Bauble Documentation

Release 1.1.1

Brett Adams

September 03, 2018

Contents

Ghini is an application for managing botanical specimen collections. With it you can create a searchable database of plant records.

It is open and free and is released under the GNU Public License

CHAPTER 1

Statements

1.1 Highlights

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

1.1.1 taxonomic information

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

1.1.2 importing data

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

1.1.3 synonyms

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

1.1.4 scientific responsible

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

1.1.5 helps off-line identification

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

1.1.6 exports and reports

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

1.1.7 annotate your info

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

1.1.8 garden or herbarium

Management of plant locations.

1.1.9 database history

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

1.1.10 simple and powerful search

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

1.1.11 database agnostic

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

1.1.12 language agnostic

The program was born in English and all its technical and user documentation is still only in that language, but the program itself has been translated and can be used in various other languages, including Spanish (86%), Portuguese (100%), French (42%), to name some Southern American languages, as well as Swedish (100%) and Czech (100%).

1.1.13 platform agnostic

Installing Ghini on Windows is an easy and linear process, it will not take longer than 10 minutes. Ghini was born on Linux and installing it on ubuntu, fedora or debian is also rather simple. It has been recently successfully tested on MacOSX 10.9.

1.1.14 easily updated

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

1.1.15 unit tested

Ghini is continuously and extensively unit tested, something that makes regression of functionality close to impossible. Every update is automatically quality checked, on the Travis Continuous Integration service. Integration of TravisCI with the github platform will make it difficult for us to release anything which has a single failing unit test.

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

1.1.16 customizable/extensible

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

1.2 Who is behind Ghini

Ghini started as a one-man project, by Brett Adams. He wrote Bauble for the Belize Botanical Garden, and later on adapted it for a couple of other users who asked him to use it. Brett made Bauble a commons, by releasing it under a GPL license.

After some years of stagnation Mario Frasca took responsibility of updating Bauble, renamed it to Ghini in order to mark the difference in activity level, and it is now Mario Frasca writing this, enhancing the software, looking for users, requesting feedback...

So even if currently behind Ghini there's again one developer, much more importantly, there is a small but growing global users community.

Translations are provided by volunteers who mostly stay behind the scenes, translating a couple of 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.

1.3 Mission

To continuously support open and free software for the documentation, research and management of biodiversity collections, for the benefit of organizations and educational institutions that have or manage botanical collections. For example botanical gardens, arboretums, herbaria, but also for people studying or working with such collections.

To stimulate users to contributing to the improvement of the software and sharing solutions; to enhance the availability and visibility of such biodiversity collections.

1.4 Vision

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

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

CHAPTER 2

Installing Ghini

2.1 Introduction

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

To install Ghini first requires that you install its dependencies that cannot be installed automatically. These include virtualenvwrapper, PyGTK and pip. Python and GTK+, you probably already have. As long as you have these packages installed then Ghini should be able to install the rest of its dependencies by itself.

Note: If you follow these installation steps, 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.

if you later choose to remove Ghini, you simply remove the virtual environment, which is a directory, with all of its content.

2.2 Installing on Linux

1. Download the *devinstall.sh* script and run it:

```
https://raw.githubusercontent.com/Ghini/ghini.desktop/master/
↪scripts/devinstall.sh
```

   Please note that the script will not help you install any extra database connector. This you will do in a later step.

Note: (technical) 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*, download the sources and connect your git checkout to the *ghini-1.0* branch (this you can consider a production line), it then builds ghini, downloading all remaining dependencies, and finally it creates a startup script in your *~/bin* folder.

Note:

(beginner) 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.

If the script ends without error, you can now start ghini:

```
~/bin/ghini
```

or update ghini to the latest released production patch:

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

The same script you can use to switch to a different production line, but at the moment there's only *ghini-1.0*.

1. on Unity, open a terminal, start ghini, its icon will show up in the launcher, you can now *lock to launcher* it.
2. If you would like to use the default SQLite database or you don't know what this means then you can skip this step. If you would like to use a database backend other than the default SQLite backend then you will also need to install a database connector.

   If you would like to use a PostgreSQL database then activate the virtual environment and install psycopg2 with the following commands:

   ```
   source ~/.virtualenvs/ghide/bin/activate
   pip install -U psycopg2
   ```

   You might need solve dependencies. How to do so, depends on which Linux flavour you are using. Check with your distribution documentation.

Next...

Connecting to a database.

2.3 Installing on MacOSX

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

One difficulty is that there are many more versions of MacOSX out there than one would want to support, and only the current and its immediately preceding release are kept up-to-date by Apple-the-firm.

Last time we tested, some of the dependencies could not be installed on MacOSX 10.5 and we assume similar problems would present themselves on older OSX versions. Ghini has been successfully tested with 10.7 and 10.9.

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).

with the above installed, 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.

the rest is just as on a normal unix machine, and we have a *devinstall.sh* script for it. Read the above Linux instructions, follow them, enjoy.

Next...

Connecting to a database.

2.4 Installing on Windows

The current maintainer of ghini.desktop has no interest in learning how to produce Windows installers, so the Windows installation is here reduced to the same installation procedure as on Unix (Linux and MacOSX).

Please report any trouble. Help with packaging will be very welcome, in particular by other Windows users.

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 Linux. The concluding steps are again Windows specific.

Note: Ghini has been installed and is known to work fine on W-XP, W-7, W-8 and W-10. However, Windows is not Ghini's development and test platform, so please report any problem you might encounter. During the installation spurious error messages can be expected and safely ignored.

Note: Direct download links are given for all needed components. They have been tested in September 2015, but things change with time. If any of the direct download links stops working, please ring the bell, so we can update the information here.

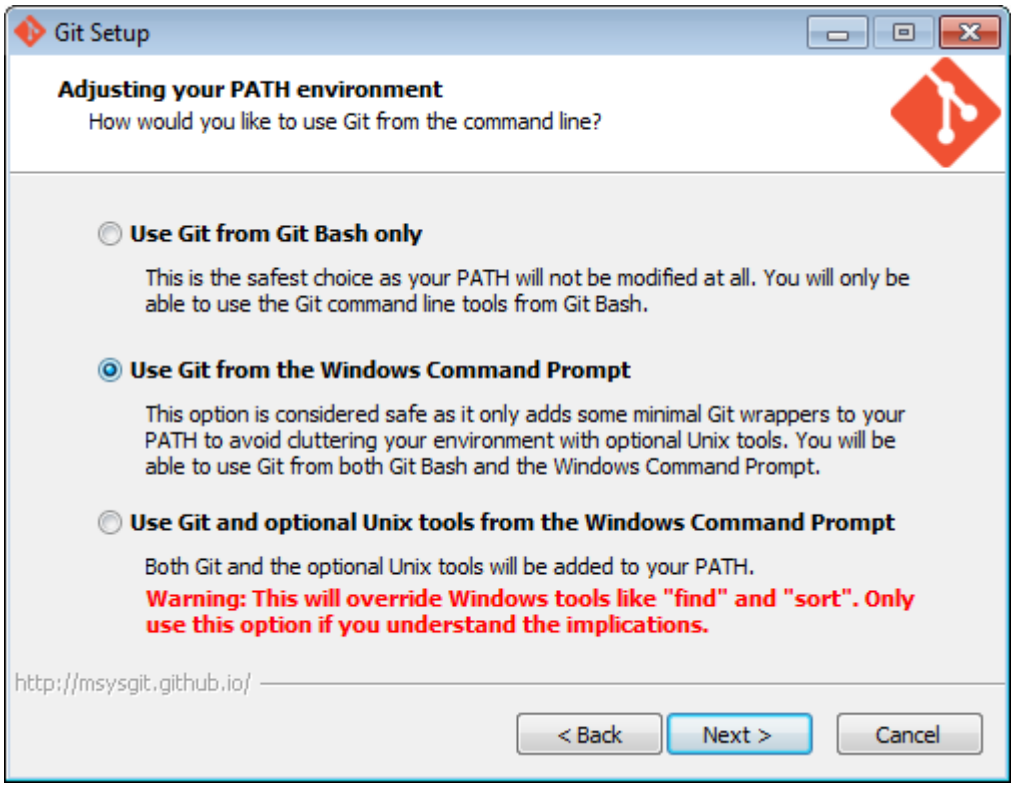
The installation steps on Windows:

1. download and install `git` (comes with a unix-like `sh` and includes `vi`) from:

   https://git-scm.com/download/win

   Direct link to download git

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



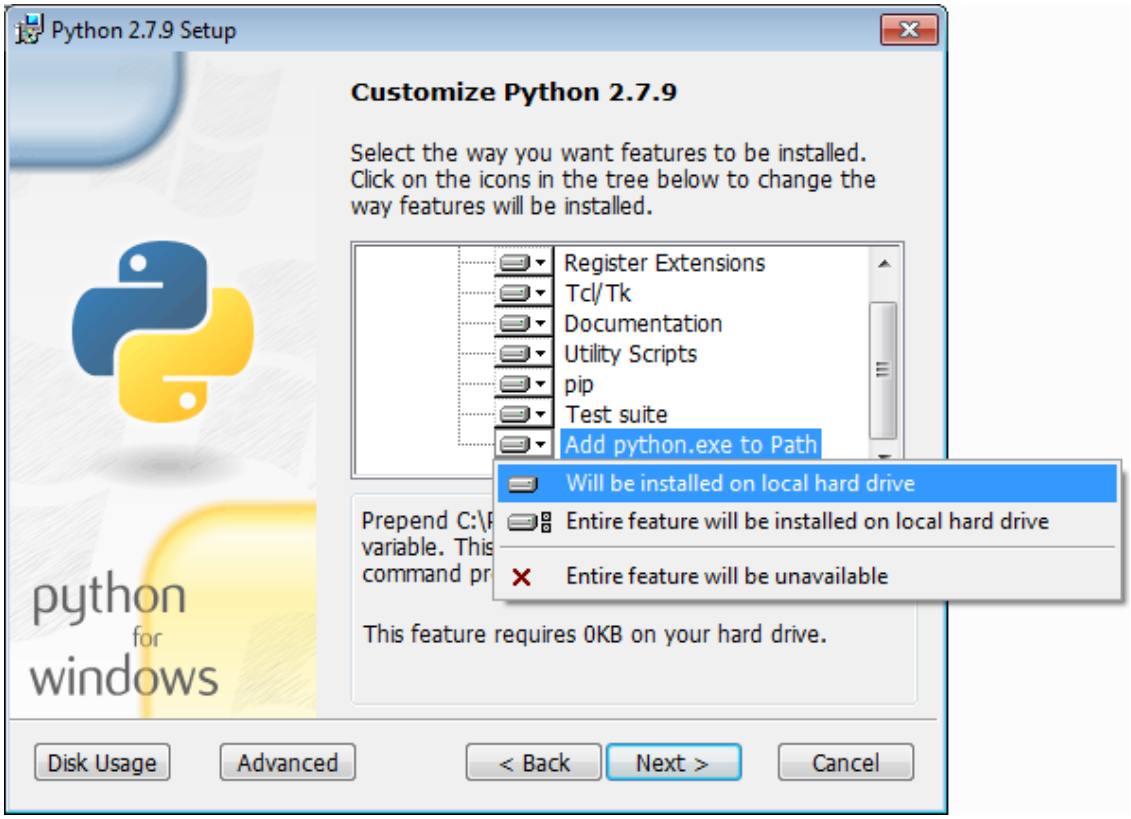
2. download and install Python 2.x (32bit) from:

http://www.python.org

Direct link to download Python

Ghini has been developed and tested using Python 2.x. It will definitely **not** run on Python 3.x. If you are interested in helping port to Python 3.x, please contact the Ghini maintainers.

when installing Python, do put Python in the PATH:

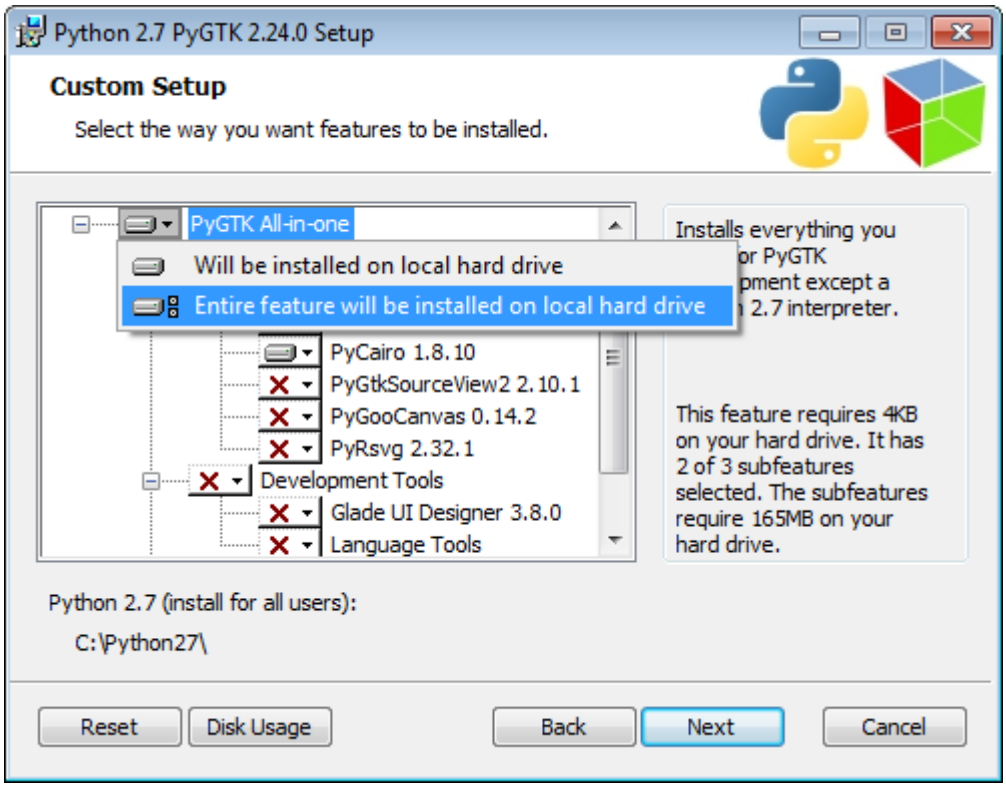


3. download `pygtk` from the following source. (this requires 32bit python). be sure you download the "all in one" version:

   http://ftp.gnome.org/pub/GNOME/binaries/win32/pygtk/

   Direct link to download PyGTK

   make a complete install, selecting everything:



4. (Windows 8.x) please consider this additional step. It is possibly necessary to avoid the following error on Windows 8.1 installations:

```
Building without Cython.
ERROR: 'xslt-config' is not recognized as an internal or␣
 ↪external command,
operable program or batch file.
```

   If you skip this step and can confirm you get the error, please inform us.

