_callback`.

What to test

First of all, open the coverage report index, and choose a file with low coverage.

For this example, run in October 2015, we landed on `bauble.plugins.plants.family`, at 33%.

https://coveralls.io/builds/3741152/source?filename=bauble%2Fplugins%2Fplants%2Ffamily.py

The first two functions which need tests, `edit_callback` and `add_genera_callback`, include creation and activation of an object relying on a custom dialog box. We should really first write unit tests for that class, then come back here.

The next function, `remove_callback`, also activates a couple of dialog and message boxes, but in the form of invoking a function requesting user input via yes-no-ok boxes. These functions we can easily replace with a function mocking the behaviour.

how to test

So, having decided what to describe in unit test, we look at the code and we see it needs discriminate a couple of cases:

parameter correctness

- the list of families has no elements.
- the list of families has more than one element.
- the list of families has exactly one element.

cascade

- the family has no genera
- the family has one or more genera

confirm

- the user confirms deletion
- the user does not confirm deletion

deleting

- all goes well when deleting the family
- there is some error while deleting the family

I decide I will only focus on the **cascade** and the **confirm** aspects. Two binary questions: 4 cases.

where to put the tests

Locate the test script and choose the class where to put the extra unit tests.

https://coveralls.io/builds/3741152/source?filename=bauble%2Fplugins%2Fplants%2Ftest.py#L273

---

what about skipped tests

> The `FamilyTests` class contains a skipped test, implementing it will be quite a bit of work because we need rewrite the FamilyEditorPresenter, separate it from the FamilyEditorView and reconsider what to do with the FamilyEditor class, which I think should be removed and replaced with a single function.

---

writing the tests

After the last test in the FamilyTests class, I add the four cases I want to describe, and I make sure they fail, and since I'm lazy, I write the most compact code I know for generating an error:

```python
def test_remove_callback_no_genera_no_confirm(self):
    1/0

def test_remove_callback_no_genera_confirm(self):
    1/0

def test_remove_callback_with_genera_no_confirm(self):
    1/0

def test_remove_callback_with_genera_confirm(self):
    1/0
```

One test, step by step

Let's start with the first test case.

When writing tests, I generally follow the pattern:

- T_0 (initial condition),
- action,
- T_1 (testing the result of the action given the initial conditions)

---

what's in a name — unit tests

There's a reason why unit tests are called unit tests. Please never test two actions in one test.

So let's describe T_0 for the first test, a database holding a family without genera:

```python
def test_remove_callback_no_genera_no_confirm(self):
    f5 = Family(family=u'Arecaceae')
    self.session.add(f5)
    self.session.flush()
```

We do not want the function being tested to invoke the interactive `utils.yes_no_dialog` function, we want `remove_callback` to invoke a non-interactive replacement function. We achieve this simply by making `utils.yes_no_dialog` point to a `lambda` expression which, like the original interactive function, accepts one parameter and returns a boolean. In this case: `False`:

```
def test_remove_callback_no_genera_no_confirm(self):
    # T_0
    f5 = Family(family=u'Arecaceae')
    self.session.add(f5)
    self.session.flush()

    # action
    utils.yes_no_dialog = lambda x: False
    from bauble.plugins.plants.family import remove_callback
    remove_callback(f5)
```

Next we test the result.

Well, we don't just want to test whether or not the object Arecaceae was deleted, we also should test the value returned by `remove_callback`, and whether `yes_no_dialog` and `message_details_dialog` were invoked or not.

A `lambda` expression is not enough for this. We do something apparently more complex, which will make life a lot easier.

Let's first define a rather generic function:

```
def mockfunc(msg=None, name=None, caller=None, result=None):
    caller.invoked.append((name, msg))
    return result
```

and we grab `partial` from the `functools` standard module, to partially apply the above `mockfunc`, leaving only `msg` unspecified, and use this partial application, which is a function accepting one parameter and returning a value, to replace the two functions in `utils`. The test function now looks like this:

```
def test_remove_callback_no_genera_no_confirm(self):
    # T_0
    f5 = Family(family=u'Arecaceae')
    self.session.add(f5)
    self.session.flush()
    self.invoked = []

    # action
    utils.yes_no_dialog = partial(
        mockfunc, name='yes_no_dialog', caller=self, result=False)
    utils.message_details_dialog = partial(
        mockfunc, name='message_details_dialog', caller=self)
    from bauble.plugins.plants.family import remove_callback
    result = remove_callback([f5])
    self.session.flush()
```

The test section checks that `message_details_dialog` was not invoked, that `yes_no_dialog` was invoked, with the correct message parameter, that Arecaceae is still there:

```python
# effect
self.assertFalse('message_details_dialog' in
                 [f for (f, m) in self.invoked])
self.assertTrue(('yes_no_dialog', u'Are you sure you want to '
                 'remove the family <i>Arecaceae</i>?')
                in self.invoked)
self.assertEquals(result, None)
q = self.session.query(Family).filter_by(family=u"Arecaceae")
matching = q.all()
self.assertEquals(matching, [f5])
```

And so on

there are two kinds of people, those who complete what they start, and so on

Next test is almost the same, with the difference that the `utils.yes_no_dialog` should return `True` (this we achieve by specifying `result=True` in the partial application of the generic `mockfunc`).

