Bauble Documentation

Release 1.0

July 10, 2015

Bauble 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

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

When you first start Bauble, and connect to a database, Bauble 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.

Bauble 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 Bauble professional to help you transform your data into Bauble's intermediate json format.

Bauble 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 Bauble database.

Bauble 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.

Bauble 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.

Bauble 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.

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

Management of plant locations.

All changes in the database is stored in the database, as history log. All changes are 'signed' and time-stamped. Bauble 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.

Bauble 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.

Bauble 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 systen 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 Bauble 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.

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%).

Installing Bauble on Windows is an easy and linear process, it will not take longer than 10 minutes. Bauble 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.

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.

Bauble 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.

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

CHAPTER 2

Installing Bauble

2.1 Installation

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

To install Bauble 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 Bauble should be able to install the rest of its dependencies by itself.

Note: If you follow these installation steps, you will end with Bauble 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 Bauble, you simply remove the virtual environment, which is a directory, with all of its content.

2.1.1 Installing on Linux

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

```
https://raw.githubusercontent.com/Bauble/bauble.classic/master/scripts/devinstall.sh
```

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

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

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

```
~/bin/bauble
```

   or update bauble to the latest released production patch:

```
~/bin/bauble -u
```

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

2. on Unity, open a terminal, start bauble, its icon will show up in the launcher, you can now *lock to launcher* it.

3. 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/bacl/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.

2.1.2 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. Bauble 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.

2.1.3 Installing on Windows

The Windows installer used to be a "batteries-included" installer, installing everything needed to run Bauble. The current maintainer of bauble.classic cannot run Windows applications. If you want to run the latest version of bauble on Windows: download and install the dependencies and then install Bauble from the source package.

Please report any trouble and help with packaging will be very welcome.

Note: Bauble has been tested with and is known to work on W-XP, W-7 and W-8. 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`).
2. download and install Python 2.x (32bit) from:

   http://www.python.org

   Bauble 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 Bauble 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. make a complete install, selecting everything:

   http://ftp.gnome.org/pub/GNOME/binaries/win32/pygtk/
4. (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
5. install `virtualenv` (using `pip`)
6. cd to your HOME dir, create the virtual environment, call it `bacl` and activate it:

```
virtualenv --system-site-packages .virtualenvs\bacl
.virtualenvs\bacl\Scripts\activate.bat
```

7. cd to where you want to get bauble.classic. a good choice would be:

```
Local\github\Bauble\
```

8. download the bauble.classic sources (using git) from:

   http://www.github.com/Bauble/bauble.classic/
9. cd into the newly created `bauble.classic` directory.
10. choose the development line you plan to follow, for example `1.0`, build, install:

```
git checkout bauble-1.0
python setup.py build
python setup.py install
```

11. create a `bauble.bat` file in your HOME dir, with this content:

```
call .virtualenvs\bacl\Scripts\activate.bat
pythonw .virtualenvs\bacl\Scripts\bauble
```

12. create a vbs file in your HOME dir, with this content:

```
    CreateObject("Wscript.Shell").Run "bauble.bat", 0, True
```

13. create a shortcut to the vbs file in the same HOME dir.
14. modify the icon of the shortcut, rename it as of your tastes.
15. drag and drop the shortcut into the Start Menu.
16. the following two, you will do regularly, to stay up-to-date with the development line you chose to follow:

```
    git pull
    python setup.py install
```

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

Next...

Connecting to a database.

2.1.4 Troubleshooting the Install

1. What are the packages that are installed by Bauble:

   The following packages are required by Bauble

   - SQLAlchemy
   - lxml

   The following packages are optional:

   - Mako - required by the template based report generator
   - gdata - required by the Picasa photos InfoBox

2. Couldn't install lxml.

   The lxml packages have to be compile with a C compiler. If you don't have a Make sure the libxml and libxsl packages are installed. Installing the Cython packages. On Linux you will have to install the gcc package. On Windows there should be a precompiled version available at http://pypi.python.org/pypi/lxml/2.1.1

3. 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 Bauble

3.1 Getting Started

3.1.1 Connecting to a database

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

From this dialog you can select the different connection parameters.

If this is the first time that you are starting Bauble then you will not having any connections to choose from. Click on the add button to create a new connection.

By default Bauble uses the file-based SQLite database. If you use the default filename then Bauble creates a database file with the same name as the connection in `~/.bauble` on Linux or `Application Data\Bauble` on Windows.

Bauble allows you to connect to any existing database. If you connect to an empty database a message will popup asking asking you if you would like to create a new database.

30. TODO: If you are connecting to an existing database you can continue to Inserting or Searching

3.1.2 Creating a new database

To create a new database you have to first connect to a database. See *Connecting to a database*.

