Ghini Documentation

Version 1.0.x

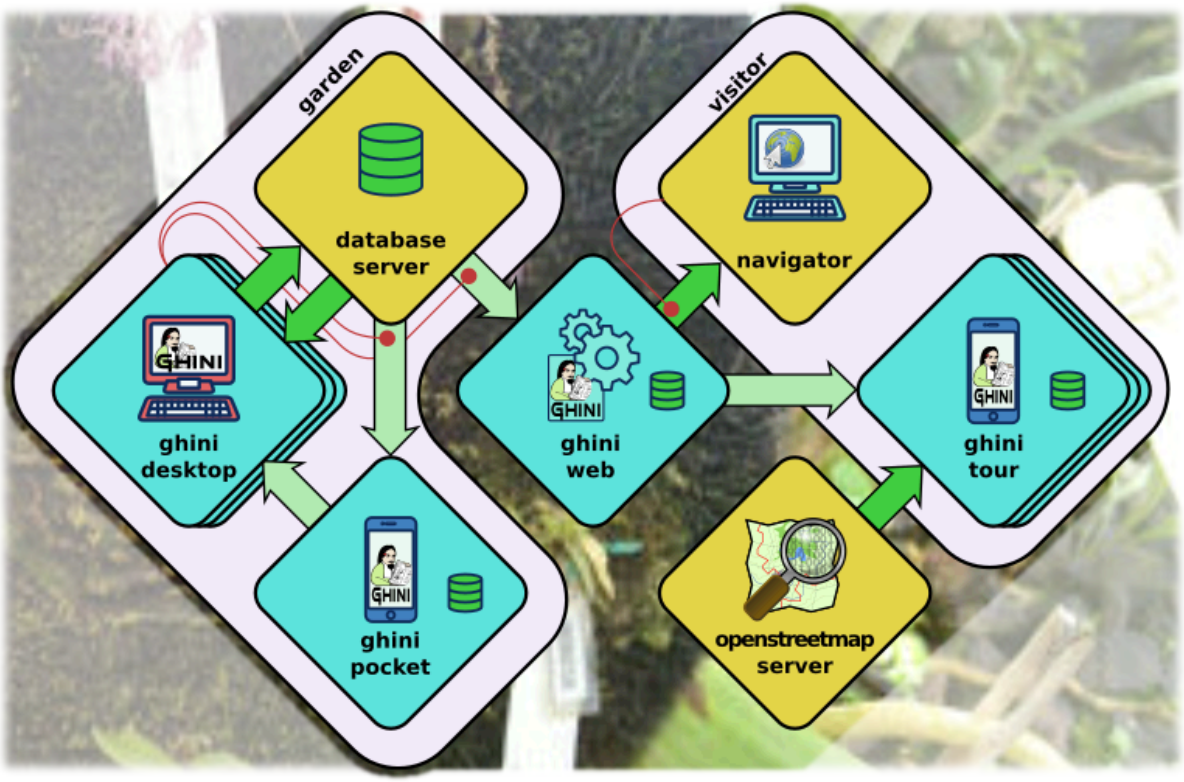
Mario Frasca

janvier 11, 2023

Table des matières

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 software 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.

CHAPITRE 1

Statements

1.1 Objectifs et les points culminants de Ghini

Ce logiciel, est-il ça que vouz cherchait ? Cette question est pour vous à répondre. Nous sommes persuadés que si vous gérez une collection botanique, vous y trouverez Ghini trop utile et nous espérons que cette page vous convaincra à ce sujet.

Cette page montre comment Ghini rend les logiciel à répondre aux besoins d'un jardin botanique.

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 Jardin Botanique

Selon Wikipedia, » un botanic(al) jardin est un jardin dédié à la collection, la culture et l'affichage d'une large gamme de plantes étiquetées avec leurs noms botaniques « et toujours selon Wikipedia, » un jardin est un espace prévu, habituellement à l'extérieur, mis de côté pour l'affichage, culture et plaisir de plantes et d'autres formes de la nature. «

Nous avons donc dans un jardin botanique les deux l'espace physique, le jardin, comme sa dynamique, les activités à laquelle est dédié le jardin, activités qui nous fait appellent le jardin un jardin botanique.

1.1.2 Logiciel jardin botanique

À l'autre bout de notre raisonnement, nous avons l'application du programme Ghini et citant à nouveau la Wikipedia, » un programme d'application est un programme informatique conçu

Fig. 1 – **le jardin physique**

Fig. 2 – **collection des activités connexes dans le jardin**

pour exécuter un groupe de fonctions coordonnées, des tâches ou activités au profit de l'utilisateur «, ou, en bref, » conçu pour aider les gens à accomplir une activité «.

Données et algorithmes au sein de Ghini ont été conçus pour représenter l'espace physique et la dynamique d'un jardin botanique.

Fig. 3 – **structure de la base de base de données de Ghini**

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**.

L'élément central dans la perspective de Ghini est la `Accession`. Suivant ses liens à d'autres objets de base de données nous permet de mieux comprendre la 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.

Une `Accession` reste dans la base de données, même si toutes ses `Plantings` ont été enlevés, vendus, ou sont morts. Avec son lien à son `Species`, une `Accession` connecte touts ses `Plantings` à la `Species`.

Accession at the base of the history of your plants

`Propagations` et `Contacts` fournissent des matières végétales pour le jardin ; Cette information est facultative et petits collectionneurs préfèrent laisser cela de côté. Un procès `Propagation` peut être infructueux, la plupart du temps qu'elle se traduira par une accès, mais il peut aussi produire légèrement différents taxons, donc la base permet de zéro ou plusieurs `Accession` par `Propagation` (0.. n). Également un `Contact` peut fournir zéro ou plusieurs `Accessions` (0.. n).

Accession and Verification opinions

Spécialistes peuvent formuler leurs avis sur la `Species` auquel appartient une `Accession`, en fournissant une `Verification`, en signant et en indiquant le niveau de confiance applicable.

Accessing your own Propagations

Si une `Accession` a été obtenue dans la pépinière du jardin d'une succès `Propagation`, les liens de `Propagation` la `Accession` et tous ses `Plantings` à un seul parent `Planting`, la graine ou le parent végétatif.

Même après les explications qui précèdent, nouveaux utilisateurs demandent généralement encore pourquoi ils ont besoin de passer par un écran `Accession` alors que tout ce qu'ils veulent est d'insérer une `Planting` dans la collection et encore une fois : quelle est cette « accession » chose quand même ? La plupart des discussions sur le net ne font pas le concept plus clair. Un de nos utilisateurs a donné un exemple dont je suis heureux d'inclure dans la documentation de Ghini.

cas d'utilisation

1. Au début de 2007, nous avons obtenu cinq plantules de * Heliconia longa * (une plante `Species`) de notre voisin (la source de `Contact`). Puisque c'était la première acquisition de l'année, nous avons nommé les 2007.0001 (nous leur avons donné un code unique unique `Accession`, avec quantité 5) et nous avons planté eux tous ensemble à un `Location` comme un `Planting` unique, aussi avec quantité 5.
2. Au moment de l'écriture, neuf ans plus tard, `Accession` 2007.0001 a 6 distinctes `Planting`, chacun à un `Location` différent dans notre jardin, obtenu par voie végétative (asexuée) des 5 plantes originales. Notre seule intervention était fractionner, déplacer et bien sûr écrire ces informations dans la base de données. La quantité totale de la plante est supérieur à 40.
3. Nouvelles `Planting` obtenues par `Propagation` sexuelle (assistée) viennent dans notre base de données sous différents codes de `Accession`, où notre jardin est la source de `Contact` et où nous savons lesquels de nos `Plantings` est le parent de la graine.

les trois cas ci-dessus se traduisent plusieurs histoires courtes utilisation :

1. activer le menu Insertion → accès, vérifier l'existence et l'exactitude de la `Species` * Heliconia longa *, spécifiez la quantité initiale de l'`Accession` ; ajouter son `Planting` à la `Location` souhaité.
2. modifier `Planting` pour corriger la quantité de plantes vivantes — répète aussi souvent que nécessaire.
3. modifier `Planting` pour diviser les `Location` distincts — ce qui produit un `Planting` différente sous la même `Accession`.
4. modifier `Planting` pour ajouter une `Propagation` (graine).
5. modifier `Planting` pour mettre à jour l'état de la `Propagation`.
6. activer le menu Insertion → accès à associer une accès à un essai réussi de `Propagation`; Ajouter le `Planting` à la `Location` souhaité.

En particulier la possibilité de scinder une `Planting` plusieurs différents `Location` et de garder tous les uniformément associée à une `Species`, ou la possibilité de garder des informations sur les `Plantings` qui ont été supprimées de la collection, aider à justifier la présence de la niveau d'abstraction `Accession`.

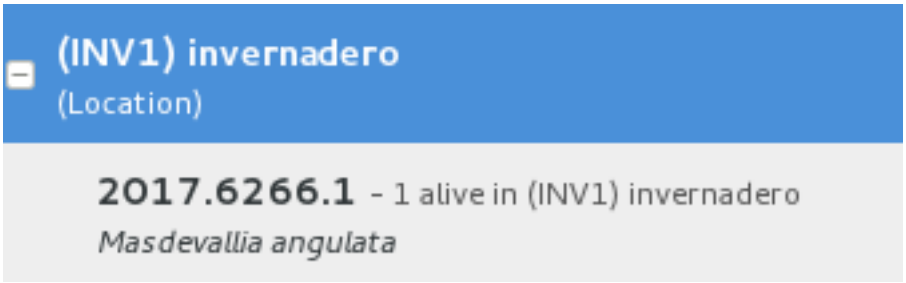
1.1.3 Hypersimplified view

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

non-ainsi-brève liste des faits saillants, vise à vous mettre en appétit.

informations taxonomiques

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.

Importation de données

Ghini vous permet d'importer toutes les données que vous mettre dans un format intermédiaire json. Ce que vous importez viendra compléter ce que vous avez déjà dans la base de données. Si vous avez besoin d'aide, vous pouvez demander certains professionnel Ghini pour vous aider à transformer vos données au format json intermédiaire de Ghini.

synonyms

Ghini permettra de que vous définir des synonymes pour les espèces, des genres, des familles. Aussi, cette information peut être représentée dans son format json intermédiaire et être importée dans une base de données existante de Ghini.

scientifique responsable

Ghini implémente le concept de « accès », intermédiaire entre les installations matérielles (ou un groupe) et abstraites de taxon. Chaque accès peut associer les mêmes plantes de taxons différents, si deux taxonomistes ne s'entendent pas sur l'identification : chaque taxonomiste peuvent avoir leur mot à dire et ne doivent pas remplacer le travail des uns et des autres. Toutes les vérifications se trouvent dans la base de données, avec la signature et l'horodatage.

aide identification hors ligne

Ghini permet de vous associer des photos à des installations physiques, cela peut aider à reconnaître la plante dans le cas où un autocollant est perdu, ou aide identification taxonomique si un taxonomiste n'est pas disponible en tout temps.

exportations et rapports

Ghini vous permet d'exporter un rapport dans n'importe quel format textuel dont vous avez besoin. Il utilise un moteur de templating puissant nommé « mako », qui permettra de que vous exportez les données dans une sélection à n'importe quel format dont vous avez besoin. Une fois installé, il existe quelques exemples dans le sous-répertoire de mako.

annoter vos infos

Vous pouvez associer des notes de plantes, des accès, des espèces… Notes peuvent être catégorisés et utilisés dans des recherches ou des rapports.

jardin ou herbier

Gestion des emplacements de la plante.

histoire de la base de données

Toutes les modifications dans la base de données est stockée dans la base de données, comme journal de l'histoire. Toutes les modifications sont « signées » et horodatées. Ghini, il est facile d'extraire la liste de tous les changements dans la journée de travail ou la semaine dernière, ou dans une période spécifique dans le passé.

Recherche simple et puissante

Ghini vous permet de rechercher la base de données en utilisant des mots-clés simples, par exemple : le nom de l'emplacement ou un nom de genre ou vous pouvez écrire des requêtes plus complexes, qui n'atteignent pas la complexité du SQL, mais vous permettent un niveau décent de localiser vos données en détail.

indépendant de la base de données

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.

indépendant du langage

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%.

indépendant de la plate-forme

Installation de Ghini sous Windows est un processus simple et linéaire, il ne faudra pas plu de 10 minutes. Ghini naquit sur Linux et l'installer sur ubuntu, fedora ou debian est par conséquent encore plus facile. MacOSX étant basé sur unix, il est possible d'exécuter avec succès la procédure d'installation de Linux sur n'importe quel ordinateur Apple récemment, après quelques étapes de préparation.

mis à jour facilement

Le processus d'installation produira une installation actualisable, où la mise à jour prendra moins d'une minute. Selon la quantité de commentaires que nous recevons, nous produirons des mises à jour tous les quelques jours ou une fois dans un certain temps.

Testé

Ghini est continuellement et unité testées, ce qui rend la régression de la fonctionnalité presque impossible. Chaque mise à jour est automatiquement qualité vérifiée, au service de l'intégration continue de Travis. Intégration de TravisCI avec la plateforme github rendra difficile pour que l'on puisse divulguer quoi que ce soit qui dispose d'une unité unique défaut d'essai.

La plupart des modifications et des ajouts nous faire, venu avec quelques essais unité supplémentaire, qui définit le comportement et fera tout changement intempestif facilement visible.

customizable/extensible

Ghini est extensible grâce à des plugins et peut être personnalisé pour répondre aux besoins de l'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

CHAPITRE 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. You may read the documentation later.

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 ?

Ghini is maintained by very few people, who focus on enhancing its functional parts, more than on writing fancy installers. Instead of several native installers we offer 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 PyGTK. 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.

2.1.1 Installing on GNU/Linux

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

1. Download the *devinstall.sh* script :
   devinstall.sh
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.

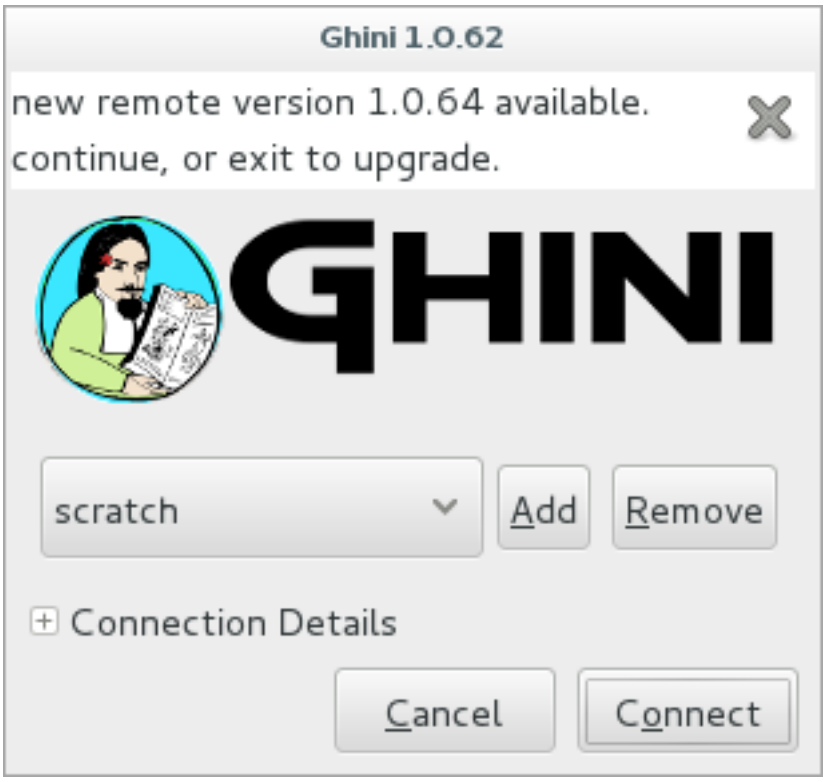
3. You can now start ghini by invoking the `ghini` script :

```
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 `1.1` if you want to help us with its development :

