"import wx": a Tale of Never-ending GUI Power

PyAr – November 16, 2012

Andrea Gavana
Maersk Oil

andrea.gavana@gmail.com
andrea.gavana@maerskoil.com

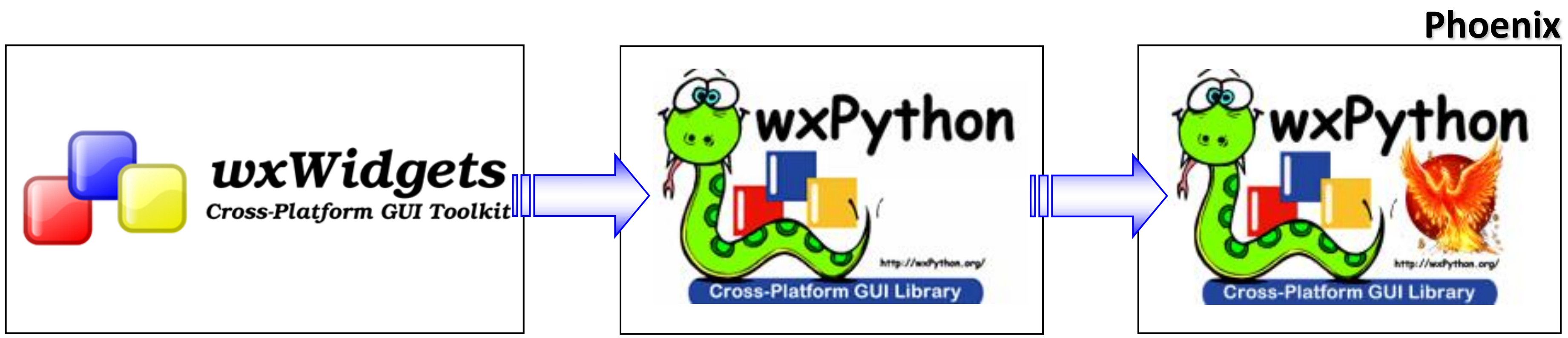
Outline

- ✓ Introduction
- ✓ wxPython architecture and package structures
- ✓ How-Tos:
  - Windows layout management
  - wxPython with threads and multiprocessing
  - Persisting GUI states
- ✓ AGW library and owner-drawn controls
- ✓ Lessons learned
- ✓ wxPython with Python 3

Presentation samples: http://www.infinity77.net/pycon/wxPython.zip

Introduction

- ✓ **wxPython** is a GUI toolkit for Python, built on the wxWidgets C++ framework
- ✓ Designed to be cross-platform, supports Windows, Linux, Unix and Mac
- ✓ Uses **native widgets** wherever possible
- ✓ Extensive library of examples, wonderful community
- ✓ wxWidgets (1992) and wxPython (1996) are mature and robust projects
- ✓ The next generation of wxPython (**Phoenix**) is almost a reality


Phoenix
wxWidgets
Cross-Platform GUI Toolkit
wxPython
http://wxPython.org/
Cross-Platform GUI Library
wxPython
http://wxPython.org/
Cross-Platform GUI Library

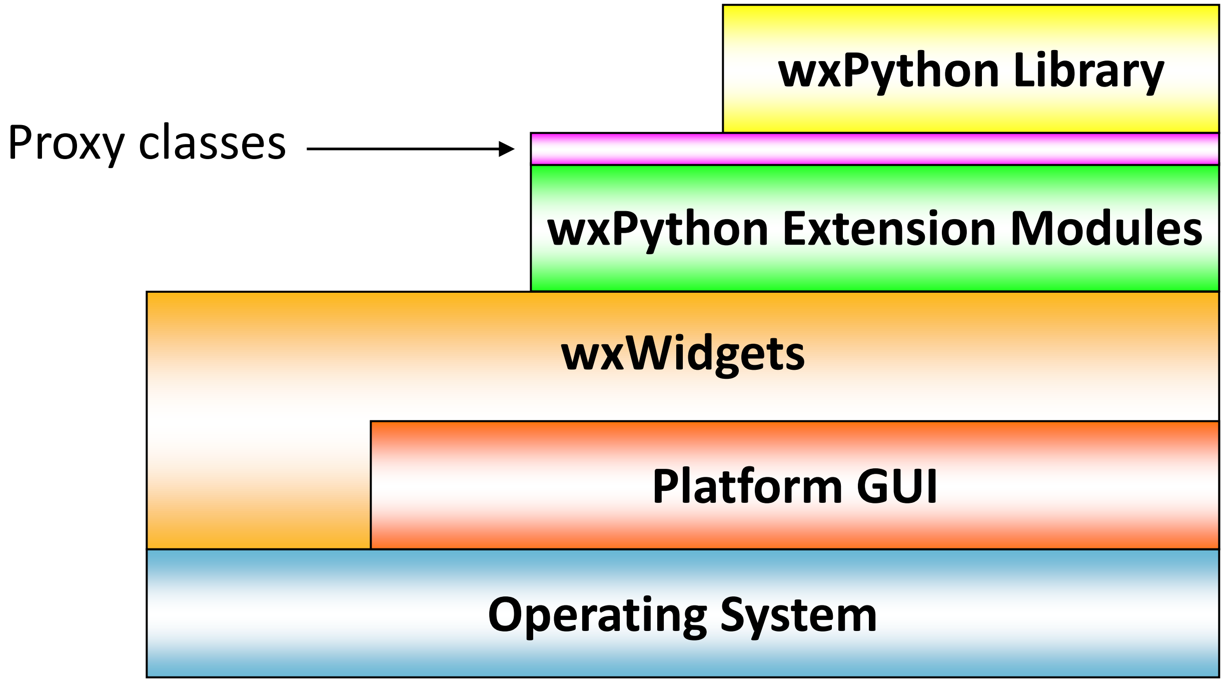
Why wxPython?

- ✓ Native look and feel on all platforms
- ✓ Vast number of widgets (native and owner-drawn)
- ✓ Permissive license
- ✓ Fast evolving pace and excellent maintenance
- ✓ *"The only reason wxPython isn't the standard Python GUI toolkit is that TkInter was there first."* (Guido van Rossum) ☺

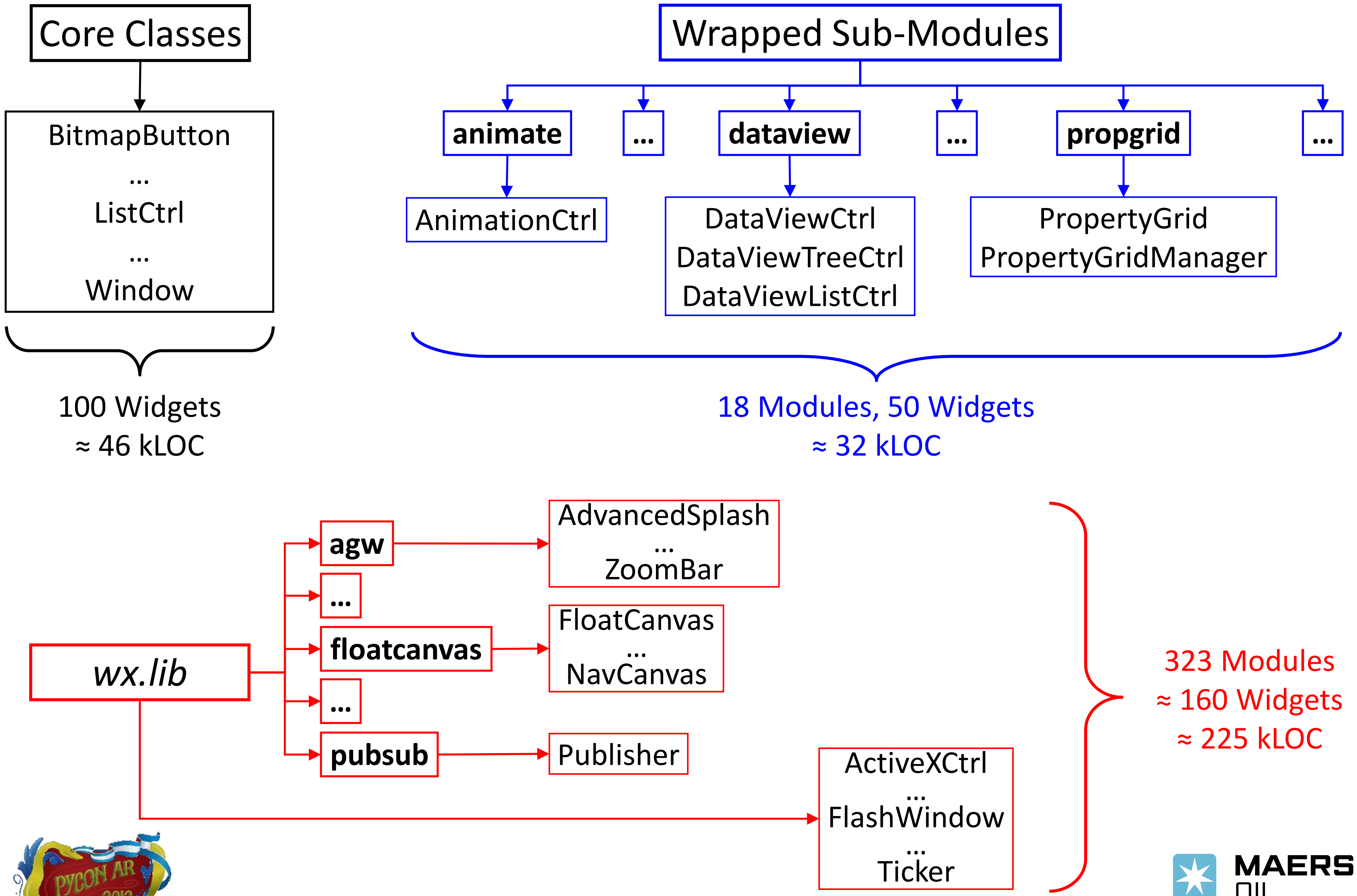
Choice is hard for newcomers:

- ✓ High quality alternatives (PyQt, PySide, PyGtk, TkInter)
- ✓ Try them all, choose **The One**

Architecture

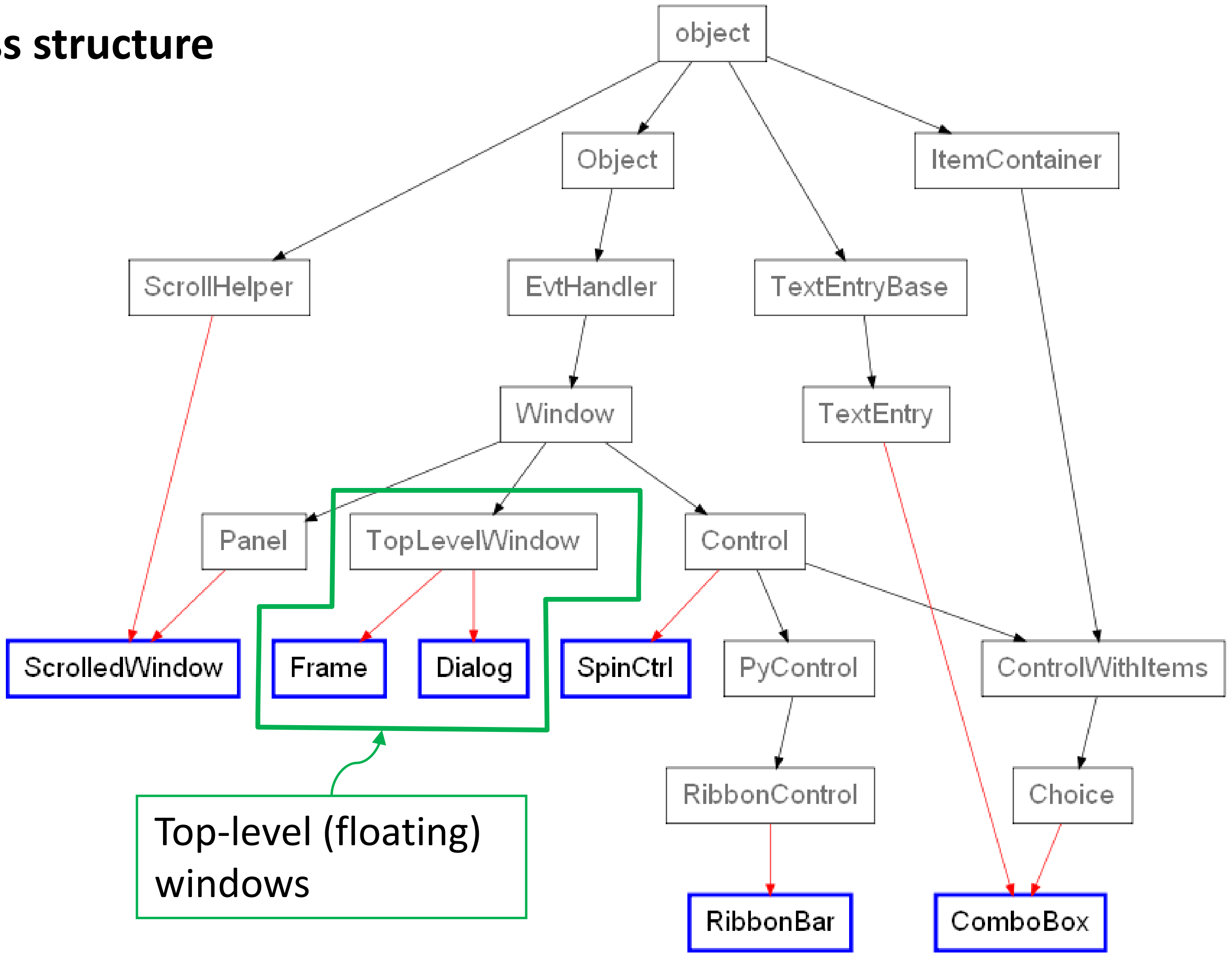


Architecture



Architecture

Partial class structure



Windows Layout Management

Layout management: describes how widgets are laid out in an application's user interface