If you are connecting using the default SQLite database backend then Bauble can handler everything that needs to be done to create a new database.

If you are connecting to a server based database like PostgreSQL will have to manually create the database and permissions for the database while Bauble will create the tables and import the default data set. Creating a database on aserver based database is beyond the scope of this manual. If you just got the chills or sick at your stomach I recommend you just stick with SQLite.

If you have connected to a database that has not yet been initialized by Bauble then you will get the following dialog:

Be careful because if you have entered the wrong connection parameters it is possible to overwrite an existing database at this connection.

If you are sure you want to create a database at this connection then select “Yes”. Bauble 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.

XXX. TODO: Once the default database has been created then you are ready to start inserting or searching...

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 The Query Builder

The Query Builder can help 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.

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.

3.2.2 The Query Language

Three are three types of search queries available in Bauble. You can search by value, expression or query.

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`

3.3 Editing and Inserting Data

The main way that we add or change information in Bauble 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 Bauble 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 Bauble 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.

3.3.7 Locations

The Location editor

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 Bauble 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 Bauble that were previously exported from Bauble. It is possible to import any CSV file but that is more advanced that this doc will cover.

To import CSV files into Bauble 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 Bauble 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 Bauble 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 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

Bauble Development

4.1 Downloading the source

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

If you want a particular version of Bauble, we release and maintain versions into branches. you should `git checkout` the branch corresponding to the version of your choice. Branch names for Bauble versions are of the form `bauble-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 Bauble code with the following command:

```
git clone https://github.com/Bauble/bauble.classic.git
```

For more information about other available code branches go to bauble.classic on github.

4.2 Building the source

Building a python program is a bit of a contraddiction. You don't normally *build* nor *compile* a python program, you run it in its environment, and python will process the modules loaded and produce faster-loading *compiled* python files. You can, however, produce a Windows executable from a python script, executable containing the whole python environment and dependencies.

4.2.1 Building (on Windows)

1. In order to build a Bauble executable you will first need to download the source code. For more information about download the Bauble source go to Downloading the source.
2. Follow all steps needed to set up a working Bauble environment from Installation, but skip the final *install* step.
3. instead of *installing* Bauble, you produce a Windows executable. This is achieved with the `py2exe` target, which is only available on Windows systems:

```
    python setup.py py2exe
```

4. At this point you can run Bauble. To run the compiled executable run:

```
    .\dist\bauble.exe
```

or copy the executable to wherever you think appropriate.

6. To optionally build an NSIS installer package you must install NSIS from nsis.sourceforge.net. After installing NSIS right click on `.\scripts\build.nsi` in Explorer and select *Compile NSIS Script*.

4.3 Extending Bauble with Plugins

Nearly everything about Bauble 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.

4.4 API Documentation

4.4.1 `bauble`

The top level module for Bauble.

`bauble.`**`version`** **= '1.0.32'**

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 Bauble application.

Parameters **`uri`** – 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 Bauble.

`bauble.`**`save_state`**`()`

Save the gui state and preferences.

4.4.2 `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`

4.4.3 `bauble.connmgr`

4.4.4 `bauble.editor`

4.4.5 `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`

4.4.6 `bauble.ui`

4.4.7 `bauble.meta`

4.4.8 `bauble.paths`

Access to standard paths used by Bauble.

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 Bauble settings should be saved.

4.4.9 `bauble.pluginmgr`

4.4.10 `bauble.prefs`

4.4.11 `bauble.task`

The bauble.task module allows you to queue up long running tasks. The running tasks still block but allows the GUI to update.

`bauble.task.`**`queue`**`(`*task*`)`
Run a task.

task should be a generator with side effects. it does not matter what it yields, it is important that it does stop from time to time yielding whatever it wants to, and causing the side effect it has to cause.

`bauble.task.`**`set_message`**`(`*msg*`)`
A convenience function for setting a message on the statusbar. Returns the message id

`bauble.task.`**`clear_messages`**`()`
Clear all the messages from the statusbar that were set with `bauble.task.set_message()`

4.4.12 `bauble.types`

4.4.13 `bauble.utils`

A common set of utility functions used throughout Bauble.

`bauble.utils.`**`find_dependent_tables`**`(`*table*, *metadata=None*`)`
Return an iterator with all tables that depend on table. The tables are returned in the order that they depend on each other. For example you know that table[0] does not depend on tables[1].

Parameters

- **`table`** – The tables who dependencies we want to find
- **`metadata`** – The `sqlalchemy.engine.MetaData` object that holds the tables to search through. If None then use bauble.db.metadata

`bauble.utils.`**`tree_model_has`**`(`*tree*, *value*`)`
Return True or False if value is in the tree.