   You can download lxml from:

```
https://pypi.python.org/pypi/lxml/3.6.0
```

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

   Direct link to download lxml

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

   On Windows, it is NOT easy to install `psycopg2` from sources, using pip, so "avoid the gory details" and use a pre-compiled pagkage from:

   http://initd.org/psycopg/docs/install.html

   Direct link to download psycopg2

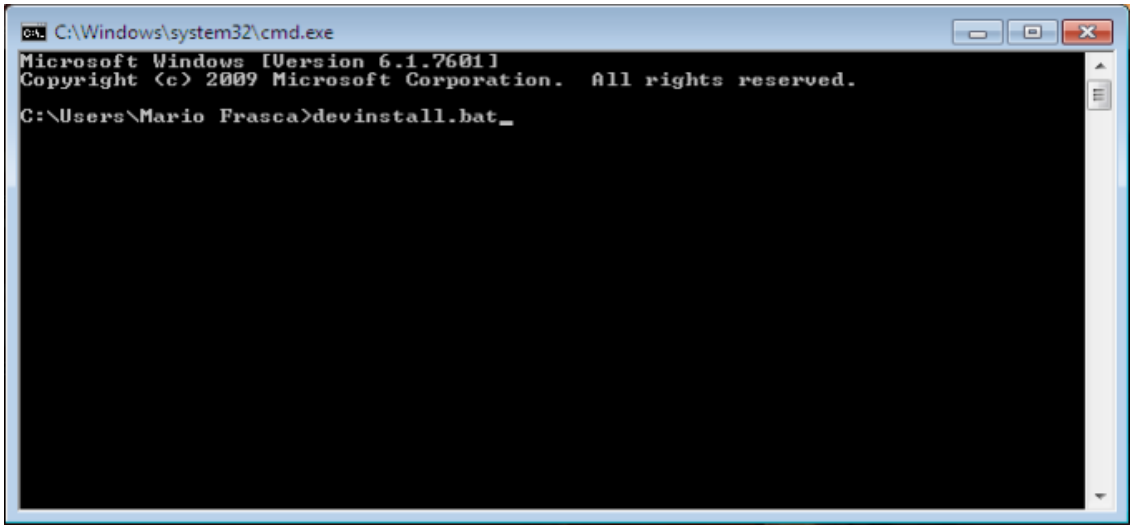
6. **REBOOT**

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

7. download and run (from `\system32\cmd.exe`) the batch file:

https://raw.githubusercontent.com/Ghini/ghini.desktop/master/scripts/devinstall.bat

right before you hit the enter key to run the script, your screen might look like something like this:



this will pull the `ghini.desktop` repository on 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 you will use to stay up-to-date with the production line you chose to follow:

   https://raw.githubusercontent.com/Ghini/ghini.desktop/master/scripts/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 start the command prompt and run `ghini-update.bat`

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

Next...

Connecting to a database.

2.5 Troubleshooting

1. any error related to lxml.

   In order to be able to compile lxml, you have to install a C compiler (on Linux this would be the `gcc` package) and Cython (a Python specialization, that gets compiled into C code. Note: Cython is not CPython).

   However, It should not be necessary to compile anything, and `pip` should be able to locate the binary modules in the online libraries.

   For some reason, this is not the case on Windows 8.1.

   https://pypi.python.org/pypi/lxml/3.6.0

   Please report any other trouble related to the installation of lxml.

2. Couldn't install gdata.

   For some reason the Google's gdata package lists itself in the Python Package Index but doesn't work properly with the easy_install command. You can download the latest gdata package from:

   http://code.google.com/p/gdata-python-client/downloads/list

   Unzip it and run `python setup.py installw` in the folder you unzip it to.

Next...

Connecting to a database.

CHAPTER 3

Using Ghini

3.1 Getting Started

3.1.1 Should you SQLite?

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

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

On the other hand, if you want to connect more than one ghini 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 create: at least one ghini user, the database you want ghini to use, and to give at lest one ghini user full permissions on its database. 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.

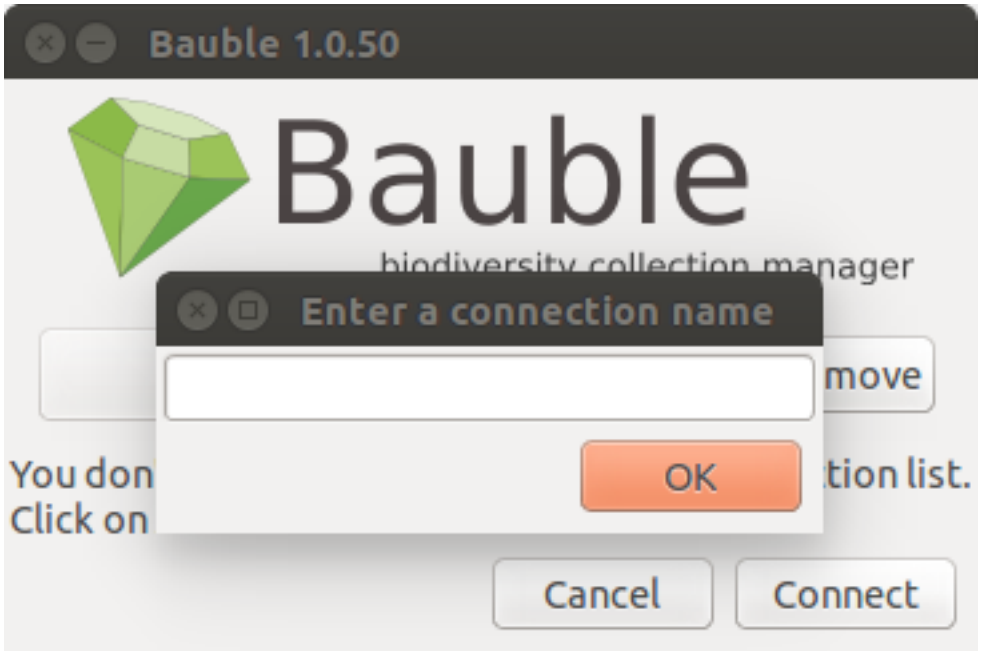
If you already got the chills or sick at your stomach, no need to worry, just stick with SQLite, you do not miss on features nor performance.

