Bauble Documentation

Release 1.0.47

Brett Adams

August 09, 2015

Contents

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

CHAPTER 1

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

1.1 taxonomic information

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.

1.2 importing data

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.

1.3 synonyms

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.

1.4 scientific responsible

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.

1.5 helps off-line identification

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.

1.6 exports and reports

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.

1.7 annotate your info

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

1.8 garden or herbarium

Management of plant locations.

1.9 database history

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.

1.10 simple and powerful search

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.

1.11 database agnostic

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 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 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.

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%).

1.13 platform agnostic

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.

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.

1.15 unit tested

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.

1.16 customizable/extensible

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/devin
```

   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.

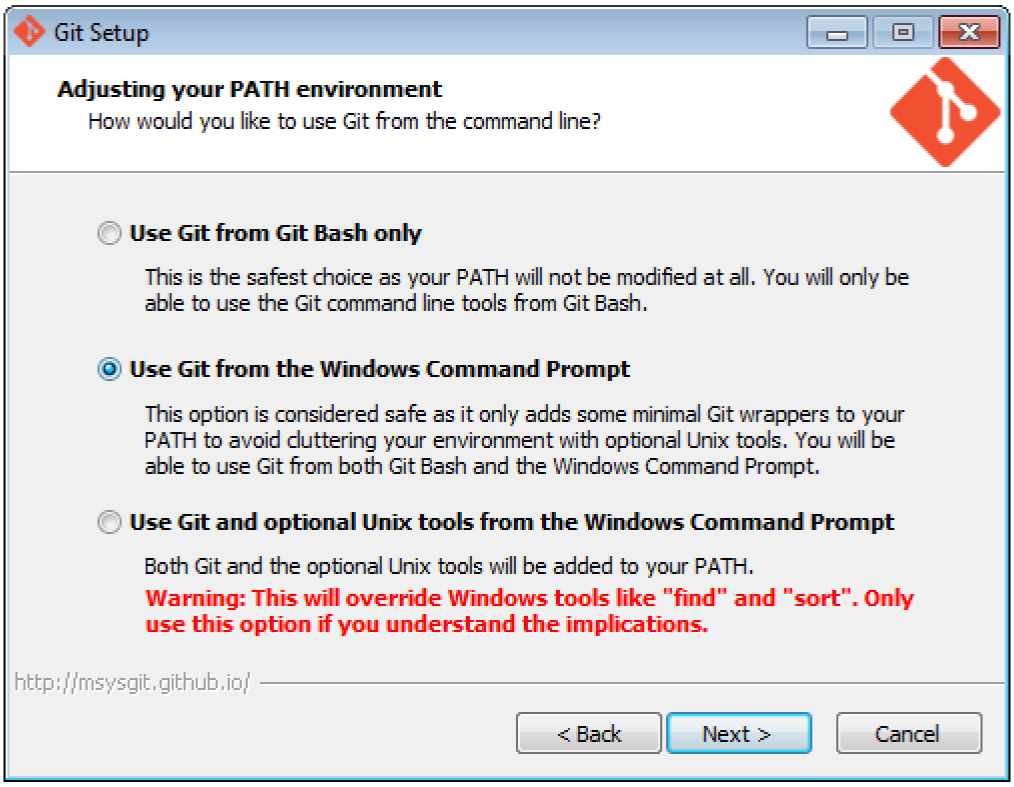
Please report any trouble and help with packaging will be very welcome.

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`).

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



Adjusting your PATH environment
How would you like to use Git from the command line?

( ) **Use Git from Git Bash only**
This is the safest choice as your PATH will not be modified at all. You will only be able to use the Git command line tools from Git Bash.

[✓ **Use Git from the Windows Command Prompt**]
This option is considered safe as it only adds some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools. You will be able to use Git from both Git Bash and the Windows Command Prompt.

( ) **Use Git and optional Unix tools from the Windows Command Prompt**
Both Git and the optional Unix tools will be added to your PATH.
Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.

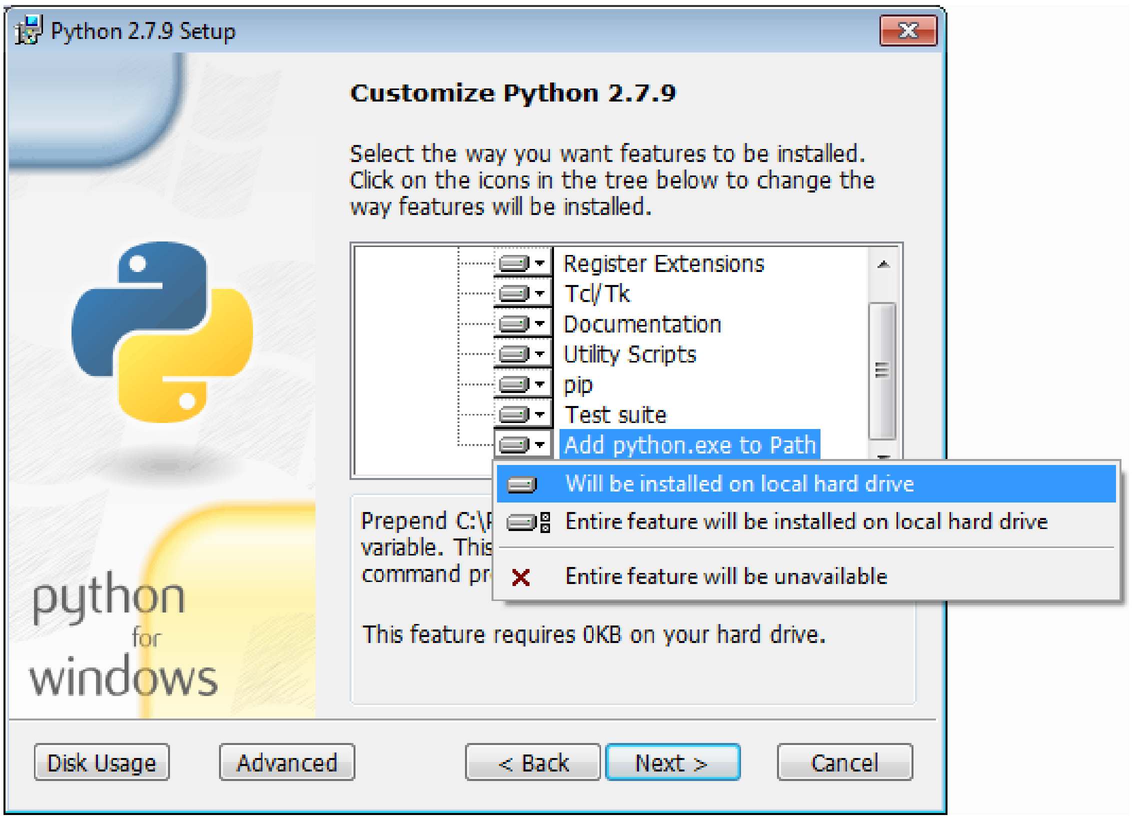
http://msysgit.github.io/

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:


Python 2.7.9 Setup

Customize Python 2.7.9

Select the way you want features to be installed.
Click on the icons in the tree below to change the
way features will be installed.

Register Extensions
Tcl/Tk
Documentation
Utility Scripts
pip
Test suite
Add python.exe to Path

Will be installed on local hard drive
Entire feature will be installed on local hard drive
Entire feature will be unavailable

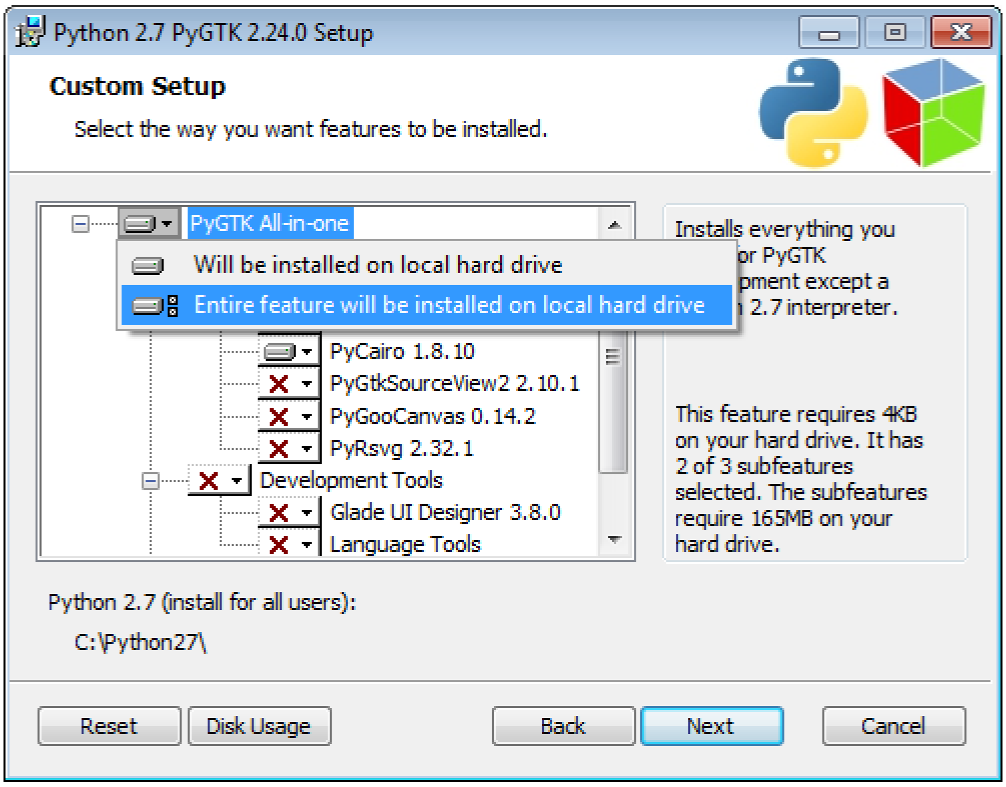
Prepend C:\P
variable. This
command pr

This feature requires 0KB on your hard drive.

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/
```