With this action, the value returned by `remove_callback` should be `True`, and there should be no Arecaceae Family in the database any more:

```python
def test_remove_callback_no_genera_confirm(self):
    # T_0
    f5 = Family(family=u'Arecaceae')
    self.session.add(f5)
    self.session.flush()
    self.invoked = []

    # action
    utils.yes_no_dialog = partial(
        mockfunc, name='yes_no_dialog', caller=self, result=True)
    utils.message_details_dialog = partial(
        mockfunc, name='message_details_dialog', caller=self)
    from bauble.plugins.plants.family import remove_callback
    result = remove_callback([f5])
    self.session.flush()

    # effect
    self.assertFalse('message_details_dialog' in
                     [f for (f, m) in self.invoked])
    self.assertTrue(('yes_no_dialog', u'Are you sure you want to '
                     'remove the family <i>Arecaceae</i>?')
                    in self.invoked)
    self.assertEquals(result, True)
    q = self.session.query(Family).filter_by(family=u"Arecaceae")
    matching = q.all()
    self.assertEquals(matching, [])
```

have a look at commit 734f5bb9feffc2f4bd22578fcee1802c8682ca83 for the other two test

functions.

Testing logging

Our `bauble.test.BaubleTestCase` objects use handlers of the class `bauble.test.MockLoggingHandler`. Every time an individual unit test is started, the `setUp` method will create a new `handler` and associate it to the root logger. The `tearDown` method takes care of removing it.

You can check for presence of specific logging messages in `self.handler.messages`. `messages` is a dictionary, initially empty, with two levels of indexation. First the name of the logger issuing the logging record, then the name of the level of the logging record. Keys are created when needed. Values hold lists of messages, formatted according to whatever formatter you associate to the handler, defaulting to `logging.Formatter("%(message)s")`.

You can explicitly empty the collected messages by invoking `self.handler.clear()`.

Putting all together

From time to time you want to activate the test class you're working at:

```
nosetests bauble/plugins/plants/test.py:FamilyTests
```

And at the end of the process you want to update the statistics:

```
./scripts/update-coverage.sh
```

7.1.8 Structure of user interface

The user interface is built according to the **Model — View — Presenter** architectural pattern. For much of the interface, **Model** is a SQLAlchemy database object, but we also have interface elements where there is no corresponding database model. In general:

- The **View** is described as part of a **glade** file. This should include the signal-callback and ListStore-TreeView associations. Just reuse the base class `GenericEditorView` defined in `bauble.editor`. When you create your instance of this generic class, pass it the **glade** file name and the root widget name, then hand this instance over to the **presenter** constructor.

  In the glade file, in the `action-widgets` section closing your GtkDialog object description, make sure every `action-widget` element has a valid `response` value. Use valid GtkResponseType values, for example:

  - GTK_RESPONSE_OK, -5
  - GTK_RESPONSE_CANCEL, -6
  - GTK_RESPONSE_YES, -8
  - GTK_RESPONSE_NO, -9

There is no easy way to unit test a subclassed view, so please don't subclass views, there's really no need to.

In the glade file, every input widget should define which handler is activated on which signal. The generic Presenter class offers generic callbacks which cover the most common cases.

- GtkEntry (one-line text entry) will handle the `changed` signal, with either `on_text_entry_changed` or `on_unique_text_entry_changed`.

- GtkTextView: associate it to a GtkTextBuffer. To handle the `changed` signal on the GtkTextBuffer, we have to define a handler which invokes the generic `on_textbuffer_changed`, the only role for this function is to pass our generic handler the name of the model attribute that receives the change. This is a workaroud for an unresolved bug in GTK.

- GtkComboBox with translated texts can't be easily handled from the glade file, so we don't even try. Use the `init_translatable_combo` method of the generic `GenericEditorView` class, but please invoke it from the **presenter**.

- The **Model** is just an object with known attributes. In this interaction, the **model** is just a passive data container, it does nothing more than to let the **presenter** modify it.

- The subclassed **Presenter** defines and implements:

  - `widget_to_field_map`, a dictionary associating widget names to name of model attributes,

  - `view_accept_buttons`, the list of widget names which, if activated by the user, mean that the view should be closed,

  - all needed callbacks,

  - optionally, it plays the **model** role, too.

The **presenter** continuously updates the **model** according to changes in the **view**. If the **model** corresponds to a database object, the **presenter** commits all **model** updates to the database when the **view** is closed successfully, or rolls them back if the **view** is canceled. (this behaviour is influenced by the parameter `do_commit`)

If the **model** is something else, then the **presenter** will do something else.

Note: A well behaved **presenter** uses the **view** api to query the values inserted by the user or to forcibly set widget statuses. Please do not learn from the practice of our misbehaving presenters, some of which directly handle fields of `view.widgets`. By doing so, these presenters prevents us from writing unit tests.

The base class for the presenter, `GenericEditorPresenter` defined in `bauble. editor`, implements many useful generic callbacks. There is a `MockView` class, that you can use when writing tests for your presenters.

Examples

`Contact` and `ContactPresenter` are implemented following the above lines. The view is defined in the `contact.glade` file.

A good example of Presenter/View pattern (no model) is given by the connection manager.

