pylax — Database front end

Binaries: http://www.pylax.org **Source code:** https://github.com/thomasfuhringer/pylax

---

Pylax is a database front end for SQLite that uses embedded Python as scripting language.

To use Pylax you download the Linux binaries, extract them to a directory and execute *start.sh*.

Hello World

In keeping with good tradition we shall run a very minimum application which is called *Hello World*. It is available here: https://github.com/thomasfuhringer/pylax/tree/master/Apps/Hello.pxa

For Pylax to start a user app it requires two files: a SQLite database with a file name ending in *.px* and a Python script file named *main.py* in the same directory. A larger app might have several Python files but one needs to be named *main.py*.

Let us look what Hello World does:

```
import pylax

form = pylax.Form(None, 20, 10, 680, 480, "Hello World")
label = pylax.Label(form, 30, 30, -20, -20, "Welcome to Pylax!")
label.alignHoriz, label.alignVert = pylax.Align.center, pylax.Align.center
```

At first it imports **pylax**. This is the connecting point of the script with the host application.:

```
form = pylax.Form(None, 20, 10, 680, 480, "Hello World")
```

creates a **Form** object with no parent, coordinates *20, 10, 680, 480* and caption “Hello World”. A **Form** is a subclass of **Widget** and as such can have a parent, within which it is displayed. In the case of a Form, however, this does not make sense, so it is None. Also the coordinates are not used for a **Form** **Widget** as long as MDI is not available.:

```
label = pylax.Label(form, 30, 30, -20, -20, "Welcome to Pylax!")
```

label is an instance of **Label** **Widget** and a child widget of *form*. It is displayed within the Form at position *30, 30* from top left and its bottom right corner is *20, 20* from the bottom right of the parent widget. If the letter two parameters were positive they would decribe the width and hight of the **Widget** object.:

```
label.alignHoriz, label.alignVert = pylax.Align.center, pylax.Align.center
```

To center the Label’s caption “Welcome to Pylax!” within the Widget’s dimensions we assign it a center alignment horizontally and vertically.

Managing Data

Obviously a database application without data is not all that interesting. So let’s look at an example that actually pulls data and allows manipulating it.

We use a simple table with this definition:

```
CREATE TABLE Item (
    ItemID                  INTEGER PRIMARY KEY,
    Name                    TEXT    UNIQUE,
    Description             TEXT,
    Picture                 BLOB,
    Price                   REAL);
```

and a very basic script to browse and edit it:

```
import pylax

ds = pylax.Dynaset("Item", "SELECT ItemID, Name, Description, Picture, Price FROM Item;")
ds.autoColumn = ds.add_column("ItemID", int, format="{:,}", key=True)
ds.add_column("Name")
ds.add_column("Description", str)
ds.add_column("Picture", bytes)
ds.add_column("Price", float)

form = pylax.Form(None, 20, 10, 680, 480, "Test Form", ds)

labelFormCaption = pylax.Label(form, 1, 1, 40, 20, dynaset=ds, column="Name", visible=False)
labelFormCaption.captionClient = form # passes any assigment to property 'data' on to proper

ds.buttonEdit = pylax.Button(form, -360, -40, 60, 20, "Edit")
ds.buttonNew = pylax.Button(form, -290, -40, 60, 20, "New")
ds.buttonDelete = pylax.Button(form, -220, -40, 60, 20, "Delete")
ds.buttonUndo = pylax.Button(form, -150, -40, 60, 20, "Undo")
ds.buttonSave = pylax.Button(form, -80, -40, 60, 20, "Save")

selectionTable = pylax.Table(form, 20, 50, -320, -50, dynaset=ds, label = pylax.Label(form, 2
selectionTable.add_column("Name", 70, "Name")
selectionTable.add_column("Description", 100, "Description")
selectionTable.showRowIndicator = True

entryID = pylax.Entry(form, -200, 60, 40, 20, dynaset=ds, column="ItemID", dataType=int, labe
entryID.editFormat="{:,}"
entryID.alignHoriz = pylax.Align.left
entryName = pylax.Entry(form, -200, 90, -110, 20, dynaset=ds, column="Name", dataType=str, la
entryPrice = pylax.Entry(form, -200, 120, -110, 20, dynaset=ds, column="Price", dataType=floa
entryPrice.format="{0:,.2f}"

r = ds.execute()
```

Here we use a **`Dynaset`** to select data from the database and hold it in an internal table.