```
~/bin/ghini -s 1.1
```

---

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 `ghide`, use git to download the sources to a directory named `~/Local/github/Ghini/ghini.desktop`, and connect this git checkout to the `ghini-1.0` 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

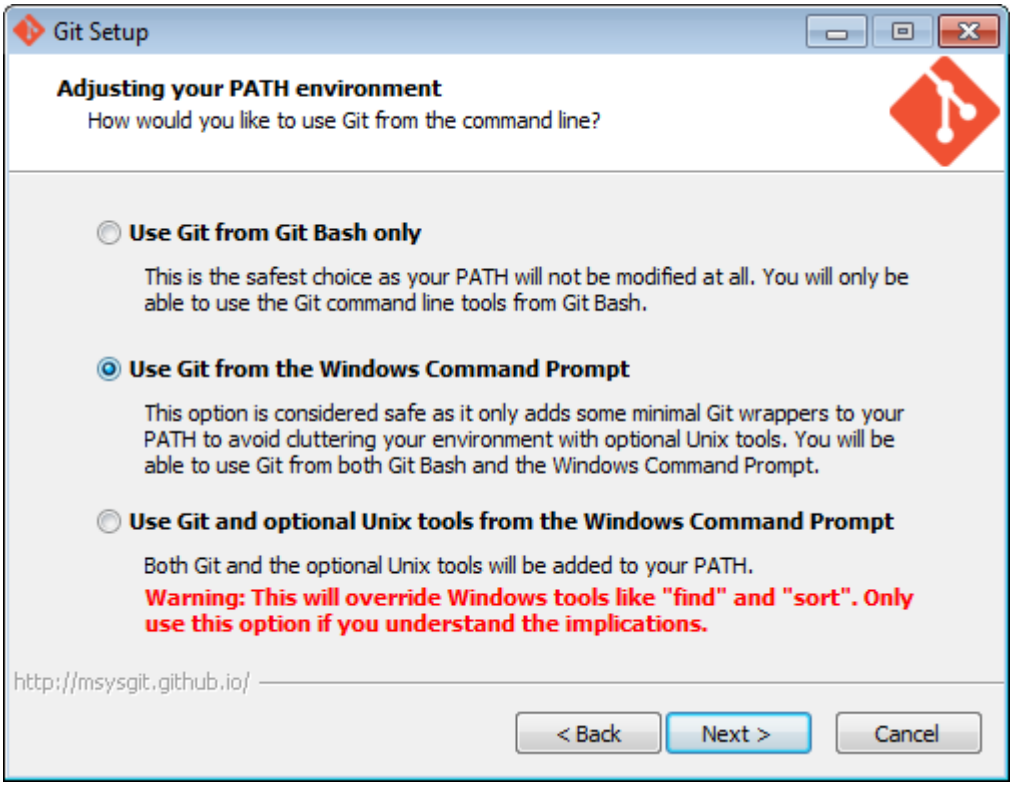
The steps described here instruct you on how to install Git, Gtk, Python, and the python database connectors. With this environment correctly set up, the Ghini installation procedure runs as on GNU/Linux. The concluding steps are again Windows specific.

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 :

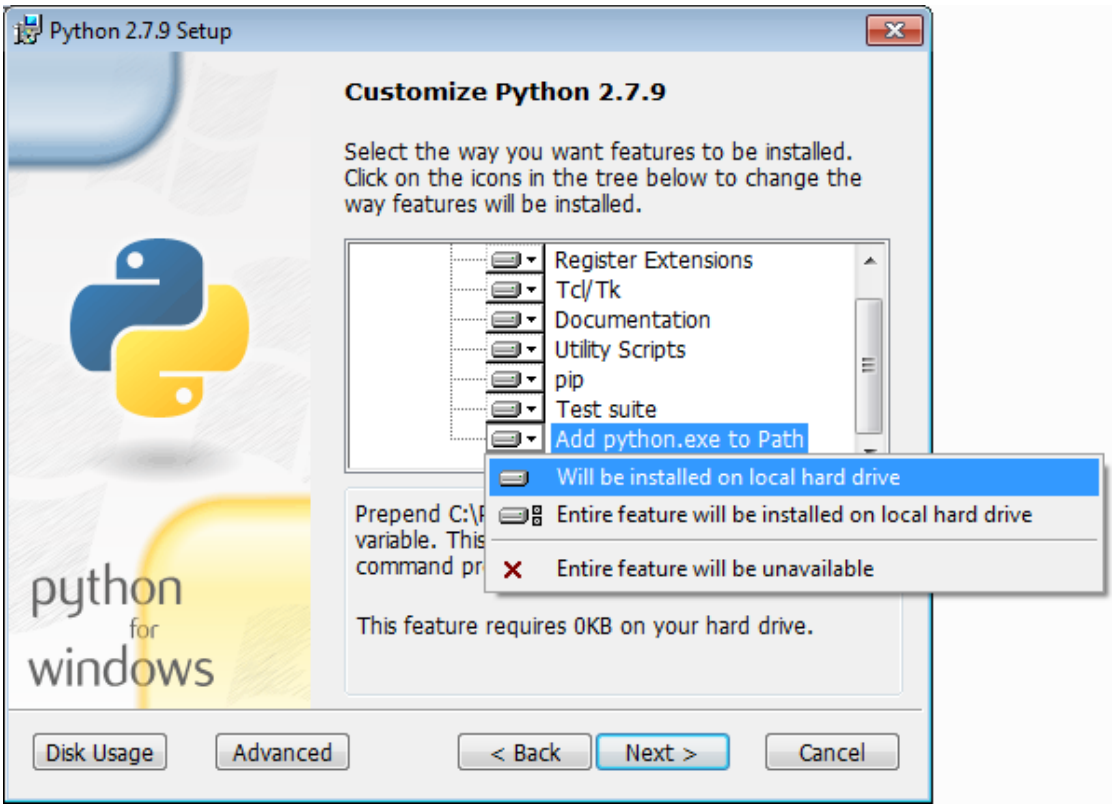
1. download and install `git` (comes with a unix-like `sh` and includes `vi`). Grab it from the Git download area.

   all default options are fine, except we need git to be executable from the command prompt :



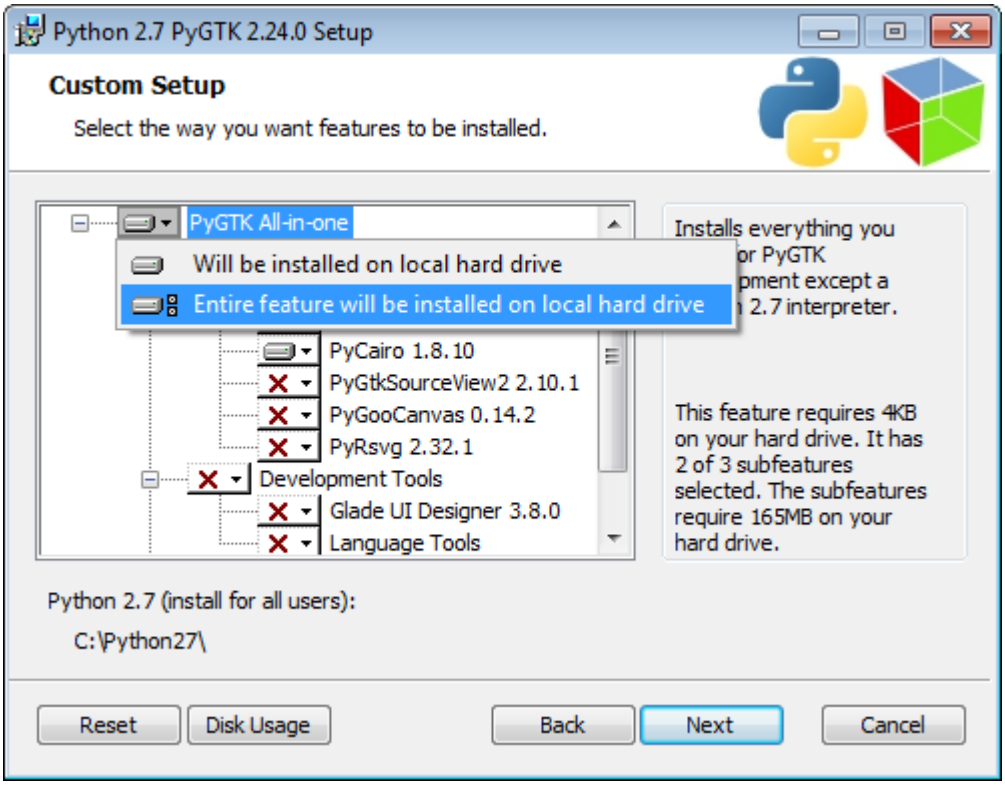
2. download and install Python 2.x (32bit). Grab it from the Python official site.

   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.

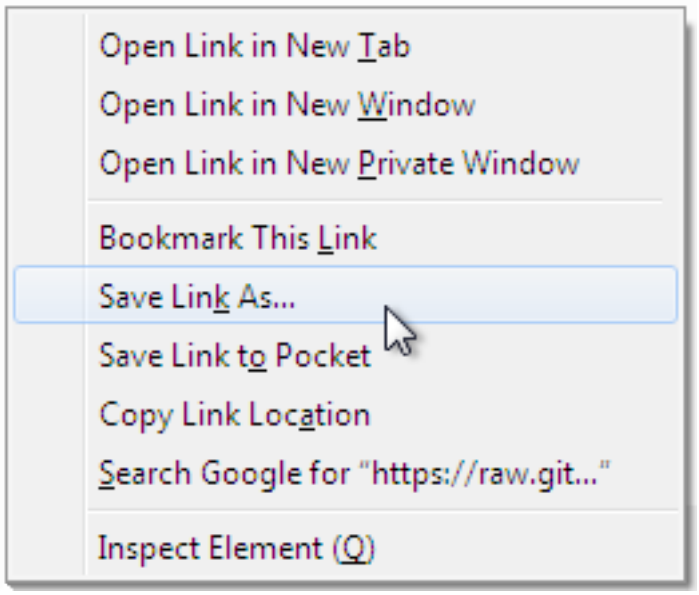
   Make a complete install, selecting everything :



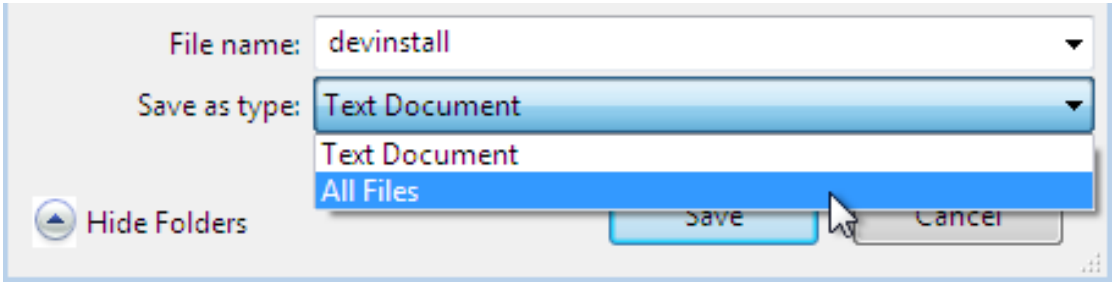
4. (Possibly necessary, maybe superfluous) install lxml, you can grab this from the pypi archives

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

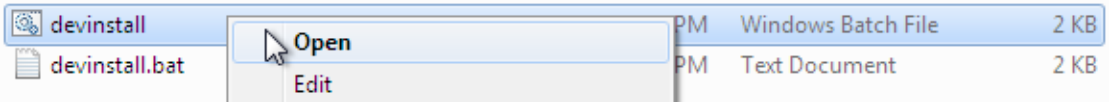
5. (definitely optional) download and install a database connector other than `sqlite3`.
   If you plan using PostgreSQL, the best Windows binary library for Python is psycopg and is Made in Italy.
6. **REBOOT**
   hey, this is Windows, you need to reboot for changes to take effect !
7. We're done with 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...



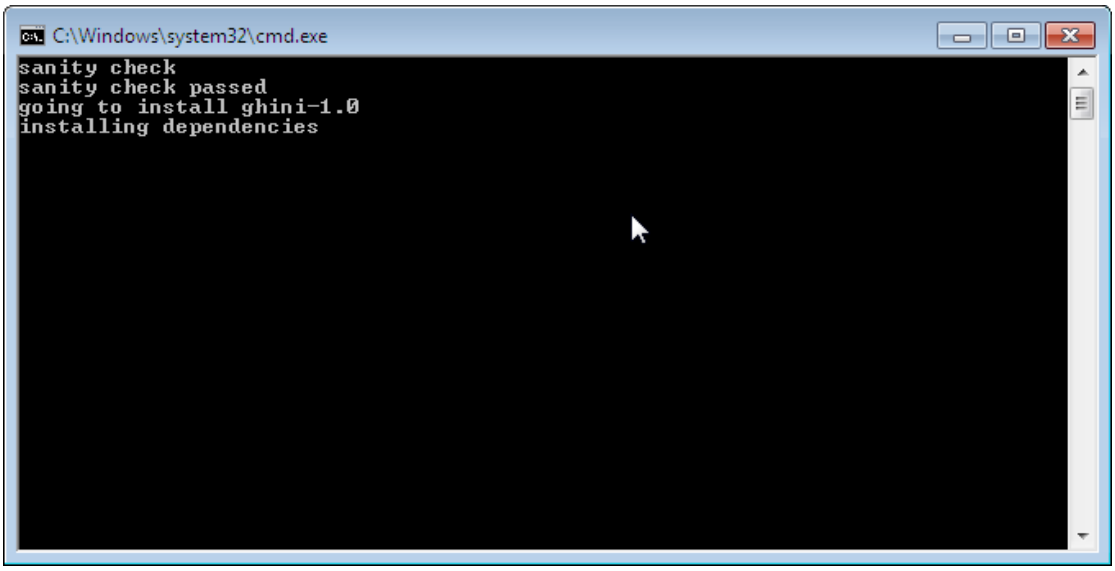
Also make sure you don't let Windows convert the script to a text document.



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.



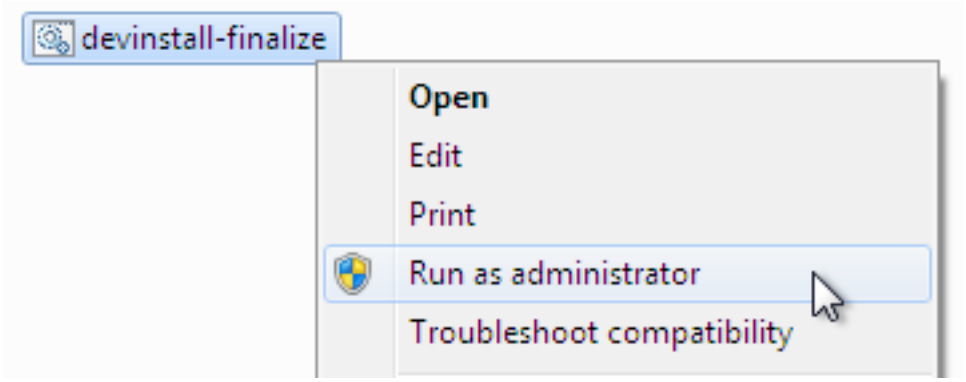
Running `devinstall.bat` will pull the `ghini.desktop` repository from github to your home directory, under `Local\github\Ghini`, checkout the `ghini-1.0` production line, create a virtual environment and install ghini into it.

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 need to generate PDF reports, you can use the XLS based report generator and 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.

If you choose for PostScript reports, you can use the Mako based report generator and there are no further dependencies.

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.

CHAPITRE 3

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 ?

Est-ce la première fois que vous utilisez Ghini, allez-vous travailler dans un environnement autonome, vous n'avez pas la moindre idée de comment gérer un système de gestion de base de données ? Si vous avez répondu oui à l'une des précédentes, vous feriez probablement mieux de vous en tenir à SQLite, la base de données de fichiers simple, rapide et sans administration.

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

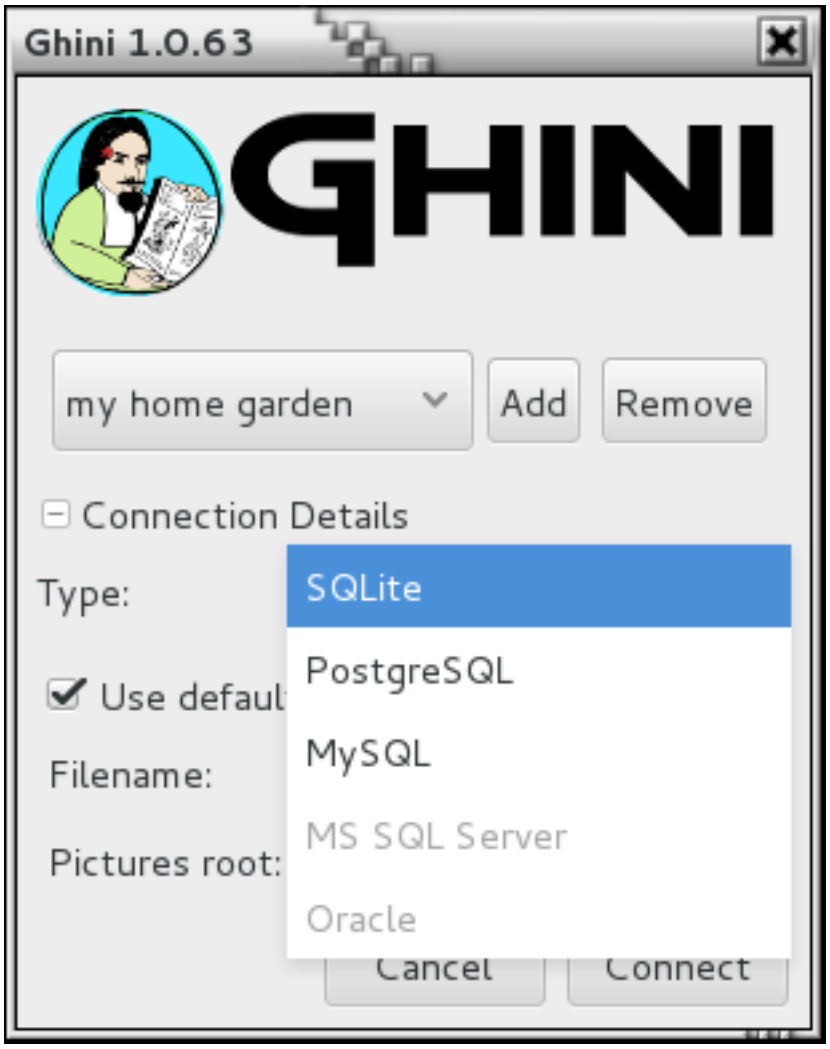
If you do not know what to do here, Ghini will help you stay safe. Activate the **Use default locations** check box and create your first connection by clicking on **Connect**.

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 *Modification et insertion de données* 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 *Modification et insertion de données*.

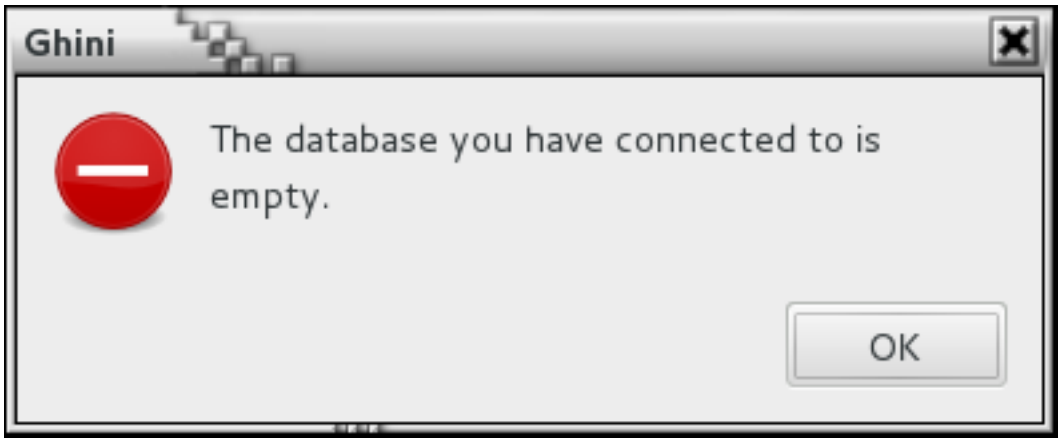
A sample SQLite database

Indeed we have a sample database, from our pilot garden « El Cuchubo », in Mompox, Colombia. We have a zipped sample database for ghini-1.0.

Download and 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 *Modification et insertion de données* 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 – **structure de la base de base de données de Ghini**

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 :

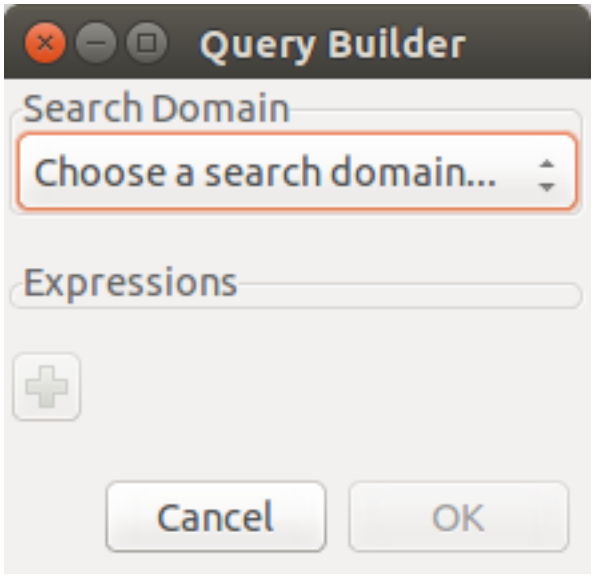
```
accession WHERE _last_updated>|datetime|-3|
```

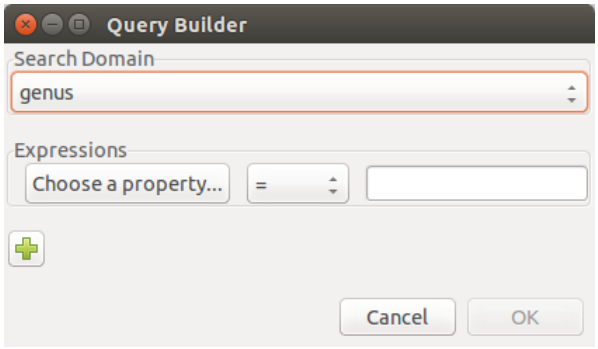
Searching with queries requires some knowledge of a little syntax and an idea of the extensive Ghini database table structure. Both you acquire with practice, and with the help of the Query Builder.

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 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
                | '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
```