We use the same architectural pattern for non-database interaction, by setting the presenter also as model. We do this, for example, for the JSON export dialog box. The following command will give you a list of `GenericEditorView` instantiations:

```
grep -nHr -e GenericEditorView\( bauble
```

7.1.9 Data streams between software components

Let's start by recalling the composition of the Ghini family, as shown in the diagram:
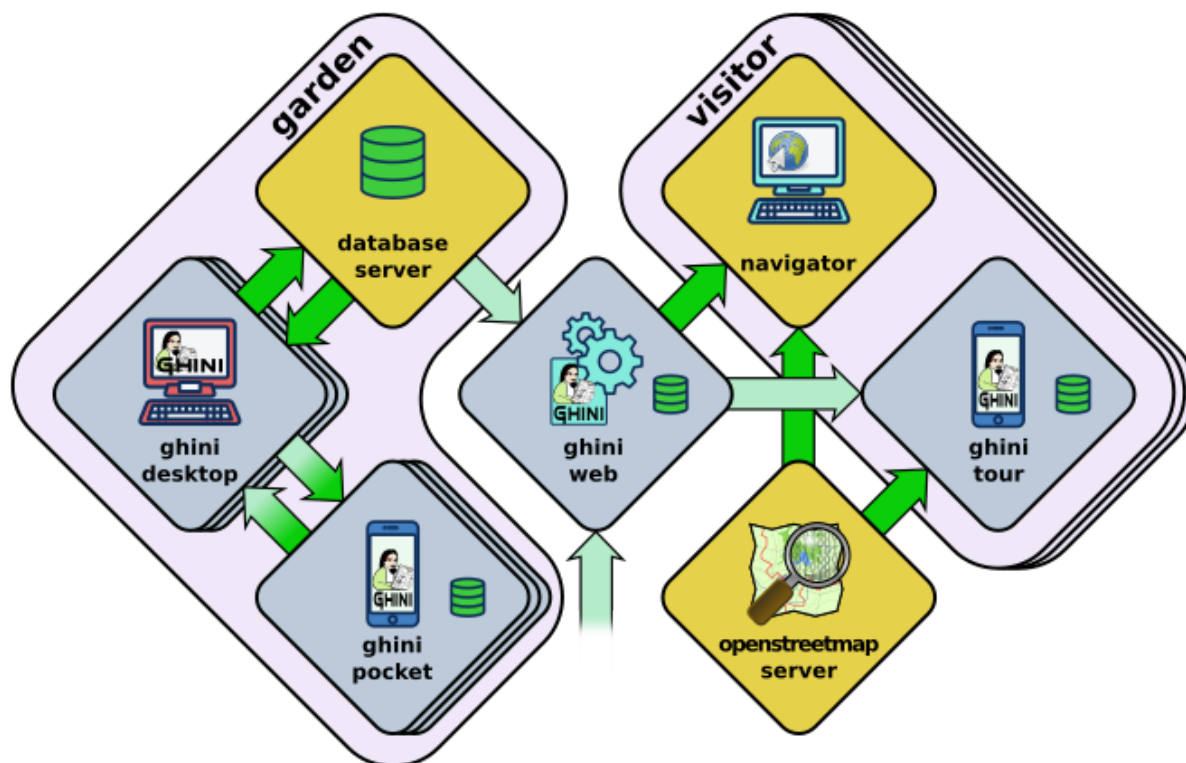
When we first introduced the diagram, we did not explain the reason why different arrows representing different data flows, had different colours: some are deep green, some in a lighter tint. If you suspected this bore a meaning then you were quite right:

Deeper green streams are constant flows of data, representing the core activity of a component, eg: the interaction between ghini.desktop and its database server, or your internet browser and ghini.web.

Lighter green streams are import/export actions, initiated by the user at the command panel of ghini.desktop, or in the ghini.tour settings page.

This is the same graph, in which all import data streams have been given an identifier.
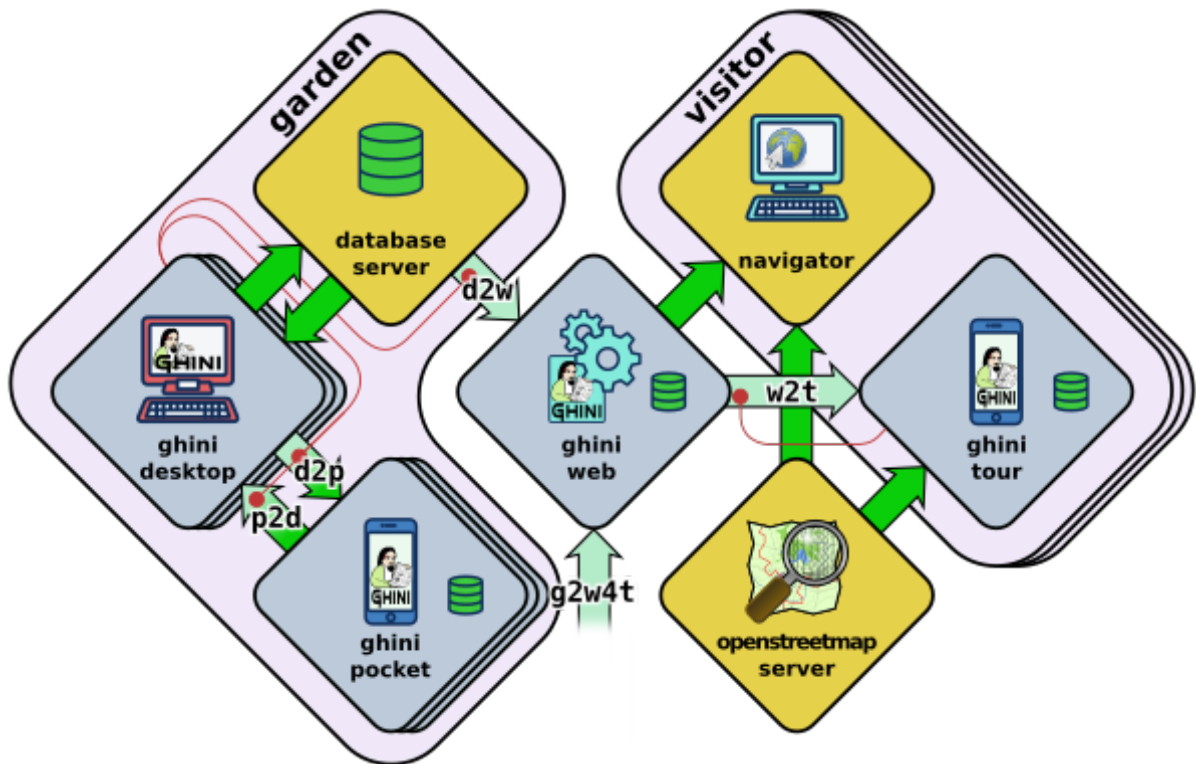
Table 1: Stream role description