✓ wxPython provides a few alternatives:

- *Absolute positioning (brute force)*: don't. No, really, **don't**
- *Sizers*: very powerful, not so simple at the beginning, but worth the effort
- *SizedControls*: add-on library to help simplify the creation of sizer-based layouts
- *AUI (Advanced User Interface)*: docking windows and automatic layout management

✓ My recommendation is to use sizers and AUI, depending on the layout you wish to build:

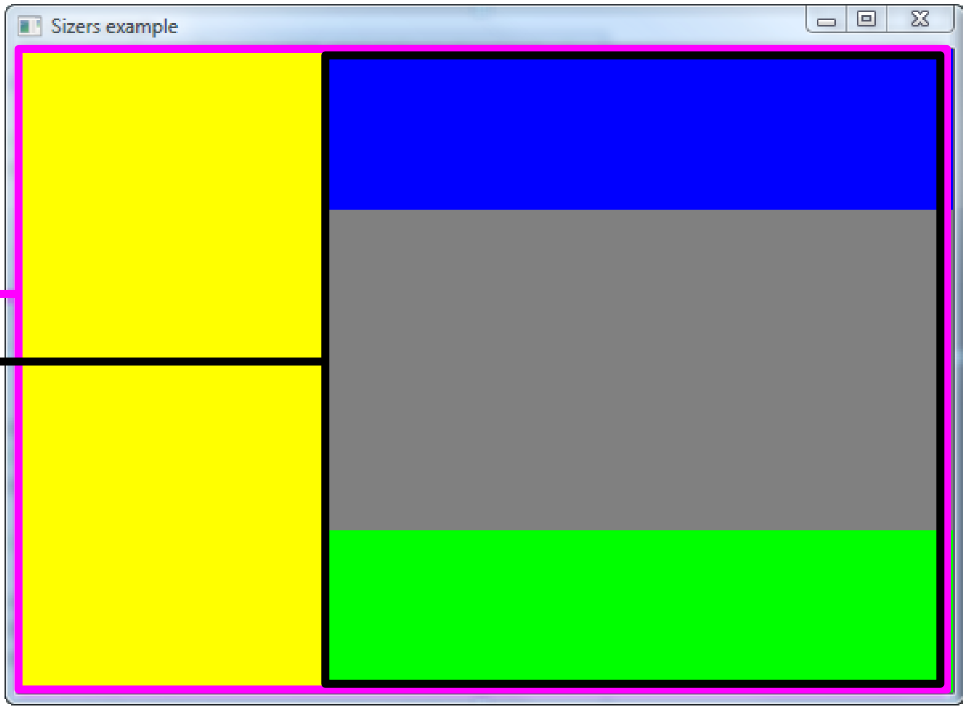
- Use sizers for sub-windows layout or complex/nested layouts
- Try AUI for the main application windows

Windows Layout Management – Sizers

- ✓ Similar to *LayoutManagers* in Java
- ✓ All items (widgets or nested sizers) added to a sizer are laid out by a specific algorithm
- ✓ Relationships defined by containment within sizers or nested sizers
- ✓ An item's position within its allotted space is also controllable:
  - Empty space on borders
  - Alignment
- ✓ You need to be able to think visually both top-down and bottom-up to capture your design

Windows Layout Management – Sizers

```
# Create a new app, don't redirect stdout/stderr to a window
app = wx.App(False)

# Create the main application window
frame = wx.Frame(parent=None, title='Sizers example')

# Make a bunch of coloured panels
yellow_panel = MakeColourWindow(frame, 'yellow')
blue_panel   = MakeColourWindow(frame, 'blue')
grey_panel   = MakeColourWindow(frame, 'grey')
green_panel  = MakeColourWindow(frame, 'green')

# Main sizer
h_sizer = wx.BoxSizer(wx.HORIZONTAL)
# Nested sub-sizer
v_sizer = wx.BoxSizer(wx.VERTICAL)

# Add the panels to the sizers
# Grey gets 1/2 of the space, blue and green 1/4 respectively
v_sizer.Add(blue_panel  , proportion=1, flag=wx.EXPAND)
v_sizer.Add(grey_panel  , proportion=2, flag=wx.EXPAND)
v_sizer.Add(green_panel , proportion=1, flag=wx.EXPAND)

# Nested sizer gets 2/3 of the space, yellow 1/3
h_sizer.Add(yellow_panel, proportion=1, flag=wx.EXPAND)
h_sizer.Add(v_sizer     , proportion=2, flag=wx.EXPAND)

# Set the sizer to the application main window
frame.SetSizer(h_sizer)
frame.Show()

app.MainLoop()
```



wxPython sample: ***sizers.py***

Windows Layout Management – AUI

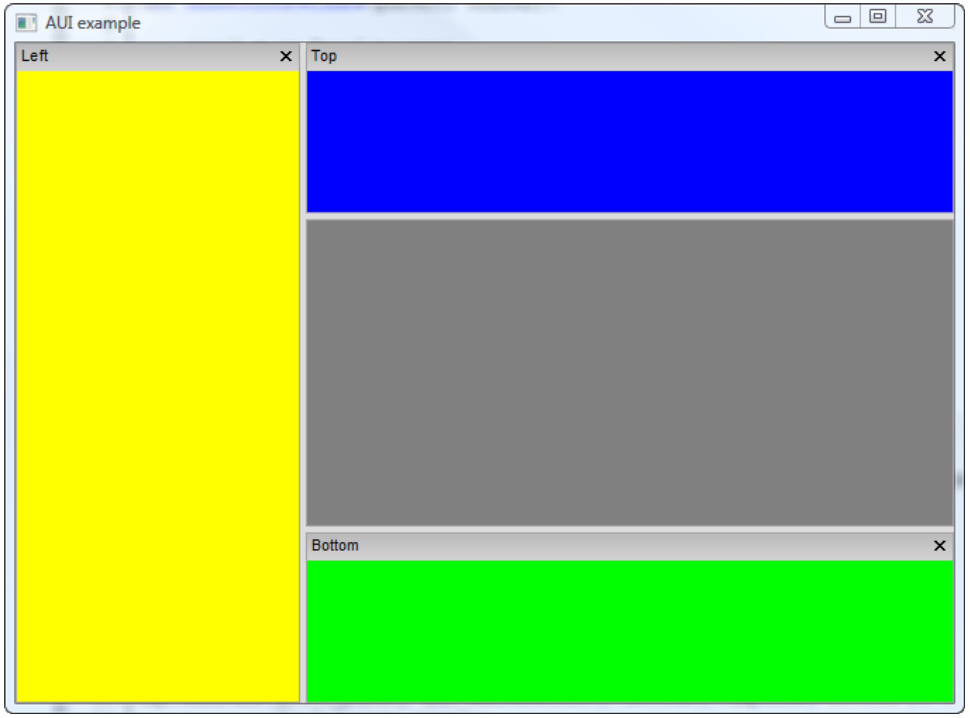
AUI is an advanced layout mechanism you can use to quickly build high-quality, cross platform user interfaces. AUI provides:

- ✓ Native, dockable floating frames
- ✓ Perspective saving and loading
- ✓ Native toolbars incorporating real-time, "spring-loaded" dragging
- ✓ Customizable floating/docking behaviour
- ✓ Completely customizable look-and-feel
- ✓ Optional transparent window effects (while dragging or docking)
- ✓ Splittable notebook control

Available as a wrapped sub-module (in *wx.aui*) or as pure-Python implementation (in *wx.lib.agw.aui*)

Windows Layout Management – AUI

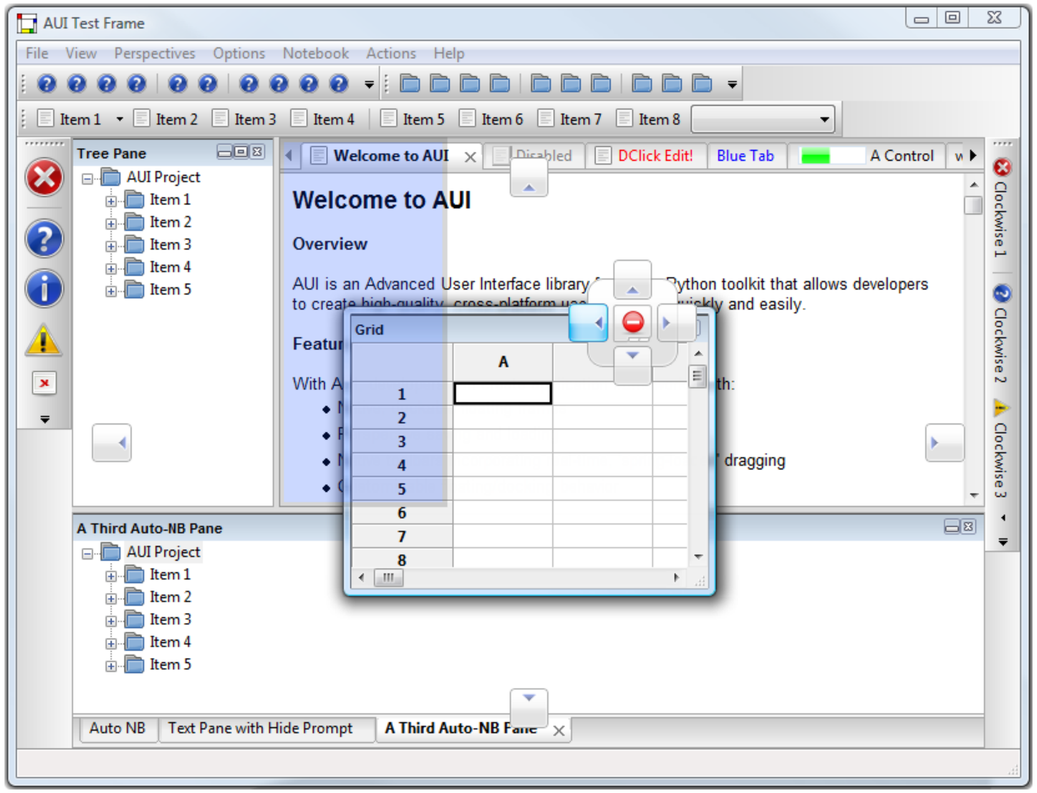
```
# Create a new app, don't redirect stdout/stderr to a window
app = wx.App(False)

# Create the main application window
frame = wx.Frame(parent=None, title='AUI example')

# Make a bunch of coloured panels
yellow_panel = MakeColourWindow(frame, 'yellow')
blue_panel   = MakeColourWindow(frame, 'blue')
grey_panel   = MakeColourWindow(frame, 'grey')
green_panel  = MakeColourWindow(frame, 'green')

# Create the window manager
mgr = aui.AuiManager(frame)

# Add the panels as in the Sizers example
mgr.AddPane(yellow_panel, aui.AuiPaneInfo().Left(). \
            Caption('Left').Layer(1))
mgr.AddPane(blue_panel  , aui.AuiPaneInfo().Top(). \
            Caption('Top'))
mgr.AddPane(grey_panel  , aui.AuiPaneInfo().CenterPane())
mgr.AddPane(green_panel , aui.AuiPaneInfo().Bottom(). \
            Caption('Bottom'))

# Commit the changes to the layout
mgr.Update()

frame.Show()
app.MainLoop()
```