3.1.2 Connecting to a database

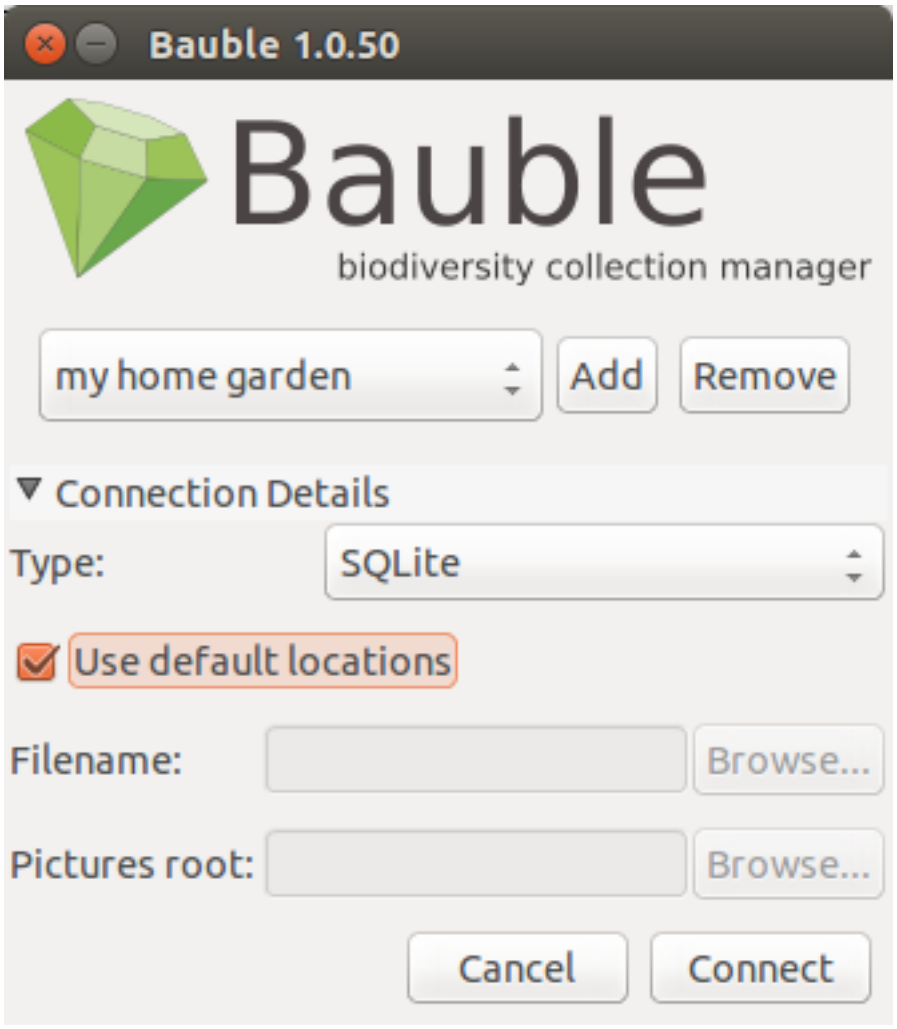
When you start Ghini the first thing that comes up is the connection dialog.

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

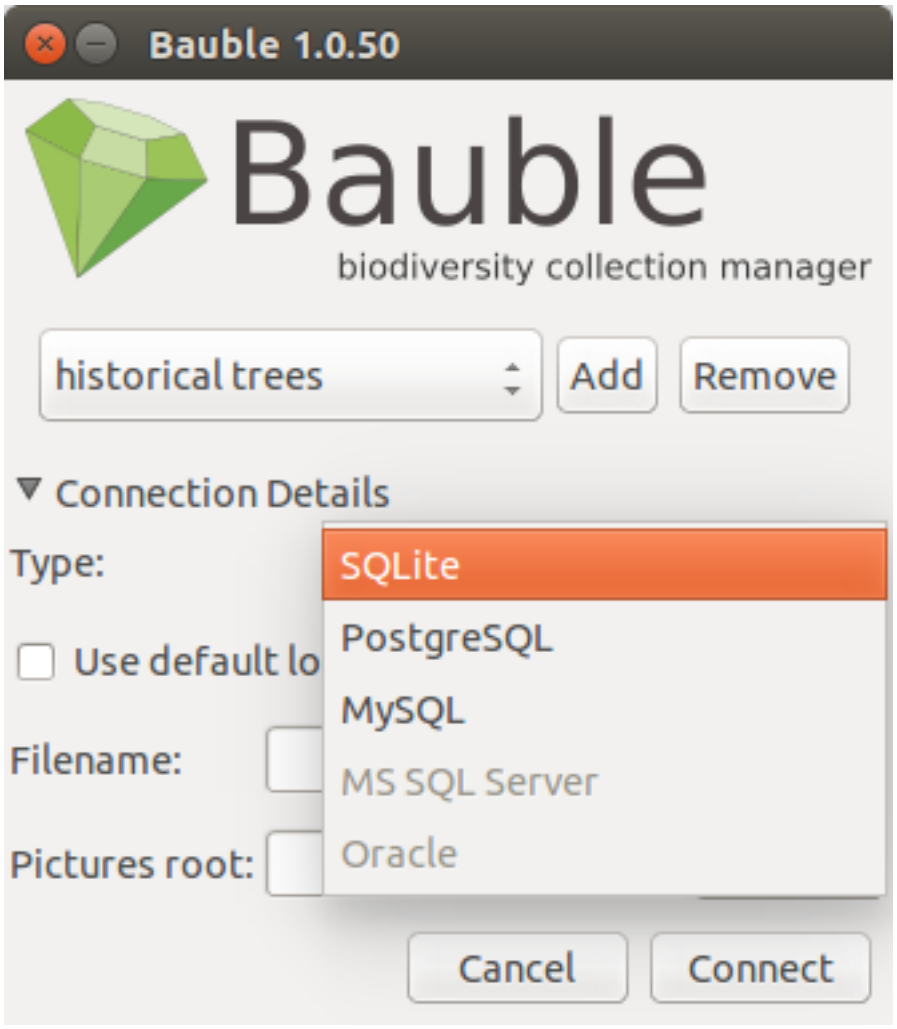
If you do not know what to do here, Ghini will help you stay safe. Activate the **Use default locations** check box and create your first connection by clicking on **Connect**.

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 `~/.ghini` on Linux/MacOSX or in `AppData\Roaming\Ghini` on Windows.