make a complete install, selecting everything:


Python 2.7 PyGTK 2.24.0 Setup

Custom Setup

Select the way you want features to be installed.

PyGTK All-in-one
Will be installed on local hard drive
Entire feature will be installed on local hard drive
PyCairo 1.8.10
PyGtkSourceView2 2.10.1
PyGooCanvas 0.14.2
PyRsvg 2.32.1
Development Tools
Glade UI Designer 3.8.0
Language Tools

Installs everything you
or PyGTK
pment except a
2.7 interpreter.

This feature requires 4KB on your hard drive. It has 2 of 3 subfeatures selected. The subfeatures require 165MB on your hard drive.

Python 2.7 (install for all users):
C:\Python27\

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. **REBOOT**

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

6. download and run the batch file:

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

   this will pull the `bauble.classic` repository on github to your home directory, under `Local\github\Bauble`, checkout the `bauble-1.0` production line, create a virtual environment and install bauble into it.

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

7. the following, you will do regularly, to stay up-to-date with the development line you chose to follow:

```
cd %HOMEDRIVE%%HOMEPATH%
.virtualenv\bacl\Scripts\activate.bat
cd Local\github\Bauble\bauble.classic
git pull
python setup.py install
```

8. you can now start bauble using the `bauble.lnk` shortcut that the installation procedure copies to the `Scripts` directory of the virtual environment:

```
%HOMEDRIVE%%HOMEPATH%\.virtualenv\bacl\Scripts\bauble.lnk
```

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.

If you plan to associate pictures to plants, specify also the *pictures root* folder. This is explained in further detail in the Plants section.

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/MacOSX or in `AppData\Roaming\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 inizialize it as a new database.

If you are connecting to an existing database you can continue to Inserting or Searching, otherwise read on to the following section.

3.1.2 Creating a new database

To inizialize a 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 handle everything that needs to be done to create a database that Bauble will then initialize.

If you are connecting to a server based database like PostgreSQL will have to manually create the database, user and permissions for the database while Bauble will create the tables and import the default data set. Creating a database on a server 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.

Once the default database has been created then you are ready to start inserting and subsequently 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 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`

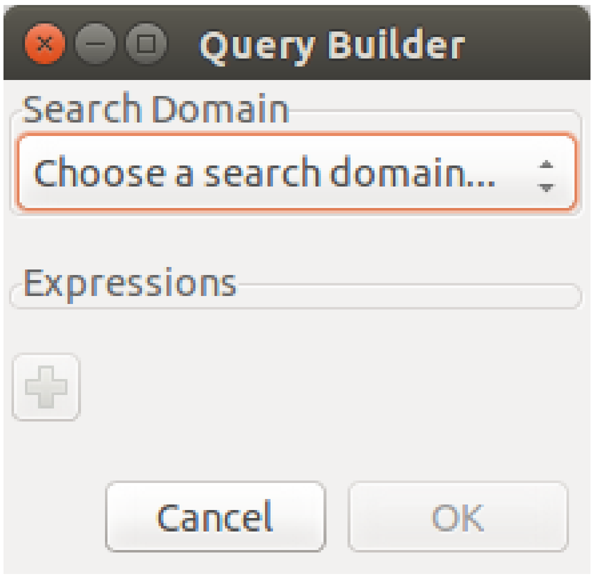
location, loc: Search `bauble.plugins.garden.Location`

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.

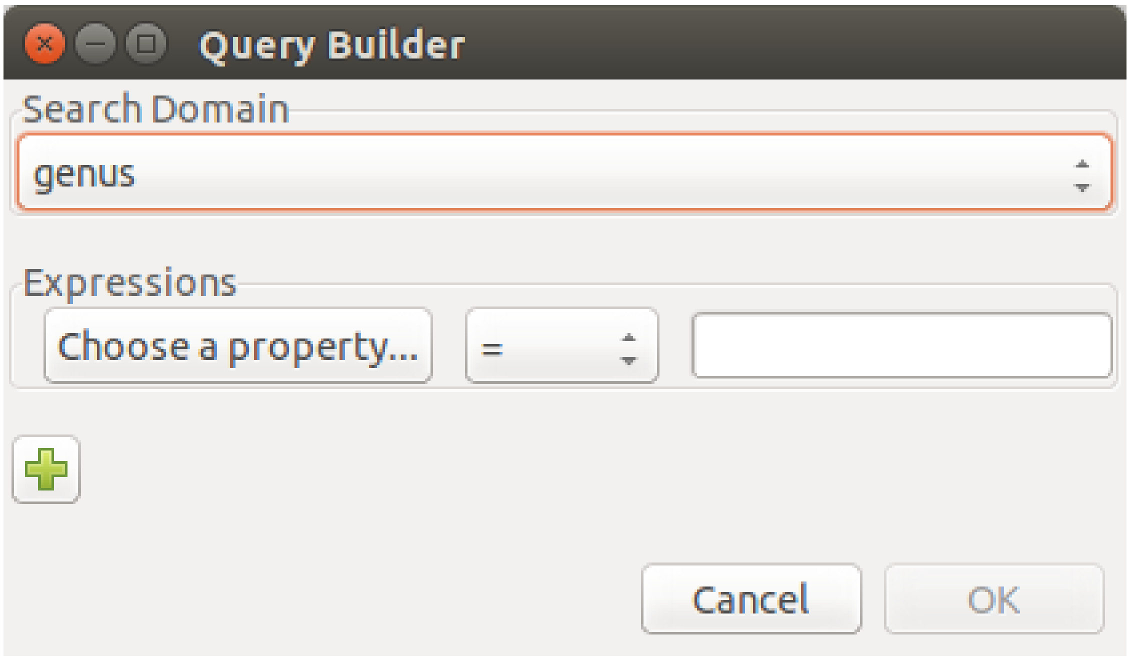
The Query Builder composes a query that will be understood by the Query Search Strategy described above. You can use the Query Builder to get a feeling of correct queries before you start typing them by hand, something that you might prefer if you are a fast typer.