(suite sur la page suivante)

(suite de la page précédente)

```
        | 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 )
```

(suite sur la page suivante)

(suite de la page précédente)

```
string_value = quoted_string | unquoted_string ;

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

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

3.3 Modification et insertion de données

Le principal moyen que nous ajouter ou modifier des informations dans Ghini est en utilisant les éditeurs. Chaque type de base de données possède son propre éditeur. Par exemple, il y a un rédacteur en chef de famille, un éditeur de genre, un éditeur d'accès, etc..

Pour créer un nouveau clic record sur le : menuselection : « Insert » menu dans la barre de menus, puis sélectionnez le type d'enregistrement vos souhaitez créer. Cela ouvre un nouvel éditeur de vide pour le type.

Pour modifier un enregistrement existant dans la base de données faites un clic droit sur un élément dans les résultats de recherche et sélectionnez : menuselection : « Modifier » dans le menu contextuel. Cela ouvre un éditeur qui permet de modifier les valeurs dans le dossier que vous avez sélectionné.

La plupart des types ont aussi des enfants que vous pouvez ajouter en un clic droit sur le parent et en sélectionnant « Add ? ? ? . . . » dans le menu contextuel. Par exemple, une famille a genre enfants : vous pouvez ajouter un genre à une famille en un clic droit sur une famille et en sélectionnant « Add genus ».

3.3.1 Notes

Presque tous les éditeurs à Ghini ont une * Notes * onglet qui devrait fonctionner de la même indépendamment de quel éditeur Utilisez-vous.

Si vous entrez une adresse web dans une note, le lien apparaît dans les liens boîte lorsque la question vous êtes édition est sélectionné dans les résultats de recherche.

Vous pouvez consulter les notes d'un élément dans la base de données à l'aide de la boîte de Notes au bas de l'écran. La boîte de Notes est désensibilisée si l'élément sélectionné ne possède

pas toutes les notes.

3.3.2 Famille

Le rédacteur en chef de famille vous permet d'ajouter ou de modifier une famille botanique.

La * famille * champ sur l'éditeur vous permet de changer l'épithète de la famille. Le champ de la famille est obligatoire.

Le * qualification * champ vous permet de changer le qualificatif de famille. La valeur peut être * sensu lato *, * sensu stricto *, ou rien.

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 selected 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.

Pour annuler vos modifications sans enregistrer les modifications, puis cliquez sur la * bouton de Cancel.

Pour sauver la famille vous travaillez, puis cliquez sur * OK *.

Pour sauver la famille, vous travaillez sur et y ajouter le genre puis cliquez sur le * genres * Ajouter bouton.

Pour ajouter une autre famille lorsque vous avez terminé celui en cours d'édition cliquez sur le * suivant * le bouton sur le bas. Ceci enregistre la famille actuelle et ouvre un nouvel éditeur de famille vide.

3.3.3 Genre

L'éditeur de genre vous permet d'ajouter ou de modifier un genre botanique.

La * famille * champ sur l'éditeur de genre vous permet de choisir la famille pour le genre. Lorsque vous commencez le type un patronyme il montrera une liste des familles à choisir. Le nom de famille doit déjà exister dans la base de données avant que vous pouvez le définir comme la famille pour le genre.

Le * genre * champ permet de définir le genre pour cette entrée.

Le * auteur * champ permet de définir le nom ou l'abréviation du ou des auteurs du genre.

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 selected 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.

Pour annuler vos modifications sans enregistrer les modifications, puis cliquez sur la * bouton de Cancel.

Pour sauver le genre vous travaillez, puis cliquez sur * OK *.

Pour sauver le genre vous travaillez sur et inscrire une espèce à elle, puis cliquez sur le * espèces * Ajouter bouton.

Pour ajouter un autre genre, lorsque vous avez fini de modifier le cours d'un clic sur le * suivant * le bouton sur le bas. Ceci enregistrer le genre actuel et ouvrir un nouvel éditeur de genre vide.

3.3.4 Species/Taxon

Pour des raisons historiques, appelées une « espèce », mais par là nous entendons un « taxon » à classer « espèce » ou plus bas. Il représente un nom unique dans la base de données. L'éditeur d'espèces permet de construire le nom mais aussi associer le taxon tels que sa distribution, de synonymes et autres informations de métadonnées.

Le * infraspécifique pièces * dans l'éditeur d'espèces permet d'indiquer le « taxon » plus loin au rang de « espèce ».

Pour annuler vos modifications sans enregistrer les modifications, puis cliquez sur la * bouton de Cancel.

Pour sauver l'espèce vous travaillez, puis cliquez sur * OK *.

Pour sauver l'espèce, vous travaillez sur et ajoutez une accès à celle-ci, puis cliquez sur le *ajouter accès* bouton.

Pour ajouter une autre espèce, lorsque vous avez fini de modifier le cours d'un clic sur le * suivant * le bouton sur le bas. Cela va sauver l'espèce actuelle et ouvrir un nouvel éditeur d'espèces vide.

3.3.5 Accessions

L'éditeur d'accès nous permet d'ajouter une accès à une espèce. Pour Ghini une accès représente . . .

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.

Source de l'accès

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 Plante

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

Vous pouvez créer plusieurs usines en utilisant des gammes dans l'entrée de code. C'est seulement lors de la création de nouvelles usines et il n'est pas possible lors de la modification des installations existantes dans la base de données.

Par exemple la plage, 3-5 créera plante avec code 3,4,5. La gamme 1,4-7,25 créera des plantes avec les codes 1,4,5,6,7,25.

Lorsque vous entrez la plage à l'entrée de code usine l'entrée s'allume bleue pour indiquer que vous créez maintenant plusieurs usines. Tous les champs qui sont définis dans ce mode seront copiés dans toutes les plantes qui sont créés.

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.

N'oubliez pas que vous avez configuré le dossier racine de photos lorsque vous avez spécifié les détails de votre connexion à la base. Encore une fois, vous devez vous assurer que l'images du dossier racine est partagé avec votre service de choix de partage de fichiers.

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-1.0, 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

L'éditeur de carte

danger zone

L'éditeur emplacement contient une section initialement cachée nommée * danger zone *. Les widgets contenues dans cette section permettent à l'utilisateur de fusionner l'emplacement actuel dans un emplacement différent, laissant l'utilisateur de corriger les fautes d'orthographe ou de mettre en œuvre des changements de politique.

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 (open them in your spreadsheet editor of choice), as labels (to be printed or engraved), as html pages or pdf or postscript documents.

3.6.1 The Report Tool

You activate the Report Tool from the main menu : *Tools→Report*. The Report Tools acts on a selection, so first select something, then start the Report Tool.

Report on the whole collection.

To produce a report on your whole plant collection, a shortcut would be from the home screen, to 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.

Reports are produced by a report engine, making use of a report template. Ghini relies upon two different report engines (Mako & XSL), and offers several report templates, meant as usable examples.