...

Module Functions and Constants

`pylax.`**`append_menu_item`**(*menuItem*)

Adds a **`MenuItem`** to the 'Apps' menu of Pylax.

`pylax.`**`message`**(*message*[, *title*])

Shows a message box displaying the string *message*, using the string *title* as window title.

`pylax.`**`status_message`**(*message*)

Displays string *message* in the status bar.

`pylax.`**`version_info`**

The version number as a tuple of integers.

`pylax.`**`copyright`**

Copyright notice.

Enumerations

`pylax.`**`Align`**

`Enum` for alignment of text in widgets, possible values: *left*, *right*, *center*, *top*, *bottom*, *block*

Classes

Dynaset

class `pylax.`**`Dynaset`**(*table*[, *query*, *parent*, *connecion*])

A dynaset manages the data traffic between the database and **`Widget`** objects. It relates to a table in the database and holds a subset of the data in it. The data can be manipulated by **`Widget`** objects and written back to the database after that. For display purposes it can also hold data from related tables in columns which are not written back to the database. *table* is the name of the primary table in the database. *query* is the SQL string used to pull data. If a *parent* **`Dynaset`** is given it will be used to synchronize a master-detail relationship. *connecion* can be a **`sqlite3.Connection`** to be used instead of the default connection.

Attributes and methods

`parent`

Master **`Dynaset`**

`autoColumn`

Can point to one of the Dynaset's **`DynasetColumn`** objects to indicate that the value in this column will be generated by the database on insert.

`lastRowID`

Row ID generated by the database for **`autoColumn`** at last insert.

`row`

A Dynaset has a row pointer. Through this attribute it is possible to get or set the current index number of the current row.

-1 means no row is selected.

`rows`

Row count. -1 if still not executed.

`query`

Query string used to pull data.

`autoExecute`

If **`True`** **`execute()`** will be triggered if parent row has changed.

`buttonNew`

A **`Button`** assigned here will enable the user to insert a new row. It will be enabled and disabled as appropriate according to the state of the Dynaset.

`buttonEdit`

A **`Button`** assigned here will enable the user to set the Dynaset and all child Dynasets into Edit mode. It will be enabled and disabled as appropriate according to the state of the Dynaset.

`buttonUndo`

A **`Button`** assigned here will enable the user to revert changes of the current row. It will be enabled and disabled as appropriate according to the state of the current row.

`buttonSave`

A **`Button`** assigned here will enable the user to save changes in the Dynaset and all child Dynasets. It will be enabled and disabled as appropriate according to the state of the Dynaset and children.

`buttonDelete`

A **`Button`** assigned here will enable the user to mark the current row for deletion. It will be enabled and disabled as appropriate according to the state of the Dynaset.

`buttonOK`

A **`Button`** assigned here will be enabled if a row is selected. It is for use in record selector dialogs.

`frozen`

If **`True`** the Dynaset is in the frozen state. This means that some child **`Dynaset`** is in the process of being edited or has been changed. For that reason the user will not be allowed to change the current row. Bound Widgets serving as row selectors (currently only **`Table`**) are disabled.

`on_parent_changed`
A callback assigned here will be triggered every time the selected record in the parent Dataset has changed.

`on_changed`
A callback assigned here will be triggered every time the parent Dataset has changed. The signature of the callback is *on_changed(self, row, column)*.

`validate`
A callback assigned here will be called before save. If it does not return **`True`** saving will be discontinued.

`whoCols`
If **`True`** the table has columns *ModDate* and *ModUser* and these will be populated automatically.

`connection`
`sqlite3.Connection` used to access the database.

`add_column`(*name*[, *type*, *key*, *format*, *default*, *defaultFunction*, *parent*])
Constructs a **`DynasetColumn`** and adds it to the Dynaset.

`get_column`(*name*)
Returns the column with the given name.

`execute`([*parameters*, *query*])
Run the query and load the result set into the internal table.

parameters is a Python dict of paramenters with values that will be substituted in the query.

query will be set as the Dynaset's new query before running, if given.

`get_row`([*number*])
Returns the **`DynasetRow`** with the given row number or, if no number provided, the current row.