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

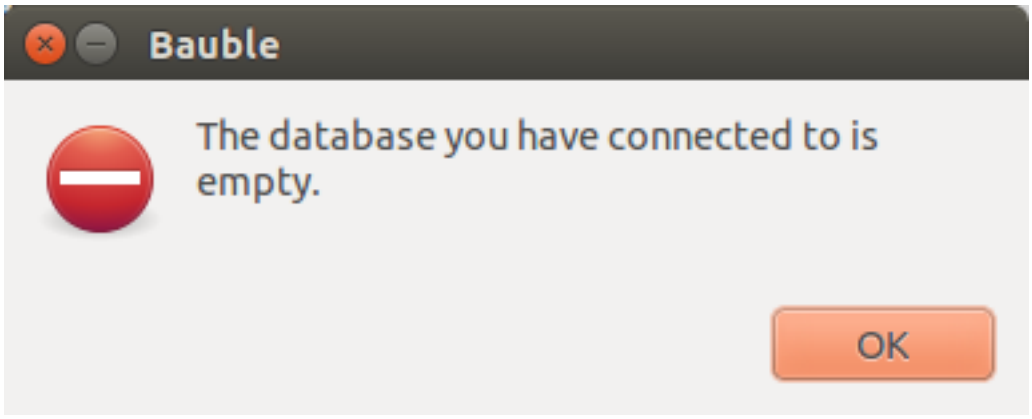
If you use a different database connector, the dialog box will look different and it will offer you the option to fine tune all parameters needed to connect to the database of your choice.

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

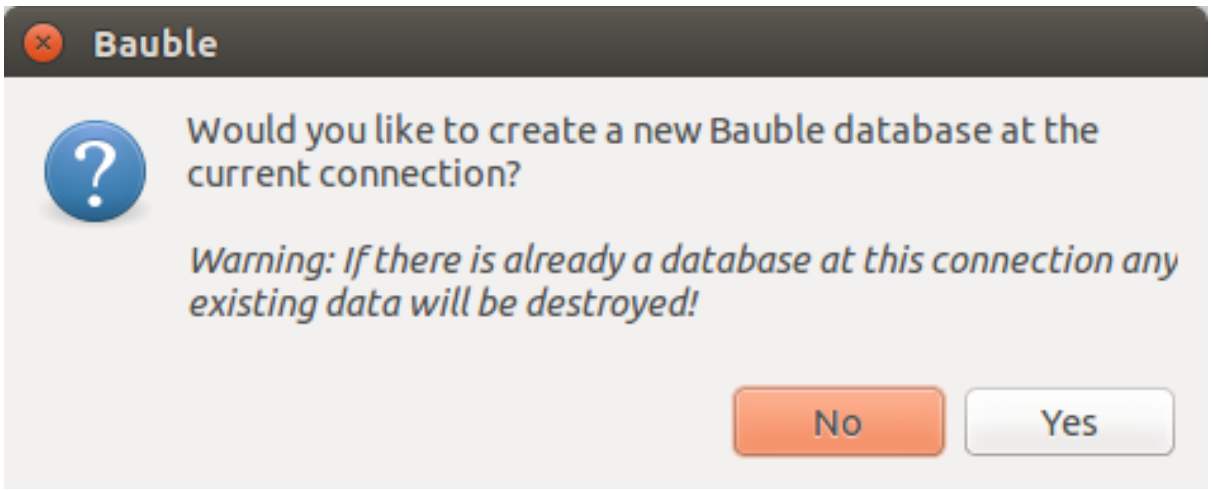
If you plan to associate pictures to plants, specify also the *pictures root* folder. The meaning of this is explained in further detail at *Pictures* in *Editing and Inserting Data*.

3.1.3 Initialize a database

First time you open a connection to a database which had never been seen by Ghini before, Ghini will first display an alert:



immediately followed by a question:



Be careful when manually specifying the connection parameters: the values you have entered may refer to an existing database, not intended for use with Ghini. By letting Ghini initialize a database, the database will be emptied and all of its content be lost.

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

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

3.2 Searching in Bauble

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 Bauble searches are listed in the main window.

3.2.1 Search Strategies

Three are three types of search strategies available in Bauble. Considering the search stragety types available in Bauble, sorted in increasing complexity: you can search by value, expression or query.

Searching by query, the most complex and powerful, is assisted by the Query Builder, described below.

All searches are case insensitive so searching for Maxillaria and maxillaria will return the same results.

Search by Value

Search by value is the simplest way to search. You just type in a string and see what matches. Which fields/columns are search for your string depends on how the different plugins are configured. For example, by default the PlantPlugin search the family name, the genus name, the species and infraspecific species names, vernacular names and geography. So if you want to search in the notes field of any of these types then searching by value is not the search you're looking for.

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

Search string are separated by spaces. For example if you enter the search string `Block 10` then Bauble 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 a while string then you should quote the string like `"Block 10"`.

Search by Expression

Searching with expression gives you a little more control over what you are searching for. It can narrow the search down to a specific domain. Expression consist of a domain, an operator and a 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 search string `gen like max%` would return all the genera whose names start with "Max". In this case the domain again is gen, the operator is like, which allows for "fuzzy" searching and the value is max%. The percent sign is used as a wild card so if you search for max% then it search for all value that start with max. If you search for %max it searches for all values that end in max. The string %max%a would search for all value that contain max and end in a.

For more information about the different search domain and their short-hand aliases, see *search-domains* .

If expression are invalid they are usually used as search by value searchs. For example the search string `gen=` will execute a search by value for the string gen and the search string `gen like` will search for the string gen and the string like.

Search by Query

Queries allow the most control over searching. With queries you can search across relations, specific columns and join search using boolean operators like AND and OR.

An example of a query would be:

```
plant where accession.species.genus.family=Fabaceae and location.
↪site="Block 10"
```

This query would return all the plants whose family are Fabaceae and are located in Block 10.

Searching with queries usually requires some knowledge of the Bauble internals and database table layouts.

A couple of useful examples:

- Which locations are in use:

  ```
  location where plants.id!=0
  ```

- Which genera are associated to at least one accession:

  ```
  genus where species.accession.id!=0
  ```

Domains

The following are the common search domain and the columns they search by default. The default columns are used when searching by value and expression. The queries do not use the default columns.

Domains family, fam: Search `bauble.plugins.plants.Family`

genus, gen: Search `bauble.plugins.plants.Genus`

species, sp: Search `bauble.plugins.plants.Species`

geography: Search `bauble.plugins.plants.Geography`

acc: Search `bauble.plugins.garden.Accession`

plant: Search `bauble.plugins.garden.Plant`

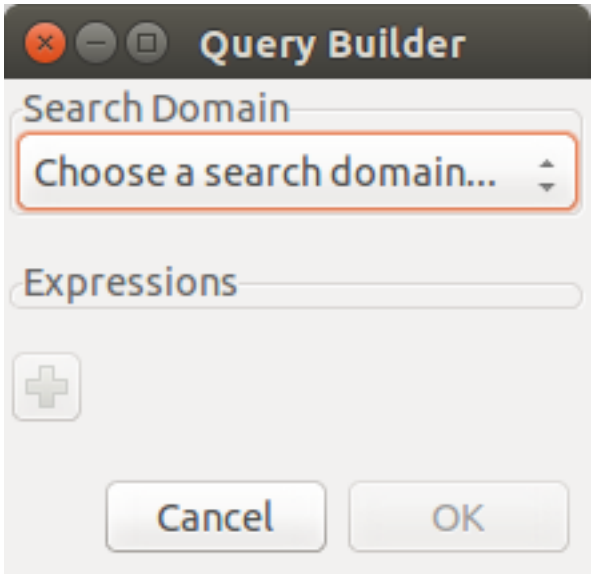
location, loc: Search `bauble.plugins.garden.Location`

3.2.2 The Query Builder

The Query Builder helps you build complex search queries through a point and click interface. To open the Query Builder click the to the left of the search entry or select *Tools→Query Builder* from the menu.