Choose the report you need, specify parameters if required, and produce the report. Ghini will open the report in the associated application.

Configuring report templates, that's a task for who installs and configures ghini at your institution. Basically, you create a template name, indicating the report engine and specifying the template. Configured templates are static, once configured you are not expected to alter them. Only the special `**scratch**` template can be modified on the fly.

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.

3.6.2 Using the Mako Report Formatter

The Mako report formatter uses the Mako template language for generating reports. More information about Mako and its language can be found at makotemplates.org.

The Mako templating system should already be installed on your computer if Ghini is installed.

Creating reports with Mako is similar in the way that you would create a web page from a template. It is much simpler than the XSL Formatter(see below) and should be relatively easy to create template for anyone with a little but of programming experience.

The template generator will use the same file extension as the template which should indicate the type of output the template with create. For example, to generate an HTML page from your template you should name the template something like *report.html*. If the template will generate a comma separated value file you should name the template *report.csv*.

The template will receive a variable called *values* which will contain the list of values in the current search.

The type of each value in *values* will be the same as the search domain used in the search query. For more information on search domains see search-domains.

If the query does not have a search domain then the values could all be of a different type and the Mako template should prepared to handle them.

3.6.3 Using the XSL Report Formatter

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

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 Importation et exportation des données

Although Ghini can be extended through plugins to support alternate import and export formats, by default it can only import and export comma separated values files or CSV.

Il y a un support pour exporter vers l'accès pour les données de Collections biologiques, il est limité.

Il y a également une prise en charge limitée pour l'exportation vers un format XML qui reflète plus ou moins exactement les tables et la ligne de la base de données.

Les renouvellements ne sont pas couverts.

> **Avertissement :** Importation de fichiers va très probablement détruire toute donnée que vous avez dans la base de données afin de s'assurer que vous avez sauvegardé vos données.

3.7.1 Importation de CSV

En général, il est préférable d'importer les fichiers CSV dans Ghini précédemment exportés de Ghini. Il est possible d'importer un fichier CSV, mais qui est plus avancé que ce doc couvrira.

To import CSV files into Ghini select *Tools→Export→Comma Separated Values* from the menu.

Une fois cliqué sur OK dans la boîte de dialogue, qui demande si vous êtes sûr que vous savez ce que vous faites, un sélecteur de fichier s'ouvre. Dans le sélecteur de fichier, sélectionnez les fichiers à importer.

3.7.2 Chargement en cours

To export the Ghini data to CSV select *Tools→Export→Comma Separated Values* 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 tablename.txt where tablename is the name of the table where the data was exported from.

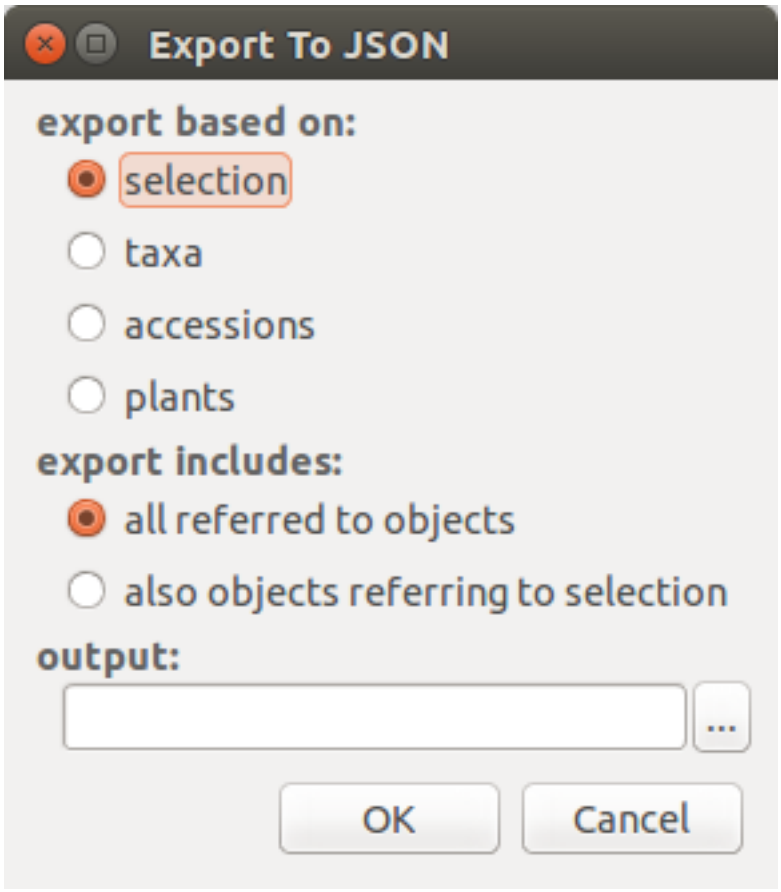
3.7.3 Importation de CD

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.

À l'aide du format d'échange de Ghini json, vous pouvez importer des données que vous avez exporté à partir d'une installation de Ghini différente.

3.7.4 Exporting to JSON

This feature is still under development.



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.

3.7.5 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.

3.7.6 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.

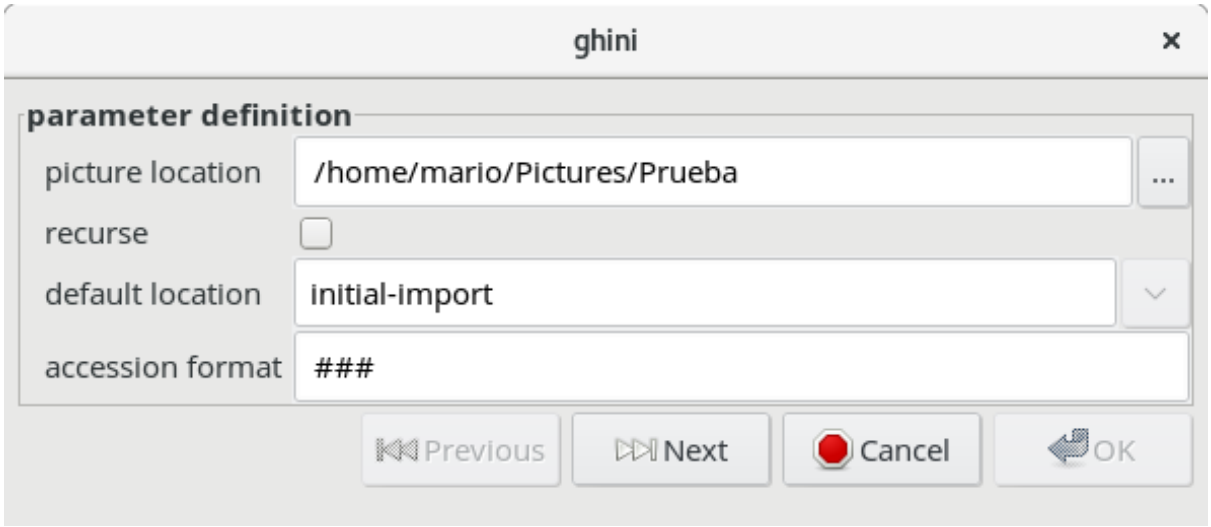
Even without using a photo collection software, you can associate pictures to accessions by following one and the same clear rule when naming picture files.

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.

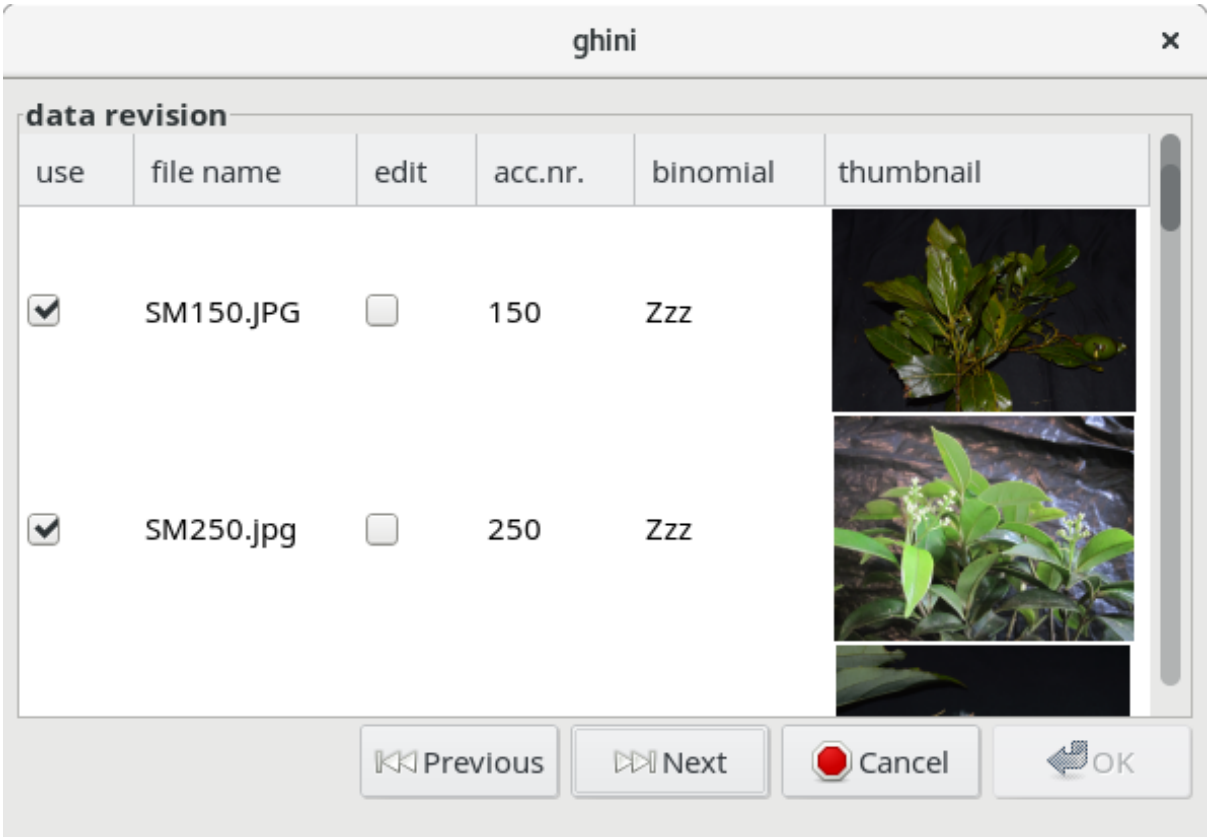
The *Tools→Import→Pictures* functionality here described is meant for importing an ordered collection of plant pictures either to initialize a ghini database, or for periodically adding to it.

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.

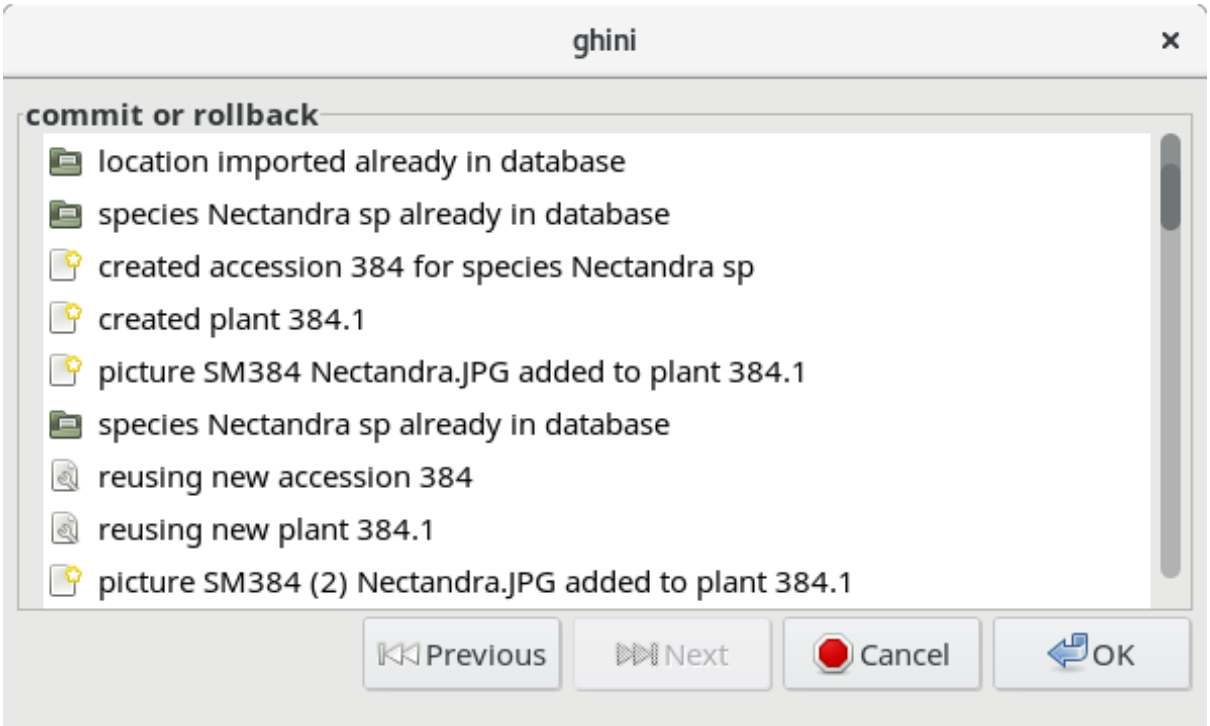
In the « parameter definition » pane you : select the directory from which you intend to import pictures ; indicate whether to import pictures recursively ; select or create a location which will be used as default location for new plants ; inform the tool about the rule you've been following when naming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

3.8 Gestion des utilisateurs

Note : Le plugin des utilisateurs Ghini n'est disponible que sur les bases de données PostgreSQL.

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 Création d'utilisateurs

Pour créer un nouvel utilisateur. . .

3.8.2 Autorisations

Ghini permet les autorisations de lecture, d'écriture et d'exécution.

CHAPITRE 4

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 Jardin Botanique de Quito

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.

Serveur de bases de données

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

Pour créer un nouvel utilisateur...

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.

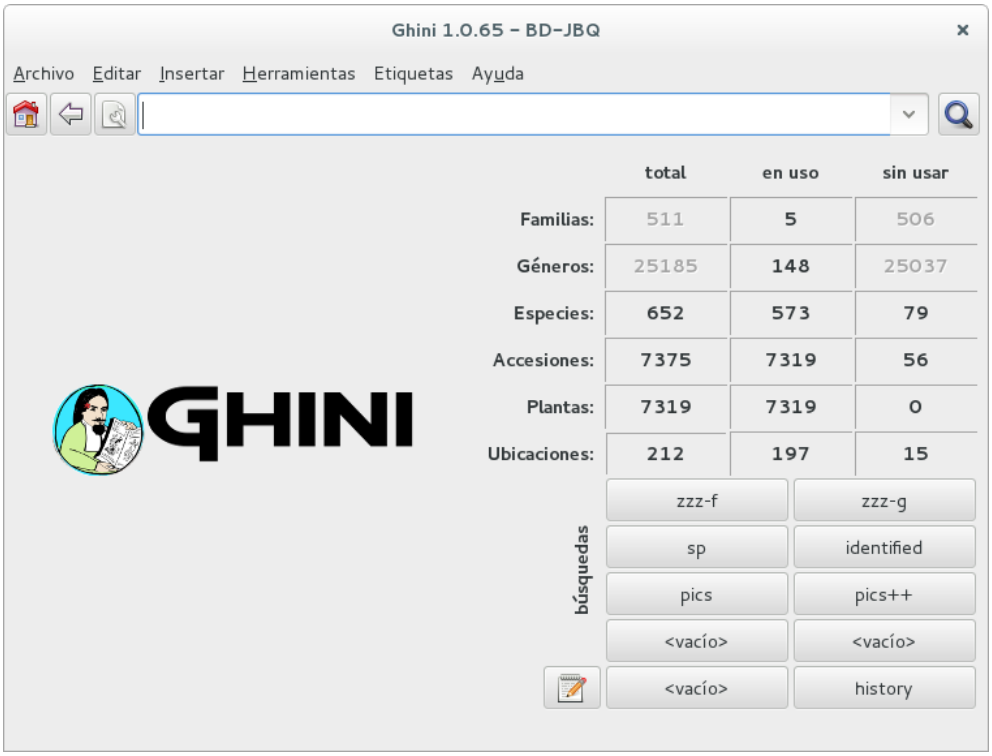
The update procedure is simple, and it depends on the operating system you use, we're not explaining here again.

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.

	total	in use	unused
Families:	511	7	504
Genera:	25394	158	25236
Species:	637	623	14
Accessions:	7722	7675	47
Plants:	7676	7676	0
Locations:	170	163	7

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.

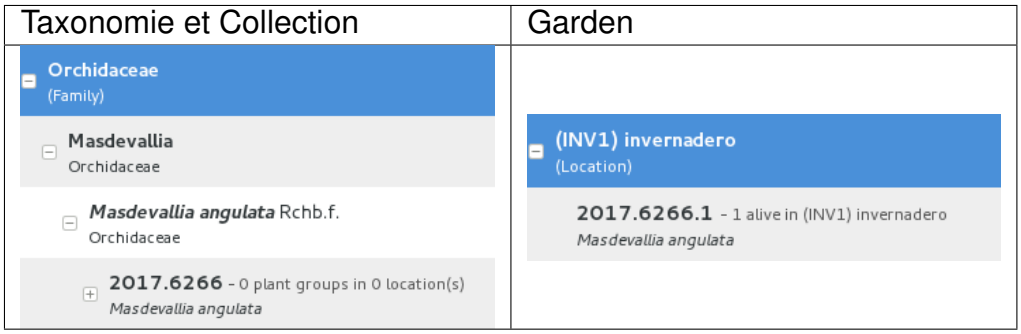
— With the  button, in the form of an arrow, you can return to your last search.

— 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.

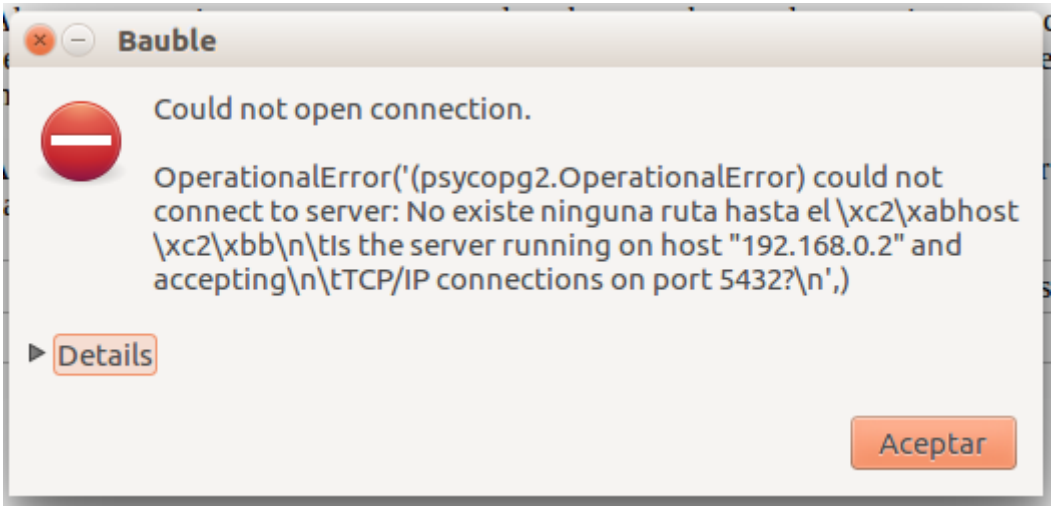


— At times, the program gives error messages. 

, 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.

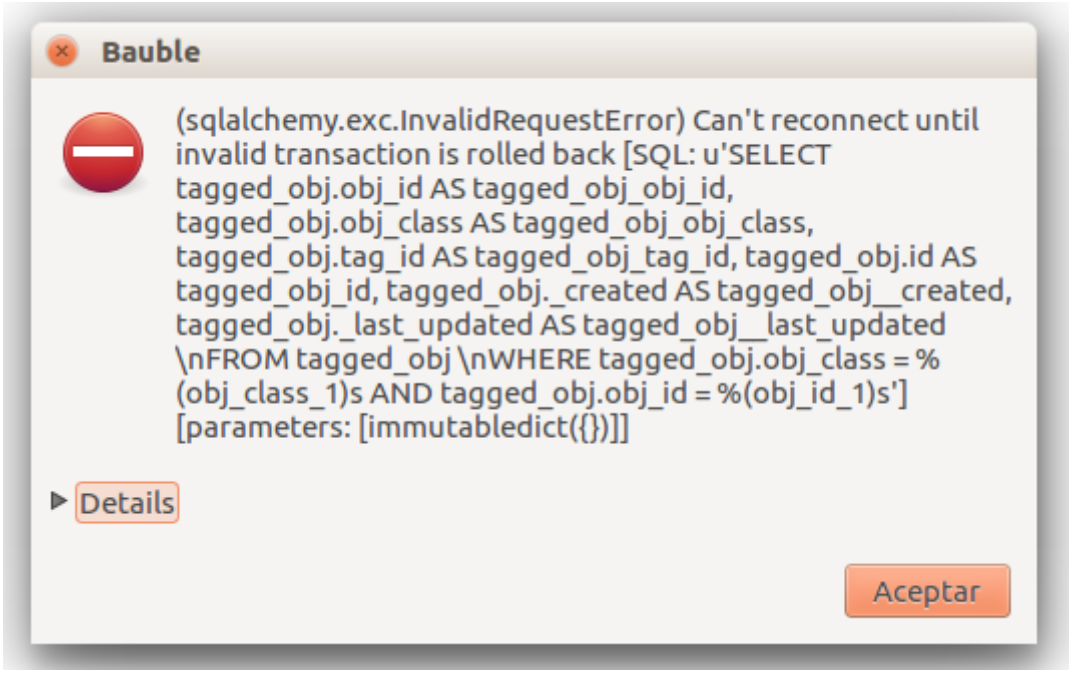


Just ignore it and try again.

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

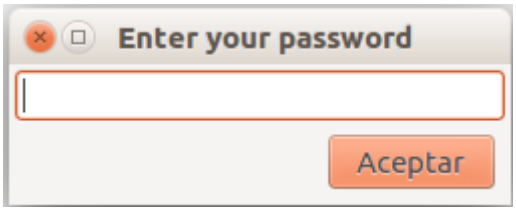
We propose to have different connection profiles, associated to different database users, each user with all needed permissions.

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

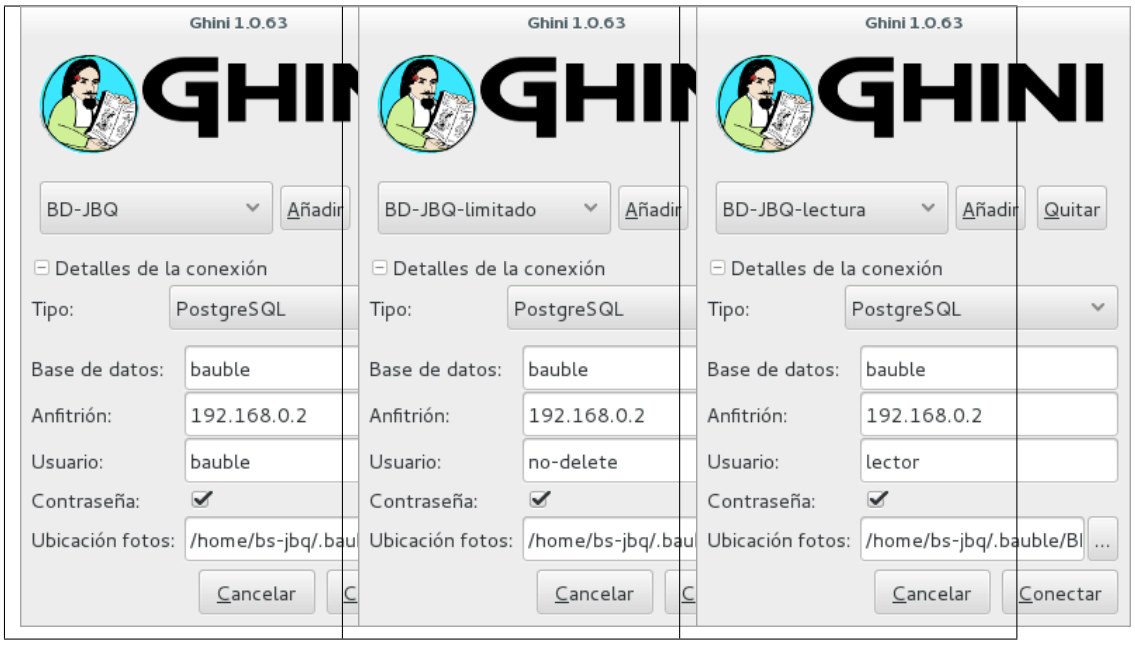
Insert and update (BD-JBQ-limitado) We use this one for those users who come help us for a limited time, and who did not get a complete introduction to database concepts. It is meant to prevent costly mistakes.

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.

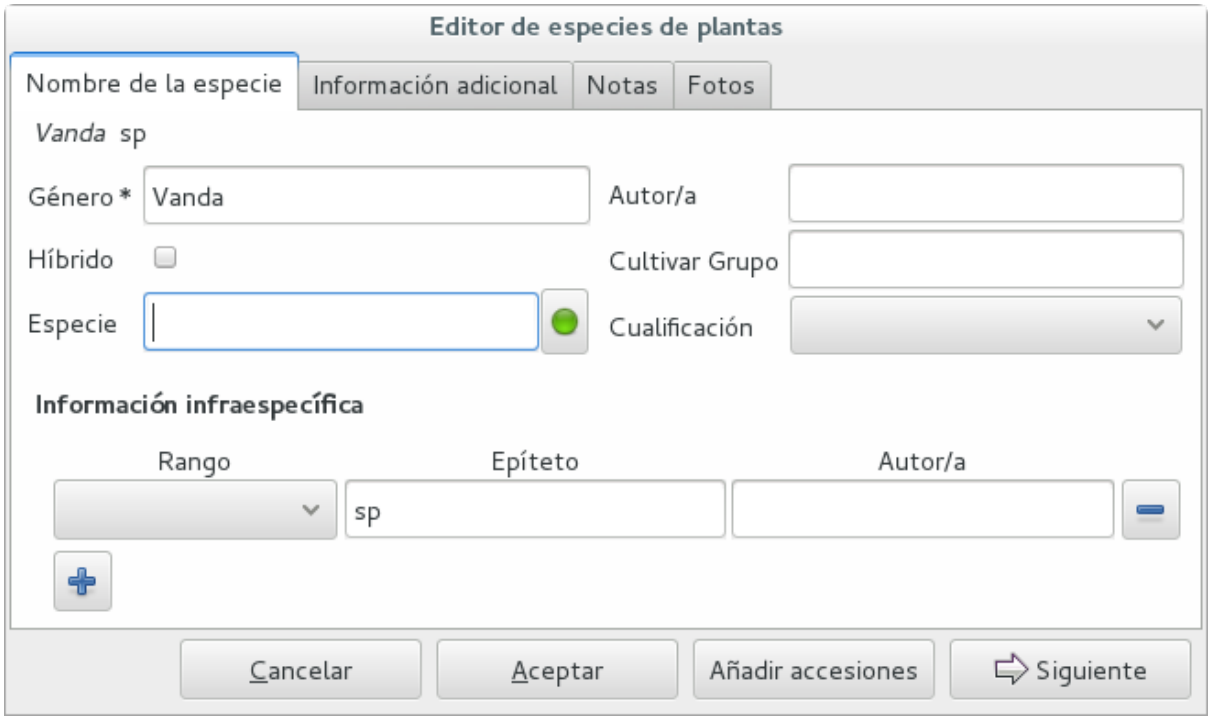
The taxonomy of the Orchidaceae family is continuously being reviewed. Genera get added, refused, reorganized, recognized as synonyms, some taxonomists prefer grouping species or genera in a new way, others split them again and differently, botanists of different nationalities may have different views 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-1.0 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-1.0 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 :

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.