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 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.

Pictures

Just as almost all objects in the Bauble 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 Bauble-1.0.41, Pictures are not kept in the database. To ensure pictures are available on all terminals where you have installed and configured Bauble, 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 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.

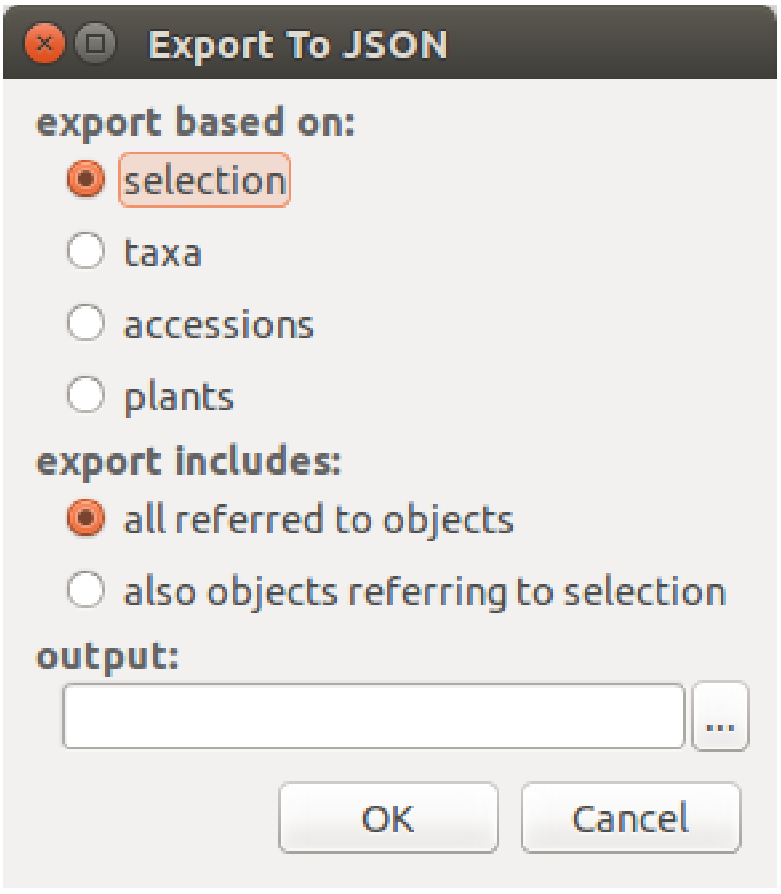
3.6.3 Importing from JSON

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 Bauble database. Converting a database into bauble json interchange format is beyond the scope of this manual, please contact one of the authors if you need any further help.

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

3.6.4 Exporting to JSON

This feature is still under development.


Export To JSON

export based on:

- [✓ selection]
- taxa
- accessions
- plants

export includes:

- [✓ all referred to objects]
- also objects referring to selection

output:

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 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, bauble puts its data in the `~/.bauble/` directory (in Windows it is somewhere in your `AppData` directory).

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.

CHAPTER 5

Bauble Development

5.1 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.

5.2 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.

5.2.1 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*.

5.3 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.

5.4 API Documentation

5.4.1 `bauble`

The top level module for Bauble.

`bauble.`**`version`** **= '1.0.47'**
    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 (*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 Bauble.

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

5.4.2 `bauble.db`

`bauble.db.`**`Session`** **= None**
bauble.db.Session is created after the database has been opened with *`bauble.db.open()`*. bauble.db.Session should be used when you need to do ORM based activities on a bauble database. To create a new Session use::Uncategorized

session = bauble.db.Session()

When you are finished with the session be sure to close the session with `session.close()`. Failure to close sessions can lead to database deadlocks, particularly when using PostgreSQL based databases.

`bauble.db.`**`engine`** **= None**
A `sqlalchemy.engine.base.Engine` used as the default connection to the database.

`bauble.db.`**`Base`** **= <class 'sqlalchemy.ext.declarative.api.Base'>**

`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`

`db.`**`metadata`** **= MetaData(bind=None)**

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

An instance of `sqlalchemy.schema.MetaData`

class `bauble.db.`**`MapperBase`**`(`*classname*, *bases*, *dict_*`)`
MapperBase adds the id, _created and _last_updated columns to all tables.

In general there is no reason to use this class directly other than to extend it to add more default columns to all the bauble tables.

class `bauble.db.`**`HistoryExtension`**
Bases: `sqlalchemy.orm.deprecated_interfaces.MapperExtension`

HistoryExtension is a `MapperExtension` that is added to all clases that inherit from bauble.db.Base so that all inserts, updates, and deletes made to the mapped objects are recorded in the *history* table.

class `bauble.db.`**`History`**`(`***kwargs*`)`

Bases: `sqlalchemy.ext.declarative.api.Base`

The history table records ever changed made to every table that inherits from Base

Table name history

Columns

id: `sqlalchemy.types.Integer` A unique identifier.

table_name: `sqlalchemy.types.String` The name of the table the change was made on.

table_id: `sqlalchemy.types.Integer` The id in the table of the row that was changed.

values: `sqlalchemy.types.String` The changed values.

operation: `sqlalchemy.types.String` The type of change. This is usually one of insert, update or delete.

user: `sqlalchemy.types.String` The name of the user who made the change.

timestamp: `sqlalchemy.types.DateTime` When the change was made.

`bauble.db.`**`open`**`(`*uri*, *verify=True*, *show_error_dialogs=False*`)`

Open a database connection. This function sets bauble.db.engine to the opened engined.

Return bauble.db.engine if successful else returns None and bauble.db.engine remains unchanged.

Parameters

- **`uri`** (*str*) – The URI of the database to open.
- **`verify`** (*bool*) – Where the database we connect to should be verified as one created by Bauble. This flag is used mostly for testing.
- **`show_error_dialogs`** (*bool*) – A flag to indicate whether the error dialogs should be displayed. This is used mostly for testing.

`bauble.db.`**`create`**`(`*import_defaults=True*`)`

Create new Bauble database at the current connection

Parameters **`import_defaults`** (*bool*) – A flag that is passed to each plugins install() method to indicate where it should import its default data. This is mainly used for testing. The default value is True

`bauble.db.`**`verify_connection`**`(`*engine*, *show_error_dialogs=False*`)`

Test whether a connection to an engine is a valid Bauble database. This method will raise an error for the first problem it finds with the database.