wxPython sample: ***aui.py***

Windows Layout Management – AUI

✓ In addition to layout management, you get fancy docking/floating windows



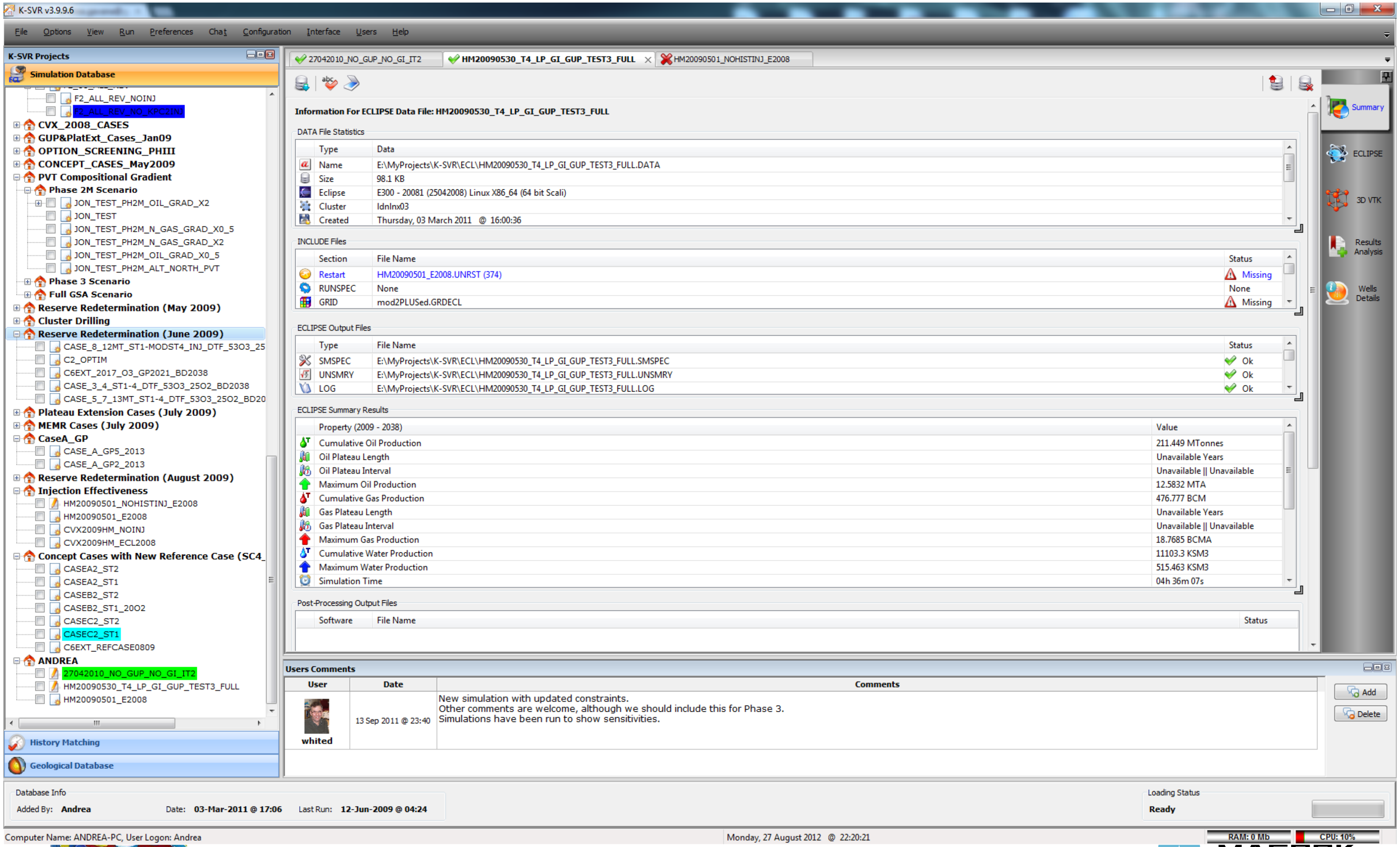
Welcome to AUI

Overview

AUI is an Advanced User Interface library ... Python toolkit that allows developers to create high-quality, cross-platform ... quickly and easily.

✓ Trust me when I say you can get quite impressive layouts…

Windows Layout Management – AUI



K-SVR v3.9.9.6

File Options View Run Preferences Chat Configuration Interface Users Help

K-SVR Projects

Simulation Database

- F2_ALL_REV_NOINJ
- F2_ALL_REV_NO_KPC2INJ
- CVX_2008_CASES
- GUP&PlatExt_Cases_Jan09
- OPTION_SCREENING_PHIII
- CONCEPT_CASES_May2009
- PVT Compositional Gradient
  - Phase 2M Scenario
    - JON_TEST_PH2M_OIL_GRAD_X2
    - JON_TEST
    - JON_TEST_PH2M_N_GAS_GRAD_X0_5
    - JON_TEST_PH2M_N_GAS_GRAD_X2
    - JON_TEST_PH2M_OIL_GRAD_X0_5
    - JON_TEST_PH2M_ALT_NORTH_PVT
  - Phase 3 Scenario
  - Full GSA Scenario
- Reserve Redetermination (May 2009)
- Cluster Drilling
- Reserve Redetermination (June 2009)
  - CASE_8_12MT_ST1-MODST4_INJ_DTF_5303_25
  - C2_OPTIM
  - C6EXT_2017_03_GP2021_BD2038
  - CASE_3_4_ST1-4_DTF_5303_2502_BD2038
  - CASE_5_7_13MT_ST1-4_DTF_5303_2502_BD20
- Plateau Extension Cases (July 2009)
- MEMR Cases (July 2009)
- CaseA_GP
  - CASE_A_GP5_2013
  - CASE_A_GP2_2013
- Reserve Redetermination (August 2009)
- Injection Effectiveness
  - HM20090501_NOHISTINJ_E2008
  - HM20090501_E2008
  - CVX2009HM_NOINJ
  - CVX2009HM_ECL2008
- Concept Cases with New Reference Case (SC4_
  - CASEA2_ST2
  - CASEA2_ST1
  - CASEB2_ST2
  - CASEB2_ST1_2002
  - CASEC2_ST2
  - CASEC2_ST1
  - C6EXT_REFCASE0809
- ANDREA
  - 27042010_NO_GUP_NO_GI_IT2
  - HM20090530_T4_LP_GI_GUP_TEST3_FULL
  - HM20090501_E2008

History Matching

Geological Database

Tabs: 27042010_NO_GUP_NO_GI_IT2 | HM20090530_T4_LP_GI_GUP_TEST3_FULL | HM20090501_NOHISTINJ_E2008

Information For ECLIPSE Data File: HM20090530_T4_LP_GI_GUP_TEST3_FULL

DATA File Statistics

Type	Data
Name	E:\MyProjects\K-SVR\ECL\HM20090530_T4_LP_GI_GUP_TEST3_FULL.DATA
Size	98.1 KB
Eclipse	E300 - 20081 (25042008) Linux X86_64 (64 bit Scali)
Cluster	ldnlnx03
Created	Thursday, 03 March 2011 @ 16:00:36

INCLUDE Files

Section	File Name	Status
Restart	HM20090501_E2008.UNRST (374)	Missing
RUNSPEC	None	None
GRID	mod2PLUSed.GRDECL	Missing

ECLIPSE Output Files

Type	File Name	Status
SMSPEC	E:\MyProjects\K-SVR\ECL\HM20090530_T4_LP_GI_GUP_TEST3_FULL.SMSPEC	Ok
UNSMRY	E:\MyProjects\K-SVR\ECL\HM20090530_T4_LP_GI_GUP_TEST3_FULL.UNSMRY	Ok
LOG	E:\MyProjects\K-SVR\ECL\HM20090530_T4_LP_GI_GUP_TEST3_FULL.LOG	Ok

ECLIPSE Summary Results

Property (2009 - 2038)	Value
Cumulative Oil Production	211.449 MTonnes
Oil Plateau Length	Unavailable Years
Oil Plateau Interval	Unavailable || Unavailable
Maximum Oil Production	12.5832 MTA
Cumulative Gas Production	476.777 BCM
Gas Plateau Length	Unavailable Years
Gas Plateau Interval	Unavailable || Unavailable
Maximum Gas Production	18.7685 BCMA
Cumulative Water Production	11103.3 KSM3
Maximum Water Production	515.463 KSM3
Simulation Time	04h 36m 07s

Post-Processing Output Files

Software	File Name	Status

Users Comments

User	Date	Comments
whited	13 Sep 2011 @ 23:40	New simulation with updated constraints. Other comments are welcome, although we should include this for Phase 3. Simulations have been run to show sensitivities.

Database Info

Added By: **Andrea** Date: **03-Mar-2011 @ 17:06** Last Run: **12-Jun-2009 @ 04:24**

Loading Status

Ready

Computer Name: ANDREA-PC, User Logon: Andrea | Monday, 27 August 2012 @ 22:20:21 | RAM: 0 Mb | CPU: 10%

Parallel wxPython

Playing nice with threads or parallel processes…

- ✓ wxPython widgets are not (easily) pickleable:
  - *multiprocessing* will complain
  - Child processes can not directly interact with the main process
- ✓ GUI-related methods/functions are not thread-safe:
  - Separate threads can not directly call GUI methods
  - The GIL is usually not your friend ☺

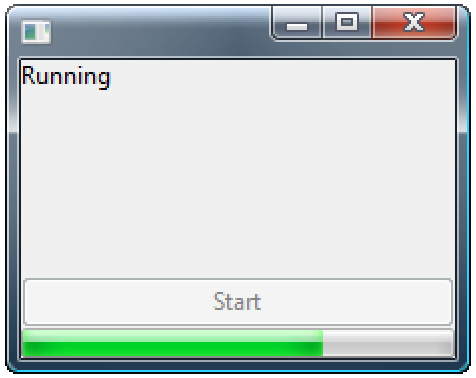
Different alternatives for handling threads:

- *wx.CallAfter*
- *wx.PostEvent*
- Using *pubsub*

Parallel wxPython – Threads

```
class MainFrame(wx.Frame):

    def __init__(self, parent):
        wx.Frame.__init__(self, parent)

        self.label = wx.StaticText(self, label="Ready")
        self.btn = wx.Button(self, label="Start")
        self.gauge = wx.Gauge(self)

        # ... Other initialization skipped ...
        self.Bind(wx.EVT_BUTTON, self.OnButton)

    def OnButton(self, event):
        self.gauge.SetValue(0)
        self.label.SetLabel("Running")

        thread = threading.Thread(target=self.LongRunning)
        thread.start()

    def OnLongRunDone(self):
        self.gauge.SetValue(100)
        self.label.SetLabel("Done")

    def LongRunning(self):
        """This runs in a different thread. Sleep is used
         to simulate a long running task."""
        time.sleep(3)
        wx.CallAfter(self.gauge.SetValue, 20)
        time.sleep(5)
        wx.CallAfter(self.gauge.SetValue, 70)
        time.sleep(4)
        wx.CallAfter(self.OnLongRunDone)
```

- ✓ GUI remains responsive
- ✓ Similar strategy can be implemented via *wx.PostEvent* or *pubsub*



wxPython samples:
threads_1.py
threads_2.py

Parallel wxPython – Processes

Multiple concurrent processes:

- Start a separate monitoring thread
- Start the processes from the thread
- Use *wx.CallAfter*, *wx.PostEvent* or *pubsub* in the thread to communicate with your GUI

```python
def LongRunning(self):
    """
    This runs in a different thread and starts multiple
    separate processes to simulate a long running task.
    """

    pool = multiprocessing.Pool(processes=4)
    tasks = range(0, 100)
    it = pool.imap(RunCalculations, tasks)

    while 1:
        try:
            value = it.next()
        except StopIteration:
            break
        else:
            wx.CallAfter(self.gauge.SetValue, value)

    wx.CallAfter(self.OnLongRunDone)
```

