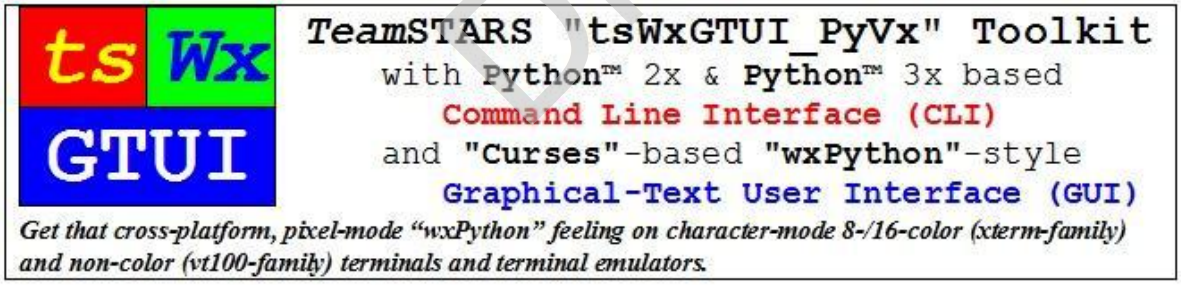
Software Gadgetry

To-Do

Vol. 12 - "tsWxGTUI_PyVx" Toolkit

Rev. 0.0.2 (Pre-Alpha)

Author(s): Richard S. Gordon



Printed on 8 September, 2015

Contents

1 REQUIREMENTS ISSUES

1.1 System Requirements

1.2 Functional Requirements

1.3 Interface Requirements

1.4 Qualification Requirements

1.5 Documentation Requirements

1.6 Release Requirements

1.7 Training Requirements

1.8 Support Requirements

2 DESIGN ISSUES

3 IMPLEMENTATION ISSUES

4 QUALIFICATION ISSUES

5 OPERATION ISSUES

5.1 Design Issues

1 2011/11/10

a) Application programs cannot delete wxPython-style GUI objects because Python's "nCurses" module does not have the "nCurses_delwin" method for deleting "nCurses" style GUI objects.

b) Not sure this issue can be resolved. A search for an explanation for why Python does not support the delvin function, came upon the following "nCurses" man page entry: Calling delwin deletes the named window, freeing all memory associated with it (it does not actually erase the window's screen image). Subwindows must be deleted before the main window can be deleted.

2 2011/11/10

a) Undependable Abort Signal (Ctrl-C) Handling. Applications occasionally terminate without restoring the display to its previous state in which keyboard input is echoed to the display.

b) Troubleshooting Hints: The operator must type "stty sane" to manuallly restore the display to its normal state. The operator might want to Clear Screen by typing Ctrl-L shell command"^L".

3 2012/07/03

a) Conflict between need for wxPython's "Show" method and wxPython emulator's need for nCurses' Panel Library

b) Limited screen real estate typically necessitates the overlapping of window objects. When the operator needs to see the features of a partially or totally hidden object, a Graphical User Interface uses a Task Bar with buttons for each frame or dialog. After moving the mouse over the button for the hidden object, the operator can click on the mouse button to bring a hidden background object to the visible foreground. The nCurses-based design provides a similar Task Bar.

c) However, the current design's use of wxWidgets-/ wxPython-style "Show" method is incompatible with use of the nCurses Panel (Stack) library as evidenced by sample code (See http://tldp.org/HOWTO/ NCURSES-Programming-HOWTO/panels.html). The sample code doesn't use components associated with "Show" method, but the wxPython emulation depends on them to accomodate a delay in the creation of nCurses windows until after the completion of associated distributed layouts.

4 2012/08/11

a) Lack of application program blocking mechanism for keyboard input.

b) Keyboard input subject to restrictions: tsWxPasswordEntryDialog is derived from tsWxTextEntryDialog. The latter uses tsWxTextEditBox which has usability issues in both non-event driven and event driven modes.

c) The non-event-driven mode uses curses.textpad.Textbox which monopolizes GUI platform and ignores mouse button clicks until the operator terminates input via double entry of control-G key. It also, responds to operator input after the control-G.

d) The event driven mode uses an python emulation of curses.textpad.Textbox that provides enhanced edit capabilities while recognizing mouse button clicks and keyboard press-release activity. However, the display blinks as each new text character is displayed. More importantly, there is no mechanism to block the application program pending the operator's completion of text input.