`bauble.utils.`**`search_tree_model`**`(`*parent*, *data*, *cmp=<function <lambda>>*`)`
Return a iterable of gtk.TreeIter instances to all occurences of data in model

Parameters

- **`parent`** – a gtk.TreeModel or a gtk.TreeModelRow instance
- **`data`** – the data to look for
- **`cmp`** – the function to call on each row to check if it matches data, default is C{lambda row, data: row[0] == data}

`bauble.utils.`**`clear_model`**`(`*obj_with_model*`)`

Parameters **`obj_with_model`** – a gtk Widget that has a gtk.TreeModel that can be retrieved with obj_with_mode.get_model

Remove the model from the object, deletes all the items in the model, clear the model and then delete the model and set the model on the object to None

`bauble.utils.`**`combo_set_active_text`**`(`*combo*, *value*`)`
does the same thing as set_combo_from_value but this looks more like a GTK+ method

`bauble.utils.`**`set_combo_from_value`**`(`*combo*, *value*, *cmp=<function <lambda>>*`)`
Find value in combo model and set it as active, else raise ValueError cmp(row, value) is the a function to use for comparison

Note: if more than one value is found in the combo then the first one in the list is set `bauble.utils.`**`combo_get_value_iter`**`(`*combo*, *value*, *cmp=<function <lambda>>*`)`

Returns a gtk.TreeIter that points to first matching value in the combo's model.

Parameters

- **`combo`** – the combo where we should search
- **`value`** – the value to search for
- **`cmp`** – the method to use to compare rows in the combo model and value, the default is C{lambda row, value: row[0] == value}

Note: if more than one value is found in the combo then the first one in the list is returned

`bauble.utils.`**`set_widget_value`**`(`*widget*, *value*, *markup=False*, *default=None*, *index=0*`)`

Parameters

- **`widget`** – an instance of gtk.Widget
- **`value`** – the value to put in the widget
- **`markup`** – whether or not value is markup
- **`default`** – the default value to put in the widget if the value is None
- **`index`** – the row index to use for those widgets who use a model

Note: any values passed in for widgets that expect a string will call the values __str__ method

`bauble.utils.`**`create_message_dialog`**`(`*msg*, *type=<enum GTK_MESSAGE_INFO of type GtkMessageType>*, *buttons=<enum GTK_BUTTONS_OK of type GtkButtonsType>*, *parent=None*`)`

Create a message dialog.

Parameters

- **`msg`** – The markup to use for the message. The value should be escaped in case it contains any HTML entities.
- **`type`** – A GTK message type constant. The default is gtk.MESSAGE_INFO.
- **`buttons`** – A GTK buttons type constant. The default is gtk.BUTTONS_OK.
- **`parent`** – The parent window for the dialog

Returns a `gtk.MessageDialog`

`bauble.utils.`**`message_dialog`**`(`*msg*, *type=<enum GTK_MESSAGE_INFO of type GtkMessageType>*, *buttons=<enum GTK_BUTTONS_OK of type GtkButtonsType>*, *parent=None*`)`

Create a message dialog with `bauble.utils.create_message_dialog()` and run and destroy it.

Returns the dialog's response.

`bauble.utils.`**`create_yes_no_dialog`**`(`*msg*, *parent=None*`)`

Create a dialog with yes/no buttons.

`bauble.utils.`**`yes_no_dialog`**`(`*msg*, *parent=None*, *yes_delay=-1*`)`

Create and run a yes/no dialog.

Return True if the dialog response equals gtk.RESPONSE_YES

Parameters

- **`msg`** – the message to display in the dialog
- **`parent`** – the dialog's parent
- **`yes_delay`** – the number of seconds before the yes button should become sensitive

`bauble.utils.`**`create_message_details_dialog`**(*msg*, *details*, *type=<enum GTK_MESSAGE_INFO of type GtkMessageType>*, *buttons=<enum GTK_BUTTONS_OK of type GtkButtonsType>*, *parent=None*)

Create a message dialog with a details expander.

`bauble.utils.`**`message_details_dialog`**(*msg*, *details*, *type=<enum GTK_MESSAGE_INFO of type GtkMessageType>*, *buttons=<enum GTK_BUTTONS_OK of type GtkButtonsType>*, *parent=None*)

Create and run a message dialog with a details expander.

`bauble.utils.`**`setup_text_combobox`**(*combo*, *values=None*, *cell_data_func=None*)

Configure a gtk.ComboBox as a text combobox

NOTE: If you pass a cell_data_func that is a method of an object that holds a reference to combo then the object will not be properly garbage collected. To avoid this problem either don't pass a method of object or make the method static

Parameters

- **`combo`** – gtk.ComboBox
- **`values`** – list vales or gtk.ListStore
- **`cell_date_func`** –

`bauble.utils.`**`setup_date_button`**(*view*, *entry*, *button*, *date_func=None*)

Associate a button with entry so that when the button is clicked a date is inserted into the entry.

