

Appendixes

Vol. 10 - "tsWxGTUI_PyVx" Toolkit

Rev. 0.0.6 (Pre-Alpha)

Author(s): Richard S. Gordon



Author Copyrights & User Licenses for "tsWxGTUI_Py2x" & "tsWxGTUI_Py3x" Software & Documentation

- Copyright (c) 2007-2009 Frederick A. Kier & Richard S. Gordon, a.k.a. *TeamSTARS*. All rights reserved.
- Copyright (c) 2010-2016 Richard S. Gordon, a.k.a. Software Gadgetry. All rights reserved.
- GNU General Public License (GPL), Version 3, 29 June 2007
- GNU Free Documentation License (GFDL) 1.3, 3 November 2008

Third-Party Component Author Copyrights & User Licenses

- Attribution for third-party work directly or indirectly associated with the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit are detailed in the "COPYRIGHT.txt", "LICENSE.txt" and "CREDITS.txt" files located in the directory named `./tsWxGTUI_PyVx_Repository/Documents`.

Draft

Contents

1	INTRODUCTION	5
1.2	APPENDIX A - BASELINE HOST COMPUTER PLATFORMS	7
1.2.1	Currently Used Platforms (Release Notes)	8
1.2.2	Previously Used Platforms (Release Notes)	13
1.2.3	Designing Embedded Systems with Linux and Python	16
1.3	APPENDIX B - API RELATIONSHIP	25
1.3.1	Comparison with “wxPython”	25
1.3.2	High, Low and Extended API Relationships	29
1.4	APPENDIX C - DELIVERABLES	51
1.5	APPENDIX D - ACCOMPLISHMENTS (Draft)	57
1.5.1	Capabilities	57
1.5.2	Limitations	70
1.5.3	Comparison of wxPython with tsWxGTUI (Development Plan)	71
1.5.4	Benefits	74
1.6	APPENDIX E - INHERITED, FIELD-PROVEN COMPUTER TECHNOLOGY	77
1.6.1	VGA-compatible Text Mode	78
1.6.2	POSIX	86
1.6.3	Command Line Interface (CLI)	94
1.6.4	Graphical User Interface (GUI)	107
1.6.5	Python Programming Language	117
1.7	APPENDIX F - HISTORY OF SYSTEM DEVELOPMENT, OPERATION, AND MAINTENANCE	171
1.7.1	System Development	171
1.7.2	System Maintenance	178
1.7.3	System Operation	181
2	SYSTEM SPECIFICATION	183
2.2	APPENDIX A - REQUIRED STATES AND MODES	185
2.2.1	Required States	185
2.2.2	Required Modes	187
2.3	APPENDIX B - SYSTEM CAPABILITY REQUIREMENTS	195
2.3.1	Hardware Capabilities	195
2.3.2	Software Capabilities	205
2.3.3	Platform Interface Capability	242
2.3.4	Application Programming Interface Capability	242
2.3.5	Operator Interface Capability	248
2.4	APPENDIX C - SYSTEM EXTERNAL INTERFACE REQUIREMENTS	251
2.4.1	(Project-unique Identifier Of Interface Template)	252
2.4.2	Interface Identification and Diagrams	254
2.5	APPENDIX D - SYSTEM INTERNAL INTERFACE REQUIREMENTS	277
2.5.1	Toolkit Architecture	277

2.6	APPENDIX E - SYSTEM INTERNAL DATA REQUIREMENTS	281
2.7	APPENDIX F - ADAPTATION REQUIREMENTS	283
2.8	APPENDIX G - SAFETY REQUIREMENTS	285
2.9	APPENDIX H - SECURITY AND PRIVACY REQUIREMENTS.....	287
2.10	APPENDIX I - SYSTEM ENVIRONMENT REQUIREMENTS	289
2.11	APPENDIX J - COMPUTER RESOURCE REQUIREMENTS	291
2.12	APPENDIX K - SYSTEM QUALITY FACTORS	295
2.13	APPENDIX L - DESIGN AND CONSTRUCTION CONSTRAINTS.....	297
2.14	APPENDIX M - PERSONNEL-RELATED REQUIREMENTS.....	299
2.15	APPENDIX N - TRAINING-RELATED REQUIREMENTS	301
2.16	APPENDIX O - LOGISTICS-RELATED REQUIREMENTS	303
2.17	APPENDIX P - OTHER REQUIREMENTS	305
2.18	APPENDIX Q - PACKAGING REQUIREMENTS.....	307
2.19	APPENDIX R - PRECEDENCE AND CRITICALITY OF REQUIREMENTS.....	309

3 INTERFACE REQUIREMENTS SPECIFICATION 311

4 SOFTWARE REQUIREMENTS SPECIFICATION 313

4.2	APPENDIX A - Nature of System and Software	314
4.2.1	Command Line and Graphical User Interfaces	315
4.2.2	System Configurations	315
4.2.3	Computer User Use Case(s).....	317
4.3	APPENDIX B - Software Design Constraints	323
4.3.1	README	324
4.3.2	README1-Introduction.txt	330
4.3.3	README2-Repository.txt.....	348
4.3.4	README3-Documents.txt.....	357
4.3.5	README4-ManPages.txt	360
4.3.6	README5-Notebooks.txt.....	367
4.3.7	README6-SourceDistributions.txt	374
4.3.8	README7-DeveloperSandboxes.txt	390
4.3.9	README8-SitePackages.txt.....	406

5 RELEASE NOTES 413

5.2	APPENDIX A - BASELINE HOST COMPUTER PLATFORMS	415
5.2.1	Currently Used Platforms (Release Notes)	416
5.2.2	Previously Used Platforms (Release Notes)	421
5.3	APPENDIX B - API RELATIONSHIP	425
5.3.1	High, Low and Extended API Relationships	425
5.4	APPENDIX C - DELIVERABLES.....	427

6 SOFTWARE USERS MANUAL 433

6.2	APPENDIX A - BASELINE HOST COMPUTER PLATFORMS	435
6.2.1	Currently Used Platforms (Release Notes)	436
6.2.2	Previously Used Platforms (Release Notes)	441
6.3	APPENDIX B - API RELATIONSHIP	445
6.3.1	Comparison of wxPython with tsWxGTUI (Development Plan)	445
6.3.2	High, Low and Extended API Relationships	448

6.4	APPENDIX C - DELIVERABLES.....	449
6.5	APPENDIX D - LOG FILES	455
6.5.1	Log File Examples	455

Draft

Draft

CHAPTER 1

1 INTRODUCTION

1.1.1.1 In This Chapter

APPENDIX A - BASELINE HOST COMPUTER PLATFORMS	7
APPENDIX B - API RELATIONSHIP	25
APPENDIX C - DELIVERABLES	51
APPENDIX D - ACCOMPLISHMENTS (Draft)	57
APPENDIX E - INHERITED, FIELD-PROVEN COMPUTER TECHNOLOGY	77
APPENDIX F - HISTORY OF SYSTEM DEVELOPMENT, OPERATION, AND MAINTENANCE	171

Draft

Draft

1.2 APPENDIX A - BASELINE HOST COMPUTER PLATFORMS

This section shall identify those host computer platforms actually used to plan, implement, test, document, deploy and maintain the "tsWxGTUI_PyVx" Toolkit.

- 1 *Currently Used Platforms*** (see "***Currently Used Platforms (Release Notes)***" on page 8) - Linux (Fedora 22 / OpenSUSE 13.2 / Scientific (CentOS) 7 / Ubuntu 12.04, 14.04 & 15.04), Mac OS X (10.7 / 10.11), Microsoft Windows (7 Pro. 32-bit with Cygwin 1.7.x / 10 Pro. 64-bit with Cygwin 2.2) / and Unix (PC-BSD 10 & OpenIndiana 151a8 / Solaris 11 / SunOS 5.11).
-

The Toolkit's Python source code has been developed and tested only with Intel x86 and x64 processors and representative GNU/Linux, Mac OS X, Microsoft Windows and Unix operating system releases.

Its source code has been compiled, interpreted and executed by Python 2x and 3x Virtual Machines developed by the Python Software Foundation (PSF).

The PSF also distributes equivalent Python 2x and 3x Virtual Machines (and the source code to build them) for other processor types and operating systems, some of which are listed below.

See file **"/tsWxGTUI_PyVx_Repository/Notebooks/EngineeringNotebook/MS-Excel-Files/Platform_Configuration.xls"** for Python version specific platform configuration details (Sheet 1 --- Host Configurations; Sheet 2 --- Terminal Configurations):

- a) 2013-model year, 3.5 GHz Intel Quad Core i7 processor-based desktop with 16 GB RAM and sufficient performance, resources and expansion capabilities to serve as the high-level baseline development and operator platform.
 - b) 2007-model year, 2.33 GHz Intel Core 2 Duo processor-based laptop with 4 GB RAM and sufficient performance, resources and expansion capabilities to serve as the moderate-level baseline development and operator platform.
 - c) 1998-model year, 366 MHz Intel Pentium II-based laptop with 384 MB RAM and expansion capabilities (marginal resources and performance) sufficient enough to serve as the low-level baseline development and operator platform.
- 2 *Previously Used Platforms*** (see "***Previously Used Platforms (Release Notes)***" on page 13) - Linux (Fedora 15 & 16 / Open SuSE 11 / Ubuntu 10.04), Mac OS X (10.4 / 10.5 / 10.6 / 10.7), Microsoft Windows (8 Professional / 7 Professional / XP Professional each with Cygwin 1.7.x) and Unix (OpenIndiana 151a6 / Solaris 11 / SunOS 5.11).
- a) 2007-model year, 2.33 GHz Intel Core 2 Duo processor-based laptop with 4 GB RAM and sufficient performance, resources and expansion capabilities to serve as the moderate-level baseline development and operator platform.
 - b) 1998-model year, 366 MHz Intel Pentium II-based laptop with 384 MB RAM and expansion capabilities (marginal resources and performance) sufficient enough to serve as the low-level baseline development and operator platform.

- 3** *Designing Embedded Systems with Linux and Python* (on page 16) - Link to YouTube Video by Mark Kohler.

1.2.1 Currently Used Platforms (Release Notes)

Host Computer Platform Configurations currently used by "tsWxGTUI_PyVx" Toolkit developers:

Draft

Draft

Make & Model	Hardware	Software
Apple 27" iMac Desktop	<p>2013-model year, 3.5 GHz Intel Quad Core i7 processor-based desktop with 16 GB RAM, 2560x1440 pixel resolution display and sufficient performance, resources and expansion capabilities to serve as the high-level baseline development and operator platform.</p> <ul style="list-style-type: none"> ▪ Its 3 TB (7200 RPM) SATA 6 Gb/s internal hard drive had 128 GB Solid State Flash memory and was used to run Apple's Mac OS X 10.9-19.10 and the hypervisor virtualization applications (Parallels Desktop 9-10 and VMware Fusion 5 and 7.1) that supported various guest operating systems. ▪ Its 3 TB (7200 RPM) SATA 6 Gb/s internal hard drive was also used to store and run configured versions of the eComStation Demo CD version 2.2 BETA (IBM OS/2 4.5 Warp descendant), CentOS Linux (7.0), Fedora Linux (21), OpenSuSE Linux (13.1), Scientific Linux (6.5), Ubuntu Linux (12.04, 14.04), Microsoft Windows (8.1 Pro, 8 Pro, 7 Pro, XP Pro and 2000 Pro), and Unix (PC-BSD (9.2, 10.0), OpenIndiana 151A8) guest operating systems. ▪ Hewlett-Packard Company P2015DN Series (26A5C8) LaserJet Printer conected via Ethernet ▪ Hewlett-Packard Company Officejet Pro 8500A (A910) e-All-in-One Series Printer connected viia Wi-Fi or USB. 	<p>Host Operating System</p> <ul style="list-style-type: none"> ▪ Apple's Mac OS X (initially Mavericks releases 10.9.x and currently Yosemite releases 10.10.x) with Python 2.6.8, 2.7.5, 3.2.2 <p>Host Hypervisor Virtualization Applications</p> <ul style="list-style-type: none"> ▪ Parallels Desktop 9-11 ▪ VMware Fusion 5 & 7.1 <p>Concurrent or Interchangeable Guest Operating Systems (cloned from Apple 17" MacBook Pro Laptop and configured to share host drives and use 1440 x 900 display)</p> <ul style="list-style-type: none"> ▪ eComStation Demo CD version 2.2 BETA (IBM OS/2 4.5 Warp descendant) runs only what was shipped on the live CD and does not include Python. Most recent Python port for OS/2 & eComStations is Python 2.7.5. For details, see the following: "http://os2ports.smedley.id.au/index.php?page=python" and "http://blog.python.org/2011/05/python-33-to-drop-support-for-os2.html" ▪ Linux (CentOS 7.1 64-bit, Debian 8.2 64-bit, Fedora 22 & 23 64-bit, LXLE 14.02 32-bit, OpenSuSE 12.2 & 13.2 64-bit, Scientific 6.4 & 7.1 64-bit, Ubuntu 12.04, 14.04 & 15.10 32-/64-bit with Python 2.7 and 3.4. ▪ Microsoft Windows (10.0 Professional 64-bit, 10.0 Professional 32-bit, 8.1 Professional 32-bit, 8 Professional 32-bit, 7 Professional 32-bit with SP1, XP Professional 32-bit with SP3 and 2000 Professional 32-bit with SP2) with POSIX-like Cygwin plug-in and Python 2.6, 2.7, 3.2, 3.3 and 3.4. (NOTE: Windows 2000 came with obsolete Web Browser that precluded Windows 2000 updates and installaion of Cygwin. After copying Windows 2000 updates and Cygwin directory from XP, Windows 2000 ran the TeamSTARS "tsWxGTUI_PyVx" Toolkit CLI and GUI tests but did not suport mouse even with xterm.) ▪ UNIX (PC-BSD 9.2 & 11.0 64-bit without Parallels Tools, OpenIndiana 151a8 32-bit without Parallels Tools, a replacement for unreleased OpenSolaris 11/SunOS 5.11)

		with Python 2.6.4 running on Apple MacBook Pro
Apple 17" MacBook Pro Laptop	<p>2007-model year, 2.33 GHz Intel Core 2 Duo processor-based laptop with 4 GB RAM, 1920x1200 pixel resolution display and sufficient performance, resources and expansion capabilities to serve as the moderate-level baseline development and operator platform.</p> <ul style="list-style-type: none"> Its 160 GB (5400 RPM) SATA 1.5 Gb/s internal hard drive was used to run Apple's Mac OS X 10.7 and the hypervisor virtualization applications (Parallels Desktop 8 and VMware Fusion 5) that supported various guest operating systems. Its 1.5 TB (7200 RPM) SATA 3 Gb/s external hard drive was used to store and run configured versions of the Ubuntu Linux (12.04), Microsoft Windows (8 Pro, 7 Pro and XP Pro) and Unix (OpenSolaris 11/OpenIndiana 151) guest operating systems. Hewlett-Packard Company P2015DN Series (26A5C8) LaserJet Printer connected via Ethernet Hewlett-Packard Company Officejet Pro 8500A (A910) e-All-in-One Series Printer connected via Wi-Fi or USB. 	<p>Host Operating System</p> <ul style="list-style-type: none"> Apple's Mac OS X 10.7.5 with Python 2.6.8, 2.7.5, 3.2.2 <p>Host Hypervisor Virtualization Applications</p> <ul style="list-style-type: none"> Parallels Desktop 8 VMware Fusion 5 <p>Interchangeable Guest Operating Systems (configured to share host drives and use 1440 x 900 display)</p> <ul style="list-style-type: none"> Linux (Fedora 20 64-bit, OpenSuSE 12.2 64-bit, Scientific (CentOS) 6.4-6.5 64-bit, Ubuntu 12.04 32-bit) with Python 2.7 and 3.2. Windows (8.1 Professional 32-bit, 8 Professional 32-bit, 7 Professional 32-bit with SP1, XP Professional 32-bit with SP3) with POSIX-like Cygwin plug-in and Python 2.6, 2.7, 3.2 and 3.3 UNIX (PC-BSD 9.2-10.0 64-bit without Parallels Tools, OpenIndiana 151A8 32-bit without Parallels Tools, a replacement for unreleased OpenSolaris 11/SunOS 5.11) with Python 2.6.4 running on Apple MacBook Pro
Dell 15.6" Inspiron 7000 Laptop	<p>1998-model year, 366 MHz Intel Pentium II-based laptop with 384 MB RAM, 640x480/1024x768 pixel resolution display and expansion capabilities (marginal resources and performance) sufficient enough to serve as the low-level baseline development and operator platform.</p> <ul style="list-style-type: none"> Its interchangeable 32 GB (4200 RPM) ATA hard drives were used to run Windows XP Pro or Ubuntu 12.04 Linux. The platform's limited memory and available PCMCIA network adapters were incompatible with later versions of Windows or with other Linux distributions. (Windows XP recognized Xircom Ethernet and 3Com Wireless adapters; Ubuntu Linux 12.04 recognized only Linksys Wireless adapter.) Hewlett-Packard Company Photosmart C3180 All-in-One Printer, Scanner, Copier connected via USB. 	<p>Interchangeable Host Operating System</p> <ul style="list-style-type: none"> Windows XP Professional with SP3) with POSIX-like Cygwin plug-in and Python 2.6, 2.7, 3.2 and 3.3 Linux (Ubuntu 12.04) with Python 2.7 and 3.3

NOTE: Since cross-platform operating system and Python virtual machine technology is also available for

non-Intel based systems, it is likely that the TeamSTARS "tsWxGTUI_PyVx" toolkit will also work on those systems which use the equivalent operating systems and Python virtual machines with 32-bit and 64-bit microprocessors from other manufacturers including:

- *AMD*
- *ARM Holdings*
- *Cyrix*
- *Freescape*
- *Intel*
- *IBM*
- *Marvell*
- *NexGen*
- *Nvidia Tegra*
- *Oracle (previously Sun)*
- *OWC*
- *Qualcomm*
- *Rise Technology*
- *Samsung*
- *SigmaTel*
- *Texas Instruments*
- *Transmeta*
- *tilera*
- *Via (Centaur Technology division)*
- *winchip*

1.2.2 Previously Used Platforms (Release Notes)

Host Computer Platform Configurations previously used by "tsWxGTUI_PyVx" Toolkit developers:

Make & Model	Hardware	Software
Apple 17" MacBook Pro Laptop	<p>2007-model year, 2.33 GHz Intel Core 2 Duo processor-based laptop with 4 GB RAM and sufficient performance, resources and expansion capabilities to serve as the moderate-level baseline development and operator platform.</p> <ul style="list-style-type: none"> Its 160 GB (5400 RPM) SATA 1.5 Gb/s internal hard drive was used to run Apple's Mac OS X 10.7 and the hypervisor virtualization applications (Parallels Desktop 4-7 and VMware Fusion 4-5) that supported various guest operating systems. Its 1.5 TB (7200 RPM) SATA 3 Gb/s external hard drive was used to store and run configured versions of the Ubuntu Linux (10.04), Microsoft Windows (7 Pro and XP Pro) and Unix (OpenSolaris 11/OpenIndiana 151) guest operating systems. 	<p>Host Operating System</p> <ul style="list-style-type: none"> Apple's Mac OS X 10.4-10.7 with Python 2.6.8 <p>Host Hypervisor Virtualization Applications</p> <ul style="list-style-type: none"> Parallels Desktop 4-7 VMware Fusion 4-5 <p>Interchangeable Guest Operating Systems (configured to share host drives and use 1440 x 900 display)</p> <ul style="list-style-type: none"> Linux (Ubuntu 10.04) with Python 2.7 and 3.2 Windows (7 Professional / XP Professional with SP3) with POSIX-like Cygwin plug-in and Python 2.6, 2.7 and 3.2 UNIX (OpenIndiana 151a5, a replacement for unreleased OpenSolaris 11/SunOS 5.11) with Python 2.6.4 running on Apple MacBook Pro
Dell 15.6" Inspiron 7000 Laptop	<p>1998-model year, 366 MHz Intel Pentium II-based laptop with 384 MB RAM and expansion capabilities (marginal resources and performance) sufficient enough to serve as the low-level baseline development and operator platform.</p> <ul style="list-style-type: none"> Its interchangeable 32 GB (4200 RPM) ATA hard drives were used to run Windows XP Pro or Ubuntu 12.04 Linux. The platform's limited memory and available PCMCIA network adapters were incompatible with later versions of Windows or with other Linux distributions. (Windows XP recognized Xircom Ethernet and 3Com Wireless adapters; Ubuntu Linux 12.04 recognized only Linksys Wireless adapter.) 	<p>Interchangeable Host Operating System</p> <ul style="list-style-type: none"> Windows XP Professional with SP3) with POSIX-like Cygwin plug-in and Python 2.6, 2.7 and 3.2 Linux (Ubuntu 12.04) with Python 2.7 and 3.2

Notes - Baseline Host Computer Platform Configurations previously used by "tsWxGTUI_PyVx" Toolkit developers	<ol style="list-style-type: none">1 Linux (Fedora 15-16) with Python 2.6, 2.7 and 3.2 running on Apple MacBook Pro - 32-/64-bit Intel Core 2 Duo Processors running with 4 GB RAM Linux Virtual Machine created and managed by Parallels Desktop 6 for Mac2 Linux (openSuSE 11) with Python 2.6, 2.7 and 3.2 running on Apple MacBook Pro - 32-/64-bit Intel Core 2 Duo Processors with 4 GB RAM running Virtual Machine created and managed by Parallels Desktop 6 for Mac3 Linux (Ubuntu 10.04) with Python 2.7 and 3.2 running on Apple MacBook Pro - 32-/64-bit Intel Core 2 Duo Processors with 4 GB RAM running Linux Virtual Machine created and managed by Parallels Desktop 6 for Mac4 Mac OS X (Tiger 10.4.0-Snow Leopard 10.6.8) with Python 2.6, 2.7 and 3.2 running on Apple MacBook Pro - 32-/64-bit Intel Core 2 Duo Processors with 4 GB RAM running Mac OS 10.4-10.6.5 Windows (XP-SP3) with UNIX-like Cygwin plug-in and Python 2.6 running on Dell Laptop - 32-bit Intel Pentium 2 Processor with 384 MB RAM running Windows XP-SP36 Windows (XP-SP3 and Release 8 Preview) with UNIX-like Cygwin plug-in and Python 2.6 running on Apple MacBook Pro - 32-/64-bit Intel Core 2 Duo Processors with 4 GB RAM running Windows Virtual Machine created and managed by Parallels Desktop 6 for Mac
---	---

Notes - Experimental Host Computer Platform Configurations previously used by "tsWxGTUI_PyVx" Toolkit developers	<p>1 Windows (7 Professional) with built-in "Command Prompt" accessory shell and Python 2.7 running on Apple MacBook Pro - 32-/64-bit Intel Core 2 Duo Processors with 4 GB RAM running Windows Virtual Machine created and managed by Parallels Desktop (8) for Mac.</p> <p>This configuration uses the Windows built-in "Command Prompt" accessory shell which can run Command Line Interface components ("tsLibCLI", "tsTestsCLI" and "tsToolsCLI") of the "tsWxGTUI_PyVx" toolkit.</p> <p>It does NOT support the low-level, "nCurses" library, a pre-requisite for running the Graphical-style User Interface components ("tsLibGUI", "tsTestsGUI" and "tsToolsGUI") of the "tsWxGTUI_PyVx" toolkit.</p>
	<p>2 Windows (7 Professional) with UNIX-like Git Bash plug-in and Python 2.7 running on Apple MacBook Pro - 32-/64-bit Intel Core 2 Duo Processors with 4 GB RAM running Windows Virtual Machine created and managed by Parallels Desktop (8) for Mac.</p> <p>This configuration, like those using Cygwin, includes a Bash shell which can run Command Line Interface components ("tsLibCLI", "tsTestsCLI" and "tsToolsCLI") of the "tsWxGTUI_PyVx" toolkit.</p> <p>Unlike those configurations using Cygwin, it does NOT support the low-level, "nCurses" library, a pre-requisite for running the Graphical-style User Interface components ("tsLibGUI", "tsTestsGUI" and "tsToolsGUI") of the "tsWxGTUI_PyVx" toolkit.</p>
	<p>3 Windows (7 Professional) and Python 2.7 with the GNUwin32 and PDCurses Version 2.6 plug-ins running on Apple MacBook Pro - 32-/64-bit Intel Core 2 Duo Processors with 4 GB RAM running Windows Virtual Machine created and managed by Parallels Desktop (8) for Mac.</p> <p>This configuration, unlike those using Cygwin, includes a DOS-like Command Prompt shell which can run Command Line Interface components ("tsLibCLI", "tsTestsCLI" and "tsToolsCLI") of the "tsWxGTUI_PyVx" toolkit.</p> <p>Like those configurations using Cygwin, PDCurses Version 2.6 does support the low-level, Python "Curses" library, a pre-requisite for running the Graphical-style User Interface components ("tsLibGUI", "tsTestsGUI" and "tsToolsGUI") of the "tsWxGTUI_PyVx" toolkit. It runs the test_PDCurses (renamed test_tsWxCurses) application. However, PDCurses Version 2.6 traps because it lacks the mouse button definitions needed by the "tsWxGraphicalTextUserInterface" module to emulate "wxPython" in order to run such applications as "test_tsWxWidgets.py".</p>

1.2.3 Designing Embedded Systems with Linux and Python

See *Designing Embedded Systems with Linux and Python*
(<http://www.youtube.com/watch?v=WZoeqnsY9AY>)

Published on Mar 12, 2012

Mark Kohler

"The continual decrease in the cost of computer hardware is allowing more embedded systems to be built with Linux and Python, instead of the traditional approach of a real-time operating system and C...."

Transcript:

"0:00%ah thanks welcome
0:04who wants to build an embedded system let's use paid time to build in
medicine
0:09system
0:09okay in the right place the story so far
0:14hardware is cheap specifically the price difference between a bit
0:19and 32-bit processors a small and dwindling I miss them more and more
0:24places we can use Python
0:25that board has a 700 megahertz processor
0:29GPA you internet USB 256 megs of RAM
0:33as perfect for running Python his car Raspberry Pi
0:37i cost \$35 dollars my premise is the programmers develop faster
0:42with Linux in Python been a real time embedded operating system and see
0:46the tools and documentation a much better
0:49even when the documentation is the source just considering Python capable
0:55systems hardware resources can vary widely and when to use three examples
0:59for the stock
1:00electronic kiosks routers
1:04and thermostats with one problem
1:08Linux distributions are designed for desktops
1:12not embedded systems the solution is
1:16is the piece as you can from a Linux distribution and use Python
1:19to build the rest the first step is to choose the right distribution for
your
1:25product
1:25kiosks usually have PC class hardware
1:29ice cream next 36 processor a touchscreen a pointing device
1:34and tens of gigabytes of storage but there are some important differences
1:39a desktop is one user and lots about locations
1:42a kiosk is one application and lots of users
1:45KS have no human administrator our configuration and upgrades must be
1:50automatic
1:51but was great support for PC hardware
1:54and gives you a number of choices for building your user interface you can
use
1:59a traditional toolkit
2:00like you tear GTK what you want to build your interface the web way
2:04you can use the kiosk mode a Firefox or chromium
2:10for door also has great PC hardware support and could be a good choice

2:14Ubuntu's advantages there is based on Debian and the Debian port project
2:18has packaged a lot more softer than for Dora so is more likely that the
software
2:22you want
2:23is already packaged further Debian packages a more granular than for Dora
2:27packages
2:28so if you're tight on space is a lot easier to slim down your storage
2:32requirements
2:33by removing packages then by digging into the individual packages
2:36and tearing out where you don't need Devin becomes a better choice than
2:41Ubuntu
2:42if you don't need to support the latest PC hardware and desktop
environments
2:46for example if you don't need a compositing window manager
2:50Debian may be a better choice the Corbin two components
2:54but the colonel and the userspace runtime: code are always on the leading
2:57edge
2:58but I can mean those components on his portable as the core components a
Debian
3:02this is a charred
3:05up the Debian other CPU are architectures Devin supports
3:09I don't expect you to read it but it by contrast a boon to support x86
3:14and one flavor a farm and if you need to swing at your system down further
3:20them Debian project is a set a bill tools that allow you to create your
own
3:24Debian based distribution
3:25with only the parts you need this can save you a tremendous amount of time
for
3:30manually trying to find packages
3:32or parts a packages the you can throw out for thermostat
3:37the hardware design must be low-cost nevertheless
3:40inexpensive system-on-chip designs mean a python is still feasible
3:44the challenge of these designs is often related to minimizing the amount
of
3:49secondary storage
3:50for this kinda system you'll probably want to start with the kernel
3:53and add pieces instep starting with the distribution
3:56and removing pieces for the smallest embedded systems
4:04busybox is a good choice busybox is an implementation at the core Linux
4:08utilities
4:09made to be as small as possible people often use it with the custom build
4:13kernel
4:13a new lip a scaled-down C library to create very small custom
distributions
4:21how does the Linux boot process work how do you build software for non x86
4:25processors
4:27how do I use a difference e-library how do I make a custom Linux
distribution
4:32Linux from scratch in its companion cross-links from scratch
4:36a book based Linux distributions explain issues like
4:39creating and using a cross compilation toolchain
4:42building custom kernels using unusual bootloaders
4:46and customizable process their excellent resources for anyone who wants to
build
4:51embedded systems with Linux
4:55if you're building from scratch you'll know exactly how your system works

4:59but if using a desktop Linux distribution the next step will be to
5:03find all the software that assumes it is running on a desktop
5:06and remove it replace it or coat around it with Python
5:10this can be a significant amount of work but it helps a lot of you know
where to
5:13look
5:15are now talk about several areas have system software which frequently
need
5:18modifications
5:19from better systems for start-up talk about three ways
5:24to do automatic upgrades
5:31this is ideal for small systems that run of Avram
5:34instead of of a disk or flash
5:38the first step is the running system download the script
5:42the script downloads a file system image and right it's a secondary
storage
5:46you reboot and the new images loaded from secondary storage to ram
5:51and Europe is done you might have a configuration partition you need to
deal
5:56with her migrate some local settings forward
5:58but that's the gist of course this approach works for other archive
formats
6:02like tar balls
6:07for systems that run of secondary storage a more complicated approach is
6:11needed
6:12you have two partitions through each contain assist the complete system
6:16you boot from one partition upgrade the other and then toggle the
bootloader
6:20so that on the Next Food you boot the upgraded partition
6:24this is called ping pong because every time you upgrade the bootloader
toggles
6:28which partition is booted
6:30downside to this approach are that you need enough space for two system
images
6:34on secondary storage
6:35and you need to be able to download a complete system image
6:41apt is Devin's package tour and in many ways what makes Debian
6:45Debian the advantage abusing this this approach and it's a big one
6:49is that you only download the packages that need upgrading instead of a
6:52complete system image
6:54but using abs for upgrading embedded systems is complicated because of
6:57inherent desktops
6:58assumptions about Debian packaging
7:02our scripts if they don't know what to do can appeal to a human for help
7:06be the user interface of the upgrade application
7:10the theory behind Debian configuration is that the human administrator is
7:13responsible for configuration
7:15and only with the administrator's approval can packages change the
7:19configuration
7:21this is great if you don't you're operating system upgrade to Munger
7:24carefully constructed Apache configuration
7:27but this approach causes problems for systems that don't happen
7:30human administrator some other disadvantages to using
7:35apt and in bed system is that the Debian packages don't support downgrade
7:39downgrades and each package has its own set of scripts
7:43and both the scripts from the old and the new software will run during an

7:46upgrade
7:46upgrade this needs careful testing
7:51the good news is that you can customize an automated upgrade process using the
7:54Python
7:54apt library and you can pre answer questions that you know packaging
7:58scripts will ask
8:02on a big system it can be difficult to difficult to foresee which software you
8:07will need to upgrade
8:09for that reason you need to you want to make sure the crusty obsolete software
8:12that you're upgrading
8:13does not control the upgrade process you want to do the minimum
8:18pull down some new bits and run them in now way
8:22you can ensure that the new softer you're upgrading to controls the upgrade
8:25process
8:26an ounce after can be written so that it can upgrade whatever pieces it needs to
8:30even if that means figuring out how to patch the bootloaders file system driver
8:37so it picked our distribution we have a plan for upgrades
8:41what other system software do need to think about
8:51time is hard dates
8:54are difficult the Warri
8:58code related to dates and times this is an area notorious for bugs
9:04but is especially true when you're adapting desktop code to an embedded
9:07system
9:08our talk about some of these problems avoid time zones as possible
9:14wristwatches do first there are a huge number of time zones
9:19just creating the user interface for specifying a time zone is not trivial
9:23second governments change the rules regarding time zones frequently
9:27and often with little notice embedded systems neither
9:31either need to have a reliable timely source of time zone information
9:36where the shooting night time zones completely GPS time does not use leap
9:43seconds
9:44the kind of said between GPs time in UTC
9:47is fifteen seconds but that of that will change over
9:51time
9:59this is almost rape in Linux
10:01elapsed time has no value Linux does not have direct support for elapsed time
10:07applications get the system time and do their own calculations
10:11applications expect to be able to do this
10:19and now often work but not always
10:23the problem is in Linux
10:31time is implemented as a single global counter and a privilege process
10:35can set the time whenever it wants depending on how much the time is
10:39adjusted
10:39different assumptions about time break for Justin's under half a second
10:45time is sloughed the amount of time in a second
10:49is me longer were shorter for time
10:52for adjustments have more than half a second
10:57system time immediately jumps forward or back this affects timeouts
11:01and crime jobs why are all the hourly jobs running at once

11:05because time jumped forward in our time can be quite of depending on whether you

11:10have been taking good care of the hardware clock

11:12and many Linux systems the hardware clock is only set on shutdown

11:16if you should if you shut down on cleanly

11:19the clock doesn't get set and you have a big time jump in your future

11:23be very careful about setting the time don't manage network manager

11:33desktop systems his network manager to control setup network interfaces

11:38choosing the best one is connectivity changes over time

11:41this approach may work if your embedded system roams across networks

11:45phones eBook readers tablets over many embedded systems

11:50routers thermostats set-top boxes

11:53act more like service doesn't mean clients is not appropriate to bring up

11:57and down network interfaces

11:58every time there's a brief conductivity outage for the system's

12:02the older I F up/down interface from Debian may be a lot more appropriate

12:06right or wrong if you're using Ubuntu or Debian

12:11you have inherited the desktop security model as a result

12:15as you are automating away configuration changes you'll need group religious

12:19Soo Do can be configured to automatically give root privileges

12:23to a select list of commands this is often the easiest way to handle the

12:27privilege escalation is required

12:29when you're on a meeting changes in configuration to have picture

12:35distribution

12:37we've thought about how to avoid the assumptions on the desktop

12:40now we need to write some code I wanna talk about software design issue this

12:44particularly prevalent

12:45in embedded system software is going to be the most abstract part my talk

12:50so to talk about something country I want to ask everybody who ex-prime works

12:54meh who prefers libraries

12:58okay welcome back to this

13:01I want everyone to rate portable code in Python

13:07quick some Python code automatically portable

13:10I haven't seen near and far pointers I'm in San

13:14Indian macros in Python code haven't seen 36 bit words

13:18to Python's portable across processors but there's more than one type of

13:22portability

13:22the kind of portability i'm talking about. is creating a single store space

13:27for a range of products

13:29which is very common embedded systems a common core functionality

13:33both somewhat during features and a variety of hardware

13:36and the feature side maybe there's a basic model in a professional model

13:40the hardware may have different sensors different network interfaces even

13:43different user interfaces

13:45this kind of portability is about keeping the code maintainable

13:49as you add support for new hardware platforms an independent feature sets

13:53the naive way to accommodate a range of products

13:58in a single source paces the if statement just write your code

14:02and when you get to a difference between the products you write something like

14:05this slide

14:07that works so what's the problem

14:11the promise that everyplace were something product depending happens
14:15your core code must have knowledge of the behavior of every product
14:19this is not maintainable we can do much better
14:25would really like to be able to work on the core code without seeing the
14:28complexity
14:29about the product price an obvious first step is to move the product
specific
14:34code
14:35in two separate modules
14:43this is a little bit better the model specific code has been moved into
14:47model-specific modulus with contain the complexity a bit
14:51but this time with court has problems the common code is dependent on the
14:56model of the model specific code
14:58this kind of dependency fan out makes testing difficult
15:01as we add products this pattern will be repeated in more and more places
15:06never left was but nevertheless
15:10that approach can work as long as your products are similar enough
15:14that they all use the same interface but it can be a bit of a trek to
find that
15:18interface
15:19and the interface may need to change over time which can produce a lot of
15:23churn
15:23in your product specific code so why is this so hard
15:27what are we doing wrong let's look at the code again
15:31the problem is we have core code
15:35depending on the details of our products if we want to have a stable
interface
15:39New York 02 depend on the commonalities not the differences
15:43in other words we need to turn the code
15:46inside out we need the product specific code
15:50to depend on the core code
16:00here we have the product Marshall calling into the core code
16:03we avoid the problem trying to find a common interface for all of our
products
16:07by inverting the call graph and making the core code be the interface
16:11another way to think about this is we have taken our core code
16:16our application and turned it from a framework
16:19into a library as a result
16:22now all of our ugly product-specific and hardware specific code is him
16:26finally we can rate core code is completely ignorant
16:31at these details why does this work
16:34the trick is by turning the code inside out
16:38we found a way to hide the right thing the key to writing modular code
16:43is figuring out what you should hide if you're having trouble finding
16:47designing a modular interface focus on what you need to hide
16:51when writing embedded system software the trick is usually to hide the
details
16:56of your hardware
16:57a good way to do that is to turn the core of your application
17:01from a framework into a library choose your distribution's
17:07avoid assumptions on the desktop and reportable Co
17:10these ideas about
17:15structuring code for portability are not mine I store them
17:19if you wanna steal them to read these papers is clinics where you can
17:26and use Python for the rest rate better embedded systems faster
17:29i'm happy to answer questions %ah

17:38 I know the stock was specifically about Linux but have you
17:43 experience with FreeBSD as the platform and what are your thoughts on a
17:48 I'm I don't have personal experience with three BST now
18:02 okay thank you user %uh may have another question
18:06 and on thanks those are interesting
18:13 arm I was 1 I'm you brought out the issue of time
18:17 are moving around and arm here II guess
18:21 I haven't had the experience love my global time being changed on me
18:26 randomly arm can you perhaps
18:29 talk about one that sort of thing happens and
18:32 yeah on to think the the time when it can take you by surprise
18:37 is if you have um a product that I
18:41 determinedly connects to the network I'm on a boon to at least when you
connect
18:45 to a network to the Internet
18:46 you automatically set the time for you secure not expecting that
18:49 um me quite a surprise yep
18:55 to have a question
18:58 here I haha om
19:02 I I know you specifically talked about Don went to end and Damian
19:07 arm and and I think you were dob kinda
19:11 concentrating more on long-run systems and and you're talking about
19:15 on a meaning the the admin tasks but you have to have you had experience
with job
19:20 like smaller implement a bold
19:23 distributions like and strong or the open in bed distributions
19:26 I haven't the last time I am
19:30 the last time I looked animal was there was inappropriate for my product
and so
19:33 on
19:34 it I'm I can I can speak to them okay
19:49 and is there any other question we still have
19:53 maybe five more minutes
20:03 so
20:04 so one other things that you mentioned was um
20:07 portability between products arm have you considered
20:11 testing products for capability
20:14 instead of just for polity verses part B so
20:18 for instance you could say like does this products have a camera if so
then
20:23 you know do something sure sure
20:28 on I I think detecting
20:32 when it when you can detect capabilities detect hardware that often is
better
20:36 than trying to configure it but
20:37 you still run into the same problems have having code that needs to be
able
20:41 to deal with
20:41 not having a camera or having a camera for instance
20:53 hi did you experiment with Gentoo
20:56 it's almost leads from scratch no I haven't that some
21:00 that would be another interesting approach to try didn't know they
support
21:04 a lot of
21:05 I'm different processors because they
21:09 they only have package manager only has combine receive peace


```
21:12basically so they can spot pretty much in Zeta diseases porch
21:15mmm-hmm but okay thanks yep
21:27hi I you mentioned about using Debbie ins
21:30ap system get home do you have any experience urges you to around with
the
21:36ap build package in Debian
21:40its serve a long same lines for Jen to wear
21:43app downloads a source in and compiles it would blow ever flags
optimizations
21:47you want I haven't used up till now
21:50okay thanks yes
21:57and given experience on power consumption running Python an embedded
22:01and how it compares to something else nope
22:05a and I were to with them
22:10systems the run out power remains for this
22:209 I think then it's gonna be us
22:24almost time so thank you very much %ah
22:30thank you on"
```

1.2.3.1 Embedded Python

From *Embedded Python* (<https://wiki.python.org/moin/EmbeddedPython>)

"Python can be used in embedded, small or minimal hardware devices, depending on how limiting the devices actually are.

Devices capable of running CPython

Some modern embedded devices have enough memory and a fast enough CPU to run a typical Linux-based environment, for example, and running CPython on such devices is mostly a matter of compilation (or cross-compilation) and tuning. which could be considered as "embedded" by modern standards and which can run tuned versions of CPython include the following:

- Gumstix
- Raspberry Pi
- BeagleBone Black
- FIC Neo1973 and Neo FreeRunner (Python on Openmoko)
- Telit GSM/GPRS modules (also available as AarLogic family GPRS/GPS QUAD Band Modules)

See also PythonForArmLinux and OpenEmbedded."

Draft

1.3 APPENDIX B - API RELATIONSHIP

An Application Programming Interface (API) is a source code based specification intended to be used as an interface by software components to communicate with each other. An API may include specifications for routines, data structures, object classes, and variables.

- 1 High Level API** - A programming interface that is the least detailed but provides more functionality within one statement than a lower-level interface. High-level interfaces are designed to enable the programmer to write code in a shorter amount of time and to be less involved with the details of the software module or hardware device that is performing the required services. For example, the programmer can invoke a high level procedure to append one or more text strings to an internal buffer.
- 2 Low Level API** - A programming interface that is the most detailed, allowing the programmer to manipulate functions within a software module or within hardware at a very granular level. To continue with the afore mentioned example, a high level procedure can then invoke one or more low level procedures that will sequentially get one text string at a time from the internal buffer, sequentially scroll the top most string off the display, then scroll up each remaining displayed string and finally outputting the new string to the bottom row of the screen.
- 3 Extended API** - A customized version of an existing programming interface that supports additional keyword value pairs and positional arguments. For example, the "tsWxGTUI_PyVx" Toolkit is a character-mode emulation of the pixel-mode "wxPython" API. It internally may require optional parameters to distinguish such features as character-mode dimensions from their pixel-mode counterparts. It may also include functionality that "wxPython" and its underlying "wxWidgets" toolkit otherwise obtain from the host operating system and its programming libraries, such as the installation of the color palette and the implementation of scrollable windows.

1.3.1 Comparison with "wxPython"

This tabulation shall briefly compare the features, capabilities and limitations of the pixel-mode "wxPython" / "wxWidgets" / "wxEmbedded" Toolkit with those of its character-mode emulator, the TeamSTARS "tsWxGTUI_PyVx" Toolkit.

Feature	"wxPython" / "wxWidgets" / "wxEmbedded"	TeamSTARS "tsWxGTUI_PyVx" Toolkit
Platform	Locally (via system console or xterm) and remotely (via vnc) accessible systems: <ul style="list-style-type: none"> ▪ Desktop ▪ Laptop ▪ Workstation ▪ Embedded (via KOAN's "wxEmbedded") 	Locally (via system console or xterm) and remotely (via xterm) accessible systems: <ul style="list-style-type: none"> ▪ Desktop ▪ Laptop ▪ Workstation ▪ Embedded
Operator Desktop Interface Hardware	<ul style="list-style-type: none"> ▪ Keyboard ▪ Mouse, trackball, touchpad or touchscreen ▪ Optional Mouseless 	<ul style="list-style-type: none"> ▪ Keyboard ▪ Mouse, trackball, touchpad or touchscreen ▪ Keyboard Shortcut Key and/or Optional Mouse (Future)

Feature	"wxPython" / "wxWidgets" / "wxEmbedded"	TeamSTARS "tsWxGTUI_PyVx" Toolkit
Platform	<p>Locally (via system console or xterm) and remotely (via vnc) accessible systems:</p> <ul style="list-style-type: none"> ▪ Desktop ▪ Laptop ▪ Workstation ▪ Embedded (via KOAN's "wxEmbedded") 	<p>Locally (via system console or xterm) and remotely (via xterm) accessible systems:</p> <ul style="list-style-type: none"> ▪ Desktop ▪ Laptop ▪ Workstation ▪ Embedded
	<ul style="list-style-type: none"> ▪ Display (68-color pixel mode) 	<ul style="list-style-type: none"> ▪ Display (68-color character mode mapped into: 8-color/64-color pair on cygwin, linux, xterm or xterm-color terminals) ▪ Display (71-color character mode mapped into: 16-color/256-color pair on xterm-16color, xterm-88color or xterm-256color terminals) ▪ Display (a single shade of white, green or orange that is "ON" depending on the single available phosphor or black when "OFF" on vt100/vt220 terminals)
Operating System & Device Driver Software	<ul style="list-style-type: none"> ▪ GNU/Linux (Linux Operating System Kernel with Unix-like GNU Command Line Interface, Graphical User Interface and Toolchain). ▪ Mac OS X (Darwin-/BSD-based Unix Operating System Kernel with GNU Command Line Interface, Toolchain and Apple Computer's Graphical User Interface and Toolchain). ▪ Microsoft Windows (with free add-on of POSIX-like Cygwin Command Line Interface and GNU tools from Red Hat) ▪ Unix (Operating System Kernel with Unix-like Command Line Interface, Graphical User Interface and Toolchain) 	<ul style="list-style-type: none"> ▪ GNU/Linux (Linux Operating System Kernel with Unix-like GNU Command Line Interface, Graphical User Interface and Toolchain). ▪ Mac OS X (Darwin-/BSD-based Unix Operating System Kernel with GNU Command Line Interface, Toolchain and Apple Computer's Graphical User Interface and Toolchain). ▪ Microsoft Windows (with free add-on of POSIX-like Cygwin Command Line Interface and GNU tools from Red Hat) ▪ Unix (Operating System Kernel with Unix-like Command Line Interface, Graphical User Interface and Toolchain)
Terminal Device Interface Software	<ul style="list-style-type: none"> ▪ "wxWidgets" internally uses host Operating System specific API for its Terminal Device Interface. ▪ It translates host Operating System specific events into "wxWidget" specific published API events. ▪ Events include keyboard key press / release, mouse button press / release and single / double click with mouse position at every clocked sample. ▪ Events also include window resizing. 	<ul style="list-style-type: none"> ▪ "tsWxGTUI_PyVx" internally uses Python "Curses" module specific API for its Terminal Device Interface. ▪ Python "Curses" module internally uses host Operating System specific "Curses" and/or "nCurses" API for its Terminal Device Interface. ▪ It translates "Curses" module specific events into published "wxWidget" specific API events. ▪ Events include keyboard key press / release, mouse button press / release and single / double / triple click with mouse position ONLY available at time of button state change.

Feature	"wxPython" / "wxWidgets" / "wxEmbedded"	TeamSTARS "tsWxGTUI_PyVx" Toolkit
Platform	<p>Locally (via system console or xterm) and remotely (via vnc) accessible systems:</p> <ul style="list-style-type: none"> ▪ Desktop ▪ Laptop ▪ Workstation ▪ Embedded (via KOAN's "wxEmbedded") 	<p>Locally (via system console or xterm) and remotely (via xterm) accessible systems:</p> <ul style="list-style-type: none"> ▪ Desktop ▪ Laptop ▪ Workstation ▪ Embedded
		<ul style="list-style-type: none"> ▪ Events also include window resizing. However, event handling is currently limited to trapping the unsupported event. Though re-initializing "Curses" to detect and use the new size is feasible and fast (50 milliseconds or more depending on host processor and terminal color palette (and associated definition of or mapping of wxPython color palette), it does not address the time consuming (20 seconds or more on host processor) complexities associated with re-initializing the "wxPython" emulation and the System Operator designated application program.
Programming Language	<ul style="list-style-type: none"> ▪ "wxWidgets" is implemented in C++ ▪ "wxPython" binding/wrapper for "wxWidgets" is implemented with SWIG in Python 2.x and Python 3.x 	<ul style="list-style-type: none"> ▪ Primary baseline "tsWxGTUI_Py2x" is implemented in Python 2.x and then debugged. ▪ Secondary baseline "tsWxGTUI_Py3x" is implemented in Python 3.x as a "port" from "tsWxGTUI_Py2x" via the standard Python "2to3" translation utility followed by minor manual debugging when appropriate
Terminal Interface Software	<ul style="list-style-type: none"> ▪ "wxWidgets" internally uses Operating System or X window system specific API for Graphical User Interface. 	<ul style="list-style-type: none"> ▪ "tsWxGTUI_PyVx" internally uses Python Curses module API for Graphical-style User Interface. ▪ The Python Curses module only implements the most commonly used subset features of the host "Curses" or "nCurses" API. ▪ The "tsWxGTUI_PyVx" Toolkit emulates a typical Operating System or X window system specific API for Graphical User Interface. It establishes the appropriate relationship between a triggering event (mouse button click) and the GUI object (button, gauge, frame, dialog, panel etc.) to receive and subsequently handle the triggering event notification.

Feature	"wxPython" / "wxWidgets" / "wxEmbedded"	TeamSTARS "tsWxGTUI_PyVx" Toolkit
Platform	<p>Locally (via system console or xterm) and remotely (via vnc) accessible systems:</p> <ul style="list-style-type: none"> ▪ Desktop ▪ Laptop ▪ Workstation ▪ Embedded (via KOAN's "wxEmbedded") 	<p>Locally (via system console or xterm) and remotely (via xterm) accessible systems:</p> <ul style="list-style-type: none"> ▪ Desktop ▪ Laptop ▪ Workstation ▪ Embedded
Operator Desktop Interface Software	<ul style="list-style-type: none"> ▪ Host Operating System specific Command Line Interface is a POSIX compatible shell (such as bash) window process ▪ Host Operating System specific Graphical User Interface features support multiple independently re-sizable and repositionable Frame and Dialog processes ▪ Host Operating System specific Graphical User Interface features reflect proprietary or industry standard placement of labels and buttons to iconize, maximize/restore and close frames and dialogs ▪ Host Operating System specific Graphical User Interface task bar features support the closing of individual Frames and Dialogs ▪ Host Operating System specific task bar features support the shifting of foreground focus from one Frame or Dialog to another 	<ul style="list-style-type: none"> ▪ Host Operating System specific Command Line Interface is a POSIX compatible shell (such as bash) window process ▪ Host Operating System specific Graphical User Interface utilizes the existing Command Line Interface shell window process ▪ "tsWxGTUI_PyVx" emulated Graphical User Interface features DO NOT support multiple independently re-sizable and repositionable Frames and Dialogs WITHOUT the System Operator first creating separate Command Line Interface shell window processes ▪ "tsWxGTUI_PyVx" emulated Graphical User Interface features reflect Microsoft Windows-style placement of labels and buttons to iconize, maximize/restore and close frames and dialogs ▪ "tsWxGTUI_PyVx" emulated Graphical User Interface task bar features DO NOT currently support the closing of individual Frames and Dialogs ▪ "tsWxGTUI_PyVx" Toolkit task bar features (future enhancement) support the shifting of foreground focus from one Frame or Dialog to another
Application Programming Interface	<ul style="list-style-type: none"> ▪ "wxWidgets" / "wxPython" API for Graphical User Interface supports bit-mapped images. ▪ It supports fixed and variable sized fonts and at least the 68 most commonly used colors. 	<ul style="list-style-type: none"> ▪ "tsWxGTUI_PyVx" emulated subset of "wxWidgets" / "wxPython" API for Graphical-style User Interface DOES NOT support bitmapped images except for the single, predefined bitmapped image used as a splash screen at startup ▪ It supports a single fixed sized font and at least the 68 most commonly used colors (which are internally mapped into the "Curses" standard 8-color or 16-color terminal hardware and terminal emulator software). Future "Curses" enhancements may enable support a single fixed sized font and at least the 68 most commonly used colors defined for optional 88-color and 256-color terminal hardware and terminal

Feature	"wxPython" / "wxWidgets" / "wxEmbedded"	TeamSTARS "tsWxGTUI_PyVx" Toolkit
Platform	Locally (via system console or xterm) and remotely (via vnc) accessible systems: <ul style="list-style-type: none"> ▪ Desktop ▪ Laptop ▪ Workstation ▪ Embedded (via KOAN's "wxEmbedded") 	Locally (via system console or xterm) and remotely (via xterm) accessible systems: <ul style="list-style-type: none"> ▪ Desktop ▪ Laptop ▪ Workstation ▪ Embedded
		emulator software. <ul style="list-style-type: none"> ▪ It supports the 1-color (white, green or orange depending on the single available phosphor) that is "ON" or "OFF" (black) associated with a VT100/VT220 Terminal emulator.

1.3.2 High, Low and Extended API Relationships

The following tables describes the conceptual purpose, scope, capabilities, limitations and relationship between the High-level, Low-Level and Extended APIs associated with the "tsWxGTUI_PyVx" Toolkit and its third-party components.

Low Level GUI "Curses"-style API Classes and associated Class Methods:

- **Pads** - A pad is like a window, except that it is not restricted by the screen size, and is not necessarily associated with a particular part of the screen. Pads can be used when a large window is needed, and only a part of the window will be on the screen at one time. Automatic refreshes of pads (such as from scrolling or echoing of input) do not occur. The refresh() and noutrefresh() methods of a pad require 6 arguments to specify the part of the pad to be displayed and the location on the screen to be used for the display. The arguments are pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol; the p arguments refer to the upper left corner of the pad region to be displayed and the s arguments define a clipping box on the screen within which the pad region is to be displayed.
- **Panels** - Panels are windows with the added feature of depth, so they can be stacked on top of each other, and only the visible portions of each window will be displayed. Panels can be added, moved up or down in the stack, and removed.
- **Windows** - a window is a rectangular area of the display with or without a border
- **Text** - A character string of one or more mono-spaced alpha-numeric, punctuation or line-draw characters.
- **Colors** - Support for terminals and terminal emulators (cygwin, linux, xterm, xterm-color, xterm-16color, xterm-88color or xterm-256color) with mouse and Red-Green-Blue color phosphors who's individual intensities can be adjusted to create a palette of terminal / terminal emulator dependent colors, ranging from 8 to 256 used to create foreground/background color pair combinations ranging from 8 x 8 to 256 x 256. However, currently the maximum number of color pairs is limited to 16 x 16.
- **Non-color** - Support for Digital Equipment Corporation VT100 and VT220 terminals and terminal emulators with/without mouse having only a single color phosphor (such as green orange or white) who's intensity can be either ON or OFF.