5 2012/11/16

a) Mouse "Click" Event handling is unreliable. What works on Windows with Cygwin sometimes crashes on Ubuntu Linux.

b) Troubleshooting Hint: Detected caret position, at time of mouse click, considered to be outside of displayed GUI object border. Perhaps displayed GUI object border must be pre-aligned with character cell display.

6 2012/11/16

a) Lack of Pending Event handling. wxPython-style event handling should include front-end (real time) dispatching or queuing with back-end (idle time) ProcessPendingEvent handling.

b) While the wxPython emulation is under construction, the work-around involves front-end use of tsWxProcessSelectedEventTable. The workaround has been used after much trial and error to demonstrate buttons, checkboxes, radio buttons and the scroll bar "arrow" buttons and gauges associated with text scrolling.

7 2013/05/09

a) Xemacs Syntax Error: "GUI item produces too long displayable string". Xemacs reports syntax error when editing tsApplication and tsCommandLineParser modules. Xemacs unable to create list of functions in the file being edited and without the list one can only use search to find a the function entry point. Xemacs takes minutes to format the file for viewing after edit changes.

b) Refactoring module to create separate files for each function restores normal Xemacs capabilities but requires modifying the function to simulate the class member behavior. Modifications include: a) import module as function; b) self.function = function; c) explicitly referencing function as self.function(self)

8 2013/07/16

a) Need Frame and Dialog control button event handlers.

b) Frame control buttons (ICONIZE, MAXIMIZE/RESTOREDOWN, and CLOSE) do not make appropriate changes to display. The handlers ony output an appropriate text message.

c) Dialog control buttons (HELP and CLOSE) do not make appropriate changes to display. The handlers ony output an appropriate text message.

9 2013/08/16

a) Hierarchical library file structure interferes with operation of pylint and pydoc.

b) Pylint is a tool that checks for errors in Python code, tries to enforce a coding standard and looks for bad code smells. This is similar but nevertheless different from what pychecker provides, especially since pychecker explicitly does not bother with coding style. The default coding style used by Pylint is close to PEP 8 (aka Guido's style guide).

c) Pydoc is a documentation module for the programming language Python. Similar to the functionality of Perldoc within Perl, Pydoc allows Python programmers to access Python's documentation help files, generate HTML pages with documentation specifics, and find the appropriate module for a particular job. Pydoc can be accessed from a module-specific GUI, from within the Python interpreter, or from a command line shell.

d) Pylint and Pydoc expect to process files from within a monolithic file directory. When invoked within a hierarchical file directory with its hierarichal "package" import mechanism, they report various import errors which degrade their usefulness.

10 2013/08/20

a) Python files must be "untabified" before being processed by tsStripComments to avoid "IndentationError".

b) Failure to "untabify" Python source files prior to processing by "tsStripComments" may need manual fix-up of IndentationError.

11 2013/08/24

a) tsStripComments renders unusable such doc string dependent modules as tsOperatorSettingsParser. Blank lines within doc strings are visually used to define the end of one paragraph and the start of a new one.

b) tsOperatorSettingsParser uses the text wrap module to ensure that text fits within the available run time display area. Text wrap depends on blank lines to avoid merging text from separate paragraphs.

c) Authoring doc strings with blank lines is both natural and effective at achieving the correct display.

d) tsStripComments originally did not recognize the appropriatness of removing some blank lines but not others. An update removed all but the first of a sequence of consecutive blank lines.

e) The main application using a tsOperatorSettingsParser-like module passes the module's launch purpose via a doc string which is subject to stripping. All other launch parameters (such as title, author, build version and build date) are passed by reference to the associated labeled doc string.

f) See BuildPurpose AttributeError for traceback example when doc string deleted and Purpose = __doc__ attempted to pass a NoneType.

g) A simple work around would be to create a purpose labeled doc string which will not be subject to comment stripping. A reference to the label can then be passed instead of the one to the unlabeled doc string.

2013/09/04

test_tsWxGridSizer event processing generates too many event messages per button click. Button click event handling needs to be directly coupled to keybad button window rather than coupled to the parent Frame provided handler.

2013/09/04

Partially-compliant wxPython User Interface. Keyboard input subject to restrictions:

tsWxPasswordEntryDialog is derived from tsWxTextEntryDialog. The latter uses tsWxTextEditBox which has usability issues in both non-event driven and event driven modes.

