Sugar Toolkit GTK3

Sugar Toolkit GTK3, or sugar3, is a toolkit for writing Sugar activities in Python:

- write a setup.py which calls `bundlebuilder`,
- write an activity/activity.info file with metadata, see `bundle`,
- write a class derived from `Activity`,
- use the `graphics` module classes to build a user interface.

Python Module Index

sugar3.activity contains following

- `sugar3.activity.activity`
- `sugar3.activity.activityfactory`
- `sugar3.activity.activityhandle`
- `sugar3.activity.activityinstance`
- `sugar3.activity.activityservice`
- `sugar3.activity.bundlebuilder`
- `sugar3.activity.i18n`
- `sugar3.activity.webactivity`
- `sugar3.activity.widgets`
- `sugar3.bundle`
- `sugar3.bundle.activitybundle`
- `sugar3.bundle.bundle`
- `sugar3.bundle.bundleversion`
- `sugar3.bundle.contentbundle`
- `sugar3.bundle.helpers`
- `sugar3.config`

- sugar3.datastore
- sugar3.datastore.datastore
- sugar3.dispatch
- sugar3.dispatch.dispatcher
- sugar3.dispatch.saferef
- sugar3.env
- sugar3.graphics
- sugar3.graphics.alert
- sugar3.graphics.animator
- sugar3.graphics.colorbutton
- sugar3.graphics.combobox
- sugar3.graphics.icon
- sugar3.graphics.iconentry
- sugar3.graphics.menuitem
- sugar3.graphics.notebook
- sugar3.graphics.objectchooser
- sugar3.graphics.palette
- sugar3.graphics.palettegroup
- sugar3.graphics.palettemenu
- sugar3.graphics.palettewindow
- sugar3.graphics.panel
- sugar3.graphics.progressicon
- sugar3.graphics.radiopalette
- sugar3.graphics.radiotoolbutton
- sugar3.graphics.scrollingdetector
- sugar3.graphics.style
- sugar3.graphics.toggletoolbutton
- sugar3.graphics.toolbarbox
- sugar3.graphics.toolbox
- sugar3.graphics.toolbutton
- sugar3.graphics.toolcombobox

- sugar3.graphics.tray
- sugar3.graphics.window
- sugar3.graphics.xocolor
- sugar3.logger
- sugar3.mime
- sugar3.network
- sugar3.power
- sugar3.presence
- sugar3.presence.activity
- sugar3.presence.buddy
- sugar3.presence.connectionmanager
- sugar3.presence.presenceservice
- sugar3.presence.sugartubeconn
- sugar3.presence.tubeconn
- sugar3.profile
- sugar3.speech
- sugar3.test
- sugar3.test.discover
- sugar3.test.uitree
- sugar3.util

sugar3.activity.activity module

A definitive reference for what a Sugar Python activity must do to participate in the Sugar desktop.

Note

This API is STABLE.

The **`Activity`** class is used to derive all Sugar Python activities. This is where your activity starts.

Derive from the class

```
from sugar3.activity.activity import Activity

class MyActivity(Activity):
    def __init__(self, handle):
        Activity.__init__(self, handle)
```

An activity must implement a new class derived from **`Activity`**.

Name the new class MyActivity, where My is the name of your activity. Use bundle metadata to tell Sugar to instantiate this class. See **`bundle`** for bundle metadata.

Create a ToolbarBox

In your `__init__()` method create a `ToolbarBox`, with an `ActivityToolbarButton`, a `StopButton`, and then call `set_toolbar_box()`.

```
from sugar3.activity.activity import Activity
from sugar3.graphics.toolbarbox import ToolbarBox
from sugar3.activity.widgets import ActivityToolbarButton
 from sugar3.activity.widgets import StopButton
class MyActivity(Activity):
    def __init__(self, handle):
        Activity.__init__(self, handle)
        toolbar_box = ToolbarBox()
        activity_button = ActivityToolbarButton(self)
        toolbar_box.toolbar.insert(activity_button, 0)
        activity_button.show()
        separator = Gtk.SeparatorToolItem(draw=False)
        separator.set_expand(True)
        toolbar_box.toolbar.insert(separator, -1)
        separator.show()
        stop_button = StopButton(self)
        toolbar_box.toolbar.insert(stop_button, -1)
        stop_button.show()
        self.set_toolbar_box(toolbar_box)
        toolbar_box.show()
```

sugar3.activity.widgets module

class sugar3.activity.widgets.**ActivityButton**(*activity*, ***kwargs*)