High Level API	Low Level API	Extended API
<p>Local Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell launch local application terminate local application terminate local shell logout of local platform host 		<p>Local & Remote Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell echo \$TERM # Display default type \$TERM=<xterm-family member ID> or <vt100-family member ID> echo \$TERM # Display changed type launch local application terminate local application login to remote platform host via "ssh <login ID>@<Host ID>:" launch remote application terminate remote application logout of remote platform host transfer file(s) from remote platform host via "sftp <login ID>@<Host ID>:<File IDs>" terminate local shell logout of local platform host
	<p>Local Host Operating System Process API</p> <ul style="list-style-type: none"> Process Launcher Process Terminator Process Cleanup Process Event Handlers Process Input/Output Handlers 	<p>Local & Remote Host Operating System Process API</p> <ul style="list-style-type: none"> Process Launcher Process Terminator Process Cleanup Process Event Handlers Process Input/Output Handlers
		<p>Command Line Interface Low Level API (ts)</p> <ul style="list-style-type: none"> tsApplication (Class and run time library component initializes and configures the application using the following keyword-value pairs and positional arguments: input provided on the command line by an operator and input provided in the parameter list of the application's invocation of the class instantiation.) tsCommandLineEnv (Class and run time library component provides platform independent configuration,

High Level API	Low Level API	Extended API
<p>Local Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell launch local application terminate local application terminate local shell logout of local platform host 		<p>Local & Remote Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell echo \$TERM # Display default type \$TERM=<xterm-family member ID> or <vt100-family member ID> echo \$TERM # Display changed type launch local application terminate local application login to remote platform host via "ssh <login ID>@<Host ID>:" launch remote application terminate remote application logout of remote platform host transfer file(s) from remote platform host via "sftp <login ID>@<Host ID>:<File IDs>" terminate local shell logout of local platform host
		<p>initialization, input/output supervisor and application launcher, event handler and terminator)</p> <ul style="list-style-type: none"> tsCommandLineInterface (Class and methods that prompt or re-prompt the operator for input, validate that the operator has supplied the expected number of inputs and that each is of the expected type.) tsCxGlobals (Module to establish configuration constants and macro-type functions for the Command Line Interface mode of the "tsWxGTUI_PyVx" Toolkit.) tsDoubleLinkedList (Class to establish a representation of a linked list with forward and backward pointers.) tsExceptions (Class to define and handle error exceptions. Maps run time exception types into 8-bit exit codes and prints associated diagnostic message and traceback info.) tsLogger (Class that emulates a subset of Python logging API. It defines and

High Level API	Low Level API	Extended API
<p>Local Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell launch local application terminate local application terminate local shell logout of local platform host 		<p>Local & Remote Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell echo \$TERM # Display default type \$TERM=<xterm-family member ID> or <vt100-family member ID> echo \$TERM # Display changed type launch local application terminate local application login to remote platform host via "ssh <login ID>@<Host ID>:" launch remote application terminate remote application logout of remote platform host transfer file(s) from remote platform host via "sftp <login ID>@<Host ID>:<File IDs>" terminate local shell logout of local platform host
		<p>handles prioritized, time and date stamped event message formatting and output to files and devices. Files are organized in a date and time stamped directory named for the launched application. Unix-type devices include syslog, stderr, stdout and stdscr (the ncurses display screen). It also supports "wxPython"-style logging of assert and check case results.)</p> <ul style="list-style-type: none"> tsOperatorSettingsParser (Class to parse the command line entered by the operator of an application program. Parsing extracts the Keyword-Value pair Options and Positional Arguments that will configure and control the application during its execution.) tsPlatformRunTimeEnvironment (Class to capture current hardware, software and network information about the run time environment for the user process.) tsReportUtilities (Class defining methods used to format information:

High Level API	Low Level API	Extended API
<p>Local Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell launch local application terminate local application terminate local shell logout of local platform host 		<p>Local & Remote Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell echo \$TERM # Display default type \$TERM=<xterm-family member ID> or <vt100-family member ID> echo \$TERM # Display changed type launch local application terminate local application login to remote platform host via "ssh <login ID>@<Host ID>:" launch remote application terminate remote application logout of remote platform host transfer file(s) from remote platform host via "sftp <login ID>@<Host ID>:<File IDs>" terminate local shell logout of local platform host
		<p>date and time (begin, end and elapsed), file size (with kilo-, mega-, giga-, tera-, peta-, exa-, zeta- and yotta-byte units) and nested Python dictionaries.)</p> <ul style="list-style-type: none"> tsSysCommands (Class definition and methods for issuing shell commands to and receiving responses from the host operating system.)
<p>Graphical User Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host and its GUI Desktop launch local application from GUI Desktop terminate local application from GUI Desktop logout of local platform host and its GUI Desktop 		<p>Local & Remote Graphical User Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell echo \$TERM # Display default type \$TERM=<xterm-family member ID> or <vt100-family member ID> echo \$TERM # Display changed type launch local application terminate local application login to remote platform host via "ssh <login ID>@<Host ID>:" launch remote application

High Level API	Low Level API	Extended API
<p>Local Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell launch local application terminate local application terminate local shell logout of local platform host 		<p>Local & Remote Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell echo \$TERM # Display default type \$TERM=<xterm-family member ID> or <vt100-family member ID> echo \$TERM # Display changed type launch local application terminate local application login to remote platform host via "ssh <login ID>@<Host ID>:" launch remote application terminate remote application logout of remote platform host transfer file(s) from remote platform host via "sftp <login ID>@<Host ID>:<File IDs>" terminate local shell logout of local platform host
		<ul style="list-style-type: none"> terminate remote application logout of remote platform host transfer file(s) from remote platform host via "sftp <login ID>@<Host ID>:<File IDs>" terminate local shell logout of local platform host
	<p>Local Host Operating System Process API</p> <ul style="list-style-type: none"> Process Launcher Process Terminator Process Cleanup Process Event Handlers Process Input/Output Handlers 	<p>Local & Remote Host Operating System Process API</p> <ul style="list-style-type: none"> Process Launcher Process Terminator Process Cleanup Process Event Handlers Process Input/Output Handlers
<p>Graphical User Interface Application Launcher API (wxWidgets & wxPython)</p> <ul style="list-style-type: none"> run time library components provide platform specific configuration, initialization, input/output supervisor and application launcher 	<p>Graphical User Interface Application Launcher API (nCurses)</p> <ul style="list-style-type: none"> application run time library components provide platform specific nCurses configuration, initialization, input/output supervisor and application launcher 	<p>Graphical User Interface Application Launcher API (tsWxGTUI)</p> <ul style="list-style-type: none"> tsWx (Module to load all symbols that should appear within the wxPython.wx emulation namespace. Included are various classes, constants, functions and methods available for use by applications built

High Level API	Low Level API	Extended API
<p>Local Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell launch local application terminate local application terminate local shell logout of local platform host 		<p>Local & Remote Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell echo \$TERM # Display default type \$TERM=<xterm-family member ID> or <vt100-family member ID> echo \$TERM # Display changed type launch local application terminate local application login to remote platform host via "ssh <login ID>@<Host ID>:" launch remote application terminate remote application logout of remote platform host transfer file(s) from remote platform host via "sftp <login ID>@<Host ID>:<File IDs>" terminate local shell logout of local platform host
		<p>with components from the wxPython emulation infrastructure.)</p> <ul style="list-style-type: none"> tsWxGlobals (Module to establish configuration constants and macro-type functions for the Graphical-style User Interface mode of the "tsWxGTUI_PyVx" Toolkit. Provides a centralized mechanism for modifying/restoring those configuration constants that can be interogated at runtime by those software components having a "need-to-know". The intent being to avoid subsequent searches to locate and modify or restore a constant appropriate to the current configuration. Provides a theme-based mechanism for modifying / restoring those configuration constants.) tsWxMultiFrameEnv (Class to enable an application using a Command Line Interface (CLI) to launch and use the same character-mode terminal with a Graphical-style User Interface (GUI). It uses application specified configuration keyword-value pair

High Level API	Low Level API	Extended API
<p>Local Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell launch local application terminate local application terminate local shell logout of local platform host 		<p>Local & Remote Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell echo \$TERM # Display default type \$TERM=<xterm-family member ID> or <vt100-family member ID> echo \$TERM # Display changed type launch local application terminate local application login to remote platform host via "ssh <login ID>@<Host ID>:" launch remote application terminate remote application logout of remote platform host transfer file(s) from remote platform host via "sftp <login ID>@<Host ID>:<File IDs>" terminate local shell logout of local platform host
		<p>options to initialize any application specific logger(s) It wraps the CLI, underlying the GUI, and the GUI with exception handlers to control the exit codes and messages used to coordinate other application programs.)</p>
<p>Graphical User Interface High Level Application Launcher API (wxWidgets & wxPython)</p> <ul style="list-style-type: none"> PyApp (start wxPython application) PySimpleApp (start simple wxPython application) PyOnDemandOutput (start Redirected Output window) 		<p>Graphical User Interface High Level Application Launcher API (tsWxGTUI)</p> <ul style="list-style-type: none"> PyApp (start wxPython application) PySimpleApp (start simple wxPython application) PyOnDemandOutput (start Redirected Output window; enhancements include date, time and message severity level annotations with and without font style and foreground/background color markup attributes)

High Level API	Low Level API	Extended API
<p>Local Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell launch local application terminate local application terminate local shell logout of local platform host 		<p>Local & Remote Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell echo \$TERM # Display default type \$TERM=<xterm-family member ID> or <vt100-family member ID> echo \$TERM # Display changed type launch local application terminate local application login to remote platform host via "ssh <login ID>@<Host ID>:" launch remote application terminate remote application logout of remote platform host transfer file(s) from remote platform host via "sftp <login ID>@<Host ID>:<File IDs>" terminate local shell logout of local platform host
	<p>Graphical User Interface Low Level Launcher (nCurses)</p> <ul style="list-style-type: none"> start (curses.start) stop (curses.stop) 	<p>Graphical User Interface Low Level Launcher API (tsWxGTUI)</p> <ul style="list-style-type: none"> start (tsWxGTUI.start) stop (tsWxGTUI.stop) tsWxGraphicalTextUserInterface (Class uses the Standard Python Curses API to initialize, manage and shutdown input, from a keyboard and mouse, and output, to a two-dimensional display screen. It identifies user terminal make, model and features. It controls terminal device startup, shutdown and exception handling. It translates "wxPython"-style terminal color, pixel and character parameters into their "Curses" counterparts. Upon startup, it briefly restores or creates, saves or loads the splash screen bitmap image as specified in the "tsWxGlobals" configuration file.)
<p>GUI (wxWidgets & wxPython)</p> <ul style="list-style-type: none"> It is a C++ library that lets developers create applications for Windows, OS X, Linux 	<p>GUI (nCurses)</p> <ul style="list-style-type: none"> It is a programming library that provides an API which allows the programmer to write text mode 	<p>GUI (tsWxGTUI)</p> <ul style="list-style-type: none"> It is a Python and nCurses based programming library that lets developers create wxWidgets and

High Level API	Low Level API	Extended API
<p>Local Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell launch local application terminate local application terminate local shell logout of local platform host 		<p>Local & Remote Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell echo \$TERM # Display default type \$TERM=<xterm-family member ID> or <vt100-family member ID> echo \$TERM # Display changed type launch local application terminate local application login to remote platform host via "ssh <login ID>@<Host ID>:" launch remote application terminate remote application logout of remote platform host transfer file(s) from remote platform host via "sftp <login ID>@<Host ID>:<File IDs>" terminate local shell logout of local platform host
<p>and UNIX on 32-bit and 64-bit architectures as well as several mobile platforms including Windows Mobile, iPhone SDK and embedded GTK+.</p>	<p>user interfaces in a terminal independent manner.</p> <ul style="list-style-type: none"> It is a toolkit for developing "GUI-like" application software that runs under a terminal emulator. It also optimizes screen changes, in order to reduce the latency experienced when using remote shells. 	<p>wxPython style applications for Windows, OS X, Linux and UNIX on 32-bit and 64-bit architectures.</p> <ul style="list-style-type: none"> It emulates a subset of the wxWidgets and wxPython API that is suitable for text mode terminals. It requires the operator to preadjust the position, size and appearance of each command shell window, within the display, before running an application program.
<ul style="list-style-type: none"> It has popular language bindings for Python, Perl, Ruby and many other languages. 		
<ul style="list-style-type: none"> Unlike other cross-platform toolkits, wxWidgets gives its applications a truly native look and feel because it uses the platform's native API rather than emulating the GUI. 		
<ul style="list-style-type: none"> It's also extensive, free, open-source and mature. 		

High Level API	Low Level API	Extended API
<p>Local Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell launch local application terminate local application terminate local shell logout of local platform host 		<p>Local & Remote Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell echo \$TERM # Display default type \$TERM=<xterm-family member ID> or <vt100-family member ID> echo \$TERM # Display changed type launch local application terminate local application login to remote platform host via "ssh <login ID>@<Host ID>:" launch remote application terminate remote application logout of remote platform host transfer file(s) from remote platform host via "sftp <login ID>@<Host ID>:<File IDs>" terminate local shell logout of local platform host
<ul style="list-style-type: none"> It supports local terminals with various keyboard, mouse and pixel mode display device configurations. 	<ul style="list-style-type: none"> It supports local and remote terminals with various keyboard, mouse and text mode display device configurations. 	<ul style="list-style-type: none"> It supports local and remote terminals with various keyboard, mouse and text mode display device configurations.
<ul style="list-style-type: none"> It enables the operator to adjust the display position, size and appearance of the top level GUI objects. 	<ul style="list-style-type: none"> It requires the operator to preadjust the position, size and appearance of each command shell window, within the display, before running an application program. It requires the operator to login to each remote computer before running an application program. 	<ul style="list-style-type: none"> It requires the operator to login to each remote computer before running an application program.
<ul style="list-style-type: none"> It enables application programmers to control the initial display position, size and appearance of the top level GUI objects. 	<ul style="list-style-type: none"> It enables application programmers to control the initial position, size and appearance of the top level GUI objects, within a command shell window. Application programs may or may not malfunction or terminate after an operator re-adjusts the size of the command shell window. For example, the Midnight Commander 	<ul style="list-style-type: none"> It enables application programmers to control the initial position, size and appearance of the top level GUI objects, within a command shell window. Application programs will malfunction or terminate after an operator re-adjusts the size of the command shell window.

High Level API	Low Level API	Extended API
<p>Local Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell launch local application terminate local application terminate local shell logout of local platform host 		<p>Local & Remote Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell echo \$TERM # Display default type \$TERM=<xterm-family member ID> or <vt100-family member ID> echo \$TERM # Display changed type launch local application terminate local application login to remote platform host via "ssh <login ID>@<Host ID>:" launch remote application terminate remote application logout of remote platform host transfer file(s) from remote platform host via "sftp <login ID>@<Host ID>:<File IDs>" terminate local shell logout of local platform host
	<p>application from the Free Software Foundation automatically adjusts itself to changes in screen size when used with an xterm type terminal emulator but needs to be manually refreshed via Ctrl-L when used with a cygwin console shell.</p>	
<p>Host Platform High Level Interface (Host)</p> <p>Platform-specific API, libraries and User Interface for starting, initializing, configuring, controlling, monitoring and terminating computer hardware and software activities.</p> <ul style="list-style-type: none"> Linux (Fedora 17-20 & Ubuntu) 10.04-12.04 Mac OS X (Lion 10.7.4-10.9.1) Microsoft Windows (XP with SP3, 7, 8 & 8.1) with Cygwin (1.7.2) add-on 	<p>Host Platform Low Level Interface (Virtual Machine)</p> <p>Programming language specific API, platform-specific libraries and User Interface for starting, initializing, configuring, controlling, monitoring and terminating application software activities.</p> <ul style="list-style-type: none"> Python 2.6.x, 2.7.x and/or Python 3.x (provides platform independent Virtual Machine API) Python Curses Module (provides terminal independent keyboard, mouse and display API) NCurses Library (implements 	<p>Host Platform Low Level Interface (tsWxGTUI analogous to host services provided by "gtk", "msw", "osx" and "unix")</p> <p>Programming language specific API, platform-specific libraries and User Interface for starting, initializing, configuring, controlling, monitoring and terminating application software activities.</p> <ul style="list-style-type: none"> tsWxGraphicalTextUserInterface (Class uses the Standard Python Curses API to initialize, manage and shutdown input, from a keyboard and mouse, and output, to a two-dimensional display screen. It identifies user terminal make, model and features. It controls terminal

High Level API	Low Level API	Extended API
<p>Local Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell launch local application terminate local application terminate local shell logout of local platform host 		<p>Local & Remote Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell echo \$TERM # Display default type \$TERM=<xterm-family member ID> or <vt100-family member ID> echo \$TERM # Display changed type launch local application terminate local application login to remote platform host via "ssh <login ID>@<Host ID>:" launch remote application terminate remote application logout of remote platform host transfer file(s) from remote platform host via "sftp <login ID>@<Host ID>:<File IDs>" terminate local shell logout of local platform host
<ul style="list-style-type: none"> Unix (SunOS 5.11) 	<p>platform specific keyboard, mouse and display API)</p>	<p>device startup, shutdown and exception handling. It translates "wxPython"-style terminal color, pixel and character parameters into their "Curses" counterparts. Upon startup, it briefly restores or creates, saves or loads the splash screen bitmap image as specified in the "tsWxGlobals" configuration file.)</p>
<ul style="list-style-type: none"> 	<p>TopLevel Windows (wxWidgets & wxPython)</p> <ul style="list-style-type: none"> Frame (A GUI object whose size and position can usually be changed by the user. It usually has thick borders and a title bar, and can optionally contain a menu bar, toolbar and status bar. A frame can contain any GUI object that is not a frame or dialog. Frames are windows associated with an application task (such as an internet WEB Browser) that can be minimized into an icon, expanded to full screen or terminated upon operator demand.) 	<p>Top Level Windows ((tsWxGTUI implements feature(s) available in wxWidgets or wxPython library)</p> <ul style="list-style-type: none"> Frame (A GUI object whose size and position can (usually) be changed by the user. It usually has thick borders and a title bar, and can optionally contain a menu bar, toolbar and status bar. A frame can contain any GUI object that is not a frame or dialog. Frames are windows associated with an application task (such as an internet WEB Browser) that can be minimized into an icon, expanded to full screen or terminated upon operator demand.) Dialog (A GUI object, such as PasswordEntryDialog or

High Level API	Low Level API	Extended API
<p>Local Command Line Interface Launcher</p> <ul style="list-style-type: none"> ▪ login to local platform host ▪ launch local shell ▪ launch local application ▪ terminate local application ▪ terminate local shell ▪ logout of local platform host 		<p>Local & Remote Command Line Interface Launcher</p> <ul style="list-style-type: none"> ▪ login to local platform host ▪ launch local shell ▪ echo \$TERM # Display default type ▪ \$TERM=<xterm-family member ID> or <vt100-family member ID> ▪ echo \$TERM # Display changed type ▪ launch local application ▪ terminate local application ▪ login to remote platform host via "ssh <login ID>@<Host ID>:" ▪ launch remote application ▪ terminate remote application ▪ logout of remote platform host ▪ transfer file(s) from remote platform host via "sftp <login ID>@<Host ID>:<File IDs>" ▪ terminate local shell ▪ logout of local platform host
	<ul style="list-style-type: none"> ▪ Dialog (A GUI object, such as PasswordEntryDialog or TextEntryDialog, with a title bar and sometimes a system menu, which can be moved around the screen. It can contain controls and other GUI objects and is often used to allow the user to make some choice or to answer a question. Dialogs are pop-up windows associated with an application task's menu bar (such as an input field for an operator's search criteria and an output field for a list of candidate WEB sites) that will be terminated upon completion.) ▪ Redirected Output (An optional window to display the UNIX-style "stdout" (progress) and "stderr" (error) messages output by an application task via a "print" statement.) 	<p>TextEntryDialog, with a title bar and sometimes a system menu, which can be moved around the screen. It can contain controls and other GUI objects and is often used to allow the user to make some choice or to answer a question. Dialogs are pop-up windows associated with an application task's menu bar (such as an input field for an operator's search criteria and an output field for a list of candidate WEB sites) that will be terminated upon completion.)</p> <ul style="list-style-type: none"> ▪ Redirected Output (An optional window to display the UNIX-style "stdout" (progress) and "stderr" (error) messages output by an application task via a "print" statement. All output is automatically prefixed with the date (year/month/day); the time (hour:minute:second.millisecond such as "2011/01/23 12:34:56.789"; a separator (" - "); an optional severity level indicator (DEBUG, WARNING, ERROR etc. When the application designer wants to draw attention to special messages, the messages may

High Level API	Low Level API	Extended API
<p>Local Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell launch local application terminate local application terminate local shell logout of local platform host 		<p>Local & Remote Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell echo \$TERM # Display default type \$TERM=<xterm-family member ID> or <vt100-family member ID> echo \$TERM # Display changed type launch local application terminate local application login to remote platform host via "ssh <login ID>@<Host ID>:" launch remote application terminate remote application logout of remote platform host transfer file(s) from remote platform host via "sftp <login ID>@<Host ID>:<File IDs>" terminate local shell logout of local platform host
		<p>be enhanced by: Reversed or custom foreground and background colors; blink, bold, dim, normal, standout and underline special effects)</p>
		<p>Top Level Windows (tsWxGTUI implements feature(s) not available in wxWidgets or wxPython library)</p> <ul style="list-style-type: none"> Redirected Output (AppendText applies Color and Font Attribute Markup) TaskBar (buttons shift focus in manner similar to native host GUI that underlies wxWidgets and wxPython)

High Level API	Low Level API	Extended API
<p>Local Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell launch local application terminate local application terminate local shell logout of local platform host 		<p>Local & Remote Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell echo \$TERM # Display default type \$TERM=<xterm-family member ID> or <vt100-family member ID> echo \$TERM # Display changed type launch local application terminate local application login to remote platform host via "ssh <login ID>@<Host ID>:" launch remote application terminate remote application logout of remote platform host transfer file(s) from remote platform host via "sftp <login ID>@<Host ID>:<File IDs>" terminate local shell logout of local platform host
	<p>Low Level Windows (wxWidgets & wxPython)</p> <ul style="list-style-type: none"> Buttons (one or more action initiating selection) Check Boxes (one or more Left & Right Aligned independent feature selecting buttons) Gauges (Horizontal & Vertical bar graphs) Menu Bars (row of one or more pull down menu selections) Menus (column of one or more action initiating selections) 	<p>Low Level Windows (tsWxGTUI implements feature(s) available in wxWidgets or wxPython library)</p> <ul style="list-style-type: none"> Buttons (one or more action initiating selection) Check Boxes (one or more Left & Right Aligned independent feature selecting buttons) Gauges (Horizontal & Vertical bar graphs) Menu Bars (row of one or more pull down menu selections) Menus (column of one or more action initiating selections)

High Level API	Low Level API	Extended API
<p>Local Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell launch local application terminate local application terminate local shell logout of local platform host 		<p>Local & Remote Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell echo \$TERM # Display default type \$TERM=<xterm-family member ID> or <vt100-family member ID> echo \$TERM # Display changed type launch local application terminate local application login to remote platform host via "ssh <login ID>@<Host ID>:" launch remote application terminate remote application logout of remote platform host transfer file(s) from remote platform host via "sftp <login ID>@<Host ID>:<File IDs>" terminate local shell logout of local platform host
	<ul style="list-style-type: none"> Panels (optionally labeled window containing other GUI objects) Radio Boxes (Horizontal & Vertical panel containing two or more radio buttons) Radio Buttons (two or more Left & Right Aligned mutually exclusive feature selecting buttons) ScrollBar (panel with Horizontal and/or Vertical buttons, that control a scrollable text window, and a gauge, to display the position and size of the displayed text relative to the non-displayed text) ScrolledWindow (panel with a Horizontal and/or Vertical ScrollBar and a scrollable text window) 	<ul style="list-style-type: none"> Panels (optionally labeled window containing other GUI objects) Radio Boxes (Horizontal & Vertical panel containing two or more radio buttons) Radio Buttons (two or more Left & Right Aligned mutually exclusive feature selecting buttons) ScrollBar (panel with Horizontal and/or Vertical buttons, that control a scrollable text window, and a gauge, to display the position and size of the displayed text relative to the non-displayed text) ScrolledWindow (panel with a Horizontal and/or Vertical ScrollBar and a scrollable text window)

High Level API	Low Level API	Extended API
<p>Local Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell launch local application terminate local application terminate local shell logout of local platform host 		<p>Local & Remote Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell echo \$TERM # Display default type \$TERM=<xterm-family member ID> or <vt100-family member ID> echo \$TERM # Display changed type launch local application terminate local application login to remote platform host via "ssh <login ID>@<Host ID>:" launch remote application terminate remote application logout of remote platform host transfer file(s) from remote platform host via "sftp <login ID>@<Host ID>:<File IDs>" terminate local shell logout of local platform host
	<ul style="list-style-type: none"> Splash Screen (Optional window to display a bitmap description of the application. It briefly appears before any top-level wxPython-style application Frames.) StaticText (panel for displaying text) Status Bars (panel with one or more Status Bar Panels) Status Bar Panels (panel for displaying a single piece of status information) TextCtrl (panel for displaying text that may be optionally formatted and scrolled) Tool Bars (row of one or more action initiating selections) 	<ul style="list-style-type: none"> Splash Screen (Optional window to display a text-based, bitmap-like description of the application. It briefly appears before any top-level wxPython-style application Frames.) StaticText (panel for displaying text) Status Bars (panel with one or more Status Bar Panels) Status Bar Panels (panel for displaying a single piece of status information) TextCtrl (panel for displaying text that may be optionally formatted and scrolled) Tool Bars (row of one or more action initiating selections)

High Level API	Low Level API	Extended API
<p>Local Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell launch local application terminate local application terminate local shell logout of local platform host 		<p>Local & Remote Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell echo \$TERM # Display default type \$TERM=<xterm-family member ID> or <vt100-family member ID> echo \$TERM # Display changed type launch local application terminate local application login to remote platform host via "ssh <login ID>@<Host ID>:" launch remote application terminate remote application logout of remote platform host transfer file(s) from remote platform host via "sftp <login ID>@<Host ID>:<File IDs>" terminate local shell logout of local platform host
		<p>Low Level Windows ((tsWxGTUI implements feature(s) not available in wxWidgets or wxPython library)</p> <ul style="list-style-type: none"> Buttons (BORDER_SIMPLE for single line label replaced by bracketed label) Panels (optionally labeled window containing other GUI objects) ScrollBar Buttons (implements feature(s) not available in nCurses library) ScrollBar Gauges (implements feature(s) not available in NCurses library) Scrolled (Template for scrollable text windows with Horizontal and/or Vertical ScrollBars) Scrolled Text (implements text markup feature(s) not available in NCurses library) ScrolledWindow (scrollable text window with Horizontal and/or Vertical ScrollBars)Task Bar Buttons (implements feature(s) not available in NCurses library to apply keyboard focus shift) TextCtrl (AppendText applies Color and Font Attribute Markup)

High Level API	Low Level API	Extended API
<p>Local Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell launch local application terminate local application terminate local shell logout of local platform host 		<p>Local & Remote Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell echo \$TERM # Display default type \$TERM=<xterm-family member ID> or <vt100-family member ID> echo \$TERM # Display changed type launch local application terminate local application login to remote platform host via "ssh <login ID>@<Host ID>:" launch remote application terminate remote application logout of remote platform host transfer file(s) from remote platform host via "sftp <login ID>@<Host ID>:<File IDs>" terminate local shell logout of local platform host
<p>GUI Object Styles (wxWidgets & wxPython)</p> <ul style="list-style-type: none"> Overlapping Tiled Border None Border Simple Splash Screen (Bitmap Image) 	<p>GUI Object Layout Sizers (wxWidgets & wxPython)</p> <ul style="list-style-type: none"> Box (automatically scaled Horizontal & Vertical Layout) Grid (automatically scaled Horizontal & Vertical Layout) Splash Screen (Bitmap Image display) Static Box (combines Box Sizer with Bordered Panel layout) 	<p>GUI Object Styles (tsWxGTUI)</p> <ul style="list-style-type: none"> Box (automatically scaled Horizontal & Vertical Layout) Grid (automatically scaled Horizontal & Vertical Layout) Splash Screen (off-line, NCurses-style Bitmap Image builder utility) Static Box (combines Box Sizer with Bordered Panel layout)
<p>Text (wxWidgets & wxPython)</p> <ul style="list-style-type: none"> Multiple Proportional (such as Times and Helvetica) and Monospaced (Courier) Font Families Multiple Sizes (8pt to 288pt) Special Effects (numerous including horizontal, vertical, rotated etc.) 	<p>Text (nCurses)</p> <ul style="list-style-type: none"> Single Monospaced (configurable by terminal operator such as Courier) Font Single Size (configurable by terminal operator such as 12pt having 8 pixel width and 12 pixel height) Special Effects (configurable by application programmer but limited to one or a combination of the following: Blinking, Bold, Dim, Normal, Reverse, Standout) 	<p>Text (tsWxGTUI)</p> <ul style="list-style-type: none"> Single Monospaced Font Single Size (conversions between wxWidgets pixel and nCurses character dimensions presumes 12pt font having 8 pixel width and 12 pixel height) Special Effects (configuration file "tsWxGlobals" establishes defaults for various nCurses text markup styles that may be changed or overridden by the application programmer)

High Level API	Low Level API	Extended API
<p>Local Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell launch local application terminate local application terminate local shell logout of local platform host 		<p>Local & Remote Command Line Interface Launcher</p> <ul style="list-style-type: none"> login to local platform host launch local shell echo \$TERM # Display default type \$TERM=<xterm-family member ID> or <vt100-family member ID> echo \$TERM # Display changed type launch local application terminate local application login to remote platform host via "ssh <login ID>@<Host ID>:" launch remote application terminate remote application logout of remote platform host transfer file(s) from remote platform host via "sftp <login ID>@<Host ID>:<File IDs>" terminate local shell logout of local platform host
	and Underline)	
<p>Color Palette (wxWidgets & wxPython)</p> <ul style="list-style-type: none"> 68-colors 4624-foreground/background color pairs 	<p>Color Palette (nCurses)</p> <ul style="list-style-type: none"> 8-colors (black, blue, cyan, green, magenta, red, white, yellow) 64-foreground/background color pairs 256-colors (option requires wide-character build of nCurses and Python Curses modules) 65536-foreground/background color pairs (option requires wide- 	<p>Color Palette (tsWxGTUI)</p> <ul style="list-style-type: none"> tsWxGraphicalTextUserInterface maps wxWidgets 68-colors (4624-color pairs) into nCurses 8-colors (64-color pairs) tsWxGraphicalTextUserInterface defines nCurses 256-colors (65536-color pairs) into wxWidgets 68-colors (4624-color pairs) and 188 other colors (60912 other color pairs) <p>Non-Color Palette (tsWxGTUI)</p>

High Level API	Low Level API	Extended API
<p>Local Command Line Interface Launcher</p> <ul style="list-style-type: none"> ▪ login to local platform host ▪ launch local shell ▪ launch local application ▪ terminate local application ▪ terminate local shell ▪ logout of local platform host 		<p>Local & Remote Command Line Interface Launcher</p> <ul style="list-style-type: none"> ▪ login to local platform host ▪ launch local shell ▪ echo \$TERM # Display default type ▪ \$TERM=<xterm-family member ID> or <vt100-family member ID> ▪ echo \$TERM # Display changed type ▪ launch local application ▪ terminate local application ▪ login to remote platform host via "ssh <login ID>@<Host ID>:" ▪ launch remote application ▪ terminate remote application ▪ logout of remote platform host ▪ transfer file(s) from remote platform host via "sftp <login ID>@<Host ID>:<File IDs>" ▪ terminate local shell ▪ logout of local platform host
	<p>character build of nCurses and Python Curses modules)</p> <p>Non-Color Palette (nCurses)</p> <ul style="list-style-type: none"> ▪ Platform-specific 1-color is visible for Foreground (White) and NOT visible (Black) for Background. The 1-color (2-color pair) is determined either by the phosphor (Green, Orange, White) used in the physical terminal's display or by the color adopted by the terminal emulator (Cygwin's console-based and xterm-based vt100 emulators use White on Black while Apple's Mac OS X xterm-based vt100 emulator uses Black on White) 	<ul style="list-style-type: none"> ▪ Platform-specific 1-color is visible for Foreground (White) and NOT visible (Black) for Background. The 1-color (2-color pair) is determined either by the phosphor (Green, Orange, White) used in the physical terminal's display or by the color adopted by the terminal emulator (Cygwin's console-based and xterm-based vt100 emulators use White on Black while Apple's Mac OS X xterm-based vt100 emulator uses Black on White)

1.4 APPENDIX C - DELIVERABLES

The following table describes the work product(s) to be delivered by the developer:

DELIVERABLE	DESCRIPTION
Software Package(s)	<p>The "tsWxGTUI_PyVx" Toolkit software:</p> <ul style="list-style-type: none"> ▪ Building Block Library Packages ("tsLibCLI" & "tsLibGUI") in Python 2x & Python 3x source code formats ▪ Tool Application Programs ("tsToolsCLI" & "tsToolsGUI") in Python 2x & Python 3x source code formats ▪ Test Application Programs ("tsTestsCLI" & "tsTestsGUI") in Python 2x & Python 3x source code formats ▪ Utility Modules ("tsUtilities") in Python 2x and Python 3x and POSIX-style shell script formats
Engineering Documentation ("tsEngineering") (Optional, Fee-based subscription)	<p>While the <i>TeamSTARS</i> "tsWxGTUI_PyVx" Toolkit is released as free, open source software, the engineering documentation establishes proprietary Copyright ownership by the original Toolkit Author(s).</p> <p>Toolkit recipients can obtain a license for the engineering documentation, if their intent is to assemble a large team to commercialize the software and want to kickstart the effort.</p> <p>The "tsEngineering" subdirectory contains a collection of Toolit Author written large, complex documents, including structured documents, featuring multiple fonts and graphic images.</p> <p>The documents are authored using the following products:</p> <ul style="list-style-type: none"> ▪ Author-it --- A help authoring tool (http://www.author-it.com) and content management system for creating, maintaining, and distributing single-sourced content. It generates Microsoft Word files with the appropriate outline numbering (for table of contents, sections, paragraphs, lists and figures) regardless of where the single-source content originated. ▪ Microsoft Office --- A collection of software applications (http://www.microsoftstore.com/store/msusa/en_US/cat/Office/categoryID.62684700) intended to be used by knowledge workers. The components are generally distributed together, have a consistent user interface and usually can interact with each other, sometimes in ways that the operating system would not normally allow. <ul style="list-style-type: none"> Access (data base) Excel (spreadsheet) PowerPoint (presentation) Word (word processor) ▪ Microsoft Visio --- A software application (http://www.microsoftstore.com/store/msusa/en_US/list/Visio/categoryID.62687700) intended to be used by knowledge workers.

DELIVERABLE	DESCRIPTION
Software Package(s)	<p>The "tsWxGTUI_PyVx" Toolkit software:</p> <ul style="list-style-type: none"> ▪ Building Block Library Packages ("tsLibCLI" & "tsLibGUI") in Python 2x & Python 3x source code formats ▪ Tool Application Programs ("tsToolsCLI" & "tsToolsGUI") in Python 2x & Python 3x source code formats ▪ Test Application Programs ("tsTestsCLI" & "tsTestsGUI") in Python 2x & Python 3x source code formats ▪ Utility Modules ("tsUtilities") in Python 2x and Python 3x and POSIX-style shell script formats
	<p>Visio (diagrams)</p> <ul style="list-style-type: none"> ▪ Fineprint Software (http://fineprint.com) <p>FinePrint (A print driver, for Microsoft Windows, which allows you to manage the printing process. It helps you to reduce printing costs while enhancing and managing complex print jobs.)</p> <p>pdfFactory Pro (An application, for Microsoft Windows, that provides Adobe PDF compatible output.)</p> <p>The documents accompany the computer software for the training and reference use of the software developer:</p> <ul style="list-style-type: none"> ▪ It explains the purpose, goals, non-goals, system requirements, interface requirements, software requirements, qualification requirements, development plan, software design, how it operates and how to install, use and troubleshoot it. ▪ It is provided in various application specific formats (such as Adobe PDF and Microsoft Office & Visio formats which may be read and edited by the free, open source, cross-platform LibreOffice Suite). <p>The "tsWxGTUI_PyVx" Toolkit software development and release documents are deliverable in Adobe's Portable Document Format and Microsoft's Word Format (with associated bit-mapped images in JPG or PNG Format):</p> <ul style="list-style-type: none"> ▪ Vol. 0 --- SDIST Announcement ▪ Vol. 1 --- SDIST Brochure ▪ Vol. 2 --- SDIST Introduction ▪ Vol. 3 --- SDIST Terms & Conditions ▪ Vol. 4 --- SDIST Software Development Plans ▪ Vol. 5 --- SDIST System Specification ▪ Vol. 6 --- SDIST Interface Requirements Specification ▪ Vol. 7 --- SDIST Software Requirements Specification ▪ Vol. 8 --- SDIST Release Notes ▪ Vol. 9 --- SDIST Software User's Manual ▪ Vol. 10 --- SDIST Appendixes ▪ Vol. 11 --- SDIST Dictionary

DELIVERABLE	DESCRIPTION
Software Package(s)	<p>The "tsWxGTUI_PyVx" Toolkit software:</p> <ul style="list-style-type: none"> ▪ Building Block Library Packages ("tsLibCLI" & "tsLibGUI") in Python 2x & Python 3x source code formats ▪ Tool Application Programs ("tsToolsCLI" & "tsToolsGUI") in Python 2x & Python 3x source code formats ▪ Test Application Programs ("tsTestsCLI" & "tsTestsGUI") in Python 2x & Python 3x source code formats ▪ Utility Modules ("tsUtilities") in Python 2x and Python 3x and POSIX-style shell script formats
	<ul style="list-style-type: none"> ▪ Vol. 12 --- SDIST To-Do List
Engineering Notes	<p>The "tsWxGTUI_PyVx" Toolkit software development notes in Microsoft's Word Format:</p> <ul style="list-style-type: none"> ▪ Class List.doc ▪ Relationships_Between_tsWxGTUI_Toolkit_APIs.doc ▪ RGB to Color Name Mapping(Triplet and Hex).doc ▪ Writing a Python Package by Tarek Ziadé.doc <p>The "tsWxGTUI_PyVx" Toolkit software development notes in Microsoft's Access Format:</p> <ul style="list-style-type: none"> ▪ tsWxPython.mdb <p>The "tsWxGTUI_PyVx" Toolkit software development diagram notes in Microsoft's Visio Format:</p> <ul style="list-style-type: none"> ▪ Distributed System.vsd ▪ Networked Architecture.vsd ▪ System Architecture.vsd ▪ tsWxPython Org Chart.vsd <p>The "tsWxGTUI_PyVx" Toolkit software development notes in Microsoft's Excel Format:</p> <ul style="list-style-type: none"> ▪ Platform_Configuration.xls <p>The "tsWxGTUI_PyVx" Toolkit software development notes in Plain Text Format:</p> <ul style="list-style-type: none"> ▪ README_BMP.txt
Equipment Operator Documentation ("tsDocuments")	<p>The "tsWxGTUI_PyVx" Toolkit installation, configuration, operation and troubleshooting documents in Plain Text Format:</p> <ul style="list-style-type: none"> ▪ tsDocCLI-1-GettingStartedFiles ▪ tsDocCLI-2-DistributionReadMeFiles ▪ tsDocCLI-3-DeveloperReadMeFiles ▪ tsDocCLI-5-UsageTermsAndConditions ▪ tsManPages ("tsLibCLI", "tsLibGUI", "tsToolsCLI", "tsToolsGUI", "tsTestsCLI", "tsTestsGUI") <p>1. Operator Documentation</p> <p>User documentation for each "tsWxGTUI" Toolkit distribution is contained, in plain text format, in the subdirectory named "/tsDocCLI". The subdirectory is organized, by topic, to contain the following:</p>

DELIVERABLE	DESCRIPTION
Software Package(s)	<p>The "tsWxGTUI_PyVx" Toolkit software:</p> <ul style="list-style-type: none"> ▪ Building Block Library Packages ("tsLibCLI" & "tsLibGUI") in Python 2x & Python 3x source code formats ▪ Tool Application Programs ("tsToolsCLI" & "tsToolsGUI") in Python 2x & Python 3x source code formats ▪ Test Application Programs ("tsTestsCLI" & "tsTestsGUI") in Python 2x & Python 3x source code formats ▪ Utility Modules ("tsUtilities") in Python 2x and Python 3x and POSIX-style shell script formats
	<p>a. How to Prepare Platform to Get Started</p> <p>The "./GettingStartedFiles" subdirectory contains the following file(s):</p> <p>"README-GettingStarted.txt" ---</p> <p>b. How to Install and Redistribute</p> <p>The "./DistributionReadMeFiles" subdirectory contains the following file(s):</p> <ul style="list-style-type: none"> ▪ "AUTHORS.txt" --- List of the principal "tsWxGTUI" Toolkit author(s) and authors credited for work covered by a prior copyright and license. ▪ "BUGS.txt" --- List of Known Problems / Issues. ▪ "CHANGE_LOG.txt" --- List of Additions, Modification and Deletions. ▪ "CONFIGURE.txt" --- Instructions for applying factory and site-specific configurations. ▪ "COPYING.txt" --- Instructions for copying all or a portion of the distribution. ▪ "FAQ.txt" --- Answers to Frequently Asked Questions. ▪ "INSTALL.txt" --- Describes steps to download, extract install and configure the "tsWxGUI" Toolkit. ▪ "LICENSE.txt" --- General and special arrangements, provisions, rules, specifications and standards that form an integral part the agreement or contract between the creator and recipient of Copyrighted and Licensed Work. ▪ "MANIFEST.txt" --- Tally List for deliverable items. ▪ "NEWS.txt" --- Announcements of new releases. ▪ "NOTICES.txt" --- Details the copyright(s) and license(s). ▪ "OPERATE.txt" --- Describes steps to use the "tsWxGUI" Toolkit. ▪ "README.txt" --- Introduces new recipients to the purpose, goals, non-goals, design and features of the computer software product. ▪ "THANKS.txt" --- Acknowledgments to those otherwise unsung heros who contributed time and effort to supporting the authors as planners, editors, designers, coders and testers. ▪ "TO-DO.txt" --- A To-Do-List provides a roadmap for development and troubleshooting work. ▪ "TROUBLE SHOOT.txt" --- Provides a list of available reference resources and a guide for planning, developing and troubleshooting a cross-platform system of hundreds of files each containing a few, tens or hundred of class, data and method definitions. Its complexity becomes apparent in the recent software Lines-Of-Code

DELIVERABLE	DESCRIPTION
Software Package(s)	<p>The "tsWxGTUI_PyVx" Toolkit software:</p> <ul style="list-style-type: none"> ▪ Building Block Library Packages ("tsLibCLI" & "tsLibGUI") in Python 2x & Python 3x source code formats ▪ Tool Application Programs ("tsToolsCLI" & "tsToolsGUI") in Python 2x & Python 3x source code formats ▪ Test Application Programs ("tsTestsCLI" & "tsTestsGUI") in Python 2x & Python 3x source code formats ▪ Utility Modules ("tsUtilities") in Python 2x and Python 3x and POSIX-style shell script formats
	<p>metrics.</p> <p>c. How to Use and Modify</p> <p>The "./DeveloperReadMeFiles" subdirectory contains the following file(s):</p> <ul style="list-style-type: none"> ▪ "README.txt" ▪ "README_1st.txt" ▪ "README_1st-PyPI-Dev-tsWxGTUI.txt" ▪ "README-01-Title_Page.txt" ▪ "README-02-Table_Of_Contents.txt" ▪ "README-03-Purpose.txt" ▪ "README-04-Goals.txt" ▪ "README-05-Non-Goals.txt" ▪ "README-06-Design_Strategy.txt" ▪ "README-07-Design_Architecture.txt" ▪ "README-08-Software_Configuration_Management.txt" ▪ "README-09-tsWxGTUI_Directories.txt" ▪ "README-10-tsToolkitCLI_Directories.txt" ▪ "README-11-tsToolkitGUI_Directories.txt" ▪ "README-12-Features.txt" ▪ "README-13-Current_Capabilities.txt" ▪ "README-14-Current_Limitations.txt" ▪ "README-15-Reference_Documents.txt" <p>d. How to Learn "tsWxGTUI" Toolkit Software Development</p> <p>The "./DeveloperTutorialFiles" subdirectory contains the following file(s):</p> <ul style="list-style-type: none"> ▪ "CLI_0_hello_world_print_statement.py" ▪ "CLI_1_hello_world_print_function.py" ▪ "CLI_2_hello_world_script_environment.py" ▪ "CLI_3_hello_world_main_module_application.py" ▪ "GUI_4_Curses_LowLevel_WidgetApi_application.py" ▪ "GUI_5_tsWxGTUI_HighLevel_WidgetApi_application.py"

DELIVERABLE	DESCRIPTION
Software Package(s)	<p>The "tsWxGTUI_PyVx" Toolkit software:</p> <ul style="list-style-type: none">▪ Building Block Library Packages ("tsLibCLI" & "tsLibGUI") in Python 2x & Python 3x source code formats▪ Tool Application Programs ("tsToolsCLI" & "tsToolsGUI") in Python 2x & Python 3x source code formats▪ Test Application Programs ("tsTestsCLI" & "tsTestsGUI") in Python 2x & Python 3x source code formats▪ Utility Modules ("tsUtilities") in Python 2x and Python 3x and POSIX-style shell script formats
	<ul style="list-style-type: none">▪ "GUI_6_tsWxGTUI_HighLevel_BoxSizerApi_application.py"▪ "ReadMe_Developer_Tutorial_Files.txt"▪ "test_tsCxGlobals.py"▪ "test_tsWxGlobals.py"▪ "tsCxGlobals.py" <p>e. Terms & Conditions</p> <p>The "./UsageTermsAndConditions" subdirectory contains the following file(s):</p> <ul style="list-style-type: none">▪ "COPYRIGHT.txt"▪ "LICENSE.txt"▪ "NOTICES.txt"▪ "SplashScreenDesignersGuide.txt"

1.5 APPENDIX D - ACCOMPLISHMENTS (Draft)

This section shall briefly state the development accomplishments for the software to which this document applies.

- *Capabilities* (on page 57)
- *Limitations* (on page 70)
- *Comparison of wxPython with tsWxGTUI* (see "*Comparison of wxPython with tsWxGTUI (Development Plan)*" on page 71)

1.5.1 Capabilities

This paragraph shall briefly state the accomplished interface and functional goals of the system and the software to which this document applies.

1.5.1.1 Platform Hardware and Software

- 1 Using "Python" technology, the toolkit enables applications to run, without modification, on platforms with:
 - a) Hardware:
 - 32-bit processor architectures
 - 64-bit processor architectures
 - b) Software:
 - POSIX-compatible operating system environments such as:
 - "Linux"
 - "Mac OS X"
 - "Unix"
 - "Windows" (requires Cygwin, the free and open source add-on, Linux-like Command Line Interface and GNU toolkit from Red Hat).
- 2 Using "Python" and "Curses" technology, the toolkit enables applications to run, without modification, on platforms with terminals that include the following keyboard, multi-color/non-color electronic display and optional mouse configurations:

Linux Terminal with xterm, xterm-color, xterm-16color, xterm-256color emulation or with non-color vt100 and vt220 emulation with/without mouse.

- a) Mac OS X iTerm and Terminal with xterm, xterm-color, xterm-16color, xterm-256color emulation or with non-color vt100 and vt220 emulation without mouse.
- b) Microsoft Windows with cygwin xterm, xterm-color, xterm-16color, xterm-256color, cygwin mintty (xterm) and cygwin console emulation with/without mouse or with non-color vt100 and vt220 emulation with/without mouse.
- c) UNIX (Solaris and OpenIndiana) Terminal with xterm, xterm-color, xterm-16color, xterm-256color emulation or with non-color vt100 and vt220 emulation with/without mouse.

1.5.1.2 tsLibCLI Software

The library of building blocks is organized, by the functional scope of each component, into a collection of "packages". When appropriate, any package may import and use the services of any other tsLibCLI package.

Source code is contained in the associated ["src"] sub-directory. Depending on its complexity, the source code of a package may also be sub-divided and organized into a set of one or more source files.

Unit/Integration Test code is contained in the associated ["test"] sub-directory. Depending on its complexity, the test code of a package may also be sub-divided and organized into a set of one or more test files.

Provides a POSIX-style Command Line Interface (CLI) that is user friendly, maintainable and easily enhanced:

1 tsApplicationPkg

Class and run time library component initializes and configures the application using the following keyword-value pairs and positional arguments: input provided on the command line by an operator and input provided in the parameter list of the application's invocation of the class instantiation.

2 tsCommandLineEnvPkg

Class and run time library component provides platform independent configuration, initialization, input/output supervisor and application launcher, event handler and terminator.

3 tsCommandLineInterfacePkg

Class and methods that prompt or re-prompt the operator for input, validate that the operator has supplied the expected number of inputs and that each is of the expected type.

4 tsConfigPkg (deprecated)

Class for a config file reader/writer that supports nested sections in the config files.

5 tsCxGlobalsPkg

Module to establish configuration constants and macro-type functions for the Command Line Interface mode of the "tsWxGTUI_PyVx" Toolkit.

6 tsDecoratorsPkg (deprecated)

Class to define a general-purpose decorator factory and common decorators that takes a caller function as input and returns a decorator with the same attributes.

7 tsDoubleLinkedListPkg

Class to establish a representation of a linked list with forward and backward pointers.

8 tsExceptionsPkg

Class to define and handle error exceptions. Maps run time exception types into 8-bit exit codes and prints associated diagnostic message and traceback info.

9 tsLoggerPkg

Class that emulates a subset of Python logging API. It defines and handles prioritized, time and date stamped event message formatting and output to files and devices. Files are organized in a date and time stamped directory named for the launched application. Unix-type devices include syslog, stderr, stdout and stdscr (the ncurses display screen). It also supports "wxPython"-style logging of assert and check case results.

10 tsOperatorSettingsParserPkg

Class to parse the command line entered by the operator of an application program. Parsing extracts the Keyword-Value pair Options and Positional Arguments that will configure and control the application during its execution.

11 tsPlatformRunTimeEnvironmentPkg

Class to capture current hardware, software and network information about the run time environment for the user process.

12 tsReportUtilitiesPkg

Class defining methods used to format information: date and time (begin, end and elapsed), file size (with kilo-, mega-, giga-, tera-, peta-, exa-, zeta- and yotta-byte units) and nested Python dictionaries.

13 tsSysCommandsPkg

Class definition and methods for issuing shell commands to and receiving responses from the host operating system.

14 tsThreadPoolPkg (deprecated)

Class to define a thread pool object that maintains a pool of worker threads to perform time consuming operations in parallel. It assigns jobs to the threads by putting them in a work request queue, where they are picked up by the next available thread. This then performs the requested operation in the background and puts the results in another queue. The thread pool object can then collect the results from all threads from this queue as soon as they become available or after all threads have finished their work. It's also possible, to define callbacks to handle each result as it comes in.

1.5.1.3 tsToolsCLI Capabilities

The library of development tools is organized, by the functional scope of each tool, into a collection of "packages". When appropriate, any package may import and use the services of any tsLibCLI and/or any other tsToolsCLI package.

Source code is contained in the associated ["src"] subdirectory. Depending on its complexity, the source code of a package may also be sub-divided and organized into a set of one or more source files.

Unit/Integration Test code is contained in the associated ["test"] sub-directory. Depending on its complexity, the test code of a package may also be sub-divided and organized into a set of one or more test files.

1 tsLinesOfCodeProjectMetricsPkg

- a) Python application program, with a Command Line Interface (CLI), that generates reports of software project progress and the estimated cost (or contributed value) of the project when it is finally completed.
- b) It scans an operator designated file directory tree containing the source files, in one or more programming language specific formats (such as Ada, Assembler, C/C++, Cobol, Fortran, PL/M, Python, Text, and various command line shells).
- c) For each file, it accumulates and reports the total number of code lines, blank/comment lines, words and characters.
- d) For each programming language format, it accumulates and reports a summary of details of the associated source files.
- e) For the entire set of source files, it accumulates and reports a summary of details.
- f) It uses the summary of the entire set of source files to derive, analyze, estimate and report metrics for the software development project (such as labor, cost, schedule and lines of code per day productivity).
- g) See the file "tsLinesOfCodeProjectMetrics.py" for details on the purpose, capabilities, limitations, usage, class(es) and method(s) associated with this package.

2 tsPlatformQueryPkg

- a) Python application program, with a Command Line Interface (CLI), that captures current hardware and software information about the run time environment available to computer programs.
- b) See the file "tsPlatformQuery.py" for details on the purpose, capabilities, limitations, usage, class(es) and method(s) associated with this package.

3 tsStripCommentsPkg

- a) Python application program, with a Command Line Interface (CLI), that transforms an annotated, development version of a directory of sub-directories and Python source files into an unannotated copy. The copy is intended to conserve storage space when installed in an embedded system. The transformation involves stripping comments and doc strings by de-tokenizing a tokenized version of each Python source file. Non-Python ("*.txt") files are trimmed of trailing whitespace.
- b) See the file "tsStripComments.py" for details on the purpose, capabilities, limitations, usage, class(es) and method(s) associated with this package.

4 tsStripLineNumbersPkg

- a) Python application program, with a Command Line Interface (CLI), that strips line numbers from source code (such as annotated listings) that, unlike programs coded in Basic, do not reference line numbers for conditional branching.
- b) See the file "tsStripLineNumbers.py" for details on the purpose, capabilities, limitations, usage, class(es) and method(s) associated with this package.

5 tsTreeCopyPkg

- a) Python application program, with a Command Line Interface (CLI), that copies the contents of an operator designated source directory to an operator designated target directory.
- b) See the file "tsTreeCopy.py" for details on the purpose, capabilities, limitations, usage, class(es) and method(s) associated with this package.

6 tsTreeTrimLinesPkg

- a) Python application program, with a Command Line Interface (CLI), that copies the contents of an operator designated source directory to an operator designated target directory after stripping superfluous white space (blanks) from the end of each line.
- b) See the file "tsTreeTrimLines.py" for details on the purpose, capabilities, limitations, usage, class(es) and method(s) associated with this package.

1.5.1.4 tsTestsCLI Capabilities**1 tsTestsCLI Functional Capabilities**

TBD

1.5.1.5 tsUtilities Capabilities**1 tsUtilities Functional Capabilities**

TBD

1.5.1.6 tsLibGUI Capabilities**1 tsLibGUI Functional Capabilities**

The library of building blocks is organized, by the functional scope of each component, into a collection of "packages". When appropriate, any package may import and use the services of any tsLibCLI and/or any other tsLibGUI package.

Source code is contained in the associated ["src"] sub-directory. Depending on its complexity, the source code of a package may also be sub-divided and organized into a set of one or more source files.

Unit/Integration Test code is contained in the associated ["test"] sub-directory. Depending on its complexity, the test code of a package may also be sub-divided and organized into a set of one or more test files.

Provides a "wxPython"-style, "nCurses"-based Graphical-style User Interface toolkit that is user friendly, maintainable and easily enhanced.

It makes efficient use of the available display real estate by avoiding the waste associated with multi-character boarder thickness.

Its high-level, character-mode GUI-style Application Programming Interface (API) emulates a subset of the pixel-mode API for the "wxPython" binding to the popular C++ language based "wxWidgets" GUI Toolkit.

The emulation uses Python's "curses" module to interface with and build its high-level, "wxPython"-style emulation from the available services of the low-level, character-mode, "nCurses" GUI-toolkit, the de-facto standard for portable advanced terminal handling.

The emulation enhances the "nCurses" service capabilities so as to identify and associate mouse clicks with the top-most, visible GUI object (Frame, Dialog, Panel, Label, Button etc.) containing the cursor tip (in a manner similar to the Host Operating System-specific GUI services used by "wxWidgets" / "wxPython").

The emulation automatically converts positioning and sizing dimensions between the pixel units used by "wxPython"-style applications and the character units used by "nCurses".

a) **tsWx**

Module to load all symbols that should appear within the wxPython.wx emulation namespace. Included are various classes, constants, functions and methods available for use by applications built with components from the "wxPython" emulation infrastructure.

Tiled and Overlapped Windows - Cannot currently shift focus to bring background GUI object(s) to foreground. **Limitation: Operator changes to the operating system command line shell window size may change windows from Tiled to/from Overlapped.**

Box and Grid Sizers - Border overlap is dependant on screen real estate available at run time. **Limitation: Operator changes to the operating system command line shell window size may change windows from Tiled to/from Overlapped.**

Buttons - Single line label is contained within square brackets ("[label]"); multi-line label is contained within box whose left and right side borders are each one 8-pixel character wide and whose top and bottom borders are each one 12-pixel character high. Active mouse-click area encompasses the label and border.

CheckBoxes - Check marks (dual- / tri-state) are displayed within square brackets (In-Active "[]", Active "[X]", Tri-state "[-]") and can be aligned to left or right of label. The active mouse-click area encompasses the check mark, label and border. Clicking on a CheckBox changes its round-robin state to the next one. Testing has NOT gone beyond basic display and mouse click interaction.

Gauges - Horizontal or Vertical bar graphs contained within a bordered box and filled from left to right or from bottom to top. The box's left and right side borders are each one 8-pixel character wide and whose top and bottom borders are each one 12-pixel character high.

MenuBars and Menus - Label hidden hot-key tokens designate the hot-key character to be underlined or highlighted. **Limitation: Testing has NOT gone beyond basic display.**

RadioBoxes and RadioButtons - Radio buttons are displayed within RadioBoxes and the button state is displayed within parentheses (In-Active "()", Active "(*)"). The active mouse-click area encompasses the Radio Button label and border. Clicking on a button changes its state to Active and changes the state of all other RadioButtons with the same Radio Box to In-Active. The RadioBox's left and right side borders are each one 8-pixel character wide and whose top and bottom borders are each one 12-pixel character high. **Limitation: Testing has NOT gone beyond basic display and mouse click interaction.**

Redirected Output Window - Displays date and time stamped operational, diagnostic and exception log messages. The messages are written to the next available empty line on the display or on the bottom line after space has been made by discarding the top line and scrolling up the remaining lines. Optional markup mechanism adds foreground/background colorization of those message beginning with a logging severity level key word (PRIVATE, DEBUG, INFO, NOTICE, WARNING, ALERT, ERROR, CRITICAL, EMERGENCY). The Redirected Output Window's left and right side borders are each one 8-pixel character wide and whose top and bottom borders are each one 12-pixel character high.

ScrollBarButton - Single line label ("<", ">", "^", "v") is contained within box whose left and right side borders are each one 8-pixel character wide and whose top and bottom borders are each one 12-pixel character high. Active mouse-click area encompasses the label and border.

ScrollBarGauge - Horizontal or Vertical bar graphs contained within a bordered box. Its highlighted and non-highlighted areas display the position and size of text displayed in an associated scrollable text window relative to the undisplayed text. The box's left and right side borders are each one 8-pixel character wide and whose top and bottom borders are each one 12-pixel character high. The concepts is describe in the folloing table:

ScrollBarGauge

The max buffer length represents the largest horizontal column or vertical row count. The gauge is empty only when there are no columns or rows displayed (i.e. when the max buffer length is zero).

The gauge is filled only when all columns or all rows are displayed.

The gauge is partially filled only when some columns or rows are displayed.