The non-event-driven mode uses curses.textpad.Textbox which monopolizes GUI platform and ignores mouse button clicks until the operator terminates input via double entry of control-G key. It also, responds to operator input after the control-G.

The event driven mode uses an python emulation of curses.textpad.Textbox that provides enhanced edit capabilities while recognizing mouse button clicks and keyboard press-release activity. However, the display blinks as each new text character

is displayed. More importantly, there is no mechanism to block the application program pending the operator's completion of text input.

2013/09/05

2to3-3.2.py RefactoringTool: There were 3 errors:

a. RefactoringTool: Can't parse ./tsWxLayout_h.py:
ParseError: bad input: type=48, value='//', context=('', (1, 0))

b. RefactoringTool: Can't parse ./tsWxLayoutGTUI.py:
TokenError: ('EOF in multi-line string', (126, 4))

c. RefactoringTool: Can't parse ./tsWxTextEntry.py:
ParseError: bad input: type=1, value='wxString',
context=(' ', (270, 31))

2013/09/05

2to3-3.2.py RefactoringTool: There were 3 errors:

a. RefactoringTool: Can't parse ./test_tsWxEvents.py:
ParseError: bad input: type=22, value='=', context=('', (878, 45))

b. RefactoringTool: Can't parse ./test_tsWxStatusBar.py:

ParseError: bad input: type=22, value='=', context=(", (859, 45))

c. RefactoringTool: Can't parse ./test_tsWxTaskBar.py:

ParseError: bad input: type=22, value='=', context=(", (657, 45))

2013/09/06

tsConfigObjectPkg only available for Python-2.x.

Python 3.x version of tsLib/tsConfObjectPkg/ reports TypeError in configobj.py line 1380, in _iced because:

'int' object "line[-1]" is not subscripted

The source version, 4.7.2, was designed for Python 2.x. Its author, Michael Foord, has not ported in to Python 3.x. He "blessed" a port created by Zubin Mithra and Prashant Kuma but it cold not be found on the referenced wiki and the 2to3 tool is unable to produce a workable version.

5.2 Display Issues

Display Issues

2010/08/25

xterm-256color consoles only support the 8x8 standard "nCurses" foreground/ background color combinations.

Use of the 68x68 standard "wxPython" foreground/background color combinations produces either the incorrect color and/or such unwanted display attributes as underlines, blinking etc.

Google Search: "NCurses 256 Color Support":

"But note that some terminals, while they can support 256 colors, are not able to change the palette. To compile NCurses with 256 color support, ...

from: www.c-for-dummies.com/nCurses/256color/

During experimentation with "init_color.c" example from Dan Gookin's "Programmer's Guide to nCurses", observed that nCurses silently rejects changes to any of its standard 8-color definitions. It also silently rejects changes to any of the

64 color pair definitions associated with the standard 8-color definitions. Observed that color numbers 8 and above can be defined and that color pair numbers 64 and above can be defined.

Conclusion: Python uses standard nCurses. It therefore only supports 8 colors.

2011/12/06

GUI Object Automatic Layout Anomalies

Use of wxPython-style sizers to automatically layout complex assemblies of GUI Objects typically produce the telescoping or overlapping of adjacent border lines into a single line.

Conclusion: Use of wxPython-style pixel dimensions are associated with the nearest available nCurses column and row. Anomalies show up only when derived pixel dimensions are not integer multiples of the pixel width and height for the fixed width font of a single character. See an example at wxStaticBoxSizer.

2011/12/06

SplashScreen Bitmap Image Compatibility

Use of a SplashScreen Bitmap image imay be platform dependent. An image created and saved on a Cygwin console reloads and is displayed only on a Cygwin console. It may not reload and display on a Linux xterm.

Conclusion: It may be necessary to create, save and install a Bitmap image for each platform. It may be desirable for the design to support platform-specific images via platform-specific names.

2013/03/05

Ubuntu Linux 12.04 terminal those runs "tsWxGTUI" applications associated with regression testing. Occassionally, a mouse click triggers an application trap for what appears to be an unknown cursor mis-alignment issue.