name	description
d2p	This is ghini.desktop's *Tools→Export→export to pocket*.
p2d	Import from the ghini.pocket log file and pictures into the central database.
d2w	Offer a selection of your garden data to a central ghini.web site, so online virtual visitors can browse it. This includes plant identification and their geographic location.
g2w	Write geographic information about non-botanic data (ie: point of interest within the garden, required by ghini.tour) in the central ghini.web site.
w2t	Importing locations and points of interest from ghini.web to tour.

We formally define all named streams, so our we know we are talking about. Moreover, streams impacting the desktop and web databases require extra thought and attention from your database manager.

7.1.10 Extending Ghini with Plugins

Nearly everything about Ghini is extensible through plugins. Plugins can create tables, define custom searchs, add menu items, create custom commands and more.

To create a new plugin you must extend the `bauble.pluginmgr.Plugin` class.

The `Tag` plugin is a good minimal example, even if the `TagItemGUI` falls outside the Model-View-Presenter architectural pattern.

7.1.11 Plugins structure

Ghini is a framework for handling collections, and is distributed along with a set of plugins making Ghini a botanical collection manager. But Ghini stays a framework and you could in theory remove all plugins we distribute and write your own, or write your own plugins that extend or complete the current Ghini behaviour.

Once you have selected and opened a database connection, you land in the Search window. The Search window is an interaction between two objects: SearchPresenter (SP) and SearchView (SV).

SV is what you see, SP holds the program status and handles the requests you express through SV. Handling these requests affect the content of SV and the program status in SP.

The search results shown in the largest part of SV are rows, objects that are instances of classes registered in a plugin.

Each of these classes must implement an amount of functions in order to properly behave within the Ghini framework. The Ghini framework reserves space to pluggable classes.

SP knows of all registered (plugged in) classes, they are stored in a dictionary, associating a class to its plugin implementation. SV has a slot (a gtk.Box) where you can add elements. At any time, at most only one element in the slot is visible.

A plugin defines one or more plugin classes. A plugin class plays the role of a partial presenter (pP - plugin presenter) as it implement the callbacks needed by the associated partial view fitting in the slot (pV - plugin view), and the MVP pattern is completed by the parent presenter (SP), again acting as model. To summarize and complete:

- SP acts as model,

- the pV partial view is defined in a glade file.

- the callbacks implemented by pP are referenced by the glade file.

- a context menu for the SP row,

- a children property.

when you register a plugin class, the SP:

- adds the pV in the slot and makes it non-visible.

- adds an instance of pP in the registered plugin classes.

- tells the pP that the SP is the model.

- connects all callbacks from pV to pP.

when an element in pV triggers an action in pP, the pP can forward the action to SP and can request SP that it updates the model and refreshes the view.

When the user selects a row in SP, SP hides everything in the pluggable slot and shows only the single pV relative to the type of the selected row, and asks the pP to refresh the pV with whatever is relative to the selected row.

Apart from setting the visibility of the various pV, nothing needs be disabled nor removed: an invisible pV cannot trigger events!

7.1.12 bug solving workflow

normal development workflow

- while using the software, you notice a problem, or you get an idea of something that could be better, you think about it good enough in order to have a very clear idea of what it really is, that you noticed. you open an issue and describe the problem. someone might react with hints.

- you open the issues site and choose one you want to tackle.

- assign the issue to yourself, this way you are informing the world that you have the intention to work at it. someone might react with hints.

- optionally fork the repository in your account and preferably create a branch, clearly associated to the issue.

- write unit tests and commit them to your branch (please do not push failing unit tests to github, run `nosetests` locally first).

- write more unit tests (ideally, the tests form the complete description of the feature you are adding or correcting).

- make sure the feature you are adding or correcting is really completely described by the unit tests you wrote.

- make sure your unit tests are atomic, that is, that you test variations on changes along one single variable. do not give complex input to unit tests or tests that do not fit on one screen (25 lines of code).

- write the code that makes your tests succeed.

- update the i18n files (run `./scripts/i18n.sh`).

- whenever possible, translate the new strings you put in code or glade files.

- when you change strings, please make sure that old translations get re-used.

- commit your changes.

- push to github.

- open a pull request.

publishing to production

please use the `publish.sh` script, in the `scritps` directory. This one takes care of every single step, and produces recognizable commit comments, it publishes the release on pypi, and in perspective it will contain all steps for producing a `deb` file, and a windows executable.

you can also do this by hand:

- open the pull request page using as base a production line `ghini-x.y`, compared to `ghini-x.y-dev`.

- make sure a `bump` commit is included in the differences.

- it should be possible to automatically merge the branches.

- create the new pull request, call it as "publish to the production line".

- you possibly need wait for travis-ci to perform the checks.

- merge the changes.

don't forget to tell the world about the new release: on facebook, the google group, in any relevant linkedin group, and on our web page.

your own fork

If you want to keep your own fork of the project, keep in mind this is full force work in progress, so staying up to date will require some effort from your side.

The best way to keep your own fork is to focus on some specific issue, work relatively quickly, often open pull requests for your work, make sure that you get it accepted. Just follow Ghini's coding style, write unit tests, concise and abundant, and there should be no problem in having your work included in Ghini's upstream.