The highlighted area for the gauge begins at the relative column or row of the displayed text. The highlighted area ends at the relative column or row of the displayed text. For example:

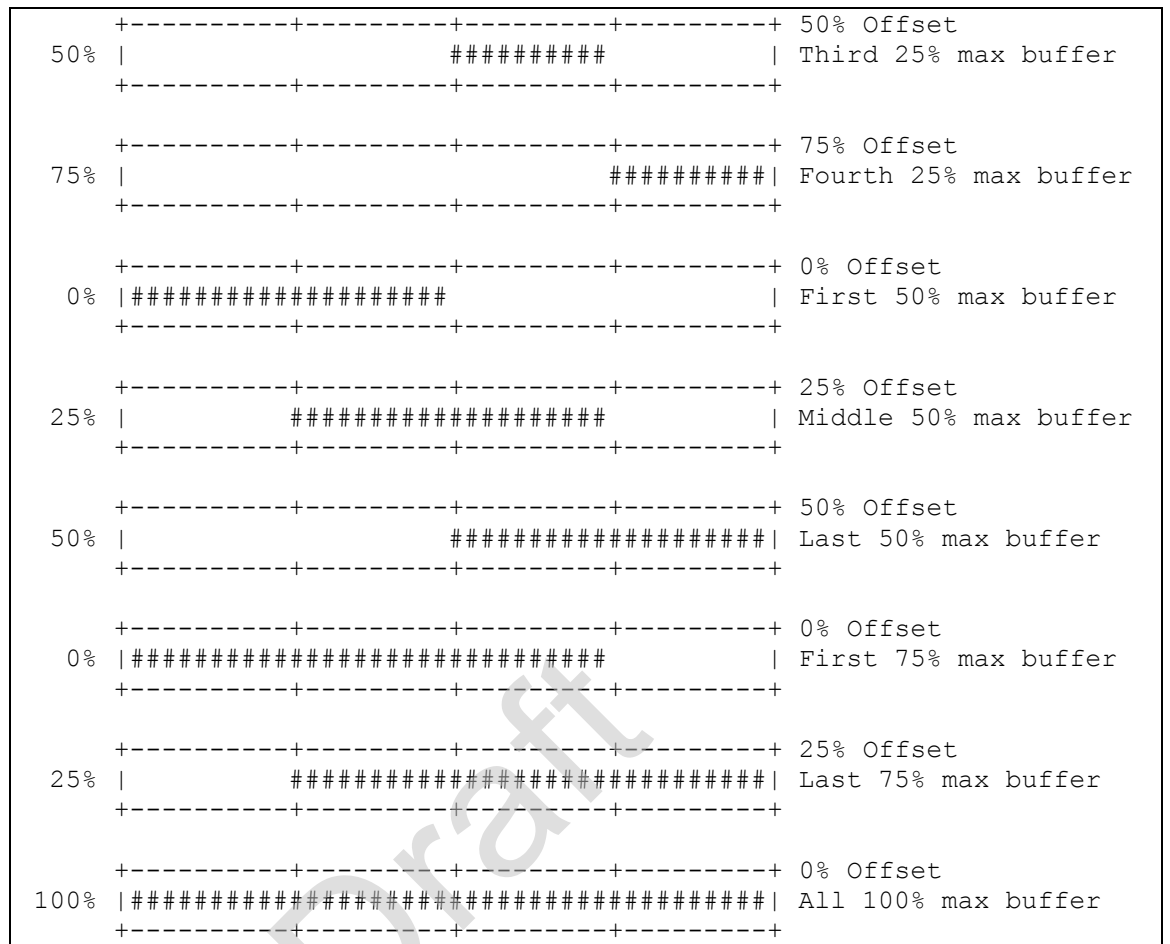
- **0%** - The column or row begins or ends at the left-most horizontal or vertical position in the list of text strings.
- **25%** - The column or row begins at the first quarter horizontal or vertical position in the list of text strings.
- **50%** - The column or row begins or ends at the midpoint horizontal or vertical position in the list of text strings.
- **75%** - The column or row begins or ends at the third quarter horizontal or vertical position in the list of text strings.
- **100%** - The column or row begins or ends at the right-most horizontal or bottom-most vertical position in the list of text strings.

The length of the displayed bar is in proportion to the percentage of columns or rows displayed relative to their associated maximums.

The operator may left click on the gauge to reposition the scrolled text offset anywhere between its 0% and 100% limits.

Sample Horizontal Gauge**Scrolled Text**

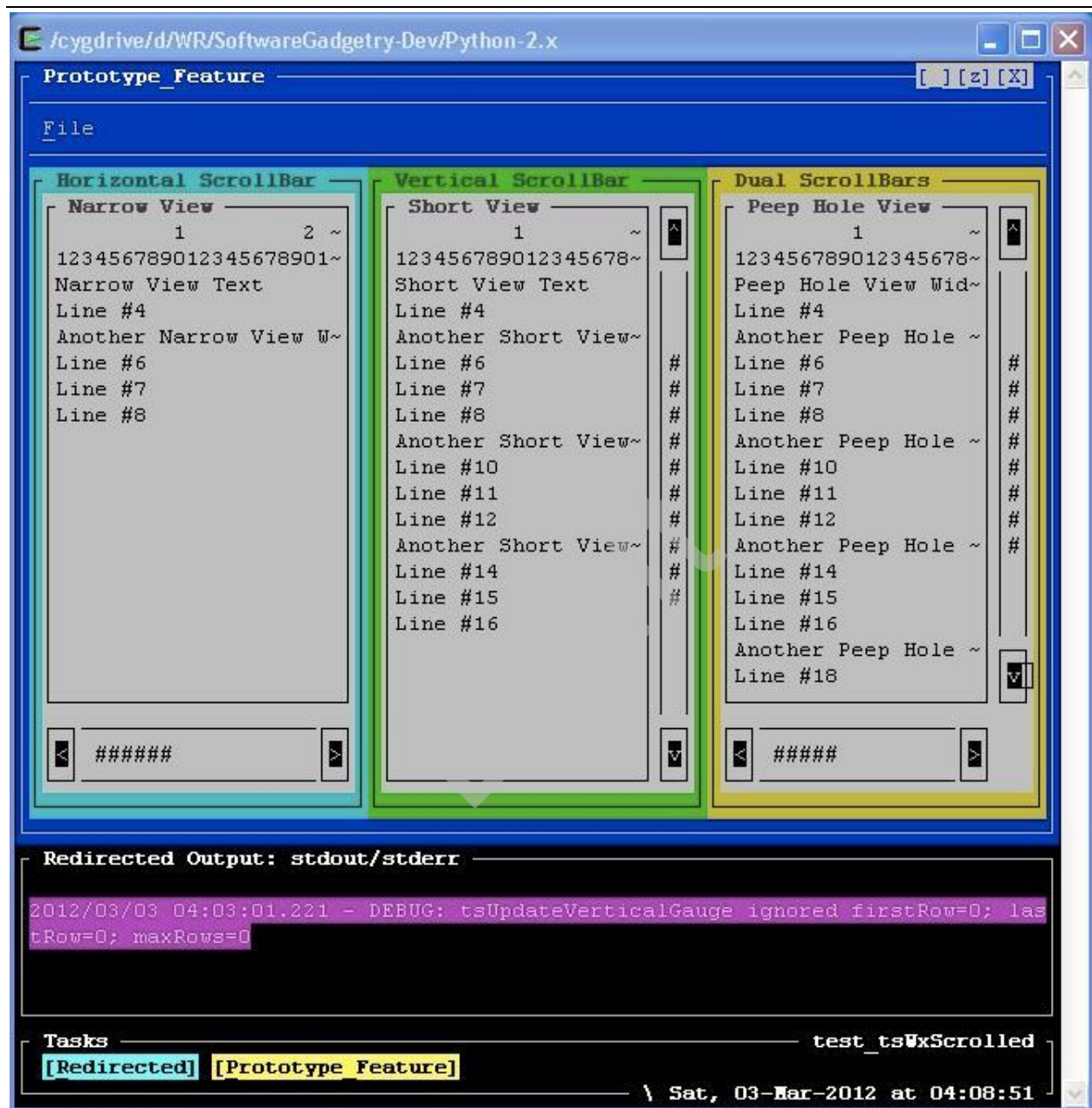
0%		+-----+-----+-----+-----+	0% Offset
			Empty 0% max buffer
		+-----+-----+-----+-----+	
0%		#####	0% Offset
			First 25% max buffer
		+-----+-----+-----+-----+	
25%		#####	25% Offset
			Second 25% max buffer
		+-----+-----+-----+-----+	



ScrollBar - Composite of two ScrollBarButtons and either one horizontal or one vertical ScrollBarGauge.

ScrolledText - A text control that allows an array of text strings to be scrolled horizontally and/or vertically. It may be scrolled in increments of single or multiple columns (via a mouse left-button single click), rows (via a mouse left-button rapid double-click) or pages (via a mouse left-button rapid triple-click).

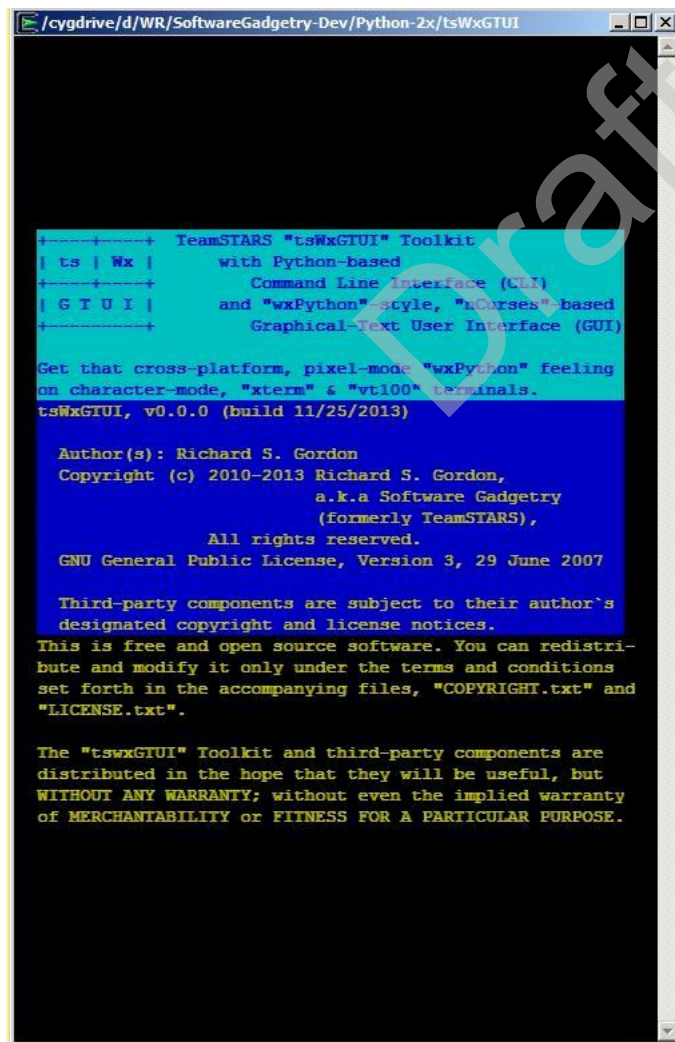
ScrolledWindows - Composite of various ScrollBarButtons, ScrollBarGauge(s) and ScrolledText window(s) as depicted in the following figure.



SplashScreen - A window with or without borders, displaying a "bitmap" (text) describing your application. The splash screen is shown during application initialization. The application then either explicitly destroys it or lets it time-out. Limitations:

- 1) Bitmap images cannot be displayed before the application starts. However, the "tsWxGraphicalTextUserInterface" module can display an existing bitmap image or create one only after it has initialized and started "nCurses".
- 2) Character-mode applications CAN create a bitmap image from only the "nCurses" handle associated with a single "wxPython" GUI object container. However, the image does not retain the foreground/background color attributes of any associated "wxPython" GUI objects in the container.
- 3) The "wxPython"-style, "nCurses"-based GUI emulations currently makes no provision for the application to delete GUI objects because of unresolved housekeeping issues.

The following screenshot captured a borderless but centered "tsWxGraphicalTextUserInterface" generated bitmap image.



The following screenshot captured an application program's boxsizer layout.



```
/cygdrive/d/WR/SoftwareGadgetry-Dev/Python-2x/tsWxGTUI

+-----+ TeamSTARS "tsWxGTUI" Toolkit
| ts | Wx |      with Python-based
+-----+      Command Line Interface (CLI)
| G T U I |      and "wxPython"-style, "nCurses"-based
+-----+      Graphical-Text User Interface (GUI)

Get that cross-platform, pixel-mode "wxPython" feeling
on character-mode, "xterm" & "vt100" terminals.

tsWxGTUI, v0.0.0 (build 11/25/2013)

Author(s): Richard S. Gordon
Copyright (c) 2010-2013 Richard S. Gordon,
              a.k.a Software Gadgetry
              (formerly TeamSTARS),
              All rights reserved.
GNU General Public License, Version 3, 29 June 2007

Third-party components are subject to their author's
designated copyright and license notices.

This is free and open source software. You can redistrib-
ute and modify it only under the terms and conditions
set forth in the accompanying files, "COPYRIGHT.txt" and
"LICENSE.txt".

The "tsWxGTUI" Toolkit and third-party components are
distributed in the hope that they will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty
of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Redirected Output: stdout/stderr
sWindow: self.ts_AssignedId=917
earliestAncestor=<tsLibGUI.tsWxPkg.src.tsWxSplashScreen.Sp
lashScreen object at 0x7f84b24c>
DescendantOrderOfShow=[799, 849, 873, 895, 917]

Tasks      test_tsWxSplashScreen
[Redirected] [Gui Test Units]
- Wed, 27-Nov-2013 at 05:45:47
```

StatusBars -

ToolBars -

TaskBar -

a) **tsWxGlobals**

Module to establish configuration constants and macro-type functions for the Graphical-style User Interface mode of the "tsWxGTUI_PyVx" Toolkit. Provides a centralized mechanism for modifying/restoring those configuration constants that can be interrogated at runtime by those software components having a "need-to-know". The intent being to avoid subsequent searches to locate and modify or restore a constant appropriate to the current configuration. Provides a theme-based mechanism for modifying / restoring those configuration constants.

b) **tsWxGraphicalTextUserInterface**

Class uses the Standard Python Curses API to initialize, manage and shutdown input, from a keyboard and mouse, and output, to a two-dimensional display screen. It identifies user terminal make, model and features. It controls terminal device startup, shutdown and exception handling. It translates "wxPython"-style terminal color, pixel and character parameters into their "Curses" counterparts. Upon startup, it briefly restores or creates, saves or loads the splash screen bitmap image as specified in the "tsWxGlobals" configuration file.

c) **tsWxMultiFrameEnv**

Class to enable an application using a Command Line Interface (CLI) to launch and use the same character-mode terminal with a Graphical-style User Interface (GUI). It uses application specified configuration keyword-value pair options to initialize any application specific logger(s). It wraps the CLI, underlying the GUI, and the GUI with exception handlers to control the exit codes and messages used to coordinate other application programs.

1.5.1.7 tsToolsGUI Capabilities

1 tsToolsGUI Functional Capabilities

TBD

1.5.1.8 tsTestsGUI Capabilities

1 tsTestsGUI Functional Capabilities

TBD

1.5.2 Limitations

This paragraph shall briefly state the unaccomplished interface and functional goals of the system and the software to which this document applies.

1.5.2.1 Platform Interface Limitations

- 1 Using "Python" technology, the toolkit enables applications to run, without modification, on platforms with 32-bit and 64-bit processor architectures under an add-on POSIX-compatible operating system environment for "Windows":
 - a) "Cygwin" is the recommended free and open source add-on, Linux-like Command Line Interface and GNU toolkit from Red Hat. Its 32-bit and 64-bit versions support "tsLibCLI" capabilities. Its "X Window System", "K Desktop Environment 3" and "GNOME" components support "tsLibGUI" capabilities.
 - b) "UWIN" is a project started at about the same time as "Cygwin". It too is an add-on, Linux-like Command Line Interface and GNU toolkit from AT&T. Individual source and binaries are available under the Open Source Eclipse Public License 1.0 at AT&T AST/UWIN open source download. Unlike Cygwin, it does not include the means to automatically download, install and configure a working system. It runs best on 32-bit versions of Windows NT/2000/XP/7 with the file system NTFS, but can run in degraded mode using FAT, and further degraded on Windows 95/98/ME. It has NOT been used with the "tsWxGTUI_PyVx" Toolkit but likely supports its "tsLibCLI" and "tsLibGUI" capabilities.
 - c) "GNUwin32" is an alternative free and open source add-on, Linux-like Command Line Interface and GNU toolkit from the Free Software Foundation. It only supports 32-bit processors. It supports "tsLibCLI" capabilities. Its "PDCurses" component does NOT yet support "tsLibGUI" capabilities.
- 2 Using "Python" and "Curses" technology, the toolkit enables applications to run, without modification, on platforms with terminals that include the following keyboard, multi-color electronic display and mouse configurations:
 - a) Linux Terminal with xterm, xterm-color, xterm-16color, xterm-256color emulation.
 - b) Mac OS X iTerm and Terminal with xterm, xterm-color, xterm-16color, xterm-256color emulation.
 - c) Microsoft Windows with cygwin xterm, xterm-color, xterm-16color, xterm-256color, cygwin mintty (xterm) emulation
 - d) UNIX (Solaris and OpenIndiana) Terminal with xterm, xterm-color, xterm-16color, xterm-256color emulation.
- 3 Using "Python" and "Curses" technology, the toolkit enables applications to run, without modification, on platforms with terminals that include the following keyboard, non-color electronic display configurations with/without mouse BUT "Hot-key" inputs are not yet supported:
 - a) Linux Terminal with non-color vt100 and vt220 emulation with/without mouse.
 - b) Mac OS X iTerm and Terminal with non-color vt100 and vt220 emulation with/without mouse.

- c) Microsoft Windows with cygwin console emulation with/without mouse or with non-color vt100 and vt220 emulation with/without mouse.
 - d) UNIX (PC-BSD, Solaris and OpenIndiana) Terminal with non-color vt100 and vt220 emulation with/without mouse.
- 4** Unsupported terminals include the following keyboard and color display:
- a) Linux Terminal with ansi and cygwin console emulation.
 - b) Mac OS X iTerm and Terminal with ansi and cygwin console emulation.
 - c) Microsoft Windows with ansi emulation.
 - d) UNIX (PC-BSD, Solaris and OpenIndiana) Terminal with ansi and cygwin console emulation.

1.5.2.2 tsLibGUI Functional Limitations

- 1** See: *Capabilities* (on page 57)
- 2** The "tsWxGTUI_PyVx" Toolkit has been designed to handle text of a uniform size (width x height). It has not been designed to handle double width, double height or Unicode characters. This does not preclude a future design change.

1.5.3 Comparison of wxPython with tsWxGTUI (Development Plan)

This tabulation shall briefly compare the features, capabilities and limitations of the pixel-mode "wxPython" / "wxWidgets" / "wxEmbedded" Toolkit with those of its character-mode emulator, the "tsWxGTUI_PyVx" Toolkit.

Feature	"wxPython" / "wxWidgets" / "wxEmbedded"	"tsWxGTUI_PyVx"
Platform	Locally (via system console or xterm) and remotely (via vnc) accessible systems: <ul style="list-style-type: none"> Desktop Laptop Workstation Embedded (via KOAN's "wxEmbedded") 	Locally (via system console or xterm) and remotely (via xterm) accessible systems: <ul style="list-style-type: none"> Desktop Laptop Workstation Embedded
Operator Desktop Interface Hardware	<ul style="list-style-type: none"> Keyboard Mouse, trackball, touchpad or touchscreen Optional Mouseless Display (pixel mode) 	<ul style="list-style-type: none"> Keyboard Mouse, trackball, touchpad or touchscreen Keyboard Shortcut Key with/without Optional Mouse (Future) Display (character mode)
Operating System & Device Driver Software	<ul style="list-style-type: none"> Linux Mac OS X Microsoft Windows Unix 	<ul style="list-style-type: none"> Linux Mac OS X Microsoft Windows (with free add-on of POSIX-like Cygwin Command Line Interface and GNU tools from Red Hat)

Feature	"wxPython" / "wxWidgets" / "wxEmbedded"	"tsWxGTUI_PyVx"
Terminal Device Interface Software	<ul style="list-style-type: none"> ▪ "wxWidgets" internally uses host Operating System specific API for its Terminal Device Interface. ▪ It translates host Operating System specific events into "wxWidget" specific published API events. ▪ Events include keyboard key press / release, mouse button press / release and single / double click with mouse position at every clocked sample. ▪ Events also include window resizing. 	<ul style="list-style-type: none"> ▪ Unix ▪ "nCurses" internally uses host Operating System specific API for its Terminal Device Interface. ▪ It translates host Operating System specific events into "nCurses" specific published API events. ▪ Events include keyboard key press / release, mouse button press / release and single / double / triple click with mouse position ONLY available at time of button state change. ▪ Events also include window resizing. However, event handling is currently limited to trapping the unsupported event. Though re-initializing "nCurses" to detect and use the new size is feasible and fast (50 milliseconds or more depending on host processor and terminal color palette (and associated definition of or mapping of wxPython color palette), it does not address the time consuming (20 seconds or more on host processor) complexities associated with re-initializing the "wxPython" emulation and the System Operator designated application program.
Programming Language	<ul style="list-style-type: none"> ▪ "wxWidgets" is implemented in C++ ▪ "wxPython" binding/wrapper for "wxWidgets" is implemented with SWIG in Python 2.x and Python 3.x 	<ul style="list-style-type: none"> ▪ "tsWxGTUI_PyVx" is implemented in Python 2.x ▪ After debugging, "tsWxGTUI_PyVx" is ported from Python 2.x to Python 3.x via the standard Python "2to3" utility with minor manual debugging when appropriate
Terminal Interface Software	<ul style="list-style-type: none"> ▪ "wxWidgets" internally uses Operating System or X window system specific API for Graphical User Interface. 	<ul style="list-style-type: none"> ▪ "tsWxGTUI_PyVx" internally uses Python Curses ("nCurses") module API for Graphical-style User Interface. The Python Curses module only implements the most commonly used subset features of the "nCurses" API. ▪ The "tsWxGTUI_PyVx" Toolkit emulates a typical Operating System or X window system specific API for Graphical User Interface.

Feature	"wxPython" / "wxWidgets" / "wxEmbedded"	"tsWxGTUI_PyVx"
Operator Desktop Interface Software	<ul style="list-style-type: none"> Host Operating System specific Command Line Interface is a POSIX compatible shell (such as bash) Host Operating System specific Graphical User Interface features support multiple independently re-sizable and repositionable Frames and Dialogs Host Operating System specific Graphical User Interface features reflect proprietary placement of labels and buttons to iconize, maximize/restore and close frames and dialogs Host Operating System specific Graphical User Interface task bar features support the closing of individual Frames and Dialogs Host Operating System specific task bar features support the shifting of foreground focus from one Frame or Dialog to another 	<ul style="list-style-type: none"> Host Operating System specific Command Line Interface is a POSIX compatible shell (such as bash) Host Operating System specific Graphical User Interface shell features support multiple independently re-sizable and repositionable Command Line Interface shell windows "tsWxGTUI_PyVx" emulated Graphical User Interface features DO NOT support multiple independently re-sizable and repositionable Frames and Dialogs WITHOUT the System Operator first creating separate Command Line Interface shell windows "tsWxGTUI_PyVx" emulated Graphical User Interface features reflect Microsoft Windows-style placement of labels and buttons to iconize, maximize/restore and close frames and dialogs "tsWxGTUI_PyVx" emulated Graphical User Interface task bar features DO NOT currently support the closing of individual Frames and Dialogs "tsWxGTUI_PyVx" Toolkit task bar features (future enhancement) support the shifting of foreground focus from one Frame or Dialog to another
Application Programming Interface	<ul style="list-style-type: none"> "wxWidgets" / "wxPython" API for Graphical User Interface supports bit-mapped images. It supports fixed and variable sized fonts and at least the 68 most commonly used colors. 	<ul style="list-style-type: none"> "tsWxGTUI_PyVx" emulated subset of "wxWidgets" / "wxPython" API for Graphical-style User Interface DOES NOT support bitmapped images except for the single, predefined bitmapped image used as a splash screen at startup It supports the 1-color (single terminal-dependant green, orange or white phosphor) that is "ON" or "OFF" (black) associated with a non-color, VT100/VT220 terminal hardware and terminal emulator software. It supports a single fixed sized font and at least the 68 most commonly used colors (which are either internally mapped into the "nCurses" standard 8-color, 16-color or defined for optional 88-color and 256-color terminal hardware and terminal emulator software). It supports a 71-color palette by augmenting the 68-color wxPython palette with the 3 colors that must be supported for the xterm-16color palette.

Feature	"wxPython" / "wxWidgets" / "wxEmbedded"	"tsWxGTUI_PyVx"
		<ul style="list-style-type: none">▪ It supports a 140-color palette by augmenting the 68-color wxPython palette with the 3 colors unique to the xterm-16color palette plus 69 other colors that demonstrate the customization potential. However, the constraint on the number of color pairs (32767) constrains the number of colors to be no more than 181 (square root of 32768) unless the design is re-designed to use a sparse matrix to avoid impractical color combinations.

1.5.4 Benefits

This paragraph shall briefly state the benefits of using the system and the software to which this document applies.

The toolkit improves the productivity of Software Engineers and System Operators by facilitating the creation and use of "user-friendly" applications.

- 1 Command Line Interface (tsToolkitCLI)** - Includes building blocks that create a sophisticated POSIX-like terminal interface featuring:
 - a) Exception Handling
 - b) Logging
 - c) Launching, event dispatching and terminating of the Graphical-style User Interface
- 2 Graphical-style User Interface (tsToolkitGUI)** - Includes building blocks that create a sophisticated Desktop, Laptop and Workstation Computer-like terminal interface featuring:
 - a) Tiled and overlapped windows
 - b) Box and grid sizers
 - c) Buttons
 - d) Check boxes
 - e) Horizontal and vertical gauges
 - f) Menu bars and menus
 - g) Radio boxes and buttons
 - h) Scrolled windows with associated scrollable text window and scrollbars with associated gauge and scroll control buttons
 - i) Redirected output window to annotate date, time and severity levels to system and application messages printed or sent to stderr and stdout
 - j) Splash screen

- k) Multi-field status bars
- l) Tool bars
- m) Task bar

3 Improved Productivity - The general-purpose, re-usable building blocks enable:

- a) The application developer to focus on the application specific functionality and not waste effort re-inventing the functionality typical of Command Line and Graphical User Interfaces.
- b) "wxPython" pixel-mode application programs run with little, if any, change. Changes are limited to eliminating use of such pixel-mode, bit-mapped image features as icons, proportional sized fonts and curved line drawing features.

Draft

Draft

1.6 APPENDIX E - INHERITED, FIELD-PROVEN COMPUTER TECHNOLOGY

The "tsWxGTUI_PyVx" Toolkit applies pre-existing, computer technology that has been field proven over many years.

Over the years, various individuals and organizations have contributed software to facilitate development of applications with "user friendly" interfaces:

- 1** *VGA-compatible Text Mode* (on page 78)
- 2** *Command Line Interface (CLI)* (on page 94)
 - a) *Shell Definition & History* (on page 98)
 - b) *Text-based User Interface (TUI)* (on page 101)
 - c) *POSIX Terminal Device Interface (TDI)* (see "*POSIX Terminal Device Interface (TDI)*" on page 105)
- 3** *Graphical User Interface (GUI)* (on page 107)
 - a) *Graphical Device Interface (GDI)* (on page 107)
 - b) *Widget Toolkit* (on page 108)
- 4** *Python Programming Language* (on page 117)

Data arrangement[edit]**Text buffer[edit]**

Each screen character is actually represented by two bytes aligned as a 16-bit word accessible by CPU at a single operation. The lower, or character byte is the actual code point for the current character set, and the higher, or attribute byte is a bit field used to select various video attributes such as color, blinking, character set, and so forth.[1] This byte-pair scheme is among the features that VGA inherited from EGA and CGA and ultimately from MDA.

Attribute								Character							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Blink	Background Color			Foreground Color				Code Point							

Numeration of bits given in a way usually used in technical documentation.

- 1** ^ Depending on mode setup, attribute bit 7 may be either the blink bit or the fourth background color bit (which allows to use all 16 colors).
- 2** ^ Attribute bit 3 also chooses between fonts A and B (see below), therefore if these font are not the same, this bit is simultaneously an additional code point bit.

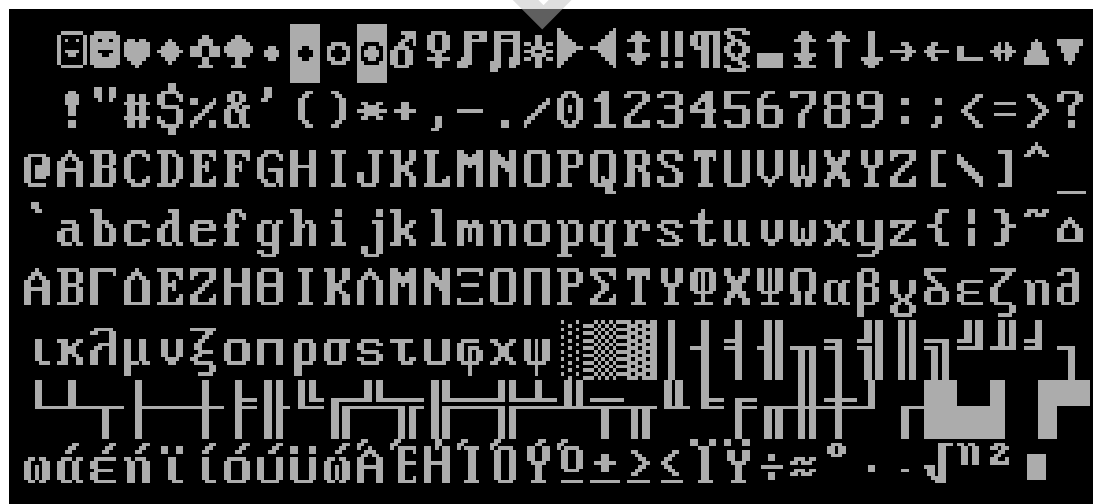
Colors are assigned in the same way as in 4-bit indexed color graphic modes, see detailed description of VGA palette. VGA modes have no need in reverse and bright attributes because foreground/background colors can be set explicitly. The VGA hardware has an ability to enable underline on some foreground/background combinations,[1] normally disabled in color modes, therefore an underline feature in a color text mode normally is unavailable. Underline is used, though, in monochrome (MDA-like) text modes, those text buffer has the same arrangement as above.

Fonts[edit]

Screen fonts used in EGA and VGA are monospace raster fonts containing 256 glyphs with 8 dots glyph width and a some glyph height fixed for each font, less or equal to 32. Each row of a glyph is coded in a 8 bit byte, with high bits at the left and low at the right. Along with several hardware-dependent fonts stored in the adapter's ROM, the text mode offers 8[1] loadable font storages. Two active font pointers (font A and font B) choose each one of available fonts, they usually (but not necessarily) point to the same font. When font A \neq font B, the attribute bit 3 (see above) acts both as a foreground color bit and as the ninth (28) code point bit. To make a 512 character set mode (instead of the common 256), available colors should be halved from 16 to 8.

There are modes with 9-dots character box width (e.g. the default 80×25 mode), where an extra column of a character matrix appears, not accessible directly. It may be left empty (all 0s), but may be enabled a special processing of characters with code points 0xB0–0xDF, which are usually box drawing. In this so named Line Graphics Enable mode,[1] that extra column become a duplicate of the rightmost addressable column (encoded by a least significant bit in each row) for 0xB0–0xDF characters, and left empty for the rest of characters. For this reason, placing letter-like characters to code points 0xB0–0xDF should be avoided.

Norton Utilities 6.01, an example of advanced TUI which redefines the character set to show tiny graphical widgets, icons and an arrow pointer in text mode.



Cursor[edit]

A shape of the cursor is restricted to rectangle which width occupies all character box and filled by the foreground color of current character box. Its height and position may be arbitrary within a character box;[2] notably, the cursor may be hidden (invisible). There is also a bit which controls cursor blink activity.[citation needed]

A mouse cursor in TUI (when implemented) is not usually the same thing as a hardware cursor, but a moving rectangle with altered background or a special glyph.

Some text-based interfaces, such as that of Impulse Tracker, went to even greater lengths to provide a smoother and more graphic-looking mouse cursor. This was done by constantly re-generating character glyphs in realtime according to the cursor's on-screen position and the underlying characters.[citation needed]

Mouse cursor in Impulse Tracker



Access methods[edit]

There are generally two way to access a VGA text for an application: through a text terminal emulation (usually by OS) or directly via memory mapped I/O. The latter method also allows reading of text buffer, for which reason it is preferred for advanced TUI programs.

From the point of view of 16-bit x86 application or system software (including BIOS), the text buffer is just a region of RAM. Its range is 0xB8000–0xBFFFF (a half of segment B800h). The text buffer data can be read and written, bitwise operations may be applied to them. A part of text buffer memory above the scope of the current mode is accessible, but is not shown.

The same physical addresses are used in CPU's protected mode: applications may either have this part of memory mapped to their address space or access it via the operating system. When an application (on a modern multitasking OS) does not have control over the console, it accesses a part of mainboard RAM instead of actual text buffer.

For computers of 1980s, very fast access to text buffer was extremely useful for a fast UI; even on relatively modern hardware an overhead of text mode emulation via hardware APA modes may be noticeable.

Modes and timings[edit]

Video signal[edit]

From the monitor's side, there is no difference in input signal in a text mode and an APA mode of the same size. A text mode signal may have the same timings than VESA standard modes. Same registers are used on adapter's side to set up these parameters in a text mode as in APA modes. Text mode output signal is essentially the same as in graphic modes, but its source is text buffer and character generator, not framebuffer as in APA.

PC common text modes[edit]

Depending on the graphics adapter used, a variety of text modes are available on IBM PC compatible computers. They are listed on the table below:

Text res.	Char. size	Graphics res.	Colors	Adapters
80×25	9×14	720×350	B&W Text	MDA, Hercules
40×25	8×8	320×200	16 colors	CGA, EGA
80×25	8×8	640×200	16 colors	CGA, EGA
80×25	8×14	640×350	16 colors	EGA
80×43	8×8	640×350	16 colors	EGA
80×25	9×16	720×400	16 colors	VGA
80×50	9×8	720×400	16 colors	VGA
80×60			16 colors	VESA-compatible Super VGA

132×25			16 colors	VESA-compatible Super VGA
132×43			16 colors	VESA-compatible Super VGA
132×50			16 colors	VESA-compatible Super VGA
132×60			16 colors	VESA-compatible Super VGA

VGA and compatible cards support MDA, CGA and EGA modes. All colored modes have the same design of text attributes. MDA modes have some specific features (see above) – a text could be emphasized with bright, underline, reverse and blinking attributes.

By far the most common text mode used in DOS environments, and initial Windows consoles, is the default 80 columns by 25 rows, or 80×25, with 16 colors. This mode was available on practically all IBM and compatible personal computers.

Two other VGA text modes, 80×43 and 80×50, exist but were very rarely used. The 40 column text modes were never very popular, and were used only for demonstration purposes or with very old hardware.

Character sizes and graphical resolutions for the extended VESA-compatible Super VGA text modes are manufacturer's dependent. Some cards (e.g. S3) supported custom very large text modes, like 100×37 or even 160×120. Like as in graphic modes, graphic adapters of 2000s commonly are capable to set up an arbitrarily-sized text mode (in reasonable limits) instead of choosing its parameters from some list. But poor software support deters widespread use of such custom modes.

SVGATextMode[edit]

On Linux and DOS systems with so named SVGA cards, a program called SVGATextMode[3] is used to set up better looking text modes than EGA and VGA standard ones. This is particularly useful for large ($\geq 17''$) monitors, where normal VGA 400 lines text mode appear as extremely low resolution. SVGATextMode allows setting of the pixel clock and higher refresh rate, larger font size, cursor size, etc., and allows a better use of the potential of a video card and monitor. In non-Windows systems, the use of SVGATextMode (or alternative options such as the Linux framebuffer) to obtain a sharp text is critical for LCD monitors of 1280×1024 (or higher resolution) because none of so named standard text modes fits to this matrix size. SVGATextMode also allows a fine tuning of video signal timings.

Despite the name of this program, only a few of its supported modes conform to SVGA (i.e. VESA) standards.

General restrictions[edit]

Such VGA text modes have some hardware-imposed limitations. Because some of them appear now too restrictive, the hardware text mode on VGA compatible video adapters has only a limited use.

Parameter	Original VGA	Modern video adapters	Remarks
Character cell (glyph) width	8 or 9 dots[1]	≤ 9 dots	Not all hardware support glyphs narrower than 8 dots;

Character cell (glyph) height	≤ 32 dots		Even width=9 looks ugly on high resolutions, particularly for people with hyperopia, and is insufficient for East Asian scripts. Height=32 is more than sufficient.
Number of character cells	At least 4,000 (reached at 80×50)	$\leq 16,384 = 214$ (memory addressing limitations)	A modern adapter, if supports non-standard modes, may produce a reasonably dense text screen even on a large monitor.
Width in character cells (characters per line)	At least 80	$\leq 256(?)$	
Height in character cells (number of lines)	At least 50 (reached at 80×50)		
Code page size (number of different glyphs displayed simultaneously)	$\leq 512 = 29$ (if font A \neq font B)		Even 512 is insufficient for comprehensive Unicode support.
	$\leq 256 = 28$ (if font A = font B)		
Number of colors	foreground: 16* background: 8 or 16**		16 of arbitrarily chosen colors, not fixed.

* 8 colors may be used by font A and other 8 colors by font B; so, if font A \neq font B (512 characters mode), then the palette should be halved and a text may effectively use only 8 colors.

** Normally, first 8 colors of the same palette. If blink is disabled, then all 16

1.6.2 POSIX

From Wikipedia, the free encyclopedia:

"POSIX (/ˈpɒzɪks/ POZ-iks), an acronym for "Portable Operating System Interface",[1] is a family of standards specified by the IEEE for maintaining compatibility between operating systems. POSIX defines the application programming interface (API), along with command line shells and utility interfaces, for software compatibility with variants of Unix and other operating systems.[2][3]

Name

Originally, the name "POSIX" referred to IEEE Std 1003.1-1988, released in 1988. The family of POSIX standards is formally designated as IEEE 1003 and the international standard name is ISO/IEC 9945.

The standards, formerly known as IEEE-IX, emerged from a project that began circa 1985. Richard Stallman suggested the name POSIX to the IEEE. The committee found it more easily pronounceable and memorable, so the committee adopted it.[2][4]

Overview

The POSIX specifications for Unix-like operating system environments originally consisted of a single document for the core programming interface, but eventually grew to 19 separate documents (for example, POSIX.1, POSIX.2 etc.) [1]. The standardized user command line and scripting interface were based on the Korn shell[citation needed]. Many user-level programs, services, and utilities including awk, echo, ed were also standardized, along with required program-level services including basic I/O (file, terminal, and network) services. POSIX also defines a standard threading library API which is supported by most modern operating systems. Nowadays, most of POSIX parts are combined into a single standard, IEEE Std 1003.1-2008, also known as POSIX.1-2008.

As of 2009, POSIX documentation is divided in two parts:

- POSIX.1-2008: POSIX Base Definitions, System Interfaces, and Commands and Utilities (which include POSIX.1, extensions for POSIX.1, Real-time Services, Threads Interface, Real-time Extensions, Security Interface, Network File Access and Network Process-to-Process Communications, User Portability Extensions, Corrections and Extensions, Protection and Control Utilities and Batch System Utilities)
- POSIX Conformance Testing: A test suite for POSIX accompanies the standard: PCTS or the POSIX Conformance Test Suite.[5]

The development of the POSIX standard takes place in the Austin Group, a joint working group linking the Open Group and the ISO organization.

Versions

Parts before 1997

Before 1997, POSIX comprised several standards:

POSIX.1

- POSIX.1, Core Services (incorporates Standard ANSI C) (IEEE Std 1003.1-1988)
 - Process Creation and Control
 - Signals
 - Floating Point Exceptions
 - Segmentation / Memory Violations
 - Illegal Instructions
 - Bus Errors
 - Timers
 - File and Directory Operations
 - Pipes
 - C Library (Standard C)
 - I/O Port Interface and Control
 - Process Triggers

POSIX.1b

- POSIX.1b, Real-time extensions (IEEE Std 1003.1b-1993)
 - Priority Scheduling
 - Real-Time Signals
 - Clocks and Timers
 - Semaphores
 - Message Passing
 - Shared Memory
 - Asynch and Synch I/O
 - Memory Locking Interface

POSIX.1c

- POSIX.1c, Threads extensions (IEEE Std 1003.1c-1995)
 - Thread Creation, Control, and Cleanup
 - Thread Scheduling
 - Thread Synchronization
 - Signal Handling

POSIX.2

- POSIX.2, Shell and Utilities (IEEE Std 1003.2-1992)
 - Command Interpreter
 - Utility Programs

Draft

Versions after 1997

After 1997, the Austin Group developed the POSIX revisions. The specifications are known under the name Single UNIX Specification, before they become a POSIX standard when formally approved by the ISO.

POSIX.1-2001

POSIX.1-2001 or IEEE Std 1003.1-2001 equates to the Single UNIX Specification version 3[6]

- This standard consisted of:
- the Base Definitions, Issue 6,
- the System Interfaces and Headers, Issue 6,
- the Commands and Utilities, Issue 6.

POSIX.1-2001 (with two TCs)

IEEE Std 1003.1-2004 involved a minor update of POSIX.1-2001. It incorporated two technical corrigenda.[7] Its contents are available on the web.[8]

POSIX.1-2008

As of 2009 POSIX.1-2008 or IEEE Std 1003.1-2008 represents the current version.[9][10] A free online copy is available.[11]

This standard consists of:

- the Base Definitions, Issue 7,
- the System Interfaces and Headers, Issue 7,
- the Commands and Utilities, Issue 7.

Controversies

512- vs 1024-byte blocks

POSIX mandates 512-byte block sizes for the `df` and `du` utilities, reflecting the default size of blocks on disks. When Richard Stallman and the GNU team were implementing POSIX for the GNU operating system, they objected to this on the grounds that most people think in terms of 1024 byte (or 1 KiB) blocks. The environment variable `POSIXLY_CORRECT` was introduced to allow the user to force the standards-compliant behaviour.[12] The variable name `POSIX_ME_HARDER` was also discussed.[13]

POSIX-oriented operating systems

Depending upon the degree of compliance with the standards, one can classify operating systems as fully or partly POSIX compatible. Certified products can be found at the IEEE's website.[14]

Fully POSIX-compliant

The following operating systems conform (i.e., are 100% compliant) to one or more of the various POSIX standards.

- A/UX
- AIX
- BSD/OS[citation needed]
- DSPnano[citation needed]
- HP-UX
- INTEGRITY
- IRIX
- LynxOS[citation needed]
- MPE/iX[citation needed]
- OS X[15]
- QNX[16]
- RTEMS (POSIX 1003.13-2003 Profile 52)
- Solaris
- Tru64
- Unison RTOS[citation needed]
- UnixWare[17]

Mostly POSIX-compliant

The following, while not officially certified as POSIX compatible, comply in large part:

- BeOS (and subsequently Haiku)
- FreeBSD[18]
- Contiki
- Darwin (core of OS X and iOS)
- illumos
- GNU/Linux (most distributions — see Linux Standard Base)
- MINIX (now MINIX3)
- NetBSD
- Nucleus RTOS
- OpenBSD
- OpenSolaris
- PikeOS RTOS for embedded systems with optional PSE51 and PSE52 partitions; see partition (mainframe)
- RTEMS – POSIX API support designed to IEEE Std. 1003.13-2003 PSE52
- Sanos
- SkyOS
- Syllable

- VSTa
- VxWorks[16] (VxWorks is often used as a shell around non-posix Kernels i.e. TiMOS/SROS)

POSIX for Windows

- Cygwin provides a largely POSIX-compliant development and run-time environment for Microsoft Windows.
 - MinGW, formerly a fork of Cygwin, provides a less POSIX-compliant development environment and supports compatible C-programmed applications via Msvcrt, Microsoft's old Visual C runtime library.

Draft

- Microsoft POSIX subsystem, an optional Windows subsystem included in Windows NT-based operating systems up to Windows 2000. POSIX-1 as it stood in 1990 revision — no threads, no sockets.
- Interix, originally OpenNT by Softway Systems, Inc., is an upgrade and replacement for Microsoft POSIX subsystem that was purchased by Microsoft in 1999. It was initially marketed as a stand-alone add-on product and then later included it as a component in Windows Services for UNIX (SFU) and finally incorporated it as a component in Windows Server 2003 R2 and later Windows OS releases under the name "Subsystem for UNIX-based Applications" (SUA); later made deprecated in 2012 (Windows 8)[19] and dropped in 2013 (2012 R2, 8.1). It enables full POSIX compliance for certain Microsoft Windows products[citation needed].
- UWIN from AT&T Research implements a POSIX layer on top of the Win32 APIs.
- MKS Toolkit, originally created for MS-DOS, is a software package produced and maintained by MKS Inc. that provides a Unix-like environment for scripting, connectivity and porting Unix and Linux software to both 32- and 64-bit Microsoft Windows systems. A subset of it was included in the first release of Windows Services for UNIX (SFU) in 1998.[20]

POSIX for OS/2

Mostly POSIX compliant environments for OS/2:

- emx+gcc – largely POSIX compliant

POSIX for DOS

Partially POSIX compliant environments for DOS include:

- emx+gcc – largely POSIX compliant
- DJGPP – partially POSIX compliant

Compliant via compatibility feature

The following are not officially certified as POSIX compatible, but they conform in large part to the standards by implementing POSIX support via some sort of compatibility feature, usually translation libraries, or a layer atop the kernel. Without these features, they are usually noncompliant.

- eCos – POSIX is part of standard distribution, and used by many applications. 'external links' section below has more information.
- MorphOS (through the built-in ixemul library)
- OpenVMS (through optional POSIX package)
- OpenVOS is an optional POSIX-compliant layer atop the Stratus VOS kernel[21]
- Plan 9 from Bell Labs APE - ANSI/POSIX Environment[22]
- Symbian OS with PIPS (PIPS Is POSIX on Symbian)
- Windows NT kernel when using Microsoft SFU 3.5 or SUA
 - Windows 2000 Server or Professional with Service Pack 3 or later. To be POSIX compliant, one must activate optional features of Windows NT and Windows 2000 Server.[23]
 - Windows XP Professional with Service Pack 1 or later
 - Windows Server 2003

- Windows Vista (Ultimate / Enterprise)
- Windows 7 (Ultimate / Enterprise)
- z/OS

Draft

1.6.3 Command Line Interface (CLI)

From Wikipedia, the free encyclopedia:

"A command-line interface (CLI), also known as command-line user interface, console user interface,[1] and character user interface (CUI), is a means of interacting with a computer program where the user (or client) issues commands to the program in the form of successive lines of text (command lines).

The CLI was the primary means of interaction with most popular operating systems in the 1970s and 1980s, including MS-DOS, CP/M, Unix, and Apple DOS. The interface is usually implemented with a command line shell, which is a program that accepts commands as text input and converts commands to appropriate operating system functions.

Command-line interfaces to computer operating systems are less widely used by casual computer users, who favor graphical user interfaces. Command-line interfaces are often preferred by more advanced computer users, as they often provide a more concise and powerful means to control a program or operating system.

Programs with command-line interfaces are generally easier to automate via scripting.

Alternatives to the command line include, but are not limited to menus (see IBM AIX SMIT for example), keyboard shortcuts, and various other desktop metaphors centered on the pointer (usually controlled with a mouse)."

* * *

"Operating system (OS) command line interfaces are usually distinct programs supplied with the operating system.

A program that implements such a text interface is often called a command-line interpreter, command processor or shell. The term 'shell', often used to describe a command-line interpreter, can be in principle any program that constitutes the user-interface, including fully graphically oriented ones—for example, the default Windows GUI is created by a shell program named EXPLORER.EXE, as defined in the SHELL=EXPLORER.EXE line in the WIN.INI configuration file.

Examples of command-line interpreters include the various Unix shells (sh, ksh, csh, tcsh, bash, etc.), the historical CP/M CCP, and MS-DOS/IBM-DOS/DR-DOS's COMMAND.COM, as well as the OS/2 and the Windows CMD.EXE programs, the latter groups being based heavily on DEC's RSX and RSTS CLIs. Under most operating systems, it is possible to replace the default shell program with alternatives; examples include 4DOS for DOS, 4OS2 for OS/2, and 4NT or Take Command for Windows."

* * *

"Application programs (as opposed to operating systems) may also have command line interfaces.

An application program may support none, any, or all of these three major types of command line interface mechanisms:

- Parameters: Most operating systems support a means to pass additional information to a program when it is launched. When a program is launched from an OS command line shell, additional text provided along with the program name is passed to the launched program.
- Interactive command line sessions: After launch, a program may provide an operator with an independent means to enter commands in the form of text.
- OS inter-process communication: Most operating systems support means of inter-process communication (for example; standard streams or named pipes). Command lines from client processes may be redirected to a CLI program by one of these methods."

Draft

1.6.3.1 Operating System Shell

From Wikipedia, the free encyclopedia:

"An operating system shell is a software component that presents a user interface to various operating system functions and services. Thus, it is nearly synonymous with "operating system user interface".[1] The shell is so called because it is an outer layer of interface between the user and the innards of the operating system (the kernel).[2]

Purpose

The services that an operating system provides to its user(s) include, but are not limited to, the following:

- primary purpose is to interpret commands.
- File management
- Process management – running and terminating applications
- Batch processing
- Operating system monitoring
- Operating system setup

Functioning

Most OS shells are not direct interfaces to the kernel, even if communicate with user via peripheral devices attached to the computer directly. Shells are actually special applications which use the kernel API in just the same way as it is used by other application programs. A shell manages the user–system interaction by prompting user(s) for input, interpreting their input, and then handling an output from the operating system.[1]

Since the OS shell is actually an application, it may easily be replaced with other similar program, for most OSes.

Remote shell

There are different approaches to remote access to an operation system, which sometimes also referred to as remote administration. The classical approach of multi-user mainframes is to provide text-based UI for each active user simultaneously by means of a text terminal connected to the mainframe via serial line or modem. This approach is now associated with Unix-like systems. Now, the Secure Shell protocol is used for a text-based UI, and for also GUI, if required, through SSH tunnelling and X Window System networking capabilities.

Likewise, a remote GUI is possible for Microsoft Windows with Remote Desktop Protocol.

Alternative approach, for GUI shells, is a desktop environment controlled both locally and remotely, such as Radmin and Windows Desktop Sharing.

In any case, a shell-level remote access provides much more essential access to the computer than client–server protocols usually do. This implies additional security threats.

Design

Most operating system shells fall into one of two categories: command-line and graphical. Command line shells provide a command-line interface (CLI) to the operating system, while graphical shells provide a graphical user interface (GUI). Other possibilities, although not so common, include voice UI and various implementations of a text-based user interface (TUI) which are not CLI.

The relative merits of CLI- and GUI-based shells are often debated. CLI proponents claim that certain operations can be performed much faster under CLI shells than under GUI shells (such as moving files[citation needed], for example). However, GUI proponents advocate the comparative usability and simplicity of GUI shells. The best choice is often determined by the way in which a computer will be used. On a server mainly used for data transfers and processing with expert administration, a CLI is likely to be the best choice. On the other hand, a GUI would probably be more appropriate for a computer to be used for secretarial work.

Command-line OS shells

Command-line OS interfaces were dominant when resources, such as primary memory and CPU performance, were scarce. In such systems as Unix and DOS command shells were centered on file system operations, although the classical Unix shell had also considerable process management capabilities. Many OS command-line shells actually became command language interpreters. This is the case, most notably, of numerous Unix shell variants.

As for other command-line interpreters, the use of an OS CLI does not imply actual text mode display. The use of command-line within a text window controlled by a GUI window manager is, as of 2012, quite common, if not dominant; see text-based user interface for details. Also shells, especially text-based ones, are often used remotely, as explained above.

Graphical (GUI) shells

A graphical user interface is possible as an extension of an operating system which traditionally uses CLI. In this case the GUI is referred to as a graphical interface or "desktop environment". But certain OSes use a GUI shell as a primary user interface.[3] As of 2012, it is common and perfectly normal that a desktop OS has both command-line and GUI shells (GUI environment). Also, GUI shells usually incorporate some features of the command-line interpreter, especially its command language.

Modern versions of Microsoft's Windows operating system utilize and only officially support Windows Explorer as their GUI shell.[4] Explorer provides the familiar desktop environment, start menu, and task bar, as well as the file management functions of the operating system. Older versions also include Program Manager which was the shell for the 3.x series of Microsoft Windows.

Many individuals and developers dissatisfied with the interface of Windows Explorer have developed software that either alters the functioning and appearance of the shell or replaces it entirely. WindowBlinds by Stardock is a good example of the former sort of application. LiteStep, GeoShell and FlyakiteOSX are good examples of the latter. This does not affect GUI applications (except for some possible changes in window manager behaviour), but provides a redesign of the interface to OS.

Proponents of Unix-like systems often argue that the GUI under UNIX is separate from the OS itself, unlike Microsoft Windows or MacOS.[5]

List of GUI shells

Microsoft Windows environments:
 Windows shell (a.k.a. Explorer.exe)
 Litestep
 GeoShell
 BB4Win
 Emerge Desktop
 SharpEnviro
Macintosh Finder
X Window System-based environments (primarily for Unix):
 Xfce
 LXDE
 MATE
 KDE
 GNOME
 Blackbox
 Common Desktop Environment
DOS Shell
AmigaOS environments.
 Workbench (GUI-Shell capabilities added since AmigaOS 2.0)
 Ambient (for MorphOS)
 Directory Opus
 ScalOS
 Wanderer (for AROS)

Terminology

Eric S. Raymond acknowledges a considerable confusion about etymology and different modern usages of the word "shell".[2] Although it is usually associated with command lines in the IT world, its usage for graphical environments is not uncommon too, especially for OS platforms where GUI historically was a primary user interface.[4][6][7] These systems may refer to its CLI shell, if present, as to command(-line) shell.[8] Some sources[3] distinguish between "GUI operating systems" (those which support a GUI natively) and GUI interfaces for operating systems.

Denotation of the term "shell" is usually restricted to few programs, including the main shell process,[4][8] which forms the core of the operating system's UI. The term "operating system user interface" may have a broader meaning, including some specialized components such as control panel."

1.6.3.1.1 Shell Definition & History

From <http://www.linfo.org/shell.html> (<http://www.linfo.org/shell.html>):

Created June 24, 2004. Last updated June 30, 2006.

Copyright © 2004 - 2006 The Linux Information Project. All Rights Reserved.

Shell Definition

A shell is a program that provides the traditional, text-only user interface for Linux and other Unix-like operating systems. Its primary function is to read commands that are typed into a console (i.e., an all-text display mode) or terminal window (an all-text window) in a GUI (graphical user interface) and then execute (i.e., run) them.

The term shell derives its name from the fact that it is an outer layer of an operating system. A shell is an interface between the user and the internal parts of the operating system (at the very core of which is the kernel).

A user is in a shell (i.e., interacting with the shell) as soon as that user has logged into the system. A shell is the most fundamental way that a user can interact with the system, and the shell hides the details of the underlying operating system from the user.

A shell prompt, also referred to as a command prompt is a character or set of characters at the start of the command line that indicates that the shell is ready to receive commands. It usually is, or ends with, a dollar sign (\$) for ordinary users and a pound sign (#) for the root (i.e., administrative) user. The term command line is sometimes used interchangeably with the shell prompt, because that is where the user enters commands. For example, instructions for performing some activity might say "Enter the following at the command line," which is the same as saying "Enter the following at the shell prompt." However, a command line is not a program but rather just the space to the right of a shell prompt.

Shells, although small in size, are sophisticated and powerful programs that are used for the following: program execution (i.e., launching), substitution of variables and of file names, input/output (I/O), redirection (i.e., sending the output from a program to a destination other than its default destination, including to be used as the input for another program), user environment control (e.g., changing the shell or the shell prompt), and serving as a programming language (i.e., a language that can be used to write shell scripts). Shells in Unix-like operating systems are unusual in that they are both an interactive command language (i.e., a language that a user can employ interactively to issue commands) and a programming language.

On systems with GUIs, many users rarely interact with the shell directly. However, GUIs are merely front ends for shells; that is, they are merely attractive interface layers that are built on top of a shell and which use the shell's commands. As shells are usually more powerful and feature-rich than GUIs, most intermediate and advanced users of Unix-like operating systems find them to be preferable to GUIs for many tasks.

A number of different shells have been developed for Unix-like operating systems. They share many similarities, but there are also some differences with regard to commands, syntax and functions that are important mainly for advanced users. Every Unix-like operating system has at least one built-in shell, and most have several.

- **sh** (the Bourne Shell) is the original UNIX shell, and it is still in widespread use today. Written by Stephen Bourne at Bell Labs in 1974, it is a simple shell with a small size and few features, perhaps the fewest of any shell for a Unix-like operating system. Bell Labs was the research and development arm of AT&T (The American Telephone and Telegraph Company), the former U.S. telecommunications monopoly. The first version of UNIX was developed at Bell Labs in 1969.

Among the functions that most users have come to expect in a shell but which are missing in sh are file name completion, command editing, command history and ease of executing multiple background processes (also referred to as jobs). A process is an instance of an executing program; a background process operates generally unnoticed while users are working with foreground processes. File name completion is the completion by the system of the names of files that have only partially typed in by a user. Command history allows users to conveniently find and reissue previously issued commands. Every Unix-like system contains sh or another shell which incorporates its commands.