Parameters

- **`engine`** (`sqlalchemy.engine.Engine`) – the engine to test
- **`show_error_dialogs`** (*bool*) – flag for whether or not to show message dialogs detailing the error, default=False

5.4.3 `bauble.connmgr`

The connection manager provides a GUI for creating and opening connections. This is the first thing displayed when Bauble starts.

class `bauble.connmgr.`**`ConnectionManager`**(*default=None*)

The main class that starts the connection manager GUI.

Parameters **`default`** – the name of the connection to select from the list of connection names

`check_parameters_valid`()

check that all of the information in the current connection is valid and return true or false

NOTE: this was meant to be used to implement an eclipse style information box at the top of the dialog but it's not really used right now

`compare_prefs_to_saved`(*name*)

name is the name of the connection in the prefs

`get_passwd`(*title='Enter your password'*, *before_main=False*)

Show a dialog with and entry and return the value entered.

`on_changed_name_combo`(*combo*, *data=None*)

the name changed so fill in everything else

`on_changed_type_combo`(*combo*, *data=None*)

the type changed so change the params_box

`on_dialog_response`(*dialog*, *response*, *data=None*)

The dialog's response signal handler.

`on_remove_button_clicked`(*button*, *data=None*)

remove the connection from connection list, this does not affect the database or its data

`parameters_to_uri`(*params*)

return connections paramaters as a uri

`remove_connection`(*name*)

if we restrict the user to only removing the current connection then it saves us the trouble of having to iter through the model

`save_current_to_prefs`()

save connection parameters from the widgets in the prefs

set_active_connection_by_name(*name*)

sets the name of the connection in the name combo, this causes on_changed_name_combo to be fired which changes the param box type and set the connection parameters

start()

Show the connection manager.

working_dbtypes

get for self.working_dbtypes property

this sets self._working_dbtypes to a dictionary where the keys are the database names and the values are the index in the connectiona manager's database types

5.4.4 bauble.editor

bauble.editor.**default_completion_cell_data_func**(*column*, *renderer*, *model*, *treeiter*, *data=None*)

the default completion cell data function for GenericEditorView.attach_completions

bauble.editor.**default_completion_match_func**(*completion*, *key_string*, *treeiter*)

the default completion match function for GenericEditorView.attach_completions, does a case-insensitive string comparison of the the completions model[iter][0]

class bauble.editor.**ValidatorError**(*msg*)

class bauble.editor.**Validator**

The interface that other validators should implement.

class bauble.editor.**StringOrNoneValidator**

If the value is an empty string then return None, else return the str() of the value.

class bauble.editor.**UnicodeOrNoneValidator**(*encoding='utf-8'*)

If the value is an empty unicode string then return None, else return the unicode() of the value. The default encoding is 'utf-8'.

class bauble.editor.**IntOrNoneStringValidator**

If the value is an int, long or can be cast to int then return the number, else return None

class bauble.editor.**FloatOrNoneStringValidator**

If the value is an int, long, float or can be cast to float then return the number, else return None

class bauble.editor.**GenericEditorView**(*filename*, *parent=None*)

An generic object meant to be extended to provide the view for a GenericModelViewPresenterEditor.

Parameters

- **filename** – a gtk.Builder UI definition

- **parent** – a gtk.Window or subclass to use as the parent window, if parent=None then bauble.gui.window is used

attach_completion(*entry*, *cell_data_func=<function default_completion_cell_data_func>*, *match_func=<function default_completion_match_func>*, *minimum_key_length=2*, *text_column=-1*)

Attach an entry completion to a gtk.Entry. The defaults values for this attach_completion assumes the completion popup only shows text and that the text is in the first column of the model.

Return the completion attached to the entry.

NOTE: If you are selecting completions from strings in your model you must set the text_column parameter to the column in the model that holds the strings or else when you select the string from the completions it won't get set properly in the entry even though you call entry.set_text().

Parameters

- **entry** – the name of the entry to attach the completion
- **cell_data_func** – the function to use to display the rows in the completion popup
- **match_func** – a function that returns True/False if the value from the model should be shown in the completions
- **minimum_key_length** – default=2
- **text_column** – the value of the text-column property on the entry, default is -1

cleanup()

Should be caled when after self.start() returns to cleanup undo any changes on the view.

By default all it does is call self.disconnect_all()

connect(*obj*, *signal*, *callback*, **args*)

Attach a signal handler for signal on obj. For more information see `gobject.connect_after()`

Parameters

- **obj** – An instance of a subclass of gobject that will receive the signal
- **signal** – the name of the signal the object will receive
- **callback** – the function or method to call the object receives the signal
- **args** – extra args to pass the the callback

connect_after(*obj*, *signal*, *callback*, **args*)
Attach a signal handler for signal on obj. For more information see `gobject.connect_after()`

Parameters

- **obj** – An instance of a subclass of gobject that will receive the signal
- **signal** – the name of the signal the object will receive
- **callback** – the function or method to call the object receives the signal
- **args** – extra args to pass the the callback

disconnect_all()
Disconnects all the signal handlers attached with `GenericEditorView.connect()` or `GenericEditorView.connect_after()`

get_window()
Return the top level window for view

init_translatable_combo(*combo*, *translations*, *default=None*, *cmp=None*)
Initialize a gtk.ComboBox with translations values where model[row][0] is the value that will be stored in the database and model[row][1] is the value that will be visible in the gtk.ComboBox.

A gtk.ComboBox initialized with this method should work with self.assign_simple_handler()

Parameters

- **combo** –
- **translations** – a list of pairs, or a dictionary, of values->translation.

on_dialog_close(*dialog*, *event=None*)
Called if self.get_window() is a gtk.Dialog and it receives the close signal.

on_dialog_response(*dialog*, *response*, **args*)
Called if self.get_window() is a gtk.Dialog and it receives the response signal.

on_window_delete(*window*, *event=None*)
Called when the window return by get_window() receives the delete event.

restore_state()
Restore the state of the view, this is usually done by getting a value by the preferences and setting the equivalent in the interface

save_state()
Save the state of the view by setting a value in the preferences that will be called restored in restore_state e.g. prefs[pref_string] = pref_value

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

Parameters

- **`widget`** – a widget or name of a widget in self.widgets
- **`value`** – the value to put in the widgets
- **`markup`** – whether the data in value uses pango markup
- **`default`** – the default value to put in the widget if value is None
- **`index`** – the row index to use for those widgets who use a model

This method called bauble.utils.set_widget_value()

class `bauble.editor.`**`GenericEditorPresenter`** (*model*, *view*)

The presenter of the Model View Presenter Pattern

Parameters

- **`model`** – an object instance mapped to an SQLAlchemy table
- **`view`** – should be an instance of GenericEditorView

The presenter should usually be initialized in the following order: 1. initialize the widgets 2. refresh the view, put values from the model into the widgets 3. connect the signal handlers

`add_problem` (*problem_id*, *problem_widgets=None*)

Add problem_id to self.problems and change the background of widget(s) in problem_widgets.

Parameters

- **`problem_id`** – A unique id for the problem.
- **`problem_widgets`** – either a widget or list of widgets whose background color should change to indicate a problem (default=None)

`assign_completions_handler` (*widget*, *get_completions*, *on_select=<function <lambda>>*)

Dynamically handle completions on a gtk.Entry.