If your fork got out of sync with Ghini's upstream: read, understand, follow the github guides configuring a remote for a fork and syncing a fork.

closing step

- review this workflow. consider this as a guideline, to yourself and to your colleagues. please help make it better and matching the practice.

7.1.13 distributing for windows

For building a Windows installer or executable you will need an installation of Windows. The methods described here has been used successfully on Windows 7, 8 and 10. Windows Vista should also work but has not been tested.

In the remainder of this section we assume you're using a Windows workstation. We also assume assume you do not use it as your software development platform. All steps described here are very similar to the steps for a normal Windows installation.

py2exe will not work with eggs

Building a Windows executable with py2exe requires packages **not** be installed as eggs. There are several methods to accomplish this, including:

- Using pip to install. The easiest method is to install into a virtual environment that doesn't currently have any modules installed as eggs using `pip install .` as described below. If you do wish to install over the top of an install with eggs (e.g. the environment created by `devinstall.bat`) you can try `pip install -I .` but your mileage may vary.

- By adding:

```
[easy_install]
zip_ok = False
```

to setup.cfg (or similarly `zip_safe = False` to `setuptools.setup()` in `setup.py`) you can use `python setup.py install` but you will need to download and install Microsoft Visual C++ Compiler for Python 2.7 to get any of the C extensions and will need a fresh virtual environment with no dependent packages installed as eggs.

1. Download and install git, Python 2.7 and PyGTK as outlined in the generic installation instructions.

2. Additionally, download and install NSIS v3.

3. A **reboot** is recommended.

we have a script that automates the remaining steps

A batch file is available that can complete the last few steps. To use it use this command:

```
scripts\build_win.bat
```

`build_win.bat` accepts 2 arguments:

1. `/e` will produce an executable only, skipping the extra step of building an installer, and will copy `win_gtk.bat` into place.

2. A path to the location for the virtual environment to use. (defaults to `"%HOMEDRIVE%%HOMEPATH%"\.virtualenvs\ghi2exe`)

e.g. to produce an executable only and use a virtual environment in a folder beside where you have ghini.desktop you could execute `scripts\build_win.bat /e . .\ghi2exe`

4. Clone ghini.desktop to wherever you want to keep it (replace `<path-to-keep-ghini>` with the path of your choice, e.g. `Local\github\Ghini\`) and checkout a production branch (`ghini-3.1` is recommended as used in the example). To do this, open a command prompt and type these commands:

```
cd <path-to-keep-ghini>
git clone https://github.com/Ghini/ghini.desktop.git
```

(continues on next page)

```
cd ghini.desktop
git checkout ghini-3.1
```

5. Install virtualenv, create a virtual environment and activate it. With only Python 2.7 on your system (where `<path-to-venv>` is the path to where you wish to keep the virtual environment) use:

```
pip install virtualenv
virtualenv --system-site-packages <path-to-venv>
call <path-to-venv>\Scripts\activate.bat
```

On systems where Python 3 is also installed you may need to either call pip and virtualenv with absolute paths e.g. `C:\Python27\Scripts\pip` or use the Python launcher e.g. `py -2.7 -m pip` (run `python --version` first to check. If you get anything other than version 2.7 you'll need to use one of these methods.)

6. Install dependencies and ghini.desktop into the virtual environment:

```
pip install psycopg2 Pygments py2exe_py2
pip install .
```

7. Build the executable:

```
python setup.py py2exe
```

The `dist` folder will now contain a full working copy of the software in a frozen, self contained state, that can be transferred however you like and will work in place. (e.g. placed on a USB flash drive for demonstration purposes or copied manually to `C:\Program Files` with a shortcut created on the desktop). To start ghini.desktop double click `ghini.exe` in explorer (or create a shortcut to it). If you have issues with the UI not displaying correctly you need to run the script `win_gtk.bat` from the `dist` folder to set up paths to the GTK components correctly. (Running `build_win /e` will place this script in the dist folder for you or you can copy it from the `scripts` folder yourself.) You will only need to run this once each time the location of the folder changes. Thereafter `ghini.exe` will run as expected.

8. Build the installer:

```
python setup.py nsis
```

This should leave a file named `ghini.desktop-<version>-setup.exe` in the `scripts` folder. This is your Windows installer.

about the installer

- Capable of single user or global installs.

- At this point in time ghini.desktop installed this way will not check or or notify you of any updated version. You will need to check yourself.

- Capable of downloading and installing optional extra components:

    - Apache FOP - If you want to use xslt report templates install FOP. FOP requires Java Runtime. If you do not currently have it installed the installer will let you know and offer to open the Oracle web site for you to download and install it from.

    - MS Visual C runtime - You most likely don't need this but if you have any trouble getting ghini.desktop to run try installing the MS Visual C runtime (e.g. rerun the installer and select this component only).

- Can be run silently from the commandline (e.g. for remote deployment) with the following arguments:

    - `/S` for silent;

    - `/AllUser` (when run as administrator) or `/CurrentUser`

    - `/C=[gFC]` to specify components where:

        `g` = Deselect the main ghini.desktop component (useful for adding optional component after an initial install)

        `F` = select Apache FOP

        `C` = select MS Visual C runtime

7.2 Template Letters

The reader getting to this point in the documentation probably understood that this Ghini project is above all a very open and collaborative project.

Here in this page you find some template letters, please use them as you think best. Adapt them to welcome new users, translate into your language print it in a nice format, and go with it to the garden next door, or on your way from A to B.