Bases: sugar3.graphics.toolbutton.ToolButton

class sugar3.activity.widgets.**ActivityToolbar**(*activity*, *orientation_left=False*)

Bases: gi.repository.Gtk.Toolbar

The Activity toolbar with the Journal entry title and sharing button

class sugar3.activity.widgets.**ActivityToolbarButton**(*activity*, ***kwargs*)

Bases: sugar3.graphics.toolbarbox.ToolbarButton

class sugar3.activity.widgets.**CopyButton**(***kwargs*)

Bases: sugar3.graphics.toolbutton.ToolButton

class sugar3.activity.widgets.**DescriptionItem**(*activity*, ***kwargs*)

Bases: sugar3.graphics.toolbutton.ToolButton

get_toolbar_box()

set_expanded(*expanded*)

property toolbar_box

class sugar3.activity.widgets.**EditToolbar**

Bases: gi.repository.Gtk.Toolbar

Provides the standard edit toolbar for Activities.

Members:

undo – the undo button redo – the redo button copy – the copy button paste – the paste button separator – A separator between undo/redo and copy/paste

This class only provides the 'edit' buttons in a standard layout, your activity will need to either hide buttons which make no sense for your Activity, or you need to connect the button events to your own callbacks:

Example from Read.activity: # Create the edit toolbar: self._edit_toolbar = EditToolbar(self._view) # Hide undo and redo, they're not needed self._edit_toolbar.undo.props.visible = False self._edit_toolbar.redo.props.visible = False # Hide the separator too: self._edit_toolbar.separator.props.visible = False

 # As long as nothing is selected, copy needs to be insensitive: self._edit_toolbar.copy.set_sensitive(False) # When the user clicks the button, call _edit_toolbar_copy_cb() self._edit_toolbar.copy.connect('clicked', self._edit_toolbar_copy_cb)

 # Add the edit toolbar: toolbox.add_toolbar(_('Edit'), self._edit_toolbar) # And make it visible: self._edit_toolbar.show()

class sugar3.activity.widgets.PasteButton(***kwargs*)

Bases: sugar3.graphics.toolbutton.ToolButton

class sugar3.activity.widgets.RedoButton(***kwargs*)

Bases: sugar3.graphics.toolbutton.ToolButton

class sugar3.activity.widgets.ShareButton(*activity*, ***kwargs*)

Bases: sugar3.graphics.radiopalette.RadioMenuButton

class sugar3.activity.widgets.StopButton(*activity*, ***kwargs*)

Bases: sugar3.graphics.toolbutton.ToolButton

class sugar3.activity.widgets.TitleEntry(*activity*, ***kwargs*)

Bases: gi.repository.Gtk.ToolItem

modify_bg(*self*, *state: Gtk.StateType*, *color: Gdk.Color = None*)

save_title(*activity*)

class sugar3.activity.widgets.UndoButton(***kwargs*)

Bases: sugar3.graphics.toolbutton.ToolButton

sugar3.graphics.icon module

Icons are small pictures that are used to decorate components. In Sugar, icons are SVG files that are re-coloured with a fill and a stroke colour. Typically, icons representing the system use a greyscale color palette, whereas icons representing people take on their selected XoColors.

Designing a Sugar Icon

If you want to make an icon to use in Sugar, start by designing something in a vector program, like Inkscape. When you are designing the icon, use a canvas that is 55x55px.

When designing your icon, use 2 colors. For example, use a black stroke and a white fill color. You need to keep this consistent and remember those colors.