Parameters

- **`widget`** – a gtk.Entry instance or widget name
- **`get_completions`** – the method to call when a list of completions is requested, returns a list of completions
- **`on_select`** – callback for when a value is selected from the list of completions

`assign_simple_handler` (*widget_name*, *model_attr*, *validator=None*)

Assign handlers to widgets to change fields in the model.

Parameters

- **`widget_name`** –
- **`model_attr`** –
- **`validator`** –

Note: Where widget is a gtk.ComboBox or gtk.ComboBoxEntry then the value is assumed to be stored in model[row][0]

`cleanup`()

Revert any changes the presenter might have done to the widgets so that next time the same widgets are open everything will be normal.

By default it only calls self.view.cleanup()

`clear_problems`()

Clear all the problems from all widgets associated with the presenter

`dirty`()

is the presenter dirty?

the presenter is dirty depending on whether it has changed anything that needs to be committed. This doesn't necessarily imply that the session is not dirty nor is it required to change back to True if the changes are committed.

`has_problems`(*widget*)

Return True/False depending on if widget has any problems attached to it.

`init_enum_combo`(*widget_name*, *field*)

Initialize a gtk.ComboBox widget with name widget_name from enum values in self.model.field

Parameters

- **`widget_name`** –
- **`field`** –

`refresh_view`()

Refresh the view with the model values. This method should be called before any signal handlers are configured on the view so that the model isn't changed when the widget values are set.

Any classes that extend GenericEditorPresenter are required to implement this method.

`remove_problem`(*problem_id*, *problem_widgets=None*)

Remove problem_id from self.problems and reset the background color of the widget(s) in problem_widgets. If problem_id is None and problem_widgets is None then method won't do anything.

Parameters

- **`problem_id`** – the problem to remove, if None then remove any problem from the problem_widget(s)

- **problem_widgets** – a gtk.Widget instance to remove the problem from, if None then remove all occurrences of problem_id regardless of the widget

set_model_attr(*attr*, *value*, *validator=None*)
It is best to use this method to set values on the model rather than setting them directly. Derived classes can override this method to take action when the model changes.

Parameters

- **attr** – the attribute on self.model to set
- **value** – the value the attribute will be set to
- **validator** – validates the value before setting it

start()
Start the presenter. This must be implemented by all classes that subclass *GenericEditorPresenter*

class bauble.editor.**GenericModelViewPresenterEditor**(*model*, *parent=None*)
GenericModelViewPresenterEditor assume that model is an instance of object mapped to a SQLAlchemy table

The editor creates its own session and merges the model into it. If the model is already in another session that original session will not be effected.

When creating a subclass of this editor then you should explicitly close the session when you are finished with it.

Parameters

- **model** – an instance of an object mapped to a SQLAlchemy Table, the model will be copied and merged into self.session so that the original model will not be changed
- **parent** – the parent windows for the view or None

attach_response(*dialog*, *response*, *keyname*, *mask*)
Attach a response to dialog when keyname and mask are pressed

commit_changes()
Commit the changes to self.session()

class bauble.editor.**NotesPresenter**(*presenter*, *notes_property*, *parent_container*)
The NotesPresenter provides a generic presenter for editor notes on an item in the database. This presenter requires that the notes property provide a specific interface.

Parameters

- **presenter** – the parent presenter of this presenter
- **notes_property** – the string name of the notes property of the presenter.model

- **parent_container** – the gtk.Container to add the notes editor box to

add_note (*note=None*)
Add a new note to the model.

5.4.5 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*

5.4.6 bauble.ui

class bauble.ui.**GUI**
Bases: object

add_menu (*name*, *menu*, *index=-1*)
add a menu to the menubar

Parameters

- **name** –
- **menu** –
- **index** –

add_to_history (*text*, *index=0*)
add text to history, if text is already in the history then set its index to index parameter

add_to_insert_menu (*editor*, *label*)
add an editor to the insert menu

Parameters

- **editor** – the editor to add to the menu
- **label** – the label for the menu item

build_tools_menu ()
Build the tools menu from the tools provided by the plugins.

This method is generally called after plugin initialization

clear_menu (*path*)
remove all the menus items from a menu

create_main_menu ()
get the main menu from the UIManager XML description, add its actions and return the menubar

get_view()
: return the current view in the view box

on_file_menu_open(*widget*, *data=None*)
: Open the connection manager.

on_go_button_clicked(*widget*)

on_tools_menu_item_activate(*widget*, *tool*)
: Start a tool on the Tool menu.

save_state()
: this is usually called from bauble.py when it shuts down

set_view(*view=None*)
: set the view, if view is None then remove any views currently set

    Parameters **view** – default=None

show_message_box(*msg*)
: Show an info message in the message drop down box

statusbar_clear()
: Call gtk.Statusbar.pop() for each context_id that had previously been pushed() onto the the statusbar stack. This might not clear all the messages in the statusbar but its the best we can do without knowing how many messages are in the stack.

5.4.7 bauble.meta

bauble.meta.**get_default**(*name*, *default=None*, *session=None*)
: Get a BaubleMeta object with name. If the default value is not None then a BaubleMeta object is returned with name and the default value given.

    If a session instance is passed (session != None) then we don't commit the session.

class bauble.meta.**BaubleMeta**(***kwargs*)
: Bases: `sqlalchemy.ext.declarative.api.Base`

    The BaubleMeta class is used to set and retrieve meta information based on key/name values from the bauble meta table.

    Table name bauble

    Columns

    name: The name of the data.

    value: The value.

5.4.8 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.

5.4.9 `bauble.pluginmgr`

Manage plugin registry, loading, initialization and installation. The plugin manager should be started in the following order:

1. load the plugins: search the plugin directory for plugins, populates the plugins dict (happens in load())

2. install the plugins if not in the registry, add properly installed plugins in to the registry (happens in load())

3. initialize the plugins (happens in init())

`bauble.pluginmgr.`**`register_command`**`(`*handler*`)`
: Register command handlers. If a command is a duplicate then it will overwrite the old command of the same name.

Parameters **`handler`** – A class which extends pluginmgr.CommandHandler

`bauble.pluginmgr.`**`load`**`(`*path=None*`)`
: Search the plugin path for modules that provide a plugin. If path is a directory then search the directory for plugins. If path is None then use the default plugins path, bauble.plugins.

This method populates the pluginmgr.plugins dict and imports the plugins but doesn't do any plugin initialization.

Parameters **`path`** (*str*) – the path where to look for the plugins

`bauble.pluginmgr.`**`init`**`(`*force=False*`)`
: Initialize the plugin manager.

1. Check for and install any plugins in the plugins dict that aren't in the registry. 2. Call each init() for each plugin the registry in order of dependency 3. Register the command handlers in the plugin's commands[]

NOTE: This should be called after after Bauble has established a connection to a database with db.open()

`bauble.pluginmgr.`**`install`**`(`*plugins_to_install*, *import_defaults=True*, *force=False*`)`

Parameters

- **plugins_to_install** – A list of plugins to install. If the string "all" is passed then install all plugins listed in the bauble.pluginmgr.plugins dict that aren't already listed in the plugin registry.
- **import_defaults** (*bool*) – Flag passed to the plugin's install() method to indicate whether it should import its default data.
- **force** (*book*) – Force, don't ask questions.

class `bauble.pluginmgr.`**`Plugin`**

tools: a list of BaubleTool classes that this plugin provides, the tools' category and label will be used in Bauble's "Tool" menu

depends: a list of names classes that inherit from BaublePlugin that this plugin depends on

cmds: a map of commands this plugin handled with callbacks, e.g dict('cmd', lambda x: handler)

description: a short description of the plugin

classmethod **`init`**`()`
init() is run when Bauble is first started

classmethod **`install`**`(`*import_defaults=True*`)`
install() is run when a new plugin is installed, it is usually only run once for the lifetime of the plugin

class `bauble.pluginmgr.`**`Tool`**

class `bauble.pluginmgr.`**`View`**`(`**args*, ***kwargs*`)`

class `bauble.pluginmgr.`**`CommandHandler`**

5.4.10 bauble.prefs

`bauble.prefs.`**`default_prefs_file`** **= '/home/docs/.bauble/config'**
The default file for the preference settings file.