Parameters

- **`view`** – a bauble.editor.GenericEditorView
- **`entry`** – the entry that the data goes into
- **`button`** – the button that enters the data in entry
- **`date_func`** – the function that returns a string represention of the date

`bauble.utils.`**`to_unicode`**(*obj*, *encoding='utf-8'*)

Return obj converted to unicode. If obj is already a unicode object it will not try to decode it to converted it to <encoding> but will just return the original obj

`bauble.utils.`**`utf8`**(*obj*)

This function is an alias for to_unicode(obj, 'utf-8')

`bauble.utils.`**`xml_safe`**(*obj*, *encoding='utf-8'*)

Return a string with character entities escaped safe for xml, if the str parameter is a string a string is returned, if str is a unicode object then a unicode object is returned

`bauble.utils.`**`xml_safe_utf8`**(*obj*)

This method is deprecated and just returns xml_safe(obj)

`bauble.utils.`**`natsort_key`**(*obj*)

a key getter for sort and sorted function the sorting is done on return value of obj.__str__() so we can sort objects as well, i don't know if this will cause problems with unicode

use like: sorted(some_list, key=utils.natsort_key)

`bauble.utils.`**`delete_or_expunge`**(*obj*)
If the object is in object_session(obj).new then expunge it from the session. If not then session.delete it.

`bauble.utils.`**`reset_sequence`**(*column*)
If column.sequence is not None or the column is an Integer and column.autoincrement is true then reset the sequence for the next available value for the column...if the column doesn't have a sequence then do nothing and return

The SQL statements are executed directly from db.engine

This function only works for PostgreSQL database. It does nothing for other database engines.

`bauble.utils.`**`make_label_clickable`**(*label*, *on_clicked*, **args*)

Parameters

- **`label`** – a gtk.Label that has a gtk.EventBox as its parent
- **`on_clicked`** – callback to be called when the label is clicked on_clicked(label, event, data)

`bauble.utils.`**`enum_values_str`**(*col*)

Parameters **`col`** – a string if table.col where col is an enum type

return a string with of the values on an enum type join by a comma

`bauble.utils.`**`which`**(*filename*, *path=None*)
Return first occurence of file on the path.

`bauble.utils.`**`ilike`**(*col*, *val*, *engine=None*)
Return a cross platform ilike function.

`bauble.utils.`**`range_builder`**(*text*)
Return a list of numbers from a string range of the form 1-3,4,5

`bauble.utils.`**`topological_sort`**(*items*, *partial_order*)
Perform topological sort.

Parameters

- **`items`** – a list of items to be sorted.
- **`partial_order`** – a list of pairs. If pair (a,b) is in it, it means that item a should appear before item b. Returns a list of the items in one of the possible orders, or None if partial_order contains a loop.

`bauble.utils.`**`get_distinct_values`**(*column*, *session*)
Return a list of all the distinct values in a table column

`bauble.utils.`**`get_invalid_columns`**(*obj, ignore_columns=['id']*)
Return column names on a mapped object that have values which aren't valid for the model.

Invalid columns meet the following criteria: - nullable columns with null values - ...what else?

`bauble.utils.`**`get_urls`**(*text*)
Return tuples of http/https links and labels for the links. To label a link prefix it with [label text], e.g. [BBG]http://belizebotanic.org

class `bauble.utils.`**`GenericMessageBox`**

Bases: `gtk.EventBox`

Abstract class for showing a message box at the top of an editor.

class `bauble.utils.`**`MessageBox`**`(`*msg=None*, *details=None*`)`

Bases: `bauble.utils.GenericMessageBox`

A MessageBox that can display a message label at the top of an editor.

class `bauble.utils.`**`YesNoMessageBox`**`(`*msg=None*, *on_response=None*`)`

Bases: `bauble.utils.GenericMessageBox`

A message box that can present a Yes or No question to the user

`bauble.utils.`**`add_message_box`**`(`*parent*, *type=1*`)`

Parameters

- **`parent`** – the parent `gtk.Box` width to add the message box to
- **`type`** – one of MESSAGE_BOX_INFO, MESSAGE_BOX_ERROR or MESSAGE_BOX_YESNO

4.4.14 `bauble.view`

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

4.4.15 `bauble.search`

4.4.16 `bauble.plugins.plants`

4.4.17 `bauble.plugins.garden`

4.4.18 `bauble.plugins.abcd`

4.4.19 `bauble.plugins.imex`

4.4.20 `bauble.plugins.report`

4.4.21 `bauble.plugins.report.xsl`

4.4.22 `bauble.plugins.report.mako`

4.4.23 `bauble.plugins.tag`

CHAPTER 5

Supporting Bauble

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

Index