Test Platforms:

Linux Ubuntu 12.04 32-bit
Mac OS X (10.7.5) 64-bit
Windows XP 32-bit (with cygwin, Unix-type environment)
Windows 7 Professional 32-bit (with cygwin, Unix-type environment)
Windows 8 Professional 32-bit (with cygwin, Unix-type environment)

Command Line Interface regression tests (Python 2.x and 3.x)

include the following:

test_tsApplication.py

test_tsCommandLineInterface.py

test_tsDecorators.py

test_tsExceptions.py

test_tsLogger.py

test_tsReportUtilies.py

test_tsThreadPool.py

tsLinesOfCode.py

Graphical User Interface regression tests (Python 2.x and 3.x)

include the following:

test_tsWxBoxSizer.py

test_tsWxGridSizer.py

test_tsWxMetrics.py

test_tsWxScrolledWindow.py (mouse click trap symptoms)

test_tsWxWidgets.py

5.3 Import Issues

Import Issues

2011/06/01

Non-compliant wxPython Library Modules

Functions, Class Methods and Class Properties:

See Microsoft Access file:

2011/06/01

Non-compliant wxPython Aoolications

Unit, Integration and System Test Applications:

test_tsWxWidgets - Runs without failure. Unit Test for widgets.

test_tsWxPySimpleApp - Runs without failure. Unit Test for startup.

2013/03/05

Named import module not found.

Despite establishment of tsLibCLI and tsLibGUI, importing within repository from test directories may fail.

Problem can be eliminated by installing "tsWxGTUI" into default Python's site-packages using the appropriate setup.py tool.

5.4 Installation Issues

Installation Issues

2013/03/25

Access to the "tsToolsCLI" services should be path independant from the top level of the "tsWxGTUI" Toolkit site package.

Access to the "tsToolsCLI" services is only via their source code path. Access should also be path independant from the top level of the "tsWxGTUI" Toolkit site package.

tsLibraryImport

tsLinesOfCode

tsPlatformQuery

tsPublish

tsReAuthor

tsReImport

tsReVersion

tsStripComments

tsStripLineNumbers

tsTreeCopy

tsTreeTrimLines

2013/03/25

Access to the "tsToolsGUI" services should be path independant from the top level of the "tsWxGTUI" Toolkit site package.

Access to the "tsToolsGUI" services is only via their source code path. Access should also be path independant from the top level of the "tsWxGTUI" Toolkit site package.

tsWxLinesOfCode (Future)

tsWxPlatformQuery (Future)

5.5 Operational Issues

Operational Issues

None

5.6 Remote Access Issues

Remote Access Issues

2012/04/07

A local platform connected to one or more remote platforms, which together are operating in "Stand-Among Mode", may report various connection errors.

Use of the OpenSSH SSH client (remote login program) via the command "ssh <user id>@<remote host id>" may require the System Administrators, Software Engineers or System Operators to temporarily supend or permanently modfy local and/or remote computer security settings:

Authorize local computer on remote system to resolve the issue:
"ssh: connect to host <IP Address> port 22; Connection refused"

Unblock firewall on remote system to resolve the issue:
"sh: connect to host <IP Address> port 22; Connection timed out"

5.7 Troubleshooting Issues

Troubleshooting Issues

2013/09/04

Instrumentation to facilitate debugging has been left activated. Module level variables have been set to enable built-in code to log and display both normal progress and unexpected events.

The "tsWxGTUI" Toolkit is a work in progress. Unless one is introducing only localized changes, remembering where to find and turn debug control flags "on" and "off" quickly become tedious and vexing. Ignoring the automatically generated information until debugging is required is relatively painless.

Consider the lines-of-code metrics and debugging effort associated with just the primary Python version 2.x files (including documentation, tests and associated data but excluding their derived Python 3.x counterparts):

Overall Source Code Feature Statistics

	FILES	CODE	CMNTS	LINES	WORDS	CHARS
Pct:		48.51%	51.49%	100.00%		

Totals:	799	211904	224945	436849	1325892	16512687
Std:	722	399	450	795	2431	38612
Avg:	1	265	282	547	1659	20667