You should also keep the subcell grid in mind when designing the icon. A grid cell (which the size of an icon) is made up of 5 by 5 subcells. To use this in Inkscape, enable the page grid (View -> Page Grid), the go to grid properties (File -> Document Properties -> Grids). In grid properties, set the “Spacing X” option to 11, “Spacing Y” to 11 and “Major grid line every” to 1.

Before your icon is ready to be used in Sugar, it needs to be Sugarized. This converts the colors to SVG entities, which allows Sugar to change the colors of the icon. To do that, just run the sugar-iconify script. Usually, it “just works” like:

```
python path/to/sugar-iconify.py -o icon.svg
```

Code Example

Example of using icons with badges:

```
from gi.repository import Gtk

from sugar3.graphics.icon import EventIcon
from sugar3.graphics.icon import Icon
from sugar3.graphics import style
from sugar3.graphics.xocolor import XoColor
from sugar3.graphics.palette import Palette

import common


test = common.Test()
test.show()

box = Gtk.Box(orientation=Gtk.Orientation.VERTICAL)
test.pack_start(box, True, True, 0)
box.show()

# An XO Icon, normal size, setting the color via the XoColor
object
icon = Icon(icon_name='computer-xo',
            pixel_size=style.STANDARD_ICON_SIZE,
            xo_color=XoColor('#00BEFF,#FF7800'))
box.pack_start(icon, False, False, 0)
icon.show()

# You can mix constructor keyword argument and setting
# properties after creation
icon = EventIcon(icon_name='network-wireless-080',
                 pixel_size=style.STANDARD_ICON_SIZE)
# Badges are little icons displayed
icon.props.badge_name = 'emblem-favorite'
# Instead of using the XoColor, you can use any SVG color
specifier:
icon.props.fill_color = 'rgb(230, 0, 10)'
icon.props.stroke_color = '#78E600'
# Unlike normal icons, EventIcons support palettes:
icon.props.palette = Palette('Hello World')
box.pack_start(icon, False, False, 0)
icon.show()


if __name__ == '__main__':
    common.main(test)
```

Badge Icons

Badge icons are small icons that are attached to the normal icon. For example, a WiFi network icon might have a star badge attached to the bottom left corner (the “attach point”) that indicates that the network is connected to.

Badge icons are displayed at _BADGE_SIZE percent (45% currently), of their main icon.

Badge icons are specified by the icon name, in the same sense that normal icons have a `Icon.set_icon_name` function.

Attach Points

Where the badge icon is placed is defined by the main icon. By default, it is centered on 0, 0. That means that the 3 quarters of the icon will be cut off! Therefore, it is helpful to define the attach points.

When Sugar loads the main icon, it looks for a .icon file. For example, if the icon path is resolved to /theme/computer-xo.svg, the /theme/computer-xo.icon will be tried to find the attach points.

The .icon files are to be formatted as follows:

```
[Icon Data]
AttachPoints=970,850
```

In this example, the badge will be centered at 97.0% on the X axis, and 85.0% on the Y axis.

class sugar3.graphics.icon.**CanvasIcon**(***kwargs*)

sugar3.graphics.toolbox module

A toolbox holds a group of toolbars in a list. One toolbar is displayed at a time. Toolbars are assigned an index and can be accessed using this index. Indices are generated in the order the toolbars are added.

class sugar3.graphics.toolbox.**Toolbox**

Bases: `gi.repository.Gtk.VBox`

Class to represent the toolbox of an activity. Groups a number of toolbars vertically, which can be accessed using their indices. The current toolbar is the only one displayed.

Emits current-toolbar-changed signal when the current toolbar is changed. This signal takes the current page index as an argument.

add_toolbar(*name*, *toolbar*)

Adds a toolbar to this toolbox. Toolbar will be added to the end of this toolbox, and it's index will be 1 greater than the previously added index (index will be 0 if it is the first toolbar added).

Parameters

- **name** (*string*) – name of toolbar to be added
- **toolbar** –
- **toolbox** (*Gtk.Toolbar to be appended to this*) –

property **current_toolbar**

Returns current toolbar.

get_current_toolbar()

Returns current toolbar.

remove_toolbar(*index*)

Removes toolbar at the index specified.