✓ GUI remains responsive

✓ You can use *multiprocessing.apply_async* as well

wxPython sample: ***process_1.py***

Parallel wxPython – Processes

Single separate process:

- Used to monitor an external applications (for example)
- Particularly useful to monitor *stdout* and *stderr*
- Use *wx.Process* and *wx.Execute* to run the separate process

```python
def LongRunning(self):
    """
    This runs in the GUI thread but uses wx.Process and
    wx.Execute to start and monitor a separate process.
    """

    cmd = 'python -u external_program.py'

    self.process = wx.Process(self)
    self.process.Redirect()

    wx.Execute(cmd, wx.EXEC_ASYNC, self.process)


def OnIdle(self, event):
    """ This event handler catches the process stdout. """

    if self.process is not None:
        stream = self.process.GetInputStream()
        if stream.CanRead():
            text = stream.read()
            self.label.AppendText(text)
```

✓ GUI remains responsive

✓ *cmd* can be any process you can start on your machine

wxPython sample: ***process_2.py***

Persisting GUIs State

Persistent GUIs automatically save their state when they are destroyed and restore it when they are recreated, even during another program invocation.

wx.lib.agw.persist is a package that does all the work for you:

- ✓ *PersistenceManager* which all persistent widgets register themselves with
- ✓ *PersistentObject* is the base class for all persistent controls
- ✓ *PersistentHandlers* which handle different kind of saving/restoring actions depending on the widget type

Persistent states include:

- ✓ Windows position, size and (AUI) layouts
- ✓ Text control values
- ✓ Radiobutton selections
- ✓ Tree controls expansion state
- ✓ List controls selections, column widths etc…

The *persist* framework handles more than 100 different widgets

Persisting GUIs State

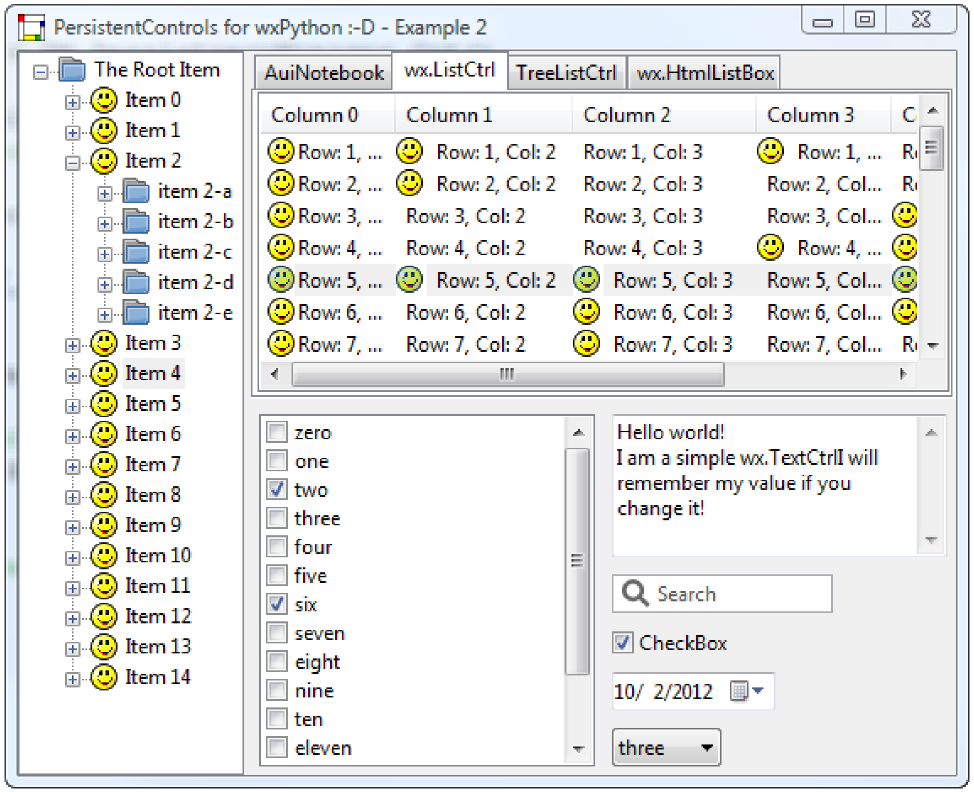
```
import wx
import wx.lib.agw.persist as PM

class MyFrame(wx.Frame):

    def __init__(self, parent):

        wx.Frame.__init__(self, parent)

        book = wx.Notebook(self, wx.ID_ANY)

        # Very important step!!
        book.SetName("MyBook") # Do not use the default name!!

        book.AddPage(wx.Panel(book), "Hello")
        book.AddPage(wx.Panel(book), "World")

        self.persistMgr = PM.PersistenceManager.Get()

        if not self.persistMgr.RegisterAndRestore(book):
            # Nothing was restored, so choose the default page
            # ourselves
            book.SetSelection(0)

        self.Bind(wx.EVT_CLOSE, self.OnClose)


    def OnClose(self, event):

        # Save the GUI state and unregister
        self.persistMgr.SaveAndUnregister()
        event.Skip()
```

Set unique window name

Register the window and restore its previous state (if any)

Save window state and unregister

Persisting GUIs State


PersistentControls for wxPython :-D - Example 2

- The Root Item
  - Item 0
  - Item 1
  - Item 2
    - item 2-a
    - item 2-b
    - item 2-c
    - item 2-d
    - item 2-e
  - Item 3
  - Item 4
  - Item 5
  - Item 6
  - Item 7
  - Item 8
  - Item 9
  - Item 10
  - Item 11
  - Item 12
  - Item 13
  - Item 14

AuiNotebook | wx.ListCtrl | TreeListCtrl | wx.HtmlListBox

Column 0	Column 1	Column 2	Column 3	C
Row: 1, ...	Row: 1, Col: 2	Row: 1, Col: 3	Row: 1, ...	R
Row: 2, ...	Row: 2, Col: 2	Row: 2, Col: 3	Row: 2, Col...	R
Row: 3, ...	Row: 3, Col: 2	Row: 3, Col: 3	Row: 3, Col...	
Row: 4, ...	Row: 4, Col: 2	Row: 4, Col: 3	Row: 4, ...	
Row: 5, ...	Row: 5, Col: 2	Row: 5, Col: 3	Row: 5, Col...	
Row: 6, ...	Row: 6, Col: 2	Row: 6, Col: 3	Row: 6, Col...	
Row: 7, ...	Row: 7, Col: 2	Row: 7, Col: 3	Row: 7, Col...	R

- [ ] zero
- [ ] one
- [x] two
- [ ] three
- [ ] four
- [ ] five
- [x] six
- [ ] seven
- [ ] eight
- [ ] nine
- [ ] ten
- [ ] eleven

Hello world!
I am a simple wx.TextCtrl I will remember my value if you change it!

Search

[x] CheckBox

10/ 2/2012

three

- ✓ You can set your own config file where states are saved
- ✓ States can be saved in *cPickle*, *ConfigObj*, *wx.Config* (and many other) formats

wxPython samples:
persist_1.py
persist_2.py

- ✓ *PersistentControls* supports all the native widgets and almost all the owner-drawn ones
- ✓ Notable exception is *wx.grid.Grid*

Owner-Drawn Controls

Custom control does not mean owner-drawn:

- ✓ A custom widget may extend the functionalities of the native one without the need of being owner-drawn
- ✓ Owner-drawn widgets are much more flexible (look and feel, behavior)
- ✓ The cost is the loss of “nativeness” and accessibility issues

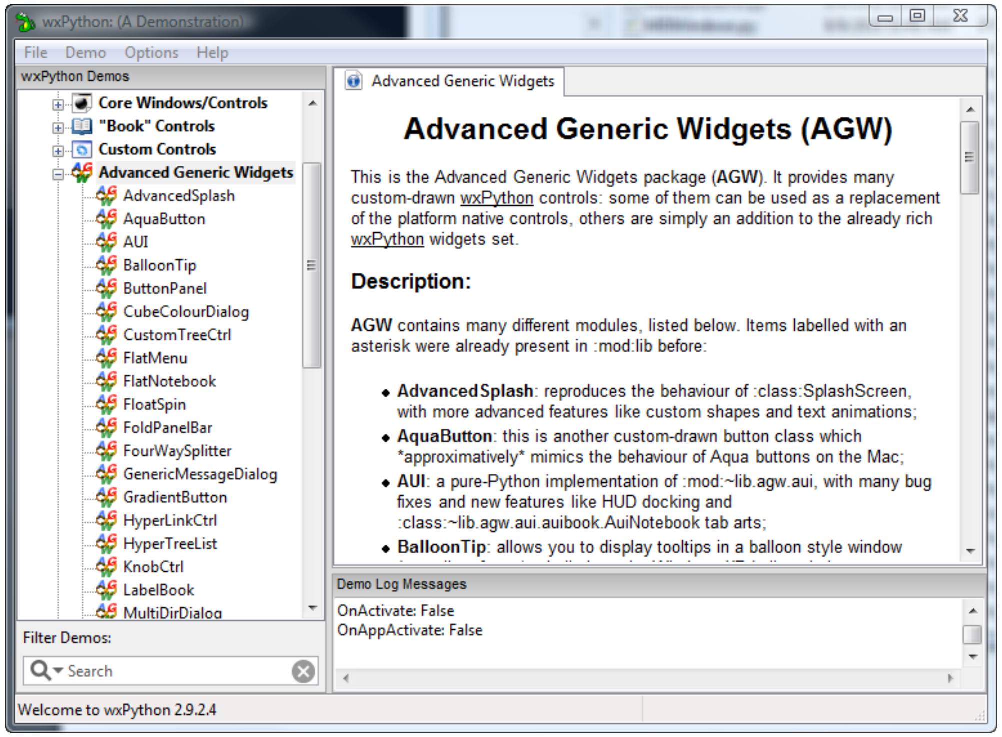
If you are looking for a specific widget…

- ✓ Do not reinvent the wheel:
  - Check the wxPython demo
  - Look inside *wx.lib* (160 custom widgets available)
- ✓ When everything else fails:
  - Check if wxWidgets contains a generic implementation of your control
  - Write your own

Owner-Drawn Controls – AGW

Advanced Generic Widgets

- ✓ A package officially distributed with wxPython (in *wx.lib.agw*)
- ✓ Contains 37 owner-drawn widgets and many useful auxiliary classes
- ✓ 150 kLOC, fully documented in Sphinx-friendly style
- ✓ Extensive demos showing all the widgets' functionalities



Advanced Generic Widgets (AGW)

This is the Advanced Generic Widgets package (**AGW**). It provides many custom-drawn wxPython controls: some of them can be used as a replacement of the platform native controls, others are simply an addition to the already rich wxPython widgets set.