- **bash** (Bourne-again shell) is the default shell on Linux. It also runs on nearly every other Unix-like operating system as well, and versions are also available for other operating systems including the Microsoft Windows systems. Bash is a superset of sh (i.e., commands that work in sh also work in bash, but the reverse is not always true), and it has many more commands than sh, making it a powerful tool for advanced users. But it is also intuitive and flexible, and thus it is probably the most suitable shell for beginners. bash was written for the GNU project (whose goal is to develop a complete, Unix-compatible, high performance and entirely free operating system), primarily by Brian Fox and Chet Ramey. Its name is a pun on the name of Steve Bourne.
- **ash** (the Almquist shell) is a clone of sh that was written by Kenneth Almquist in 1989. It is the shell that is the most compliant with sh, and it does not have any additional functions that other interactive shells such as bash and tcsh offer. ash also features small memory requirements compared to the other sh-compliant shells and is thus well suited for systems with small memories, especially small embedded systems (i.e., systems built into other products). It is available on most commercial Unix-like systems.
- **csh** (the C shell) has a syntax that resembles that of the highly popular C programming language (also developed at Bell Labs), and thus it is sometimes preferred by programmers. It was created in 1978 by Bill Joy (who also wrote the vi text editor and later co-founded Sun Microsystems) at the University of California at Berkeley (UCB).
- **ksh** (the Korn shell) is a superset of sh developed by David Korn at Bell Labs in 1983. It contains many features of the C shell as well, including a command history, which was inspired by the requests of Bell Labs users. It also features built-in arithmetic evaluation and advanced scripting capabilities similar to those found in powerful programming languages such as awk, sed and perl.
- **tcsh** (the TENEX C shell) is based on csh but also has programmable file name completion, command line editing, a command history mechanism and other features lacking in csh. It is named after the TENEX operating system, which inspired the author of tcsh. tcsh replaced the csh as the default shell on some BSD operating systems (i.e., FreeBSD and Darwin).
- **zsh** (the Z shell) is similar to ksh, but it also includes many features from csh. That is, it attempts to combine the programmability and syntax of the Korn shell with useful features from the C shell (which has some disadvantages as a programming language). It was written by Paul Falstad around 1990.

The great flexibility of shells on Unix-like operating systems is further enhanced by the fact that it is extremely easy to change the current shell. The shell for any user can be switched temporarily, or that user's default shell can be changed.

Some other families of operating systems also have shells. For example, MS-DOS became the shell on earlier versions of Microsoft Windows. However, it was replaced with a shell emulator on later versions (e.g., Windows 2000 and Windows XP), apparently because of the Microsoft philosophy of attempting to make the GUI all-powerful and de-emphasizing the command line."

1.6.3.1.2 Entry Point

From Wikipedia, the free encyclopedia:

"In computer programming, an entry point is where control enters a program or piece of code.

Usage customs

Contemporary

In most of today's popular computer systems, such as Microsoft Windows and Unix, a computer program usually only has a single entry point.

In C and C++ programs this is the `main()` function.

Historical

Historically, and in some contemporary legacy systems, such as VMS and OS/400, computer programs have a multitude of entry points, each corresponding to the different functionalities of the program. The usual way to denote entry points, as used system-wide in VMS and in PL/I and MACRO programs, is to append them at the end of the name of the executable image, delimited by a dollar sign (\$), e.g. `directory.exe$make`. The Apple 1 computer also used this to some degree. For example an alternative entry point in Apple 1 BASIC would keep the BASIC program[clarification needed] useful when the reset button was accidentally pushed.

1.6.3.2 Text-based User Interface (TUI)

From Wikipedia, the free encyclopedia

"Text-based user interface (TUI), also called textual user interface or terminal user interface,[clarification needed] is a retronym that was coined sometime after the invention of graphical user interfaces, to distinguish them from user interfaces that were text-based. The concept of TUI refers primarily to the way of output and does not coincide with command-line interfaces which is a certain user input mode. An advanced TUI may, like GUIs, use the entire screen area and does not necessarily provide line-by-line output, although TUIs only use text, symbols and colors available on a given text environment."

* * *

"From text application's point of view, there exists following three possibilities about the text screen and communications with it, ordered by decreasing of accessibility.

- 1 A genuine text mode display, controlled by a video adapter or the central processor itself. This is a normal condition for a locally-running application on various types of personal computers and mobile devices. If not deterred by the operating system, a smart program may exploit the full power of a hardware text mode.
- 2 A text mode emulator. Examples are `xterm` for X Window System and `win32 console` (in a window mode) for Microsoft Windows. This usually supports programs which expect a real text mode display, but may run considerably slower. Certain functions of an advanced text mode, such as an own font uploading, almost certainly become unavailable.

- 3** A remote text terminal. The communication capabilities usually become reduced to a serial line or its emulation, possibly with few `ioctl()`s as an out-of-band channel in such cases as Telnet and Secure Shell. This is the worst case, because software restrictions hinder the use of capabilities of a remote display device.

Draft

Under Linux and other Unix-like systems, a program easily accommodates to any of three cases because the same interface (namely, standard streams) is used to control the display and keyboard. Also, specialized programming libraries help to output the text in a way appropriate to the given display device and interface to it. See below a comparison to Windows."

* * *

"American National Standards Institute (ANSI) standard ANSI X3.64 defines a standard set of escape sequences that can be used to drive terminals to create TUIs (see ANSI escape code). Escape sequences may be supported for all three cases mentioned in the above section, allowing random cursor movements and color changes. However, not all terminals follow this standard, and many non-compatible but functionally equivalent sequences exist."

* * *

"On IBM Personal Computers and compatibles, the Basic Input Output System (BIOS) and DOS system calls provide a way to write text on the screen, and the ANSI.SYS driver could process standard ANSI escape sequences. However, programmers soon learned that writing data directly to the screen buffer was far faster and simpler to program, and less error-prone; see VGA-compatible text mode for details. This change in programming methods resulted in many DOS TUI programs. The win32 console environment is notorious for its emulation of certain EGA/VGA text mode features, particularly a random access to the text buffer, even if the application runs in a window. On the other hand, programs running under Windows (both native and DOS applications) have much less control of the display and keyboard than GNU/Linux. Linux and DOS programs can have, because of aforementioned win32 console layer.

Mouse cursor in Impulse Tracker. A more precise cursor (per-pixel resolution) was achieved by regenerating the glyphs of characters used where the cursor was visible, at each mouse movement in real-time.[citation needed]

Most often those programs used a blue background for the main screen, with white or yellow characters, although commonly they had also user color customization. Later, the interface became deeply influenced by graphical user interfaces (GUI), adding pull-down menus, overlapping windows, dialog boxes and GUI widgets operated by mnemonics or keyboard shortcuts. Soon mouse input was added – either at text resolution as a simple colored box or at graphical resolution thanks to the ability of the Enhanced Graphics Adapter (EGA) and Video Graphics Array (VGA) display adapters to redefine the text character shapes by software – providing additional functions.

Some notable programs of this kind were Microsoft Word, DOS Shell, WordPerfect, Norton Commander, Turbo Vision based Borland Turbo Pascal and Turbo C (the latter included the conio library), Lotus 1-2-3 and many others. Some of these interfaces survived even during the Microsoft Windows 3.1x period in the early 1990s. For example, the Microsoft C 6.0 compiler, used to write true GUI programs under 16-bit Windows, still has its own TUI.

Since its start, Microsoft Windows includes a console to display DOS software. Later versions added the Win32 console as a native interface for command-line interface and TUI programs. The console usually opens in window mode, but it can be switched to full true text mode screen and vice versa by pressing the Alt and Enter keys together. Full-screen mode is not available in Windows Vista and later, but may be used with some workarounds.[1]"

* * *

"In Unix-like operating systems, TUIs are often constructed using the terminal control library `curses`, or `ncurses`, a mostly compatible library.

The advent of the `curses` library with Berkeley Unix created a portable and stable API for which to write TUIs. The ability to talk to various text terminal types using the same interfaces led to more widespread use of "visual" Unix programs, which occupied the entire terminal screen instead of using a simple line interface. This can be seen in text editors such as `vi`, mail clients such as `pine` or `mutt`, system management tools such as `SMIT`, `SAM`, FreeBSD's `Sysinstall` and web browsers such as `lynx`. Some applications, such as `w3m`, and older versions of `pine` and `vi` use the less-able `termcap` library, performing many of the functions associated with `curses` within the application.

In addition, the rise in popularity of Linux brought many former DOS users to a Unix-like platform, which has fostered a DOS influence in many TUIs. The program `minicom`, for example, is modeled after the popular DOS program `Telnet`. Some other TUI programs, such as the `Twin` desktop, were ported over.

The free software program `GNU Screen` provides for managing multiple sessions inside a single TUI, and so can be thought of as being like a window manager for text-mode interfaces. `Tmux` can also do this.

The proprietary OS X text editor `BBEdit` includes a shell worksheet function that works as a full-screen shell window."

* * *

"Modern embedded systems are capable of displaying TUI on a monitor like personal computers. This functionality is usually implemented using specialized integrated circuits, modules, or using FPGA.

Video circuits or modules are usually controlled using VT100-compatible command set over UART,[citation needed] FPGA designs usually allow direct video memory access.[citation needed]"

1.6.3.3 POSIX Terminal Device Interface (TDI)

From Wikipedia, the free encyclopedia:

"The POSIX terminal interface is the generalized abstraction, comprising both an Application Programming Interface for programs, and a set of behavioural expectations for users of a terminal, as defined by the POSIX standard and the Single Unix Specification. It is a historical development from the terminal interfaces of BSD version 4 and Seventh Edition Unix."

* * *

"A multiplicity of I/O devices are regarded as "terminals" in Unix systems.[1][2] These include:

- serial devices connected by a serial port such as printers/teleprinters, teletypewriters, modems supporting remote terminals via dial-up access, and directly-connected local terminals[1][3][4][5]
- display adapter and keyboard hardware directly incorporated into the system unit, taken together to form a local "console", which may be presented to users and to programs as a single CRT terminal or as multiple virtual terminals[1]
- software terminal emulators, such as the xterm, Konsole, GNOME Terminal, and Terminal programs, and network servers such as the rlogin daemon and the SSH daemon, which make use of pseudoterminals"

* * *

"Unlike its mainframe and minicomputer contemporaries, the original Unix system was developed solely for dumb terminals, and that remains the case today.[6] A terminal is a character-oriented device, comprising streams of characters received from and sent to the device.[6][7] Although the streams of characters are structured, incorporating control characters, escape codes, and special characters, the I/O protocol is not structured as would be the I/O protocol of smart, or intelligent, terminals. There are no field format specifications. There's no block transmission of entire screens (input forms) of input data."

* * *

"Terminal intelligence and capabilities

Intelligence: terminals are dumb, not intelligent

Main article: terminal intelligence

Unlike its mainframe and minicomputer contemporaries, the original Unix system was developed solely for dumb terminals, and that remains the case today.[6] A terminal is a character-oriented device, comprising streams of characters received from and sent to the device.[6][7] Although the streams of characters are structured, incorporating control characters, escape codes, and special characters, the I/O protocol is not structured as would be the I/O protocol of smart, or intelligent, terminals. There are no field format specifications. There's no block transmission of entire screens (input forms) of input data.

Capabilities: terminfo, termcap, curses, et al.

Main article: terminal capabilities

The "capabilities" of a terminal comprise various dumb terminal features that are above and beyond what is available from a pure teletypewriter, which programs can make use of. They (mainly) comprise escape codes that can be sent to or received from the terminal. The escape codes sent to the terminal perform various functions that a CRT terminal (or software terminal emulator) is capable of that a teletypewriter is not, such as moving the terminal's cursor to positions on the screen, clearing and scrolling all or parts of the screen, turning on and off attached printer devices, programmable function keys, changing display colours and attributes (such as reverse video), and setting display title strings. The escape codes received from the terminal signify things such as function key, arrow key, and other special keystrokes (home key, end key, help key, PgUp key, PgDn key, insert key, delete key, and so forth).[8][9]

These capabilities are encoded in databases that are configured by a system administrator and accessed from programs via the terminfo library (which supersedes the older termcap library), upon which in turn are built libraries such as the curses (programming library) and ncurses libraries. Application programs use the terminal capabilities to provide textual user interfaces with windows, dialogue boxes, buttons, labels, input fields, menus, and so forth.[10][11]

Controlling environment variables: TERM et al.

The particular set of capabilities for the terminal that a (terminal-aware) program's input and output uses is obtained from the database rather than hardwired into programs and libraries, and is controlled by the TERM environment variable (and, optionally for the termcap library, the TERMCAP environment variable), and, optionally for the terminfo library, the TERMINFO environment variable).[10] This variable is set by whatever terminal monitor program spawns the programs that then use that terminal for its input and output, or sometimes explicitly. For example:

- The `getty` program (or equivalent) sets the TERM environment variable according to a system database (variously `inittab` or the configuration files for the `tty` or `launchd` programs) defining what local terminals are attached to what serial ports and what terminal types are provided by local virtual terminals or the local system console.
- A dial-up user on a remote terminal is not using the type of terminal that the system commonly expects on that dial-up line, and so manually sets the TERM environment variable immediately after login to the correct type. (More usually, the terminal type set by the `getty` program for the dial-up line, that the system administrator has determined to be used most often by dial-up users with remote terminals, matches the one used by the dial-up user and that user has no need to override the terminal type.)
- The SSH server daemon (or equivalent such as the `rlogin` daemon) sets the TERM environment variable to the same terminal type as the SSH client.[12]
- The software terminal emulator, using a pseudoterminal, sets the TERM environment variable to specify the type of terminal that it is emulating. Emulated terminals often do not exactly match real terminal hardware, and terminal emulators have type names dedicated for their use. The `xterm` program (by default) sets `xterm` as the terminal type, for example.[13] The `screen` program sets `screen` as the terminal type."

1.6.4 Graphical User Interface (GUI)

From Wikipedia, the free encyclopedia:

"The graphical user interface is presented /displayed on the screen. It is the result of processed user input and usually the primary interface for human-machine interaction. The Touch user interfaces popular on small mobile devices are an overlay of the visual output to the visual input.

In computing,[1] graphical user interface (GUI, sometimes pronounced 'gooey')[2] is a type of user interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation, as opposed to text-based interfaces, typed command labels or text navigation. GUIs were introduced in reaction to the perceived steep learning curve of command-line interfaces (CLI),[3][4][4] which require commands to be typed on the keyboard.

The actions in GUI are usually performed through direct manipulation of the graphical elements.[5] Besides in computers, GUIs can be found in hand-held devices such as MP3 players, portable media players, gaming devices, household appliances, office, and industry equipment. The term GUI is usually not applied to other low-resolution types of interfaces with display resolutions, such as video games (where HUD[6] is preferred), or not restricted to flat screens, like volumetric displays[7] because the term is restricted to the scope of two-dimensional display screens able to describe generic information, in the tradition of the computer science research at the PARC (Palo Alto Research Center)."

1.6.4.1 Graphical Device Interface (GDI)

From Wikipedia, the free encyclopedia:

"The Graphics Device Interface (GDI) is a Microsoft Windows application programming interface and core operating system component responsible for representing graphical objects and transmitting them to output devices such as monitors and printers.

GDI is responsible for tasks such as drawing lines and curves, rendering fonts and handling palettes. It is not directly responsible for drawing windows, menus, etc.; that task is reserved for the user subsystem, which resides in user32.dll and is built atop GDI. Other systems have components that are similar to GDI, for example Mac OS X's Quartz and GTK's GDK/Xlib.

GDI's most significant advantages over more direct methods of accessing the hardware are perhaps its scaling capabilities and its abstract representation of target devices. Using GDI, it is very easy to draw on multiple devices, such as a screen and a printer, and expect proper reproduction in each case. This capability is at the center of all What You See Is What You Get applications for Microsoft Windows.

Simple games that do not require fast graphics rendering may use GDI. However, GDI is relatively hard to use for advanced animation, and lacks a notion for synchronizing with individual video frames in the video card, lacks hardware rasterization for 3D, etc. Modern games usually use DirectX or OpenGL instead, which let programmers exploit the features of modern hardware."

1.6.4.2 Widget Toolkit

From Wikipedia, the free encyclopedia (http://en.wikipedia.org/wiki/Widget_toolkit):

"In computing, a widget toolkit, widget library, or GUI toolkit is a set of widgets for use in designing applications with graphical user interfaces (GUIs). The toolkit itself is a piece of software which is usually built on the top of an operating system, windowing system, or window manager and provides programs with an application programming interface (API), allowing them to make use of widgets. Each widget facilitates a specific user-computer interaction, and appears as a visible part of the computer's GUI. Widget toolkits can be either native or cross platform.

Widgets that are provided by a toolkit typically adhere to a unified design specification, including aesthetics, to lend a sense of overall cohesion among various parts of the application and between various applications within the GUI.

Widget toolkits also contain software to assist in the creation of window managers, as windows themselves are considered widgets. Some widgets support interaction with the user, for example labels, buttons, and check boxes. Others act as containers that group the widgets added to them, for example windows, panels, and tabs.

The graphical user interface of a program is commonly constructed in a cascading manner, with widgets being added directly to on top of existing widgets. In many implementations application windows are added directly to the desktop by the window manager, and can be stacked layered on top of each other through various means. Each window is associated with a particular application which controls the widgets added to its canvas, which can be watched and modified by their associated applications.

Most widget toolkits use event-driven programming as a model for interaction.[1] The toolkit handles user events, for example when the user clicks on a button. When an event is detected, it is passed on to the application where it is dealt with. The design of those toolkits has been criticized for promoting an oversimplified model of event-action, leading programmers to create error-prone, difficult to extend and excessively complex application code.[2] Finite State Machines and Hierarchical State Machines have been proposed as high-level models to represent the interactive state changes for reactive programs.

The look and feel of the widgets can be hard-coded in the toolkit, but some widget toolkit APIs decouple the look and feel from the definition of the widgets, allowing the widgets to be themed. (see pluggable look and feel)."

See ***Partial List of Widget Toolkits*** (on page 108)

1.6.4.2.1 Partial List of Widget Toolkits

Excerpted From Wikipedia, the free encyclopedia.

- ***Low Level*** (on page 109)
- ***High Level*** (on page 109)

1.6.4.2.1.1 Low Level

Excerpt From Wikipedia, the free encyclopedia:

- 1 Macintosh Toolbox/Carbon
- 2 Intrinsic
- 3 Intuition
- 4 Windows API
- 5 Xlib
- 6 XCB

1.6.4.2.1.2 High Level

Excerpt From Wikipedia, the free encyclopedia:

- *On Mac OS X* (on page 109)
- *On Windows* (on page 109)
- *On Unix under X11* (on page 110)

1.6.4.2.1.2.1 *On Mac OS X*

Excerpt From Wikipedia, the free encyclopedia:

- 1 Carbon
- 2 Cocoa
- 3 MacApp
- 4 MacZoop
- 5 PowerPlant

1.6.4.2.1.2.2 *On Windows*

Excerpt From Wikipedia, the free encyclopedia:

- 1 Microsoft Foundation Class Library
- 2 Object Windows Library
- 3 Silverlight
- 4 SmartWin++
- 5 Visual Component Library
- 6 Windows Forms
- 7 Windows Presentation Foundation
- 8 Windows Template Library
- 9 WinRT XAML

1.6.4.2.1.2.3 On Unix under X11

Excerpt From Wikipedia, the free encyclopedia:

- 1** Athena (Xaw)
- 2** InterViews
- 3** LessTif
- 4** Motif
- 5** OPEN LOOK
- 6** C or C++
 - a) CEGUI
 - b) Component Library for Cross Platform
 - c) FLTK
 - d) FOX toolkit
 - e) OpenGL User Interface Library
 - f) GTK+
 - g) Juce
 - h) Qt
 - i) Wt
 - j) Tk
 - k) TnFOX
 - l) Ultimate++
 - m) Visual Component Framework
 - n) wxWidgets
 - o) YAAF
 - p) XForms
 - q) XVT
- 7** Python
 - a) Pyjamas
 - b) PyQt
 - c) PyGTK
 - d) PyGObject
 - e) PySide
 - f) Tkinter
 - g) Urwid (see *Urwid - Console User Interface Library* (<http://excess.org/urwid/>))
 - h) wxPython

1.6.4.2.2 Widget Toolkit Application Programming Interface (API)

From Wikipedia, the free encyclopedia:

"Widget toolkits introduce use of standardized building blocks (widgets, sizers and events) that facilitate the development of application programs requiring multiple concurrent operator input and computer output activities.

Modern examples include the display managers for Linux, Mac OS X and Microsoft Windows and third part toolkits such as Tk, Urwid (see *Urwid - Console User Interface Library* (<http://excess.org/urwid/>)), wxWigtets and the X11 project's X-window system.

Features typically include:

- top level frame and dialog widgets for windows associated with concurrent processes (tasks)
- lower level operator input widgets for windows associated with buttons, check boxes, radio buttons grouped within radio boxes
- lower level computer output widgets for scrollable text windows and their associated scroll bar gauge (to display the starting point and size of the displayed text relative to any non-displayed text) and scroll controlling arrow buttons to change the starting column and/or row of the displayed text.
- sizers for automatically subdividing a higher level window and setting the position and size of the resulting subwindows
- events for processing asynchronous occurrences that are dispatched to the associated event handler.
- various proportional and fixed size fonts with attributes such as blink, bold, italics, normal, thin, and underlined
- various colors and associated foreground/background color pair combinations

Application software developers typically use widget toolkits offering a high-level API (such as the cross platform C++ based "wxWidgets" and its "wxPython" for Python language programmers). The widget toolkit with the high-level API is itself typically constructed using the low-level API of more primitive building blocks provided by the host Operating System or by a *POSIX Terminal Device Interface (TDI)* (see "*POSIX Terminal Device Interface (TDI)*" on page 105) (such as "curses")."

1.6.4.2.2.1 "wxWidgets" High-Level API for Pixel-mode , Cross-Platform GUI Toolkit

From Wikipedia, the free encyclopedia:

"wxWidgets (formerly wxWindows) is a widget toolkit and tools library for creating graphical user interfaces (GUIs) for cross-platform applications. wxWidgets enables a program's GUI code to compile and run on several computer platforms with minimal or no code changes. It covers systems such as Microsoft Windows, OS X (Carbon and Cocoa), iOS (Cocoa Touch), Linux/Unix (X11, Motif, and GTK+), OpenVMS, OS/2 and AmigaOS. A version for embedded systems is under development.[2]

wxWidgets is used across many industry sectors, most notably by Xerox, Advanced Micro Devices (AMD), Lockheed Martin, NASA and the Center for Naval Analyses. It is also used in the public sector and education by, for example, Dartmouth Medical School, National Human Genome Research Institute, National Center for Biotechnology Information, and many others.[3] wxWidgets is used in many open source projects,[4] and by individual developers. A wide choice of compilers and other tools to use with wxWidgets allows development of highly sophisticated applications on a tight budget.[3]

It is free and open source software, distributed under the terms of the wxWidgets License, which satisfies those who wish to produce for GPL and proprietary software.[5]

History

wxWidgets (initially wxWindows) was started in 1992 by Julian Smart at the University of Edinburgh.[6] He attained an honours degree in Computational science from the University of St Andrews in 1986, and is still a core developer.[7][8]

On February 20, 2004, the developers of wxWindows announced that the project was changing its name to wxWidgets, as a result of Microsoft requesting Julian Smart to respect Microsoft's United Kingdom trademark of the term Windows.[9]

Major release versions were 2.4 on 6 January 2003, 2.6 on 21 April 2005 and 2.8.0 on 14 December 2006. Version 3.0 was released on 11 November 2013.

wxWidgets has participated in the Google Summer of Code since 2006.[10][11]

License

wxWidgets is distributed under a custom made wxWindows License, similar to the GNU Lesser General Public License, with an exception stating that derived works in binary form may be distributed on the user's own terms.[5] This license is a free software license approved by the FSF,[17] making wxWidgets free software. It has been approved by the Open Source Initiative (OSI).[18]

Official support

Supported platforms

wxWidgets is supported on the following platforms.[19]

- Windows - wxMSW (Windows 95, 98, Me; NT, 2000, XP, Vista, 7, 8)
- Linux/Unix wxGTK, wxX11, wxMotif

- OS X - wxMac (10.3 using Carbon)
- OS/2 - wxOS2, wxPM, wxWidgets for GTK+ or Motif can be compiled on OS/2
- Embedded platforms - wxEmbedded[2]"

Draft

1.6.4.2.2.1.1 *"wxPython" High-Level API for "wxWidgets" GUI Toolkit Wrapper*

From Wikipedia, the free encyclopedia:

"wxPython is a wrapper for the cross-platform GUI API (often referred to as a 'toolkit') wxWidgets (which is written in C++) for the Python programming language. It is one of the alternatives to Tkinter, which is bundled with Python. It is implemented as a Python extension module (native code). Other popular alternatives are PyGTK and PyQt. Like wxWidgets, wxPython is free software.

License

Being a wrapper, wxPython uses the same free software licence used by wxWidgets (wxWindows License)[1]—which is approved by Free Software Foundation and Open Source Initiative.

History

wxPython was born when Robin Dunn needed a GUI to be deployed on HP-UX systems and also on Windows 3.1 in a few weeks time. While evaluating commercial solutions, he ran across Python bindings for the wxWidgets toolkit. Thus, he learned Python and, in a short time, became one of the main developers of wxPython (which grew from those initial bindings), together with Harri Pasanen. The first versions of the wrapper were created by hand. However, soon the code base became very difficult to maintain and keep in sync with wxWidgets releases. Later versions were created with SWIG, greatly decreasing the amount of work to update the wrapper. The first "modern" version was announced in 1998.[2]

Example

This is a simple "Hello world" module, depicting the creation of the two main objects in wxPython (the main window object and the application object), followed by passing the control to the event-driven system (by calling MainLoop()) which manages the user-interactive part of the program.

```
#!/usr/bin/env python

import wx

class TestFrame(wx.Frame):
    def __init__(self, parent, title):
        wx.Frame.__init__(self, parent, wx.ID_ANY, title=title)
        text = wx.StaticText(self, label="Hello, world!")

app = wx.App(redirect=False)
frame = TestFrame(None, "Hello, world!")
frame.Show()
app.MainLoop()
```

Applications Developed with wxPython

- BitTorrent, a peer-to-peer Bit Torrent application
- Chandler, a Personal Information Manager
- Dropbox, a storage provider/file synchroniser

- Phatch, a Photo Batch Processor
- Métamorphose - A batch renamer
- PlayOnLinux and PlayOnMac - Wine front-ends
- GRASS GIS, a free, open source geographical information system
- Google Drive, desktop client for the Google cloud-based storage system.[3]"

Draft

1.6.4.2.2.1.1.1 "nCurses" Low Level API for Text-mode, Cross-Platform GUI Toolkit

From Wikipedia, the free encyclopedia:

"ncurses (new curses) is a programming library that provides an API which allows the programmer to write text-based user interfaces in a terminal-independent manner. It is a toolkit for developing "GUI-like" application software that runs under a terminal emulator. It also optimizes screen changes, in order to reduce the latency experienced when using remote shells.

History

The N in ncurses comes from the word new. This is because ncurses is a free software emulation (clone) of the System V Release 4.0 (SVr4) curses, which was itself an enhancement over the discontinued classic 4.4 BSD curses.[2] The XSI Curses standard issued by X/Open is explicitly and closely modeled on System V.

curses

The first curses library was developed at the University of California at Berkeley, for a BSD operating system, around 1980 to support a screen-oriented game. It originally used the termcap library, which was used in other programs, such as the vi editor.[2]

The success of the BSD curses library prompted Bell Labs to release an enhanced curses library in their System III and System V Release 1 Unix systems. This library was more powerful and instead of using termcap, it used terminfo. However, due to AT&T policy regarding source-code distribution, this improved curses library did not have much acceptance in the BSD community.[2]

pcurses

Around 1982, Pavel Curtis started work on a freeware clone of the Bell Labs curses, named pcurses, which was maintained by various people through 1986.[3]

ncurses

The pcurses library was further improved when Zeyd Ben-Halim took over the development effort in late 1991.[2][3][4] The new library was released as ncurses in November 1993, with version 1.8.1 as the first major release. Subsequent work, through version 1.8.8 (1995), was driven by Eric S. Raymond, who added the form and menu libraries written by Juergen Pfeifer.[5] Since 1996, it has been maintained by Thomas E. Dickey.[3]

Most ncurses calls can be easily ported to the old curses. System V curses implementations can support BSD curses programs with just a recompilation.[6] However, a few areas are problematic, such as handling terminal resizing, since no counterpart exists in the old curses.

Terminal database

Ncurses can use either terminfo (with extensible data) or termcap. Other implementations of curses generally use terminfo; a minority use termcap. Few (mytinfo was an older exception[7]) use both.

License

Ncurses is a part of the GNU Project. It is one of the few GNU files not distributed under the GNU GPL or LGPL; it is distributed under a permissive free software licence, similar to the MIT License.[8] This is due to the agreement made with the Free Software Foundation at the time the developers assigned their copyright.

When the agreement was made to pass on the rights to the FSF, there was a clause that stated

The Foundation promises that all distribution of the Package, or of any work "based on the Package", that takes place under the control of the Foundation or its agents or assignees, shall be on terms that explicitly and perpetually permit anyone possessing a copy of the work to which the terms apply, and possessing accurate notice of these terms, to redistribute copies of the work to anyone on the same terms.[8]

According to the maintainer Thomas E. Dickey, this precludes relicensing to the GPL in any version, since it would place restrictions on the programs that will be able to link to the libraries.[8]

Programs using ncurses

There are hundreds of programs which use ncurses.[9][10] Some, such as GNU Screen and w3m, use only the termcap interface, performing screen management within the application. Others, such as GNU Midnight Commander and YaST, use the curses programming interface."

1.6.5 Python Programming Language

- 1 *Python Technology Overview* (on page 118)
- 2 *CLI Shell Mode Examples* (on page 123)
- 3 *TUI Low-Level Curses Mode Examples* (on page 144)
- 4 *GUI High-Level wxPython Mode Examples* (on page 156)

1.6.5.1 Python Technology Overview

From Wikipedia, the free encyclopedia:

"Python is a widely used general-purpose, high-level programming language.[12][13][14] Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C.[15][16] The language provides constructs intended to enable clear programs on both a small and large scale.[17]

Python supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library.[18]

Like other dynamic languages, Python is often used as a scripting language, but is also used in a wide range of non-scripting contexts. Using third-party tools, Python code can be packaged into standalone executable programs (such as Py2exe, or Pyinstaller[19]). Python interpreters are available for many operating systems.

CPython, the reference implementation of Python, is free and open source software and has a community-based development model, as do nearly all of its alternative implementations. CPython is managed by the non-profit Python Software Foundation.

History

Python was conceived in the late 1980s[20] and its implementation was started in December 1989[21] by Guido van Rossum at CWI in the Netherlands as a successor to the ABC language (itself inspired by SETL)[22] capable of exception handling and interfacing with the Amoeba operating system.[2] Van Rossum is Python's principal author, and his continuing central role in deciding the direction of Python is reflected in the title given to him by the Python community, Benevolent Dictator for Life (BDFL).

Python 2.0 was released on 16 October 2000, with many major new features including a full garbage collector and support for Unicode. With this release the development process was changed and became more transparent and community-backed.[23]

Python 3.0 (also called Python 3000 or py3k), a major, backwards-incompatible release, was released on 3 December 2008[24] after a long period of testing. Many of its major features have been backported to the backwards-compatible Python 2.6 and 2.7.[25]

Features and philosophy

Python is a multi-paradigm programming language: object-oriented programming and structured programming are fully supported, and there are a number of language features which support functional programming and aspect-oriented programming (including by metaprogramming[26] and by magic methods).[27] Many other paradigms are supported using extensions, including design by contract[28][29] and logic programming.[30]

Python uses dynamic typing and a combination of reference counting and a cycle-detecting garbage collector for memory management. An important feature of Python is dynamic name resolution (late binding), which binds method and variable names during program execution.

The design of Python offers only limited support for functional programming in the Lisp tradition. The language has `map()`, `reduce()` and `filter()` functions, comprehensions for lists, dictionaries, and sets, as well as generator expressions.[31] The standard library has two modules (`itertools` and `functools`) that implement functional tools borrowed from Haskell and Standard ML.[32]

The core philosophy of the language is summarized by the document "PEP 20 (The Zen of Python)", which includes aphorisms such as:[33]

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.

Rather than requiring all desired functionality to be built into the language's core, Python was designed to be highly extensible. Python can also be embedded in existing applications that need a programmable interface. This design of a small core language with a large standard library and an easily extensible interpreter was intended by Van Rossum from the very start because of his frustrations with ABC (which espoused the opposite mindset).[20]

While offering choice in coding methodology, the Python philosophy rejects exuberant syntax, such as in Perl, in favor of a sparser, less-cluttered grammar. As Alex Martelli put it: "To describe something as clever is not considered a compliment in the Python culture." [34] Python's philosophy rejects the Perl "there is more than one way to do it" approach to language design in favor of "there should be one—and preferably only one—obvious way to do it".[33]

Python's developers strive to avoid premature optimization, and moreover, reject patches to non-critical parts of CPython which would offer a marginal increase in speed at the cost of clarity.[35] When speed is important, Python programmers use PyPy, a just-in-time compiler, or move time-critical functions to extension modules written in languages such as C. Cython is also available which translates a Python script into C and makes direct C level API calls into the Python interpreter.

An important goal of the Python developers is making Python fun to use. This is reflected in the origin of the name which comes from Monty Python,[36] and in an occasionally playful approach to tutorials and reference materials, for example using spam and eggs instead of the standard foo and bar.[37][38]

A common neologism in the Python community is *pythonic*, which can have a wide range of meanings related to program style. To say that code is *pythonic* is to say that it uses Python idioms well, that it is natural or shows fluency in the language, that it conforms with Python's minimalist philosophy and emphasis on readability. In contrast, code that is difficult to understand or reads like a rough transcription from another programming language is called *unpythonic*.

Users and admirers of Python—most especially those considered knowledgeable or experienced—are often referred to as Pythonists, Pythonistas, and Pythoneers.[39][40]

Syntax and semantics

Python is intended to be a highly readable language. It is designed to have an uncluttered visual layout, frequently using English keywords where other languages use punctuation. Furthermore Python has a smaller number of syntactic exceptions and special cases than C or Pascal.[41]

Indentation

Python uses whitespace indentation, rather than curly braces or keywords, to delimit blocks; a feature also termed the off-side rule. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block.[42] It is considered beneficial by Python programmers,[43] others have criticized it.[44]

Statements and control flow

Python's statements include (among others):

- The if statement, which conditionally executes a block of code, along with else and elif (a contraction of else-if).
- The for statement, which iterates over an iterable object, capturing each element to a local variable for use by the attached block.
- The while statement, which executes a block of code as long as its condition is true.
- The try statement, which allows exceptions raised in its attached code block to be caught and handled by except clauses; it also ensures that clean-up code in a finally block will always be run regardless of how the block exits.
- The class statement, which executes a block of code and attaches its local namespace to a class, for use in object-oriented programming.
- The def statement, which defines a function or method.
- The with statement (from Python 2.5), which encloses a code block within a context manager (for example, acquiring a lock before the block of code is run and releasing the lock afterwards, or opening a file and then closing it), allowing RAII-like behavior.
- The pass statement, which serves as a NOP. It is syntactically needed to create an empty code block.
- The assert statement, used during debugging to check for conditions that ought to apply.
- The yield statement, which returns a value from a generator function. From Python 2.5, yield is also an operator. This form is used to implement coroutines.
- The import statement, which is used to import modules whose functions or variables can be used in the current program.

Python does not support tail-call optimization or first-class continuations, and, according to Guido van Rossum, it never will.[45][46] However, better support for coroutine-like functionality is provided in 2.5, by extending Python's generators.[47] Prior to 2.5, generators were lazy iterators; information was passed unidirectionally out of the generator. As of Python 2.5, it is possible to pass information back into a generator function, and as of Python 3.3, the information can be passed through multiple stack levels.[48]

Expressions

Python expressions are similar to languages such as C and Java:

- Addition, subtraction, and multiplication are the same, but the behavior of division differs (see Mathematics for details). Python also adds the `**` operator for exponentiation.
- In Python, `==` compares by value, in contrast to Java, where it compares by reference. (Value comparisons in Java use the `equals()` method.) Python's `is` operator may be used to compare object identities (comparison by reference). Comparisons may be chained, for example `a <= b <= c`.
- Python uses the words `and`, `or`, `not` for its boolean operators rather than the symbolic `&&`, `||`, `!` used in Java and C.
- Python has a type of expression termed a list comprehension. Python 2.4 extended list comprehensions into a more general expression termed a generator expression.[31]
- Anonymous functions are implemented using lambda expressions; however, these are limited in that the body can only be a single expression.
- Conditional expressions in Python are written as `x if c else y`[49] (different in order of operands from the `?:` operator common to many other languages).
- Python makes a distinction between lists and tuples. Lists are written as `[1, 2, 3]`, are mutable, and cannot be used as the keys of dictionaries (dictionary keys must be immutable in Python). Tuples are written as `(1, 2, 3)`, are immutable and thus can be used as the keys of dictionaries, provided all elements of the tuple are immutable. The parentheses around the tuple are optional in some contexts. Tuples can appear on the left side of an equal sign; hence a statement like `x, y = y, x` can be used to swap two variables.
- Python has a "string format" operator `%`. This functions analogous to `printf` format strings in C, e.g. `"foo=%s bar=%d" % ("blah", 2)` evaluates to `"foo=blah bar=2"`. In Python 3 and 2.6+, this was supplemented by the `format()` method of the `str` class, e.g. `"foo={0} bar={1}".format("blah", 2)`.
- Python has various kinds of string literals:
 - Strings delimited by single or double quotation marks. Unlike in Unix shells, Perl and Perl-influenced languages, single quotation marks and double quotation marks function similarly. Both kinds of string use the backslash (`\`) as an escape character and there is no implicit string interpolation such as `"$foo"`.
 - Triple-quoted strings, which begin and end with a series of three single or double quotation marks. They may span multiple lines and function like here documents in shells, Perl and Ruby.
 - Raw string varieties, denoted by prefixing the string literal with an `r`. No escape sequences are interpreted; hence raw strings are useful where literal backslashes are common, such as regular expressions and Windows-style paths. Compare `"@-quoting"` in C#.

- Python has index and slice expressions on lists, denoted as `a[key]`, `a[start:stop]` or `a[start:stop:step]`. Indexes are zero-based, and negative indexes are relative to the end. Slices take elements from the start index up to, but not including, the stop index. The third slice parameter, called step or stride, allows elements to be skipped and reversed. Slice indexes may be omitted, for example `a[:]` returns a copy of the entire list. Each element of a slice is a shallow copy.

In Python, a distinction between expressions and statements is rigidly enforced, in contrast to languages such as Common Lisp, Scheme, or Ruby. This leads to some duplication of functionality. For example:

- List comprehensions vs. for-loops
- Conditional expressions vs. if blocks
- The `eval()` vs. `exec()` built-in functions (in Python 2, `exec` is a statement); the former is for expressions, the latter is for statements.

Statements cannot be a part of an expression and so list and other comprehensions or lambda expressions, all being expressions, cannot contain statements. A particular case of this is that an assignment statement such as `a = 1` cannot form part of the conditional expression of a conditional statement. This has the advantage of avoiding a classic C error of mistaking an assignment operator `=` for an equality operator `==` in conditions: `if (c = 1) { ... }` is valid C code but `if c = 1: ...` causes a syntax error in Python.

Methods

Methods on objects are functions attached to the object's class; the syntax `instance.method(argument)` is, for normal methods and functions, syntactic sugar for `Class.method(instance, argument)`. Python methods have an explicit `self` parameter to access instance data, in contrast to the implicit `self` in some other object-oriented programming languages (for example, Java, C++ or Ruby).[50]

Typing

Python uses duck typing and has typed objects but untyped variable names. Type constraints are not checked at compile time; rather, operations on an object may fail, signifying that the given object is not of a suitable type. Despite being dynamically typed, Python is strongly typed, forbidding operations that are not well-defined (for example, adding a number to a string) rather than silently attempting to make sense of them.

Python allows programmers to define their own types using classes, which are most often used for object-oriented programming. New instances of classes are constructed by calling the class (for example, `SpamClass()` or `EggsClass()`), and the classes themselves are instances of the metaclass type (itself an instance of itself), allowing metaprogramming and reflection.

Prior to version 3.0, Python had two kinds of classes: "old-style" and "new-style".[51] Old-style classes were eliminated in Python 3.0, making all classes new-style. In versions between 2.2 and 3.0, both kinds of classes could be used. The syntax of both styles is the same, the difference being whether the class object is inherited from, directly or indirectly (all new-style classes inherit from `object` and are instances of `type`)."

For the official introduction and tutorials see: <http://www.python.org/>

1.6.5.2 CLI Shell Mode Examples

1.6.5.2.1 Python (CLI-mode) Interpreter Demo

EXAMPLE	SOURCE CODE
Command Line Interface launches Python 2.7 Interpreter	<pre>\$ python2.7 Python 2.7.3 (default, Dec 18 2012, 13:50:09) [GCC 4.5.3] on cygwin Type "help", "copyright", "credits" or "license" for more information. >>> print "Hello World via Python Print (Statement)" Hello World via Python Print (Statement) >>> print("Hello World via Python Print (Function)") Hello World via Python Print (Function)</pre>
Command Line Interface launches Python 3.2 Interpreter	<pre>\$ python3.2 Python 3.2.3 (default, Jul 23 2012, 16:48:24) [GCC 4.5.3] on cygwin Type "help", "copyright", "credits" or "license" for more information. >>> print "Hello World via Python Print (Statement)" File "<stdin>", Line 1 print "Hello World via Python Print (Statement)" ^ SyntaxError: invalid syntax >>> print("Hello World via Python Print (Function)") Hello World via Python Print (Function)</pre>

1.6.5.2.2 Python (CLI-mode) Unstructured Script Demo

EXAMPLE	SOURCE CODE
Command Line Interface launches Python 2.7 Application Script	<pre>\$ python2.7 the_CLI_Script.py #!/usr/bin/env python #----- print "Hello World via Python Print (Statement)" print("Hello World via Python Print (Function)")</pre>

1.6.5.2.3 Python (CLI-mode) Structured Script Demo

EXAMPLE	SOURCE CODE
Command Line Interface launches Python 2.7 Structured Script	<pre>\$ python2.7 the_CLI_Main_Program.py #!/usr/bin/env python #-----</pre>

```
def print_statement(text):  
    print '%s (statement)' % text  
  
#-----  
  
def print_function(text):  
    print('%s (function)' % text)  
  
#-----  
  
if __name__ == '__main__':  
    message = "Hello World via Python Print"  
    print_statement(message)  
    print_function(message)
```

1.6.5.2.4 Python (CLI-mode) Logging Module Demo

From <http://docs.python.org/2/howto/logging.html> (<http://docs.python.org/2/howto/logging.html>):

"Logging HOWTO

Author: Vinay Sajip <vinay_s:ajip at red-dove dot com>

Basic Logging Tutorial

Logging is a means of tracking events that happen when some software runs. The software's developer adds logging calls to their code to indicate that certain events have occurred. An event is described by a descriptive message which can optionally contain variable data (i.e. data that is potentially different for each occurrence of the event). Events also have an importance which the developer ascribes to the event; the importance can also be called the level or severity.

When to use logging

Logging provides a set of convenience functions for simple logging usage. These are debug(), info(), warning(), error() and critical(). To determine when to use logging, see the table below, which states, for each of a set of common tasks, the best tool to use for it.

TASK YOU WANT TO PERFORM	THE BEST TOOL FOR THE TASK
Display console output for ordinary usage of a command line script or program	print()
Report events that occur during normal operation of a program (e.g. for status monitoring or fault investigation)	logging.info() (or logging.debug() for very detailed output for diagnostic purposes)

Issue a warning regarding a particular runtime event <code>warnings.warn()</code> in library code if the issue is avoidable and the client application should be modified to eliminate the warning	<code>logging.warning()</code> if there is nothing the client application can do about the situation, but the event should still be noted
Report an error regarding a particular runtime event	Raise an exception
Report suppression of an error without raising an exception (e.g. error handler in a long-running server process)	<code>logging.error()</code> , <code>logging.exception()</code> or <code>logging.critical()</code> as appropriate for the specific error and application domain

The logging functions are named after the level or severity of the events they are used to track. The standard levels and their applicability are described below (in increasing order of severity):

LEVEL	WHEN IT'S USED
DEBUG	Detailed information, typically of interest only when diagnosing problems.
INFO	Confirmation that things are working as expected.
WARNING	An indication that something unexpected happened, or indicative of some problem in the near future (e.g. 'disk space low'). The software is still working as expected.
ERROR	Due to a more serious problem, the software has not been able to perform some function.
CRITICAL	A serious error, indicating that the program itself may be unable to continue running.

The default level is WARNING, which means that only events of this level and above will be tracked, unless the logging package is configured to do otherwise.

Events that are tracked can be handled in different ways. The simplest way of handling tracked events is to print them to the console. Another common way is to write them to a disk file."

1.6.5.2.5 Python (CLI-mode) "tsWxGTUI" tsPlatformQuery Demo

EXAMPLE	SOURCE CODE
Command Line Interface launches Python 2.7 "tsToolsCLI" Application Program	<pre>\$ python2.7 tsPlatformQuery.py</pre>
	See the following source code.

```
#!/usr/bin/env python
#"Time-stamp: <06/10/2013  8:09:49 PM rsg>"
'''
tsPlatformQuery - Tool to capture current hardware and software
information about the run time environment available to computer
programs.
'''
#####
#
# File: tsPlatformQuery.py
#
# Purpose:
#
#     Tool to capture current hardware and software information
#     about the run time environment available to computer
#     programs.
#
# Limitations:
#
#     1. Host processor hardware support includes various
#        releases of Pentium, PowerPC, SPARC.
#
#     2. Host operating system software support includes
#        various releases of Cygwin, Linux ('SuSE', 'Debian',
#        'redhat', 'mandrake'), Mac OS ('X'), Unix ('Solaris')
#        and Windows ('98', '2000', 'XP', 'Vista').
#
#     3. Host virtual machine software support includes
#        various releases of Java and Python.
#
# Usage (example):
#
#     tsPlatformQuery.py
#
# Methods:
#
#     OperatorSettingsParser - Class to incorporate features in
#         tsOperatorSettingsParser.
#
#     OperatorSettingsParser.__init__ - Class constructor.
#
#     OperatorSettingsParser.getOptions - Parse the command line and
#         return a list of positional arguments and a dictionary
#         of keyword options.
#
#     OperatorSettingsParser.getRuntimeTitle - Method to return
#         Run Time Title, or Build Title, whichever was actually
#         used in command line, stripping it of any file path.
#
#     OperatorSettingsParser.getRuntimeTitleVersionDate - Method to
#         return the Build Title, Version and Date (with any
#         Run Time update) for use in command line parsing help
#         and error display output.
#
#     OperatorSettingsParser.theMainApplication - Display program name,
```



```

__line3__
__line4__

mainTitleVersionDate = __line1__

#-----

import os
import string
import sys
from optparse import OptionParser
import textwrap

#-----

try:

    import tsLibCLI

except ImportError, importCode:

    print('%s: ImportError (tsLibCLI); ' % __title__ + \
          'importCode=%s' % str(importCode))

#-----

DEBUG = False

#-----

try:

    import tsExceptions as tse
    import tsPlatformRunTimeEnvironment as tsquery
    from tsReportUtilities import TsReportUtilities as tsru

    import tsCommandLineEnv

except ImportError, importCode:

    print('%s: ImportError (tsLibCLI); ' % __title__ + \
          'importCode=%s' % str(importCode))

#-----

class OperatorSettingsParser(object):
    '''
    Class to incorporate features in tsOperatorSettingsParser.
    '''
    def __init__(self):
        '''
        Class constructor.
        '''

#-----
```

```
#####
# The following "Help" messages are used to configure each of the
# Python version specific command line parser.
#####

#-----

self.aboutHelp = textwrap.dedent('''

show a summary of the terms & conditions for users of this
software (including the applicable product identification,
authors, contributors, copyrights, licenses and purpose)
and exit

''')

#-----

self.debugHelp = textwrap.dedent('''

log/display application program progress and diagnostic messages
useful in debugging and troubleshooting.
(default = False).

''')

#-----

self.descriptionHelp = textwrap.dedent('''

BACKGROUND

This is a tool to capture current hardware and software information
about the run time environment available to computer programs.

''')

#-----

self.outputHelp = textwrap.dedent('''

log/display current hardware and software information
about the run time environment to this output file
(default = "%s.txt")

''') % self.getRuntimeTitle()

#-----

self.syntaxHelp = textwrap.dedent('''

Syntax:

    <python-interpreter> %prog [Option(s)]

Examples:
```

```
Python      application & option(s)
-----
python      %prog

python2.7   %prog  [-h] [-v] [-a] [-o] [-d] [-V]

python3.3   %prog  [--help] [--version] \\  
                [--about] [--output] \  
                [--debug] [--Verbose]

-----
Key:

    "python"      - Default interpreter for platform

    "python2.7"   - First alternate interpreter for platform

    "python3.3"   - Second alternate interpreter for platform

    "["          - Brackets enclose option keywords and values

    "{}"          - Braces enclose option value choices, if any, and
                    positional arguments, if any

''').replace('%prog', self.getRuntimeTitle())

#-----

self.usageHelp = textwrap.dedent('''

    %prog [Option(s)]

Examples:

    %prog

    %prog  [-h] [-v] [-a] \\  
                [-o] \\  
                [-d] [-V]

    %prog  [--help] [--version] [--about] \\  
                [-output] \\  
                [--debug] [--Verbose]

Purpose:

    Capture current hardware and software information about
    the run time environment available to computer programs.
''').replace('%prog', self.getRuntimeTitle())

#-----

self.verboseHelp = textwrap.dedent('''

log/display verbose troubleshooting details for application
```

```

    program activity tracking diagnostic messages (default = False)

    '''

    #-----

    self.versionHelp = textwrap.dedent('''

    show the build version of this software (including its title,
    version and date) and exit

    ''')

    #-----

def getOptions(self):
    '''
    Parse the command line and return a list of positional arguments
    and a dictionary of keyword options.
    '''
    parser = OptionParser(usage=self.usageHelp)

    #-----

    parser.add_option(
        '-v', '--version',
        action='store_true',
        dest='version',
        default=False,
        help=self.versionHelp)

    #-----

    parser.add_option(
        '-a', '--about',
        action='store_true',
        dest='about',
        default=False,
        help=self.aboutHelp)

    #-----

    parser.add_option(
        '-o', '--output',
        action='store',
        dest='output',
        default='./%s.txt' % self.getRuntimeTitle(),
        type=str,
        help=self.outputHelp)

    #-----

    parser.add_option(
        '-d', '--debug',
        action='store_true',
        dest='debug',

```

```
        default=False,
        help=self.debugHelp)

#-----

parser.add_option(
    '-v', '--Verbose',
    action='store_true',
    dest='Verbose',
    default=False,
    help=self.verboseHelp)

(optionsOptParse, argsOptParse) = parser.parse_args()
if len(argsOptParse) != 0:
    parser.print_help()
    sys.exit(1)

maxArgs = min(0, len(argsOptParse))
if len(argsOptParse) > maxArgs:
    # parser.error("wrong number of arguments")
    extraArgs = []
    for index in range(maxArgs, len(argsOptParse)):
        extraArgs += argsOptParse[index]
    parser.error("\n\n\tinvalid argument(s) = %s" % str(extraArgs))

args = argsOptParse
options = {}
options['about'] = optionsOptParse.about
options['debug'] = optionsOptParse.debug
# options['help'] = optionsOptParse.help
options['output'] = optionsOptParse.output
options['version'] = optionsOptParse.version
options['Verbose'] = optionsOptParse.Verbose

if optionsOptParse.about:
    print('About:\n')

    buildHeader = __header__
    for line in buildHeader.split('\n'):
        print('\t%s' % line)

    print('Purpose:\n')
    for line in __doc__.split('\n'):
        print('\t%s' % string.lstrip(line))

    print('No Error')
    sys.exit(0)

if optionsOptParse.version:
    print('Version:')
    for line in textwrap.wrap(mainTitleVersionDate):
        print('\n\t%s\n' % string.lstrip(line))
    print('No Error')
    sys.exit(0)

return (args, options)
```

```

#-----

def getRunTimeTitle(self):
    '''
    Return Run Time Title, or Build Title, whichever was actually
    used in command line, stripping it of any file path.
    '''
    # Capture Command Line Arguments.
    argv = sys.argv

    # Separate File Path from its associated File Name and Extension
    (filePath, fileNameExt) = os.path.split(argv[0])

    # Separate File Name from its associated Extension
    (fileName, fileExt) = os.path.splitext(fileNameExt)

    if DEBUG:

        fmt1 = 'tsApplication.getRunTimeTitle:'
        fmt2 = '(filePath, fileNameExt) = %s' % (
            str((filePath, fileNameExt)))
        fmt3 = '(fileName, fileExt) = %s' % (
            str((fileName, fileExt)))
        msg = '\n\t%s\n\n\t\t%s\n\n\t\t%s' % (fmt1, fmt2, fmt3)
        self.logger.debug(msg)

    runTimeTitle = fileName
    return (runTimeTitle)

#-----

def getRunTimeTitleVersionDate(self):
    '''
    Return Run Time Title, or Build Title, whichever was actually
    used in command line, stripping it of any file path.
    '''
    runTimeTitle = self.getRunTimeTitle()
    if runTimeTitle == '__title__':

        runTimeTitleVersionDate = mainTitleVersionDate

    else:

        runTimeTitleVersionDate = '%s, v%s (build %s)' % (
            runTimeTitle, __version__, __date__)

    return (runTimeTitleVersionDate)

#-----

if __name__ == '__main__':

    #-----

    def theMainApplication(*args, **kw):

```

```
'''
Display program name, version and date. Receive and validate
command line options and arguments. Display help, error and
status information. Initiate the file directory copy.
'''
theOperatorSettingsParser = OperatorSettingsParser()
print('\n%s\n' % (
    theOperatorSettingsParser.getRuntimeTitleVersionDate()))

(myArgs, myOptions) = theOperatorSettingsParser.getOptions()

myOutputPath = myOptions['output']
myRunTimeEnvironment = tsquery.PlatformRunTimeEnvironment()
myRunTimeEnvironment.logPlatformInfo(fileName= myOutputPath)

sys.stdout.write('\tResults are available in "%s".\n\n' % \
    myOutputPath)

prototype = theMainApplication
myApp = tsCommandLineEnv.CommandLineEnv(

    buildTitle=__title__,
    buildVersion=__version__,
    buildDate=__date__,
    buildAuthors=__authors__,
    buildCopyright=__copyright__,
    buildLicense=__license__,
    buildCredits=__credits__,
    buildTitleVersionDate=mainTitleVersionDate,
    buildHeader=__header__,
    buildPurpose=__doc__,

    enableDefaultCommandLineParser=True,

    logs=[],

    runTimeEntryPoint=prototype)

myApp.Wrapper()
```

1.6.5.2.6 **Python (CLI-mode) "tsWxGTUI_PyVx" tsCommandLineEnv Demo**

EXAMPLE	SOURCE CODE
Command Line Interface launches Python 2.7 "tsWxGTUI_PyVx" (CLI-mode) Application Program	\$ python2.7 test_tsCommandLineEnv.py
	See the following source code.


```

#!/usr/bin/env python
#"Time-stamp: <12/09/2013 8:10:08 AM rsg>"
'''
test_tsCommandLineEnv.py - Test application for class to establish
an environment for a Command Line User Interface.
'''
#####
#
# File: test_tsCommandLineEnv.py
#
# Purpose:
#
#     Test application for class to establish an environment for
#     a Command Line User Interface.
#
# Limitations:
#
#     1) The "tsApplication" module can be used to launch only the
#         Command Line Interface portion of an applications. The
#         application designer is responsible for producing Unix-style
#         exit codes and messages, upon application termination.
#
#     2) The "tsCommandLineEnv" module, a derivative of "tsApplication"
#         can be used to launch only the Command Line Interface portion
#         of an applications. It provides a wrapper that produces Unix-
#         style exit codes and messages, upon application termination.
#
#     3) The "tsWxMultiFrameEnv" module, a derivative of "tsCommand
#         LineEnv" can be used to launch both the Command Line Interface
#         and Graphical-style User Interface portions of an applications.
#         It provides a wrapper that produces Unix-style exit codes and
#         messages, upon application termination.
#
#     4) Command line keyword-value pair option and positional argument
#         parsing uses off-the-shelf, Python version appropriate modules.
#         To facilitate portability of this sample run time environment
#         manager from one Python platform to another, it automatically
#         selects and uses latest one of following:
#
#         a) "argparse" module used with Python 2.7.0 or later
#
#         b) "optparse" module used with Python 2.3.0 or later
#
#         c) "getopt" module used with Python 1.6.0 or later
#
# Notes:
#
#     1) "tsApplication" is a base class for launching the appli-
#         cation specified by an operator. It initializes and con-
#         figures the application using the following keyword
#         value pairs and positional arguments:
#
#         a) Input provided on the command line by an operator. The
#            command line uses a Unix-style, free-format to promote