Distribution of Source Code Feature Statistics by File Types

TYPES	FILES	CODE	CMNTS	LINES	WORDS	CHARS	%-LINES
.ada	4	904	176	1080	4604	33596	0.25%
.adb	9	701	387	1088	3814	32307	0.25%
.ads	5	148	202	350	1886	14552	0.08%
.asm	15	355	452	807	3032	17602	0.18%
.bas	8	2160	8	2168	7184	46828	0.50%
.bash	4	24	140	164	772	4600	0.04%
.bat	4	16	8	24	32	204	0.01%
.c	22	6185	5117	11302	32960	262430	2.59%
.cpp	8	3712	1704	5416	14532	159776	1.24%
.f	8	2380	464	2844	9308	85340	0.65%
.f77	4	920	348	1268	4208	29836	0.29%
.f90	48	7620	5683	13303	48953	419506	3.05%
.ftn	4	236	168	404	1548	10160	0.09%
.h	10	307	223	530	1734	15481	0.12%
.inc	12	696	2292	2988	11708	133320	0.68%
.pas	23	4416	3782	8198	29599	231170	1.88%
.plm	8	3436	2980	6416	24768	191336	1.47%
.py	562	176695	179318	356013	1042775	12263799	81.50%
.sh	12	993	315	1308	5351	60881	0.30%

.txt	28	0	20372	20372	74999	2479155	4.66%
.y	1	0	806	806	2125	20808	0.18%

2013/09/04

Development notes ("comments" and "doc strings") are embedded within each Python source file.

Prior to publication the notes must be reviewed and edited to ensure accuracy, completeness and clarity.

For a pre-release said notes should be stripped out of each source file to be published. The "tsStripComments.py" tool is provided for this purpose. Even after note stripping, all executable statements (including imports, definitions, logic and input/output) will be fully human readable.

The "tsStripComments.py" tool must be used with care on those top-level application or test files which provide an operator with command line help. This is because general purpose command line parsers, such as "tsOperatorSettingsParser", expect to receive an application-specific purpose-defining doc string, text contained within triple quotes (such as """Module Purpose """ or '''Module Purpose''') that must not be stripped. The purpose doc string is typically unlabeled at or near the beginning of the file. It is referenced by "__doc__". To ensure that

it is not stripped, it is suggested that the "__doc__" label be explicitly defined via one of the following:

```
# Implicit label definition override
__doc__ = """Module Purpose"""
__doc__ = '''Module Purpose'''

# Explicit new label definition
purpose = """Module Purpose"""
purpose = '''Module Purpose'''
```

5.8 User Input Issues

User Input Issues

2008/09/03

Manually Sized Curses Screen

Host Operating Systems typically provide a Graphical User Interface that operates in pixel-mode and can be programatically sized by an application during its startup.

Terminal emulators typically provide a Graphical-style User Interface that operates in character-mode but programatically sized by an application during its startup.

Troubleshooting Hints:

a. Operating Systems typically provide user changeable properties to change the font and layout size of the shell window inwhich the application will be run.

b. If the application aborts itself because it started in an under sized shell window, the operator should re-adjust the shell window's font and layout properties as appropriate.

2010/08/25

Partially-compliant wxPython User Interface

Keyboard input subject to restrictions:

a. Apple "Cmd" (Command) Not supported on computer platforms running Mac OS X, Linux or Windows.

b. Only "character" events supported; "key" press/release events Not supported.

c. Only "Shift", "Ctrl" or"Alt" key can be pressed in combination with a character key; combinations of "Shift", "Ctrl" and "Alt" keys are Not supported. (Note: The "Alt" key event internally involves two separate "nCurses" getchar operations.)

2012/08/11

Partially-compliant wxPython User Interface

Keyboard input subject to restrictions:

a. tsWxPasswordEntryDialog is derived from tsWxTextEntryDialog.

The latter uses tsWxTextEditBox which has usability issues in both non-event driven and event driven modes.

b. The non-event-driven mode uses curses.textpad.Textbox which monopolizes GUI platform and ignores mouse button clicks until the operator terminates input via double entry of control-G key. It also, responds to operator input after the control-G.

c. The event driven mode uses an python emulation of curses.textpad.Textbox that provides enhanced edit capabilities while recognizing mouse button clicks and keyboard press-release activity. However, the display blinks as each new text character is displayed. More importantly, there is no mechanism to block the application program pending the operator's completion of text input.

2013/07/18

Partially-compliant wxPython User Interface

test_tsWxGridSizer event processing generates too many event messages per button click. Button click event handling needs to be directly coupled to keybad button window rather than coupled to the parent Frame provided handler.