Discuss them with a group of your local friends, and if you can use them to spread the use of Ghini, to make the Ghini users community grow, but also to create a (voluntary, or paid) part-time job for you, then they have reached their ideal purpose.

7.2.1 Open letter to the garden of your choice

Dear conservator or scientist,

You are reading Ghini's presentation letter. Ghini is a libre software project on GitHub for botany. Brought to you by a small community of coders, botanists, translators, and supported by botanical institutions around the world, for all their collection management needs.

As a complete software solution of standalone programs, data servers and handheld clients, Ghini is dedicated to helping gardens manage and publish their collection:
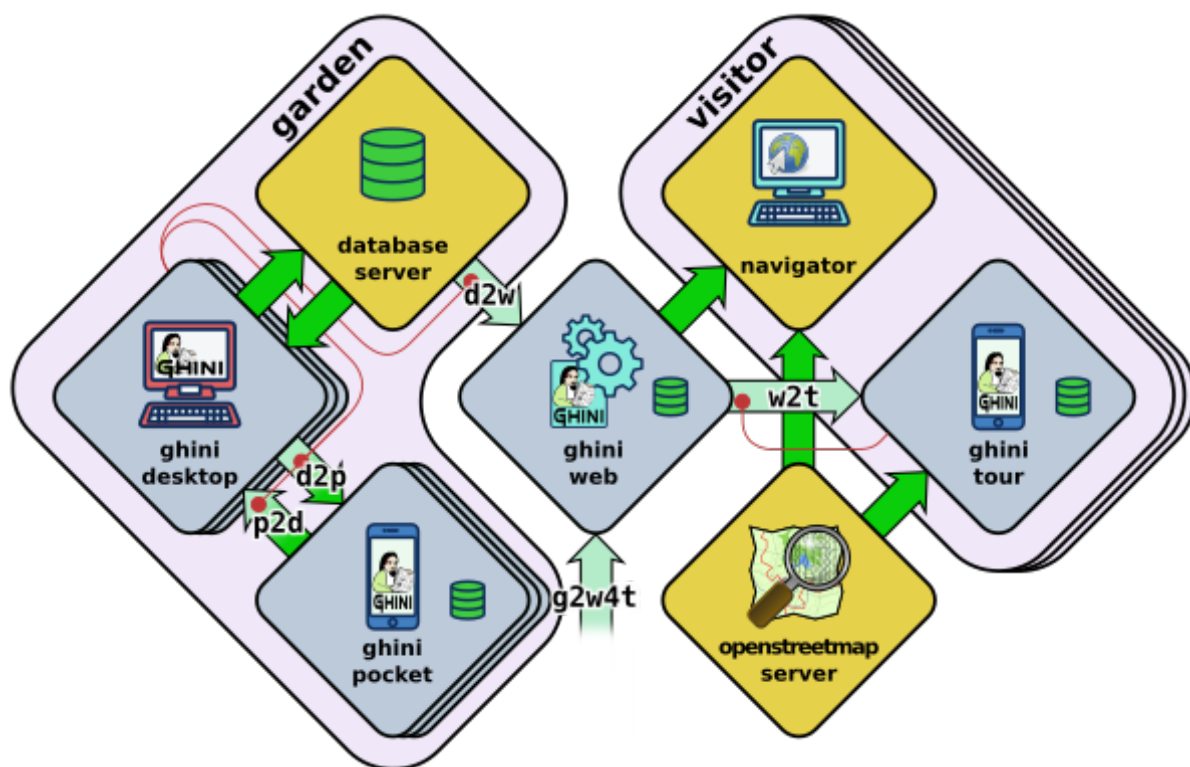
- **ghini.desktop**, is the core of Ghini, and it allows entering and correcting data, navigating links, producing reports, import and or exporting several standard or ad-hoc formats, reviewing your taxonomy using online sources. All according to best practices suggested by top gardens and formalized in standard formats, like ABCD or ITF2, but also as elaborated by our developers, based on the feedback by Ghini users. Developed and continously tested on GNU/Linux, but runs equally well on Windows, or macOS. [1]

- **ghini.pocket** is your garden companion within Ghini. An Android app you can install from F-Droid or the Play Store, it assists you in collecting or correcting data while in the field, associating pictures to your plants, and verifying taxonomic info. Back to office, you just import your collected data into the desktop client. **ghini.pocket** reduces the time spent in front of your desktop PC to a true minimum.

- **ghini.web** is a web server and a courtesy data hub service, offering you world wide visibility: Export a selection of your data from your desktop database, and offer it for publication to the Ghini project, for inclusion on http://gardens.ghini.me/. **ghini.web** serves a world map with participating gardens, georeferenced plants and info panels.

- **ghini.tour**, a geographic tour Android app aimed at visitors, using OpenStreetMap as a base map, retrieving its data, gardens and virtual panels, from the web data aggregator `ghini.web`.

cut-cut-cut

- *The remainder is even more a template than the above, please cut and reduce, but make sure that you convey the below message, which is about feedom, sharing, community forming.*

- *You also want to add a few lines about yourself, or you can keep it to the talk you hope to have with the conservator, scientific director, or garden manager once you get invited.*

All software within the Ghini family is licensed according to the GNU Public License v2+, v3+. The GPL is one of the strongest copyleft licenses. In short, the GPL translates the ethical scientific need to share knowledge, into legal terms. If you want to read more about it, please refer to https://www.gnu.org/licenses/copyleft.html

Ghini's idea about knowledge and software ownership is that software is procedural knowledge and as such, should be made a "commons": With software as a commons, "libre software" and more specifically "Copylefted software", you not only get the source code, you receive the right to adapt it, and the invitation to study and learn from it, and to share it, both share forward to colleagues, and share back to the source. With proprietary software, you are buying your own ignorance, and with that, your dependency.