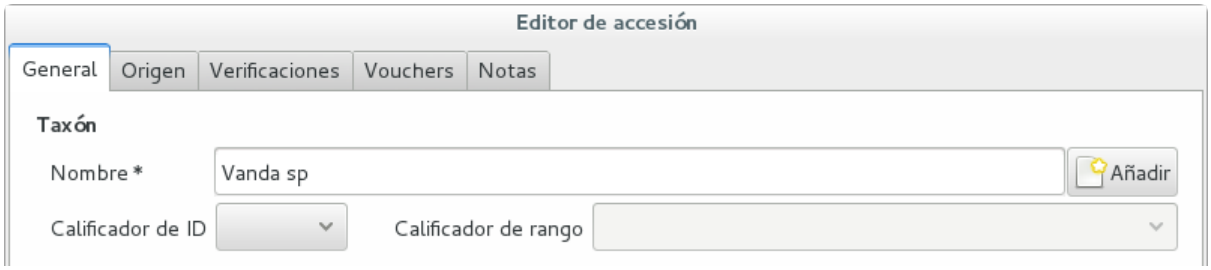
In particular, the subfamily information is relevant, because there are subfamilies within the Orchidaceae family which are not further separated.

— 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.

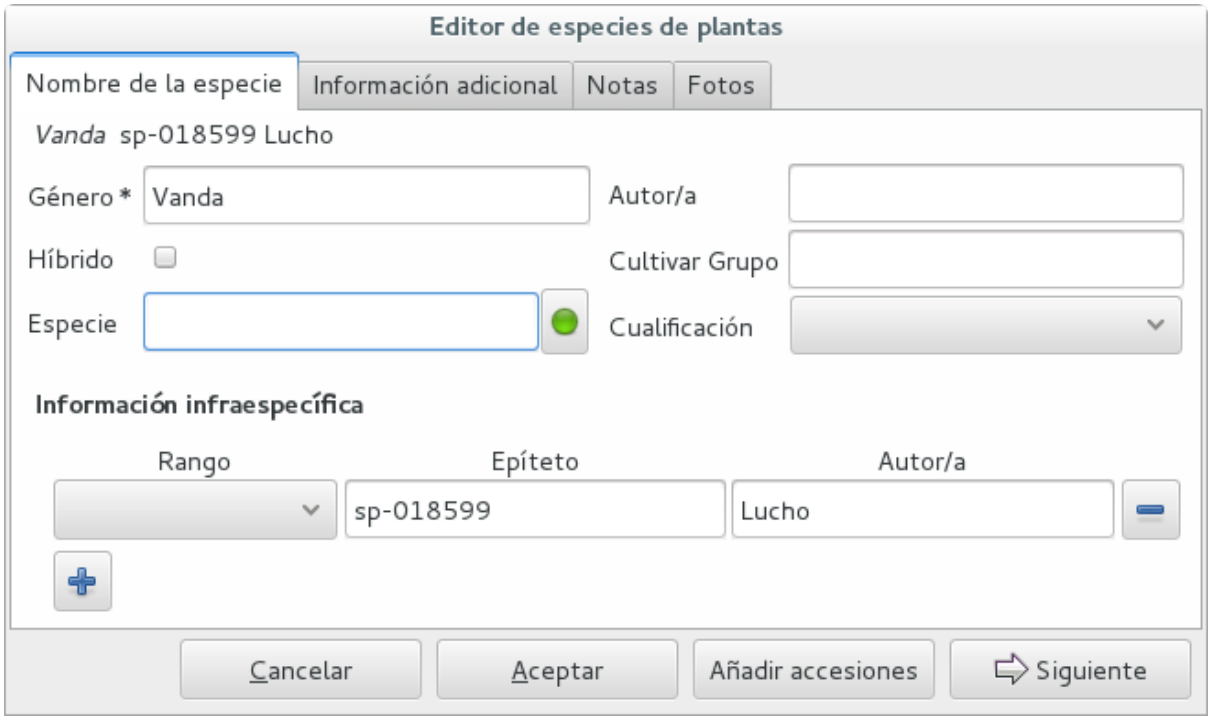


We do not edit the species, because there might be totally unrelated accessions connected to the same placeholder species.

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.

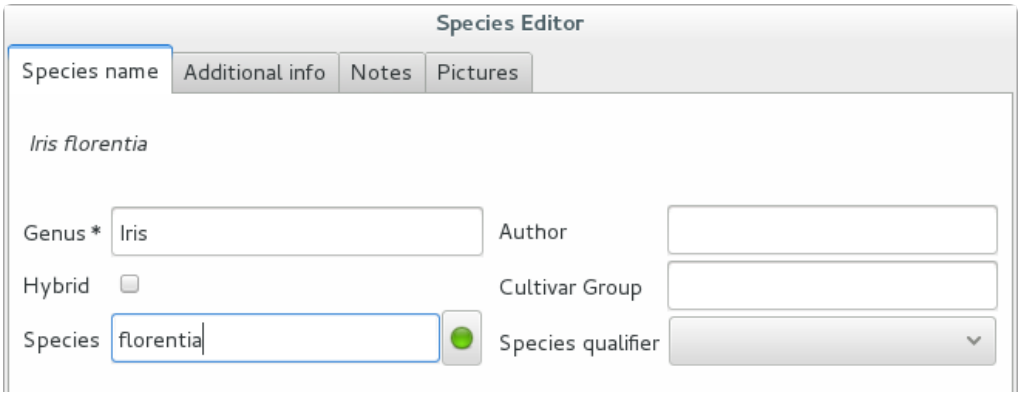
Last minute species

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

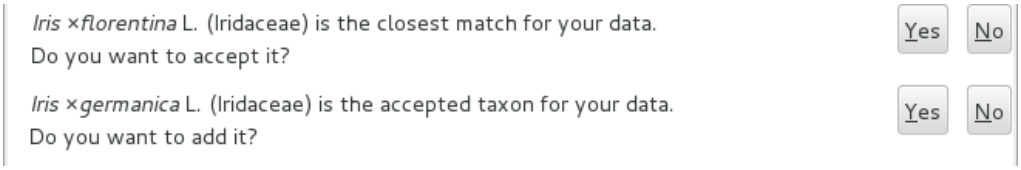
Adding a new species — the plant list.

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.

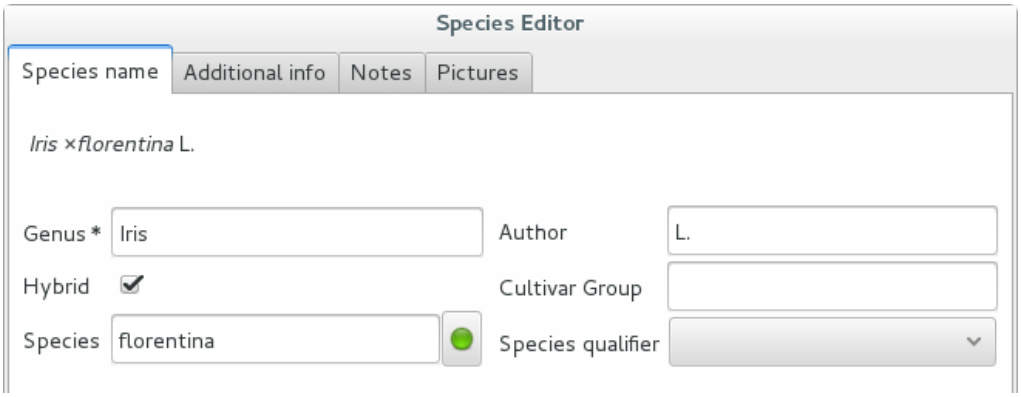


Next to the species epithet field there's a small button,  , which connects us to the plant list. Click on it, a message area appears at the top of the window.

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 especificos 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)

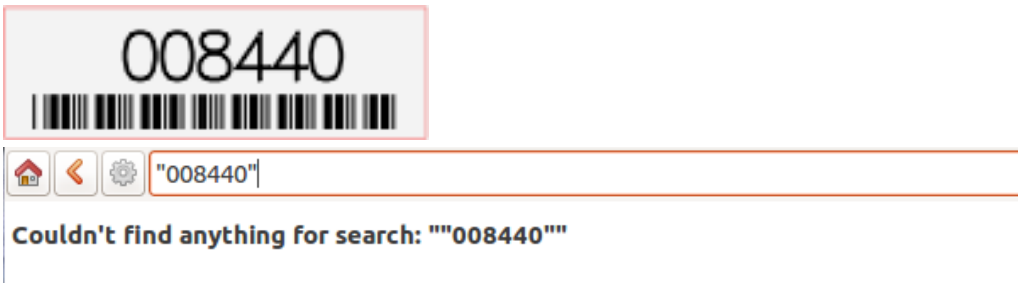
— Adding an Accession for a Plant

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 counterpart.

Existing plant, found in the garden with its own label

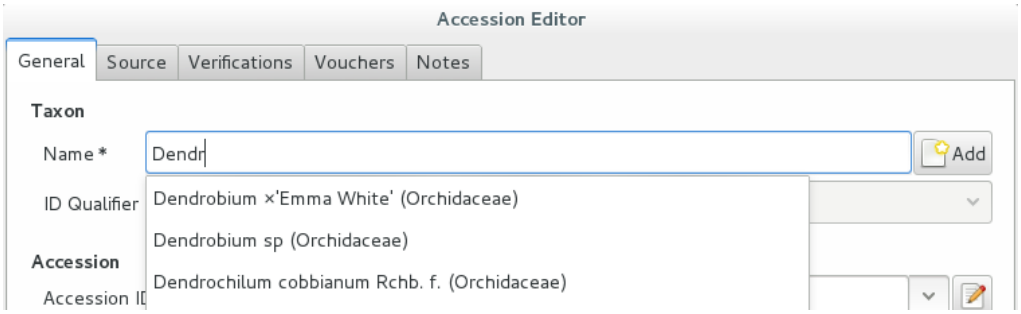
This activity starts with a plant, which was found at a specific garden location, an accession label, and the knowledge that the accession code is not in the database.



For this example, let's assume we are going to insert this information in the database.

Accès	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 :

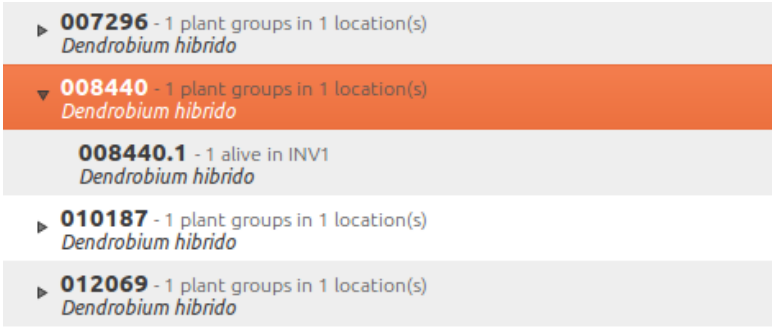
Accession ID	Type of Material	Quantity	Provenance
008440	Plante	1	Unknown

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	Plante	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

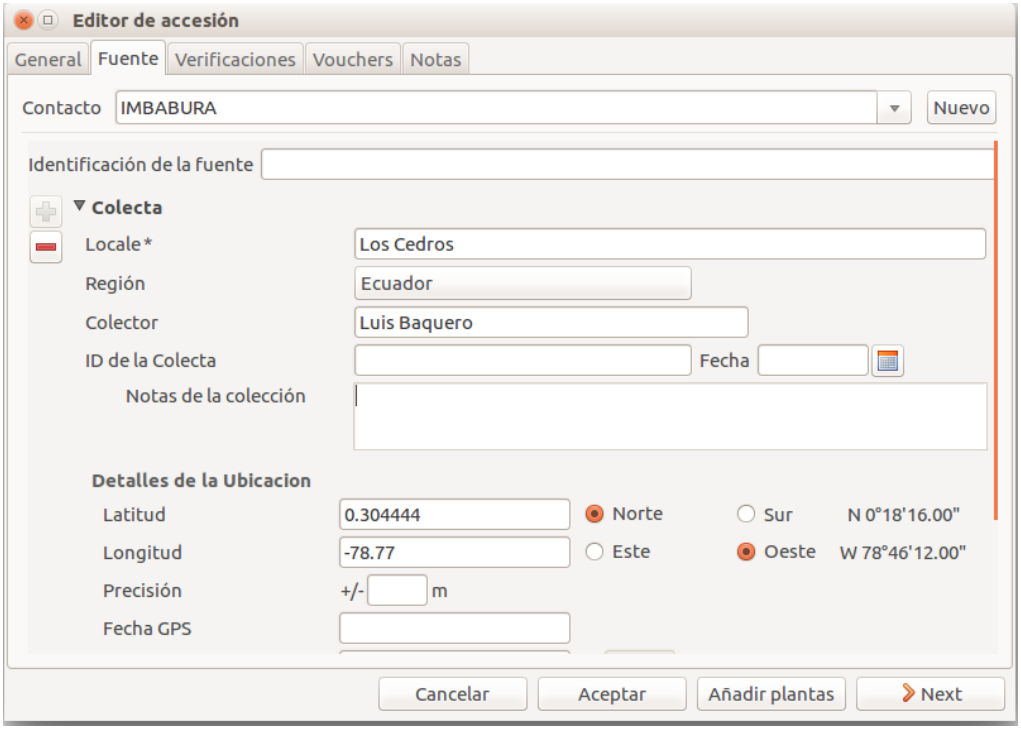
The accession editor offers the option « origin » option. When a plant is traceable to a wild source, we can specified its specific origin. We want to comply with ITF2, and ghini-1.0 only partly respects that standard. The ITF2 complying options are :

— 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.

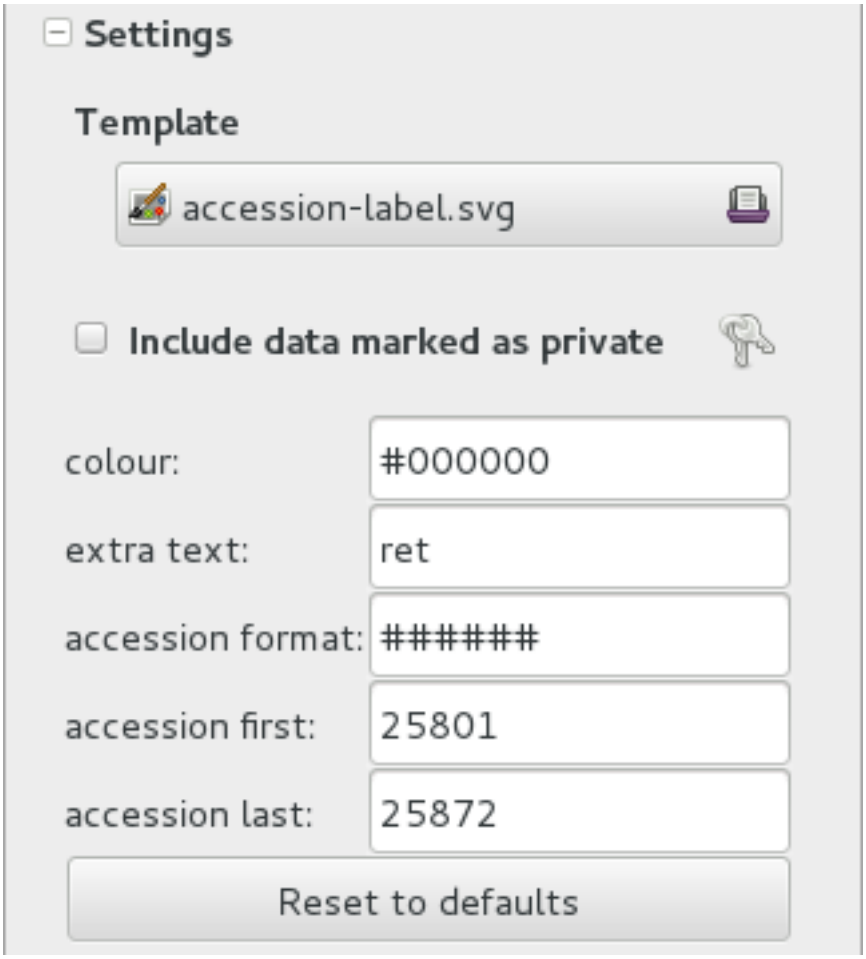
In the first case it's easy selecting all plants in the Location, we just type the location name, or give the search `location like <location name>`.

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.

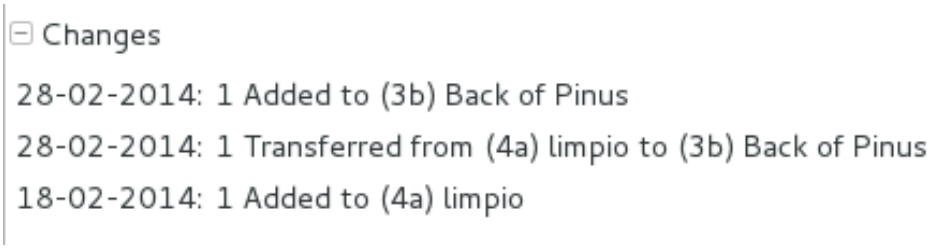
We type the accession code in the search entry field, with quotes, hit enter. The search results now shows the accession, and it tells us how many plants belong to it. Click on the squared **+** in the results row, so we now also see a row for the plant belonging to the accession.

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 :

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



— 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.

---

exportations et rapports

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

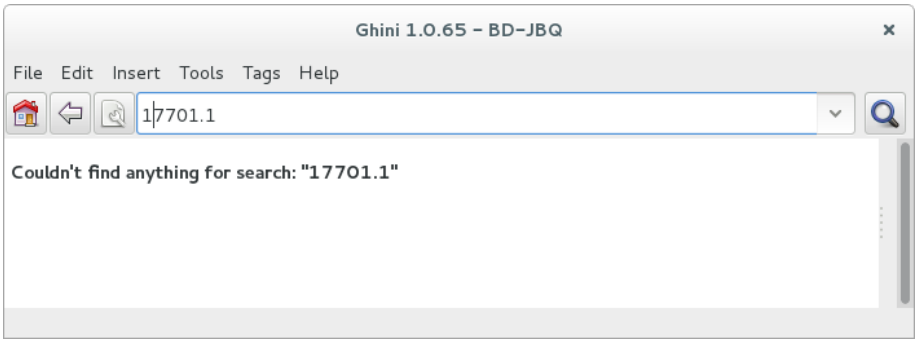
You search the database in order to edit the data further, or because you want to produce a report. Anyway you start with typing something in the search field

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

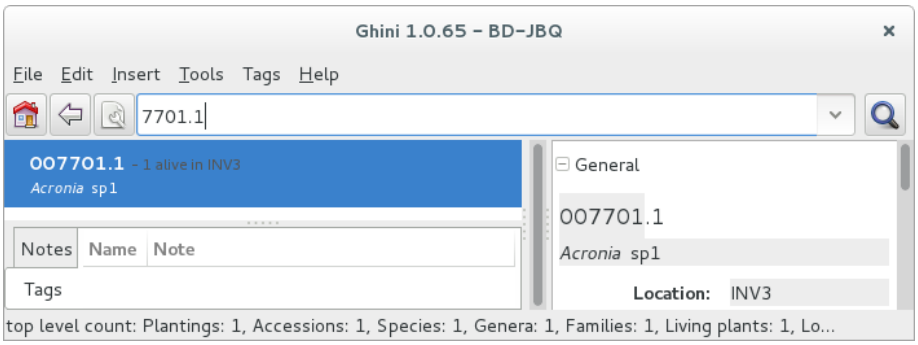
Search in order to edit (plant or accession)

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 :

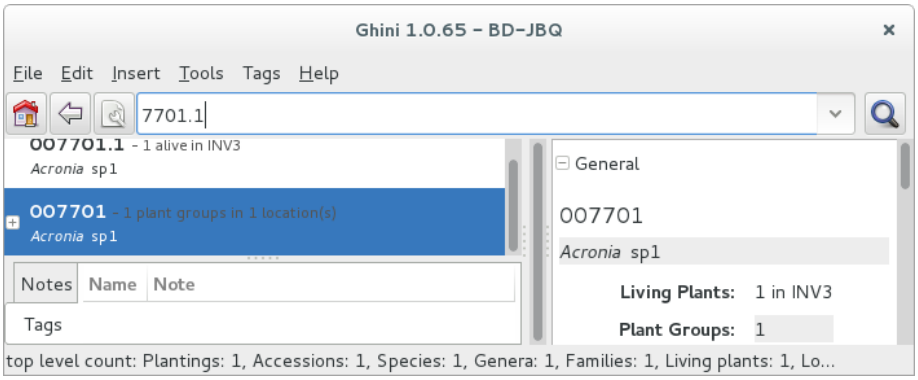


Other example, plant `007701.1` is in the database :



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

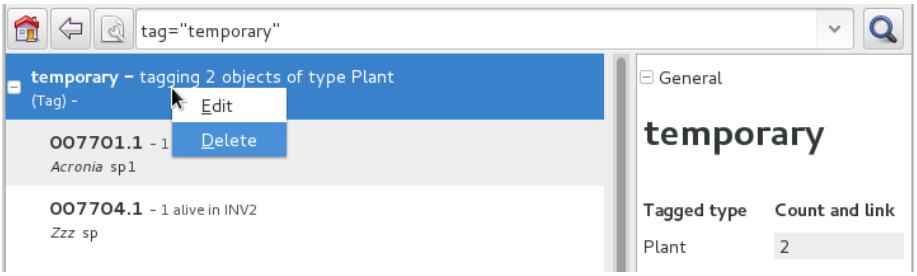
Sometimes we have to take the same action on objects of the same type, but we don't manage to quickly think of a search query that would group all that we need and exclude all we do not need.

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.

CHAPITRE 5

Administration

5.1 Administration de bases de données

Si vous utilisez un vrai SGBD pour contenir vos données botaniques, alors vous devez faire quelque chose pour administrer votre base de données. La admistration de une base de données est bien au-delà de la portée du présent document, nous faisons nos utilisateurs au courant.

5.1.1 SQLite

SQLite n'est pas ce que l'on envisagerait un SGBD réel : chaque base de données SQLite n'est rien plus que un seul fichier. Faites des copies de sécurité et tous sera bien. Si vous ne savez pas où sont vos fichiers de base de données, considérez que, par défaut, Ghini met ses données dans le répertoire `~/.bauble/`.

Dans Windows, il est quelque part dans votre répertoire « AppData », probablement en `AppData\Roaming\Bauble`. Gardez à l'esprit que Windows fait de son mieux pour cacher la structure de répertoire « AppData » pour les utilisateurs normaux.

La manière la plus rapide consiste à ouvrir avec l'Explorateur de fichiers : taper `%AppData%` et appuyez sur Entrez.

5.1.2 MySQL

Veuillez vous référer à la documentation officielle.

La sauvegarde et la restauration des bases de données sont décrites en détail à partir de la page.

5.1.3 PostgreSQL

Veuillez vous reporter à la documentation officielle. Une description très approfondie de vos options de sauvegarde commence au chapitre 24.

5.2 Configuration de Ghini

Ghini utilise un fichier de configuration pour stocker des valeurs à travers les appels du logiciel. Ce fichier est associé à un compte d'utilisateur et chaque utilisateur aura son propre fichier de configuration.

Pour consulter le contenu du fichier de configuration Ghini, tapez `:prefs` dans la zone de saisie de texte où vous normalement tapez votre recherche, puis appuyez sur Entrez.

Vous ne devez normalement pas peaufiner le fichier de configuration, mais vous pouvez le faire si vous voulez avec un normal programme d'édition de texte. Le fichier de configuration Ghini est par défaut au même emplacement que les bases de données SQLite.

5.3 Signaler les erreurs

Si vous remarquez quelque chose d'inattendu dans le comportement de Ghini, s'il vous plaît envisager de déposer un *issue* sur le site de développement de Ghini.

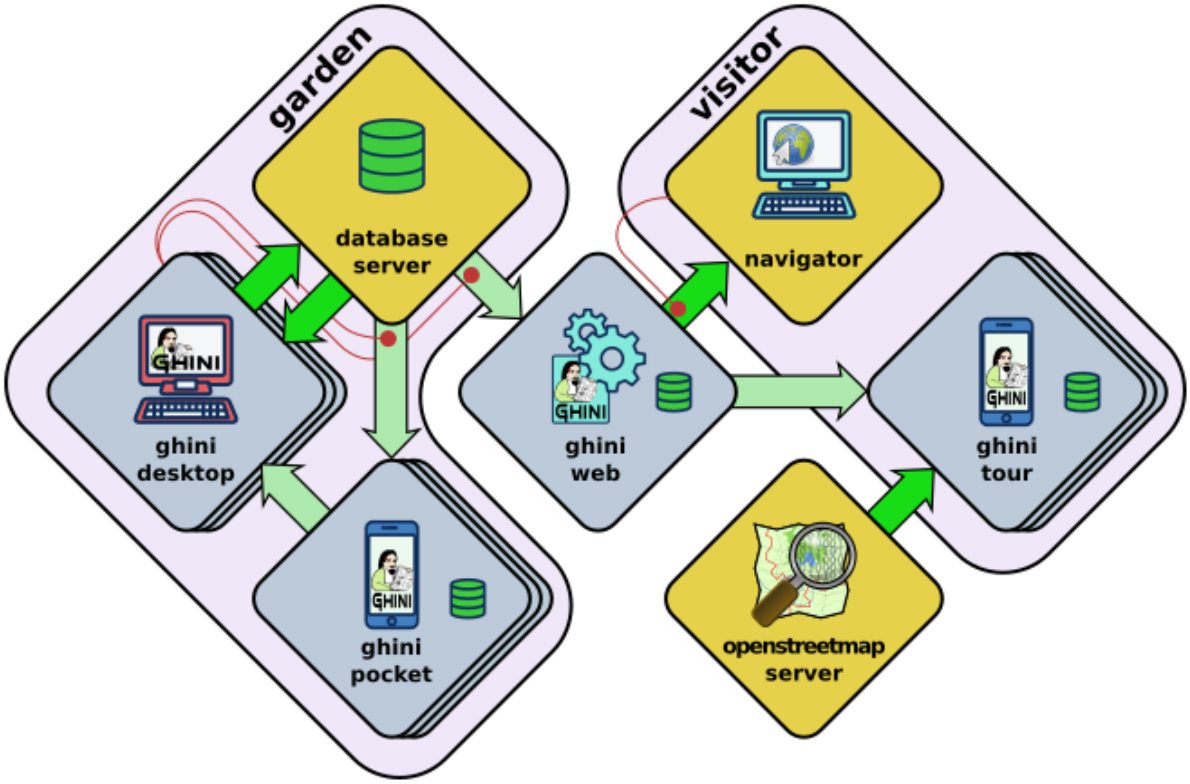
Le site de développement de Ghini est accessible via le menu aide.

CHAPITRE 6

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 *data streams between software components*.

6.1.2 ghini.web

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

Its most visible part runs at http://gardens.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 software 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 software components*.

6.1.4 data streams between software components

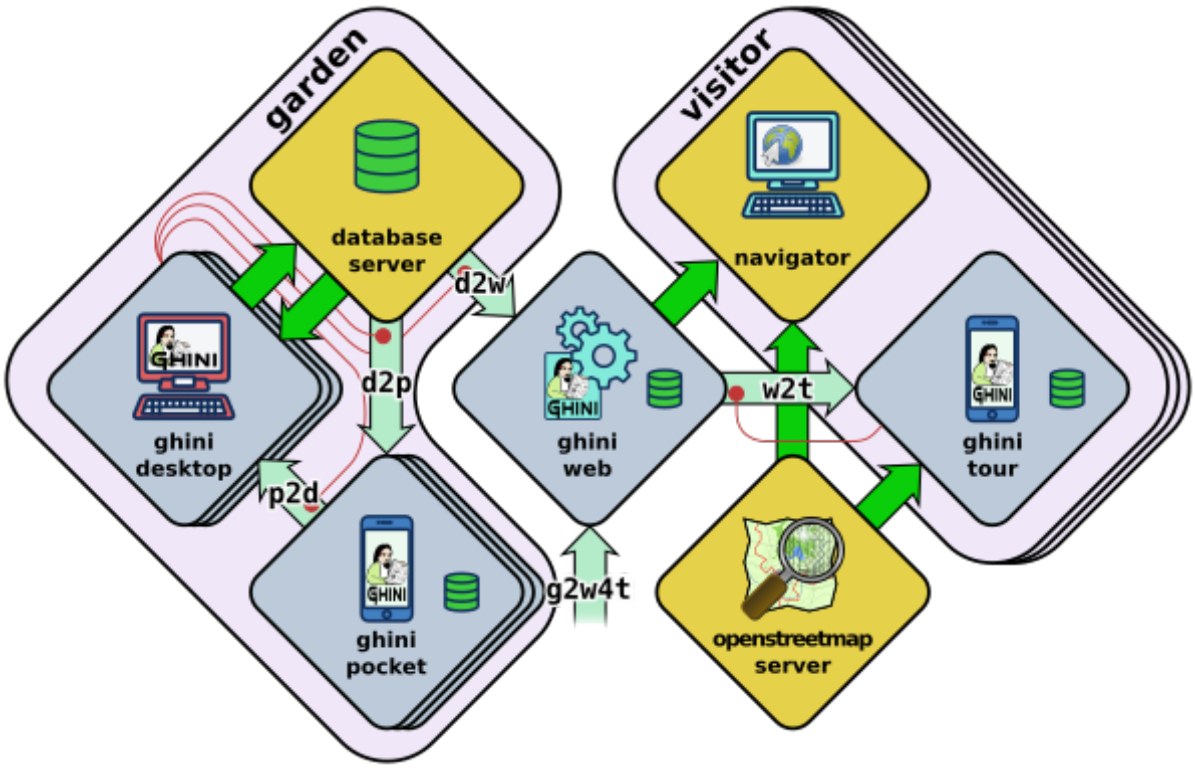
Note : This section contains technical information for database managers and software developers.

In the diagram showing the composition of the Ghini family, the alert reader noticed how different arrows representing different data flows, had different colours : some are deep green, some have a lighter tint.

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.



d2p : copy a snapshot of the desktop database to ghini.pocket

— export the desktop database to a pocket snapshot
— copy the snapshot to the handheld device

ghini.pocket integrates closely with ghini.desktop, and it's not a tool for the casual nor the external user. One task of your garden database manager is to regularly copy an updated database snapshot to your Android device.

We advise enabling USB debugging on the device. In perspective, this will allow ghini.desktop writing directily into the ghini.pocket device.

Export the file from ghini.desktop, call the file pocket.db, copy it to the phone :