`get_data`(*column*[, *row*])
Returns the data value for a given column and row. *column* can be a str with the name of the column or a **`DynasetColumn`** If *row* is not provided, the current row will be used.

`set_data`(*column*, *data*[, *row*])
Sets the data value for a given column and row.

column can be a str with the name of the column or a **`DynasetColumn`** object.

If *row* is not provided, the current row will be used.

`get_row_data`([*row*])
Returns the **`DynasetRow`** at the given row. If *row* is not provided, the current row will be used.

`get_column_data_sum`(*column*)
Adds all the values in the given column and returns it as int or float (depending on the data type associated with the column). *column* can be a str with the name of the column or a **`DynasetColumn`**

`clear`()
Deletes all the rows of data.

`save`()
Writes all changes to the corresponding database table.

Named tuples

A **Dynaset** stores the data retrieved from the database in a table the elements of which can be accessed as **namedtuple** objects.

pylax.**DynasetColumn**

A **namedtuple** with these elements:

name: Name of column in query

index: Position in query

type: Data type

key: True if column is part of the primary key, False if non-key database column, None if not part of the database table

default: Default value

get_default: Function providing default value

format: Default display format

parent: Coresponding column in parent Dynaset

pylax.**DynasetRow**

A **namedtuple** with these elements:

data: Tuple of data pulled, in the order as queried

dataOld: Tuple of data before modification, or None if the row is clean

new: True if the row is still not in database

delete: True it the row is to be removed from the database

Widget

class pylax.**Widget**(*parent*[, *left*, *top*, *right*, *bottom*, *caption*, *dynaset*, *column*, *dataType*, *format*, *label*, *visible*])

Widget is the base class from which all data aware widgets that can be displayed on a **Window** are derived.

The construtor parameters are also available as attributes:

parent

The parent **Widget** within which it is displayed, in many cases a **Form**.

caption

Only used with select subclasses (**Window**, **Form**, **Entry**).

dynaset

The **Dynaset** object the widget is bound to.

dataColumn

The **DynasetColumn** object of the **Dynaset** object the widget is bound to.

dataType

The Python data type the widget can hold.

format

The Python format string in the **str.format()** syntax that is used to render the data.

editFormat

The Python format string in the **str.format()** syntax that is used to render the data when in edit mode.

window

The **Window** the Widget is on.

left
: Distance from left edge of parent, if negative from right.

top
: Distance from top edge of parent, if negative from bottom

right
: Width or, if zero or negative, distance of right edge from right edge of parent

bottom
: Height or, if zero or negative, distance of bottom edge from from bottom edge of parent

visible
: By default **True**

label
: A Widget can have a **Label** Widget attached to it. This way certain operations on the Widget will have an effect on both Widgets (currently unused).

data
: The data value currently held

Window

class pylax.**Window**

A GUI window which serves as a canvas to hold **Widget** objects. Itself it is derived from **Widget**. It serves also as the base class for **Form**.

Attributes and methods

parent
: The parent widget within which it is displayed, in many cases a **Form**.

caption
: Assigning a str here makes it appear as the Form's title.

nameInCaption
: If **True** the name of the Form is prepended if a caption is assigned to **caption**.

before_close
: A callable assigned here is called right before the Window is closed, typically to perform cleanup tasks.

focus
: The **Widget** that currently holds the keyboard focus

visible
: By default **True**.

readOnly
: If not **True**, the data will not be editable.

validate
: A callback assigned here will be called if the content was changed and the user tries to navigate to another widget. If it does not return **True** change of focus will be undone.

position
: A tuple with the *(x, y)* position of the widget on the screen.

size
: A tuple with the *(width, height)* of the widget on the screen.

minWidth

The minimum with of the Window.

minHeight

The minimum height of the Window.

close()

Close and destroy

wait_for_close()

This blocks until the window is closed (by some other callback). For using the Window as a dialog.

Form

class pylax.**Form**

A Form is the primary container to display Pylax widgets. A Pylax app will usually open at least one Form. Currently it is implemented as a tab in the client area. The aspiration for the future is to allow an MDI mode. Derived from **Window**.

Attributes and methods

caption

Assigning a str here makes it appear as the Form's title.

nameInCaption

If **True** the name of the Form is prepended if a caption is assigned to **caption**.