Description:

AGW contains many different modules, listed below. Items labelled with an asterisk were already present in :mod:lib before:

- **AdvancedSplash**: reproduces the behaviour of :class:SplashScreen, with more advanced features like custom shapes and text animations;
- **AquaButton**: this is another custom-drawn button class which *approximatively* mimics the behaviour of Aqua buttons on the Mac;
- **AUI**: a pure-Python implementation of :mod:~lib.agw.aui, with many bug fixes and new features like HUD docking and :class:~lib.agw.aui.auibook.AuiNotebook tab arts;
- **BalloonTip**: allows you to display tooltips in a balloon style window

```
OnActivate: False
OnAppActivate: False
```

Welcome to wxPython 2.9.2.4

- Most of the widgets derived from C++ wxWidgets generic implementations
- Everything is pure-Python – no wrappers
- Code maintenance is straightforward
- Every wxPython user can easily write a patch for any AGW widget

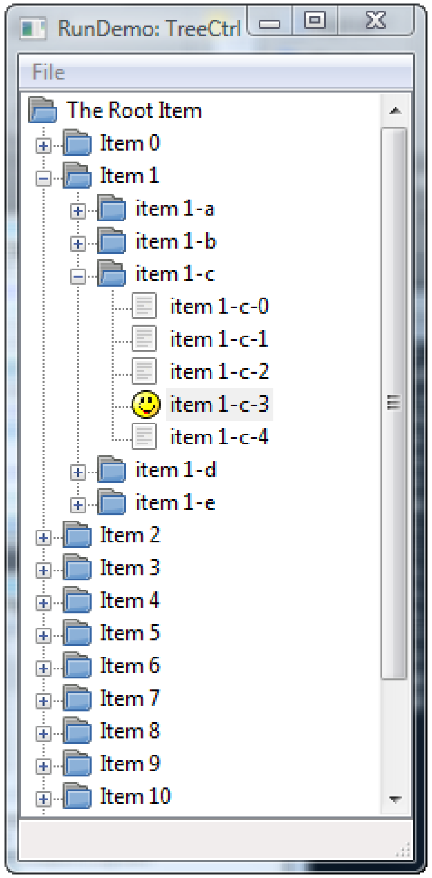
MAERSK OIL

Owner-Drawn Controls – AGW

Derived from wxWidgets

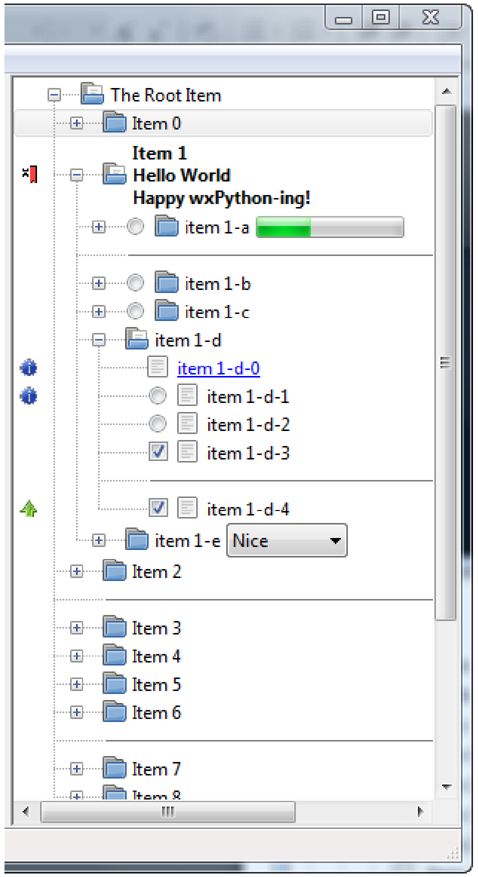
CustomTreeCtrl

wx.TreeCtrl



- Checkbox and radiobutton type tree items
- Hyperlink type tree items
- Multiline text items
- Separator-style items
- Enabling/disabling tree items
- Any widget can be attached next to an item
- Custom item selection styles (gradients)
- Multiple images for tree items
- Ellipsization and tooltips on long items

CustomTreeCtrl



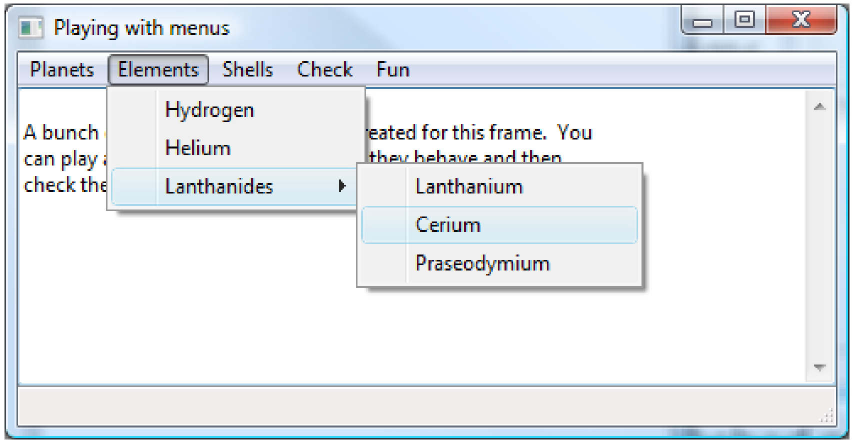
wxPython sample: ***customtreectrl.py***

Owner-Drawn Controls – AGW

Derived from wxWidgets

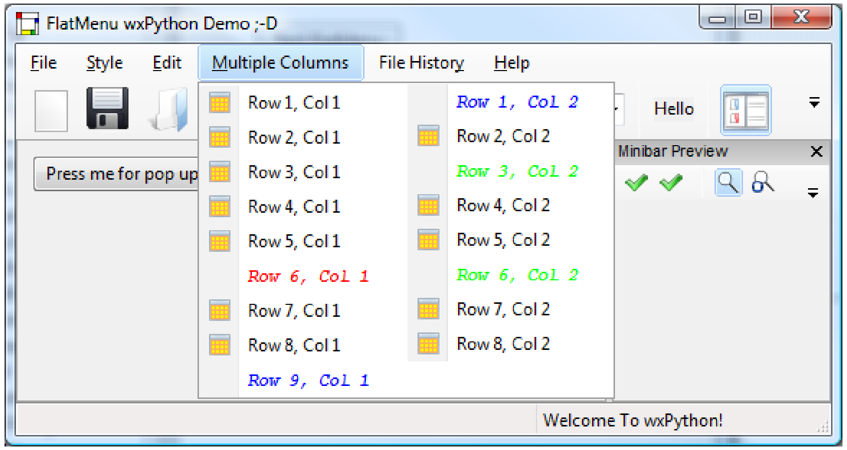
FlatMenu

wx.Menu



A bunch … can play … check the … eated for this frame. You … they behave and then

FlatMenu



Custom color schemes	Multiple columns menus
Context menus for menu items	Menu transparency
File history support	Menu background image
Integrated toolbar & mini-bar	Drop-down customization arrow

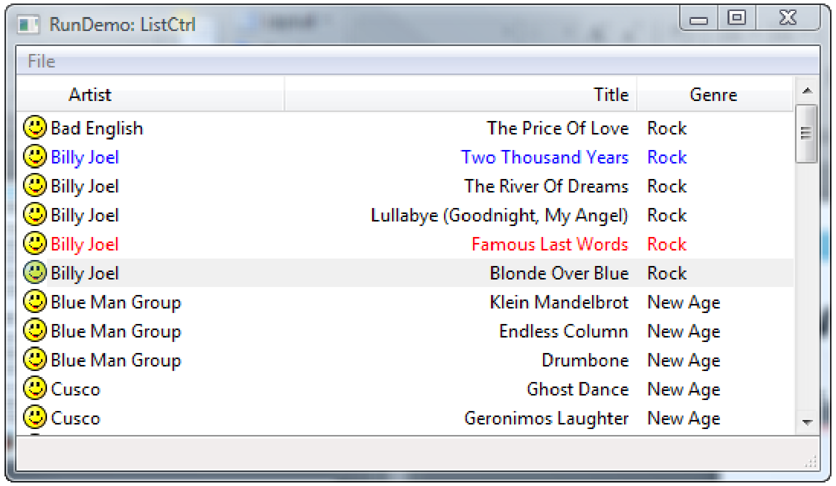
wxPython sample: ***flatmenu.py***

Owner-Drawn Controls – AGW

Derived from wxWidgets

UltimateListCtrl

wx.ListCtrl

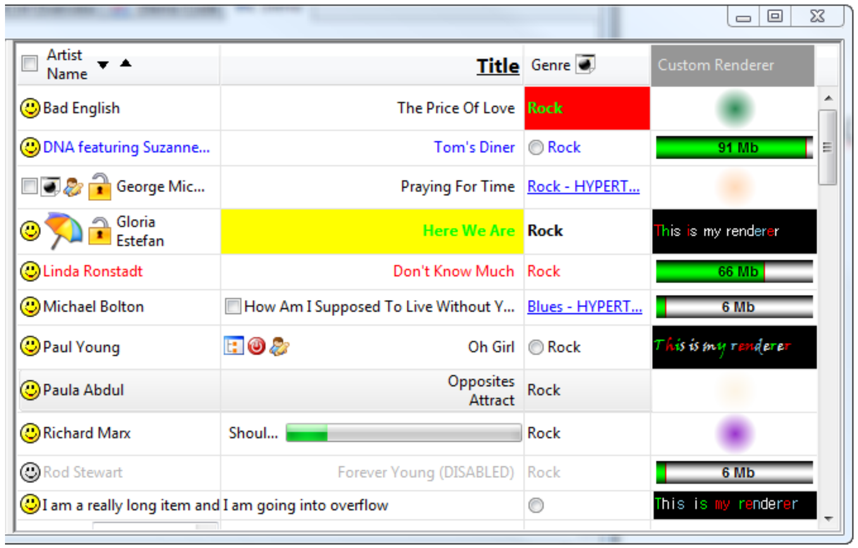


RunDemo: ListCtrl

File

Artist	Title	Genre
Bad English	The Price Of Love	Rock
Billy Joel	Two Thousand Years	Rock
Billy Joel	The River Of Dreams	Rock
Billy Joel	Lullabye (Goodnight, My Angel)	Rock
Billy Joel	Famous Last Words	Rock
Billy Joel	Blonde Over Blue	Rock
Blue Man Group	Klein Mandelbrot	New Age
Blue Man Group	Endless Column	New Age
Blue Man Group	Drumbone	New Age
Cusco	Ghost Dance	New Age
Cusco	Geronimos Laughter	New Age

UltimateListCtrl



Artist Name	Title	Genre	Custom Renderer
Bad English	The Price Of Love	Rock	
DNA featuring Suzanne...	Tom's Diner	Rock	91 Mb
George Mic...	Praying For Time	Rock - HYPERT...	
Gloria Estefan	Here We Are	Rock	This is my renderer
Linda Ronstadt	Don't Know Much	Rock	66 Mb
Michael Bolton	How Am I Supposed To Live Without Y...	Blues - HYPERT...	6 Mb
Paul Young	Oh Girl	Rock	This is my renderer
Paula Abdul	Opposites Attract	Rock	
Richard Marx	Shoul...	Rock	
Rod Stewart	Forever Young (DISABLED)	Rock	6 Mb
I am a really long item and I am going into overflow			This is my renderer

Multiple images for items	Flexible item formatting
Checkbox and radiobutton items	Overflowing items
Multiline text items and hyperlinks	Custom renderers
Enabling/disabling items	Variable row heights
Any widget can be attached to an item	Hide/show columns