```
adb -d push /tmp/pocket.db /sdcard/Android/data/me.ghini.
↪pocket/files/
```

The above location is valid even if your phone does not have a memory card.

Other options include bluetooth, or whatever other way you normally use to copy regular files into your Android device.

---

p2d : import from the ghini.pocket log file and pictures into the central database

even if we're still calling it "inventory log", ghini.pocket's log contains more than just inventory corrections.

— produce a log on the handheld device
— import the log in the desktop database

first of all, copy the collected information from ghini.pocket into your computer :

```
export DIR=/some/directory/on/your/computer
adb -d pull /sdcard/Android/data/me.ghini.pocket/files/
↪searches.txt $DIR
adb -d pull -a /sdcard/Android/data/me.ghini.pocket/files/
↪Pictures $DIR
```

then use ghini.desktop to import this information into your database.

---

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

---

g2w : 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)

CHAPITRE 7

Ghini Development

7.1 Manuel du développeur

Si vous avez exécuté les instructions d'installation `devinstall`, vous avez téléchargé les sources, et elles sont maintenant connectées au dépôt github. Vous êtes donc dans la situation la plus idéale pour commencer à examiner le logiciel, comprendre son fonctionnement, contribuer au développement de ghini.desktop.

7.1.1 Aider le développement de Ghini

Si vous souhaitez contribuer à Ghini, vous pouvez le faire de différents manières :

— Utiliser le logiciel, notez les choses que vous n'aimez pas, ouvrir une *issue* pour chacun d'eux. Un développeur va réagir plus tôt que vous pouvez imaginer.

— Si vous avez une idée de ce que vous ne trouvez pas dans le logiciel, mais ne pouvez pas tout à fait il codifier dans des questions distinctes, vous pourriez envisager l'embauche d'un professionnel. Ça c'est la meilleure façon de s'assurer que une chose entre rapidement dans Ghini. Assurez-vous que le développeur ouvre les *issue* et publie leur contribution sur github.

— Traduisez ! Toutes aide pour les traductions seront les bienvenues, alors s'il vous plaît, traduisez ! Vous pouvez faire cela sans rien n'installer sur votre ordinateur, simplement en utilisant le service de traduction en ligne offert par http://hosted.weblate.org/

— fourchéz le dépôt, choisissez un problème, résolvez-le, ouvrez une demande *pull request*. Voir le « flux de travail résolution de bogue » ci-dessous.

Si vous n'avez pas encore installé Ghini et veulez avoir un regard sur son histoire de code, vous pouvez ouvrir notre page projet github et voir tout ce qui s'est passé autour de Ghini depuis sa création, vers l'an 2004.

Si vous installez le logiciel en suivant les instructions `devinstall`, vous avez l'historique complet dans votre clone git local.

7.1.2 Source du logiciel, versions, branches

Si vous voulez une version en particulièr de Ghini, on libère et maintent les versions en forme de branches. Vous devriez faire le `git checkout` de la branche correspondant à la version de votre choix.

ligne de production

Les noms de branche pour les versions stable (production) Ghini sont de la forme `ghini-x.y` (par exemple : `ghini-1.0`); les noms de branche où sont publiées les versions tests Ghini sont de la forme `ghini-x.y-dev` (par exemple : `ghini-1. 0-dev`).

7.1.3 Flux de travail de développement

Notre flux de travail prévoit de publiquer continuement à la branche de test, puosser souvent à github, laisser travis-ci et coveralls.io vérifier la qualité du contenu de la branche de test, enfin, de temps en temps, de fusionner la branche de test dans la version correspondante.

Lorsque je travaille à des questions plus vastes, qui semblent prendre plus de quelques jours, je pourrais ouvrir une branche liée à la question. Je ne fais cela pas trop souvent.

grandes questions

Face à un seul problème plus vaste, créez une étiquette de branche à l'extrémité d'une ligne de développement principal (par exemple : `ghini-1.0-dev`) et suivez le workflow décrit à

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

En bref

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

Travailler sur la nouvelle branche temporaire. Lorsque vous êtes prêt, allez à github, fusionner la branche avec la ligne de développement principale d'où vous avez creé la branche, résoudre les conflits, le cas échéant, puis supprimez la branche temporaire.

Lorsque vous êtes prêt pour la publication, fusionner la ligne de développement dans la ligne correspondante.

7.1.4 Mise à jour de l'ensemble des chaînes traduisibles

De temps à autre, pendant le processus de mise à jour le logiciel, vous ajoutez ou modifiez les chaînes dans les sources python, dans la documentation, dans les sources de glade. La plupart de nos chaînes est traduisible et est offerts sur weblate pour contribuer, sous forme de plusieurs fichiers `.po`.

Un `po` est composé de paires de parties de texte, original et la traduction corréspondante et est spécifique à une langue. Quand un traducteur ajoute une traduction sur weblate, cela arrive à notre dépôt sur github. Quand un programmeur ajoute une chaîne au logiciel, cela arrive sur weblate « à être traduite ».

Weblate héberge le projet Ghini. Dans ce projet, nous avons des composants, dont chacun correspond à une branche d'un dépôt sur github. Chaque composant accepte des traductions en plusieurs langues.

composant	dépôt	branche
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
Tour	ghini.tour	master

Pour mettre à jour les fichiers `po` relatif à le *logiciel*, vous définissez le répertoire de travail à la racine de votre copie locale de *ghini.desktop* et vous exécutez le script :

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

Pour mettre à jour les fichiers `po` relatif à la *documentation*, vous définissez le répertoire de travail à la racine de votre copie locale de *ghini.desktop-docs.i18n*, et vous exécutez le script :

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

Lorsque vous exécutez une des scripts mentionnés ci-dessus, il est trés probable que vous avez besoin de commettre *touts* fichier `po` du projet. Vous pouvez examiner les modifications avant de les engager dans le référentiel. Ça c'est plus important lorsque vous effectuez une correction marginale à une chaîne, comme enlever une faute de frappe.

Quelque chose qui se passe : se présenter à un conflit. Résoudre les conflits n'est pas difficile une fois que vous savez comment le faire. Tout d'abord, ajouter weblate comme remote :

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

Ensuite, assurez-vous que nous sommes dans le dépôt correct, sur la branche correcte, mettre à jour le remote, fusionner avec elle :