This fancy term "copyleft" instead of just "libre software", means the software you received is libre software with one extra freedom, guaranteeing every right you were granted upon receiving the software is never lost.

In short: you can consider any GPL licensed software as your own, but **not** your *exclusive* own.

With copylefted software you are free —actually welcome— to employ local software developers in your neighbourhood to alter the software according to your needs, please do this on GitHub, fork the code, develop just as openly as the common practice within Ghini, and whenever you want, open a pull request so your edits can be considered for inclusion in the main branch. Ghini is mostly continuously unit tested, so before your code is added to the main branch, it should follow our quality guidelines for contributions. With libre software you acquire freedom and contribute to it, something that earns you visibility: Your additions stays yours, you share them back to the community, and will see them completed and made better by others. Having your code added to the main branch simplifies your upgrade procedure.

You can also contribute to the software by helping translate it into your native language. [5]

Some videos are published on YouTube, highlighting some of the software capabilities. [6]

Share back with the community. Several developers have spent cumulatively many thousand hours developing this software, and we're sharing with the community. We hope by this to stimulate a community sentiment in whoever starts using what we have produced.

In case you're interested in publishing your tree collection on the net, I would be happy to include your plants, species, coordinates to http://gardens.ghini.me. Georeferenced textual information panels are also very welcome, all offered as a courtesy: We're still defining the offer. The idea behind this is allowing visitors to explore aggregated garden collections, and the current focus is on trees.

A small example: http://gardens.ghini.me/#garden=Jardín%20el%20Cuchubo

cut-cut-cut

Thanks for your consideration; please let me know if you have any questions,

Mario Frasca MSc

[1] http://ghini.readthedocs.io/ - http://ghini.github.io/

[2] https://play.google.com/store/apps/details?id=me.ghini.pocket

[3] http://gardens.ghini.me/

[4] https://play.google.com/store/apps/details?id=me.ghini.tour

[5] https://hosted.weblate.org/projects/ghini/#languages

[6] https://www.youtube.com/playlist?list=PLtYRCnAxpinU_8WEDuRlgsYnNVe4J_4kv

7.2.2 free botanic data management systems

Many institutions still consider software an investment, an asset that is not to be shared with others, as if it was some economic good that can't be duplicated, like gold, or cryptocurrencies.

As of right now, very few copylefted programs exist for botanic data management:

- `ghini.desktop`, born as `bauble.classic` and made a commons by the Belize Botanical Garden. Next to the bauble-derived `desktop` program, Ghini offers three more components: a `pocket` data collecting Android app, a Node.js `web` server, aggregating data from different gardens and presenting it geographically, finally a geographic `tour` app aimed at visitors using the web data aggregator as its data source. You can find every Ghini component on GitHub: http://github.com/Ghini

- Specify 6 and 7, made a Commons by the Kansas University. A bit complex to set up, very difficult to configure and tricky to update. The institutions I've met who tried it, only the bigger ones, with in-house software management capabilities manage to successfully use it. They use it for very large collections (millions of elements). Specify targets naturaly history rather than living collections, but in this it is extremely generic: it adapts to herbaria, seed collections, but also to collections of eggs, organic material, fossils, preserved dead animals, possibly even viruses, I'm not sure. It is this extreme flexibility that makes its configuration a bit of a complex task. Specify is also on GitHub: https://github.com/specify and is licensed as GPLv2+. Support is provided to US institutions.

- Botalista, a French/Swiss cooperation, is GPL as far as rumours go. Its development has yet to go public. It was most active in early 2016, by end 2018 it seems to have faded away.

- `bauble.web` is an experimental web server by the author of `bauble.classic`. `bauble.classic` has been included into Ghini, to become `ghini.desktop`. Bauble uses a very permissive license, making it libre, but not copylefted. As much as 50% of bauble.web and possibly 30% of ghini.desktop is shared between the two projects. Bauble seems to be stagnating, and has not yet reached a production-ready stage.

- `Taxasoft-BG`, by Eric Gouda, a Dutch botanist, specialist in Bromeliaceae, collection manager at the Utrecht botanical garden. Taxasoft-BG is a typical LAMP (Linux-Apache-MySql-Php) program. It was Mario Frasca who convinced Eric to publish what he is doing, to license it under the AGPL, and to publish as far as possible the most current version of the program. Not yet "production ready", you find it on github: https://github.com/Taxasoft/Taxasoft-BG and https://github.com/Ejgouda/Taxasoft-BG

- `BG-Recorder`, by the BGCI, runs on Windows, and requires Access. Developed mostly between 1997 and 2003, it has not been maintained ever since and isn't actively distributed by the BGCI. I've not managed to find a download link nor its license statement. It is still mentioned as *the free option* for botanic database management.

Of the above, only `ghini.desktop` satisfies these conditions: Copylefted, available, documented, maintained, easy to install and configure. Moreover: Cross platform and internationalized.

7.2.3 Welcome to Ghini/Bauble

Dear new user,

Welcome to Ghini/Bauble.

As the maintainer, I have received your registration for bauble.classic/ghini.desktop, many thanks for taking your time to fill in the form.

I see you are using bauble.classic-1.0.55, whereas 1.0.55 is the last released version of bauble.classic, however, bauble.classic is now unmaintained and superseded by the fully compatible, but slightly aesthetically different ghini.desktop. Install it following the instructions found at http://ghini.rtfd.io

The registration service says you're not yet using the newest Python2 version available. As of 2018-05-01, that is 2.7.15. Using any older version does not necessitate problems, but in case anything strange happens, please update your Python (and PyGTK) before reporting any errors.