wxPython sample: ***ultimatelistctrl.py***

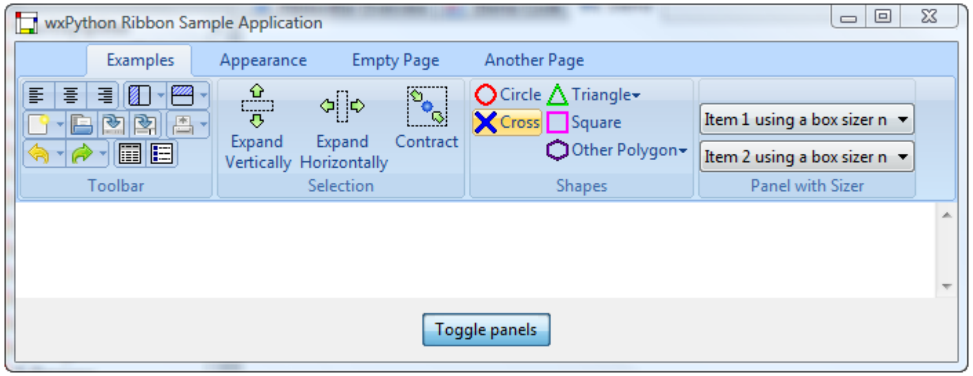
PYCON AR 2012

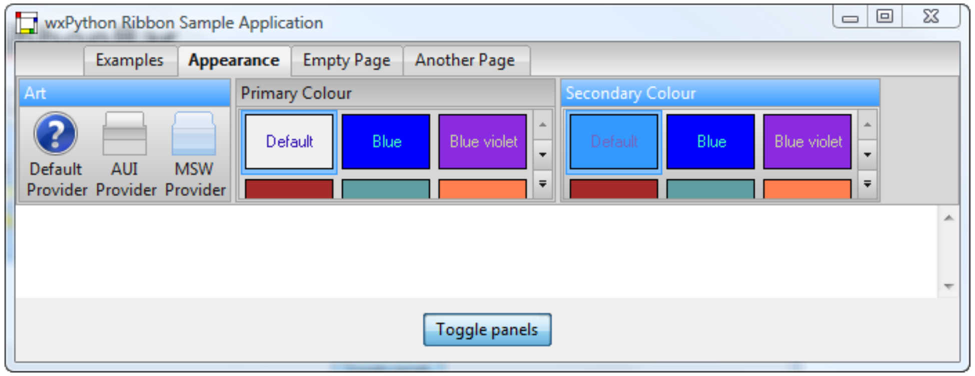
MAERSK OIL

Owner-Drawn Controls – AGW

Derived from wxWidgets

RibbonBar





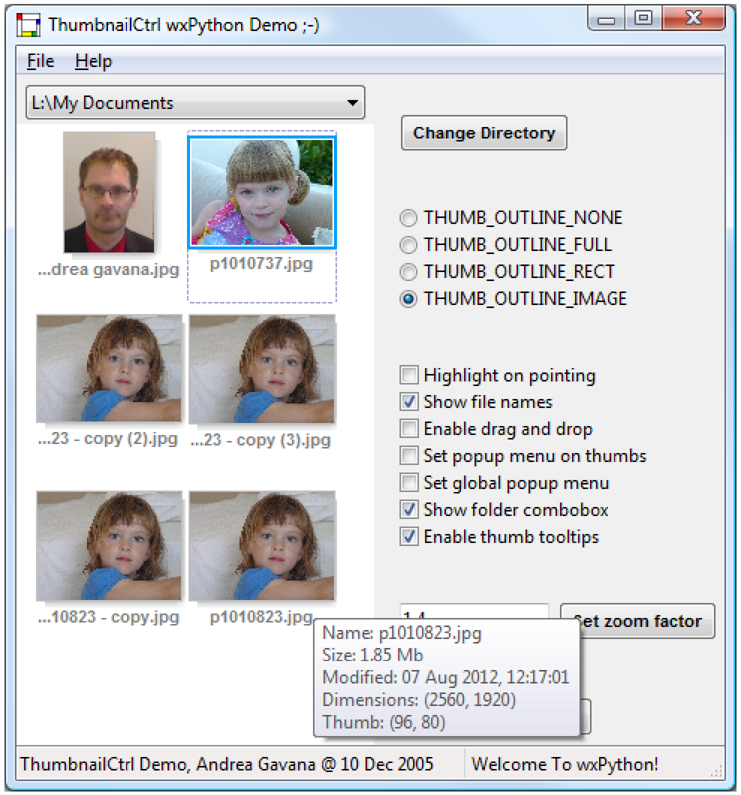
- Similar to MS Office Ribbon
- Ribbon items expand/collapse depending on the window size
- Custom color schemes
- Toolbars, tabbed panels and galleries
- More than 100 color settings
- Buttons with toggle behavior and popup menus

wxPython sample: ***ribbonbar.py***

Owner-Drawn Controls – AGW

“Create your own…”

ThumbnailCtrl



( ) THUMB_OUTLINE_NONE
( ) THUMB_OUTLINE_FULL
( ) THUMB_OUTLINE_RECT
(●) THUMB_OUTLINE_IMAGE

- [ ] Highlight on pointing
- [x] Show file names
- [ ] Enable drag and drop
- [ ] Set popup menu on thumbs
- [ ] Set global popup menu
- [x] Show folder combobox
- [x] Enable thumb tooltips

```
Name: p1010823.jpg
Size: 1.85 Mb
Modified: 07 Aug 2012, 12:17:01
Dimensions: (2560, 1920)
Thumb: (96, 80)
```

ThumbnailCtrl Demo, Andrea Gavana @ 10 Dec 2005 | Welcome To wxPython!

- Creates multiple image thumbnails from a folder
- Works with PIL or with the standard wxPython image processing classes (customizable)
- Drag and drop of thumbnails to other applications
- Highlight thumbnails on mouse over
- Thumbnail rotation, zoom and font/color settings
- Lightning-fast as it uses multiple independent threads to generate the thumbnails
- Multi-processing support will be added in the near future (parallel thumbnail generation)

wxPython sample: ***thumbnailctrl.py***

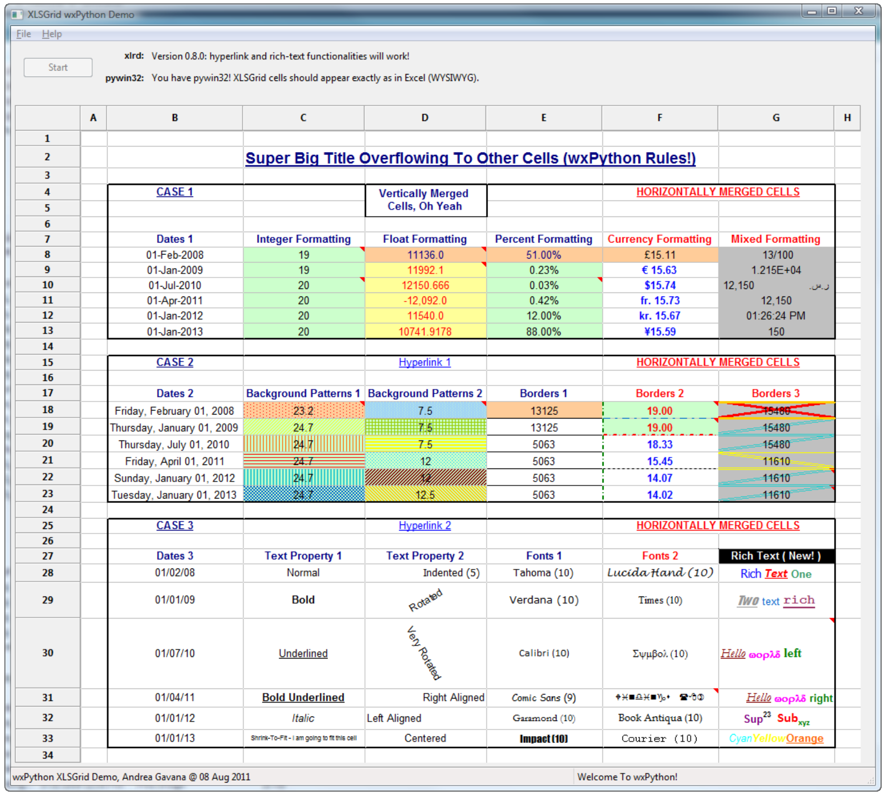
PYCON AR 2012

MAERSK OIL

Owner-Drawn Controls – AGW

“Create your own…”

XLSGrid



XLSGrid wxPython Demo

xlrd: Version 0.8.0: hyperlink and rich-text functionalities will work!

pywin32: You have pywin32! XLSGrid cells should appear exactly as in Excel (WYSIWYG).

	A	B	C	D	E	F	G	H
1								
2			Super Big Title Overflowing To Other Cells (wxPython Rules!)					
3								
4		CASE 1		Vertically Merged Cells, Oh Yeah		HORIZONTALLY MERGED CELLS		
5								
6								
7		Dates 1	Integer Formatting	Float Formatting	Percent Formatting	Currency Formatting	Mixed Formatting	
8		01-Feb-2008	19	11136.0	51.00%	£15.11	13/100	
9		01-Jan-2009	19	11992.1	0.23%	€ 15.63	1.215E+04	
10		01-Jul-2010	20	12150.666	0.03%	$15.74	12,150 ر.س.	
11		01-Apr-2011	20	-12,092.0	0.42%	fr. 15.73	12,150	
12		01-Jan-2012	20	11540.0	12.00%	kr. 15.67	01:26:24 PM	
13		01-Jan-2013	20	10741.9178	88.00%	¥15.59	150	
14								
15		CASE 2		Hyperlink 1		HORIZONTALLY MERGED CELLS		
16								
17		Dates 2	Background Patterns 1	Background Patterns 2	Borders 1	Borders 2	Borders 3	
18		Friday, February 01, 2008	23.2	7.5	13125	19.00	15480	
19		Thursday, January 01, 2009	24.7	7.5	13125	19.00	15480	
20		Thursday, July 01, 2010	24.7	7.5	5063	18.33	15480	
21		Friday, April 01, 2011	24.7	12	5063	15.45	11610	
22		Sunday, January 01, 2012	24.7	12	5063	14.07	11610	
23		Tuesday, January 01, 2013	24.7	12.5	5063	14.02	11610	
24								
25		CASE 3		Hyperlink 2		HORIZONTALLY MERGED CELLS		
26								
27		Dates 3	Text Property 1	Text Property 2	Fonts 1	Fonts 2	Rich Text (New!)	
28		01/02/08	Normal	Indented (5)	Tahoma (10)	*Lucida Hand (10)*	Rich ***Text*** One	
29		01/01/09	**Bold**	Rotated	Verdana (10)	Times (10)	***Two*** text rich	
30		01/07/10	Underlined	Very Rotated	Calibri (10)	Συμβολ (10)	*Hello* ωορλδ **left**	
31		01/04/11	**Bold Underlined**	Right Aligned	*Comic Sans (9)*	♦⌘■♎⌘■♑• ☎✇✇	*Hello* ωορλδ **right**	
32		01/01/12	*Italic*	Left Aligned	Garamond (10)	Book Antiqua (10)	Sup^{23} Sub$_{xyz}$	
33		01/01/13	Shrink-To-Fit - I am going to fit this cell	Centered	**Impact (10)**	Courier (10)	CyanYellowOrange	
34								

wxPython XLSGrid Demo, Andrea Gavana @ 08 Aug 2011 | Welcome To wxPython!

- Any cell background and fill pattern
- All border types and colors exposed by Excel
- Any cell font, text color and rotation
- Alignment (LTR and RTL), shrink-to-fit and wrapping
- Rich text and hyperlinks support
- Comments on cells
- Merging of cells and overflowing
- Column and row sizes respected
- Uses *xlrd* and, if available, Mark Hammond’s *pywin32* packages

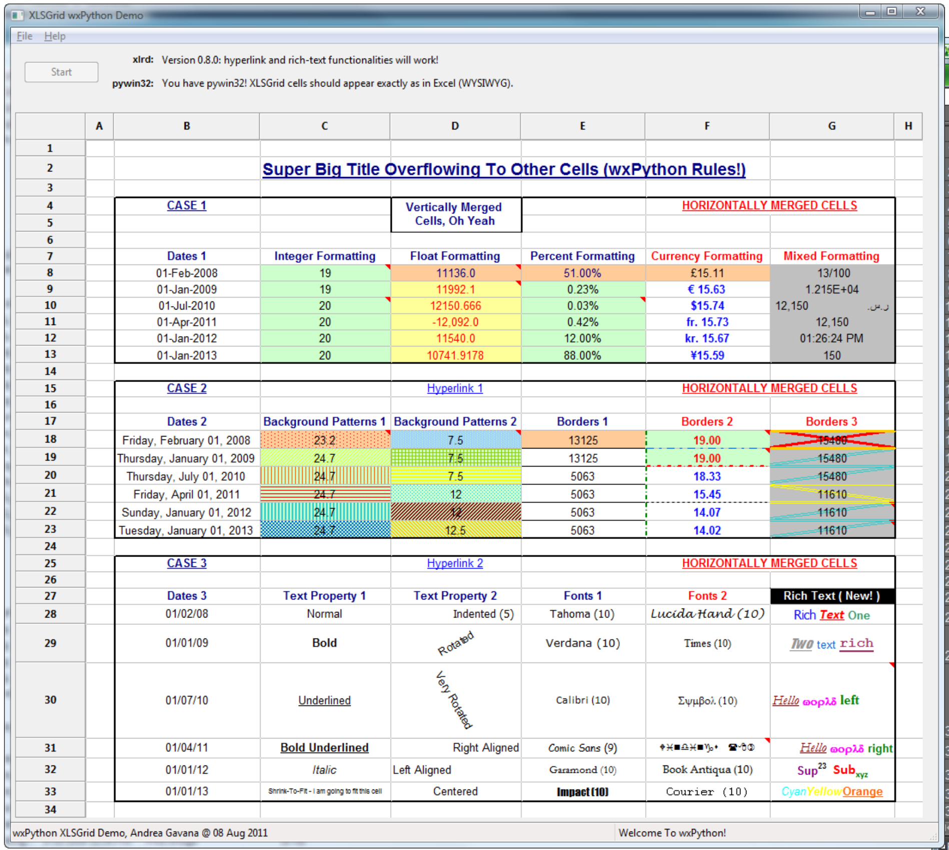
wxPython sample: ***xlsgrid.py***

Owner-Drawn Controls – AGW

"Create your own…"