```
git checkout ghini-1.0-dev
git remote update
git merge weblate-doc10/ghini-1.0-dev
```

Notre documentation sur readthedocs a une version anglaise et plusieurs traductions. Ici on suive la description de la localisation, il n'y a rien que nous avons nous-mêmes inventé ici.

Readthedocs vérifie le reglage *Language* du projet et appelle `sphinx-intl` pour produire la documentation mise en forme dans la langue cible. Avec la configuration par défaut — dont nous n'avons pas modifié — `sphinx-intl` attend un fichier `po` par le document source, nommé que le document source, et qu'ils sont tous résident dans le répertoire `local/$(LANG)/LC_MESSAGES/`.

En revanche, Weblate (et nous-mêmes) préfère un fichier unique `po` par langue et on les garde tous dans le même répertoire `/po`, tout comme nous le faisons pour le projet logiciel : `/po/$(LANG).po`.

Afin de ne pas répéter cette information et de laisser les deux systèmes fonctionnent à leur manière naturelle, nous avons deux ensembles de liens symboliques (git rend hommage à eux).

Pour résumer : lorsqu'un fichier dans la documentation est mise à jour, le script `runme.sh` :

1. Copie les fichiers `rst` du logiciel à la documentation ;
2. Crée un nouveau fichier de `pot` pour chacun des fichiers `rst` ;
3. fusionne tous les fichiers `pot` dans un seul `doc.pot` ;
4. Utilize le nouveau `doc.pot` pour mettre à jour tous les fichiers `doc.po` (un par langue) ;
5. créer tous les liens symboliques :
   a. ceux utilisées par `sphinx-intl` en `/local/$(LANG)/LC_MESSAGES/`
   b. ceux utilisées par weblate en `/po/$(LANG).po`

Nous pourrions certainement écrire ce qui précède dans un Makefile, ou mieux encore l'inclure dans `/doc/Makefile`. Qui sait, peut-être que nous le ferons.

7.1.5 Produire les documents localement

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 you install `ghini.desktop` or one of the Android apps, the installation doesn't assume a specific run-time language : a user can change their 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.

At run-time 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.

The path followed by translations is : You edit strings on Weblate, Weblate keeps accumulating them until you are done, or you don't interact with Weblate for a longer while ; Weblate pushes the strings to github, directly into the development line `ghini-1.0-dev` ; I see them and I might blindly trust or prefer to 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. As said, this lands in the development line `ghini-1.0-dev`, which I regularly publish to the production line `ghini-1.0`, 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`, it is similar, but the notification is handled by the Android system. We publish on the Play Store, and depending on your settings, your phone will update the software automatically, or only notify you, or do nothing. It depends on how you configured automatic updates.

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 Ajout des tests unitaires manquant

Si vous êtes intéressé à contribuer au développement de Ghini, un bon moyen de le faire serait de nous aider à trouver et écrire les tests unitaires manquantes.

Une fonction bien testée est un dont le comportement vous ne pouvez pas modifier sans casser au moins un test unitaire.

Nous sommes tous d'accord qu'en théorie la théorie et la pratique correspondant parfaitement et que l'un d'abord écrit des essais, puis implémente la fonction. En pratique, toutefois, pratique ne correspond pas à la théorie, et nous avons été l'écriture de tests après avoir écrit et même les fonctions d'édition.

Cette section décrit le processus d'ajout de tests unitaires pour les `bauble.plugins.plants.family.remove_callback`.

POUR TESTER

Tout d'abord, ouvrir l'index de rapport de couverture et choisissez un fichier à faible couverture.

Pour cet exemple, exécutez en octobre 2015, nous avons atterri sur `bauble.plugins.plants.family`, à 33 %.

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

Les deux premières fonctions nécessitant des tests, `edit_callback` et `add_genera_callback`, incluent la création et l'activation d'un objet en s'appuyant sur une boîte de dialogue personnalisée. On doit vraiment première écriture des tests unitaires pour cette classe, puis revenir ici.

La fonction suivante, `remove_callback`, active également quelques boîtes de dialogue et de message, mais sous forme d'appel d'une fonction demandant l'entrée de l'utilisateur par l'intermédiaire de boîtes oui-non-OK. Ces fonctions nous pouvons facilement remplacer une fonction se moquant du comportement.

How to

Ainsi, après avoir décidé ce qu'il faut décrire en test unitaire, nous regardons le code et nous voyons qu'il faut distinguer deux cas :

le paramètre correct
- — la liste des familles comporte aucun élément.
- — la liste des familles a plus d'un élément.
- — la liste des familles a exactement un élément.

cascade
- — la famille n'a aucun genres
- — la famille a un ou plusieurs genres

confirm
- — l'utilisateur confirme la suppression
- — l'utilisateur ne confirme pas la suppression

deleting
- — tout se passe bien lors de la suppression de la famille
- — Il y a une erreur lors de la suppression de la famille

Je décide que je vais me concentrer uniquement sur le **cascade** et **confirmer** aspects. Deux questions binaires : 4 cas.

où mettre les tests

Recherchez le script de test et choisissez la classe où mettre les tests unitaires supplémentaires.

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

---

que fair des tests "skipped"

La classe de `FamilyTests` contient un test ignoré, sa mise en œuvre va être un peu de travail parce que nous devons réécrire le FamilyEditorPresenter, séparer de la FamilyEditorView et reconsidérer ce qu'il faut faire avec la classe FamilyEditor, qui je pense devrait être supprimé et remplacé par une seule fonction.

---

les essais d'écriture

Après le dernier essai dans la classe de FamilyTests, j'ai ajouter les quatre cas je veux décrire, et je m'assure qu'ils ne parviennent pas, et puisque je suis paresseux, j'écris le code plus compact je sais pour générer une erreur :

```
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
```

Un test, étape par étape

Commençons par le premier cas de test.

Lorsque vous écrivez des tests, j'ai suivent généralement le modèle :

— T_0 (condition initiale),
— action,
— T_1 (tester le résultat de l'action, compte tenu des conditions initiales)

---

what's in a name — unit tests

Il y a une raison pourquoi les tests unitaires sont appelés des tests unitaires. Ne Testez jamais deux actions dans un essai.

Nous allons donc décrire T_0 pour le premier test, une base de données contenant une famille sans genres :

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

Nous ne voulons pas la fonction testée pour appeler la fonction interactive `utils.yes_no_dialog`, nous voulons des `remove_callback` pour invoquer une fonction de

remplacement non interactif. Nous accomplissons ceci simplement en faisant le point de `utils.yes_no_dialog` à une expression `lambda` qui, comme la fonction interactive originale, accepte un seul paramètre et retourne une valeur booléenne. Dans ce cas : `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)
```

Ensuite, nous vérifions le résultat.

Eh bien, nous ne voulons juste tester si oui ou non l'objet Arecaceae a été supprimé, nous aussi devrions tester la valeur renvoyée par `remove_callback`, et si les `yes_no_dialog` et `message_details_dialog` ont été invoquées ou non.

Une expression `lambda` n'est pas assez pour cela. Nous faisons quelque chose d'apparemment plus complexes, qui rendra la vie beaucoup plus facile.

Nous allons d'abord définir une fonction plutôt générique :

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

et de nous saisir `partielle` du module standard `functools`, partiellement appliquer la précède `mockfunc`, laissant seulement `msg` non précisée, et employer cette application partielle, qui est une fonction acceptant un paramètre et retournant une valeur, afin de remplacer les deux fonctions dans `utils`. La fonction de test ressemble maintenant à ceci :

```
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()
```

La section test vérifie que `message_details_dialog` ne était pas invoquées, que

`yes_no_dialog` a été, avec le paramètre message correct, Arecaceae est toujours là :

```
# 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])
```

et ainsi de suite...

> « Il y a deux sortes de gens, ceux qui terminent ce qu'ils entreprennent et ainsi de suite »

Prochaine épreuve est presque le même, avec la différence que le `utils.yes_no_dialog` doit retourner `vrai` (cela nous atteindre en spécifiant `résultat = True` dans l'application partielle du générique `mockfunc`).

Avec cette action, la valeur retournée par `remove_callback` doit être `True`, et il ne devrait y avoir aucune famille Arecaceae dans la base de données plus :

```
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")
```

(suite sur la page suivante)

(suite de la page précédente)

```
    matching = q.all()
    self.assertEquals(matching, [])
```

Jetez un oeil à 734f5bb9feffc2f4bd22578fcee1802c8682ca83 de validation pour les deux autres fonctions de test.

Tests de journalisation

Nos objets de `bauble.test.BaubleTestCase` utilisent des gestionnaires de la classe `bauble.test.MockLoggingHandler`. Chaque fois qu'un test unitaire individuel est démarré, la méthode `paramètres` créera un nouveau `gestionnaire` et associez-le à l'enregistreur de la racine. La méthode du `démontage` prend soin de retirer.

Vous pouvez vérifier la présence des messages de journalisation spécifique dans `self.handler.messages`. `messages` sont un dictionnaire, initialement vide, avec deux niveaux d'indexation. Tout d'abord le nom du logger délivrant la journalisation enregistrent, puis le nom du niveau de l'enregistrement de journalisation. Les clés sont créées lorsque nécessaire. Valeurs tiennent des listes de messages, mis en forme selon quel que soit le formateur, vous associez au gestionnaire, ou par défaut `logging.Formatter("%(message)s")`.

Vous pouvez explicitement vider les messages collectés en invoquant des `self.handler.clear()`.

Synergie et regroupement

De temps en temps vous voulez activer la classe de test que vous travaillez à :

```
nosetests bauble/plugins/plants/test.py:FamilyTests
```

Et à la fin du processus vous souhaitez mettre à jour les statistiques :

```
./scripts/update-coverage.sh
```

7.1.8 Structure de l'interface utilisateur

L'interface utilisateur est construit selon le **modèle** — **vue** — **présentateur** modèle architectural. Pendant une grande partie de l'interface, **modèle** est un objet de base de données de SQLAlchemy, mais nous avons aussi des éléments de l'interface où il n'y a aucun modèle de base de données correspondante. En général :

— Le **vue** est décrite dans le cadre d'un **fichier glade**. Cela devrait inclure le signal-rappel et associations ListStore-TreeView. Simplement réutiliser la `GenericEditorView` `bauble.editor`, selon la classe de base. Lorsque vous créez votre instance de cette classe générique, passez-lui le **glade** nom de fichier et le nom du widget racine, puis remettre cette instance à la **présentateur** constructeur.
Dans le fichier de la clairière, dans la section `action-widgets` fermer votre description de l'objet GtkDialog, assurez-vous que tous les éléments `action-widget`

a une valeur valide `réponse`. Utilisez les valeurs valides de GtkResponseType, par exemple :
— GTK_RESPONSE_OK, -5
— GTK_RESPONSE_CANCEL, -6
— GTK_RESPONSE_YES, -8
— GTK_RESPONSE_NO, -9
Il n'y a aucun moyen facile d'écrire test unitaire une vue sous-classé, donc s'il vous plaît ne pas sous-classer aucune vue, il n'y a vraiment pas besoin de faire ça.
Dans le fichier glade, chaque widget d'entrée devrait définir quel gestionnaire est activé sur lesquels le signal. La classe générique de présentateur propose des rappels génériques qui couvrent les plupart des cas courants.
— GtkEntry (entrée de texte d'une ligne) gérera le signal `changed`, avec `on_text_entry_changed` ou `on_unique_text_entry_changed`.
— GtkTextView : associez-le à un GtkTextBuffer. Pour gérer le signal `changed` sur le GtkTextBuffer, nous devons définir un gestionnaire qui invoque le générique `on_textbuffer_changed`, le seul rôle de cette fonction est de transmettre notre gestionnaire générique le nom de l'attribut de modèle qui reçoit le changement. Il s'agit d'un workaroud pour un " bug non résolu dans GTK <http : stackoverflow.com/questions/32106765/= » »>" _.</http :>
— GtkComboBox avec textes traduits ne peut pas être facilement manipulé du fichier glade, donc nous n'essayez même pas. Utilisez la méthode de `init_translatable_combo` de la classe générique de `GenericEditorView`, mais s'il vous plaît appeler à partir du **présentateur**.

— Le **modèle** est juste un objet avec des attributs connus. Dans cette interaction, les **modèle** est juste un conteneur de données passives, il ne fait rien plus que de laisser le **présentateur** modifiez-le.
— Le sous-classé **présentateur** définit et met en œuvre :
  — `widget_to_field_map`, un dictionnaire associant les noms de widget au nom des attributs de modèle,
  — `view_accept_buttons`, la liste des noms de widget qui, si activé par l'utilisateur, cela signifie que la vue doit être fermée,
  — tous besoin de rappels,
  — éventuellement, il joue le **modèle** rôle, trop.

  Le **présentateur** réactualise la **modèle** selon les changements dans le **vue**. Si le **modèle** correspond à un objet de base de données, le **présentateur** commet tous **mises à jour modèle** de la base de données lorsque le **vue** est fermé avec succès, ou les annule si la **vue** est annulée. (ce comportement est influencé par le paramètre `do_commit`)
  Si le **modèle** est autre chose, puis la **présentateur** va faire quelque chose d'autre.