Also thank you for enabling the "sentry" errors and warnings handler. With that enabled, Ghini/Bauble will send any error or warning you might encounter to a central server, where a developer will be able to examine it. If the warning was caused by an error in the software, its solution will be present in a subsequent release of the software

If you haven't already, to enable the sentry and warnings handler, open the ":config" page in Ghini and double click on the row "bauble.use_sentry_client".

I hope Ghini already matches your expectations, if this is not the case, the whole Ghini community would be very thankful if you took the time to report your experience with it.

The above is one way to contribute to Ghini's development. Others are: - contribute ideas, writing on the bauble google forum (https://groups.google.com/forum/#!forum/bauble), - contribute documentation, or translations (https://hosted.weblate.org/projects/ghini/), - give private feedback, writing to ghini@anche.no, - rate and discuss Ghini openly, and promote its adoption by other institutions, - open an issue on GitHub (https://github.com/Ghini/ghini.desktop/issues/), - contribute code on GitHub (fork the project on (https://github.com/Ghini/ghini.desktop/), - hire a developer and have a set of GitHub issues solved, per-haps your own - let me include your garden on the still experimental worldmap (http://gardens.ghini.me)

I sincerely hope you will enjoy using this copylefted, libre software

Best regards, Mario Frasca

https://ghini.github.io https://github.com/Ghini/ghini.desktop/issues/

7.2.4 Do you want to join Ghini?

Note: I generally send a note similar to the following, to GitHub members who "star" the project, or to WebLate contributors doing more than one line, and at different occasions. If it's from GitHub, and if they stated their geographic location in their profile, I alter the letter by first looking on institutos botánicos if there's any relevant garden in their neighbourhood.

Dear GitHub member, student, colleague, translator, botanist,

Thank you warmly for your interest in the Ghini project!

From your on-line profile on github, I see you're located in Xxxx, is that correct?

If you are indeed in Xxxx, you live very close to gardens Yyyy and Zzzz. Maybe you would consider the following proposition? All would start by contacting the botanical garden there, and get to know what software they use (what it offers, and at which price) and if they're interested in switching to ghini.desktop+pocket+tour+web.

The business model within Ghini is that the software is free and you get it for free, but time is precious and if a garden needs help, they should be ready to contribute. Maybe you already have a full-time job and don't need more things to do, but in case you're interested, or you have friends who would be, I'm sure we can work something out.

Let me know where you stand.

best regards, and again thanks for all your contributed translations.

Mario Frasca

7.2.5 Letter to garden network

good morning,

I wish to establish contact with people within the (eg: council of botanic gardens), who could be open for examining a proposal. I am the current maintainer of the free software Ghini, for the management of botanic collections. Could you introduce me to one of the gardens in your area, that are looking for a strong data management system, even ones with more time than money? My advantage is increased feedback (and a Russian translation). This evaluation of the software can be expanded to further help develop it, for your and our use.

The Ghini software is software made into a commons, meaning you can freely copy it, learn from it, and you may enhance it, but you can't appropriate it. This is different from closed software, where copying is not just discouraged but strictly prohibited.

7.2.6 Letter to botanic institution developing proprietary software

Dear Peter, John, James, good morning. (names of your contact people)

this is (you), of the Ghini Organization (maybe you want to get in touch with me before you say this), (introduce how you know each other, you possibly visited them, whatever). I'm not

a botanist (or are you), (myself: a mathematician specializing in information science), and as Peter already knows I endorse the view of software as a commons, rather than just an economic asset. I am trying to establish a communication line with your institution, and get in touch with the licensing devision makers of the software you develop. It would be a big advantage to all, if it was copylefted, and published. I would be happy to contribute to its development.

Unfortunately most institutions consider software an investment, not to be shared with others, as if it was an economic good that can't be duplicated, like gold. As of now, I am aware of the existence of very few copylefted programs for botanic data management:

- ghini.desktop, born as bauble.classic and made a Commons by the Belize Botanical Garden. (more recently Bauble regressed to 2008, chose for a different license, which makes it free but not copylefted, and it is again stagnating). ghini.desktop has three more components, a pocket data collecting Android app, a nodeJS web server, aggregating data from different gardens, and presenting it geographically, again a geographic tour app aimed at visitors and using the web data aggregator as its data source. Every Ghini component is on GitHub: http://github.com/Ghini

- specify 6 and 7, made a Commons by the Kansas University. A bit complex to set up, and very difficult to configure, tricky to update. the institutions I've met who tried it, only the bigger ones, with in-house software management capabilities manage to successfully use it. Specify is also on GitHub: https://github.com/specify

- botalista, a French/Swiss cooperation, is GPL as far as I know, but it wasn't yet released for public download. very active around 2016, now stagnating.

- Taxasoft-BG, from Eric Gouda, a Dutch botanist, specialist in Bromeliaceae, collection manager at the Utrecht botanical garden. it was me who convinced him to publish what he was doing, and to publish it under the GPL, but I'm afraid he never posted any update of what he's been doing after 2016, April 13th and I'm also afraid he forgot to clearly specify the license. I know that he actively works at it and at its distribution. You can find it on GitHub: https://github.com/Ejgouda/Taxasoft-BG

Of the above, only ghini.desktop satisfies the conditions: Copylefted, available, documented, maintained, easy to install and configure. moreover: Cross platform, internationalized.

Maybe you can add yourself to the list? Or your decision makers could even add your own system? Peter, what's your opinion in this?

Friendly regards,

Mario Frasca

CHAPTER 8

Supporting Ghini

If you're using Ghini, or if you feel like helping its development anyway, please consider donating