```

```
#         future enhancement and on-going maintenance.
#
#         b) Input provided in the parameter list of the applica-
#            tion's invocation of the class instantiation. The par-
#            ameter list uses a Python-style free-format to promote
#            future enhancement and on-going maintenance.
#
# 2) "tsCommandlineEnv" is a convenience package wrapping term-
#    inal keyboard input, video display scrolled text output,
#    "tsLogger" and "tsException" services. It is a base class
#    for "tsWxMultiFrameEnv".
#
# 3) "tsWxMultiFrameEnv" is a convenience package wrapping
#    terminal keyboard & mouse input, video display row and
#    column addressable, field-editable output, "tsLogger"
#    and "tsException" services.
#
# 4) Standardized command line option parsing methods return
#    a dictionary, of keyword value pairs, and a list, of
#    positional arguments in the manner of the "optparse"
#    module. This should facilitate application support for
#    the evolving Python command line option parsing modules.
#    However, this requires that "tsApplication" stubs and
#    application parsing methods include a small amount of
#    extra code for the appropriate "argparse", "optparse"
#    and "getopt" output format conversion.
#
# 5) The various command line parser modules produce usage
#    help. The content and format may vary based on the
#    parser's capabilities and limitations. Considerable
#    care needs to be taken to minimize the difference in
#    order to produce a software product that retains its
#    look and feel for the end-user, regardless of the
#    hardware and software platform being used.
#
# Usage (example):
#
#     ## Import Module
#     from tsCommandLineEnv import CommandLineEnv
#
#     ## Generalized Form to Instantiate and Launch an application
#     ## module that uses a Command Line Interface (CLI) to a
#     ## character-mode terminal with optional logging.
#     ##
#     ## See this File's header for examples of those application
#     ## specific source code descriptions associated with
#     ## parameter identifiers having double-underscore ("__")
#     ## prefix and suffix.
#
#     myApp = CommandLineEnv(
#         # All applications (with Command Line Interface or
#         # Graphical-style User Interface) begin with the following
#         # Command Line Interface Launch configuration item list:
#
#         buildTitle=__title__,
#         buildVersion=__version__,
```

```

#         buildDate=__date__,
#         buildAuthors=__authors__,
#         buildCopyright=__copyright__,
#         buildLicense=__license__,
#         buildCredits=__credits__,
#         buildTitleVersionDate=mainTitleVersionDate,
#         buildHeader=__header__,
#         buildPurpose=__doc__,
#
#         #####
#
#         # Python version appropriate Command Line Interface
#         # module(s) may be enabled to obtain non-Application-
#         # specific Keyword-Value pair Options and Positional
#         # Arguments and associated command line help:
#
#         #     "argparse" module - introduced with Python 2.7.0
#         #     "optparse" module - introduced with Python 2.3.0
#         #     "getopt"  module - introduced with Python 1.6.0
#
#         enableDefaultCommandLineParser=False # Disable unless True
#
#         #####
#
#         # When customized logging is appropriate, some applica-
#         # tions use the following application-specific Launch
#         # configuration item:
#
#         logs=['1st-Non-Default', ..., 'Nth-Non-Default'],
#
#         # When basic logging is appropriate, some applications
#         # use the following non-application-specific Launch
#         # configuration item:
#
#         logs=[],
#
#         # All applications (with Command Line Interface or
#         # Graphical-style User Interface) wrapup their Configuration
#         # item list as follows:
#
#         runTimeEntryPoint=main)
#
#         ## Launch via reference to appropriate Module Method
#         myApp.Wrapper()
#
# CLI Methods:
#
#         exitTest - Optional Simulated Input / Output Exception to
#                   induce termination with an exit code and message.
#
#         mainTest - Command Line Interface. It gathers and displays
#                   operator input as keyword-value pair options and
#                   positional arguments.
#
#         EntryPoint - Application Programming Interface, It
#                   gathers and displays configuration parameters as

```

```
#             keyword-value pair options and positional arguments.
#
# Modifications:
#
#     2013/12/08 rsg Initial version.
#
# ToDo:
#
#     2013/05/04 rsg Merge command line parser change(s) from
#                 "test_tsApplication", as corrections occur.
#
#####

__title__      = 'test_tsCommandLineEnv'
__version__    = '1.0.0'
__date__       = '12/08/2013'
__authors__    = 'Richard S. Gordon'
__copyright__  = 'Copyright (c) 2007-2013 ' + \
                 '%s.\n\t\tAll rights reserved.' % __authors__
__license__    = 'GNU General Public License, ' + \
                 'Version 3, 29 June 2007'
__credits__    = '\n\n Credits: ' + \
                 '\n\n\t tsLibCLI Import & Application Launch Features: ' + \
                 '\n\t Copyright (c) 2007-2009 Frederick A. Kier. ' + \
                 '\n\t\tAll rights reserved.' + \
                 '\n\n\t Python Logging Module API Features: ' + \
                 '\n\t Copyright (c) 2001-2013 ' + \
                 'Python Software Foundation.' + \
                 '\n\t\tAll rights reserved.' + \
                 '\n\t PSF License Agreement for Python 2.7.3 & 3.3.0'

__line1__ = '%s, v%s (build %s)' % (__title__, __version__, __date__)
__line2__ = 'Author(s): %s' % __authors__
__line3__ = '%s' % __copyright__

if len(__credits__) == 0:
    __line4__ = '%s' % __license__
else:
    __line4__ = '%s%s' % (__license__, __credits__)

__header__ = '\n\n%s\n\n %s\n %s\n %s\n' % (__line1__,
                                             __line2__,
                                             __line3__,
                                             __line4__)

mainTitleVersionDate = __line1__

#-----

import os.path
import platform
import sys

#-----

try:
```

```

import tsLibCLI

except ImportError, importCode:

    print('%s: ImportError (tsLibCLI); ' % __title__ + \
          'importCode=%s' % str(importCode))

try:

    import tsExceptions as tse
    import tsLogger as Logger

    import tsOperatorSettingsParser

    from tsCommandLineEnv import CommandLineEnv

except ImportError, importCode:

    print('%s: ImportError (tsLibCLI); ' % __title__ + \
          'importCode=%s' % str(importCode))

#-----

__help__ = '''
This program demonstrates an environment for a Command Line User Interface.
'''

DEBUG = False

DebugKeyValueUndefined      = False
DebugSimulatedKeyErrorTrap = False
EnableOptionsGNU = True

theAssignedLogger = None

runTimeTitleEnabled = True
tracebackEnabled = False

#-----

if __name__ == "__main__":

    #-----

    def exitTest():
        '''
        Simulated Input / Output Exception to induce termination
        with an exit code and message.
        '''

        exceptionName = tse.INPUT_OUTPUT_EXCEPTION
        errorName = 'Oops! Invalid Name'

        myLogger = Logger.TsLogger(threshold=Logger.DEBUG,
```

```
name='exitTest')

message = 'ExitTest'

myLogger.debug('***** ExitTest %s / %s *****' % (
    exceptionName, errorName))
raise tse.InputOutputException(errorName, message)
##
myParser = tsOperatorSettingsParser.TsOperatorSettingsParser()
##
msg = '\n%s\n' % myParser.getRuntimeTitleVersionDate()
##
print(msg)
##
myLogger.debug(msg)

##
rawArgsOptions = sys.argv[1:]
##
msg = '\trawArgsOptions=%s' % str(rawArgsOptions)
##
print(msg)
##
myLogger.debug(msg)
##
maxArgs = len(rawArgsOptions)

##
(args, options) = myParser.parseCommandLineDispatch()
##
msg = 'type(args=%s)=%s' % (str(args), type(args))
##
print(msg)
##
myLogger.debug(msg)

##
msg = 'type(options=%s)=%s' % (str(options), type(options))
##
print(msg)
##
myLogger.debug(msg)

##
if True or DEBUG:
##
    label = myParser.getRuntimeTitle()

##
    fmt1 = '%s.mainTest (parameter list): ' % label
##
    fmt2 = 'args=%s;\n\t\t\tkw=%s' % (str(args), str(kw))
##
    msg = '\n\t\t\t%s\n\t\t\t%s' % (fmt1, fmt2)
##
    print(msg)
##
    myLogger.debug(msg)

##
    fmt1 = '%s.mainTest (command line argv): ' % label
##
    fmt2 = 'args=%s' % str(args)
##
    fmt3 = 'options (unsorted)=%s' % str(options)
##
    msg = '\n\t\t\t%s\n\t\t\t%s;\n\t\t\t%s' % (fmt1, fmt2, fmt3)
##
    print(msg)
##
    myLogger.debug(msg)

##
    fmt1 = '\n\t\t\t%s.mainTest (command line argv): ' % label
##
    fmt2 = '\n\t\t\t\targs (positional) =%s' % str(args)
##
    keys = sorted(options.keys())
##
    text = ''
##
    for key in keys:
##
        try:
##
            value = '"%s"' % options[key]
##
        except Exception, errorCode:
##
            value = ''
##
        if text == '':
##
            text = '{\n\t\t\t\t\t%s: %s' % (str(key), str(value))
##
        else:
##
            text += ', \n\t\t\t\t\t%s: %s' % (str(key), str(value))
```

```

##         text += '}'
##         fmt3 = '\n\t\toptions (sorted)= %s' % text
##         msg = fmt1 + fmt2 + fmt3
##         print(msg)
##         myLogger.debug(msg)

##         return (args, options)

#-----

def mainTest(*args, **kw):
    '''
    Command Line Interface. It gathers and displays operator
    input as keyword-value pair options and positional arguments.
    '''
    myLogger = Logger.TsLogger(threshold=Logger.DEBUG,
                               name='mainTest')
    myParser = tsOperatorSettingsParser.TsOperatorSettingsParser()
    msg = '\n%s\n' % myParser.getRunTimeTitleVersionDate()
    print(msg)
    myLogger.debug(msg)

    rawArgsOptions = sys.argv[1:]
    msg = '\trawArgsOptions=%s' % str(rawArgsOptions)
    print(msg)
    myLogger.debug(msg)
    maxArgs = len(rawArgsOptions)

    (args, options) = myParser.parseCommandLineDispatch()
    msg = 'type(args=%s)=%s' % (str(args), type(args))
    print(msg)
    myLogger.debug(msg)

    msg = 'type(options=%s)=%s' % (str(options), type(options))
    print(msg)
    myLogger.debug(msg)

    if True or DEBUG:
        label = myParser.getRunTimeTitle()

        fmt1 = '%s.mainTest (parameter list): ' % label
        fmt2 = 'args=%s;\n\t\t\tkw=%s' % (str(args), str(kw))
        msg = '\n\t%s\n\t\t\t%s' % (fmt1, fmt2)
        print(msg)
        myLogger.debug(msg)

        fmt1 = '%s.mainTest (command line argv): ' % label
        fmt2 = 'args=%s' % str(args)
        fmt3 = 'options (unsorted)=%s' % str(options)
        msg = '\n\t%s\n\t\t\t%s;\n\t\t\t%s' % (fmt1, fmt2, fmt3)
        print(msg)
        myLogger.debug(msg)

        fmt1 = '\n\t%s.mainTest (command line argv): ' % label
        fmt2 = '\n\t\t\targs (positional) =%s' % str(args)
        keys = sorted(options.keys())

```



```
#-----  
  
myApp = CommandLineEnv(  
    buildTitle=__title__,  
    buildVersion=__version__,  
    buildDate=__date__,  
    buildAuthors=__authors__,  
    buildCopyright=__copyright__,  
    buildLicense=__license__,  
    buildCredits=__credits__,  
    buildTitleVersionDate=mainTitleVersionDate,  
    buildHeader=__header__,  
    buildPurpose=__doc__,  
#  
    enableDefaultCommandLineParser=False, # Disable unless True  
#  
    logs=[],  
#  
    runTimeEntryPoint=EntryPoint)  
  
myApp.Wrapper()
```

1.6.5.3 TUI Low-Level Curses Mode Examples

1.6.5.3.1 Python (TUI-mode) "Curses" TextPad I/O Demo

From <http://docs.python.org/2/library/curses.html> (<http://docs.python.org/2/library/curses.html>):

"15.11. curses — Terminal handling for character-cell displays

Platforms: Unix

Changed in version 1.6: Added support for the ncurses library and converted to a package.

The curses module provides an interface to the curses library, the de-facto standard for portable advanced terminal handling.

While curses is most widely used in the Unix environment, versions are available for DOS, OS/2, and possibly other systems as well. This extension module is designed to match the API of ncurses, an open-source curses library hosted on Linux and the BSD variants of Unix.

Note

Since version 5.4, the ncurses library decides how to interpret non-ASCII data using the `nl_langinfo` function. That means that you have to call `locale.setlocale()` in the application and encode Unicode strings using one of the system's available encodings. This example uses the system's default encoding:

EXAMPLE	SOURCE CODE
Command Line Interface launches Python 2.7 "Curses" (TUI-mode) Application Program	\$ <code>python2.7 test_tsCursesSample.py</code>
	See the following source code.

```

#!/usr/bin/env python
# "Time-stamp: <12/09/2013  9:20:38 AM rsg>"
__doc__ = '''
test_tsCursesSample.py - Test application for Python "Curses"
module.

From http://docs.python.org/2/library/curses.html:
15.11. curses --- Terminal handling for character-cell displays

Boilerplate, comments and bug fixes added by Richard S. Gordon.
'''

#####
#
# File: test_tsCursesSample.py
#
# Purpose:
#
#     Test application for Python "Curses" module.
#
# Usage (example):
#
#     # Execute
#
#     python test_tsCursesSample.py
#
# Capabilities:
#
#     1. Initialize and startup the standard terminal display screen.
#
#     2. Create a curses window and editpad.
#
#     3. Set or clear curses echo mode,
#
#     4. Input from terminal keyboard.
#
#     5. Output to terminal display screen.
#
#     6. Shutdown and terminate the standard terminal display screen.
#
# Limitations:
#
#     1. Does not reveal window and edit box borders.
#
# Notes:
#
#     None.
#
# Classes:
#
#     None.
#
# Methods:
#
#     TBD.
#

```

```
# Modifications:
#
#     2013/12/09 rsg Initial version.
#
# ToDo:
#
#     TBD.
#
#####

__title__      = 'test_tsCursesSample'
__version__    = '1.0.0'
__date__       = '12/09/2013'
__authors__    = 'Richard S. Gordon'
__copyright__  = 'Copyright (c) 2013 ' + \
                 '%s.\n\t\tAll rights reserved.' % __authors__
__license__    = 'GNU General Public License, ' + \
                 'Version 3, 29 June 2007'
__credits__    = '\n\n Credits: ' + \
                 '\n\n\t terminalsize (https://gist.github.com/ + \
                 'jtriley/1108174) Features: ' + \
                 '\n\t Copyright (c) 2011 Justin T. Riley.' + \
                 '\n\t\tAll rights reserved.' + \
                 '\n\t GNU General Public License, ' + \
                 'Version 3, 29 June 2007'

__line1__ = '%s, v%s (build %s)' % (__title__, __version__, __date__)
__line2__ = 'Author(s): %s' % __authors__
__line3__ = '%s' % __copyright__

if len(__credits__) == 0:
    __line4__ = '%s' % __license__
else:
    __line4__ = '%s%s' % (__license__, __credits__)

__header__ = '\n\n%s\n\n %s\n %s\n %s\n' % (__line1__,
                                             __line2__,
                                             __line3__,
                                             __line4__)

mainTitleVersionDate = __line1__

#-----

## Note

##     Since version 5.4, the ncurses library decides how to interpret
##     non-ASCII data using the nl_langinfo function. That means that you
##     have to call locale.setlocale() in the application and encode
##     Unicode strings using one of the system's available encodings.
##     This example uses the system's default encoding:

import locale
locale.setlocale(locale.LC_ALL, '')
code = locale.getpreferredencoding()
```

```

import curses
import curses.textpad
import time

enableTextPad = True
ms = 15 * 1000

# Initialize and startup
# the curses terminal screen
stdscr = curses.initscr()

# Since user keyboard input will be
# output to the curses terminal screen,
# by this program, disable or enable
# the optional echo of the input by
# the curses terminal interface driver.
curses.noecho()
#curses.echo()

# Define an area on the curses terminal
# screen at column 20, row 7 that is
# 40 columns wide by 5 rows high
begin_x = 20
begin_y = 7
height = 5
width = 40
win = curses.newwin(height, width, begin_y, begin_x)

if enableTextPad:

    # Create an interactive area for curses to
    # receive and display user keyboard input.
    # The curses.textpad enables the user to
    # use emacs control keys to edit the input.
    tb = curses.textpad.Textbox(win)
    text = tb.edit()
    msg = 'TextPad returned: \n"%s"' % text
    # curses.addstr(4,1,text.encode('utf_8'))
    stdscr.addstr(4,1,msg.encode('utf_8'))

    stdscr.refresh()
    curses.doupdate()
    curses.napms(ms)

else:

    hw = "Hello world!"
    msg = hw
    stdscr.addstr(4,1,msg.encode('utf_8'))

    stdscr.refresh()
    curses.doupdate()
    curses.napms(ms)

while 1:

```

```
c = stdscr.getch()

if c == ord('p'):

    pass

elif c == ord('q'):
    break # Exit the while()

elif c == curses.KEY_HOME:

    x = y = 0

stdscr.refresh()
curses.doupdate()
curses.napms(ms)

##win.show(True)

#
# Shutdown and terminate
# the curses terminal screen
curses.endwin()
```

1.6.5.3.2 Python (TUI-mode) "Curses" Multi-Panel Cotrol Dmo

EXAMPLE	SOURCE CODE
Command Line Interface launches Python 2.7 "Curses" (TUI-mode) Application Program	<code>\$ python2.7 test_nCursesDemo.py</code>
	See the following source code.

```
#!/usr/bin/env python
# "Time-stamp: <12/09/2013 6:13:41 AM rsg>"
__doc__ = '''
test_nCursesDemo.py - Test application for Python "Curses"
module.

From http://docs.python.org/2/library/curses.html:
15.11. curses --- Terminal handling for character-cell displays

Boilerplate, comments and bug fixes added by Richard S. Gordon.
'''

#####
#
# File: test_nCursesDemo.py
#
# Purpose:
#
#     Test application for Python "Curses" module.
#
# Usage (example):
#
#     # Execute
#
#     python test_tsCursesSampl.py
#
# Capabilities:
#
#     1. Initialize and startup the standard terminal display screen.
#
#     2. Create a curses window and multiple panels.
#
#     3. Waits for various input from terminal keyboard.
#
#         Changes panel layout.
#
#         Output panel layout to terminal display screen.
#
#     4. Shutdown and terminate the standard terminal display screen.
#
#     5. Displays __heading__ with Copyright, Credits, License etc.
#
# Limitations:
#
#     None.
#
# Notes:
#
#     None.
#
# Classes:
#
#     None.
#
# Methods:
#
```

```
#      TBD.
#
# Modifications:
#
#      2013/12/09 rsg Initial version.
#
# ToDo:
#
#      TBD.
#
#####

__title__      = 'test_nCursesDemo'
__version__    = '1.0.0'
__date__       = '12/09/2013'
__authors__    = 'Richard S. Gordon'
__copyright__  = 'Copyright (c) 2013 ' + \
                 '%s.\n\t\tAll rights reserved.' % __authors__
__license__    = 'GNU General Public License, ' + \
                 'Version 3, 29 June 2007'
__credits__    = '\n\n Credits: ' + \
                 '\n\n\t Python Curses Module API Features: ' + \
                 '\n\t Copyright (c) 2001-2013 ' + \
                 'Python Software Foundation.' + \
                 '\n\t\tAll rights reserved.' + \
                 '\n\t PSF License Agreement for Python 2.7.3 & 3.3.0'

__line1__ = '%s, v%s (build %s)' % (__title__, __version__, __date__)
__line2__ = 'Author(s): %s' % __authors__
__line3__ = '%s' % __copyright__

if len(__credits__) == 0:
    __line4__ = '%s' % __license__
else:
    __line4__ = '%s%s' % (__license__, __credits__)

__header__ = '\n\n%s\n\n %s\n %s\n %s\n' % (__line1__,
                                             __line2__,
                                             __line3__,
                                             __line4__)

mainTitleVersionDate = __line1__

#-----

## Note

##      Since version 5.4, the ncurses library decides how to interpret
##      non-ASCII data using the nl_langinfo function. That means that you
##      have to call locale.setlocale() in the application and encode
##      Unicode strings using one of the system's available encodings.
##      This example uses the system's default encoding:

import locale
locale.setlocale(locale.LC_ALL, '')
code = locale.getpreferredencoding()
```



```

###!/usr/bin/env python
###
### $Id$
###
### (n)curses exerciser in Python, an interactive test for the curses
### module. Currently, only the panel demos are ported.

import curses
from curses import panel

def wGetchar(win = None):
    if win is None: win = stdscr
    return win.getch()

def Getchar():
    wGetchar()

#
# Panels tester
#
def wait_a_while():
    if nap_msec == 1:
        Getchar()
    else:
        curses.napms(nap_msec)

def saywhat(text):
    stdscr.move(curses.LINES - 1, 0)
    stdscr.clrtoeol()
    stdscr.addstr(text)

def mkpanel(color, rows, cols, tly, tlx):
    win = curses.newwin(rows, cols, tly, tlx)
    pan = panel.new_panel(win)
    if curses.has_colors():
        if color == curses.COLOR_BLUE:
            fg = curses.COLOR_WHITE
        else:
            fg = curses.COLOR_BLACK
        bg = color
        curses.init_pair(color, fg, bg)
        win.bkgdset(ord(' '), curses.color_pair(color))
    else:
        win.bkgdset(ord(' '), curses.A_BOLD)

    return pan

def pflush():
    panel.update_panels()
    curses.doupdate()

def fill_panel(pan):
    win = pan.window()
    num = pan.userptr()[1]

    win.move(1, 1)

```

```
win.addstr("-pan%c-" % num)
win.clrtoeol()
win.box()

maxy, maxx = win.getmaxyx()
for y in range(2, maxy - 1):
    for x in range(1, maxx - 1):
        win.move(y, x)
        win.addch(num)

def demo_panels(win):
    global stdscr, nap_msec, mod
    stdscr = win
    nap_msec = 1
    mod = ["test", "TEST", "(*)", "*(*)", "<-->", "LAST"]

    stdscr.refresh()

    for y in range(0, curses.LINES - 1):
        for x in range(0, curses.COLS):
            stdscr.addstr("%d" % ((y + x) % 10))
    for y in range(0, 1):
        p1 = mkpanel(curses.COLOR_RED,
                     curses.LINES // 2 - 2,
                     curses.COLS // 8 + 1,
                     0,
                     0)
        p1.set_userptr("p1")

        p2 = mkpanel(curses.COLOR_GREEN,
                     curses.LINES // 2 + 1,
                     curses.COLS // 7,
                     curses.LINES // 4,
                     curses.COLS // 10)
        p2.set_userptr("p2")

        p3 = mkpanel(curses.COLOR_YELLOW,
                     curses.LINES // 4,
                     curses.COLS // 10,
                     curses.LINES // 2,
                     curses.COLS // 9)
        p3.set_userptr("p3")

        p4 = mkpanel(curses.COLOR_BLUE,
                     curses.LINES // 2 - 2,
                     curses.COLS // 8,
                     curses.LINES // 2 - 2,
                     curses.COLS // 3)
        p4.set_userptr("p4")

        p5 = mkpanel(curses.COLOR_MAGENTA,
                     curses.LINES // 2 - 2,
                     curses.COLS // 8,
                     curses.LINES // 2,
                     curses.COLS // 2 - 2)
        p5.set_userptr("p5")
```

```
fill_panel(p1)
fill_panel(p2)
fill_panel(p3)
fill_panel(p4)
fill_panel(p5)
p4.hide()
p5.hide()
pflush()
saywhat("press any key to continue")
wait_a_while()

saywhat("h3 s1 s2 s4 s5;press any key to continue")
p1.move(0, 0)
p3.hide()
p1.show()
p2.show()
p4.show()
p5.show()
pflush()
wait_a_while()

saywhat("s1; press any key to continue")
p1.show()
pflush()
wait_a_while()

saywhat("s2; press any key to continue")
p2.show()
pflush()
wait_a_while()

saywhat("m2; press any key to continue")
p2.move(curses.LINES // 3 + 1, curses.COLS // 8)
pflush()
wait_a_while()

saywhat("s3; press any key to continue")
p3.show()
pflush()
wait_a_while()

saywhat("m3; press any key to continue")
p3.move(curses.LINES // 4 + 1, curses.COLS // 15)
pflush()
wait_a_while()

saywhat("b3; press any key to continue")
p3.bottom()
pflush()
wait_a_while()

saywhat("s4; press any key to continue")
p4.show()
pflush()
wait_a_while()
```

```
saywhat("s5; press any key to continue")
p5.show()
pflush()
wait_a_while()

saywhat("t3; press any key to continue")
p3.top()
pflush()
wait_a_while()

saywhat("t1; press any key to continue")
p1.show()
pflush()
wait_a_while()

saywhat("t2; press any key to continue")
p2.show()
pflush()
wait_a_while()

saywhat("t3; press any key to continue")
p3.show()
pflush()
wait_a_while()

saywhat("t4; press any key to continue")
p4.show()
pflush()
wait_a_while()

for itmp in range(0, 6):
    w4 = p4.window()
    w5 = p5.window()

    saywhat("m4; press any key to continue")
    w4.move(curses.LINES // 8, 1)
    w4.addstr(mod[itmp])
    p4.move(curses.LINES // 6, itmp * curses.COLS // 8)
    w5.move(curses.LINES // 6, 1)
    w5.addstr(mod[itmp])
    pflush()
    wait_a_while()

    saywhat("m5; press any key to continue")
    w4.move(curses.LINES // 6, 1)
    w4.addstr(mod[itmp])
    p5.move(curses.LINES // 3 - 1, itmp * 10 + 6)
    w5.move(curses.LINES // 8, 1)
    w5.addstr(mod[itmp])
    pflush()
    wait_a_while()

saywhat("m4; press any key to continue")
p4.move(curses.LINES // 6, (itmp + 1) * curses.COLS // 8)
pflush()
```

```
wait_a_while()

saywhat("t5; press any key to continue")
p5.top()
pflush()
wait_a_while()

saywhat("t2; press any key to continue")
p2.top()
pflush()
wait_a_while()

saywhat("t1; press any key to continue")
p1.top()
pflush()
wait_a_while()

saywhat("d2; press any key to continue")
del p2
pflush()
wait_a_while()

saywhat("h3; press any key to continue")
p3.hide()
pflush()
wait_a_while()

saywhat("d1; press any key to continue")
del p1
pflush()
wait_a_while()

saywhat("d4; press any key to continue")
del p4
pflush()
wait_a_while()

saywhat("d5; press any key to continue")
del p5
pflush()
wait_a_while()
if nap_msec == 1:
    break
nap_msec = 100

#
# one fine day there'll be the menu at this place
#
curses.wrapper(demo_panels)
print(__header__)
```

1.6.5.4 GUI High-Level wxPython Mode Examples

1.6.5.4.1 Python (GUI-mode) "wxPython" Simple Frame Demo

EXAMPLE	SOURCE CODE
Command Line Interface launches Python 2.7 "wxPython" (GUI-mode) Application Program	\$ python2.7 the_wxPython_Main_Program.py
	<pre>#!/usr/bin/env python #----- import wx class TestFrame(wx.Frame): def __init__(self, parent, title): wx.Frame.__init__(self, parent, wx.ID_ANY, title=title) text = wx.StaticText(self, label="Hello, world!") #----- if __name__ == '__main__': app = wx.App(redirect=False) frame = TestFrame(None, "Hello, world!") frame.Show() app.MainLoop()</pre>

1.6.5.4.2 Python (GUI-mode) "tsWxGTUI_PyVx" tsWxMultiFrameEnv Demo

EXAMPLE	SOURCE CODE
Command Line Interface launches Python 2.7 "tsWxGTUI_PyVx" (GUI- mode) Application Program	\$ python2.7 test_tsWxMultiFrameEnv.py
	See the following source code.

```

#!/usr/bin/env python
#"Time-stamp: <12/09/2013  8:14:13 AM rsg>"
'''
test_tsWxMultiFrameEnv.py - Test application for class to
establish environment for a Multi-Frame Graphical User Interface.
It creates a simple Frame and displays "Hello World!" as
StaticText.
'''

#####
#
# File: test_tsWxMultiFrameEnv.py
#
# Purpose:
#
#     Test application for class to establish environment for
#     a Multi-Frame Graphical User Interface. It creates a simple
#     Frame and displays "Hello World!" as StaticText.
#
# Limitations:
#
#     1) The "tsApplication" module can be used to launch only the
#         Command Line Interface portion of an applications. The
#         application designer is responsible for producing Unix-style
#         exit codes and messages, upon application termination.
#
#     2) The "tsCommandLineEnv" module, a derivative of "tsApplication"
#         can be used to launch only the Command Line Interface portion
#         of an applications. It provides a wrapper that produces Unix-
#         style exit codes and messages, upon application termination.
#
#     3) The "tsWxMultiFrameEnv" module, a derivative of "tsCommand
#         LineEnv" can be used to launch both the Command Line Interface
#         and Graphical-style User Interface portions of an applications.
#         It provides a wrapper that produces Unix-style exit codes and
#         messages, upon application termination.
#
#     4) Command line keyword-value pair option and positional argument
#         parsing uses off-the-shelf, Python version appropriate modules.
#         To facilitate portability of this sample run time environment
#         manager from one Python platform to another, it automatically
#         selects and uses latest one of following:
#
#         a) "argparse" module used with Python 2.7.0 or later
#
#         b) "optparse" module used with Python 2.3.0 or later
#
#         c) "getopt" module used with Python 1.6.0 or later
#
# Notes:
#
#     1) "tsApplication" is a base class for launching the appli-
#         cation specified by an operator. It initializes and con-
#         figures the application using the following keyword
#         value pairs and positional arguments:

```

```
#
#
#     a) Input provided on the command line by an operator. The
#         command line uses a Unix-style, free-format to promote
#         future enhancement and on-going maintenance.
#
#     b) Input provided in the parameter list of the applica-
#         tion's invocation of the class instantiation. The par-
#         ameter list uses a Python-style free-format to promote
#         future enhancement and on-going maintenance.
#
# 2) "tsCommandlineEnv" is a convenience package wrapping term-
#     inal keyboard input, video display scrolled text output,
#     "tsLogger" and "tsException" services. It is a base class
#     for "tsWxMultiFrameEnv".
#
# 3) "tsWxMultiFrameEnv" is a convenience package wrapping
#     terminal keyboard & mouse input, video display row and
#     column addressable, field-editable output, "tsLogger"
#     and "tsException" services.
#
# 4) Standardized command line option parsing methods return
#     a dictionary, of keyword value pairs, and a list, of
#     positional arguments in the manner of the "optparse"
#     module. This should facilitate application support for
#     the evolving Python command line option parsing modules.
#     However, this requires that "tsApplication" stubs and
#     application parsing methods include a small amount of
#     extra code for the appropriate "argparse", "optparse"
#     and "getopt" output format conversion.
#
# 5) The various command line parser modules produce usage
#     help. The content and format may vary based on the
#     parser's capabilities and limitations. Considerable
#     care needs to be taken to minimize the difference in
#     order to produce a software product that retains its
#     look and feel for the end-user, regardless of the
#     hardware and software platform being used.
#
# Usage (example):
#
# #####
#
# ## Import Module
# from tsWxMultiFrameEnv import MultiFrameEnv
#
# ## Generalized Form to Instantiate and Launch an application
# ## module that uses a Command Line Interface (CLI) to a
# ## character-mode terminal with optional logging.
# ##
# ## Configuration options enable the CLI application to launch
# ## and use the same character-mode terminal with a Graphical-
# ## style User Interface (GUI).
# ##
# ## See this File's header for examples of those application
# ## specific source code descriptions associated with
# ## parameter identifiers having double-underscore ("__")
```



```

#     ## prefix and suffix.
#     ##
#     ## See the test_tsWxWidgets.py File's header for examples of
#     ## the gui application options.
#
myApp = MultiFrameEnv(
#     #####
#     # All applications (with Command Line Interface or
#     # Graphical-style User Interface) begin with the following
#     # Command Line Interface Launch configuration item list:
#
#     buildTitle=__title__,
#     buildVersion=__version__,
#     buildDate=__date__,
#     buildAuthors=__authors__,
#     buildCopyright=__copyright__,
#     buildLicense=__license__,
#     buildCredits=__credits__,
#     buildTitleVersionDate=mainTitleVersionDate,
#     buildHeader=__header__,
#     buildPurpose=__doc__,
#
#     #####
#
#     # Python version appropriate Command Line Interface
#     # module(s) may be enabled to obtain non-Application-
#     # specific Keyword-Value pair Options and Positional
#     # Arguments and associated command line help:
#
#     #     "argparse" module - introduced with Python 2.7.0
#     #     "optparse" module - introduced with Python 2.3.0
#     #     "getopt"  module - introduced with Python 1.6.0
#
#     enableDefaultCommandLineParser=False # Disable unless True
#
#     #####
#
#     # When appropriate, some applications also use the following
#     # Graphical-style User Interface Launch configuration item list:
#
#     guiRequired=True,
#     guiTopLevelObjectId=-1,
#     guiMessageRedirect=True,
#     guiMessageFilename=None,
#     guiTopLevelObject=_Communicate,
#     guiTopLevelObjectName='Sample',
#     guiTopLevelObjectTitle='widgets_communicate',
#
#     #####
#
#     # When customized logging is appropriate, some applica-
#     # tions use the following application-specific Launch
#     # configuration item:
#
#     logs=['1st-Non-Default', ..., 'Nth-Non-Default'],
#

```

```
#           # When basic logging is appropriate, some applications
#           # use the following non-application-specific Launch
#           # configuration item:
#
#           logs=[],
#
#           #####
#           # All applications, with Command Line Interface or with
#           # both Command Line and Graphical-style User Interfaces,
#           # wrapup their Configuration item list as follows:
#
#           runTimeEntryPoint=main)
#
#           #####
#
#           ## Launch via reference to appropriate Module Method
#           myApp.Wrapper()
#
# CLI Methods:
#
#           exitTest - Optional Simulated Input / Output Exception to
#                   induce termination with an exit code and message.
#
#           mainTest - Command Line Interface. It gathers and displays
#                   operator input as keyword-value pair options and
#                   positional arguments.
#
#           EntryPoint - Application Programming Interface, It
#                   gathers and displays configuration parameters as
#                   keyword-value pair options and positional arguments.
#
# GUI Class & Methods:
#
#           _Prototype          - Class to establish the frame that
#                               contains the application specific
#                               graphical components.
#
#           _Prototype.OnAbout  - Event Handler for Mouse Click on About button
#
#           _Prototype.OnHelp   - Event Handler for Mouse Click on Help button
#
#           _Prototype.OnMove   - Unused Event Handler for Window Re-sizing
#
#           _Prototype.OnQuit   - Event Handler for Mouse Click on Close button
#
#           _Prototype.__init__ - Class constructor
#
#           _Prototype.tsGetTheId - Return ID associated with class instance
#
#           nextWindowId - Generates a unique GUI object Id
#
# Modifications:
#
#           2013/12/08 rsg Initial version.
#
```

```

# ToDo:
#
#     None.
#
#####

__title__      = 'test_tsWxMultiFrameEnv'
__version__    = '1.0.0'
__date__       = '12/08/2013'
__authors__    = 'Richard S. Gordon'
__copyright__  = 'Copyright (c) 2007-2013 ' + \
                 '%s.\n\t\tAll rights reserved.' % __authors__
__license__    = 'GNU General Public License, ' + \
                 'Version 3, 29 June 2007'
__credits__    = '\n\n Credits: ' + \
                 '\n\n\t tsLibGUI Import & Application Launch Features: ' + \
                 '\n\t Copyright (c) 2007-2009 Frederick A. Kier.' + \
                 '\n\t\t\tAll rights reserved.' + \
                 '\n\n\t Python Curses Module API & ' + \
                 'Run Time Library Features:' + \
                 '\n\t Copyright (c) 2001-2013 ' + \
                 'Python Software Foundation.' + \
                 '\n\t\t\tAll rights reserved.' + \
                 '\n\t PSF License Agreement for Python 2.7.3 & 3.3.0' + \
                 '\n\n\t wxWidgets (formerly wxWindows) & ' + \
                 'wxPython API Features:' + \
                 '\n\t Copyright (c) 1992-2008 Julian Smart, ' + \
                 'Robert Roebling,' + \
                 '\n\t\t\tVadim Zeitlin and other members of the ' + \
                 '\n\t\t\t\t\twxWidgets team.' + \
                 '\n\t\t\t\t\tAll rights reserved.' + \
                 '\n\t wxWindows Library License' + \
                 '\n\n\t nCurses API & Run Time Library Features:' + \
                 '\n\t Copyright (c) 1998-2011 ' + \
                 'Free Software Foundation, Inc.' + \
                 '\n\t\t\t\t\tAll rights reserved.' + \
                 '\n\t GNU General Public License, ' + \
                 'Version 3, 29 June 2007'

__line1__ = '%s, v%s (build %s)' % (__title__, __version__, __date__)
__line2__ = 'Author(s): %s' % __authors__
__line3__ = '%s' % __copyright__

if len(__credits__) == 0:
    __line4__ = '%s' % __license__
else:
    __line4__ = '%s%s' % (__license__, __credits__)

__header__ = '\n\n%s\n\n %s\n %s\n %s\n' % (__line1__,
                                              __line2__,
                                              __line3__,
                                              __line4__)

mainTitleVersionDate = __line1__

#-----

```

```
import os.path
import sys
import time
import traceback

#-----

# Enable Command Line Interface Application Launch Support
#
# NOTE: tsCommandLineEnv is a convenience package wrapping terminal
#       keyboard input, scrolled video display output, tsLogger and
#       tsException services.

try:

    import tsLibCLI

except ImportError, importCode:

    print('%s: ImportError (tsLibCLI); ' % __title__ + \
          'importCode=%s' % str(importCode))

try:

    import tsExceptions as tse
    import tsLogger as Logger
    import tsOperatorSettingsParser

except ImportError, importCode:

    print('%s: ImportError (tsLibGUI); ' % __title__ + \
          'importCode=%s' % str(importCode))

try:

    import tsLibGUI

except ImportError, importCode:

    print('%s: ImportError (tsLibGUI); ' % __title__ + \
          'importCode=%s' % str(importCode))

try:

    import tsWx as wx

    from tsWxMultiFrameEnv import MultiFrameEnv

    MultiFrameEnvWrapperEnable = True

except ImportError, importCode:

    print('%s: ImportError (tsLibGUI); ' % __title__ + \
          'importCode=%s' % str(importCode))
```

```

MultiFrameEnvWrapperEnable = False

#-----

DEBUG = True # Set True to log run time parameters and exit.
VERBOSE = False

DebugSimulatedKeyErrorTrap = True

PySimpleAppMode = False # Set True only for Redirection, TaskBar and
                        # Single The Frame with no Dialogs

debugExitEnabled = False # True
frameSizingEnabled = True
redirectEnabled = True
runTimeTitleEnabled = True
splashScreenSeconds = 0
tracebackEnabled = False

#-----

__help__ = '''
This program demonstrates an environment for a multi-frame
and multi-dialog Graphical-style User Interface.
'''
#-----

lastWindowId = None

def nextWindowId():
    global lastWindowId
    if lastWindowId is None:
        lastWindowId = 1 # wx.ID_ANY
    else:
        lastWindowId += 1
    return (lastWindowId)

#-----

class _Prototype(wx.Frame):
    '''
    Class to establish the frame that contains the application specific
    graphical components.
    '''
    def tsGetTheId(self):
        '''
        Return the ID associated with this class instance.
        '''
        return id(self)

#-----

def __init__(self,
              parent,
              id=nextWindowId(),

```

```
        title=wx.EmptyString,
        pos=wx.DefaultPosition,
        size=wx.DefaultSize,
        style=wx.DEFAULT_FRAME_STYLE,
        name=wx.FrameNameStr):
'''
Init the Frame
Show the Frame
'''

if frameSizingEnabled:

    wx.Frame.__init__(self,
                      parent,
                      id=nextWindowId(),
                      title=title,
                      pos=pos,
                      size=((280 * 3) / 2, (200 * 3) / 2),
                      style=style,
                      name=name)

    # TBD - This should NOT change Frame color.
    # It should only change Panel color.
##    self.ForegroundColour = wx.COLOR_YELLOW
##    self.BackgroundColour = wx.COLOR_MAGENTA

else:

    # Establish character and pixel position of this canvas
    # Set at top left row and column of area to be centered, by
    # default, in the user terminal screen.
    begin_y = -1
    begin_x = -1
    thePos = wx.tsGetPixelValues(begin_x, begin_y)

    # Establish character and pixel size of this canvas
    # for the wxPython application "tsWxGUI_test1.py".
    #
    # VGA Display (640 x 480 pixels) with Courier (8 x 12 pixels)
    # monospaced font characters contains (80 Cols x 40 Rows).

    # Set typical console area
    max_x = -1 # 80
    max_y = -1 # 30
    theSize = wx.tsGetPixelValues(max_x, max_y)

    wx.Frame.__init__(
        self,
        parent,
        id,
        name='Frame',
        title=title,
        pos=thePos,
        size=theSize,
        style=wx.DEFAULT_FRAME_STYLE)
```

```

# Cannot log before GUI started via Frame.
self.logger.notice(
    'Begin %s (0x%X).' % ('Prototype', self.tsGetTheId()))
print(
    'Begin %s (0x%X).' % ('Prototype', self.tsGetTheId()))

#-----
# Begin Prototype
#-----

self.Centre()
self.Show()

theFrame = self

theRect = theFrame.Rect
theClientArea = theFrame.ClientArea
print('Frame: Rect=%s; ClientArea=%s' % (str(theRect),
                                         str(theClientArea)))

theText = "Hello, World! "
print("theText=%s" % theText)

# Allow additional width for cursor
theTextWidth = (len(theText) + 1) * wx.pixelWidthPerCharacter
theTextHeight = 1 * wx.pixelHeightPerCharacter

theTextOffset = wx.Point(
    (theClientArea.x + (
        (theClientArea.width - theTextWidth) // 2)),
    (theClientArea.y + (
        (theClientArea.height - theTextHeight) // 2)))
print("theTextOffset=%s" % str(theTextOffset))

theTextSize = wx.Size(theTextWidth, theTextHeight)
print("theTextSize=%s" % str(theTextSize))

text = wx.StaticText(
    theFrame,
    pos=theTextOffset,
    size=theTextSize,
    label=theText)

self.Show(show=True)

#-----
# End Prototype
#-----

# TBD - This should NOT change Frame color.
# It should only change Panel color.
##     self.ForegroundColour = wx.COLOR_YELLOW
##     self.BackgroundColour = wx.COLOR_MAGENTA

self.logger.debug(
    'End %s (0x%X).' % ('Prototype', self.tsGetTheId()))

```

```
#-----

def OnMove(self, event):
    pos = event.GetPosition()
    self.posCtrl.SetValues('%s, %s' % (pos.x, pos.y))
    print('Prototype OnMove: value=%s' % str(pos))

#-----

def OnQuit(self, event):
    '''
    Close the Frame
    '''
    print('Prototype OnQuit: value=%d' % -1)
    self.Close()

#-----

def OnHelp(self, event):
    '''
    Show the help dialog.
    '''
    dlg = wx.MessageDialog(
        self,
        __help__,
        "%s Help" % __title__,
        wx.OK | wx.ICON_INFORMATION)

    dlg.ShowModal()
    dlg.Destroy()
    print('Prototype OnHelp: value=%d' % -1)

#-----

def OnAbout(self, event):
    '''
    Show the about dialog.
    '''
    dlg = wx.MessageDialog(
        self,
        __header__,
        'About %s' % __title__,
        wx.OK | wx.ICON_INFORMATION)

    dlg.ShowModal()
    dlg.Destroy()
    print('Prototype OnAbout: value=%d' % -1)

#-----

if __name__ == '__main__':

    #-----

    def exitTest():
```



```
'''
Simulated Input / Output Exception to induce termination
with an exit code and message.
'''
exceptionName = tse.INPUT_OUTPUT_EXCEPTION
errorName = 'Oops! Invalid Name'

myLogger = Logger.TsLogger(threshold=Logger.DEBUG,
                           name='exitTest')

message = 'Exit Test'

myLogger.debug('***** Exit Test %s / %s *****' % (
    exceptionName, errorName))
raise tse.InputOutputException(errorName, message)

#-----
def mainTest(*args, **kw):
    '''
    Command Line Interface. It gathers and displays operator
    input as keyword-value pair options and positional arguments.
    '''
    myLogger = Logger.TsLogger(threshold=Logger.DEBUG,
                              name='mainTest')
    myParser = tsOperatorSettingsParser.TsOperatorSettingsParser()
    msg = '\n%s\n' % myParser.getRunTimeTitleVersionDate()
    print(msg)
    myLogger.debug(msg)

    rawArgsOptions = sys.argv[1:]
    msg = '\t rawArgsOptions=%s' % str(rawArgsOptions)
    print(msg)
    myLogger.debug(msg)
    maxArgs = len(rawArgsOptions)

    (args, options) = myParser.parseCommandLineDispatch()
    msg = 'type(args=%s)=%s' % (str(args), type(args))
    print(msg)
    myLogger.debug(msg)

    msg = 'type(options=%s)=%s' % (str(options), type(options))
    print(msg)
    myLogger.debug(msg)

    if True or DEBUG:
        label = myParser.getRunTimeTitle()

        fmt1 = '%s.mainTest (parameter list): ' % label
        fmt2 = 'args=%s;\n\t\t\t kw=%s' % (str(args), str(kw))
        msg = '\n\t\t\t %s\n\t\t\t %s' % (fmt1, fmt2)
        print(msg)
        myLogger.debug(msg)

        fmt1 = '%s.mainTest (command line argv): ' % label
        fmt2 = 'args=%s' % str(args)
```

(

```

        args = myApp.args
        options = myApp.options

    msg = '\n\n\tResults:\n\t\tArgs=%s;\n\t\tOptions=%s' % (
        str(args), str(options))
    print(msg)
    myLogger.debug(msg)

#-----

myApp = MultiFrameEnv(

    buildTitle=__title__,
    buildVersion=__version__,
    buildDate=__date__,
    buildAuthors=__authors__,
    buildCopyright=__copyright__,
    buildLicense=__license__,
    buildCredits=__credits__,
    buildTitleVersionDate=mainTitleVersionDate,
    buildHeader=__header__,
    buildPurpose=__doc__,

#
    enableDefaultCommandLineParser=False, # Disable unless True
#

    guiMessageFilename=None,
    guiMessageRedirect=True,
    guiRequired=True,
    guiTopLevelObject=_Prototype,
    guiTopLevelObjectId=wx.ID_ANY,
    guiTopLevelObjectName=wx.FrameNameStr,
    guiTopLevelObjectParent=None,
    guiTopLevelObjectPosition=wx.DefaultPosition,
    guiTopLevelObjectSize=wx.DefaultSize,
    guiTopLevelObjectStatusBar=None, # TBD - Implementation
    guiTopLevelObjectStyle=wx.DEFAULT_FRAME_STYLE,
    guiTopLevelObjectTitle= __title__,

#

    runTimeEntryPoint=EntryPoint)

myApp.Wrapper()

```

Draft

1.7 APPENDIX F - HISTORY OF SYSTEM DEVELOPMENT, OPERATION, AND MAINTENANCE

The "tsWxGTUI_PyVx" Toolkit applies those software engineering criteria and practices that have been employed and perfected by software professionals, including its author(s), over many years.

- *System Development* (on page 171)
- *System Maintenance* (on page 178)
- *System Operation* (on page 181)

1.7.1 System Development

System development for the "tsWxGTUI_PyVx" Toolkit is characterized by the following:

- *Rationale* (on page 172)
- *Requirements* (on page 172)
- *Goals* (on page 172)
- *Non-Goals* (on page 173)
- *Assumptions* (on page 174)
- *Deliverables* (on page 175)
- *Stakeholders* (on page 176)
- *Prerequisites* (on page 177)
- *Dependents* (on page 178)

1.7.1.1 Rationale

The design of the "tsWxGTUI_PyVx" Toolkit synergistically integrates and builds upon various free, open software components. Components have been chosen because they have an extensive track record of field-proven development, enhancement, maintenance and support that is comparable to their Commercial-Off-The-Shelf (COTS) component counterparts:

- Of the various GUI toolkits, the C++ based "wxWidgets" and its "wxPython" binding for Python programmers have become quite popular. They support the creation of GUI applications that will run without change on platforms having pixel-mode displays. It must be noted that the wxPython 2.8.12 release is not tracking the current wxWidgets 2.9.4 release which is evolving a somewhat modified Application Programming Interface.
- The underlying C++ based "wxWidgets" GUI library is predicated on each platform having its own native pixel-mode GUI library. It is not known how difficult it would be to introduce support for platforms without native pixel-mode displays.
- It is presumed that the design and implementation of "nCurses" and "curses" terminal independent character-mode libraries would require re-engineering a replacement for a sub-set of the C++ based "wxWidgets" GUI library so as to work around its inability to manipulate icons, proportional sized fonts and other pixel-type data.
- Emulating the sub-set of the Python-based "wxPython" API suitable for character-mode displays should be an order of magnitude easier. It would avoid the effort to first reverse-engineer the existing "wxWidgets" GUI library followed by the effort to engineer the appropriate changes.

1.7.1.2 Requirements

NOTE: There are no published internal functional and interface specifications for "wxPython" and "wxWidgets". However, there are the published external API and demo code for many of the "wxPython" classes, methods and data objects.

- Provide a POSIX-style Command Line Interface toolkit that is user friendly, maintainable and easily enhanced. It shall provide Key-word options and Positional arguments. It shall provide Logging, Event Handling, Report Utilities and Exit Codes.
- Provide a "wxPython"-style, "nCurses"-based Graphical-style User Interface toolkit that is user friendly, maintainable and easily enhanced.
- It must make efficient use of the available display real estate by avoiding the waste associated with multi-character boarder thickness.

1.7.1.3 Goals

This section summarizes the requirements implicit and explicit in the "tsWxGTUI_PyVx" Toolkit's purpose.

This is a high-level view of functional, interface, design, implementation, operation, quality and reliability requirements that are within the work scope.

- 1 A means for one or more operators to interactively monitor and control local and remote computer equipment via either or both of the following, as appropriate to the application:

- a) Command Line Interface (CLI)
 - b) Graphical User Interface (GUI)
- 2** A means for the application developer to interact with the local and remote computer equipment and operator(s) via:
- a) Popular, cross-platform Application Programming Interfaces (APIs). The Command Line Interface and Graphical User Interface APIs must be free, open source and field-proven. The APIs must also have an ongoing and extensive track record of active use, maintenance and enhancement.
 - b) Libraries of building block modules, tests, tools and utilities
 - c) Designs suitable for installation and use in embedded system which typically have limited, application-specific processing, memory, communication, input/output and file storage resources. Some systems may only have character-mode operator interface hardware suitable for the host computer operating system's command line interface console.
 - d) CLI and GUI interfaces that are attractive and user friendly.
- 3** A means for the developer to re-use an existing desktop, laptop, workstation and embedded system applications on an extensive variety of platforms including cell phones, laptops, desktops, workstations and super computers with 32-bit and 64-bit processors from various manufacturers.
- 4** A means for the developer to troubleshoot design and user issues via date and time stamped operational, diagnostic and exception logs.
- 5** A means for the developer to co-ordinate the automated operation and failure recovery of multiple applications by use of Unix-style exit codes and error messages.
- 6** A means for the operator to use a mouse, trackball or touchpad to select GUI objects when the platform supports such a device.
- 7** A means for the operator to use a keyboard to select GUI objects when the platform does not support a mouse, trackball or touchpad.
- 8** A means for the application and toolkit developer to import application and library building-block components from a nested, multi-level directory file system. This is intended to facilitate development without and before the creation and installation of released site-packages. This eliminates the need for building and installing candidate bug fixes to the site-packages before they can even be tested.

1.7.1.4 Non-Goals

This section summarizes those requirements explicitly beyond the scope of the "tsWxGTUI_PyVx" Toolkit's purpose.

This is a high-level view of functional, interface, design, implementation, operation, quality and reliability requirements that are beyond the work scope.

- 1** A means to emulate each and every capability of the Application Programming Interface (API) of the pixel-mode Graphical User Interface (such as the C++ language based "wxWidgets" and its Python language based "wxPython" wrapper) because the "tsWxGTUI_PyVx" Toolkit is designed for embedded systems which typically have limited, application-specific processing, memory, communication, input/output and file storage resources. Some may only have character-mode hardware suitable for their operating system's command line console.

- 2 A means to develop support for pixel-mode GUI features such as icons, proportional fonts and other bit-mapped images because many embedded system displays operate only in character-mode.
- 3 A means to programmatically re-size the top level application Frame or any other GUI object because:
 - a) The host operating system's command line shell window is opened upon operator login and the size of the login window establishes the initial size of the character-mode display (such as "stdscr", the Python "Curses" or GNU "nCurses" low-level GUI-style standard screen).
 - b) The application that directly use the character-mode display (identified as "stdscr" in Python "Curses" or GNU "nCurses") are efficient enough to respond to operator re-sizing events related to the top-level "stdscr" or any associated GUI -style objects. They can also use the "pad" feature to create virtual windows whose out-of-bounds contents are clipped (not displayed) without triggering an error. Those application are responsible for programmatically re-sizing low-level GUI objects and any dependent ones.
 - c) It is not yet known how to make applications programmatically re-size any or all of the high- and low-level GUI-style objects and any dependent ones, without a complete, time consuming application restart.
- 4 A means to re-compile and re-build the "nCurses" library and associated Python wrapper with the wide character option needed to support Unicode characters and 256-colors because its installation could adversely effect the operation of operating system utilities.
- 5 A means to extend the Python Curses module to include support for all of the currently available "nCurses" methods and functions.
- 6 A means to resolve anomalies in look and feel of ansi, vt100 and xterm-256color terminal emulations that are platform-specific.

1.7.1.5 Assumptions

Assumptions for development and use of the "tsWxGTUI_PyVx" Toolkit:

- 1 In accordance with the license terms and conditions of the open source licenses approved by the Open Source Initiative (see Note "About Open Source Licenses" at the bottom of this page):
 - a) It is permissible and feasible to create a Python-based emulation of the Application Programming Interface (API) of software such as the C++-based "wxWidgets" and the "wxPython" binding (wrapper) used by Python programmers.
 - b) It is also permissible and feasible to create a Python-based emulation of those undocumented components of software by reverse engineering the associated source code (such as the C++-based "wxWidgets").
- 2 The "tsWxGTUI_PyVx" Toolkit will consist of at least two components:
 - a) A Command Line Interface Toolkit ("tsToolkitCLI") component.
 - b) A Graphical-style User Interface Toolkit ("tsToolkitGUI") component.
- 3 The "tsToolkitGUI" component will be dependent on the "tsToolkitCLI" component because:
 - a) "tsToolkitCLI" applications are launched by the System Administrator, Software Engineer or System Operator.
 - b) "tsToolkitGUI" applications are launched and its exception handling and termination is wrapped by the "tsToolkitCLI" applications.

- 4 The "tsToolkitCLI" and "tsToolkitGUI" components will be iteratively developed, debugged, tested and qualified for Python 2.x .
 - a) The "tsToolkitCLI" will be developed, debugged, tested and qualified and updated first.
 - b) The "tsToolkitGUI" will be developed, debugged, tested and qualified and updated next.
- 5 As the iterative development, testing, qualification and upgrades progresses, the Python 2.x version of "tsWxGTU" Toolkit shall be ported to Python 3.x.
- 6 The Python 2.x and Python 3.x versions of the "tsWxGTUI_PyVx" Toolkit will be released separately.
- 7 Each release will contain all associated source files for libraries, tools, tests, utilities and documentation.
- 8 Each Python 2.x or 3.x release version will be used to create CLI-style and GUI-style applications that will run unchanged on the 2.x or 3.x version of the Python Virtual Machine appropriate for the host computer system's processor and operating system.

Excerpted from: <http://opensource.org/licenses>:

"About Open Source Licenses

Open source licenses are licenses that comply with the Open Source Definition — in brief, they allow software to be freely used, modified, and shared. To be approved by the Open Source Initiative (also known as the OSI), a license must go through the Open Source Initiative's license review process."