XLSGrid

MS Excel



XLSGrid wxPython Demo

File Help

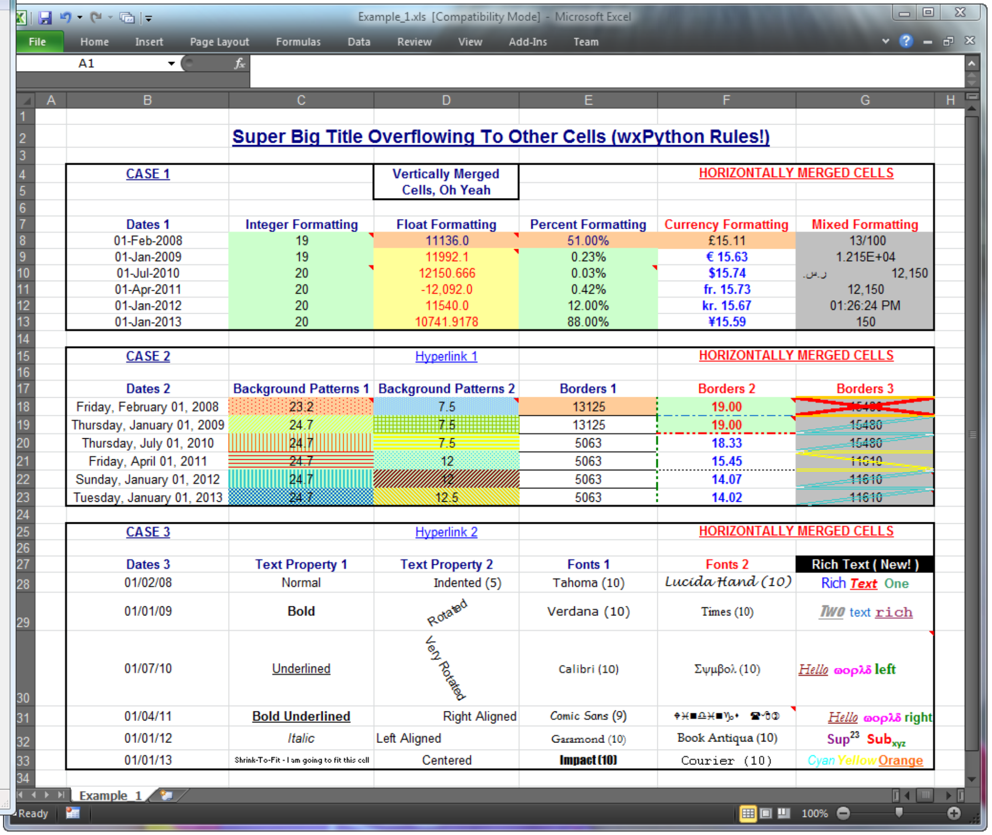
Start

xlrd: Version 0.8.0: hyperlink and rich-text functionalities will work!

pywin32: You have pywin32! XLSGrid cells should appear exactly as in Excel (WYSIWYG).

	A	B	C	D	E	F	G	H
1								
2		Super Big Title Overflowing To Other Cells (wxPython Rules!)						
3								
4		CASE 1		Vertically Merged Cells, Oh Yeah		HORIZONTALLY MERGED CELLS		
5								
6								
7		Dates 1	Integer Formatting	Float Formatting	Percent Formatting	Currency Formatting	Mixed Formatting	
8		01-Feb-2008	19	11136.0	51.00%	£15.11	13/100	
9		01-Jan-2009	19	11992.1	0.23%	€ 15.63	1.215E+04	
10		01-Jul-2010	20	12150.666	0.03%	$15.74	12,150 ر.س	
11		01-Apr-2011	20	-12,092.0	0.42%	fr. 15.73	12,150	
12		01-Jan-2012	20	11540.0	12.00%	kr. 15.67	01:26:24 PM	
13		01-Jan-2013	20	10741.9178	88.00%	¥15.59	150	
14								
15		CASE 2		Hyperlink 1		HORIZONTALLY MERGED CELLS		
16								
17		Dates 2	Background Patterns 1	Background Patterns 2	Borders 1	Borders 2	Borders 3	
18		Friday, February 01, 2008	23.2	7.5	13125	19.00	15480	
19		Thursday, January 01, 2009	24.7	7.5	13125	19.00	15480	
20		Thursday, July 01, 2010	24.7	7.5	5063	18.33	15480	
21		Friday, April 01, 2011	24.7	12	5063	15.45	11610	
22		Sunday, January 01, 2012	24.7	12	5063	14.07	11610	
23		Tuesday, January 01, 2013	24.7	12.5	5063	14.02	11610	
24								
25		CASE 3		Hyperlink 2		HORIZONTALLY MERGED CELLS		
26								
27		Dates 3	Text Property 1	Text Property 2	Fonts 1	Fonts 2	Rich Text (New!)	
28		01/02/08	Normal	Indented (5)	Tahoma (10)	Lucida Hand (10)	Rich ***Text*** One	
29		01/01/09	**Bold**	Rotated	Verdana (10)	Times (10)	***Two*** text rich	
30		01/07/10	Underlined	Very Rotated	Calibri (10)	Σψμβολ (10)	*Hello* ωορλδ **left**	
31		01/04/11	**Bold Underlined**	Right Aligned	Comic Sans (9)	[illegible]	*Hello* ωορλδ **right**	
32		01/01/12	*Italic*	Left Aligned	Garamond (10)	Book Antiqua (10)	Sup^{23} **Sub**$_{xyz}$	
33		01/01/13	Shrink-To-Fit - I am going to fit this cell	Centered	**Impact (10)**	Courier (10)	*CyanYellow*Orange	
34								

wxPython XLSGrid Demo, Andrea Gavana @ 08 Aug 2011 | Welcome To wxPython!



Example_1.xls [Compatibility Mode] - Microsoft Excel

File Home Insert Page Layout Formulas Data Review View Add-Ins Team

A1

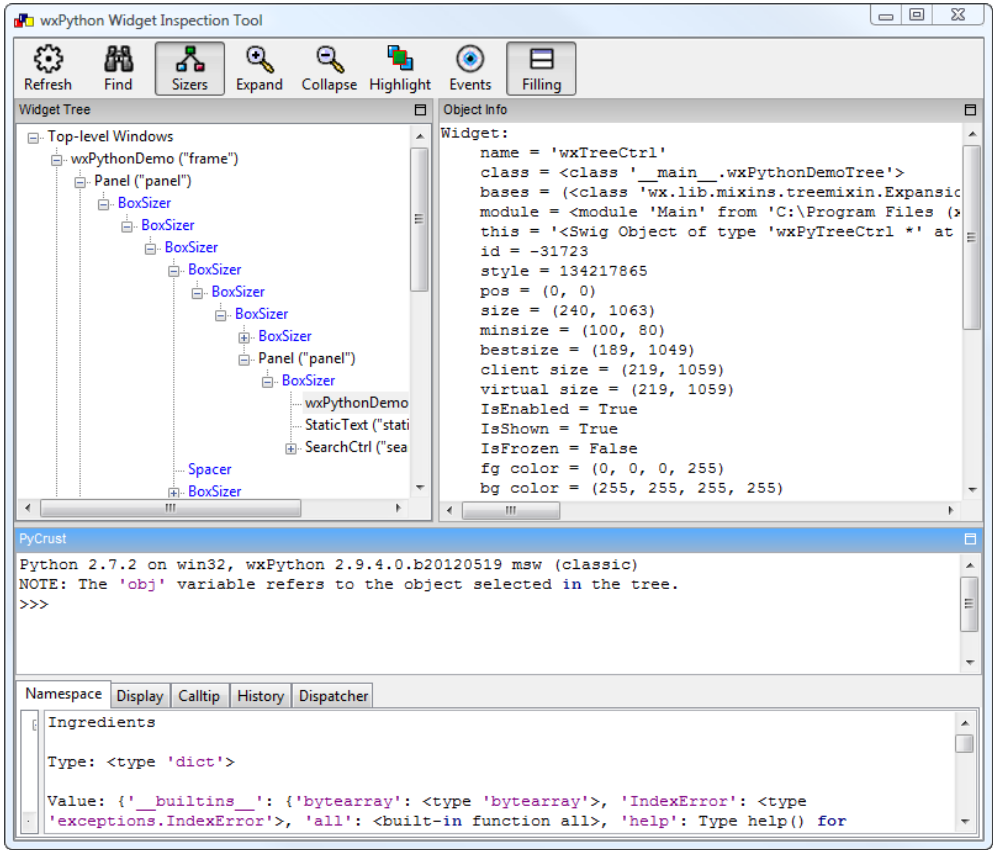
Example_1

Ready 100%

Lessons Learned

Generic (personal) advices:

✓ Use the Widget Inspection Tool (WIT) to debug a GUI layout



```
Top-level Windows
  wxPythonDemo ("frame")
    Panel ("panel")
      BoxSizer
        BoxSizer
          BoxSizer
            BoxSizer
              BoxSizer
                BoxSizer
                  BoxSizer
                  Panel ("panel")
                    BoxSizer
                      wxPythonDemo
                      StaticText ("stati
                      SearchCtrl ("sea
              Spacer
              BoxSizer
```

```
Widget:
    name = 'wxTreeCtrl'
    class = <class '__main__.wxPythonDemoTree'>
    bases = (<class 'wx.lib.mixins.treemixin.Expansic
    module = <module 'Main' from 'C:\Program Files (x
    this = '<Swig Object of type 'wxPyTreeCtrl *' at
    id = -31723
    style = 134217865
    pos = (0, 0)
    size = (240, 1063)
    minsize = (100, 80)
    bestsize = (189, 1049)
    client size = (219, 1059)
    virtual size = (219, 1059)
    IsEnabled = True
    IsShown = True
    IsFrozen = False
    fg color = (0, 0, 0, 255)
    bg color = (255, 255, 255, 255)
```

```
Python 2.7.2 on win32, wxPython 2.9.4.0.b20120519 msw (classic)
NOTE: The 'obj' variable refers to the object selected in the tree.
>>>
```

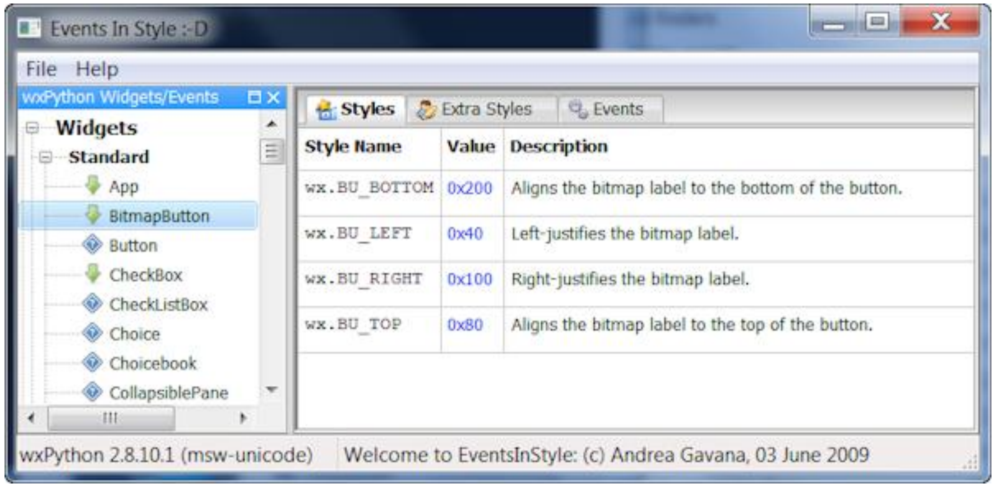
```
Ingredients

Type: <type 'dict'>

Value: {'__builtins__': {'bytearray': <type 'bytearray'>, 'IndexError': <type
'exceptions.IndexError'>, 'all': <built-in function all>, 'help': Type help() for
```

- Displays widgets/sizers hierarchy
- Shows controls attributes (size, position, colors, etc…)
- Sizers/widgets can be highlighted
- Watch events stream
- Can be used to easily spot a wrong parent/child relationship
- Powerful resource to inspect any widget internal structure

Lessons Learned

✓ Don't try and guess event names and window styles

- Peruse the documentation and the wxPython demo
- Use the magical *EventsInStyle*



Style Name	Value	Description
wx.BU_BOTTOM	0x200	Aligns the bitmap label to the bottom of the button.
wx.BU_LEFT	0x40	Left-justifies the bitmap label.
wx.BU_RIGHT	0x100	Right-justifies the bitmap label.
wx.BU_TOP	0x80	Aligns the bitmap label to the top of the button.

wxPython 2.8.10.1 (msw-unicode)   Welcome to EventsInStyle: (c) Andrea Gavana, 03 June 2009