Label

class pylax.**Label**

Shows boilerplate text on the **Window**.

captionClient

If a **Widget** is assigned here, any value assingned to **data** will be passed on to **caption** of that widget.

Typically the **Form** will be assigned here and the Label bound to a name column so that name will appear as the caption of the **Form**.

Entry

class pylax.**Entry**

Single line data entry

Attributes and methods

inputString

The currently entered input as a raw string.

inputData

The currently entered input converted to the Widget's data type (not yet written to the **Dynaset**, for validation purposes.

alignHoriz

Horizontal alignment. By default pylax.Align.left for data type str and pylax.Align.right for numeric data types

on_click_button

If a callable is assigned here a Find icon (magnifying glass) will be displayed in the Entry and the callable is called when the icon is clicked.

ComboBox

class pylax.**ComboBox**

ComboBox which allows selection from a drop down list of data values.

noneSelectable

If set **True** it is possible to make no selection.

append(*value*[, *key*])

Appends o tuple of (*value*, *key*) to the list of available items. *key* is displayed in the widget, *value* returned as **Data** If *key* is not given, *value* will be assumed.

Button

class pylax.**Button**

Push button to trigger a callback.

on_click

If a callable is assigned here it is called when the button is clicked.

Table

class pylax.**Table**

Displays a **Dynaset** object's data in a table and allows the user to select a row.

columns

List of the Table's **TableColumn** objects

showRowIndicator

If set **True** a column on the left will be displayed that indicates the status of the data in the row.

add_column(*caption*, *width*[, *data*, *type*, *editable*, *format*, *widget*, *autoSize*])

Appends a **TableColumn** *data* must be a **DynasetColumn** of the Table's **Dynaset** object.

TableColumn

class pylax.**TableColumn**

Column used in a **Table**. Not a subclass of **Widget**.

data

Bound **DynasetColumn** object.

type

Data type, must be in line with the data type of the **DynasetColumn**.

format

Display format

Tab

class pylax.**Tab**

A tabbed notebook container. Holds **TabPage** objects.

TabPage

class pylax.**TabPage**(*parent*)

Page in a **Tab** Widget. *parent* in contructor must be the **Tab** object.

Box

class pylax.**Box**

Container that can hold other Widgets.

A **Splitter** object holds two Box Widgets.

Splitter

class pylax.**Splitter**

A widget with two adjustable panes. Each one is a **Box** object.

box1

Box object on the left or top.

box2

Box object on the right or bottom.

vertical

If **True** panes are arranged top/bottom.

box1Resize

If **True** Box 1 resizes with window

box2Resize

If **True** Box 2 resizes with window

position

Position of the separator

spacing

Width of the separator

Image

class pylax.**Image**

Displays an image bitmap. Currently only JPG format is supported and the size is converted to 320 x 320. For this reason the widget should be sized 320 x 320 on the screen. In Edit mode a click on the widget brings up a file selector dialog which allows to select a JPG file to be imported (and converted to a size of 320 x 320).

Canvas

class pylax.**Canvas**

A widget for very basic drawing.

on_paint

Drawing operations should be done in a callback which is assigned here.

penColor

A tuple of (R, G. B) values to be used for drawing.

fillColor

A tuple of (R, G. B) values to be used for filling rectangles.

textColor

A tuple of (R, G. B) values to be used for drawing text.

point(*x*, *y*)

Draws a point at the given coordinates.

move_to(*x*, *y*)

Sets x, y as the origin of the next line operation.

line_to(*x*, *y*)

Draws a line to x, y.

rectangle(*x*, *y*, *width*, *height*[, *radius*])

Draws a rectangle. If *radius* is provided, corners will be rounded.

text(*x*, *y*, *string*)

Writes *string* at given position.

repaint()

Triggers the paint process.

Menu

class pylax.**Menu**(*caption*)

A submenu.

caption must be a str to be displayed in the menu,

append(*item*)

Add either a **MenuItem** object or a **Menu** object to serve as a submenu.

MenuItem

class pylax.**MenuItem**(*caption*, *on_click*)

An item in a **Menu** object. Can also be inserted into Pylax' *App* menu through **append_menu_item()**.

caption must be a str to be displayed in the menu,

on_click is the callback to be triggered on selection