`bauble.prefs.`**`config_version_pref`** **= 'bauble.config.version'**
The preferences key for the bauble version of the preferences file.

`bauble.prefs.`**`date_format_pref`** **= 'bauble.default_date_format'**
The preferences key for the default data format.

`bauble.prefs.`**`parse_dayfirst_pref`** **= 'bauble.parse_dayfirst'**
The preferences key for to determine whether the date should come first when parsing date string. For more information see the `dateutil.parser.parse()` method.

Values: True, False

`bauble.prefs.`**`parse_yearfirst_pref`** **= 'bauble.parse_yearfirst'**
The preferences key for to determine whether the date should come first when parsing date string. For more information see the `dateutil.parser.parse()` method.

Values: True, False

`bauble.prefs.`**`units_pref`** **= 'bauble.units'**

The preferences key for the default units for Bauble.

Values: metric, imperial

5.4.11 `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()`*

5.4.12 `bauble.types`

class `bauble.btypes.`**`Enum`**`(`*values*, *empty_to_none=False*, *strict=True*, *translations={}*, ***kwargs*`)`

Bases: `sqlalchemy.sql.type_api.TypeDecorator`

A database independent Enum type. The value is stored in the database as a Unicode string.

class `bauble.btypes.`**`Date`**`(`**args*, ***kwargs*`)`

Bases: `sqlalchemy.sql.type_api.TypeDecorator`

A Date type that allows Date strings

class `bauble.btypes.`**`DateTime`**`(`**args*, ***kwargs*`)`

Bases: `sqlalchemy.sql.type_api.TypeDecorator`

A DateTime type that allows strings

5.4.13 `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

5.4.14 `bauble.view`

class `bauble.view.`**`Action`**(*name*, *label*, *tooltip=None*, *stock_id=None*, *callback=None*, *accelerator=None*, *multiselect=False*, *singleselect=True*)

Bases: `gtk.Action`

An Action allows a label, tooltip, callback and accelerator to be called when specific items are selected in the SearchView

class `bauble.view.`**`InfoBox`**(*tabbed=False*)

Bases: `gtk.Notebook`

Holds list of expanders with an optional tabbed layout.

The default is to not use tabs. To create the InfoBox with tabs use InfoBox(tabbed=True). When using tabs then you can either add expanders directly to the InfoBoxPage or using InfoBox.add_expander with the page_num argument.

Also, it's not recommended to create a subclass of a subclass of InfoBox since if they both use bauble.utils.BuilderWidgets then the widgets will be parented to the infobox that is created first and the expanders of the second infobox will appear empty.

`add_expander`(*expander*, *page_num=0*)

Add an expander to a page.

Parameters

- **`expander`** – The expander to add.
- **`page_num`** – The page number in the InfoBox to add the expander.

`on_switch_page`(*notebook*, *dummy_page*, *page_num*, **args*)

Called when a page is switched

`update`(*row*)

Update the current page with row.

class `bauble.view.`**`InfoBoxPage`**

Bases: `gtk.ScrolledWindow`

A `gtk.ScrolledWindow` that contains *`bauble.view.InfoExpander`* objects.

`add_expander`(*expander*)

Add an expander to the list of exanders in this infobox

Parameters **`expander`** – the bauble.view.InfoExpander to add to this infobox

`get_expander`(*label*)

Returns an expander by the expander's label name

Parameters **`label`** – the name of the expander to return

`remove_expander`(*label*)

Remove expander from the infobox by the expander's label bel

Parameters **`label`** – the name of th expander to remove

Return the expander that was removed from the infobox.

`update`(*row*)
Updates the infobox with values from row

Parameters **`row`** – the mapper instance to use to update this infobox, this is passed to each of the infoexpanders in turn

class `bauble.view.`**`InfoExpander`**(*label*, *widgets=None*)
Bases: `gtk.Expander`

an abstract class that is really just a generic expander with a vbox to extend this you just have to implement the update() method

`set_widget_value`(*widget_name*, *value*, *markup=False*, *default=None*)
a shorthand for L{bauble.utils.set_widget_value()}

`update`(*value*)
This method should be implemented by classes that extend InfoExpander

class `bauble.view.`**`PropertiesExpander`**
Bases: *`bauble.view.InfoExpander`*

`update`(*row*)
” Update the widget in the expander.

class `bauble.view.`**`LinksExpander`**(*notes=None*)
Bases: *`bauble.view.InfoExpander`*

class `bauble.view.`**`SearchView`**
Bases: *`bauble.pluginmgr.View`*

The SearchView is the main view for Bauble. It manages the search results returned when search strings are entered into the main text entry.

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

5.4.15 `bauble.search`

class `bauble.search.`**`SearchParser`**
The parser for bauble.search.MapperSearch

`parse_string`(*text*)
request pyparsing object to parse text

text can be either a query, or a domain expression, or a list of values. the *self.statement* pyparsing object parses the input text and return a pyparsing.ParseResults object that represents the input

class `bauble.search.`**`SearchStrategy`**
Interface for adding search strategies to a view.

`search`(*text*, *session=None*)

Parameters

- **`text`** – the search string
- **`session`** – the session to use for the search

Return an iterator that iterates over mapped classes retrieved from the search.

class `bauble.search.`**`MapperSearch`**

Bases: *`bauble.search.SearchStrategy`*

Mapper Search support three types of search expression: 1. value searches: search that are just list of values, e.g. value1, value2, value3, searches all domains and registered columns for values 2. expression searches: searched of the form domain=value, resolves the domain and searches specific columns from the mapping 3. query searchs: searches of the form domain where ident.ident = value, resolve the domain and identifiers and search for value

`search`(*text*, *session=None*)

Returns a set() of database hits for the text search string.

If session=None then the session should be closed after the results have been processed or it is possible that some database backends could cause deadlocks.

class `bauble.search.`**`QueryBuilder`**(*parent=None*)

Bases: `gtk.Dialog`

5.4.16 `bauble.plugins.plants`

class `bauble.plugins.plants.`**`Family`**(***kwargs*)

Bases: `sqlalchemy.ext.declarative.api.Base`, `bauble.db.Serializable`

Table name family

Columns

family: The name of the family. Required.

qualifier: The family qualifier.

Possible values:

- 19. lat.: aggregrate family (senso lato)
- 19. str.: segregate family (senso stricto)
- ‘’: the empty string

Properties

synonyms: An association to _synonyms that will automatically convert a Family object and create the synonym.