The following OSI-approved licenses are popular, widely used, or have strong communities:

- Apache License 2.0
- BSD 3-Clause "New" or "Revised" license
- BSD 2-Clause "Simplified" or "FreeBSD" license
- GNU General Public License (GPL)
- GNU Library or "Lesser" General Public License (LGPL)
- MIT license
- Mozilla Public License 2.0
- Python License (Python-2.0) (overall Python license)
- wxWindows Library License (WXwindows)

1.7.1.6 Deliverables

The following table describes the work product(s) to be delivered by the developer:

DELIVERABLE	DESCRIPTION
Software Package(s)	<p>Consists of the appropriate components:</p> <ul style="list-style-type: none"> ▪ Source Modules ▪ Tool Modules ▪ Test Modules ▪ Utility Modules

DELIVERABLE	DESCRIPTION
Internal Documentation	Describes programmer usage: <ul style="list-style-type: none">▪ How to Code▪ How to Build▪ How to Package
User Documentation	Describes operator usage: <ul style="list-style-type: none">▪ How to Install▪ How to Run▪ How to Troubleshoot

1.7.1.7 Stakeholders

The "tsWxGTUI_PyVx" Toolkit project stakeholder include those persons, groups or organizations with an interest in a project:

- 1** Sponsor, Acquirerer and Developer Stakeholders:
 - a) "tsWxGTUI_PyVx" Toolkit developer(s).
 - b) CLI-style and GUI-style Application developers (i.e., "tsWxGTUI_PyVx" Toolkit customers).
- 2** Third-party Software Contributor Stakeholder Candidates:
 - a) The Free Software Foundation ("nCurses" Library and "GNU" Toolkit)
 - b) The Python Software Foundation ("Curses", "Logging" and other Modules)
 - c) The wxWindows Organization ("wxWidgets" and "wxPython" Toolkit Products)
- 3** Third-party Forum Stakeholder Candidates:
 - a) "wxWidgets" Developers Group on Google.
 - b) LinkedIn "Python Community" Group
 - c) LinkedIn "Real Time Embedded Engineering" Groups
 - d) Slashdot
 - e) StackOverflow
- 4** Third-party Software Repository Stakeholder Candidates:
 - a) Python Package Index or "PyPI" is the official third-party software repository for the Python programming language.
 - b) SourceForge
 - c) GitHub

Excerpted From Wikipedia, the free encyclopedia:

Stakeholder may refer to:

- Stakeholder (corporate), an accountant, group, organization, member or system who affects or can be affected by an organization's actions
- Stakeholder, an entity that can be affected by the results of that in which they are said to be stakeholders, i.e., that in which they have a stake.
 - Project stakeholder, a person, group or organization with an interest in a project
 - Stakeholder theory, a theory that identifies and models the groups which are stakeholders of a corporation or project
 - Stakeholder analysis, the process of identifying those affected by a project or event
- Stakeholder (law), a third party who temporarily holds money or property while its owner is still being determined

1.7.1.8 Prerequisites

The prerequisites of the "tsWxGTUI_PyVx" Toolkit include the following:

COMPONENT	REQUIREMENT
Hardware	<ul style="list-style-type: none"> ▪ Computer - Any make and model with enough processing capability for the application. ▪ Display - Minimum character dimensions (34 column by 9 row) ▪ Keyboard - PC-compatible keyboard with standard alpha-numeric, punctuation, control and function keys. ▪ Mouse - PC-compatible mouse, trackball, touchpad or touchscreen with left button, center button or wheel, and right button.
Software	<ul style="list-style-type: none"> ▪ nCurses or Curses terminal independent control library. ▪ Unix-style Command Line Interface Shell (Bash, Cygwin etc.) ▪ X11 Library and Terminal Emulator. ▪ Python 2.x/3.x Virtual Machine and associated Interpreter. ▪ Text File editor suitable for creating and modifying Python source code. Developer uses XEmacs. ▪ wxPython and associated documentation and demo. Developer uses this to verify that application will work unchanged on various platforms. ▪ Python Debugger. Developer uses WingIDE to troubleshoot complex failures.

1.7.1.9 Dependents

The dependents of the "tsWxGTUI_PyVx" Toolkit include the following:

- Documentation Requirements.

1.7.2 System Maintenance

System maintenance for the "tsWxGTUI_PyVx" Toolkit is characterized by the following:

- *Developer Platform Maintenance* (on page 179)
- *Operator Platform Maintenance* (on page 180)

Draft

1.7.2.1 Developer Platform Maintenance

Phase	Developer's Platform	Developer's Guest Operating System Platform
1	<p>Backup or Install Host Updates, when needed and convenient</p> <ul style="list-style-type: none"> Operating System (such as Linux, Mac OS X, Microsoft Windows and Unix) nCurses Library Source Code Revision Control Tool for software developers (such as Bazaar, ClearCase, CVS, Git, Mercurial, RCS and Subversion) Source Code Editor (such as Eclipse, Emacs, Gedit, Vim and Xemacs) Source Code Compare and Merge Tool (such as Deltawalker, Diff, Kdff) Python 2.x Interpreter and Library Python 3.x Interpreter and Library Python Integrated Development Environment Toolkit (such as Idle, Komodo and Wing) Guest Operating System Virtualization Toolkit (such as Parallels and VMware Fusion) Virtualized Guest Operating System(s) (such as Linux, Microsoft Windows and Unix) 	<p>Backup or Install Guest Updates, when needed and convenient</p> <ul style="list-style-type: none"> Operating System (such as Linux, Mac OS X, Microsoft Windows and Unix) nCurses Library Source Code Revision Control Tool for software developers (such as Bazaar, ClearCase, CVS, Git, Mercurial, RCS and Subversion) Source Code Editor (such as Eclipse, Emacs, Gedit, Vim and Xemacs) Source Code Compare and Merge Tool (such as Deltawalker, Diff, Kdff) Python 2.x Interpreter and Library Python 3.x Interpreter and Library Python Integrated Development Environment Toolkit (such as Idle, Komodo and Wing)
2	<p>Backup or Install "tsWxGTUI_PyVx" Toolkit Updates, when needed and convenient</p> <ul style="list-style-type: none"> tsWxGTUI Toolkit Documentation Files tsToolkitCLI (tsLibCLI, tsToolsCLI, tsTestsCLI, tsUtilities) tsToolkitGUI (tsLibGUI, tsToolsGUI, tsTestsGUI) applications developed with "tsWxGTUI_PyVx" Toolkit 	<p>Backup or Install "tsWxGTUI_PyVx" Toolkit Updates, when needed and convenient</p> <ul style="list-style-type: none"> tsWxGTUI Toolkit Documentation Files tsToolkitCLI (tsLibCLI, tsToolsCLI, tsTestsCLI, tsUtilities) tsToolkitGUI (tsLibGUI, tsToolsGUI, tsTestsGUI) applications developed with "tsWxGTUI_PyVx" Toolkit
3	Free-up and Defragment Disk Space on Host Platform, when needed and convenient	Free-up and Defragment Disk Space on Guest Platform, when needed and convenient
4	<p>Before running "python setup.py install" and before debugging modifications to "tsWxGTUI_PyVx" Toolkit source code:</p> <ul style="list-style-type: none"> Must delete "tsWxGTUI_PyVx" Toolkit entries from Python 2.x site-package Must delete "tsWxGTUI_PyVx" Toolkit entries from Python 3.x site-package 	<p>Before running "python setup.py install" and before debugging modifications to "tsWxGTUI_PyVx" Toolkit source code:</p> <ul style="list-style-type: none"> Must delete "tsWxGTUI_PyVx" Toolkit entries from Python 2.x site-package Must delete "tsWxGTUI_PyVx" Toolkit entries from Python 3.x site-package

1.7.2.2 Operator Platform Maintenance

Phase	Operator's Platform	Operator's Guest Operating System Platform
1	Backup or Install Host Updates, when needed and convenient <ul style="list-style-type: none"> Operating System (such as Linux, Mac OS X, Microsoft Windows and Unix) nCurses Library Python 2.x Interpreter and Library Python 3.x Interpreter and Library Python Integrated Development Environment Toolkit (such as Idle, Komodo and Wing) Guest Operating System Virtualization Toolkit (such as Parallels and VMware Fusion) Virtualized Guest Operating System(s) (such as Linux, Microsoft Windows and Unix) 	Backup or Install Guest Updates, when needed and convenient <ul style="list-style-type: none"> Operating System (such as Linux, Mac OS X, Microsoft Windows and Unix) nCurses Library Python 2.x Interpreter and Library Python 3.x Interpreter and Library Python Integrated Development Environment Toolkit (such as Idle, Komodo and Wing)
2	Backup or Install "tsWxGTUI_PyVx" Toolkit Updates, when needed and convenient <ul style="list-style-type: none"> tsWxGTUI Toolkit Documentation Files tsToolkitCLI (tsLibCLI, tsToolsCLI, tsTestsCLI, tsUtilities) tsToolkitGUI (tsLibGUI, tsToolsGUI, tsTestsGUI) applicaions developed with "tsWxGTUI_PyVx" Toolkit 	Backup or Install "tsWxGTUI_PyVx" Toolkit Updates, when needed and convenient <ul style="list-style-type: none"> tsWxGTUI Toolkit Documentation Files tsToolkitCLI (tsLibCLI, tsToolsCLI, tsTestsCLI, tsUtilities) tsToolkitGUI (tsLibGUI, tsToolsGUI, tsTestsGUI) applicaions developed with "tsWxGTUI_PyVx" Toolkit
3	Free-up and Defragment Disk Space on Host Platform, when needed and convenient	Free-up and Defragment Disk Space on Guest Platform, when needed and convenient
4	Before running "python setup.py install": <ul style="list-style-type: none"> Must delete "tsWxGTUI_PyVx" Toolkit entries from Python 2.x site-package Must delete "tsWxGTUI_PyVx" Toolkit entries from Python 3.x site-package 	Before running "python setup.py install": <ul style="list-style-type: none"> Must delete "tsWxGTUI_PyVx" Toolkit entries from Python 2.x site-package Must delete "tsWxGTUI_PyVx" Toolkit entries from Python 3.x site-package

1.7.3 System Operation

Isolated (Stand Alone) Mode	Networked (Stand Among) Mode
<ul style="list-style-type: none">▪ login to local platform host▪ launch local shell▪ re-position and re-size shell window if and as appropriate	<ul style="list-style-type: none">▪ login to local platform host▪ launch local shell▪ re-position and re-size shell window if and as appropriate
<ul style="list-style-type: none">▪ launch local application▪ terminate local application	<ul style="list-style-type: none">▪ launch local application▪ terminate local application
	<ul style="list-style-type: none">▪ login to remote platform host▪ launch remote application▪ terminate remote application▪ logout of remote platform host
<ul style="list-style-type: none">▪ terminate local shell▪ logout of local platform host	<ul style="list-style-type: none">▪ terminate local shell▪ logout of local platform host

Draft

2 SYSTEM SPECIFICATION

2.1.1.1 In This Chapter

APPENDIX A - REQUIRED STATES AND MODES	185
APPENDIX B - SYSTEM CAPABILITY REQUIREMENTS	195
APPENDIX C - SYSTEM EXTERNAL INTERFACE REQUIREMENTS	251
APPENDIX D - SYSTEM INTERNAL INTERFACE REQUIREMENTS	277
APPENDIX E - SYSTEM INTERNAL DATA REQUIREMENTS	281
APPENDIX F - ADAPTATION REQUIREMENTS.....	283
APPENDIX G - SAFETY REQUIREMENTS	285
APPENDIX H - SECURITY AND PRIVACY REQUIREMENTS	287
APPENDIX I - SYSTEM ENVIRONMENT REQUIREMENTS	289
APPENDIX J - COMPUTER RESOURCE REQUIREMENTS	291
APPENDIX K - SYSTEM QUALITY FACTORS	295
APPENDIX L - DESIGN AND CONSTRUCTION CONSTRAINTS	297
APPENDIX M - PERSONNEL-RELATED REQUIREMENTS	299
APPENDIX N - TRAINING-RELATED REQUIREMENTS	301
APPENDIX O - LOGISTICS-RELATED REQUIREMENTS	303
APPENDIX P - OTHER REQUIREMENTS	305
APPENDIX Q - PACKAGING REQUIREMENTS	307
APPENDIX R - PRECEDENCE AND CRITICALITY OF REQUIREMENTS	309

Draft

2.2 APPENDIX A - REQUIRED STATES AND MODES

If the system is required to operate in more than one state or mode having requirements distinct from other states or modes, this paragraph shall identify and define each state and mode. Examples of states and modes include: idle, ready, active, post-use analysis, training, degraded, emergency, backup, wartime, peacetime. The distinction between states and modes is arbitrary. A system may be described in terms of states only, modes only, states within modes, modes within states, or any other scheme that is useful. If no states or modes are required, this paragraph shall so state, without the need to create artificial distinctions. If states and/or modes are required, each requirement or group of requirements in this specification shall be correlated to the states and modes. The correlation may be indicated by a table or other method in this paragraph, in an appendix referenced from this paragraph, or by annotation of the requirements in the paragraphs where they appear.

The system hardware and software shall provide the means for developing, documenting, enhancing, maintaining, supporting, troubleshooting and using the "tsWxGTUI_PyVx" Toolkit and those application programs created with it. The system is required to operate in the following states and modes:

1 *Required States* (on page 185)

2 *Required Modes* (on page 187)

2.2.1 Required States

2.2.1.1 Pre-Startup State

- 1** In the event that the platform has not been prepared for use by the System Operator, the System Administrator shall perform the following actions:
 - a) Install or upgrade and validate the operability of the Linux, Mac OS X, Microsoft Windows, Unix or other operating system appropriate to the platform.
 - b) Install or upgrade and validate the operability the Python interpreter and virtual machine appropriate for the platform operating system.
 - c) Install or upgrade and validate the operability the "tsWxGTUI_PyVx" Toolkit appropriate for the platform operating system.
- 2** In the event that the platform has not been prepared for use, the System Operator shall perform the following actions:
 - a) Login to the the local or remote session.
 - b) Establish the location and size of the local or remote terminal screen

2.2.1.2 Startup State

- 1** Upon the operator's application of electrical power, or activation of the computer "reset", the computer hardware shall be initialized or re-initialized to clear any previous error and restore normal operating conditions.
- 2** The computer shall load and execute its built-in Power-On-Self Test (POST) to verify its readiness for normal operation.

Excerpt From Wikipedia, the free encyclopedia

- a) POST includes routines to set an initial value for internal and output signals and to execute internal tests, as determined by the device manufacturer. These initial conditions are also referred to as the device's state. They may be stored in firmware or included as hardware, either as part of the design itself, or they may be part of semiconductor substrate either by virtue of being part of a device mask, or after being burned into a device such as a programmable logic array (PLA).
 - b) Test results may either be displayed on a panel that is part of the device, or output via bus to an external device. They may also be stored internally, or may exist only until the next power-down. In some cases, such as in aircraft and automobiles, only the fact that a failure occurred may be displayed (either visibly or to an on-board computer) but may also upload detail about the failure(s) when a diagnostic tool is connected.
 - c) POST protects the bootstrapped code from being interrupted by faulty hardware. Diagnostic information provided by a device, for example when connected to an engine analyzer, depends on the proper function of the device's internal components. In these cases, if the device is not capable of providing accurate information—which ensures that the device is safe to run—subsequent code (such as bootstrapping code) may **not** be permitted to run.
- 3** The computer shall load and execute its Operating System software (such as Linux, Mac OS X, Microsoft Windows or Unix).

2.2.1.3 Operating State

- 1** The computer shall load and execute the applicable user interface software:
 - a) Desktop environment and Command Line Interface (such as the Unix-style bash shell or the Microsoft Command Prompt shell accessory).
 - b) Desktop environment and Graphical User Interface (such as the Unix-style GNOME and KDE or Microsoft Windows).
- 2** The computer shall interact with the operator to load, execute and terminate operator designated system and application software.

2.2.1.4 Shutdown State

- 1** Upon the occurrence of a non-recoverable malfunction or operator shutdown command, the computer hardware shall display a diagnostic or shutdown message.
- 2** The computer shall terminate any running application software.
- 3** The computer shall terminate any running operating system software.
- 4** The computer shall halt pending the next power-on.

2.2.2 Required Modes

2.2.2.1 Video Adapter Modes

Excerpt From: http://www.webopedia.com/TERM/G/graphics_mode.html

Many video adapters support several different modes of resolution, all of which are divided into two general categories: character mode and graphics mode.

Of the two modes, graphics mode is the more sophisticated. Programs that run in graphics mode can display an unlimited variety of shapes and fonts, whereas programs running in character mode are severely limited. Programs that run entirely in graphics mode are called graphics-based programs.

In character mode, the display screen is treated as an array of blocks, each of which can hold one ASCII character. In graphics mode, the display screen is treated as an array of pixels. Characters and other shapes are formed by turning on combinations of pixels.

Draft

2.2.2.1.1 **Text Mode**

Excerpt From Wikipedia, the free encyclopedia

Text mode is a kind of computer display mode in which the content of the screen is internally represented in terms of textual characters rather than individual pixels. Typically, the screen consists of a uniform rectangular grid of character cells, each of which contains one of the characters of a character set. Text mode is contrasted to all points addressable (APA) mode or other kinds of computer graphics modes. The main purpose of the text mode is to implement a text user interface, but the latter concept is broader. mode video rendering came to prominence in the early 1970s, when video-oriented text terminals started to replace teleprinters in the interactive use of computers. mode applications communicate with the user with command-line interfaces and text user interfaces. Many character sets used in text mode applications also contain a limited set of predefined semi-graphical characters usable for drawing boxes, and other rudimentary graphics which can be used to highlight the content or to simulate widget or control interface objects found in GUI programs. A typical example is the IBM code page 437 character set. advantages of text modes as compared to graphics modes include lower memory consumption and faster screen manipulation. Also, text mode applications have relatively low bandwidth requirements in remote terminal use. An obvious disadvantage of text mode is the restricted screen content, which makes text mode impractical for many types of applications. important characteristic of text mode programs is that they assume monospace fonts, where every character has the same width on screen, which allows to easily maintain the vertical alignment when displaying semi-graphical characters. This was influenced by the use of early teletype-like and daisy-like fixed-pitch type printers, but also by applications designed for punched cards. This way, the output seen on the screen could be sent directly to the printer maintaining exactly the same format, somewhat in a fashion that is now called WYSIWYG (What You See Is What You Get). on the environment, the screen buffer can be directly addressable. Programs that display output on remote video terminals must issue special control sequences to manipulate the screen buffer. The most popular standards for such control sequences are ANSI and VT100. accessing the screen buffer through control sequences may lose synchronization with the actual display, so that many text mode programs have a redisplay everything command, often associated with the Ctrl-L key combination.

Norton Utilities 6.01, an example of advanced TUI which redefines the character set to show tiny graphical widget, icons and an arrow pointer in text mode.

The border between text mode and graphical programs can sometimes be fuzzy, especially on the PC's VGA hardware, because many later text mode programs tried to push the model to the extreme by playing with the video controller. For example, they redefined the character set in order to create custom semi-graphical characters, or even created the appearance of a graphical mouse by redefining the appearance of the characters over which the mouse was shown at a given time. mode rendering with user-defined characters has also been useful for 2D computer and video games because the game screen can be manipulated much faster than with pixel-oriented rendering. modern programs with a graphical interface simulate the display style of text mode programs, notably when it is important to preserve the vertical alignment of text, e.g., during computer programming. There exist also software components to emulate text mode, such as terminal emulators or command line consoles. In Microsoft Windows, the Win32 console usually opens in emulated, graphical window mode but it can be switched to full screen, true text mode and vice versa by pressing the Alt and Enter keys together.

2.2.2.1.2 Pixel Mode

Excerpt From: http://en.wikipedia.org/wiki/Fixed_pixel_display

Fixed pixel displays are display technologies such as LCD and plasma that use an unfluctuating matrix of pixels with a set number of pixels in each row and column. With such displays, adjusting (scaling) to different aspect ratios because of different input signals requires complex processing.

In contrast, the CRTs electronics architecture "paints" the screen with the required number of pixels horizontally and vertically. CRTs can be designed to more easily accommodate a wide range of inputs (VGA, XVGa, NTSC, HDTV, etc.).

Draft

2.2.2.2 User Interface Modes

2.2.2.2.1 Command Line Mode

Excerpt From: http://en.wikipedia.org/wiki/Command-line_interface

A command-line interface (CLI) is an interface or dialog between the user and a program, or between two programs, where a line of text (a command line) is passed between the two. Many graphical interfaces, such as the OS/2 Presentation Manager and the various Windows shell use command-lines to call helper programs to open documents and programs. The commands are stored in the graphical shell or in files like the registry or the OS/2 os2user.ini file.

Command-line interfaces replaced point-and-load menus (Basic, MS-DOS Executive in Windows 2.x) and externally loaded programs (such as punched cards and game cartridges), when computers became sufficiently powerful to handle parsing of user input, and keep several programs in a runnable condition. This happened at the conversion to 16-bit computers, although text-menus continued to be supplied with computers well after this time.

The command-line interface derives from the teletype or teletypewriter machines. These were originally connected to remote machines of the same kind, which allowed an early text conversations between remote users. The conversations appeared on the output paper as the messages were sent by the user (by pressing the 'enter' key). One could feed pre-loaded messages through an attached paper tape reader.

Teletype machines were connected to the early computers. Users then had teletype conversations with computers. The commands were sent by the user and the computer, the conversation recorded on the paper. Eventually, the paper was replaced by a text screen with scrolling lines, and eventually one where the computer could address the position on the screen.

The invention of the Standard Input/Output interface allowed command line output to be redirected to input for another program to look at. Various console hacks allowed one to store and edit commands in software, as well as use 'character-based graphics', which presented the user with a dialog, rather than a line interface.

The alternative to the command at the line is a graphical dialog. Command-line options becomes buttons and switches presented on the dialog, while sending the dialog or command does much the same thing. DBase allowed one to build command lines from dialogs and further editing the command on its line. Take Command allows one to launch a dialog in stead of options on the command line.

The Command Line Interface continues to co-evolve with GUIs like those provided by Microsoft Windows, Mac OS and the X Window System. Programs that make use of external helper programs, often make use of command lines embedded in the GUI interface or configuration. In some applications, such as MATLAB, AutoCAD or EAGLE, a CLI is integrated with the GUI, with some benefits of both. Commands exist to open directory windows, read and write to the clipboard and do other graphical things directly from the batch files or the command line.

Usage

A CLI is used whenever a large vocabulary of commands or queries, coupled with a wide (or arbitrary) range of options, can be entered more rapidly as text than with a pure GUI. This is typically the case with operating system command shells. CLIs are also used by systems with insufficient resources to support a graphical user interface. Some computer language systems (such as Python, Forth, LISP and many dialects of BASIC) provide an interactive command-line mode to allow for experimentation.

CLIs are often used by programmers and system administrators, in engineering and scientific environments, and by technically advanced personal computer users. CLIs are also popular among people with visual disability, since the commands and responses can be displayed using Refreshable Braille displays.

A program that implements such a text interface is often called a command-line interpreter, command processor or shell, whereby the term shell, often used to describe a command-line interpreter, can be in principle any program that constitutes the user-interface, including fully graphically oriented ones—for example, the default Windows GUI is created by a shell program named EXPLORER.EXE, as defined in the SHELL=EXPLORER.EXE line in the WIN.INI configuration file.

Examples of command-line interpreters include the various Unix shells (sh, ksh, csh, tcsh, bash, etc.), the historical CP/M CCP, and MS-DOS/IBM-DOS/DR-DOS's COMMAND.COM, as well as the OS/2 and the Windows CMD.EXE programs, the latter groups being based heavily on DEC's RSX and RSTS CLIs. Under most operating systems, it is possible to replace the default shell program by more specialized or powerful alternatives; some widespread examples include 4DOS for DOS, 4OS2 for OS/2, and 4NT or Take Command for Windows.

There are command-line interpreters for editing text files like ED and EDLIN, DEBUG, for disk management DISKPART, DFSEE, calculators (PC-DOS ACALC), all of which present a usable command prompt.

In November 2006, Microsoft released version 1.0 of Windows PowerShell (formerly codenamed Monad), which combined features of traditional Unix shells with their object-oriented .NET Framework. MinGW and Cygwin are open-source packages for Windows that offer a Unix-like CLI. Microsoft provides MKS Inc.'s ksh implementation MKS Korn shell for Windows through their Services for UNIX add-on.

The latest versions of the Macintosh operating system are based on a variation of Unix called Darwin. On these computers, users can access a Unix-like command-line interface called Terminal found in the Applications Utilities folder. (This terminal uses bash by default.)

Some applications provide both a CLI and a GUI. In some cases, the GUI is a wrapper around a CLI application; other times, there is a CLI to control a GUI application. The engineering/scientific numerical computation package MATLAB provides no GUI for some calculations, but the CLI can handle any calculation. The three-dimensional-modelling program Rhinoceros 3D provides a CLI as well as a distinct scripting language. In some computing environments, such as the Oberon or Smalltalk user interface, most of the text which appears on the screen may be used for giving commands.

2.2.2.2.2 Graphical Mode

Excerpt From: http://www.webopedia.com/TERM/G/Graphical_User_Interface_GUI.html

Abbreviated GUI (pronounced GOO-ee). A program interface that takes advantage of the computer's graphics capabilities to make the program easier to use. Well-designed graphical user interfaces can free the user from learning complex command languages. On the other hand, many users find that they work more effectively with a command-driven interface, especially if they already know the command language. user interfaces, such as Microsoft Windows and the one used by the Apple Macintosh, feature the following basic components:

- **pointer:** A symbol that appears on the display screen and that you move to select objects and commands. Usually, the pointer appears as a small angled arrow. Text -processing applications, however, use an I-beam pointer that is shaped like a capital I.
- **pointing device:** A device, such as a mouse or trackball, that enables you to select objects on the display screen.
- **icons:** Small pictures that represent commands, files, or windows. By moving the pointer to the icon and pressing a mouse button, you can execute a command or convert the icon into a window. You can also move the icons around the display screen as if they were real objects on your desk.
- **desktop:** The area on the display screen where icons are grouped is often referred to as the desktop because the icons are intended to represent real objects on a real desktop.
- **windows:** You can divide the screen into different areas. In each window, you can run a different program or display a different file. You can move windows around the display screen, and change their shape and size at will.
- **menus:** Most graphical user interfaces let you execute commands by selecting a choice from a menu.

The first graphical user interface was designed by Xerox Corporation's Palo Alto Research Center in the 1970s, but it was not until the 1980s and the emergence of the Apple Macintosh that graphical user interfaces became popular. One reason for their slow acceptance was the fact that they require considerable CPU power and a high-quality monitor, which until recently were prohibitively expensive. addition to their visual components, graphical user interfaces also make it easier to move data from one application to another. A true GUI includes standard formats for representing text and graphics. Because the formats are well-defined, different programs that run under a common GUI can share data. This makes it possible, for example, to copy a graph created by a spreadsheet program into a document created by a word processor. DOS programs include some features of GUIs, such as menus, but are not graphics based. Such interfaces are sometimes called graphical character-based user interfaces to distinguish them from true GUIs.

2.2.2.2.1 wxPython

Of the various Graphical User Interface toolkits, "wxPython" has become quite popular. Here is an excerpt from its on-line *wxPython Introduction* (<http://www.wxpython.org/what.php>;) documentation:

"wxPython is a GUI toolkit for the Python programming language. It allows Python programmers to create programs with a robust, highly functional graphical user interface, simply and easily. It is implemented as a Python extension module (native code) that wraps the popular wxWindows cross platform GUI library, which is written in C++.

Like Python and wxWindows, wxPython is Open Source, which means that it is free for anyone to use and the source code is available for anyone to look at and modify. Or anyone can contribute fixes or enhancements to the project.

wxPython is a cross platform toolkit. This means that the same program will run on multiple platforms without modification. Currently supported platforms are 32-bit Microsoft Windows, most Unix or unix-like systems, and Macintosh OS X. Since the language is Python, wxPython programs are simple, easy to write and to understand."

Draft

Draft

2.3 APPENDIX B - SYSTEM CAPABILITY REQUIREMENTS

This paragraph shall be divided into subparagraphs to itemize the requirements associated with each capability of the system. A "capability" is defined as a group of related requirements. The word "capability" may be replaced with "function," "subject," "object," or other term useful for presenting the requirements.

The system hardware and software (see *APPENDIX J - COMPUTER RESOURCE REQUIREMENTS* (on page 291) for configuraton usability considerations) shall provide the means for developing, documenting, enhancing, maintaining, supporting, troubleshooting and using the "tsWxGTUI_PyVx" Toolkit, and those application programs created with it, with different types of computer hardware and/or with different software packages:

- 1 *Hardware Capabilities* (on page 195)
- 2 *Software Capabilities* (on page 205)
- 3 *Platform Interface Capability* (on page 242)
- 4 *Application Programming Interface Capability* (on page 242)
- 5 *Operator Interface Capability* (on page 248)

2.3.1 Hardware Capabilities

The system hardware shall provide the following cross-platform capabilities:

- 1 *Central Processing Unit Capability* (on page 196)
- 2 *Random Access Memory Capability* (on page 196)
- 3 *Non-Volatile Storage Capability* (on page 196)
- 4 *Network Interface Device Capability* (on page 196)
- 5 *Keyboard Device Capability* (on page 196)
- 6 *Pointing Device Capability* (on page 197)
- 7 *Display Device Capability* (on page 197)
- 8 *Printer Device Capability* (on page 205)

2.3.1.1 Central Processing Unit Capability

The system hardware shall provide a required electronic device that loads and executes machine-readable instructions. It shall performs arithmetic and logic. It shall load, modify and store machine-readable data.

The processor may be any of the popular 32-/64-bit single or multi-core devices from companies such as:

- 1** Advanced Micro Devices
- 2** Advanced RISC Machines, Ltd.
- 3** Freescale Semiconductor, Inc.
- 4** International Business Machines
- 5** Intel
- 6** Mototola
- 7** Oracle/Sun Micro Systems
- 8** Silicon Graphics Inc.

2.3.1.2 Random Access Memory Capability

The system hardware shall provide a required electronic device that temporarily receives, stores and returns machine-readable machine instructions and data. The amount of memory is platform and application specific.

2.3.1.3 Non-Volatile Storage Capability

The system hardware shall provide a required internal and optional external electronic device (Flash Memory) or electro-mechanical device (Hard Drive) whose non-volatile memory receives, stores and can then return source and machine-readable data after electrical power interruptions. The optional external device may be local or remote. When remote, the system must include a Network Interface Device.

2.3.1.4 Network Interface Device Capability

The system hardware shall optionally provide an electronic device (Modem) that inputs and outputs data over an optional wired or wireless telecommunications circuit.

2.3.1.5 Keyboard Device Capability

The system hardware shall provide a hand-operated electronic or electro-mechanical device to input alpha-numeric and control data via push buttons.

2.3.1.6 Pointing Device Capability

The system hardware shall optionally provide a hand-operated electronic or electro-mechanical Mouse, Trackball, Touchpad or Touchscreen device that controls the coordinates of a cursor on the computer screen as you move the positioning control (hand, finger or stylus) around.

2.3.1.7 Display Device Capability

The system hardware shall provide a required electronic device that outputs alpha-numeric, graphic and control data to the computer screen:

- 1 Set of terminal-independent video display features that facilitate the construction of character-mode user interface applications:
 - a) DISPLAY_BLINK (display and then hide the designated text at periodic intervals)
 - b) DISPLAY_BOLD (display a thicker or extra-bright version of the designated text)
 - c) DISPLAY_DIM (display the designated text with half brightness)
 - d) DISPLAY_NORMAL (display the designated text with normal brightness)
 - e) DISPLAY_REVERSE (display the designated text with transposed foreground and background brightness and color)
 - f) DISPLAY_STANDOUT (display the designated text with the best highlighting mode of the terminal)
 - g) DISPLAY_UNDERLINE (display the designated text with a line under a word or phrase, especially for emphasis)
- 2 Non-color terminak or terminal emulator:
 - a) vt100 (black with associated single shade of white/green/orange color phosphor)
 - b) vt220 (black with associated single shade of white/green/orange color phosphor)
- 3 Multi-color terminal or terminal emulator:
 - a) xterm (8-color palette with 64-color pairs)
 - 0: COLOR_BLACK,
 - 1: COLOR_RED,
 - 2: COLOR_GREEN,
 - 3: COLOR_YELLOW,
 - 4: COLOR_BLUE,
 - 5: COLOR_MAGENTA,
 - 6: COLOR_CYAN,
 - 7: COLOR_WHITE
 - }

b) xterm-color (8-color palette with 64-color pairs)

```
{  
  0: COLOR_BLACK,  
  1: COLOR_RED,  
  2: COLOR_GREEN,  
  3: COLOR_YELLOW,  
  4: COLOR_BLUE,  
  5: COLOR_MAGENTA,  
  6: COLOR_CYAN,  
  7: COLOR_WHITE  
}
```

c) xterm-16color (16-color palette with 256-color pairs)

```
{  
  0: COLOR_BLACK,  
  1: COLOR_RED,  
  2: COLOR_GREEN,  
  3: COLOR_YELLOW,  
  4: COLOR_BLUE,  
  5: COLOR_MAGENTA,  
  6: COLOR_CYAN,  
  7: COLOR_WHITE,  
  8: COLOR_GRAY,  
  9: COLOR_MAROON,  
 10: COLOR_LIME_GREEN,  
 11: COLOR_OLIVE,  
 12: COLOR_NAVY,  
 13: COLOR_PURPLE,  
 14: COLOR_TEAL,  
 15: COLOR_SILVER  
}
```

d) xterm-88color (71-color palette with 5041-color pairs)

```
{  
  0: COLOR_BLACK,  
  1: COLOR_RED,
```


2: COLOR_GREEN,
3: COLOR_YELLOW,
4: COLOR_BLUE,
5: COLOR_MAGENTA,
6: COLOR_CYAN,
7: COLOR_WHITE,
8: COLOR_GRAY,
9: COLOR_MAROON,
10: COLOR_LIME_GREEN,
11: COLOR_OLIVE,
12: COLOR_NAVY,
13: COLOR_PURPLE,
14: COLOR_TEAL,
15: COLOR_SILVER,
16: COLOR_AQUAMARINE,
17: COLOR_BLUE_VIOLET,
18: COLOR_BROWN,
19: COLOR_CADET_BLUE,
20: COLOR_CORAL,
21: COLOR_CORNFLOWER_BLUE,
22: COLOR_DARK_GRAY,
23: COLOR_DARK_GREEN,
24: COLOR_DARK_OLIVE_GREEN,
25: COLOR_DARK_ORCHID,
26: COLOR_DARK_SLATE_BLUE,
27: COLOR_DARK_SLATE_GRAY,
28: COLOR_DARK_TURQUOISE,
29: COLOR_DIM_GRAY,
30: COLOR_FIREBRICK,
31: COLOR_FOREST_GREEN,
32: COLOR_GOLD,
33: COLOR_GOLDENROD,
34: COLOR_GREEN_YELLOW,
35: COLOR_INDIAN_RED,

36: COLOR_KHAKI,
37: COLOR_LIGHT_BLUE,
38: COLOR_LIGHT_GRAY,
39: COLOR_LIGHT_STEEL_BLUE,
40: COLOR_MEDIUM_AQUAMARINE,
41: COLOR_MEDIUM_BLUE,
42: COLOR_MEDIUM_FOREST_GREEN,
43: COLOR_MEDIUM_GOLDENROD,
44: COLOR_MEDIUM_ORCHID,
45: COLOR_MEDIUM_SEA_GREEN,
46: COLOR_MEDIUM_SLATE_BLUE,
47: COLOR_MEDIUM_SPRING_GREEN,
48: COLOR_MEDIUM_TURQUOISE,
49: COLOR_MEDIUM_VIOLET_RED,
50: COLOR_MIDNIGHT_BLUE,
51: COLOR_ORANGE,
52: COLOR_ORANGE_RED,
53: COLOR_ORCHID,
54: COLOR_PALE_GREEN,
55: COLOR_PINK,
56: COLOR_PLUM,
57: COLOR_SALMON,
58: COLOR_SEA_GREEN,
59: COLOR_SIENNA,
60: COLOR_SKY_BLUE,
61: COLOR_SLATE_BLUE,
62: COLOR_SPRING_GREEN,
63: COLOR_STEEL_BLUE,
64: COLOR_TAN,
65: COLOR_THISTLE,
66: COLOR_TURQUOISE,
67: COLOR_VIOLET,
68: COLOR_VIOLET_RED,
69: COLOR_WHEAT,

70: COLOR_YELLOW_GREEN

}

e) xterm-256color (140-color palette with 19600-color pairs)

{

0: COLOR_BLACK,

1: COLOR_RED,

2: COLOR_GREEN,

3: COLOR_YELLOW,

4: COLOR_BLUE,

5: COLOR_MAGENTA,

6: COLOR_CYAN,

7: COLOR_WHITE,

8: COLOR_GRAY,

9: COLOR_MAROON,

10: COLOR_LIME_GREEN,

11: COLOR_OLIVE,

12: COLOR_NAVY,

13: COLOR_PURPLE,

14: COLOR_TEAL,

15: COLOR_SILVER,

16: COLOR_AQUAMARINE,

17: COLOR_BLUE_VIOLET,

18: COLOR_BROWN,

19: COLOR_CADET_BLUE,

20: COLOR_CORAL,

21: COLOR_CORNFLOWER_BLUE,

22: COLOR_DARK_GRAY,

23: COLOR_DARK_GREEN,

24: COLOR_DARK_OLIVE_GREEN,

25: COLOR_DARK_ORCHID,

26: COLOR_DARK_SLATE_BLUE,

27: COLOR_DARK_SLATE_GRAY,

28: COLOR_DARK_TURQUOISE,

29: COLOR_DIM_GRAY,

30: COLOR_FIREBRICK,
31: COLOR_FOREST_GREEN,
32: COLOR_GOLD,
33: COLOR_GOLDENROD,
34: COLOR_GREEN_YELLOW,
35: COLOR_INDIAN_RED,
36: COLOR_KHAKI,
37: COLOR_LIGHT_BLUE,
38: COLOR_LIGHT_GRAY,
39: COLOR_LIGHT_STEEL_BLUE,
40: COLOR_MEDIUM_AQUAMARINE,
41: COLOR_MEDIUM_BLUE,
42: COLOR_MEDIUM_FOREST_GREEN,
43: COLOR_MEDIUM_GOLDENROD,
44: COLOR_MEDIUM_ORCHID,
45: COLOR_MEDIUM_SEA_GREEN,
46: COLOR_MEDIUM_SLATE_BLUE,
47: COLOR_MEDIUM_SPRING_GREEN,
48: COLOR_MEDIUM_TURQUOISE,
49: COLOR_MEDIUM_VIOLET_RED,
50: COLOR_MIDNIGHT_BLUE,
51: COLOR_ORANGE,
52: COLOR_ORANGE_RED,
53: COLOR_ORCHID,
54: COLOR_PALE_GREEN,
55: COLOR_PINK,
56: COLOR_PLUM,
57: COLOR_SALMON,
58: COLOR_SEA_GREEN,
59: COLOR_SIENNA,
60: COLOR_SKY_BLUE,
61: COLOR_SLATE_BLUE,
62: COLOR_SPRING_GREEN,
63: COLOR_STEEL_BLUE,

64: COLOR_TAN,
65: COLOR_THISTLE,
66: COLOR_TURQUOISE,
67: COLOR_VIOLET,
68: COLOR_VIOLET_RED,
69: COLOR_WHEAT,
70: COLOR_YELLOW_GREEN,
71: COLOR_ALICE_BLUE,
72: COLOR_ANTIQUE_WHITE,
73: COLOR_AZURE,
74: COLOR_BEIGE,
75: COLOR_BISQUE,
76: COLOR_BLANCHED_ALMOND,
77: COLOR_BURLYWOOD,
78: COLOR_CHARTREUSE,
79: COLOR_CHOCOLATE,
80: COLOR_CORNSILK,
81: COLOR_CRIMSON,
82: COLOR_DARK_BLUE,
83: COLOR_DARK_CYAN,
84: COLOR_DARK_GOLDENROD,
85: COLOR_DARK_KHAKI,
86: COLOR_DARK_MAGENTA,
87: COLOR_DARK_ORANGE,
88: COLOR_DARK_RED,
89: COLOR_DARK_SALMON,
90: COLOR_DARK_SEA_GREEN,
91: COLOR_DARK_VIOLET,
92: COLOR_DEEP_PINK,
93: COLOR_DEEP_SKY_BLUE,
94: COLOR_DODGER_BLUE,
95: COLOR_FLORAL_WHITE,
96: COLOR_GAINSBORO,
97: COLOR_GHOST_WHITE,

98: COLOR_HONEYDEW,
99: COLOR_HOT_PINK,
100: COLOR_INDIGO,
101: COLOR_IVORY,
102: COLOR_LAVENDER,
103: COLOR_LAVENDER_BLUSH,
104: COLOR_LAWN_GREEN,
105: COLOR_LEMON_CHIFFON,
106: COLOR_LIGHT_CORAL,
107: COLOR_LIGHT_CYAN,
108: COLOR_LIGHT_GOLDENROD_YELLOW,
109: COLOR_LIGHT_GREEN,
110: COLOR_LIGHT_PINK,
111: COLOR_LIGHT_SALMON,
112: COLOR_LIGHT_SEA_GREEN,
113: COLOR_LIGHT_SKY_BLUE,
114: COLOR_LIGHT_SLATE_GRAY,
115: COLOR_LIGHT_YELLOW,
116: COLOR_LINEN,
117: COLOR_MEDIUM_PURPLE,
118: COLOR_MINT_CREAM,
119: COLOR_MISTY_ROSE,
120: COLOR_MOCCASIN,
121: COLOR_NAVAJO_WHITE,
122: COLOR_OLD_LACE,
123: COLOR_OLIVE_DRAB,
124: COLOR_PALE_GOLDENROD,
125: COLOR_PALE_TURQUOISE,
126: COLOR_PALE_VIOLET_RED,
127: COLOR_PAPAYA_WHIP,
128: COLOR_PEACH_PUFF,
129: COLOR_PERU,
130: COLOR_POWDER_BLUE,
131: COLOR_ROSY_BROWN,

```

132: COLOR_ROYAL_BLUE,
133: COLOR_SADDLE_BROWN,
134: COLOR_SANDY_BROWN,
135: COLOR_SEA_SHELL,
136: COLOR_SLATE_GRAY,
137: COLOR_SNOW,
138: COLOR_TOMATO,
139: COLOR_WHITE_SMOKE
}

```

2.3.1.8 Printer Device Capability

The system hardware shall optionally provide an electro-mechanical device that outputs alpha-numeric, graphic and control data to individual sheets of paper via Laser or Ink (applied via impact or spray).

2.3.2 Software Capabilities

The system software shall provide the following cross-platform capabilities::

- 1 *Host Computer Operating System Capability* (on page 205)
- 2 *Command Line Interface Capability* (on page 208)
- 3 *Graphical-Style User Interface Capability* (on page 208)
- 4 *Text Editing Capability* (on page 209)
- 5 *Word Processor Capability* (on page 209)
- 6 *Python Programming Capability* (on page 210)
- 7 *Debugging Capability* (on page 210)
- 8 *Host Console Terminal Application Capability* (on page 211)
- 9 *Optional Terminal Device Emulation Capability* (on page 212)
- 10 *"tsWxGTUI_PyVx" Toolkit Capability* (see *"tsWxGTUI" Toolkit Capability* on page 212)
- 11 *Platform Interface Capability* (on page 242)
- 12 *Application Programming Interface Capability* (on page 242)
- 13 *Operator Interface Capability* (on page 248)

2.3.2.1 Host Computer Operating System Capability

The system software shall provide any of the popular 32-/64-bit multi-user, multi-process and multi-tased/multi-threaded operating system kernels with associated device drivers and run time libraries.

- 1 **Cygwin** (1.7.28)

- a) A free, Linux-style Windows 32-/64-bit add-on from Red Hat.

2 GNU/Linux

- a) CentOS (7.0)

A distribution derived from the same sources used by Red Hat, maintained by a dedicated volunteer community of developers with both 100% Red Hat-compatible versions and an upgraded version that is not always 100% upstream compatible.

- b) Debian (8)

A non-commercial distribution and one of the earliest, maintained by a volunteer developer community with a strong commitment to free software principles and democratic project management

- c) Fedora (16-21)

A community distribution sponsored by American company Red Hat.

- d) OpenSUSE (13.1)

A community distribution mainly sponsored by German company SUSE.

SUSE Linux Enterprise, derived from openSUSE, is maintained and commercially supported by SUSE.

- e) Red Hat Enterprise Linux

A derivative of Fedora, maintained and commercially supported by Red Hat

- f) Scientific (6.5-7.0)

A Linux distribution produced by Fermi National Accelerator Laboratory. It is a free and open source operating system based on Red Hat Enterprise Linux and aims to be "as close to the commercial enterprise distribution as we can get it".

This product is derived from the free and open source software made available by Red Hat, Inc., but is not produced, maintained or supported by Red Hat. Specifically, this product is built from the source code for Red Hat Enterprise Linux versions, under the terms and conditions of Red Hat Enterprise Linux's EULA and the GNU General Public License.

- g) Ubuntu (2.04 LTS & 14.04 LTS)

A popular desktop and server distribution derived from Debian, maintained by British company Canonical Ltd.

3 OS X (formerly known as Mac OS X) (10.3-10.10)

A series of Unix-based graphical interface operating systems developed and marketed by Apple Inc. It is designed to run on Mac computers.

Versions 10.5 "Leopard" running on Intel processors, 10.6 "Snow Leopard", 10.7 "Lion", 10.8 "Mountain Lion", 10.9 "Mavericks", and 10.10 "Yosemite" have obtained UNIX 03 certification.

iOS, which runs on the iPhone, iPod Touch, iPad, and the 2nd and 3rd generation Apple TV, shares the Darwin core and many frameworks with OS X.

- a) Panther (10.3)

- b) Tiger (10.4)

- c) Leopard (10.5)
- d) Snow Leopard (10.6)
- e) Lion (10.7)
- f) Mountain Lion (10.8)
- g) Mavericks (10.9)
- h) Yosemite (10.10)

4 Microsoft Windows (XP, Vista, 7, 8, 8.1)

A metafamily of graphical operating systems developed, marketed, and sold by Microsoft. It consists of several families of operating systems, each of which cater to a certain sector of the computing industry. Active Windows families include Windows NT, Windows Embedded and Windows Phone; these may encompass subfamilies, e.g. Windows Embedded Compact (Windows CE) or Windows Server. Defunct Windows families include Windows 9x and Windows Mobile.

Microsoft Windows will only require "Cygwin", the free Linux-like plug-in from Red Hat for users of the "wxPython"-style, "nCurses"-based Graphical-Text User Interface. (Home or Professional editions of Windows XP, 7, 8, 8.1)

Microsoft introduced an operating environment named Windows on November 20, 1985 as a graphical operating system shell for MS-DOS in response to the growing interest in graphical user interfaces (GUIs).

Microsoft Windows came to dominate the world's personal computer market with over 90% market share, overtaking Mac OS, which had been introduced in 1984. However, it is outsold by Android on smartphones and tablets.

Standard "Command Prompt" and "PowerShell" accessories only support Command Line Interface mode.

Optional "Cygwin" add-on provides support for the character-mode, Graphical-style User Interface via shells, such as "bash", and the POSIX-compatible Application Programming Interface to the "curses" terminal control library for Unix-like systems.

- a) XP (NT 5.1)
- b) Vista (NT 6.0)
- c) 7 (NT 6.1)
- d) 8.0 (NT 6.2)
- e) 8.1 (NT 6.3)

5 Unix

A multitasking, multiuser computer operating system that exists in many variants. The original Unix was developed at AT&T's Bell Labs research center by Ken Thompson, Dennis Ritchie, and others. From the power user's or programmer's perspective, Unix systems are characterized by a modular design that is sometimes called the "Unix philosophy," meaning the OS provides a set of simple tools that each perform a limited, well-defined function, with a unified filesystem as the main means of communication and a shell scripting and command language to combine the tools to perform complex workflows.

- a) FreeBSD (9.2, 10.0)

A free Unix-like operating system descended from Research Unix via the Berkeley Software Distribution (BSD). Although for legal reasons FreeBSD cannot use the Unix trademark, it is a direct descendant of BSD, which was historically also called "BSD Unix" or "Berkeley Unix."

b) OpenIndiana (151a8)

A free and open-source, Unix operating system derived from OpenSolaris. Developers forked OpenSolaris after Oracle Corporation discontinued it, in order to continue development and distribution of the source code. The OpenIndiana project is stewarded by the illumos Foundation, which also stewards the illumos operating system. OpenIndiana's developers strive to make it "the defacto OpenSolaris distribution installed on production servers where security and bug fixes are required free of charge".

c) OpenSolaris (11)

A descendant of the UNIX System V Release 4 (SVR4) code base developed by Sun and AT&T in the late 1980s. It is the only version of the System V variant of UNIX available as open source.

d) PC-BSD, or PCBSD (10)

A Unix-like, desktop-oriented operating system built upon the most recent releases of FreeBSD. It aims to be easy to install by using a graphical installation program, and easy and ready-to-use immediately by providing KDE SC, LXDE, Xfce, and MATE as the graphical user interface.

2.3.2.2 Command Line Interface Capability

The system software shall provide a command-line interface (CLI), also known as command-line user interface, console user interface, and character user interface (CUI), as a means of interacting with a computer program where the user issues commands to the program in the form of successive lines of text (command lines).

See: *Console Terminal Application Capability* (see "*Host Console Terminal Application Capability*" on page 211)

2.3.2.3 Graphical-Style User Interface Capability

The system software shall provide a graphical user interface (GUI) as a type of user interface that allows users to interact with electronic devices through graphical icons, buttons, check boxes, gauges, sliders, scrollbars and other visual indicators, as opposed to text-based interfaces, typed command labels or text navigation. GUIs were introduced in reaction to the perceived steep learning curve of command-line interfaces (CLI), which require commands to be typed on the keyboard.

1 Linux

GNOME - A desktop environment which runs on multiple operating systems, its main focus being those based on the Linux kernel.

KDE - A desktop environment designed to run on Linux, FreeBSD, Solaris, Microsoft Windows, and OS X systems.

X11 - A windowing system for bitmap displays, common on UNIX-like operating systems.

2 Mac OS X

Apple Inc. proprietary.

X11 - A windowing system for bitmap displays, common on UNIX-like operating systems.

3 Microsoft Windows

Microsoft Corporation proprietary.

4 Unix

GNOME - A desktop environment which runs on multiple operating systems, its main focus being those based on the Linux kernel.

KDE - A desktop environment designed to run on Linux, FreeBSD, Solaris, Microsoft Windows, and OS X systems.

X11 - A windowing system for bitmap displays, common on UNIX-like operating systems.

2.3.2.4 Text Editing Capability

The system software shall provide a tool to create and modify files that use a simple character set, such as ASCII, to represent numbers, letters, and a small number of symbols. The only non-printing characters in the file that can be used to format the text, are newline, tab, and formfeed.

1 Linux

GNU Emacs/XEmacs

vim

2 Mac OS X

TextEdit

TextWrangler

3 Microsoft Windows

Notepad

XEmacs

4 Unix

GNU Emacs/XEmacs

vim

2.3.2.5 Word Processor Capability

The system software shall provide a tool to create and modify files that contain formatted text, such as enabling text to appear in boldface and italics, that use multiple fonts, and that can be structured into columns and tables. These capabilities were once associated only with desktop publishing, but are now available in the simplest word processor.

1 Linux

LibreOffice

2 Mac OS X

LibreOffice

Microsoft Office

3 Microsoft Windows

Author-it

LibreOffice

Microsoft Office

4 Unix

LibreOffice

2.3.2.6 Debugging Capability

The system software shall provide a tool used to debug computer programs. The tool typically displays the lines of source code. It enables the user to set break points to trace the execution path. It enables the user to display the current value of constants and variables.

1 Linux

GNU gdb

Python Idle

WingIDE Pro

2 Mac OS X

GNU gdb

Python Idle

WingIDE Pro

3 Microsoft Windows

GNU gdb

Python Idle

WingIDE Pro

4 Unix

GNU gdb

Python Idle

2.3.2.7 Python Programming Capability

For those applications that require it, the system software shall provide tools to execute and troubleshoot source code components written in in application-specific releases of:

1 A default third-generation Python programming language.

Python 3.2.0-3.2.5

NOTE: It may not be trivial or even practical to back port and re-engineer some of the functional and interface capabilities to Python 3.0-3.1 from those available in Python 3.2.0-3.2.5.

It may be trivial to re-engineer some of the functional and interface capabilities to Python 3.3 -3.4 from those available in Python 3.2.0-3.2.5.

2 An default second-generation Python programming language.

Python 2.6.6-2.7.6

NOTE: It may not be trivial or even practical to back port and re-engineer some of the functional and interface capabilities to Python 2.0-2.6.5 from those available in 2.6.6-2.7.6.

3 An optional first-generation Python programming language.

Python 1.6.x

NOTE: It may be quite a challenge to back port and re-engineer some of the functional and interface capabilities to Python 1.6.x from those available in 2.6.6-2.7.6.

2.3.2.8 Host Console Terminal Application Capability

For those applications that require it, the system software shall use the system's Terminal Control Library Interface and provide application tools to emulate various system console hardware terminal features.

Each of the following application or utility programs provides its own collection of user selectable preferences for setting cursor type, font type/size, color palette, blinking cursor/text, default terminal emulator type etc.

1 GNU/Linux

GNOME Terminal

KDE Terminal

2 Mac OS X

iTerm --- Third-party utility, organizes its preferences into the General, Appearance, Profiles, Keys, Pointer and Arrangments tabs.

Terminal --- Utility similar to ones available on GNU/Linux and other Unix systems. It organizes its preferences into the General, Profiles, Window Groups and Encodings tabs.

3 Microsoft Windows (*using Cygwin, the free add-on from Red Hat, that provides a Linux-like Command Line Interface and GNU Toolkit*)

Cygwin Terminal (mintty)

4 Microsoft Windows (*Native, without using Cygwin, the free add-on from Red Hat, that provides a Linux-like Command Line Interface and GNU Toolkit*)

Command Prompt (supports a non-Linux/non-Unix Command Line Interface that does NOT support Python Curses and its character-mode Graphical-style User Interface)

Excerpt from http://en.wikipedia.org/wiki/Command_Prompt:

"Command Prompt (executable name cmd.exe) is the Microsoft-supplied command-line interpreter on OS/2, Windows CE and on Windows NT-based operating systems (including Windows 2000, XP, Vista, 7, 8, Server 2003, Server 2008, Server 2008 R2 and Server 2012). It is the analog of COMMAND.COM in MS-DOS and Windows 9x systems (where it is called "MS-DOS Prompt"), or of the Unix shells used on Unix-like systems."

Windows PowerShell (supports a non-Linux/non-Unix Command Line Interface that does NOT support Python Curses and its character-mode Graphical-style User Interface)

Excerpt from http://en.wikipedia.org/wiki/Windows_PowerShell:

"Windows PowerShell is a task automation and configuration management framework from Microsoft, consisting of a command-line shell and associated scripting language built on the .NET Framework. PowerShell provides full access to COM and WMI, enabling administrators to perform administrative tasks on both local and remote Windows systems as well as WS-Management and CIM enabling management of remote Linux systems and network devices."

5 Unix

Terminal

See *Optional Terminal Emulation Capability* (see "*Optional Terminal Device Emulation Capability*" on page 212) for additional terminal hardware emulation requirements.

2.3.2.9 Optional Terminal Device Emulation Capability

For those applications that require it, the system software shall enhance the *Host Console Terminal Application Capability* (on page 211) with one or more of the following optional terminal emulation features:

2.3.2.10 "tsWxGTUI" Toolkit Capability

This paragraph shall identify a required system capability and shall itemize the requirements associated with the capability. If the capability can be more clearly specified by dividing it into constituent capabilities, the constituent capabilities shall be specified in subparagraphs. The requirements shall specify required behavior of the system and shall include applicable parameters, such as response times, throughput times, other timing constraints, sequencing, accuracy, capacities (how much/how many), priorities, continuous operation requirements, and allowable deviations based on operating conditions. The requirements shall include, as applicable, required behavior under unexpected, unallowed, or "out of bounds" conditions, requirements for error handling, and any provisions to be incorporated into the system to provide continuity of operations in the event of emergencies. Paragraph 3.3.x of this DID provides a list of topics to be considered when specifying requirements regarding inputs the system must accept and outputs it must produce.

2.3.2.10.1 "tsToolkitCLI" Command Line Interface Capability

The "tsWXTGUI" Toolkit software shall provide a library of building blocks and application programs that support the development of non-graphical applications. Such applications are invoked via textual commands entered via the console keyboard. Output is viewed on the console display.

2.3.2.10.1.1 CLI Terminal Interface Capability

The "tsWXTGUI" Toolkit software shall provide the following:

- 1 **Hardware** - Terminal for input from operator (such as an electronic keyboard) and hardware for output to operator (such as an electronic display or printer).
- 2 **Software** - Shell application (typically provided by the operating system) to receive, interpret and process command input from operator that cause actions which gather, process, format and output information to operator. It is a command processor that's typically run in a one dimensional text window (on the line displaying the command prompt), allowing the user to type commands which cause actions. It can also read commands from a file, called a script.

Command Line Interface mode input/output streams:

- **stdin** - POSIX-type standard input stream of one or more characters from an operator's keyboard or redirected input from a file or pipe (connection with the output stream from another running program).
- **stdout** - POSIX-type standard output stream of one or more characters to an operator's display or redirected output to a file or pipe (connection with the input stream of another running program).
- **stderr** - POSIX-type standard error output stream of one or more characters to an operator's display, system administrator's display or redirected output to a file or pipe (connection with the input stream of another running program).

Upon successfully launching the Graphical-style User Interface mode, and until terminating it, the following input/output streams become available:

- **stdscr** - Curses-type standard screen output stream of one or more characters to an operator's display. Before an application can use this stream, the application must initialize curses. This is done by calling the curses module's `initscr()` function, which will determine the terminal type, send any required setup codes to the terminal, and create various internal data structures. If successful, `initscr()` returns a window object representing the entire screen; this is usually called `stdscr`, after the name of the corresponding C variable.
- **stdscr.getch** - The `getch()` method returns an integer; if it's between 0 and 255, it represents the ASCII code of the key pressed. Values greater than 255 are special keys such as Page Up, Home, or the cursor keys. A value of -1 represents an input timeout. You can compare the value returned to constants such as `curses.KEY_PPAGE`, `curses.KEY_HOME`, or `curses.KEY_LEFT`. You can also compare the value returned to constants such as `curses.BUTTON1_PRESSED`, `curses.BUTTON1_RELEASED`, `curses.BUTTON1_CLICKED`, `curses.BUTTON1_DOUBLE_CLICKED` and `curses.BUTTON1_TRIPLE_CLICKED`.
- **stdscr.getstr** - The `getstr()` method retrieves an entire string. It isn't used very often, because its functionality is quite limited; the only editing keys available are the backspace key and the Enter key, which terminates the string. It can optionally be limited to a fixed number of characters.