Note : Un sage **présentateur** utilise le **vue** api pour interroger les valeurs insérées par l'utilisateur ou pour régler par la force des statuts de widget. S'il vous plaît n'apprennent pas de la pratique de nos animateurs de mauvaise conduites, certains dont manipuler directement les champs du `view.widgets`. Ce faisant, ces présentateurs nous empêche d'écrire des tests unitaires.

La classe de base pour le présentateur, `GenericEditorPresenter`, définie dans `bauble.editor`, implémente de nombreux rappels génériques utiles. Il y a une classe de `MockView`, que vous pouvez utiliser lorsque vous écrivez des tests à vos présentations.

Examples

`Contact` et `ContactPresenter` sont mis en œuvre suivant les lignes ci-dessus. L'affichage est défini dans le fichier `contact.glade`.

Un bon exemple de modèle de mode/Présentateur (pas de modèle) est donné par le gestionnaire de connexions.

Nous utilisons le même schéma architectural d'interaction non-base de données, en définissant le présentateur aussi comme modèle. Nous faisons cela, par exemple, pour la boîte de dialogue exportation JSON. La commande suivante vous donnera une liste des instanciations de `GenericEditorView`:

```
grep -nHr -e GenericEditorView\( bauble
```

7.1.9 Ghini qui s'étend avec des Plugins

Presque tout sur Ghini est extensible grâce aux plugins. Les plugins peuvent créer des tables, définir des recherches personnalisées, ajouter des éléments de menu, créer des commandes personnalisées et plus encore.

Pour créer un nouveau plugin nous devez étendre la classe `bauble.pluginmgr.Plugin`.

Le plugin `Tag` est un bon exemple de minime, même si le `TagItemGUI` n'entre pas dans le modèle architectural de Model-View-Presenter.

7.1.10 Structure de plugins

Ghini est un cadre pour la gestion des collections et est distribué avec un ensemble de plugins des Ghini un gestionnaire de collection botanique. Mais Ghini reste un cadre, et vous pouvez en théorie supprimer tous les plugins, nous distribuons et écrire vos propres, ou écrire vos propres plugins que prolonger ou compléter le comportement actuel de Ghini.

Une fois que vous avez sélectionné et ouvert une connexion de base de données, vous atterrissez dans la fenêtre de recherche. Le moteur de recherche est une interaction entre deux objets : SearchPresenter (SP) et SearchView (SV).

SV est ce que vous voyez, SP détient le statut de programme et gère les requêtes que vous exprimez par le biais de SV. Traitement de ces demandes affectent le contenu de la SV et l'état du programme de SP.

Les résultats de recherche affichés dans la plus grande partie du SV sont rangées, les objets qui sont des instances de classes enregistrés dans un plugin.

Chacune de ces classes doit implémenter une somme de fonctions afin de se comporter correctement dans le cadre de Ghini. Le cadre Ghini réserve un espace aux classes enfichables.

SP connaît de toutes les classes (branchés) enregistrés, ils sont stockés dans un dictionnaire, associant une classe pour sa mise en oeuvre du plugin. SV a un emplacement (un gtk. Box) où vous pouvez ajouter des éléments. À tout moment, au plus qu'un seul élément dans la fente est visible.

Un plugin définit une ou plusieurs classes de plugin. Une classe de plugin joue le rôle d'un présentateur partiel (pP - plugin présentateur) car il implémente les rappels nécessaires par la vue partielle associée dans la fente (pV - plugin vue), et le MVP est complété par le présentateur de parent (SP), agissant à nouveau comme modèle. Pour résumer et remplir :

— SP sert de modèle,
— la vue partielle pV est définie dans un fichier glade.
— les rappels mis en œuvre par pP sont référencées par le fichier glade.
— un menu contextuel pour la ligne SP,
— une propriété children.

Lorsque vous vous inscrivez à une classe de plugin, le SP :

— Ajoute le pV dans la fente et le rend invisible.
— Ajoute une instance de pP dans les classes de plugin enregistrés.
— raconte le pP que la SP est le modèle.
— se connecte tous les rappels de pV à pP.

Lorsqu'un élément dans la pV de déclencher une action en pP, le pP peut transmettre l'action au SP et peut demander des SP qu'il met à jour le modèle et actualise l'affichage.

Lorsque l'utilisateur sélectionne une ligne dans SP, SP cache tout dans la fente pluggable montre seulement le seul pV par rapport au type de la ligne sélectionnée et demande à la pP pour actualiser le pV avec tout ce qui est relatif à la ligne sélectionnée.

En dehors de définir la visibilité du pV divers, rien doit être désactivé ni supprimé : un pV invisible ne peut pas déclencher des événements !

7.1.11 flux de travail résolution de bogue

flux de travail de développement normal

— Lorsque vous utilisez le logiciel, vous remarquez un problème, ou vous faire une idée de quelque chose qui pourrait être mieux, vous en pensez assez bon pour avoir une idée très claire de ce qu'il est vraiment, que vous avez remarqué. vous ouvrez un problème et décrivez le problème. quelqu'un pourrait réagir avec des notes.
— vous ouvrez le site de questions et choisissez celui que vous souhaitez aborder.
— assignez la question à vous-même, de cette façon que vous êtes informer le monde que vous avez l'intention d'y travailler. quelqu'un pourrait réagir avec des notes.
— éventuellement la fourche le dépôt dans votre compte et de préférence créer une branche, clairement liée à la question.
— écrire des tests unitaires et elles s'engagent à votre succursale (s'il vous plaît ne pas pousser à défaut des tests unitaires à github, exécutez tout d'abord localement : `nosetests`).
— écrire des tests unitaires plus (idéalement, les tests constituent une description complète de la fonctionnalité que vous souhaitez ajouter ou corriger).
— Assurez-vous que la fonctionnalité vous ajoutez ou correction est vraiment complètement décrite par les tests unitaires, que vous avez écrit.
— Assurez-vous que vos tests unitaires sont atomiques, c'est-à-dire que vous testiez variations sur changements le long d'une seule variable. ne donnez pas de complexe d'entrée pour les tests unitaires ou les tests qui ne tiennent pas sur un seul écran (25 lignes de code).

— écrivez le code qui rend vos tests à réussir.
— mettre à jour les fichiers i18n (exécuter `./scripts/i18n.sh`).
— la mesure du possible, traduire les nouvelles chaînes que vous mettez dans les fichiers de code ou de la clairière.
— Lorsque vous changez de chaînes, veuillez vous assurer que les anciennes traductions obtenir réutilisées.
— Validez vos modifications.
— pousser sur github.
— Ouvrez une demande de tirer.

publication à la 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 :

— Ouvrez la traction demande page utilisant comme base une ligne de production `ghini-x.y`, par rapport à `ghini-x.y-dev`.
— Assurez-vous qu'un commit `bump` est inclus dans les différences.
— Il devrait être possible de fusionner automatiquement les branches.
— Créez la nouvelle demande de tirer, appelez-le « publier à la chaîne de production ».
— éventuellement, vous devez attendre pour travis-ci effectuer les contrôles.
— fusionner les modifications.

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.

étape de clôture

— revue de ce flux de travail. Considérez ceci comme une ligne directrice, à vous-même et à vos collègues. s'il vous plaît aider à faire mieux et correspondant à la pratique.

7.1.12 Distributing ghini.desktop

Python Package Index - PyPI

This is not much mentioned, but we keep ghini.desktop on the Python Package Index, so you could install it by no more than :

```
pip install ghini.desktop
```

There are a couple packages that can't be installed with `pip`, but otherwise that's really all you need to type, and it's platform independent.

Publishing on PyPI is a standard `setup` command :

```
python setup.py sdist --formats zip upload -r pypi
```

Windows

For building a Windows installer or executable you need a running Windows system. The methods described here has been used successfully on Windows 7, 8 and 10. Windows Vista should also work but has not been tested.

If you are on GNU/Linux, or on OSX, you are not interested in the remainder of this section. None of Ghini's contributors knows how to produce a Windows installer without having a Windows system.

The goal of the present instructions is to help you produce a Windows installer, that is a single executable that you can run on any Windows workstation and that will install a specific version of ghini.desktop. This is achieved with the NSIS script-driven installer authoring tool.

As a side product of the installer production, you will have a massive but relocatable directory, which you can copy to a USB drive and which will let you use the software without needing an installation.

The files and directories relevant to this section :

- `scripts/build-win.bat` — the single batch script to run.
- `setup.py` — implements the NSIS and py2exe commands.
- `scripts/build-multiuser.nsi` — the nsis script, used by the above.
- `nsis/` — contains redistributable NSIS files, put here for conveniency.
- `ghini-runtime/` — built by `py2exe`, used by `nsis`.
- `dist/` — receives the executable installation file.

Most steps are automated in the `build-win.bat` script. Installation of a few tools needs to be done manually :

1. Download and install Git, Python 2.7 and PyGTK.
   This is outlined in the `devinstall`-based installation instructions.
2. Download and install NSIS v3.
3. A **reboot** is recommended.

4. Clone the ghini.desktop repository.

   Use your own fork if you plan contributing patches, or the organization's repository `https://github.com/Ghini/ghini.desktop.git` if you only wish to follow development.

   Clone the repository from GitHub to wherever you want to keep it, and checkout a branch. Replace `<path-to-keep-ghini>` with the path of your choice, e.g. `Local\github\Ghini\`. Production branch `ghini-1.0` 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 <ghini.desktop repository URL>
cd ghini.desktop
git checkout ghini-1.0
```

The result of the above is a complete development environment, on Windows, with NSIS. Use it to follow development, or to propose your pull requests, and to build Windows installers.

All subsequent steps are automated in the `scripts\build_win.bat` script. Run it, and after a couple of minutes you should have a new `dist\ghini.desktop-<version>-setup.exe` file, and a working, complete relocatable directory named `ghini-runtime`.

Read the rest if you need details about the way the script works.

The `build_win.bat` script

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` — executable only.

   Produce an executable only, skipping the extra step of building an installer, and will copy `win_gtk.bat` into place.

2. `venv_path` — A path to the location for the virtual environment to use.

   Defaults to `"%HOMEDRIVE%%HOMEPATH%"\.virtualenvs\%CHECKOUT%-exe`, where `CHECKOUT` corresponds to the name of the branch you checked out.

If you want 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`

---

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 :

— Install using `pip`. 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.

The included `build-win` script uses the `pip` method.

---

installing virtualenv and working with environments

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.)

Populate the virtual environment

Install dependencies and ghini.desktop into the virtual environment :

```
pip install psycopg2 Pygments py2exe_py2
pip install .
```

Compile for Windows

Build the executable :

```
python setup.py py2exe
```

The `ghini-runtime` folder will now contain a full working copy of the software in a frozen, self contained state.

This folder is what is packaged by NSIS.

This same folder can also 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).

Fixing paths to GTK components.

If you run the relocatable compiled program, unpackaged, you might occasionally have trouble with the GUI not displaying correctly.

Should this happen, you need to set up paths to the GTK components correctly. You can do this by running the `win_gtk.bat`, from the `ghini-runtime` folder.

You will only need to run this once each time the location of the folder changes. Thereafter `ghini.exe` will run as expected.

Finally, invoke NSIS

Build the installer :

```
python setup.py nsis
```

This should leave a file named `ghini.desktop-<version>-setup.exe` in the `dist` 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

Debian

Between 2009 and 2010 someone packaged the then already obsolete Bauble 0.9.7 for Debian, and the package was included in Ubuntu. That version is still being distributed, regardless being it impossible to install.

Only recently has Mario Frasca produced a new bauble debian package, for the latest bauble.classic version 1.0.56, and proposed for inclusion in Debian. View it on mentors. This version depends on `fibra`, a package that was never added to Debian and which Mario also has packaged and proposed for inclusion in Debian. Mario has been trying to activate some Debian Developer, to take action. There's not much more we can do, other than wait for a sponsor, and hoping the package will eventually get all the way to Ubuntu.

Once we get in contact with a Debian Sponsor who will review what we publish on mentors, then we will be definitely expected to keep updating the debian package for `ghini.desktop` and `fibra`.

I am not going to explain in a few words the content of several books on Debian packaging. Please choose your sources. For a very compact idea of what you're expected to do, have a look at `scripts/pubish.sh`.

7.2 Template Letters

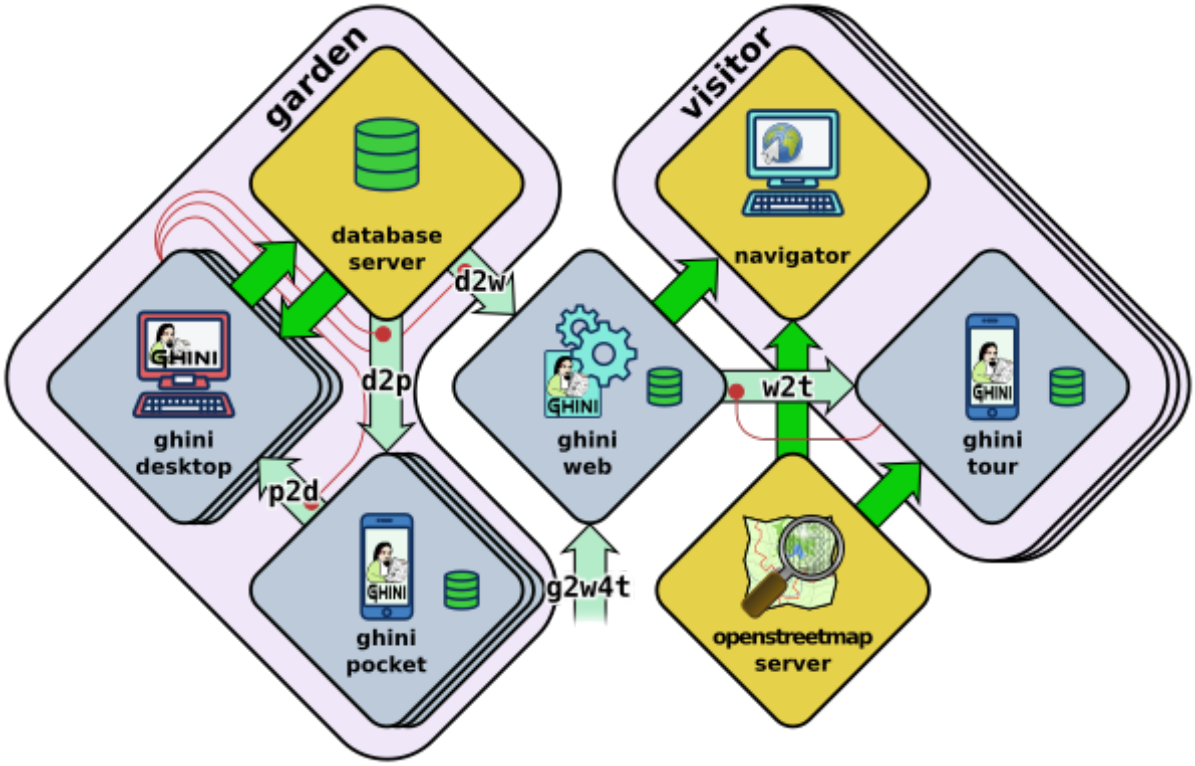
The reader getting to this point in the documentation probably understood that this Ghini project is above all a very open and collaborative project.

Here in this page you find some template letters, used to welcome new users, or that you can correct, print, and go with it to a garden, and propose them to adopt Ghini, or share with a group of your local friends, so you can make Ghini become a (voluntary, or paid) part-time job for you.

7.2.1 Dear conservator or scientist,

You are reading Ghini's presentation letter. Ghini is a libre software project on GitHub, focusing on botany. Brought to you by a small community of coders, botanists, translators, and supported by a few institutions around the world, among which, gardens that have adopted it for all their collection management needs.

The Ghini family is a software suite composed of standalone programs, data servers and hand-held clients, for data management, and publication :



— Ghini's core, `ghini.desktop`, lets you
  — enter and correct your data
  — navigate its links,
  — produce reports
  — import and or export using several standard or ad-hoc formats
  — review your taxonomy using online sources

  all according best practices suggested by top gardens, formalized in standard formats like ABCD, ITF2, but also as elaborated by our developers, based on the feedback of Ghini users.

  `ghini.desktop` is developed and continously tested on GNU/Linux, but runs equally well on Windows, or OSX. [1]
— `ghini.pocket` is your full time garden companion, an Android app installed from the Play Store,
  — assisting you in collecting or correcting data while in the field,
  — associate pictures to your plants, and verify taxonomic information.
  — Import your collected data into the desktop client when back in the office,

  `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 handle it for publication to the Ghini project, and we will include it at http://gardens.ghini.me/, at no cost while we're able to do that, or for a guaranteed minimal amount of time if you are able to support our hosting costs. `ghini.web` serves a world map to help locate participating gardens, and within each garden, its contributed georeferenced plants.
— `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`.

All software within the Ghini family is either licensed GNU Public License v2+ or v3+. It is a strong copyleft license. 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.

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.

Thanks for your consideration ; please let me know if you have any questions,

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

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.

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. `ghini.desktop` has three more components, a pocket data collecting Android app, a Node.js web server, aggregating data from different gardens and presenting it geographically, again 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. Specify 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 such a complex task. Specify is also on GitHub : https://github.com/specify and is licensed as GPLv2+.
- Botalista, a French/Swiss cooperation, is GPL as far as rumours go. Its development has yet to go public.
- `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. It was Mario Frasca who convinced Eric to publish what he was doing, licensing it under the GPL, but the repository was not updated after 2016, April 13th and Eric forgot to explicitly specify the license. You find it on github : 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

CHAPITRE 8

Supporting Ghini

If you're using Ghini, or if you feel like helping its development anyway, please consider donating.