Constraints The family table has a unique constraint on family/qualifier.

class `bauble.plugins.plants.family.`**`FamilySynonym`**`(`*synonym=None*, ***kwargs*`)`

Bases: `sqlalchemy.ext.declarative.api.Base`

Table name family_synonyms

Columns *family_id*:

synonyms_id:

Properties *synonyms*:

family:

class `bauble.plugins.plants.`**`Genus`**`(`***kwargs*`)`

Bases: `sqlalchemy.ext.declarative.api.Base`, `bauble.db.Serializable`

Table name genus

Columns

genus**:** The name of the genus. In addition to standard generic names any additional hybrid flags or genera should included here.

qualifier**:** Designates the botanical status of the genus.

Possible values:

• 19. lat.: aggregrate genus (sensu lato)

• 19. str.: segregate genus (sensu stricto)

author**:** The name or abbreviation of the author who published this genus.

Properties

family**:** The family of the genus.

synonyms**:** The list of genera who are synonymous with this genus. If a genus is listed as a synonym of this genus then this genus should be considered the current and valid name for the synonym.

Contraints The combination of genus, author, qualifier and family_id must be unique.

class `bauble.plugins.plants.genus.`**`GenusSynonym`**`(`*synonym=None*, ***kwargs*`)`

Bases: `sqlalchemy.ext.declarative.api.Base`

Table name genus_synonym

class `bauble.plugins.plants.`**`Species`**`(`**args*, ***kwargs*`)`

Bases: `sqlalchemy.ext.declarative.api.Base`, `bauble.db.Serializable`, `bauble.db.DefiningPictures`

Table name species

Columns *sp*: *sp2*: *sp_author*:

***hybrid*:** Hybrid flag

infrasp1: *infrasp1_rank*: *infrasp1_author*:

infrasp2: *infrasp2_rank*: *infrasp2_author*:

infrasp3: *infrasp3_rank*: *infrasp3_author*:

infrasp4: *infrasp4_rank*: *infrasp4_author*:

cv_group: *trade_name*:

***sp_qual*:** Species qualifier

Possible values: *agg.*: An aggregate species

s. lat.: aggregrate species (sensu lato)

s. str.: segregate species (sensu stricto)

***label_distribution*:** UnicodeText This field is optional and can be used for the label in case str(self.distribution) is too long to fit on the label.

Properties *accessions*:

vernacular_names:

default_vernacular_name:

synonyms:

distribution:

Constraints The combination of sp, sp_author, hybrid, sp_qual, cv_group, trade_name, genus_id

class `bauble.plugins.plants.species.`**`SpeciesSynonym`**(*synonym=None*, ***kwargs*)

Bases: `sqlalchemy.ext.declarative.api.Base`

Table name species_synonym

class `bauble.plugins.plants.species.`**`VernacularName`**(***kwargs*)

Bases: `sqlalchemy.ext.declarative.api.Base`, `bauble.db.Serializable`

Table name vernacular_name

Columns

***name*:** the vernacular name

***language*:** language is free text and could include something like UK or US to identify the origin of the name

***species_id*:** key to the species this vernacular name refers to

Properties

Constraints

class `bauble.plugins.plants.species.`**`DefaultVernacularName`**`(`*`**kwargs`*`)`

Bases: `sqlalchemy.ext.declarative.api.Base`

Table name default_vernacular_name

DefaultVernacularName is not meant to be instantiated directly. Usually the default vernacular name is set on a species by setting the default_vernacular_name property on Species to a VernacularName instance

Columns

id: Integer, primary_key

species_id: foreign key to species.id, nullable=False

vernacular_name_id:

Properties

Constraints

class `bauble.plugins.plants.`**`SpeciesDistribution`**`(`*`**kwargs`*`)`

Bases: `sqlalchemy.ext.declarative.api.Base`

Table name species_distribution

Columns

Properties

Constraints

class `bauble.plugins.plants.`**`Geography`**`(`*`**kwargs`*`)`

Bases: `sqlalchemy.ext.declarative.api.Base`

Represents a geography unit.

Table name geography

Columns *name*:

tdwg_code:

iso_code:

parent_id:

Properties *children*:

Constraints

5.4.17 `bauble.plugins.garden`

class `bauble.plugins.garden.`**`Accession`**`(`*`*args`*`, `*`**kwargs`*`)`

Bases: `sqlalchemy.ext.declarative.api.Base`, `bauble.db.Serializable`

Table name accession

Columns

code: **`sqlalchemy.types.Unicode`** the accession code

prov_type: **`bauble.types.Enum`** the provenance type

Possible values:

- first column of prov_type_values

wild_prov_status: **`bauble.types.Enum`** this column can be used to give more provenance information

Possible values:

- union of first columns of wild_prov_status_values,
- purchase_prov_status_values,
- cultivated_prov_status_values

date_accd: **`bauble.types.Date`** the date this accession was accessioned

id_qual: **`bauble.types.Enum`** The id qualifier is used to indicate uncertainty in the identification of this accession

Possible values:

- aff. - affinity with
- cf. - compare with
- forsan - perhaps
- near - close to
- ? - questionable
- incorrect

id_qual_rank: **`sqlalchemy.types.Unicode`** The rank of the species that the id_qaul refers to.

private: **`sqlalchemy.types.Boolean`** Flag to indicate where this information is sensitive and should be kept private

species_id: **`sqlalchemy.types.Integer()`** foreign key to the species table

Properties

species: the species this accession refers to

source: source is a relation to a Source instance

plants: a list of plants related to this accession

verifications: a list of verifications on the identification of this accession

Constraints

class `bauble.plugins.garden.accession.`**`AccessionNote`**(***kwargs*)

Bases: `sqlalchemy.ext.declarative.api.Base`, `bauble.db.Serializable`

Notes for the accession table

class `bauble.plugins.garden.`**`Plant`**(***kwargs*)

Bases: `sqlalchemy.ext.declarative.api.Base`, `bauble.db.Serializable`, `bauble.db.DefiningPictures`

Table name plant

Columns

***code*: `sqlalchemy.types.Unicode`** The plant code

***acc_type*: `bauble.types.Enum`** The accession type

Possible values:

- Plant: Whole plant
- Seed/Spore: Seed or Spore
- Vegetative Part: Vegetative Part
- Tissue Culture: Tissue culture
- Other: Other, probably see notes for more information
- None: no information, unknown

***accession_id*: `sqlalchemy.types.Integer`** Required.

***location_id*: `sqlalchemy.types.Integer`** Required.

Properties

***accession*:** The accession for this plant.

***location*:** The location for this plant.

***notes*:** The notes for this plant.

Constraints The combination of code and accession_id must be unique.

classmethod **`get_delimiter`**(*refresh=False*)

Get the plant delimiter from the BaubleMeta table.