2.3.2.10.1.2 "tsLibCLI" Capability

The "tsWXTGUI" Toolkit software shall provide the following:

- 1 tsApplicationPkg** - Python base class to initialize and configure the application program launched by an operator. It enables an application launched via a Command Line Interface (CLI) to initialize, configure and use the same character-mode terminal with a Graphical-style User Interface (GUI).

 - a) Input provided on the command line by an operator. The command line uses a Unix-style, free-format to promote future enhancement and on-going maintenance.
 - b) Input provided in the parameter list of the application's invocation of the class instantiation. The parameter list uses a Python-style free-format to promote future enhancement and on-going maintenance.

- 2 tsCommandLineEnvPkg** - Python class to initialize and configure the application program launched by an operator. It delivers those keyword-value pair options and positional arguments specified by the application, in its invocation parameter list. It wraps the Command Line Interface application with exception handlers to control exit codes and messages that may be used to co-ordinate other application programs..
- 3 tsCommandLineInterfacePkg** - Python class establishes methods that prompt or re-prompt the operator for input, validate that the operator has supplied the expected number of inputs and that each is of the expected type.
- 4 tsCxGlobalsPkg** - Python class to provide a theme-based centralized mechanism for modifying/restoring those configuration constants that can be interrogated at runtime by those software components having a "need-to-know". The intent being to avoid subsequent searches to locate and modify or restore a constant appropriate to the current configuration, users and their activities.
- 5 tsExceptionPkg** - Python class to define and handle error exceptions. It maps various error exceptions into an 8-bit exit code and message that can be used to coordinate the actions of a set of shell scripts.
- 6 tsLoggerPkg** - Python class to define and handle event message timestamping, formatting and output. It also supports logging of assert and check case results.

It supports the following message severity levels: NOTSET (lowest priority), DEBUG, INFO, NOTICE, WARNING, ALERT, ERROR, CRITICAL, EMERGENCY (highest priority) and PRIVATE.

It defines and handles event message processing associated with the following: wxPython/wxWidget exceptions: wxASSERT, wxASSERT_MSG, wxCHECK, wxCHECK2, wxCHECK2_MSG, wxCHECK_MSG, wxCHECK_RET, wxFAIL, wxFAIL_COND_MSG, wxFAIL_MSG and wxTRAP.

- 7 tsOperatorSettingsParserPkg** - Class to parse command line options and arguments.

Supports one or more of the parser module(s) available in the Python version(s) supported by the application.

 - a) "argparse" (introduced with Python 2.7.0)
 - b) "optparse" (introduced with Python 2.3.0)
 - c) "getopt" (introduced with Python 1.6.0)

When used with Python 2.7 or Python 3.2, the "tsOperatorSettings.py" module (a "tsLibCLI" component of the "tsWxGTUI_PyVx" Toolkit) supports the current and legacy Python version specific parser module(s) as an experimental and educational opportunity.

However, when one seeks to back port applications to Python 2.0-2.6 or Python 3.0-3.1, the "tsOperatorSettings.py" module must be stripped of unsupported Python version specific parser modules in order to prevent a program trap which will block the application from running.

- 8 tsPlatformRunTimeEnvironmentPkg** - Class to capture current hardware and software information about the run time environment for the user process.
- 9 tsReportUtilityPkg**- Class defining methods used to format information such as time, date, data size, elapsed time and nested dictionary contents.
- 10 tsSysCommandsPkg** - Class definition and methods for issuing shell commands to the host operating system.

2.3.2.10.1.3 "tsToolsCLI" Capability

The "tsWXTGUI" Toolkit software shall provide the following:

- 1 tsLinesOfCodeProjectMetrics** - Python application program, with a Command Line Interface (CLI), that generates reports of software project progress and the estimated cost (or contributed value) of the project when it is finally completed.
 - a) It scans an operator designated file directory tree containing the source files, in one or more programming language specific formats (such as Ada, Assembler, C/C++, Cobol, Fortran, PL/M, Python, Text, and various command line shells).

For each file, it accumulates and reports the total number of code lines, blank/comment lines, words and characters.

For each programming language format, it accumulates and reports a summary of details of the associated source files.

For the entire set of source files, it accumulates and reports a summary of details.

It uses the summary of the entire set of source files to derive, analyze, estimate and report metrics for the software development project (such as labor, cost, schedule and lines of code per day productivity).
 - b) It supports one or more of the parser module(s) available in the Python version(s) supported by the application.

"argparse" (introduced with Python 2.7.0)

"optparse" (introduced with Python 2.3.0)

"getopt" (introduced with Python 1.6.0)

When used with Python 2.7 or Python 3.2, the "tsOperatorSettings.py" module (a "tsLibCLI" component of the "tsWxGTUI_PyVx" Toolkit) supports the current and legacy Python version specific parser module(s) as an experimental and educational opportunity.

However, when one seeks to back port applications to Python 2.0-2.6 or Python 3.0-3.1, the "tsOperatorSettings.py" module must be stripped of unsupported Python version specific parser modules in order to prevent a program trap which will block the application from running.

- 2 tsPlatformQuery** - Python application program, with a Command Line Interface (CLI), that captures current hardware and software information about the run time environment for the user process.

(a) Host processor hardware support includes various releases of x86, PowerPC, SPARC.

(b) Host operating system software support includes various releases of Cygwin, Linux ('Debian', 'Fedora', 'mandriva', 'redhat', 'Scientific / Centos', 'SuSE'), Mac OS X, Microsoft Windows ('XP', 'Vista', '7', '8', '8.1') and Unix ('FreeBSD / PC-BSD', 'IRIX', 'OpenIndiana', 'Solaris / SunOS').

(c) Host virtual machine software support includes various releases of Java and Python.

- 3 tsStripComments** - Python application program, with a Command Line Interface (CLI), that transforms an annotated, development version of a directory of sub-directories and Python source files into an unannotated copy.

The copy is intended to conserve storage space when installed in an embedded system. The transformation involves stripping comments and doc strings by de-tokenizing a tokenized version of each Python source file. Non-Python files are trimmed of trailing whitespace.

- 4 tsStripLineNumbers** - Python application program, with a Command Line Interface (CLI), that strips line numbers from source code (such as annotated FORTRAN program listings) that (unlike BASIC program listings) do not reference line numbers for conditional branching.

Output from fixed format FORTRAN (F77) code is NOT corrected to ensure that each statement first character begins in column 7 and that each ampersand ("&") continuation character is in column 6.

- 5 tsTreeCopy** - Python application program, with a Command Line Interface (CLI), that copies the contents of a source directory to a target directory.

- 6 tsTreeTrimLines** - Python application program, with a Command Line Interface (CLI), that copies the contents of a source directory to a target directory after stripping superfluous white space (blanks) from end of each line.

2.3.2.10.1.4 "tsTestsCLI" Capability

This section is under construction.

2.3.2.10.2 "tsToolkitGUI" Graphical-style User Interface Capability

The "tsWXTGUI" Toolkit software shall provide a library of building blocks and application programs that support the development of graphical-style applications. Such applications are invoked via textual commands entered via the console keyboard and by graphical commands entered via mouse (cursor position and button click). Output is viewed on the console display.

2.3.2.10.2.1 GUI Terminal Interface Capability

The "tsWXTGUI" Toolkit software shall provide the following:

- 1 Hardware** - Terminal for input from operator (such as an electronic keyboard, mouse, trackball, touchpad or touchscreen) and hardware for output to operator (such as an electronic display or printer).
- 2 Software** - Shell application (typically provided by the operating system) to receive, interpret and process command input from operator that cause actions which gather, process, format and output information to operator. It is a command processor that's typically run in a two dimensional graphical screen, allowing the user to "click" on objects (such as buttons, check boxes, radio buttons, windows and dialogs) and type commands which cause actions. It can also read commands from a file, called a script.

- **Platform Dependent API** - Software components that provide an Application Programming Interface that is uniquely customized for the platform's physical hardware and operating system software (such as Linux, Mac OS X and Windows).
- **Terminal Independent API** - Software components of "nCurses" that provide a common Application Programming Interface regardless of a platform's physical hardware and operating system software.
- **wxPython API** - Software components of "tsWxGraphicalTestUserInterface" that provide a subset of the "wxPython" Application Programming Interface. It includes class definition and methods to initialize, configure, create GUI objects and shutdown the platform's nCurses-based terminal interface.

2.3.2.10.2.2 "tsLibGUI" Capability

The "tsWXTGUI" Toolkit software shall provide the following:

- **Desktop API** (on page 218) - Describes the host operating system-style Graphical User Interface features (such as multiple top-level windows, task bar and redirected stderr and stdout displays) that the Software Engineer can include for input and output interactions with the System OperatorSystem Operator.
- **High-Level Widget API** (on page 219) - Identifies the wxPython-style Graphical User Interface features (such as frames, dialogs and buttons) that the Software Engineer can include for input and output interactions with the System Operator.
- **Event API** (see "*Event API (Draft)*" on page 223) - Describes the basic and additional wxPython-style Graphical User Interface features used for sending, receiving and handling event notifications (such as clock tick, button clicked and shift in window focus).
- **Low-Level Widget API** (on page 241) - Identifies the basic nCurses-style Graphical User Interface features that the Software Engineer can use to construct High Level Widgets that customize input and output interactions with the local or remote terminal.
- **System Preference API** (see "*System Preferences API (Draft)*" on page 241) - Describes the wxPython-style Graphical User Interface features (such as Symbolic Constants, Foreground / Background Colors, Default Styles / Themes and utility functions) that the Software Engineer can use to standardize the look and feel of input and output interactions with the System Operator.

2.3.2.10.2.2.1 Desktop API

from http://en.wikipedia.org/wiki/Multiple_document_interface:

"Graphical computer applications with a multiple document interface (MDI) are those whose windows reside under a single parent window (usually except for modal windows), as opposed to all windows being separate from each other (single document interface). Such systems often allow child windows to embed other windows inside them as well, creating complex nested hierarchies. In the usability community, there has been much debate about which interface type is preferable. Generally, SDI is seen as more useful in cases where users work with more than one application. Software companies have used both interfaces with mixed responses. For example, Microsoft changed its Office applications from SDI to MDI mode and then back to SDI, although the degree of implementation varies from one component to another.

The disadvantage of MDI usually cited is the lack of information about the currently opened windows: In order to view a list of windows open in MDI applications, the user typically has to select a specific menu ("window list" or something similar), if this option is available at all. With an SDI application, the window manager's task bar or task manager (if any) can display the currently opened windows. In recent years, applications have increasingly added "task-bars" and "tabs" to show the currently opened windows in an MDI application, which has made this criticism somewhat obsolete. Some people call this interface "tabbed document interface" (TDI). When tabs are used to manage windows, individual ones usually cannot be resized or used simultaneously."

Multiple Document Interface capabilities currently include:

- 1 Screens** - A display (shown with black or white background) of a physical terminal device or a GUI object representing an emulated terminal whose size, position and font can (usually) be changed by the user. It usually has thick borders and a title bar, and can optionally contain a menu bar, toolbar and status bar. A screen can contain one or more frames and dialogs within which may be those GUI objects that are not frames or dialogs.
- 2 Multi-Frame Environment** - A non-wxPython feature consisting of one or more Frames (shown with blue background) and Dialogs (shown with cyan background) associated with the GUI application.
- 3 Redirected Output** - A non-wxPython feature consisting of a Frame (shown with green background) containing one or more lines of text messages. The message text is produced by the application's use of print or write statements with output normally intended for the "stderr" and "stdout" devices. The GUI toolkit intercepts the output and redirects it to a reserved frame on the display screen after annotating it with a date and time stamp.
- 4 Task Bar** - A non-wxPython feature consisting of a Frame (shown with black background) that monitors and controls the application tasks. The top-right line identifies the name of the application executable. The bottom-right line contains a character-mode symbol that appears to rotate. The rotation denotes progress. The middle line(s) contain buttons for each of the application's Frame and Dialog "tasks". If a mouse button is clicked when the cursor is over a button, an associated function or method will be initiated. The function or method will send a signal to notify other GUI objects of the event. The event will raise the focus of the selected task so that its frame or dialog becomes the top-most overlays.

NOTES:

The Desktop API includes a MultiFrame Environment that establishes the following Top Level Windows:

- 1) The Application Programmer designated Top Level Frame and Dialog, at top of screen.
- 2) The Redirected I/O Frame provides the operator with the most recent stderr, stdout and print messages, in middle of screen.
- 3) The Task Bar Frame provides the operator with a list of currently open frame and dialog windows from which to click on and change the focus, at bottom of screen.

2.3.2.10.2.2.2 High-Level Widget API

NOTES:

- 1) As a character-mode GUI-style toolkit, this product does not support those "wxPython" features associated with graphical elements such as bit images, icons, proportional-spaced fonts or HTML and XML text markup. The current release supports the porting of "wxPython" 2.8.9.2 application(s) to platforms running Python 2.5.x, 2.6.x, 2.7.x, 3.1.x and 3.2.x.
 - 2) Technical and resource issues drive the development effort. Initially, development focussed on establishing the feasibility of emulating core components of the the "wxPython" and associated "wxWidgets" API. Development then iteratively seeks to establish usability-enhancing components and to then to identify and resolve any appearance, behavior and API-conformance issues.
 - 3) See the unpublished "**tsWxGTUI_PyVx**" Vol. 7 - Design Notes (it was created only for personal use) for the identification and an overview of those currently implemented class modules, ones currently under construction and ones for which development applicability and/or commitments are still TBD.
-

The High-Level Widget API identifies the basic wxPython-style Graphical User Interface features that the Software Engineer can include for input and output interactions with the System Operator. It includes such widgets as the following:

- 1 Button** - GUI object that may contain text or character-mode icons. If a mouse button is clicked when the cursor is over the object, an associated function or method will be initiated. The function or method will send a signal to notify other GUI objects of the event. Buttons are clickable windows that trigger an associated event (such as start, pause, resume or terminate an operation).
- 2 Carat** - GUI object that may contain a set of special character-mode symbols, usually including a solid rectangle or a blinking underline character, showing the position where the typed text will appear.
- 3 Check Box** - GUI object that may contain text or character-mode icons. If a mouse button is clicked when the cursor is over the object, it initiates an associated function or method. The function or method will toggle the check box to the next "ON", "OFF" or "TRI-STATE" value. The function or method will send a signal to notify other GUI objects of the event. Checkboxes are buttons to toggle the enabled or disabled state of an associated feature (such as start or stop logging)
- 4 Cursor** - A special character-mode symbol, usually a solid rectangle or a blinking underline character, that signifies where the next character will be displayed on the screen.

- 5 Dialog** - A GUI object with a title bar and sometimes a system menu, which can be moved around the screen. It can contain controls and other GUI objects and is often used to allow the user to make some choice or to answer a question. Dialogs are pop-up windows associated with an application task's menu bar (such as an input field for an operator's search criteria and an output field for a list of candidate WEB sites) that will be terminated upon completion.
- 6 Frame** - GUI object whose size and position can (usually) be changed by the user. It usually has thick borders and a title bar, and can optionally contain a menu bar, toolbar and status bar. A frame can contain any GUI object that is not a frame or dialog. Frames are windows associated with an application task (such as an internet WEB Browser) that can be minimized into an icon, expanded to full screen or terminated upon operator demand.
- 7 Gauge** - GUI object whose horizontal or vertical bar shows a quantity (often time). It supports two working modes: determinate and indeterminate progress. First is the usual working mode while the second can be used when the program is doing some processing but you don't know how much progress is being done. In this case, you can periodically call the Pulse function to make the progress bar switch to indeterminate mode (graphically it's usually a set of blocks which move or bounce in the bar control). Gauges are used to indicate progress of an associated operation (scale with range such as 0% to %100%).
- 8 Menu** - GUI object that features a popup (or pull down) list of items, one of which may be selected before the menu goes away (clicking elsewhere dismisses the menu). It may be used to construct either menu bars or popup menus.
- 9 Menu Bar** - GUI object that features a series of menus accessible from the top of a frame. Each operator selectable entry triggers an associated event (such as create a file).
- 10 Panel** - GUI object that features a window on which controls are placed. Panels are usually placed within a frame. Its main feature over its parent window class is code for handling child windows and TAB traversal. Panels are container windows for other Lower Level GUI Objects.
- 11 Radio Box** - GUI objects that are used to select one of number of mutually exclusive choices. A radio box is displayed as a vertical column or horizontal row of labelled radio buttons. Radio Boxes are a collection of Buttons associated with the same group (such a AM or FM band) that are interdependent such that any one can become activated after the others simultaneously become deactivated.
- 12 Radio Button** - GUI object that may contain text or character-mode icons. If a mouse button is clicked when the cursor is over the object, it initiates an associated function or method. The function or method will turn the associated radio button "ON" and turn "OFF" all other radio buttons within the associated radio box. The function or method will send a signal to notify other objects of the event. Radio Buttons are used to activate one of the associated features (such as broadcast channel selection) after the other features associated with the same Radio Box group (such a AM or FM band) have become deactivated.
- 13 ScrollBar** - GUI object that includes a horizontal or vertical ScrollBarGauge between two ScrollBarButtons. The operator uses it to re-position (pan) the contents of an associated ScrolledText window.
- 14 ScrollBarButton** - GUI object that includes one arrow symbol ("<", ">", "^" or "V") that indicates the horizontal or vertical direction of scrolling. When the operator clicks on a ScrollBarButton, the contents of the associated ScrolledText window moves.

- A single Left Click on one of the two horizontal ScrollBarButtons moves the text by one column. A rapid double Left Click on one of the two horizontal ScrollBarButtons moves the ScrolledText by one page (the horizontal column width). A single Right Click on one of the two horizontal ScrollBarButtons moves the text to the appropriate horizontally end point (either left/right most column).
 - A single Left Click on one of the two vertical ScrollBarButtons moves the text by one row. A rapid double Left Click on one of the two vertical ScrollBarButtons moves the ScrolledText by one page (the vertical row height). A single Right Click on one of the two vertical ScrollBarButtons moves the text to the appropriate vertical end point (top/bottom most row).
- 15 ScrollBarGauge** - GUI object located between two ScrollBarButtons, contains a bar graph that indicates the position and size of the displayed text relative to the non-displayed text.
- When the bar graph is empty (blank), there is no text available to be displayed.
 - When it is completely filled, all of the available text is displayed.
 - When it is partially filled, the starting point of the graph displays the starting point of the displayed text relative to the available text. The size of the filled graph displays the size of the displayed text relative to the available text.
 - When the operator single Left Clicks on a ScrollBarGauge between its two associated ScrollBarButtons, the contents of the ScrolledText window moves in proportion to the difference between the click and the associated text end positions.
- 16 Scrolled** - An application-independent GUI object base class for ScrolledWindow. It lays out the position and size of one ScrolledText window and one or two ScrollBars each with their associated ScrollBarGauge and pair of ScrollBarButtons. It enables an operator to select the columns and or rows of text to be fit and displayed within the available peep hole viewing area.
- 17 ScrolledText** - GUI object that allows one or more lines of text to be appended to a retained list. In conjunction with one or two associated pairs of ScrollBarButtons, it enables an operator to select the columns and or rows of text to be fit and displayed within the available peep hole viewing area.
- 18 ScrolledWindow** - GUI object that instantiates an application-specific instance of Scrolled to lay out the position and size of one ScrolledText window and one or two ScrollBars each with their associated ScrollBarGauge and pair of ScrollBarButtons. It enables an operator to select the columns and or rows of text to be fit and displayed within the available peep hole viewing area.
- 19 Splash Screen** - GUI object that simulates a pixel-mode bitmap image which is displayed upon application startup. It may contain text or character-mode icons. It features a window with a thin border and text describing the application. The bitmap-like display may be created from Panel, BoxSizer, GridSizer and TextCtrl widgets. It is created and shown during application initialization, before the application's own window. It either explicitly destroys itself or disappears after a it time-out.
- 20 Status Bar** - GUI object that feature a narrow window that can be placed along the bottom of a frame to give small amounts of status information. The object can contain one or more fields, one or more of which can be variable length according to the size of the window.
- 21 Static Text** - GUI object that features a text control that displays one or more lines of read-only text.
- 22 Text Ctrl** - GUI object that features a text control which allows text to be displayed and edited. It may be single line or multi-line. The text may be indented, line-wrapped and scrolled to fit the available display area.

23 Tool Bar (Future) - GUI object that features a series of tool entries accessible from the top of a frame. Each operator selectable entry triggers an associated event that starts the selected tool in a new Frame.

NOTES:

1) Event Handling support currently associates mouse clicks with the enclosing GUI object that is not obscured by overlaying GUI objects. The toolkit generates an event notification and dispatches it to the event handler designated by the application. It will dispatch unhandled event notifications to a default handler.

2) Keyboard Input Events are detected. The raw characters are echoed but are NOT currently forwarded to the window object having focus.

3) Mouse Input Events are detected. The x-y coordinates and button state are echoed. Single Left Clicks, Double Left Click and Right Clicks are forwarded to the window object positioned to receive and process the event. More complex GUI event detection, analysis and processing is not yet supported. Platform-specific vt100/vt220 terminal emulators do NOT conform to the xterm/xterm-color mouse click event protocol recognized by nCurses. Instead of a single mouse click event, the vt100/vt220 terminal emulators generate a string of escape sequence events. It is unknown if the "tsWxGTUI_PyVx" Toolkit can develop logic to synthesize an xterm/xterm-color mouse click event from the string of vt100/vt220 escape sequence events.

4) Automatic layout support is currently provided by the BoxSizer and GridSizer widgets or by combinations of them. More complex GUI object positioning and sizing is not yet supported.

2.3.2.10.2.2.3 Event API (Draft)

 from http://docs.wxwidgets.org/trunk/overview_events.html

Introduction to Events

Like with all the other GUI frameworks, the control of flow in wxWidgets applications is event-based: the program normally performs most of its actions in response to the events generated by the user. These events can be triggered by using the input devices (such as keyboard, mouse, joystick) directly or, more commonly, by a standard control which synthesizes such input events into higher level events: for example, a wxButton can generate a click event when the user presses the left mouse button on it and then releases it without pressing Esc in the meanwhile. There are also events which don't directly correspond to the user actions, such as wxTimerEvent or wxSocketEvent.

But in all cases wxWidgets represents these events in a uniform way and allows you to handle them in the same way wherever they originate from. And while the events are normally generated by wxWidgets itself, you can also do this, which is especially useful when using custom events (see Custom Event Summary).

To be more precise, each event is described by:

Event type: this is simply a value of type wxEventType which uniquely identifies the type of the event. For example, clicking on a button, selecting an item from a list box and pressing a key on the keyboard all generate events with different event types.

Event class carried by the event: each event has some information associated with it and this data is represented by an object of a class derived from wxEvent. Events of different types can use the same event class, for example both button click and listbox selection events use wxCommandEvent class (as do all the other simple control events), but the key press event uses wxKeyEvent as the information associated with it is different.

Event source: wxEvent stores the object which generated the event and, for windows, its identifier (see Window Identifiers). As it is common to have more than one object generating events of the same type (e.g. a typical window contains several buttons, all generating the same button click event), checking the event source object or its id allows to distinguish between them.

Event Handling

There are two principal ways to handle events in wxWidgets. One of them uses event table macros and allows you to define the binding between events and their handlers only statically, i.e., during program compilation. The other one uses wxEvtHandler::Bind<>() call and can be used to bind and unbind, the handlers dynamically, i.e. during run-time depending on some conditions. It also allows the direct binding of events to:

A handler method in another object.

An ordinary function like a static method or a global function.

An arbitrary functor like boost::function<>.

The static event tables can only handle events in the object where they are defined so using `Bind<>()` is more flexible than using the event tables. On the other hand, event tables are more succinct and centralize all event handler bindings in one place. You can either choose a single approach that you find preferable or freely combine both methods in your program in different classes or even in one and the same class, although this is probably sufficiently confusing to be a bad idea.

Also notice that most of the existing wxWidgets tutorials and discussions use the event tables because they historically preceded the apparition of dynamic event handling in wxWidgets. But this absolutely doesn't mean that using the event tables is the preferred way: handling events dynamically is better in several aspects and you should strongly consider doing it if you are just starting with wxWidgets. On the other hand, you still need to know about the event tables if only because you are going to see them in many samples and examples.

So before you make the choice between static event tables and dynamically connecting the event handlers, let us discuss these two ways in more detail. In the next section we provide a short introduction to handling the events using the event tables. Please see [Dynamic Event Handling](#) for the discussion of `Bind<>()`.

Event Handling with Event Tables

To use an event table you must first decide in which class you wish to handle the events. The only requirement imposed by wxWidgets is that this class must derive from `wxEvtHandler` and so, considering that `wxWindow` derives from it, any classes representing windows can handle events. Simple events such as menu commands are usually processed at the level of a top-level window containing the menu, so let's suppose that you need to handle some events in `MyFrame` class deriving from `wxFrame`.

First define one or more event handlers. They are just simple methods of the class that take as a parameter a reference to an object of a `wxEvent`-derived class and have no return value (any return information is passed via the argument, which is why it is non-const). You also need to insert a macro

```
wxDECLARE_EVENT_TABLE()
```

somewhere in the class declaration. It doesn't matter where it appears but it's customary to put it at the end because the macro changes the access type internally so it's safest if nothing follows it. The full class declaration might look like this:

```
class MyFrame : public wxFrame
```

```
{
```

```
public:
```

```
    MyFrame(...) : wxFrame(...) { }
```

```
...
```

protected:

```
int m_whatever;
```

private:

```
// Notice that as the event handlers normally are not called from outside
```

```
// the class, they normally are private. In particular they don't need
```

```
// to be public.
```

```
void OnExit(wxCommandEvent& event);
```

```
void OnButton1(wxCommandEvent& event);
```

```
void OnSize(wxSizeEvent& event);
```

```
// it's common to call the event handlers OnSomething() but there is no
```

```
// obligation to do that; this one is an event handler too:
```

```
void DoTest(wxCommandEvent& event);
```

```
DECLARE_EVENT_TABLE()
```

```
};
```

Next the event table must be defined and, as with any definition, it must be placed in an implementation file. The event table tells wxWidgets how to map events to member functions and in our example it could look like this:

```
wxBEGIN_EVENT_TABLE(MyFrame, wxFrame)
```

```
EVT_MENU(wxID_EXIT, MyFrame::OnExit)
```

```
EVT_MENU(DO_TEST, MyFrame::DoTest)
```

```
EVT_SIZE(MyFrame::OnSize)
```

```
EVT_BUTTON(BUTTON1, MyFrame::OnButton1)
```

```
wxEND_EVENT_TABLE()
```

Notice that you must mention a method you want to use for the event handling in the event table definition; just defining it in `MyFrame` class is not enough.

Let us now look at the details of this definition: the first line means that we are defining the event table for `MyFrame` class and that its base class is `wxFrame`, so events not processed by `MyFrame` will, by default, be handled by `wxFrame`. The next four lines define bindings of individual events to their handlers: the first two of them map menu commands from the items with the identifiers specified as the first macro parameter to two different member functions. In the next one, `EVT_SIZE` means that any changes in the size of the frame will result in calling `OnSize()` method. Note that this macro doesn't need a window identifier, since normally you are only interested in the current window's size events.

The `EVT_BUTTON` macro demonstrates that the originating event does not have to come from the window class implementing the event table -- if the event source is a button within a panel within a frame, this will still work, because event tables are searched up through the hierarchy of windows for the command events. (But only command events, so you can't catch mouse move events in a child control in the parent window in the same way because `wxMouseEvent` doesn't derive from `wxCommandEvent`. See below for how you can do it.) In this case, the button's event table will be searched, then the parent panel's, then the frame's.

Finally, you need to implement the event handlers. As mentioned before, all event handlers take a `wxEvent`-derived argument whose exact class differs according to the type of event and the class of the originating window. For size events, `wxSizeEvent` is used. For menu commands and most control commands (such as button presses), `wxCommandEvent` is used. When controls get more complicated, more specific `wxCommandEvent`-derived event classes providing additional control-specific information can be used, such as `wxTreeEvent` for events from `wxTreeCtrl` windows.

In the simplest possible case an event handler may not use the event parameter at all. For example,

```
void MyFrame::OnExit(wxCommandEvent& WXUNUSED(event))
{
    // when the user selects "Exit" from the menu we should close

    Close(true);
}
```

In other cases you may need some information carried by the event argument, as in:

```
void MyFrame::OnSize(wxSizeEvent& event)
{
    wxSize size = event.GetSize();
```

... update the frame using the new size ...

```
}
```

You will find the details about the event table macros and the corresponding wxEvent-derived classes in the discussion of each control generating these events.

Dynamic Event Handling

The possibilities of handling events in this way are rather different. Let us start by looking at the syntax: the first obvious difference is that you need not use `DECLARE_EVENT_TABLE()` nor `BEGIN_EVENT_TABLE()` and the associated macros. Instead, in any place in your code, but usually in the code of the class defining the handler itself (and definitely not in the global scope as with the event tables), call its `Bind<>()` method like this:

```
MyFrame::MyFrame(...)
{
    Bind(wxEVT_COMMAND_MENU_SELECTED, &MyFrame::OnExit, this, wxID_EXIT);
}
```

Note that this pointer must be specified here.

Now let us describe the semantic differences:

Event handlers can be bound at any moment. For example, it's possible to do some initialization first and only bind the handlers if and when it succeeds. This can avoid the need to test that the object was properly initialized in the event handlers themselves. With `Bind<>()` they simply won't be called if it wasn't correctly initialized.

As a slight extension of the above, the handlers can also be unbound at any time with `Unbind<>()` (and maybe rebound later). Of course, it's also possible to emulate this behaviour with the classic static (i.e., bound via event tables) handlers by using an internal flag indicating whether the handler is currently enabled and returning from it if it isn't, but using dynamically bind handlers requires less code and is also usually more clear.

Almost last but very, very far from least is the increased flexibility which allows to bind an event to:

A method in another object.

An ordinary function like a static method or a global function.

An arbitrary functor like `boost::function<>`.

This is impossible to do with the event tables because it is not possible to specify these handlers to dispatch the event to, so it necessarily needs to be sent to the same object which generated the event. Not so with `Bind<>()` which can be used to specify these handlers which will handle the event. To give a quick example, a common question is how to receive the mouse movement events happening when the mouse is in one of the frame children in the frame itself. Doing it in a naive way doesn't work:

A `EVT_LEAVE_WINDOW(MyFrame::OnMouseLeave)` line in the frame event table has no effect as mouse move (including entering and leaving) events are not propagated up to the parent window (at least not by default).

Putting the same line in a child event table will crash during run-time because the `MyFrame` method will be called on a wrong object -- it's easy to convince oneself that the only object that can be used here is the pointer to the child, as `wxWidgets` has nothing else. But calling a frame method with the child window pointer instead of the pointer to the frame is, of course, disastrous.

However writing

```
MyFrame::MyFrame(...)
{
    m_child->Bind(wxEVT_LEAVE_WINDOW, &MyFrame::OnMouseLeave, this);
}
```

will work exactly as expected. Note that you can get the object that generated the event -- and that is not the same as the frame -- via `wxEvt::GetEventObject()` method of event argument passed to the event handler.

Really last point is the consequence of the previous one: because of increased flexibility of `Bind()`, it is also safer as it is impossible to accidentally use a method of another class. Instead of run-time crashes you will get compilation errors in this case when using `Bind()`.

Let us now look at more examples of how to use different event handlers using the two overloads of `Bind()` function: first one for the object methods and the other one for arbitrary functors (callable objects, including simple functions):

In addition to using a method of the object generating the event itself, you can use a method from a completely different object as an event handler:

```
void MyFrameHandler::OnFrameExit( wxCommandEvent & )
{
    // Do something useful.
}
```

```
MyFrameHandler myFrameHandler;
```

```
MyFrame::MyFrame()
```

```

{
    Bind( wxEVT_COMMAND_MENU_SELECTED, &MyFrameHandler::OnFrameExit,
        &myFrameHandler, wxID_EXIT );
}

```

Note that MyFrameHandler doesn't need to derive from wxEvtHandler. But keep in mind that then the lifetime of myFrameHandler must be greater than that of MyFrame object -- or at least it needs to be unbound before being destroyed.

To use an ordinary function or a static method as an event handler you would write something like this:

```

void HandleExit( wxCommandEvent & )
{
    // Do something useful
}

```

```

MyFrame::MyFrame()
{
    Bind( wxEVT_COMMAND_MENU_SELECTED, &HandleExit, wxID_EXIT );
}

```

And finally you can bind to an arbitrary functor and use it as an event handler:

```

struct MyFunctor
{
    void operator()( wxCommandEvent & )
    {
        // Do something useful
    }
};

```

```

MyFunctor myFunctor;

```

```
MyFrame::MyFrame()
{
    Bind( wxEVT_COMMAND_MENU_SELECTED, &myFunctor, wxID_EXIT );
}
```

A common example of a functor is `boost::function`:

```
using namespace boost;
```

```
void MyHandler::OnExit( wxCommandEvent & )
{
    // Do something useful
}
```

```
MyHandler myHandler;
```

```
MyFrame::MyFrame()
{
    function< void ( wxCommandEvent & ) > exitHandler( bind( &MyHandler::OnExit, &myHandler, _1 ));

    Bind( wxEVT_COMMAND_MENU_SELECTED, exitHandler, wxID_EXIT );
}
```

With the aid of `boost::bind`() you can even use methods or functions which don't quite have the correct signature:

```
void MyHandler::OnExit( int exitCode, wxCommandEvent &, wxString goodByeMessage )
{
    // Do something useful
}
```



```
}
```

```
MyHandler myHandler;
```

```
MyFrame::MyFrame()
```

```
{
```

```
function< void ( wxCommandEvent & ) > exitHandler(
```

```
bind( &MyHandler::OnExit, &myHandler, EXIT_FAILURE, _1, "Bye" ));
```

```
Bind( wxEVT_COMMAND_MENU_SELECTED, exitHandler, wxID_EXIT );
```

```
}
```

To summarize, using `Bind<>()` requires slightly more typing but is much more flexible than using static event tables so don't hesitate to use it when you need this extra power. On the other hand, event tables are still perfectly fine in simple situations where this extra flexibility is not needed.

How Events are Processed

The previous sections explain how to define event handlers but don't address the question of how exactly `wxWidgets` finds the handler to call for the given event. This section describes the algorithm used in detail. Notice that you may want to run the Event Sample while reading this section and look at its code and the output when the button which can be used to test the event handlers execution order is clicked to understand it better.

When an event is received from the windowing system, `wxWidgets` calls `wxEvtHandler::ProcessEvent()` on the first event handler object belonging to the window generating the event. The normal order of event table searching by `ProcessEvent()` is as follows, with the event processing stopping as soon as a handler is found (unless the handler calls `wxEvtHandler::Skip()` in which case it doesn't count as having handled the event and the search continues):

Step 0) Before anything else happens, `wxApp::FilterEvent()` is called. If it returns anything but -1 (default), the event handling stops immediately.

Step 1) If this event handler is disabled via a call to `wxEvtHandler::SetEvtHandlerEnabled()` the next three steps are skipped and the event handler resumes at step (5).

Step 2) If the object is a `wxWindow` and has an associated validator, `wxValidator` gets a chance to process the event.

Step 3) The list of dynamically bound event handlers, i.e., those for which `Bind()` was called, is consulted. Notice that this is done before checking the static event table entries, so if both a dynamic and a static event handler match the same event, the static one is never going to be used unless `wxEvtHandler::Skip()` is called in the dynamic one.

Step 4) The event table containing all the handlers defined using the event table macros in this class and its base classes is examined. Notice that this means that any event handler defined in a base class will be executed at this step.

Step 5) The event is passed to the next event handler, if any, in the event handler chain, i.e., the steps (1) to (4) are done for it. Usually there is no next event handler so the control passes to the next step but see Event Handlers Chain for how the next handler may be defined.

Step 6) If the object is a `wxWindow` and the event is set to propagate (by default only `wxCommandEvent`-derived events are set to propagate), then the processing restarts from the step (1) (and excluding the step (7)) for the parent window. If this object is not a window but the next handler exists, the event is passed to its parent if it is a window. This ensures that in a common case of (possibly several) non-window event handlers pushed on top of a window, the event eventually reaches the window parent.

Step 7) Finally, i.e., if the event is still not processed, the `wxApp` object itself (which derives from `wxEvtHandler`) gets a last chance to process it.

Please pay close attention to step 76 People often overlook or get confused by this powerful feature of the `wxWidgets` event processing system. The details of event propagation up the window hierarchy are described in the next section.

Also please notice that there are additional steps in the event handling for the windows-making part of `wxWidgets` document-view framework, i.e., `wxDocParentFrame`, `wxDocChildFrame` and their MDI equivalents `wxDocMDIParentFrame` and `wxDocMDIChildFrame`. The parent frame classes modify step (2) above to send the events received by them to `wxDocManager` object first. This object, in turn, sends the event to the current view and the view itself lets its associated document process the event first. The child frame classes send the event directly to the associated view which still forwards it to its document object. Notice that to avoid remembering the exact order in which the events are processed in the document-view frame, the simplest, and recommended, solution is to only handle the events at the view classes level, and not in the document or document manager classes

How Events Propagate Upwards

As mentioned above, the events of the classes deriving from `wxCommandEvent` are propagated by default to the parent window if they are not processed in this window itself. But although by default only the command events are propagated like this, other events can be propagated as well because the event handling code uses `wxEvtHandler::ShouldPropagate()` to check whether an event should be propagated. It is also possible to propagate the event only a limited number of times and not until it is processed (or a top level parent window is reached).

Finally, there is another additional complication (which, in fact, simplifies life of wxWidgets programmers significantly): when propagating the command events up to the parent window, the event propagation stops when it reaches the parent dialog, if any. This means that you don't risk getting unexpected events from the dialog controls (which might be left unprocessed by the dialog itself because it doesn't care about them) when a modal dialog is popped up. The events do propagate beyond the frames, however. The rationale for this choice is that there are only a few frames in a typical application and their parent-child relation are well understood by the programmer while it may be difficult, if not impossible, to track down all the dialogs that may be popped up in a complex program (remember that some are created automatically by wxWidgets). If you need to specify a different behaviour for some reason, you can use `wxWindow::SetExtraStyle(wxWS_EX_BLOCK_EVENTS)` explicitly to prevent the events from being propagated beyond the given window or unset this flag for the dialogs that have it on by default.

Typically events that deal with a window as a window (size, motion, paint, mouse, keyboard, etc.) are sent only to the window. Events that have a higher level of meaning or are generated by the window itself (button click, menu select, tree expand, etc.) are command events and are sent up to the parent to see if it is interested in the event. More precisely, as said above, all event classes not deriving from `wxCommandEvent` (see the wxEvent inheritance map) do not propagate upward.

In some cases, it might be desired by the programmer to get a certain number of system events in a parent window, for example all key events sent to, but not used by, the native controls in a dialog. In this case, a special event handler will have to be written that will override `ProcessEvent()` in order to pass all events (or any selection of them) to the parent window.

Event Handlers Chain

The step 4 of the event propagation algorithm checks for the next handler in the event handler chain. This chain can be formed using `wxEvtHandler::SetNextHandler()`:

overview_events_chain.png

(referring to the image, if A->ProcessEvent is called and it doesn't handle the event, B->ProcessEvent will be called and so on...).

Additionally, in the case of `wxWindow` you can build a stack (implemented using `wxEvtHandler` double-linked list) using `wxWindow::PushEventHandler()`:

overview_events_winstack.png

(referring to the image, if W->ProcessEvent is called, it immediately calls A->ProcessEvent; if nor A nor B handle the event, then the `wxWindow` itself is used -- i.e. the dynamically bind event handlers and static event table entries of `wxWindow` are looked as the last possibility, after all pushed event handlers were tested).

By default the chain is empty, i.e. there is no next handler.

Custom Event Summary

General approach

As each event is uniquely defined by its event type, defining a custom event starts with defining a new event type for it. This is done using `wxDEFINE_EVENT()` macro. As an event type is a variable, it can also be declared using `wxDECLARE_EVENT()` if necessary.

The next thing to do is to decide whether you need to define a custom event class for events of this type or if you can reuse an existing class, typically either `wxEvent` (which doesn't provide any extra information) or `wxCommandEvent` (which contains several extra fields and also propagates upwards by default). Both strategies are described in details below. See also the Event Sample for a complete example of code defining and working with the custom event types.

Finally, you will need to generate and post your custom events. Generation is as simple as instantiating your custom event class and initializing its internal fields. For posting events to a certain event handler there are two possibilities: using `wxEvtHandler::AddPendingEvent` or using `wxEvtHandler::QueueEvent`. Basically you will need to use the latter when doing inter-thread communication; when you use only the main thread you can also safely use the former. Last, note that there are also two simple global wrapper functions associated to the two `wxEvtHandler` mentioned functions: `wxPostEvent()` and `wxQueueEvent()`.

Using Existing Event Classes

If you just want to use a `wxCommandEvent` with a new event type, use one of the generic event table macros listed below, without having to define a new event class yourself.

Example:

```
// this is typically in a header: it just declares MY_EVENT event type
```

```
wxDECLARE_EVENT(MY_EVENT, wxCommandEvent);
```

```
// this is a definition so can't be in a header
```

```
wxDEFINE_EVENT(MY_EVENT, wxCommandEvent);
```

```
// example of code handling the event with event tables
```

```
BEGIN_EVENT_TABLE(MyFrame, wxFrame)
```

```
    EVT_MENU (wxID_EXIT, MyFrame::OnExit)
```

```
    ...
```

```
    EVT_COMMAND (ID_MY_WINDOW, MY_EVENT, MyFrame::OnMyEvent)
```

```
END_EVENT_TABLE()
```

```
void MyFrame::OnMyEvent(wxCommandEvent& event)
```

```
{
```

```
    // do something
```

```
    wxString text = event.GetText();
```

```
}
```

```
// example of code handling the event with Bind<>():
```

```
MyFrame::MyFrame()
```

```
{
```

```
    Bind(MY_EVENT, &MyFrame::OnMyEvent, this, ID_MY_WINDOW);
```

```
}
```

```
// example of code generating the event
```

```
void MyWindow::SendEvent()
```

```
{
```

```
    wxCommandEvent event(MY_EVENT, GetId());
```

```
    event.SetEventObject(this);
```

```
    // Give it some contents
```

```
    event.SetText("Hello");
```

```
    // Do send it
```

```
    ProcessWindowEvent(event);
```

```
}
```

Defining Your Own Event Class

Under certain circumstances, you must define your own event class e.g., for sending more complex data from one place to another. Apart from defining your event class, you also need to define your own event table macro if you want to use event tables for handling events of this type.

Here is an example:

```
// define a new event class

class MyPlotEvent: public wxEvent
{
public:
    MyPlotEvent(wxEventType eventType, int winid, const wxPoint& pos)
        : wxEvent(winid, eventType),
          m_pos(pos)
    {
    }

    // accessors
    wxPoint GetPoint() const { return m_pos; }

    // implement the base class pure virtual
    virtual wxEvent *Clone() const { return new MyPlotEvent(*this); }

private:
    const wxPoint m_pos;
};

// we define a single MY_PLOT_CLICKED event type associated with the class
```

```

// above but typically you are going to have more than one event type, e.g. you
// could also have MY_PLOT_ZOOMED or MY_PLOT_PANNED &c -- in which case you
// would just add more similar lines here
wxDEFINE_EVENT(MY_PLOT_CLICKED, MyPlotEvent);

// if you want to support old compilers you need to use some ugly macros:
typedef void (wxEvtHandler::*MyPlotEventFunction)(MyPlotEvent&);
#define MyPlotEventHandler(func) wxEVENT_HANDLER_CAST(MyPlotEventFunction, func)

// if your code is only built using reasonably modern compilers, you could just
// do this instead:
#define MyPlotEventHandler(func) (&func)

// finally define a macro for creating the event table entries for the new
// event type
//
// remember that you don't need this at all if you only use Bind<>() and that
// you can replace MyPlotEventHandler(func) with just &func unless you use a
// really old compiler
#define MY_EVT_PLOT_CLICK(id, func) \
    wx__DECLARE_EVT1(MY_PLOT_CLICKED, id, MyPlotEventHandler(func))

// example of code handling the event (you will use one of these methods, not
// both, of course):

```

```
BEGIN_EVENT_TABLE(MyFrame, wxFrame)

    EVT_PLOT(ID_MY_WINDOW, MyFrame::OnPlot)

END_EVENT_TABLE()
```

```
MyFrame::MyFrame()
{
    Bind(MY_PLOT_CLICKED, &MyFrame::OnPlot, this, ID_MY_WINDOW);
}
```

```
void MyFrame::OnPlot(MyPlotEvent& event)
{
    ... do something with event.GetPoint() ...
}
```

// example of code generating the event:

```
void MyWindow::SendEvent()
{
    MyPlotEvent event(MY_PLOT_CLICKED, GetId(), wxPoint(...));
    event.SetEventObject(this);
    ProcessWindowEvent(event);
}
```

Miscellaneous Notes

Event Handlers vs Virtual Methods

It may be noted that wxWidgets' event processing system implements something close to virtual methods in normal C++ in spirit: both of these mechanisms allow you to alter the behaviour of the base class by defining the event handling functions in the derived classes.

There is however an important difference between the two mechanisms when you want to invoke the default behaviour, as implemented by the base class, from a derived class handler. With the virtual functions, you need to call the base class function directly and you can do it either in the beginning of the derived class handler function (to post-process the event) or at its end (to pre-process the event). With the event handlers, you only have the option of pre-processing the events and in order to still let the default behaviour happen you must call `wxEvent::Skip()` and not call the base class event handler directly. In fact, the event handler probably doesn't even exist in the base class as the default behaviour is often implemented in platform-specific code by the underlying toolkit or OS itself. But even if it does exist at wxWidgets level, it should never be called directly as the event handlers are not part of wxWidgets API and should never be called directly.

User Generated Events vs Programmatically Generated Events

While generically wxEvents can be generated both by user actions (e.g., resize of a wxWindow) and by calls to functions (e.g., `wxWindow::SetSize`), wxWidgets controls normally send wxCommandEvent-derived events only for the user-generated events. The only exceptions to this rule are:

`wxNotebook::AddPage`: No event-free alternatives

`wxNotebook::AdvanceSelection`: No event-free alternatives

`wxNotebook::DeletePage`: No event-free alternatives

`wxNotebook::SetSelection`: Use `wxNotebook::ChangeSelection` instead, as `wxNotebook::SetSelection` is deprecated

`wxTreeCtrl::Delete`: No event-free alternatives

`wxTreeCtrl::DeleteAllItems`: No event-free alternatives

`wxTreeCtrl::EditLabel`: No event-free alternatives

All `wxTextCtrl` methods

`wxTextCtrl::ChangeValue` can be used instead of `wxTextCtrl::SetValue` but the other functions, such as `wxTextCtrl::Replace` or `wxTextCtrl::WriteText` don't have event-free equivalents.

Pluggable Event Handlers

TODO: Probably deprecated, `Bind()` provides a better way to do this

In fact, you don't have to derive a new class from a window class if you don't want to. You can derive a new class from `wxEvtHandler` instead, defining the appropriate event table, and then call `wxWindow::SetEventHandler` (or, preferably, `wxWindow::PushEventHandler`) to make this event handler the object that responds to events. This way, you can avoid a lot of class derivation, and use instances of the same event handler class (but different objects as the same event handler object shouldn't be used more than once) to handle events from instances of different widget classes.

If you ever have to call a window's event handler manually, use the `GetEventHandler` function to retrieve the window's event handler and use that to call the member function. By default, `GetEventHandler` returns a pointer to the window itself unless an application has redirected event handling using `SetEventHandler` or `PushEventHandler`.

One use of `PushEventHandler` is to temporarily or permanently change the behaviour of the GUI. For example, you might want to invoke a dialog editor in your application that changes aspects of dialog boxes. You can grab all the input for an existing dialog box, and edit it 'in situ', before restoring its behaviour to normal. So even if the application has derived new classes to customize behaviour, your utility can indulge in a spot of body-snatching. It could be a useful technique for on-line tutorials, too, where you take a user through a series of steps and don't want them to diverge from the lesson. Here, you can examine the events coming from buttons and windows, and if acceptable, pass them through to the original event handler. Use `PushEventHandler/PopEventHandler` to form a chain of event handlers, where each handler processes a different range of events independently from the other handlers.

Window Identifiers

Window identifiers are integers, and are used to uniquely determine window identity in the event system (though you can use it for other purposes). In fact, identifiers do not need to be unique across your entire application as long they are unique within the particular context you're interested in, such as a frame and its children. You may use the `wxID_OK` identifier, for example, on any number of dialogs as long as you don't have several within the same dialog.

If you pass `wxID_ANY` to a window constructor, an identifier will be generated for you automatically by `wxWidgets`. This is useful when you don't care about the exact identifier either because you're not going to process the events from the control being created or because you process the events from all controls in one place (in which case you should specify `wxID_ANY` in the event table or `wxEvtHandler::Bind` call as well). The automatically generated identifiers are always negative and so will never conflict with the user-specified identifiers which must be always positive.

See Standard event identifiers for the list of standard identifiers available. You can use `wxID_HIGHEST` to determine the number above which it is safe to define your own identifiers. Or, you can use identifiers below `wxID_LOWEST`. Finally, you can allocate identifiers dynamically using `wxNewId()` function too. If you use `wxNewId()` consistently in your application, you can be sure that your identifiers don't conflict accidentally.

Generic Event Table Macros

<code>EVT_CUSTOM(event, id, func)</code>	Allows you to add a custom event table entry by specifying the event identifier (such as <code>wxEVT_SIZE</code>), the window identifier, and a member function to call.
<code>EVT_CUSTOM_RANGE(event, id1, id2, func)</code>	The same as <code>EVT_CUSTOM</code> , but responds to a range of window identifiers.

func)	
EVT_COMMAND(id, event, func)	The same as EVT_CUSTOM, but expects a member function with a wxCommandEvent argument.
EVT_COMMAND_RANGE(id1, id2, event, func)	The same as EVT_CUSTOM_RANGE, but expects a member function with a wxCommandEvent argument.
EVT_NOTIFY(event, id, func)	The same as EVT_CUSTOM, but expects a member function with a wxNotifyEvent argument.
EVT_NOTIFY_RANGE(event, id1, id2, func)	The same as EVT_CUSTOM_RANGE, but expects a member function with a wxNotifyEvent argument.

List of wxWidgets events

For the full list of event classes, please see the event classes group page.

2.3.2.10.2.2.4 Low-Level Widget API

The Low-Level Widget API identifies the basic Curses-/nCurses-style (Terminal Control Library) Graphical User Interface features available to the Software Engineer that supports input and output interactions with the local or remote terminal. It is used exclusively to construct, enhance and maintain the wxPython-style, High-Level Widget API emulation. It includes the following GUI objects:

- 1 **keyboard** - a Curses controlled object with alphabetic, numeric, punctuation and control buttons used by an operator to enter input.
- 2 **mouse** - a Curses controlled object with a position sensing roller ball and one or more buttons used by an operator to enter input that selects a GUI object on the display screen.
- 3 **display screen** - a Curses controlled object with multiple columns and rows that displays information output by the application for use by an operator
- 4 **window** - a Curses controlled object displayed in a designated portion of the display screen for application specific information
- 5 **pad** - a Curses controlled object like a window, except that it is not restricted by the screen size, and is not necessarily associated with a particular part of the screen
- 6 **panel** - a Curses controlled object like a window with the added feature of depth, so it can be stacked on top of another, and only the visible portions of each window will be displayed. Panels can be added, moved up or down in the stack, and removed

2.3.2.10.2.2.5 System Preferences API (Draft)

The computer software product "tsWxGTUI_PyVx" Toolkit includes a configuration file, "tsWxGlobals.py" that may be customized to determine the look and feel of "wxPython" emulation.

- 1 Symbolic Constants
- 2 Foreground and Background Colors
- 3 Default Styles and Themes

4 Unique ID Generators**5 Dimensional Unit Conversion Functions****6 Text Font and Color Attribute Markup Constants****2.3.2.10.2.3 "tsToolsGUI" Capability**

There are currently no GUI applications.

2.3.2.10.2.4 "tsTestsGUI" Capability

This section is under construction.

2.3.3 Platform Interface Capability

2.3.3.1 Keyboard Interface Capability**2.3.3.2 Mouse Interface Capability****2.3.3.3 Display Interface Capability****2.3.3.4 Event Logging Interface Capability****2.3.3.5 Process & Thread Launching & Terminating Capability****2.3.3.6 Process Scheduling Capability****2.3.3.7 Inter-Process Communication Capability**

2.3.4 Application Programming Interface Capability

2.3.4.1 Python Built-In Module API Capability (Draft)

Reference:

Python 2.x (2.7.9) Module Index

<https://docs.python.org/2.7/py-modindex.html>

Python 3.x (3.4.2) Module Index

<https://docs.python.org/3.2/py-modindex.html>

Modules (only main-level shown):

1. argparse
2. copy
3. ctypes

4. curses
5. errno
6. fcntl
7. fnmatch
8. getopt
9. imp
10. logger
11. math
12. optprse
13. os
14. platform
15. shlex
16. shutil
17. signal
18. stat
19. struct
20. subprocess
21. syslog
22. system
23. termios
24. textwrap
25. thread
26. threading
27. time
28. traceback

2.3.4.2 High-Level GUI "wxPython"-style API Capability (Draft)

The pixel-mode wxPython API represents a blending of the wxWidgets C++ class library with the Python programming language.

Version	Description
2.8.12 (Second generation)	<p>The "tsWxGTUI_PyVx" Toolkit creates a character-mode emulation of a subset of this wxPython pixel-mode version.</p> <p>http://docs.wxwidgets.org/2.8/</p> <ul style="list-style-type: none">▪ wxWidgets 2.8.12: A portable C++ and Python GUI toolkit▪ Julian Smart, Robert Roebling, Vadim Zeitlin, Robin Dunn, et al▪ March, 2011
3.0.0 (Third generation)	<p>http://docs.wxwidgets.org/3.0/</p> <ul style="list-style-type: none">▪ wxWidgets 3.0.0: A portable C++ and Python GUI toolkit▪ Julian Smart, Robert Roebling, Vadim Zeitlin, Robin Dunn, et al▪ November 11, 2013

Excerpt from /tsLibGUI/tsWxPkg/src/tsWx.py:

Identifies currently available components of the emulated wxPython API.

```

try:

    if ReadyForIntegration:

        # Release candidates suitable for integration with applications.
        # Troubleshooting will be required before uncommenting.

        # Prototypes suitable for integration with applications.

        from tsWxGlobals import *

        Debug_CLI_Configuration = Troubleshooting['Debug_CLI_Configuration'],
        Debug_CLI_Exceptions = Troubleshooting['Debug_CLI_Exceptions']
        Debug_CLI_Launch = Troubleshooting['Debug_CLI_Launch']
        Debug_CLI_Progress = Troubleshooting['Debug_CLI_Progress']
        Debug_CLI_Termination = Troubleshooting['Debug_CLI_Termination']
        Debug_GUI_Configuration = Troubleshooting['Debug_GUI_Configuration']
        Debug_GUI_Exceptions = Troubleshooting['Debug_GUI_Exceptions']
        Debug_GUI_Launch = Troubleshooting['Debug_GUI_Launch']
        Debug_GUI_Progress = Troubleshooting['Debug_GUI_Progress']
        Debug_GUI_Termination = Troubleshooting['Debug_GUI_Termination']

        from tsWxColor import Color
        from tsWxColorDatabase import ColorDatabase
        from tsWxApp import App
        from tsWxButton import Button
        from tsWxFrameButton import FrameButton
        from tsWxDialogButton import DialogButton
        from tsWxCheckBox import CheckBox
        from tsWxControl import Control
        from tsWxDialog import Dialog
        from tsWxFrame import Frame
        from tsWxNonLinkedList import NonLinkedList
        from tsWxScreen import Screen
        from tsWxSplashScreen import SplashScreen
        from tsWxGauge import Gauge
        from tsWxMenu import Menu
        from tsWxMenuBar import MenuBar
        from tsWxKeyboardState import KeyboardState
        from tsWxMouseState import MouseState
        from tsWxTextEditText import TextEditText
        from tsWxTextEntryDialog import TextEntryDialog
        from tsWxPasswordEntryDialog import PasswordEntryDialog
##         from tsWxDebugHandlers import DebugHandlers
        from tsWxObject import Object
        from tsWxPanel import Panel
        from tsWxPoint import Point
        from tsWxPyApp import PyApp
        from tsWxPyEventBinder import PyEventBinder
        from tsWxPyOnDemandOutputWindow import PyOnDemandOutputWindow
        from tsWxPySimpleApp import PySimpleApp
        # from tsWxCommandLineEnv import CommandLineEnv
        from tsWxPySizer import PySizer
        from tsWxRadioBox import RadioBox
        from tsWxRadioButton import RadioButton

```

```
from tsWxRect import Rect
from tsWxScrollBarButton import ScrollBarButton
from tsWxScrollBarGauge import ScrollBarGauge
from tsWxScrollBar import ScrollBar
from tsWxScrolledText import ScrolledText
from tsWxScrolled import Scrolled
from tsWxScrolledWindow import ScrolledWindow
from tsWxSize import Size
from tsWxSizer import Sizer
from tsWxSizerSpacer import SizerSpacer
from tsWxSizerItemList import SizerItemList
from tsWxSizerItem import SizerItem
from tsWxStaticLine import StaticLine
from tsWxStaticText import StaticText
from tsWxGridSizer import GridSizer
from tsWxStatusBar import StatusBar
from tsWxSystemSettings import SystemSettings
from tsWxTextCtrl import TextCtrl
from tsWxTimer import Timer
from tsWxTopLevelWindow import TopLevelWindow
from tsWxWindow import Window
from tsWxEvent import *
from tsWxStaticBox import StaticBox
import tsWxKeyEvent as KeyEvent
import tsWxMouseEvent as MouseEvent
import tsWxEvtHandler as tsEvtHandler
import tsWxShowEvent as tsShowEvent
from tsWxAcceleratorEntry import AcceleratorEntry
from tsWxAcceleratorTable import AcceleratorTable
from tsWxCaret import Caret
from tsWxDisplay import Display
from tsWxEventLoop import EventLoop
from tsWxEventLoopActivator import EventLoopActivator
from tsWxFocusEvent import FocusEvent
from tsWxValidator import Validator
from tsWxEventDaemon import EventDaemon
from tsWxEventQueueEntry import EventQueueEntry
from tsWxEventTableEntry import EventTableEntry
#from tsWxEvent import Event
#from tsWxEvent import PyEventBinder
#from tsWxEvtHandler import EvtHandler
##      from tsWxMouseEvent import MouseEvent
##      from tsWxKeyEvent import KeyEvent
from tsWxBoxSizer import BoxSizer
from tsWxCursor import Cursor
from tsWxStaticBoxSizer import StaticBoxSizer

except ImportError, importCode:

    print('tsWx: ImportError (tsLibGUI); ' + \
          'importCode=%s' % str(importCode))

except Exception, exceptionCode:

    print('tsWx: Exception (tsLibGUI); ' + \
          'exceptionCode=%s' % str(exceptionCode))
```


2.3.4.3 Low-Level GUI "nCurses"-style API Capability (Draft)

Low Level GUI "Curses"-style API Classes and associated Class Methods:

- **Pads** - A pad is like a window, except that it is not restricted by the screen size, and is not necessarily associated with a particular part of the screen. Pads can be used when a large window is needed, and only a part of the window will be on the screen at one time. Automatic refreshes of pads (such as from scrolling or echoing of input) do not occur. The refresh() and noutrefresh() methods of a pad require 6 arguments to specify the part of the pad to be displayed and the location on the screen to be used for the display. The arguments are pminrow, pmincol, sminrow, smincol, smaxrow, smaxcol; the p arguments refer to the upper left corner of the pad region to be displayed and the s arguments define a clipping box on the screen within which the pad region is to be displayed.
- **Panels** - Panels are windows with the added feature of depth, so they can be stacked on top of each other, and only the visible portions of each window will be displayed. Panels can be added, moved up or down in the stack, and removed.
- **Windows** - a window is a rectangular area of the display with or without a border
- **Text** - A character string of one or more mono-spaced alpha-numeric, punctuation or line-draw characters.
- **Colors** - Support for terminals and terminal emulators (cygwin, linux, xterm, xterm-color, xterm-16color, xterm-88color or xterm-256color) with mouse and Red-Green-Blue color phosphors who's individual intensities can be adjusted to create a palette of terminal / terminal emulator dependent colors, ranging from 8 to 256 used to create foreground/background color pair combinations ranging from 8 x 8 to 256 x 256. However, currently the maximum number of color pairs is limited to 16 x 16.
- **Non-color** - Support for Digital Equipment Corporation VT100 and VT220 terminals and terminal emulators with/without mouse having only a single color phosphor (such as green, orange or white) who's intensity can be either ON or OFF.