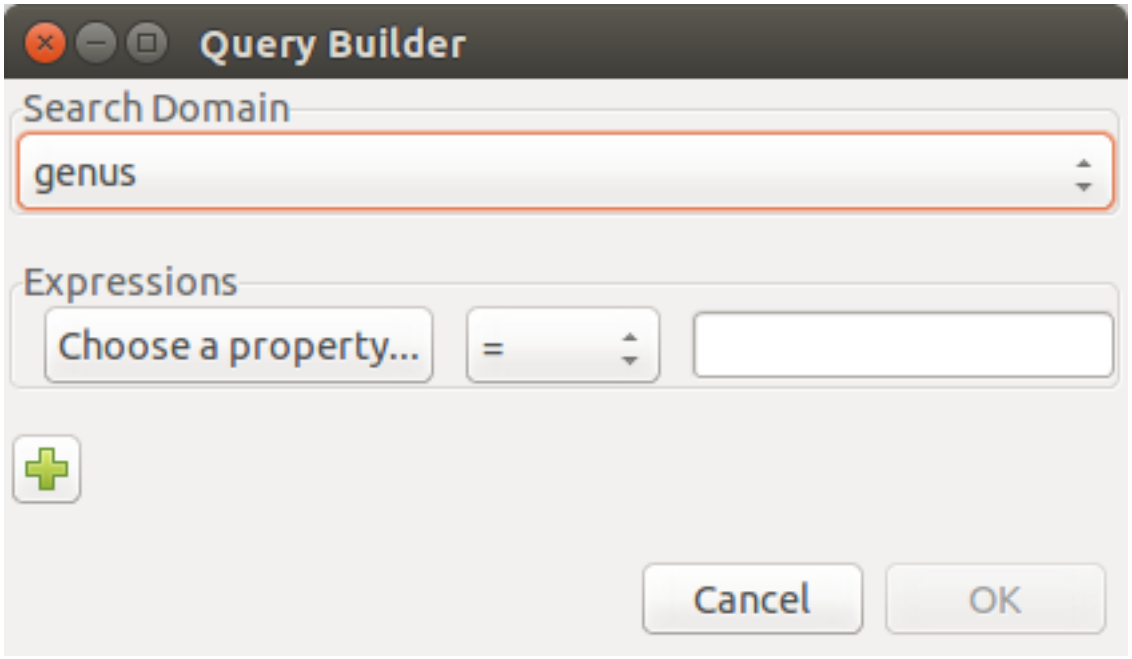
The Query Builder composes a query that will be understood by the Query Search Strategy described above. You can use the Query Builder to get a feeling of correct queries before you start typing them by hand, something that you might prefer if you are a fast typer.

After opening the Query Builder you must select a search domain. The search domain will determine the type of data that is returned and the properties that you can search.



The search domain is similar to a table in the database and the properties would be the columns on the table. Often the table/domain and properties/columns are the same but not always.

Once a search domain is selected you can then select a property of the domain to compare values to. The search operator can then be changed for how you want to make the search comparison. Finally you must enter a value to compare to the search property.



If the search property you have selected can only have specific values then a list of possible values will be provided for you to choose from.

If multiple search properties are necessary then clicking on the plus sign will add more search properties. Select And/Or next to the property name choose how the properties will be combined in the search query.

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

3.3 Editing and Inserting Data

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

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

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

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

3.3.1 Notes

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

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

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

3.3.2 Family

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

The *Family* field on the editor will change the name of the family. The Family field is required.

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

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

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

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

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

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

3.3.3 Genus

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

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

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

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

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

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

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

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

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

3.3.4 Species/Taxon

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

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

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

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

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

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

3.3.5 Accessions

The Accession editor allows us to add an accession to a species. In Ghini an accession represents a group of plants or clones. The accession would refer maybe a group of seed or cuttings from a species. A plant would be an individual from that accesssion, i.e. a specific plant in a specific location.

Accession Source

The source of the accessions lets you add more information about where this accession came from. At the moment the type of the source can be either a Collection or a Donation.

Collection

A Collection.

Donation

A Donation.

3.3.6 Plant

The Plant editor.

Creating multiple plants

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

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

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

Pictures

Just as almost all objects in the Ghini database can have *Notes* associated to them, Plants can have *Pictures*: next to the tab for Notes, the Plants editor contains an extra tab called "Pictures". You can associate as many pictures as you might need to a plant.

When you associate a picture to a plant, 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-1.0.58, 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 file sharing service like Copy or Dropbox. The personal choice of the writer of this document is to use Copy, because it offers much more space and because of its "Fair Storage" policy.

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

When a Plant 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.

3.3.7 Locations

The Location editor

danger zone

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

3.4 Tagging

Tagging is an easy way to give context to an object or create a collection of object that you want to recall later. For example if you want to collect a bunch of plants that you later want to create a report from you can tag them with the string "for that report i was thinking about". You can then select "for that report i was thinking about" from the tags menu to show you all the objects you tagged.

Tagging can be done two ways. By selecting one or more items in the search results and pressing Ctrl-T or by selecting *Tag→Tag Selection* from the menu. If you have selected multiple items then only that tags that are common to all the selected items will have a check next to it.

3.5 Generating reports

3.5.1 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 Bauble 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 seperated 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 *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.

3.5.2 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.

3.6 Importing and Exporting Data

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

There is some support for exporting to the Access for Biological Collections Data it is limited.

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

Exporting ABCD and XML will not be covered here.

> **Warning:** Importing files will most likely destroy any data you have in the database so make sure you have backed up your data.

3.6.1 Importing from CSV

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

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

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

3.6.2 Exporting to CSV

To export the Ghini data to CSV select *Tools→Export→Comma Seperated 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.