The delimiter is cached the first time it is retrieved. To refresh the delimiter from the database call with refresh=True.

class `bauble.plugins.garden.plant.`**`PlantNote`**(***kwargs*)

Bases: `sqlalchemy.ext.declarative.api.Base`, `bauble.db.Serializable`

class `bauble.plugins.garden.plant.`**`PlantChange`**(***kwargs*)

Bases: `sqlalchemy.ext.declarative.api.Base`

class `bauble.plugins.garden.plant.`**`PlantStatus`**`(`***kwargs*`)`
Bases: `sqlalchemy.ext.declarative.api.Base`

date: date checked status: status of plant comment: comments on check up checked_by: person who did the check

class `bauble.plugins.garden.`**`Location`**`(`***kwargs*`)`
Bases: `sqlalchemy.ext.declarative.api.Base`, `bauble.db.Serializable`

Table name location

Columns *name*:

description:

Relation *plants*:

class `bauble.plugins.garden.propagation.`**`Propagation`**`(`***kwargs*`)`
Bases: `sqlalchemy.ext.declarative.api.Base`

class `bauble.plugins.garden.propagation.`**`PropRooted`**`(`***kwargs*`)`
Bases: `sqlalchemy.ext.declarative.api.Base`

Rooting dates for cutting

class `bauble.plugins.garden.propagation.`**`PropCutting`**`(`***kwargs*`)`
Bases: `sqlalchemy.ext.declarative.api.Base`

A cutting

class `bauble.plugins.garden.propagation.`**`PropSeed`**`(`***kwargs*`)`
Bases: `sqlalchemy.ext.declarative.api.Base`

class `bauble.plugins.garden.source.`**`Source`**`(`***kwargs*`)`
Bases: `sqlalchemy.ext.declarative.api.Base`

class `bauble.plugins.garden.source.`**`SourceDetail`**`(`***kwargs*`)`
Bases: `sqlalchemy.ext.declarative.api.Base`

class `bauble.plugins.garden.source.`**`Collection`**`(`***kwargs*`)`
Bases: `sqlalchemy.ext.declarative.api.Base`

Table name collection

Columns *collector*: `sqlalchemy.types.Unicode`

collectors_code: `sqlalchemy.types.Unicode`

date: `sqlalchemy.types.Date`

locale: `sqlalchemy.types.UnicodeText`

latitude: `sqlalchemy.types.Float`

longitude: `sqlalchemy.types.Float`

gps_datum: `sqlalchemy.types.Unicode`

geo_accy: `sqlalchemy.types.Float`

elevation: `sqlalchemy.types.Float`

elevation_accy: `sqlalchemy.types.Float`

habitat: `sqlalchemy.types.UnicodeText`

geography_id: `sqlalchemy.types.Integer`

notes: `sqlalchemy.types.UnicodeText`

accession_id: `sqlalchemy.types.Integer`

Properties

Constraints

class `bauble.plugins.garden.accession.`**`Verification`**`(`***kwargs*`)`
Bases: `sqlalchemy.ext.declarative.api.Base`

Table name verification

Columns

verifier: `sqlalchemy.types.Unicode` The name of the person that made the verification.

date: `sqlalchemy.types.Date` The date of the verification

reference: `sqlalchemy.types.UnicodeText` The reference material used to make this verification

level: `sqlalchemy.types.Integer` Determines the level or authority of the verifier. If it is not known whether the name of the record has been verified by an authority, then this field should be None.

Possible values:

- 0: The name of the record has not been checked by any authority.
- 1: The name of the record determined by comparison with other named plants.
- 2: The name of the record determined by a taxonomist or by other competent persons using herbarium and/or library and/or documented living material.
- 3: The name of the plant determined by taxonomist engaged in systematic revision of the group.
- 4: The record is part of type gathering or propagated from type material by asexual methods

notes: `sqlalchemy.types.UnicodeText` Notes about this verification.

accession_id: `sqlalchemy.types.Integer` Foreign Key to the *`Accession`* table.

species_id: `sqlalchemy.types.Integer` Foreign Key to the `Species` table.

prev_species_id: `Integer` Foreign key to the `Species` table. What it was verified from.

class `bauble.plugins.garden.accession.`**`Voucher`**`(`***kwargs*`)`

Bases: `sqlalchemy.ext.declarative.api.Base`

Table name voucher

Columns

herbarium: `sqlalchemy.types.Unicode` The name of the herbarium.

code: `sqlalchemy.types.Unicode` The herbarium code.

parent_material: `sqlalchemy.types.Boolean` Is this voucher the parent material of the accession. E.g did the seed for the accession from come the plant used to make this voucher.

accession_id: `sqlalchemy.types.Integer` A foreign key to `Accession`

5.4.18 `bauble.plugins.abcd`

`bauble.plugins.abcd.`**`validate_xml`**`(`*root*`)`

Validate root against ABCD 2.06 schema

Parameters **`root`** – root of an XML tree to validate against

Returns True or False depending if root validates correctly

`bauble.plugins.abcd.`**`create_abcd`**`(`*decorated_objects*, *authors=True*, *validate=True*`)`

Parameters

- **`objects`** – a list/tuple of objects that implement the ABCDDecorator interface
- **`authors`** – flag to control whether to include the authors in the species name
- **`validate`** – whether we should validate the data before returning

Returns a valid ABCD ElementTree

class `bauble.plugins.abcd.`**`ABCDAdapter`**`(`*obj*`)`

An abstract base class for creating ABCD adapters.

`extra_elements``(`*unit*`)`

Add extra non required elements

`get_AuthorTeam``()`

Get the Author string.

get_FirstEpithet ()
Get the first epithet.

get_FullScientificNameString (*authors=True*)
Get the full scientific name string.

get_GenusOrMonomial ()
Get the Genus string.

get_InformalNameString ()
Get the common name string.

get_UnitID ()
Get a value for the UnitID

get_family ()
Get a value for the family.

class `bauble.plugins.abcd.`**ABCDExporter**
Export Plants to an ABCD file.

5.4.19 `bauble.plugins.imex`

5.4.20 `bauble.plugins.report`

5.4.21 `bauble.plugins.report.xsl`

The PDF report generator module.

This module takes a list of objects, get all the plants from the objects, converts them to the ABCD XML format, transforms the ABCD data to an XSL formatting stylesheet and uses a XSL-PDF renderer to convert the stylesheet to PDF.

5.4.22 `bauble.plugins.report.mako`

5.4.23 `bauble.plugins.tag`

`bauble.plugins.tag.`**remove_callback** (*tags*)

Parameters **tags** – a list of *Tag* objects.

`bauble.plugins.tag.`**get_tagged_objects** (*tag*, *session=None*)
Return all object tagged with tag.

Parameters

- **tag** – A string or *Tag*
- **session** –

`bauble.plugins.tag.`**untag_objects** (*name*, *objs*)
Remove the tag name from objs.

Parameters

- **`name`** (*str*) – The name of the tag
- **`objs`** (*list*) – The list of objects to untag.

`bauble.plugins.tag.`**`tag_objects`**(*name*, *objs*)

Tag a list of objects.

Parameters

- **`name`** (*str*) – The tag name, if its a str object then it will be converted to unicode() using the default encoding. If a tag with this name doesn't exist it will be created
- **`obj`** (*list*) – A list of mapped objects to tag.

`bauble.plugins.tag.`**`get_tag_ids`**(*objs*)

Parameters **`objs`** – a list or tuple of objects

Return a list of tag id's for tags associated with obj, only returns those tag ids that are common between all the objs

class `bauble.plugins.tag.`**`Tag`**(***kwargs*)

Table name tag

Columns

tag: `sqlalchemy.types.Unicode` The tag name.

description: `sqlalchemy.types.Unicode` A description of this tag.

class `bauble.plugins.tag.`**`TaggedObj`**(***kwargs*)

Table name tagged_obj

Columns

obj_id: `sqlalchemy.types.Integer` The id of the tagged object.

obj_class: `sqlalchemy.types.Unicode` The class name of the tagged object.

tag_id: `sqlalchemy.types.Integer` A ForeignKey to *`Tag`*.

class `bauble.plugins.tag.`**`TagItemGUI`**(*values*)

Interface for tagging individual items in the results of the SearchView

CHAPTER 6

Supporting Bauble

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

Python Module Index

Index