Parameters

index (*int*) – index of the toolbar to be removed

set_current_toolbar(*index*)

Sets the current toolbar to that of the index specified and displays it.

Parameters

index (*int*) – index of toolbar to be set as current toolbar

sugar3.graphics.toolbutton module

The toolbutton module provides the ToolButton class, which is a Gtk.ToolButton with icon and tooltip styled for Sugar.

Example

Add a tool button to a window

from gi.repository import Gtk from sugar3.graphics.toolbutton import ToolButton

def __clicked_cb(button):
print("tool button was clicked")

w = Gtk.Window() w.connect('destroy', Gtk.main_quit) b = ToolButton(icon_name='dialog-ok', tooltip='a tooltip') b.connect('clicked', __clicked_cb) w.add(b) w.show_all()

Gtk.main()

STABLE.

class sugar3.graphics.toolbutton.**ToolButton**(*icon_name=None*, ***kwargs*)

Bases: `gi.overrides.Gtk.ToolButton`

The ToolButton class manages a Gtk.ToolButton styled for Sugar.

Keyword Arguments

- **icon_name** (*string*) – name of themed icon.
- **accelerator** (*string*) – keyboard shortcut to be used to activate this button.

- **tooltip** (*string*) – tooltip to be displayed when user hovers over button.
- **hide_tooltip_on_click** (*bool*) – Whether or not the tooltip is hidden when user clicks on button.

accelerator

Return accelerator that activates the button.

create_palette()

do_clicked()

Implementation method for hiding the tooltip when the button is clicked.

do_draw(*cr*)

Implementation method for drawing the button.

do_get_property(*pspec*)

do_set_property(*pspec*, *value*)

get_accelerator()

Return accelerator that activates the button.

get_hide_tooltip_on_click()

Return True if the tooltip is hidden when a user clicks on the button, otherwise return False.

get_icon_name()

Return icon name, or None if there is no icon name.

get_palette()

get_palette_invoker()

get_tooltip()

Return the tooltip.

hide_tooltip_on_click

Return True if the tooltip is hidden when a user clicks on the button, otherwise return False.

icon_name

Return icon name, or None if there is no icon name.

palette

palette_invoker

set_accelerator(*accelerator*)

Set accelerator that activates the button.

Parameters

accelerator (*string*) – accelerator to be set.

set_hide_tooltip_on_click(*hide_tooltip_on_click*)

Set whether or not the tooltip is hidden when a user clicks on the button.

Parameters

- **hide_tooltip_on_click** (*bool*) – True if the tooltip is
- **otherwise.** (*hidden on click, and False*) –

set_icon_name(*icon_name*)

Set name of icon.

Parameters

icon_name (*string*) – name of icon

set_palette(*palette*)

set_palette_invoker(*palette_invoker*)

set_tooltip(*tooltip*)

Set the tooltip.

Parameters

tooltip (*string*) – tooltip to be set.

tooltip

Return the tooltip.

sugar3.graphics.toolbutton.**setup_accelerator**(*tool_button*)

HelloWorld Activity:
A case study for developing an activity

Sugar codebase has many activities 200-250 which are entirely made up of GTK and Pygame. Different activities are made up of different functions and lots of unique features, hence it's not practically possible to look at the codebase of Sugar activities hence we are discussing Hello-World Activity. We are selecting hello-world activity because it's the most fundamental activity Sugar has as it covers the basics of GTK.

```
import gi
gi.require_version('Gtk', '3.0')
from gi.repository import Gtk

from gettext import gettext as _

from sugar3.activity import activity
from sugar3.graphics.toolbarbox import ToolbarBox
from sugar3.activity.widgets import StopButton
from sugar3.activity.widgets import ActivityToolbarButton


class HelloWorldActivity(activity.Activity):
    """HelloWorldActivity class as specified in activity.info"""

    def __init__(self, handle):
        """Set up the HelloWorld activity."""
        activity.Activity.__init__(self, handle)

        # we do not have collaboration features
        # make the share option insensitive
        self.max_participants = 1

        # toolbar with the new toolbar redesign
        toolbar_box = ToolbarBox()

        activity_button = ActivityToolbarButton(self)
        toolbar_box.toolbar.insert(activity_button, 0)
        activity_button.show()

        separator = Gtk.SeparatorToolItem()
        separator.props.draw = False
        separator.set_expand(True)
        toolbar_box.toolbar.insert(separator, -1)
        separator.show()

        stop_button = StopButton(self)
```