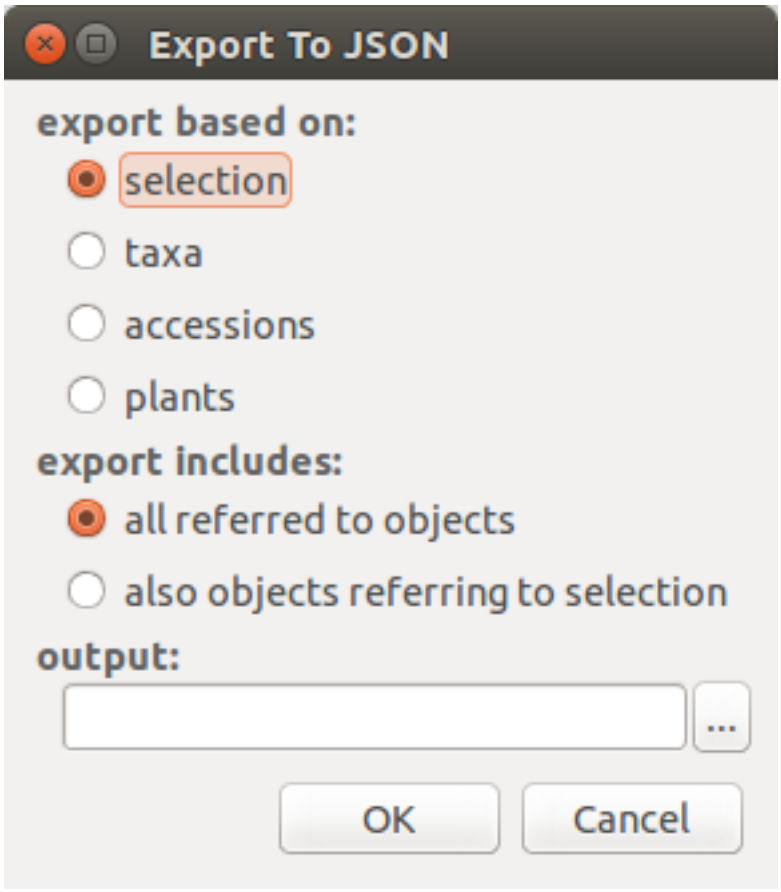
3.6.3 Importing from JSON

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

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

3.6.4 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 Managing Users

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

The Bauble User's Plugin will allow you to create and manage the permissions of users for your Bauble database.

3.7.1 Creating Users

To create a new user. . .

3.7.2 Permissions

Bauble allows read, write and execute permissions.

CHAPTER 4

Administration

4.1 Database Administration

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

4.1.1 SQLite

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

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

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

4.1.2 MySQL

Please refer to the official documentation.

4.1.3 PostgreSQL

Please refer to the official documentation. A very thorough discussion of your backup options starts at chapter_24.

4.2 Ghini Configuration

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

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

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

4.3 Reporting Errors

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

Ghini development site can be accessed via the Help menu.

CHAPTER 5

Ghini Development

5.1 Downloading the source

The Ghini source can be downloaded from our source repository on github.

If you want a particular version of Ghini, we release and maintain versions into branches. You should `git checkout` the branch corresponding to the version of your choice. Branch names for Ghini versions are of the form `ghini-x.y`, where x.y can be 1.0, for example. Our workflow is to commit to the *master* development branch or to a *patch* branch and to include the commits into a *release* branch when ready.

To check out the most recent code from the source repository you will need to install the Git version control system. Git is incuded in all reasonable Linux distributions and can be installed on all current operating systems.

Once you have installed Git you can checkout the latest Ghini code with the following command:

```
git clone https://github.com/Ghini/ghini.desktop.git
```

For more information about other available code branches go to ghini.desktop on github.

5.2 Development Workflow

5.2.1 production line

A ghini production line is a branch. Currently there is only one production line, that is ghini-1.0. In perspective, we will have several one, each in use by one or more gardens.

As long as we have only one production line, I keep working on the master branch, unless I later realize the work is going to take longer than one or two days.

5.2.2 batches of simple issues

For issues that can be managed in one or two commits, and as long as there's no other activity on the repository, work on the master branch, accumulate issue-solving commits, finally merge master into the production line ghini-1.0.

5.2.3 larger issues

When facing a single larger issue, create a branch tag, and follow the workflow described at

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

in short:

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

work on the new branch. When ready, go to github, merge the branch with master, solve conflicts where necessary, delete the temporary branch.

when ready for publication, merge master into the production line.

5.3 Setting up the testing environment

In Ghini, a developer installation starts with our installation procedure for a standard user installation. What you still need to set up to start contributing quality code is a decent editor, and the testing environment.

So first choose a decent editor, and here opinions vary and all are equally valuable, here we describe how to set up `nose`, the testing environment at the base of our unit test suites.

A standard user installation gets you Ghini installed in a virtual environment, this virtual environment is enough for running the program, but misses two modules for unit testing: `nose` and `coverage`. You simply need activate the environment, and install the them:

```
``. ~/.virtualenv/ghide/bin/activate``
``pip install coverage nose -I``
```

the `-I` option is necessary to make sure that the two modules get installed in the virtual environment, whether they are already in your global installation or not.

at this point you should be able to run the test suite:

```
``cd ~/Local/github/Ghini/ghini.desktop/``
``. ~/.virtualenv/ghide/bin/activate``
``./scripts/update-coverage.sh``
```

5.4 Adding missing unit tests

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

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

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

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

5.4.1 What to test

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

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

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

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

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

5.4.2 how to test

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

parameter correctness

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

cascade

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

confirm

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

deleting

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

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

5.4.3 where to put the tests

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

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

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

5.4.4 writing the tests

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

```
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
```

5.4.5 One test, step by step

Let's start with the first test case.

When writing tests, I generally follow the pattern:

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

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

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

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

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

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

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

Next we test the result.

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

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

Let's first define a rather generic function:

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

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

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

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

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

```
# 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])
```

5.4.6 And so on

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

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

With this action, the value returned by `remove_callback` should be `True`, and there should

be no Arecaceae Family in the database any more:

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

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

    # effect
    self.assertFalse('message_details_dialog' in
                     [f for (f, m) in self.invoked])
    self.assertTrue(('yes_no_dialog', u'Are you sure you want to '
                     'remove the family <i>Arecaceae</i>?')
                    in self.invoked)
    self.assertEquals(result, True)
    q = self.session.query(Family).filter_by(family=u"Arecaceae")
    matching = q.all()
    self.assertEquals(matching, [])
```

have a look at commit 734f5bb9feffc2f4bd22578fcee1802c8682ca83 for the other two test functions.

5.4.7 Putting all together

From time to time you want to activate the test class you're working at:

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

And at the end of the process you want to update the statistics:

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

5.5 Plugins structure

Ghini is a framework for handling collections, and is distributed along with a set of plugins making Ghini a botanical collection manager. But Ghini stays a framework and you could in theory remove all plugins we distribute and write your own, or write your own plugins that extend or complete the current Ghini behaviour.

Once you have selected and opened a database connection, you land in the Search window. The Search window is an interaction between two objects: SearchPresenter (SP) and SearchView (SV).

SV is what you see, SP holds the program status and handles the requests you express through SV. Handling these requests affect the content of SV and the program status in SP.

The search results shown in the largest part of SV are rows, objects that are instances of classes registered in a plugin.

Each of these classes must implement an amount of functions in order to properly behave within the Ghini framework. The Ghini framework reserves space to pluggable classes.

SP knows of all registered (plugged in) classes, they are stored in a dictionary, associating a class to its plugin implementation. SV has a slot (a gtk.Box) where you can add elements. At any time, at most only one element in the slot is visible.

A plugin defines one or more plugin classes. A plugin class plays the role of a partial presenter (pP - plugin presenter) as it implement the callbacks needed by the associated partial view fitting in the slot (pV - plugin view), and the MVP pattern is completed by the parent presenter (SP), again acting as model. To summarize and complete:

- SP acts as model,
- the pV partial view is defined in a glade file.
- the callbacks implemented by pP are referenced by the glade file.
- a context menu for the SP row,
- a children property.

when you register a plugin class, the SP:

- adds the pV in the slot and makes it non-visible.
- adds an instance of pP in the registered plugin classes.
- tells the pP that the SP is the model.
- connects all callbacks from pV to pP.

when an element in pV triggers an action in pP, the pP can forward the action to SP and can request SP that it updates the model and refreshes the view.

When the user selects a row in SP, SP hides everything in the pluggable slot and shows only the single pV relative to the type of the selected row, and asks the pP to refresh the pV with whatever is relative to the selected row.

Apart from setting the visibility of the various pV, nothing needs be disabled nor removed: an invisible pV cannot trigger events!

5.6 Developer's Manual

5.6.1 helping ghini development

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

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

- use the software, note the things you don't like, open issue for each of them. a developer will react.
- if you have an idea of what you miss in the software but can't quite formalize it into separate issues, you could consider hiring a professional. this is the best way to make sure that something happens quickly on Ghini. do make sure the developer opens issues and publishes their contribution on github.
- translate! any help with translations will be welcome, so please do! you can do this without installing anything on your computer, just using the on-line translation service offered by http://hosted.weblate.org/
- fork the respository, choose an issue, solve it, open a pull request. see the *bug solving workflow* below.

5.6.2 bug solving workflow

normal development workflow

- while using the software, you notice a problem, or you get an idea of something that could be better, you think about it good enough in order to have a very clear idea of what it really is, that you noticed. you open an issue and describe the problem. someone might react with hints.
- you open the issues site and choose one you want to tackle.
- assign the issue to yourself, this way you are informing the world that you have the intention to work at it. someone might react with hints.
- optionally fork the repository in your account and preferably create a branch, clearly associated to the issue.
- write unit tests and commit them to your branch (do not commit failing unit tests to the `master` branch).
- write more unit tests (ideally, the tests form the complete description of the feature you are adding or correcting).
- make sure the feature you are adding or correcting is really completely described by the unit tests you wrote.

- make sure your unit tests are atomic, that is, that you test variations on changes along one single variable. do not give complex input to unit tests or tests that do not fit on one screen (25 lines of code).
- write the code that makes your tests succeed.
- update the i18n files (run `./scripts/i18n.sh`).
- whenever possible, translate the new strings you put in code or glade files.
- commit your changes.
- push to github.
- open a pull request.

publishing to production

- open the pull request page using as base the production line, compared to `master`.
- make sure a `bump` commit is included in the differences.
- it should be possible to automatically merge the branches.
- create the new pull request, call it as "publish to the production line".
- you possibly need wait for travis-ci to perform the checks.
- merge the changes.
- tell the world about it: on facebook, the google group, linkedin, ...

closing step

- review this workflow. consider this as a guideline, to yourself and to your colleagues. please help make it better and matching the practice.

5.7 Extending Ghini with Plugins

Nearly everything about Ghini is extensible through plugins. Plugins can create tables, define custom searchs, add menu items, create custom commands and more.

To create a new plugin you must extend the `bauble.pluginmgr.Plugin` class.

5.8 structure of user interface

the user interface is built according to the Model-View-Presenter architectural pattern. The **view** is described in a **glade** file and is totally dumb, you do not subclass it because it implements no behaviour and because its appearance is, as said, described elsewhere (the **glade** file), including the association signal-callbacks. The **model** simply follows the sqlalchemy practices.

You will subclass the **presenter** in order to:

- define `widget_to_field_map`, the association from name of view object to name of model attribute,
- override `view_accept_buttons`, the list of widget names which, if activated by the user, mean that the view should be closed,
- define all needed callbacks,

The presenter should not know of the internal structure of the view, instead, it should use the view api to set and query the values inserted by the user. The base class for the presenter, `GenericEditorPresenter` defined in `bauble.editor`, implements many generic callbacks.

Model and Presenter can be unit tested, not the View.

The `Tag` plugin is a good minimal example, even if the `TagItemGUI` falls outside this description. Other plugins do not respect the description.

A good example of Presenter/View pattern (no model) is given by the connection manager.

We use the same architectural pattern for non-database interaction, by setting the presenter also as model. We do this, for example, for the JSON export dialog box.

5.9 API Documentation

5.9.1 bauble

The top level module for Ghini.

bauble.**version = '1.1.1'**
str(object='') -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

bauble.**gui = None**
bauble.gui is the instance `bauble.ui.GUI`

bauble.**command_handler**(*cmd*, *arg*)
Call a command handler.

Parameters

- **cmd** (*str*) – The name of the command to call
- **arg** (*list*) – The arg to pass to the command handler

bauble.**main**(*uri=None*)
Run the main Ghini application.

Parameters **uri** (*str*) – the URI of the database to connect to. For more information about database URIs see http://www.sqlalchemy.org/docs/05/dbengine.html#create-engine-url-arguments

bauble.**main_is_frozen**()
Return True if we are running in a py2exe environment, else return False

bauble.**quit**()
Stop all tasks and quit Ghini.

bauble.**save_state**()
Save the gui state and preferences.

5.9.2 bauble.db

bauble.db.**Base**
All tables/mappers in Bauble which use the SQLAlchemy declarative plugin for declaring tables and mappers should derive from this class.

An instance of sqlalchemy.ext.declarative.Base

bauble.db.**metadata**
The default metadata for all Bauble tables.

An instance of sqlalchemy.schema.MetaData

5.9.3 bauble.connmgr

5.9.4 bauble.editor

5.9.5 bauble.i18n

The i18n module defines the _() function for creating translatable strings.

_() is added to the Python builtins so there is no reason to import this module more than once in an application. It is usually imported in *bauble*

5.9.6 bauble.ui

5.9.7 bauble.meta

5.9.8 bauble.paths

Access to standard paths used by Ghini.

bauble.paths.**main_dir**()
Returns the path of the bauble executable.

bauble.paths.**lib_dir**()
Returns the path of the bauble module.

bauble.paths.**locale_dir**()
Returns the root path of the locale files

bauble.paths.**user_dir**()

Returns the path to where user data are saved.

this is not the same as Application Data, for app_data is going to be replaced at each new installation or upgrade of the software. user_data is responsibility of the user and the software should use it, not overrule it.

not implemented yet. will be a configuration item.

5.9.9 bauble.pluginmgr

5.9.10 bauble.prefs

5.9.11 bauble.task

5.9.12 bauble.types

5.9.13 bauble.utils

5.9.14 bauble.view

class bauble.view.SearchView.**ViewMeta**

5.9.15 bauble.search

5.9.16 bauble.plugins.plants

5.9.17 bauble.plugins.garden

5.9.18 bauble.plugins.abcd

5.9.19 bauble.plugins.imex

5.9.20 bauble.plugins.report

5.9.21 bauble.plugins.report.xsl

5.9.22 bauble.plugins.report.mako

5.9.23 bauble.plugins.tag

CHAPTER 6

Supporting Ghini

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

Python Module Index

b

Index