- Displays window styles and extra styles for all widgets
- Shows appropriate events depending on the widget
- Always uses the latest docs available (from the web)

✓ Bind an event to the widget that generates the event

- i.e., use `self.button.Bind()` instead of `self.Bind()`
- http://wiki.wxpython.org/self.Bind%20vs.%20self.button.Bind

Lessons Learned

✓ It’s insanely easy to port a wxWidgets C++ generic widget to wxPython

- If a C++ version exists, convert it to Python instead of reinventing the wheel
- http://wiki.wxpython.org/Porting%20Widgets%20From%20C%2B%2B

✓ When writing owner-drawn controls

- Use automatic double-buffering: all platforms support it, via *wx.AutoBufferedPaintDC*
- Always try to guess (or calculate) a reasonable default size for your widget
- http://wiki.wxpython.org/CreatingCustomControls

✓ When reporting a problem/issue/bug on the wxPython mailing list

- Mention platform, Python and wxPython versions
- Include a small, runnable sample app that demonstrate the problem
- Be sure you have run the Widget Inspection Tool (WIT)

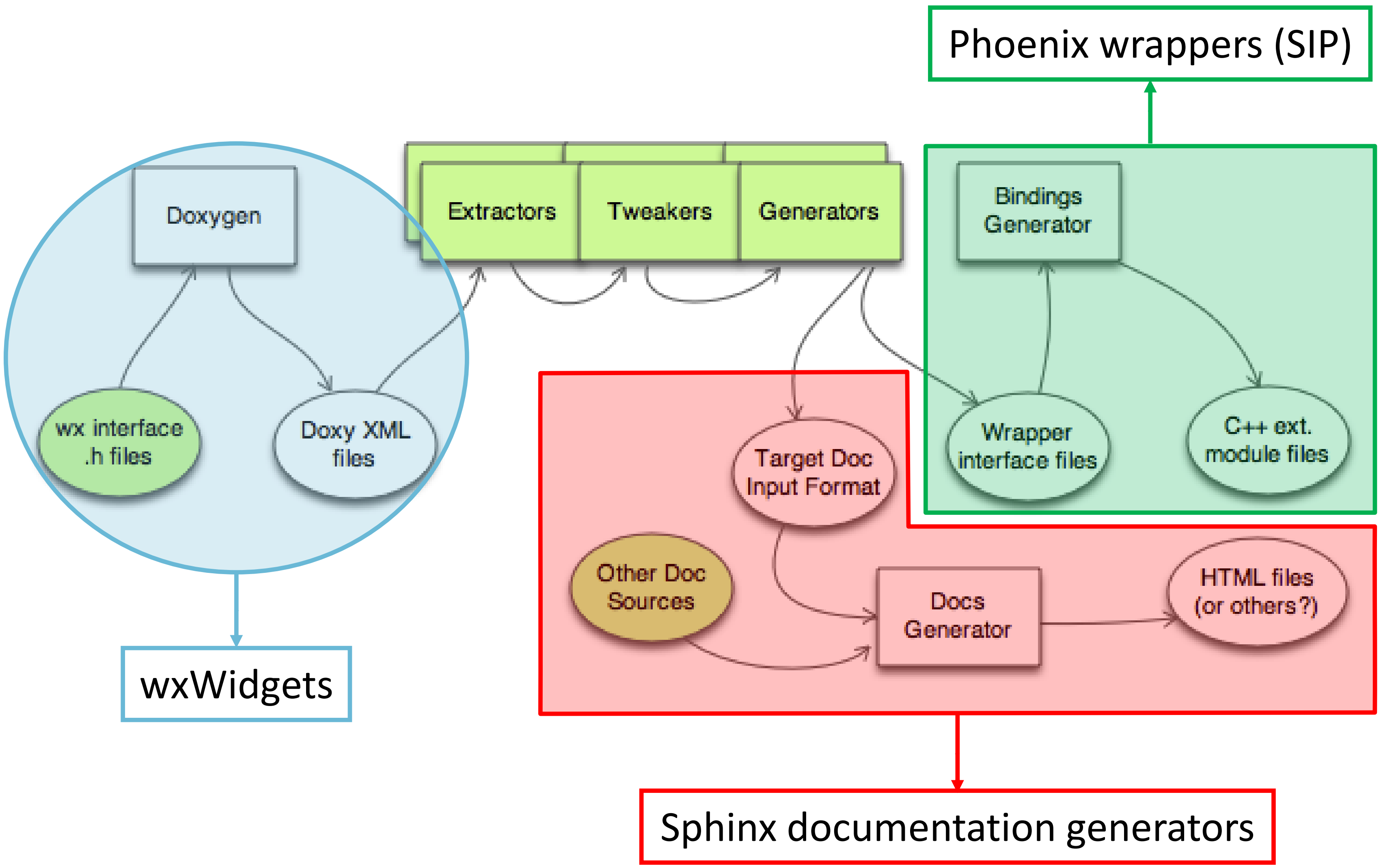
wxPython and Python 3

General considerations:

- ✓ Serious efforts to make wxPython compatible with Python 3 started in 2012
- ✓ Community was until recently disinterested in Python 3 support
- ✓ Major hassle to support Python 2 and Python 3
- ✓ Don’t insist on backward compatibility
- ✓ Move from Doxygen/Epydoc to Sphinx for the documentation
- ✓ Wrappers for wxWidgets C++ classes are generated with SIP instead of SWIG
- ✓ Python 2.7 and Python 3.2+ supported (no older releases)
- ✓ Better/more stable handling of the GIL

The project is referred to as *Phoenix*, to distinguish it from wxPython *Classic*

wxPython and Python 3 – Implementation



wxPython and Python 3 – *wx.lib*

Support for Python 2 and 3…

1. `text = wx.TextCtrl(parent, value=u'Hello')` → Syntax error in Python 3.2

```
if PY3:
    def u(s):
        return s
else:
    def u(s):
        return unicode(s, 'unicode_escape')
```

   There are literally thousands of these `u'something'` in *wx.lib…*

2. *cPickle* vs. *pickle*, *cStringIO* vs. *StringIO* (and *BytesIO*), byte and text literals
3. *print* vs. *print()* – why oh why…
4. Removal of *cmp=* as keyword for *sort*
5. Many others…

We created a bridge tool (*wx2to3.py*) similar to the *six* package

wxPython and Python 3 – Backward Incompatibilities

1. Overloaded methods:

SetDimensions *(x, y, width, height, sizeFlags=wx.SIZE_AUTO)*
SetRect *(rect)*
SetSize *(size)*
SetSizeWH *(width, height)*
Classic

SetSize *(*args, **kwds)*
Phoenix

wx.Window example

2. *wx.PyDeadObjectError → RuntimeError*
3. *wx.PyAssertionError → wx.wxAssertionError*
4. Reorganization of the *wx* namespace and sub-modules
5. 2-phase creation has changed:

```
class MyDialog(wx.Dialog):
    def __init__(self, parent, ID, title):
        pre = wx.PreDialog()
        pre.SetExtraStyle(wx.FRAME_EX_CONTEXTHELP)
        pre.Create(parent, ID, title)
        self.PostCreate(pre)
```
Classic

```
class MyDialog(wx.Dialog):
    def __init__(self, parent, ID, title):
        wx.Dialog.__init__(self)
        self.SetExtraStyle(wx.FRAME_EX_CONTEXTHELP)
        self.Create(parent, ID, title)
```
Phoenix

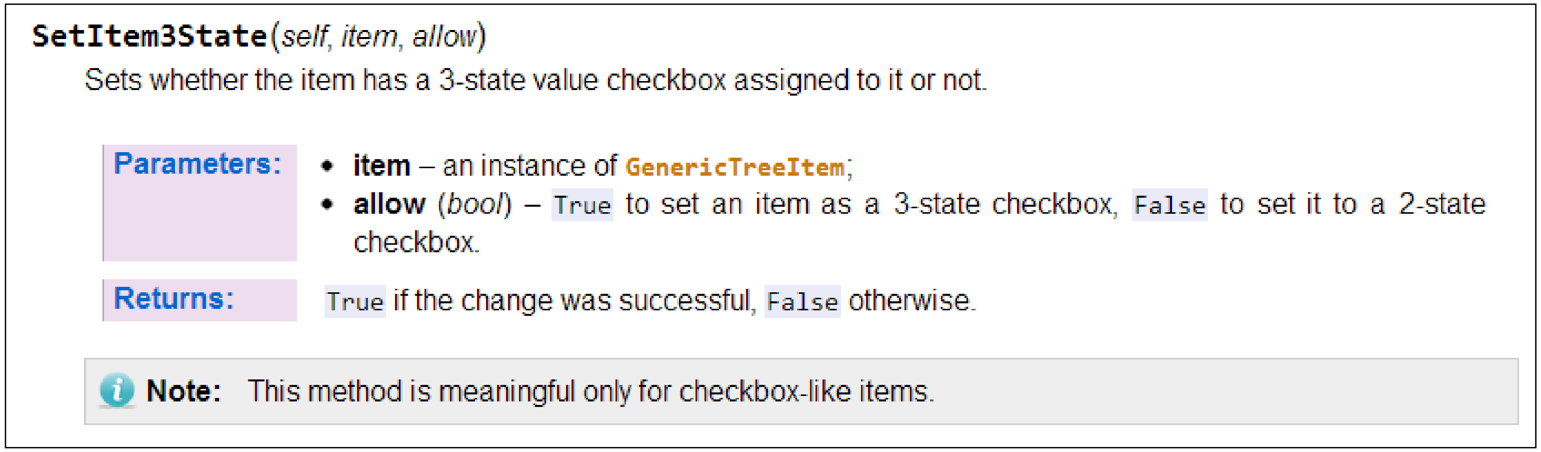
wxPython and Python 3 – Current Status

- ✓ Wrapped core classes (≈100 widgets) work with Python 2 and Python 3
  - http://wxpython.org/Phoenix/ItsAlive/
- ✓ Pure-Python controls:
  - Few modules in *wx.lib* have been ported
  - Almost all AGW widgets are Python 3-ready
  - Two different SVN repositories (*Classic* and *Phoenix*) for these widgets
- ✓ *Phoenix* can already be used in production mode if you only need core controls
- ✓ Daily preview snapshot builds are available:
  - http://wxpython.org/Phoenix/snapshot-builds/
- ✓ Buildbot builds and results display for all platforms:
  - http://buildbot.wxpython.org:8010/

wxPython and Python 3 – Current Status

✓ Docstrings are extracted from wxWidgets C++ docs, tweaked and adapted to Phoenix Python syntax

✓ Sphinx is then used on these modified docstrings:



`SetItem3State`(*self, item, allow*)

Sets whether the item has a 3-state value checkbox assigned to it or not.

Parameters:	• **item** – an instance of `GenericTreeItem`; • **allow** (*bool*) – `True` to set an item as a 3-state checkbox, `False` to set it to a 2-state checkbox.
Returns:	`True` if the change was successful, `False` otherwise.

Note: This method is meaningful only for checkbox-like items.

✓ Lots of inline samples/code snippets in the documentation (we need more)

✓ Documentation builds are automated via buildbot

Please consider contributing to the documentation effort!

wxPython and Python 3 – Roadmap

- ✓ Current roadmap considers *Phoenix* to be complete by Q1/Q2 2013
  - But this is just a guesstimate
- ✓ For existing applications, transition from *Classic* to *Phoenix* may take some effort
  - Mostly due to backward-incompatible changes between *Phoenix* and *Classic*
  - But I ported most of AGW to *Phoenix* in about 6 hours
- ✓ Once *Phoenix* is up and running, *Classic* will be discontinued
- ✓ Testers are more than welcome ☺
  - Batter the wrapped core classes for robustness
  - Abuse the *wx.lib* and AGW widgets to uncover incompatible leftovers

wxPython sample: ***python3.py***

Conclusions

A few useful links

- ✓ Download wxPython: http://wxpython.org/download.php
- ✓ wxPython Wiki: http://wiki.wxpython.org/

- ✓ AGW main page: http://xoomer.virgilio.it/infinity77/AGW_Docs/index.html

- ✓ Phoenix Project: http://wiki.wxpython.org/ProjectPhoenix
- ✓ Phoenix docs:
  - http://wxpython.org/Phoenix/docs/html/main.html
  - http://wxpython.org/Phoenix/docs/html/DocstringsGuidelines.html

- ✓ Presentation samples: http://www.infinity77.net/pycon/wxPython.zip

Thank You

Questions?

Comments?