toolbar_box.toolbar.insert(stop_button, -1)
stop_button.show()

self.set_toolbar_box(toolbar_box)
toolbar_box.show()

 # label with the text, make the string translatable
label = Gtk.Label(_("Hello World!"))
self.set_canvas(label)
label.show()

Now let's understand the GTK code used here.

```
import gi
gi.require_version('Gtk', '3.0')
from gi.repository import Gtk
```

the GTK components used in the provided Python code snippet for Hello-world activity:

1. **Imports and Version Requirements:**
   - `import gi`**: Imports the GObject introspection module.**
   - `gi.require_version('Gtk', '3.0')`**: Ensures that the version of GTK being used is 3.0 or later.**
   - `from gi.repository import Gtk`**: Imports the GTK module from GObject introspection.**
2. **Sugar Specific Imports:**
   - `from sugar3.activity import activity`**: Imports the base** `Activity` **class from Sugar.**
   - `from sugar3.graphics.toolbarbox import ToolbarBox`**: Imports the** `ToolbarBox` **class from Sugar for managing toolbars.**
   - `from sugar3.activity.widgets import StopButton`**: Imports the** `StopButton` **widget for stopping activities.**
   - `from sugar3.activity.widgets import ActivityToolbarButton`**: Imports the** `ActivityToolbarButton` **widget for activity-specific toolbar buttons.**
3. **Activity Class Definition:**
   - `HelloWorldActivity(activity.Activity)`**: Defines a class** `HelloWorldActivity` **that inherits from** `activity.Activity`**, representing a Sugar activity.**
4. **Initialization (**`__init__` **method):**
   - `activity.Activity.__init__(self, handle)`**: Initialises the activity, passing** `handle` **to the superclass constructor.**
5. **Setting up Activity Specifics:**

- `self.max_participants = 1`: Sets `max_participants` to 1, indicating the activity doesn't support collaboration.

6. Creating the Toolbar:
   - `toolbar_box = ToolbarBox()`: Creates a `ToolbarBox` instance for managing the toolbar.
   - `activity_button = ActivityToolbarButton(self)`: Creates an `ActivityToolbarButton` instance for the activity.
   - `toolbar_box.toolbar.insert(activity_button, 0)`: Inserts `activity_button` at the beginning of the toolbar.
   - `separator = Gtk.SeparatorToolItem()`: Creates a separator (`SeparatorToolItem`) for spacing in the toolbar.
   - `toolbar_box.toolbar.insert(separator, -1)`: Inserts `separator` at the end of the toolbar.
   - `stop_button = StopButton(self)`: Creates a `StopButton` instance for stopping the activity.
   - `toolbar_box.toolbar.insert(stop_button, -1)`: Inserts `stop_button` at the end of the toolbar.
7. Displaying the Toolbar:
   - `self.set_toolbar_box(toolbar_box)`: Sets the `ToolbarBox` instance as the activity's toolbar.
   - `toolbar_box.show()`: Shows the `ToolbarBox`.
8. Setting the Canvas (Main Content):
   - `label = Gtk.Label(_("Hello World!"))`: Creates a `Gtk.Label` widget with the text "Hello World!" (using translation).
   - `self.set_canvas(label)`: Sets the label as the main content (canvas) of the activity.
   - `label.show()`: Shows the label.

In summary, this code sets up a basic Sugar activity (`HelloWorldActivity`) with a toolbar containing an activity button, a separator, and a stop button. The main content of the activity is a label that displays "Hello World!".