Reference:

Python 2.x (2.7.6) Curses Module

<https://docs.python.org/2.7/library/curses.html#module-curses>

Python 3.x (3.2.5) Curses Module

<https://docs.python.org/3.2/library/curses.html#module-curses>

Excerpt from Python 2.7.6 Global Module Index for Curses Module:

"The curses library supplies a terminal-independent screen-painting and keyboard-handling facility for text-based terminals; such terminals include VT100s, the Linux console, and the simulated terminal provided by X11 programs such as xterm and rxvt. Display terminals support various control codes to perform common operations such as moving the cursor, scrolling the screen, and erasing areas. Different terminals use widely differing codes, and often have their own minor quirks.

In a world of X displays, one might ask "why bother"? It's true that character-cell display terminals are an obsolete technology, but there are niches in which being able to do fancy things with them are still valuable. One is on small-footprint or embedded Unixes that don't carry an X server. Another is for tools like OS installers and kernel configurators that may have to run before X is available.

The curses library hides all the details of different terminals, and provides the programmer with an abstraction of a display, containing multiple non-overlapping windows. The contents of a window can be changed in various ways— adding text, erasing it, changing its appearance—and the curses library will automatically figure out what control codes need to be sent to the terminal to produce the right output.

The curses library was originally written for BSD Unix; the later System V versions of Unix from AT&T added many enhancements and new functions. BSD curses is no longer maintained, having been replaced by ncurses, which is an open-source implementation of the AT&T interface. If you're using an open-source Unix such as Linux or FreeBSD, your system almost certainly uses ncurses. Since most current commercial Unix versions are based on System V code, all the functions described here will probably be available. The older versions of curses carried by some proprietary Unixes may not support everything, though.

No one has made a Windows port of the curses module. On a Windows platform, try the Console module written by Fredrik Lundh. The Console module provides cursor-addressable text output, plus full support for mouse and keyboard input, and is available from <http://effbot.org/zone/console-index.htm>."

Users of various Microsoft Windows distributions (8.1, 8, 7, Vista and XP) may install "Cygwin", a free, Unix-like environment and command-line interface add-on from Red Hat. "Cygwin" provides native integration of Windows-based applications, data, and other system resources with applications, software tools, and data of the Unix-like environment. The running "Cygwin" Command Line Interface includes a fully functional version of ncurses that enables Python applications, and the "tsWxGTUI_PyVx" Toolkit, to use the Python curses module.

2.3.5 Operator Interface Capability

2.3.5.1 Command Line Interface (CLI) Capability

See *Command Line Interface Capability* (see "*tsToolkitCLI*" *Command Line Interface Capability*" on page 213)

2.3.5.2 Graphical-style User Interface (GUI) Capability

See *Graphical-style User Interface Capability* (see "*tsToolkitGUI*" *Graphical-style User Interface Capability*" on page 216)

Draft

Draft

2.4 APPENDIX C - SYSTEM EXTERNAL INTERFACE REQUIREMENTS

This paragraph shall be divided into subparagraphs to specify the requirements, if any, for the system's external interfaces. This paragraph may reference one or more Interface Requirements Specifications (IRSSs) or other documents containing these requirements.

The system hardware and software shall provide the means for interfacing the "tsWxGTUI_PyVx" Toolkit, and those application programs created with it, with different types of computer hardware and/or with different software packages:

- 1 *Hardware Components* (on page 255)
- 2 *Software Components* (on page 264)
- 3 *Architecture* (on page 267)

2.4.1 (Project-unique Identifier Of Interface Template)

This paragraph (beginning with 3.3.2) shall identify a system external interface by project-unique identifier, shall briefly identify the interfacing entities, and shall be divided into subparagraphs as needed to state the requirements imposed on the system to achieve the interface. Interface characteristics of the other entities involved in the interface shall be stated as assumptions or as "When [the entity not covered] does this, the system shall...", not as requirements on the other entities. This paragraph may reference other documents (such as data dictionaries, standards for communication protocols, and standards for user interfaces) in place of stating the information here. The requirements shall include the following, as applicable, presented in any order suited to the requirements, and shall note any differences in these characteristics from the point of view of the interfacing entities (such as different expectations about the size, frequency, or other characteristics of data elements):

- a. Priority that the system must assign the interface
- b. Requirements on the type of interface (such as real-time data transfer, storage-and-retrieval of data, etc.) to be implemented
- c. Required characteristics of individual data elements that the system must provide, store, send, access, receive, etc., such as:
 - 1) Names/identifiers
 - a) Project-unique identifier
 - b) Non-technical (natural-language) name
 - c) DoD standard data element name
 - d) Technical name (e.g., variable or field name in code or database)
 - e) Abbreviation or synonymous names
 - 2) Data type (alphanumeric, integer, etc.)
 - 3) Size and format (such as length and punctuation of a character string)
 - 4) Units of measurement (such as meters, dollars, nanoseconds)
 - 5) Range or enumeration of possible values (such as 0-99)
 - 6) Accuracy (how correct) and precision (number of significant digits)
 - 7) Priority, timing, frequency, volume, sequencing, and other constraints, such as whether the data element may be updated and whether business rules apply

- 8) Security and privacy constraints
- 9) Sources (setting/sending entities) and recipients (using/receiving entities)

d. Required characteristics of data element assemblies (records, messages, files, arrays, displays, reports, etc.) that the system must provide, store, send, access, receive, etc., such as:

- 1) Names/identifiers
 - a) Project-unique identifier
 - b) Non-technical (natural language) name
 - c) Technical name (e.g., record or data structure name in code or database)
 - d) Abbreviations or synonymous names
- 2) Data elements in the assembly and their structure (number, order, grouping)
- 3) Medium (such as disk) and structure of data elements/assemblies on the medium
- 4) Visual and auditory characteristics of displays and other outputs (such as colors, layouts, fonts, icons and other display elements, beeps, lights)
- 5) Relationships among assemblies, such as sorting/access characteristics
- 6) Priority, timing, frequency, volume, sequencing, and other constraints, such as whether the assembly may be updated and whether business rules apply
- 7) Security and privacy constraints
- 8) Sources (setting/sending entities) and recipients (using/receiving entities)

e. Required characteristics of communication methods that the system must use for the interface, such as:

- 1) Project-unique identifier(s)
- 2) Communication links/bands/frequencies/media and their characteristics
- 3) Message formatting
- 4) Flow control (such as sequence numbering and buffer allocation)
- 5) Data transfer rate, whether periodic/aperiodic, and interval between transfers

- 6) Routing, addressing, and naming conventions
 - 7) Transmission services, including priority and grade
 - 8) Safety/security/privacy considerations, such as encryption, user authentication, compartmentalization, and auditing

- f. Required characteristics of protocols the system must use for the interface, such as:
 - 1) Project-unique identifier(s)
 - 2) Priority/layer of the protocol
 - 3) Packeting, including fragmentation and reassembly, routing, and addressing
 - 4) Legality checks, error control, and recovery procedures
 - 5) Synchronization, including connection establishment, maintenance, termination
 - 6) Status, identification, and any other reporting features

- g. Other required characteristics, such as physical compatibility of the interfacing entities (dimensions, tolerances, loads, plug compatibility, etc.), voltages, etc.

2.4.2 Interface Identification and Diagrams

This paragraph shall identify the required external interfaces of the system. The identification of each interface shall include a project-unique identifier and shall designate the interfacing entities (systems, configuration items, users, etc.) by name, number, version, and documentation references, as applicable. The identification shall state which entities have fixed interface characteristics (and therefore impose interface requirements on interfacing entities) and which are being developed or modified (thus having interface requirements imposed on them). One or more interface diagrams shall be provided to depict the interfaces.

2.4.2.1 Hardware Components

The hardware components include one or more of the following:

- **Computer Processor, Memory, Storage and Network Devices** (on page 255) - Identifies hardware components required for local and remote computer processing of commands and data from System Operator or Software Engineer.
- **Keyboard, Mouse, Trackball and Touchpad Devices** (on page 255) - Describes hardware components required for appropriate input of commands and data by System Operator or Software Engineer.
- **Display and Printer Devices** (see "**Display and Printer Output Devices**" on page 257) - Describes hardware components required for reporting of appropriate output of command responses and data to System Operator or Software Engineer.

2.4.2.1.1 Computer Processor, Memory, Storage and Network Devices

The hardware components include one or more of the following:

- **Central Processing Unit** - A required electronic device that loads and executes machine-readable instructions. It performs arithmetic and logic. It loads, modifies and stores machine-readable data. The processor may be any of the popular 32-/64-bit single or multi-core devices.
- **Random Access Memory** - A required electronic device that temporarily receives, stores and returns machine-readable machine instructions and data.
- **Non-Volatile Storage** - A required internal and optional external electronic device (Flash Memory) or electro-mechanical device (Hard Drive) whose non-volatile memory receives, stores and can then return source and machine-readable data after electrical power interruptions. The optional external device may be local or remote. When remote, the system must include a Network Interface Device.
- **Network Interface Device** - An optional electronic device (Modem) that inputs and outputs data over an optional wired or wireless telecommunications circuit.

2.4.2.1.2 Keyboard, Mouse, Trackball and Touchpad Devices

NOTE: Pointing Device input is only available with "cygwin X11 shell", "Cygwin Mintty shell", "xterm", "xterm-color" and "xterm-256color" consoles or terminal emulators. It is not available with "vt100", "vt220" and "ansi" consoles or terminal emulators.

The hardware components include one or more of the following:

- **Keyboard Device** - A required hand-operated electronic or electro-mechanical device to input alpha-numeric and control data via push buttons.
- **Pointing Device** - An optional hand-operated electronic or electro-mechanical Mouse, Trackball, Touchpad or Touchscreen device that controls the coordinates of a cursor on the computer screen as you move the positioning control (hand, finger or stylus) around.

Terminal Type	User Input Source
Cygwin Bash Shell	Keyboard
Cygwin Mintty	Keyboard & Pointing Device
Cygwin X11 Desktop	Keyboard & Pointing Device
Git Bash Shell	Keyboard

Linux Terminal	Keyboard & Pointing Device
Mac OS X iTerm / iTerm2	Keyboard & Pointing Device
Mac OS X Terminal	Keyboard
Microsoft Windows PowerShell and Command Prompt shell	Keyboard
vt100 (non-color)	Keyboard
vt220 (non-color)	Keyboard
xterm (8-color)	Keyboard & Pointing Device
xterm-color (obsolete)	Keyboard & Pointing Device
xterm-16color	Keyboard & Pointing Device
xterm-88color	Keyboard & Pointing Device
xterm-256color	Keyboard & Pointing Device

2.4.2.1.3 Display and Printer Output Devices

NOTES:

- 1) The *TeamSTARS* "tsWxGTUI_PyVx" Toolkit depends on the user to activate and configure the display as appropriate for the user's application.
- 2) For "vt100" and "vt220" consoles and terminal emulators, the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit supports 'white' on 'black' output for the cygwin console and 'black' on 'white' output for the xterm console.
- 3) For "cygwin", "linux", "xterm" and "xterm-color" consoles and terminal emulators, the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit maps the standard 68-color "wxPython" palette to the standard 8-color "curses" palette ('black', 'blue', 'cyan', 'green', 'magenta', 'red', 'white', 'yellow').
- 4) For "linux", "xterm-16color" and "xterm-88color" and "xterm-256color" consoles and terminal emulators, the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit maps the optional 71-color "wxPython" palette to the standard 16-color "curses" palette ('black', 'blue', 'cyan', 'gray', 'green', 'lime green', 'magenta', 'maroon', 'navy', 'olive', 'purple', 'red', 'silver', 'teal', 'white', 'yellow').
- 5) For "xterm-88color" consoles and terminal emulators, the "tsWxGTUI_PyVx" Toolkit constructs the optional 71-color "wxPython" palette.
- 6) For "xterm-256color" consoles and terminal emulators, the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit constructs the optional 140-color "wxPython" palette.

The hardware components include one or more of the following:

- **Display Device** - A required electronic device that outputs alpha-numeric, graphic and control data to the computer screen.
- **Printer Device** - An optional electro-mechanical device that outputs alpha-numeric, graphic and control data to individual sheets of paper.

See the following for configuration options:

- ***Character & Pixel Screen Size*** (on page 258)
- ***Standard 1-color Monochrome palette*** (on page 264)
- ***Standard 8-color "Curses" palette*** (on page 261)
- Standard 16-color "Curses" palette
- ***Standard 68-color "wxPython" palette*** (on page 258)
- Optional 71-color "wxPython" palette
- Optional 140-color "wxPython" palette

2.4.2.1.3.1 Character & Pixel Screen Size

The "tsWxGTUI-Toolkit" supports monochrome vt100 and color cygwin, xterm, xterm-color and xterm-256color consoles with the following screen sizes:

Terminal	Pixel Display Terminal Graphical (Font: 8x12)	Character Display Terminal Non- Graphical
minimum	272 col x 108 row	34 col x 9 row
CGA	320 col x 200 row	40 col x 16 row
VT100	640 col x 288 row	40 col x 24 row
VGA	640 col x 480 row	80 col x 40 row
SVGA	800 col x 600 row	100 col x 50 row
XGA	1024 col x 768 row	128 col x 64 row
MAC	1152 col x 870 row	144 col x 72 row
SXGA	1280 col x 1024 row	160 col x 85 row
WXGA	1366 col x 768 row	170 col x 64 row
SXGA+	1400 col x 1050 row	175 col x 87 row
UXGA	1600 col x 1200 row	200 col x 100 row
WSXGA	1680 col x 1050 row	210 col x 87 row
UXGA	1680 col x 1200 row	210 col x 100 row
WUXGA	1920 col x 1200 row	240 col x 100 row
OXGA	2048 col x 1600 row	256 col x 133 row

NOTES:

1) The application developer may have to re-compile and rebuild a platform's "ncurses" or "curses" library with the wide character option in order to support anything other than the standard 8-color or optional 16-color "curses" palette.

2) The operator typically has control of the actual font. Smaller fonts enable the display of more information on a standard size terminal. The tabulation only depicts the display size associated with a 8x12 Font.

2.4.2.1.3.2 Standard 68-color "wxPython" palette

For 8-color or 64-color pair limited "cygwin", "linux", "xterm" and "xterm-color" compatible displays, the TeamSTARS "tsWxGTUI_PyVx" Toolkit supports the standard 68-color "wxPython" palette:

1 'aquamarine'

- 2** 'black'
- 3** 'blue'
- 4** 'blue violet'
- 5** 'brown'
- 6** 'cadet blue'
- 7** 'coral'
- 8** 'cornflower blue'
- 9** 'cyan'
- 10** 'dark gray'
- 11** 'dark green'
- 12** 'dark olive green'
- 13** 'dark orchid'
- 14** 'dark slate blue'
- 15** 'dark slate gray'
- 16** 'dark turquoise'
- 17** 'dim gray'
- 18** 'firebrick'
- 19** 'forest green'
- 20** 'gold'
- 21** 'goldenrod'
- 22** 'gray'
- 23** 'green'
- 24** 'green yellow'
- 25** 'indian red'
- 26** 'khaki'
- 27** 'light blue'
- 28** 'light gray'
- 29** 'light steel blue'
- 30** 'lime green'
- 31** 'magenta'
- 32** 'maroon'
- 33** 'medium aquamarine'
- 34** 'medium blue'
- 35** 'medium forest green'

- 36** 'medium goldenrod'
- 37** 'medium orchid'
- 38** 'medium sea green'
- 39** 'medium slate blue'
- 40** 'medium spring green'
- 41** 'medium turquoise'
- 42** 'medium violet red'
- 43** 'midnight blue'
- 44** 'navy'
- 45** 'orange'
- 46** 'orange red'
- 47** 'orchid'
- 48** 'pale green'
- 49** 'pink'
- 50** 'plum'
- 51** 'purple'
- 52** 'red'
- 53** 'salmon'
- 54** 'sea green'
- 55** 'sienna'
- 56** 'sky blue'
- 57** 'slate blue'
- 58** 'spring green'
- 59** 'steel blue'
- 60** 'tan'
- 61** 'thistle'
- 62** 'turquoise'
- 63** 'violet'
- 64** 'violet red'
- 65** 'wheat'
- 66** 'white'
- 67** 'yellow'
- 68** 'yellow green'

NOTES:

- 1) Python uses the "curses" or "ncurses" library with its standard 8-color palette.
- 2) Discovered the following with Google Search for "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, ... www.c-for-dummies.com/ncurses/256color/"

2.4.2.1.3.3 Standard 8-color "Curses" palette

For "cygwin", "linux", "xterm" and "xterm-color" consoles and terminal emulators, the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit maps the standard 68-color "wxPython" palette to the standard 8-color "curses" palette:

- 1 'black'
- 2 'blue'
- 3 'cyan'
- 4 'green'
- 5 'magenta'
- 6 'red'
- 7 'white'
- 8 'yellow'

Color8SubstitutionMap:

- 1 COLOR_AQUAMARINE: COLOR_CYAN,
- 2 COLOR_BLACK: COLOR_BLACK,
- 3 COLOR_BLUE: COLOR_BLUE,
- 4 COLOR_BLUE_VIOLET: COLOR_BLUE,
- 5 COLOR_BROWN: COLOR_YELLOW,
- 6 COLOR_CADET_BLUE: COLOR_BLUE,
- 7 COLOR_CORAL: COLOR_RED,
- 8 COLOR_CORNFLOWER_BLUE: COLOR_BLUE,
- 9 COLOR_CYAN: COLOR_CYAN,
- 10 COLOR_DARK_GRAY: COLOR_BLACK,
- 11 COLOR_DARK_GREEN: COLOR_GREEN,
- 12 COLOR_DARK_OLIVE_GREEN: COLOR_GREEN,
- 13 COLOR_DARK_ORCHID: COLOR_MAGENTA,
- 14 COLOR_DARK_SLATE_BLUE: COLOR_BLUE,
- 15 COLOR_DARK_SLATE_GRAY: COLOR_BLACK,

- 16** COLOR_DARK_TURQUOISE: COLOR_CYAN,
- 17** COLOR_DIM_GRAY: COLOR_BLACK,
- 18** COLOR_FIREBRICK: COLOR_RED,
- 19** COLOR_FOREST_GREEN: COLOR_GREEN,
- 20** COLOR_GOLD: COLOR_YELLOW,
- 21** COLOR_GOLDENROD: COLOR_YELLOW,
- 22** COLOR_GRAY: COLOR_BLACK,
- 23** COLOR_GREEN: COLOR_GREEN,
- 24** COLOR_GREEN_YELLOW: COLOR_GREEN,
- 25** COLOR_INDIAN_RED: COLOR_RED,
- 26** COLOR_KHAKI: COLOR_YELLOW,
- 27** COLOR_LIGHT_BLUE: COLOR_BLUE,
- 28** COLOR_LIGHT_GRAY: COLOR_BLACK,
- 29** COLOR_LIGHT_STEEL_BLUE: COLOR_BLUE,
- 30** COLOR_LIME_GREEN: COLOR_GREEN,
- 31** COLOR_MAGENTA: COLOR_MAGENTA,
- 32** COLOR_MAROON: COLOR_RED,
- 33** COLOR_MEDIUM_AQUAMARINE: COLOR_CYAN,
- 34** COLOR_MEDIUM_BLUE: COLOR_BLUE,
- 35** COLOR_MEDIUM_FOREST_GREEN: COLOR_GREEN,
- 36** COLOR_MEDIUM_GOLDENROD: COLOR_YELLOW,
- 37** COLOR_MEDIUM_ORCHID: COLOR_MAGENTA,
- 38** COLOR_MEDIUM_SEA_GREEN: COLOR_GREEN,
- 39** COLOR_MEDIUM_SLATE_BLUE: COLOR_BLUE,
- 40** COLOR_MEDIUM_SPRING_GREEN: COLOR_GREEN,
- 41** COLOR_MEDIUM_TURQUOISE: COLOR_CYAN,
- 42** COLOR_MEDIUM_VIOLET_RED: COLOR_RED,
- 43** COLOR_MIDNIGHT_BLUE: COLOR_BLUE,
- 44** COLOR_NAVY: COLOR_BLUE,
- 45** COLOR_OLIVE: COLOR_GREEN,
- 46** COLOR_ORANGE: COLOR_RED,
- 47** COLOR_ORANGE_RED: COLOR_RED,
- 48** COLOR_ORCHID: COLOR_MAGENTA,
- 49** COLOR_PALE_GREEN: COLOR_GREEN,

50 COLOR_PINK: COLOR_RED,
51 COLOR_PLUM: COLOR_MAGENTA,
52 COLOR_PURPLE: COLOR_RED,
53 COLOR_RED: COLOR_RED,
54 COLOR_SALMON: COLOR_RED,
55 COLOR_SEA_GREEN: COLOR_GREEN,
56 COLOR_SIENNA: COLOR_RED,
57 COLOR_SILVER: COLOR_WHITE,
58 COLOR_SKY_BLUE: COLOR_CYAN,
59 COLOR_SLATE_BLUE: COLOR_BLUE,
60 COLOR_SPRING_GREEN: COLOR_GREEN,
61 COLOR_STEEL_BLUE: COLOR_BLUE,
62 COLOR_TAN: COLOR_YELLOW,
63 COLOR_TEAL: COLOR_CYAN,
64 COLOR_THISTLE: COLOR_BLACK,
65 COLOR_TURQUOISE: COLOR_CYAN,
66 COLOR_VIOLET: COLOR_MAGENTA,
67 COLOR_VIOLET_RED: COLOR_RED,
68 COLOR_WHEAT: COLOR_YELLOW,
69 COLOR_WHITE: COLOR_WHITE,
70 COLOR_YELLOW: COLOR_YELLOW,
71 COLOR_YELLOW_GREEN: COLOR_YELLOW

2.4.2.1.3.4 Standard 1-color Monochrome palette

From http://en.wikipedia.org/wiki/Monochrome_monitor:

A monochrome monitor is a type of computer display which was very common in the early days of computing, from the 1960s through the 1980s, before color monitors became popular. They are still widely used in applications such as computerized cash register systems. Unlike color monitors, which display text and graphics in multiple colors through the use of alternating-intensity red, green, and blue phosphors, monochrome monitors have only one color of phosphor (mono means "one", and chrome means "color"). All text and graphics are displayed in that color. Some monitors have the ability to vary the brightness of individual pixels, thereby creating the illusion of depth and color, exactly like a black-and-white television.

Monochrome monitors are commonly available in three colors: if the P1 phosphor is used, the screen is green monochrome. If the P3 phosphor is used, the screen is amber monochrome. If the P4 phosphor is used, the screen is white monochrome (known as "page white"); this is the same phosphor as used in early television sets. An amber screen was claimed to give improved ergonomics, specifically by reducing eye strain; this claim appears to have little scientific basis.[1]

For "vt100" and "vt220" consoles and terminal emulators, the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit supports 'white' (for which the host operating system may substitute 'green', 'orange' or another color) on 'black' output for the cygwin console and 'black' on 'white' output for the xterm console displays. The *TeamSTARS* "tsWxGTUI_PyVx" Toolkit maps the standard 68-color "wxPython" palette to the standard 1-color "monochrome" palette:

- 1 'black' (color Off)
- 2 'white' (color On)

2.4.2.2 Software Components

The software components include the following:

- Host computer operating system and its associated editing, debugging and terminal emulator applications.
- Python Virtual Machine and its associated Python source code compiler and byte code interpreter.
- The "tsWxGTUI_PyVx" Toolkit and its associated library of general purpose, re-usable utility modules.

The "tsWxGTUI_PyVx" Toolkit library emulates a subset of the Application Programming Interface (API) of the C++ language based "wxWidgets" Graphical User Interface Toolkit and its Python language based "wxPython" wrapper.

Whereas "wxWidgets" and "wxPython" support local pixel-mode terminals, the "tsWxGTUI Toolkit""tsWxGTUI_PyVx" Toolkit is designed for use by developers of Python language based applications that will run on Unix-like platforms having local or remote character-mode terminals. It enables a developer to do the following:

- 1 Port "wxPython" applications or create terminal-independent ones that can run in the local or remote console session of a computer running:

- a) Linux (such as Debian GNU, Fedora, Gentoo, Hard Hat, Mandriva, Red Hat, SuSE, Ubuntu, Yellow Dog)
 - b) Unix (such as BSD/OS, FreeBSD, HP-UX, IBM-AIX, IRIX, Mac OS X, SCO, Solaris)
 - c) Windows (such as Windows 8 Professional, 7 Professional, Vista Professional, XP Professional, 2000 Professional) when augmented with Cygwin, the free Linux-like environment for 32-bit/64-bit Windows
- 2** Interface, via a platform's "ncurses" or "curses" programming library, with an operator selected and configured terminal emulator:
- a) Cygwin console
 - b) VT100
 - c) Xterm
 - d) Xterm-color
 - e) Xterm-256color (*Note: Only 8-color palette is available. Availability of the 256-color palette is beyond the scope of the "tsWxGTUI_PyVx" developer. It requires that each system administrator assume the responsibility for re-compiling, building, installing and trouble shooting the wide character version of "ncurses" and its Python wrapper.*)
- 3** Display multiple rows and columns of text and a limited set of box drawing, line drawing and pseudo graphics characters in color or black and white (with normal, bold, reverse, standout and blinking attributes) in one or more windows on a two-dimensional console screen. The color palette depends on the operator selected terminal emulator:
- a) Xterm-256color terminals support the standard 68-color "wxPython" palette.
 - b) Xterm-color terminals support the standard 8-color curses palette which will be substituted for application references to the standard 68-color "wxPython" palette.
 - c) Xterm terminals support the standard 8-color curses palette which will be substituted for application references to the standard 68-color "wxPython" palette.
 - d) VT100 terminal support a single phosphor dependent color (such as green, amber or white on black).
 - e) Cygwin terminals support the standard 8-color curses palette which will be substituted for application references to the standard 68-color "wxPython" palette.
- 4** Accept and interpret operator input from:
- a) console keyboard
 - b) pointing device (mouse, trackball, touchpad or touchscreen)

2.4.2.2.1 Operating System

Platform-specific, multi-user (enables a user to login locally and the same or another user to login remotely), multi-process (enables a user to launch multiple applications and/or multiple instances of the same application), multi-threaded (enables each application to concurrently execute multiple tasks which may independently block while waiting for a triggering event notification) operating systems for 32-bit/64-bit processors:

- 1** Linux (such as Debian GNU, Fedora, Gentoo, Hard Hat, Mandriva, Red Hat, Scientific / Centos, SuSE, Ubuntu, Yellow Dog)

- 2** Mac OS X (Unix-like, Darwin-based such as 10.4/Tiger - 10.9/Mavericks)
- 3** Microsoft Windows (such as Windows 8.1 Professional, 8 Professional, 7 Professional, Vista Professional, XP Professional) when augmented with Cygwin, the free Linux-like Command Line Interface and GNU toolkit from Red Hat
- 4** Unix (such as FreeBSD/PC-BSD, HP-UX, IBM-AIX, IRIX, OpenIndiana, SCO, Solaris/SunOS)

Associated with each operating system are the following platform-specific components:

- Kernels
- Device Drivers
- Run Time Libraries
- Terminal Emulators
- Command Line Interface
- Graphical User Interface
- Network Interface

2.4.2.2.2 Python

Python technology makes the following contributions to the "tsWxGTUI_PyVx" Toolkit:

- 1** Enables the embedded system application programmer to design processor-independent applications that run, with little or no modification, on platforms with 32-bit and 64-bit processor architectures under "Linux", "Mac OS X", "MS Windows" (with UNIX-style "Cygwin" plug-in) and "UNIX" type operating systems.
- 2** Enables the embedded system application programmer to design terminal-independent applications that run, with little or no modification, on platforms with the pixel-mode and character-mode terminals and terminal emulators available on the local and remote computer platforms.
- 3** Python's "curses" module interfaces to the low-level, character-mode, "nCurses" GUI-toolkit, the de-facto standard for portable advanced terminal handling.
- 4** Converts positioning and sizing dimensions between the pixel-mode units used by "wxPython" applications and the character units used by "nCurses".
- 5** Converts wxPython-style color palettes and font attributes to their "nCurses" equivalents.

Cross-platform Python installations include:

1 Python 2.x

The popular, high-level, cross-platform, second generation Python 2.x programming language which is widely available.

A wiki posting by the Python Software Foundation announced that Python 3.0 was released in 2008. The final 2.x version 2.7 release came out in mid-2010, with a statement of extended support for this end-of-life release.

The 2.x branch will see no new major releases after that.

2 Python 3.x

The popular, high-level, cross-platform, third generation Python 3.x programming language which is growing in availability.

A wiki posting by the Python Software Foundation announced that Python 3.0 was released in 2008. 3.x is under active development and has already seen over five years of stable releases, including version 3.3 in 2012 and 3.4 in 2014. This means that all recent standard library improvements, for example, are only available by default in Python 3.x.

Guido van Rossum (the original creator of the Python language) decided to clean up Python 2.x properly, with less regard for backwards compatibility than is the case for new releases in the 2.x range. The most drastic improvement is the better Unicode support (with all text strings being Unicode by default) as well as saner bytes/Unicode separation.

Besides, several aspects of the core language (such as print and exec being statements, integers using floor division) have been adjusted to be easier for newcomers to learn and to be more consistent with the rest of the language, and old cruft has been removed (for example, all classes are now new-style, "range()" returns a memory efficient iterable, not a list as in 2.x).

3 Associated with each Python installation are the following platform-specific components:

- a) Virtual Machine
- b) Run Time Libraries
- c) Debuggers

2.4.2.2.3 Application Programs

Platform-specific application programs include:

- File Managers (Midnight Commander on Linux and Unix; Finder and PathFinder on Mac OS X; and Total Commander on Microsoft Windows)
- Text Editors (XEmacs on Linux, Mac OS X, Microsoft Windows and Unix)
- Python Integrated Development Environments (Python Idle, WingIDE etc.)

2.4.2.3 Architecture

There are two architectural views of the design:

- ***Stand Alone System Architecture*** (on page 268) - Description of the components of and relationship between components of an isolated system operating by itself.
- ***Stand Among System Architecture*** (on page 272) - Description of the components of and relationship between two or more networked systems operating in collaboration with each other.

2.4.2.3.1 "tsWxGTUI_PyVx" Toolkit Block Diagram

This Block Diagram depicts the organizational, functional and interface relationship between the TeamSTARS "tsWxGTUI_PyVx" Toolkit components and users (System Administrator, Software Engineer, System Operator and Field Service personnel):

- 1** the external System Operator interface to "tsToolkitCLI"
- 2** the internal "tsToolkitCLI" interface to "tsToolkitGUI"
- 3** the internal System Operator interfaces:
 - a) to "tsUtilities", "tsToolsCLI" and "tsTestsCLI" via "tsToolkitCLI"
 - b) to "tsToolsGUI" and "tsTestsGUI" via "tsToolkitCLI" and "tsToolkitGUI"

Graphical-style User Interface (tsToolkitGUI)	
▪ The "tsToolsGUI" is a set of graphical-style user interface application programs for tracking software development metrics.	▪ The "tsTestsGUI" is a set of graphical-style user interface application programs for regression testing and tutorial demos.
▪ The "tsLibGUI" is a library of graphical-style user interface building blocks that establishes the emulated run time environment for the high-level, pixel-mode, "wxPython" GUI Toolkit via the low-level, character-mode, "nCurses" Text User Interface Toolkit.	

^ ^ |
| | |
| | v

Command Line Interface (tsToolkitCLI)	
▪ The "tsToolsCLI" is a set of command line interface application programs and utility scripts for: checking source code syntax and style via "plint"; generating Unix-style "man page" documentation from source code comments via "pydoc"; and installing, modifying for publication, and tracking software development metrics.	▪ The "tsTestsCLI" is a set of command line interface application programs and scripts for regression testing and tutorial demos.
▪ The "tsLibCLI" is a library of command line building blocks that establishes the POSIX-style, run time environment for pre-processing source files, launching application programs, handling events (registering events with date, time and event severity annotations) and configuring console terminal and file system input and output.	
▪ The "tsUtilities" is a library of computer system configuration, diagnostic, installation, maintenance and performance tool components for various host hardware and software platforms.	

^ ^ |
| | +- > Operator Display & Log Files
| +----- Operator Keyboard
+----- Operator Mouse

2.4.2.3.2 Stand Alone System Architecture

The *Team*STARS "tsWxGTUI_PyVx" Toolkit provides:

1. Python version-specific components for its Command Line Interface ("tsToolkitCLI")
2. Python version-specific components for its Graphical-style User Interface ("tsToolkitGUI").

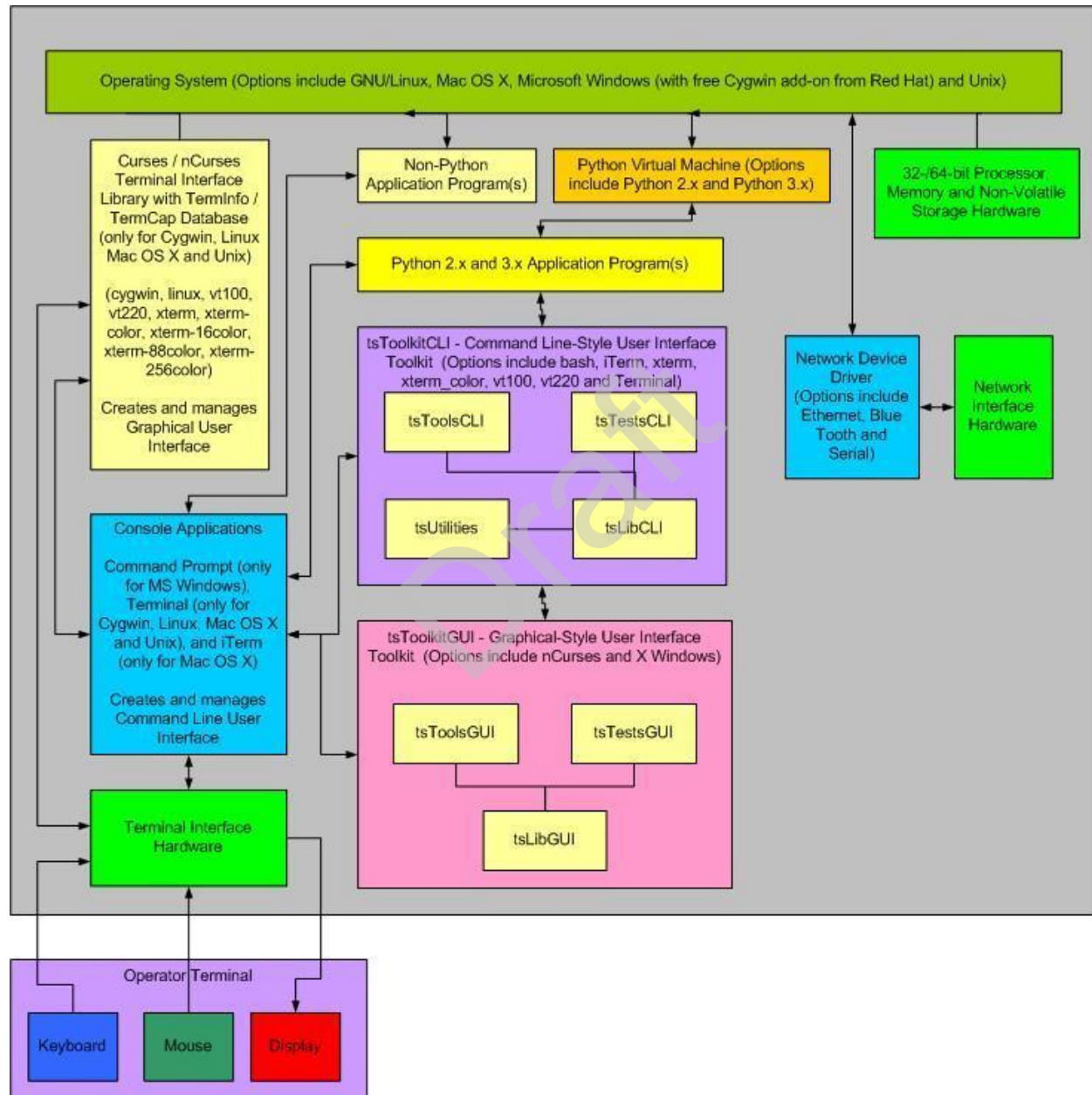
The following description uses the component names as depicted in the Block Diagram

This section depicts and describes the organization, function of and interface between various system hardware and software components and "tsWxGTUI_PyVx" Toolkit users (System Administrator, Software Engineer, System Operator and Field Service personnel):

- 1 the external System Operator interface to "tsToolkitCLI"
- 2 the internal "tsToolkitCLI" interface to "tsToolkitGUI"
- 3 the internal System Operator interfaces:

- a) to "tsLibCLI", "tsToolsCLI" . "tsTestsCLI" and "tsUtilities" via "tsToolkitCLI"
- b) to "tsLibGUI", "tsToolsGUI" and "tsTestsGUI" via "tsToolkitGUI" and "tsToolkitCLI"

This depiction represents a typical Stand Alone System configuration. In this configuration, the optional Network Hardware Interface and its associated Network Device Driver Interface should not be used, even if present, in order to avoid activities that adversely impact system performance.



- 1 **Operating System** - The platform specific software (such as Linux, MacOS X, Microsoft Windows and Unix) that coordinates and manages the time-shared use of a platform's processor, memory, storage and input/output hardware resources by multiple application programs and their associated users/operators.

2 Operator Terminal - A device for human interaction that includes:

- a) A Keyboard unit for text input
- b) A Mouse unit (mouse, trackball, trackpad or touchscreen with one or more physical or logical buttons) for selecting one of many displayed GUI objects to initiate an associated action.
- c) A Display unit (1-color "ON"/"OFF" or multi-color two-dimensional screen) for output of text and graphic-style, tiled and overlaid boxes.

3 Terminal Hardware Interface - The platform specific hardware with connections to the device units of the Operator Terminal.

- a) A PS/2 Port is a type of input port developed by IBM for connecting a mouse or keyboard to a Personal Computer. It supports a mini DIN plug containing just 6 pins.
- b) An RS-232C or RS-422 port for connecting a mouse for position and button-click input.
- c) A Universal Serial Bus (USB) port is an industry standard first developed in the mid-1990s that was designed to standardize the connection of computer peripherals (including keyboards, pointing devices, digital cameras, printers, portable media players, disk drives and network adapters) to personal computers, both to communicate and to supply electric power. It has effectively replaced a variety of earlier interfaces, such as serial and parallel ports, as well as separate power chargers for portable devices.

The USB 1.0 specification was introduced in January 1996. It defined data transfer rates of 1.5 Mbit/s "Low Speed" and 12 Mbit/s "Full Speed". The first widely used version of USB was 1.1 (now called "Full-Speed"), which was released in September 1998. Its 12 Mbit/s data rate was intended for higher-speed devices such as disk drives, and its lower 1.5 Mbit/s rate for low data rate devices such as joysticks.

The USB 2.0 (now called "Hi-Speed") specification was released in April 2000. It defined a higher data transfer rate, with the resulting specification achieving 480 Mbit/s, a 40-times increase over the original USB 1.1 specification.

The USB 3.0 specification (now called "SuperSpeed") was preleased in November 2008. It defined an even higher data transfer rate (up to 5 Gbit/s) and was backwards-compatible with USB 2.0. It added a new, higher speed bus called SuperSpeed in parallel with the USB 2.0 bus.

- d) A Video Adapter is a computer circuit card that provides digital-to-analog conversion, video RAM, and a video controller so that data can be sent to a computer's display. It typically adheres to the de facto standard, Video Graphics Array (VGA). VGA describes how data - essentially red, green, blue data streams - is passed between the computer and the display. It also describes the frame refresh rates in hertz. It also specifies the number and width of horizontal lines, which essentially amounts to specifying the resolution of the pixels that are created. VGA supports four different resolution settings and two related image refresh rates. The maximum VGA resolution setting produces a display that is 640 pixels wide by 480 pixels high. For a character font that is 8 pixels wide by 12 pixels high, the longest line of text will be 80 characters wide and there can be up to 40 lines of text displayed at any moment. Higher resolutions, such as SVGA, are supported by more advanced Video Adapters. The higher resolution settings typically require use of proportionally larger displays in order to maintain the size and legibility of the displayed text.

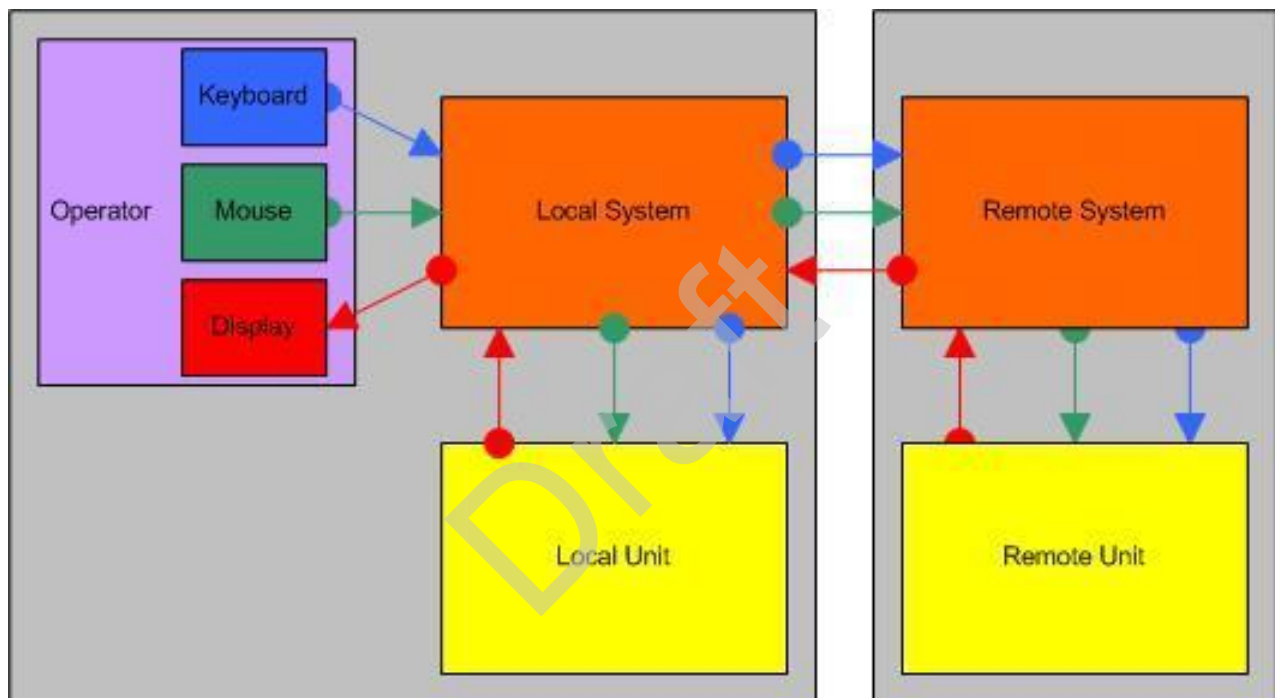
4 Terminal Device Driver - The platform specific software for transforming data (such as single button scan codes, multi-button flags and pointer position) to and from the platform independent formats (such as upper and lower case text, display screen column and row and displayed colors, fonts and special effects) used by the Command Line Interface and Graphical User Interface software.

- 5 **Command Line-Style Interface ("tsToolKitCLI")** - The platform specific keywords arguments, positional arguments and their associated values and syntax of text used to request services from the Operating System and various Application Programs.
 - a) **"tsLibCLI"** --- A library of command line building blocks that establishes the POSIX-/Unix-style, run time environment for pre-processing source files, launching application programs, handling events (registering events with date, time and event severity annotations) and configuring console terminal and file system input and output.
 - b) **"tsToolsCLI"** --- A set of command line interface application programs and utility scripts for: checking source code syntax and style via "plint"; generating Unix-style "man page" documentation from source code comments via "pydoc"; and installing, modifying for publication and tracking software development metrics.
 - c) **"tsTestsCLI"** --- A set of command line interface application programs and utility scripts for unit, integration and system level regression testing.
 - d) **"tsUtilities"** --- A library of computer system configuration, diagnostic, installation, maintenance and performance tool components for various host hardware and software platforms.
- 6 **Graphical-Style User Interface ("tsToolKitGUI")** - The platform specific tiled, overlaid and click-to-select Frames, Dialogs, Pull-down Menus, Buttons, CheckBoxes, Radio Buttons, Scrollbars and associated keywords, values and syntax of text used to request services from the Operating System and various Application Programs.
 - a) **"tsLibGUI"** --- A library of graphical-style user interface building blocks that establishes the emulated run time environment for the high-level, pixel-mode, "wxPython" GUI Toolkit via the low-level, character-mode, "nCurses" Text User Interface Toolkit.
 - b) **"tsToolsGUI"** --- A set of graphical-style user interface application programs for tracking software development metrics.
 - c) **"tsTestsGUI"** --- A set of graphical-style user interface application programs and command line interface utility scripts for unit, integration and system level regression testing.
- 7 **Python Application Program** - The application specific program that performs its service when executed by the Python Virtual Machine.
- 8 **Non-Python Application Program** - The application specific program that performs its service when its pre-compiled, platform specific machine code is executed. Typically, these services are used to analyze, edit, view, copy, move or delete those data and log files which are of interest or no longer needed.
- 9 **Python Virtual Machine** - The platform specific program that loads, interprets and executes the platform independent source code of a Python language application program.
- 10 **Processor, Memory, Storage and Communication Hardware** - Platform specific resources that are required by the Operating System and Application software.
- 11 **Network Hardware Interface** - The optional platform specific ethernet, blue-tooth and RS-232 serial port hardware for physical connections between the local system and one or more remote systems. It may also include such external hardware as gateways, routers, network bridges, switches, hubs, and repeaters. It may also include hybrid network devices such as multilayer switches, protocol converters, bridge routers, proxy servers, firewalls, network address translators, multiplexers, network interface controllers, wireless network interface controllers, modems, ISDN terminal adapters, line drivers, wireless access points, networking cables and other related hardware.

In this configuration, the Local (Left) and Remote (Right) systems must first be networked via the available communication resources (Network Interface Hardware and Network Device Driver Interface).

Once networked, the local system operator must login to the Remote system via the "ssh user@Remote" command. The Local and Remote Terminal Device Interface then establishes a logical communication channel for exchanging keyboard, mouse and display information.

For each login Local and Remote session, the Operator may then select and run an Application Program. As depicted in the following figure. Application Programs run as Local Units on the system to which the Operator first logged in. Application Programs run as Remote Units on the systems to which the operator logged in via the "ssh user@Remote" command.



NOTE: Concurrent login sessions are a convenience that must be used with caution. Time critical, resource intensive activities ought to be performed individually or sequentially (but NOT concurrently) on a Stand Alone System. Only non-time critical, non-resource intensive activities may be performed concurrently on Stand Alone or Stand Among Systems.

- 1 **Local System (Left)** --- Operator opens one or more Command Line Interface Shells.
 - One or more newly opened shells may be used to run a Python or non-Python Application Program as its **Local Unit (Left.)**
 - One or more newly opened shells may be used to login to a Remote system (via the "ssh user@Remote" command). Once Remote login has been successful, the operator may run a Python or non-Python Application Program as a **Remote Unit (Right)**.
- 2 **Local Unit (Left)** --- A Python or non-Python Application Program that has been launched in a Local Command Line Interface Shell.
- 3 **Remote System (Right)** - Same Operator opens one or more Command Line Interface Shells.

- One or more newly opened shells may be used to run a Python or non-Python Application Program as its **Remote Unit (Right)**.

The **Multi-Session Desktop** (on page 274) figure depicts the desktop of a multi-user, multi-process, multi-threaded computer running the Professional Edition of Microsoft Windows 7. Among the background of desktop icons, there are two *TeamSTARS* "tsWxGTUI_PyVx" Toolkit sessions. The time each session displays synchronizes within its own one second refresh interval. The local session, on the left, is actively running Python 3.x on the Windows platform. The remote session, on the right, is actively running Python 2.x on the Mac OS X Yosemite platform which also serves as the Parallels 10 Hypervisor host for diverse Guest Operating Systems including:

Linux (CentOS7 64-bit, Fedora 21 64-bit, Scientific 6.5 32-bit and Ubuntu 12.04 32-bit)

Windows (98 SE 16-bit, XP 32-bit, 7 32-bit, 8 32-bit and 8.1 32-bit)

Unix (FreeBSD 9.2, PC-BSD 10, OpenIndiana 151a8 and OpenSolaris 11 32-bit)

- One or more newly opened shells may be used to login to a Remote system (via the "ssh user@Remote command). Once Remote login has been successful, the operator may run a Python or non-Python Application Program as a **Remote Unit (Right)**.

4 Remote Unit (Right) --- A Python or non-Python Application Program that has been launched in a Remote Command Line Interface Shell.

2.4.2.3.3.1 Multi-Session Desktop

From Wikipedia, the free encyclopedia

"In computer science, in particular networking, a session is a semi-permanent interactive information interchange, also known as a dialogue, a conversation or a meeting, between two or more communicating devices, or between a computer and user (see Login session). A session is set up or established at a certain point in time, and then torn down at some later point. An established communication session may involve more than one message in each direction. A session is typically, but not always, stateful, meaning that at least one of the communicating parts needs to save information about the session history in order to be able to communicate, as opposed to stateless communication, where the communication consists of independent requests with responses.

An established session is the basic requirement to perform a connection-oriented communication. A session also is the basic step to transmit in connectionless communication modes. However any unidirectional transmission does not define a session.[1]

Communication sessions may be implemented as part of protocols and services at the application layer, at the session layer or at the transport layer in the OSI model.

1 Application layer examples:

- a) HTTP sessions, which allow associating information with individual visitors
- b) A telnet remote login session

2 Session layer example:

- a) A Session Initiation Protocol (SIP) based Internet phone call

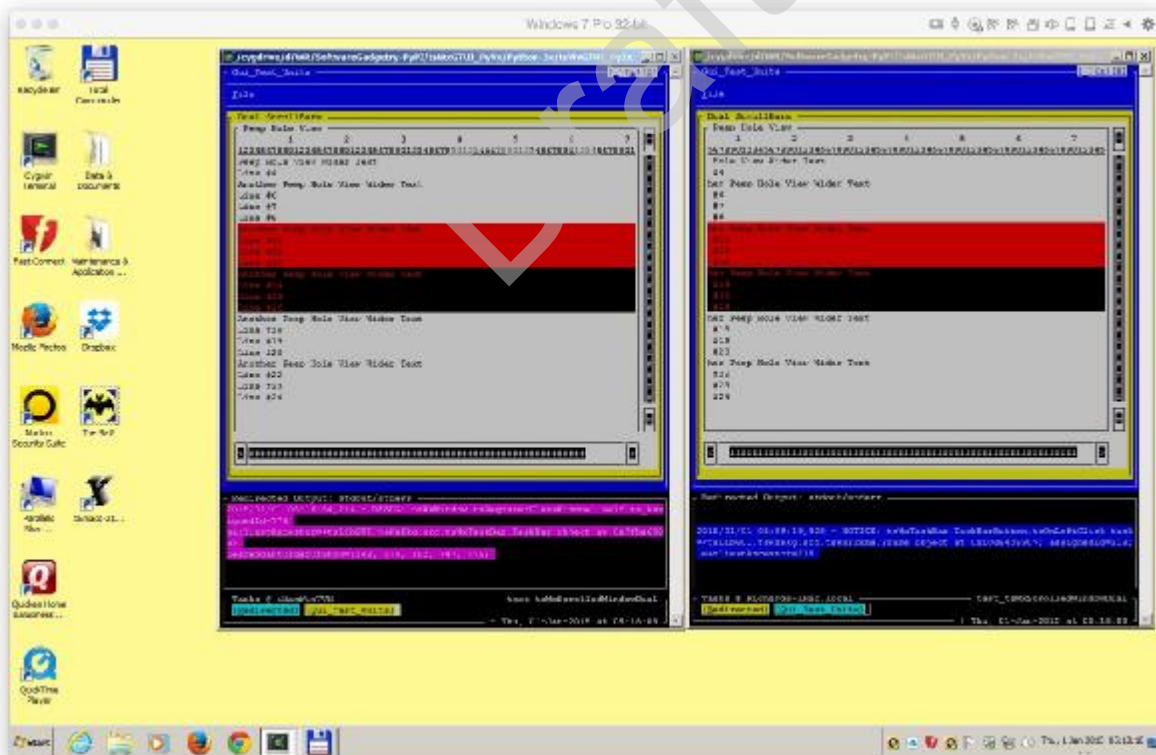
3 Transport layer example:

- a) A TCP session, which is synonymous to a TCP virtual circuit, a TCP connection, or an established TCP socket.

In the case of transport protocols that do not implement a formal session layer (e.g., UDP) or where sessions at the application layer are generally very short-lived (e.g., HTTP), sessions are maintained by a higher level program using a method defined in the data being exchanged. For example, an HTTP exchange between a browser and a remote host may include an HTTP cookie which identifies state, such as a unique session ID, information about the user's preferences or authorization level.

HTTP/1.0 was thought to only allow a single request and response during one Web/HTTP Session. However a workaround was created by David Hostettler Wain in 1996 such that it was possible to use session IDs to allow multiple phase Web Transaction Processing (TP) Systems (in ICL[disambiguation needed] nomenclature), with the first implementation being called Deity. Protocol version HTTP/1.1 further improved by completing the Common Gateway Interface (CGI) making it easier to maintain the Web Session and supporting HTTP cookies and file uploads.

Most client-server sessions are maintained by the transport layer - a single connection for a single session. However each transaction phase of a Web/HTTP session creates a separate connection. Maintaining session continuity between phases required a session ID. The session ID is embedded within the <A HREF> or <FORM> links of dynamic web pages so that it is passed back to the CGI. CGI then uses the session ID to ensure session continuity between transaction phases. One advantage of one connection-per-phase is that it works well over low bandwidth (modem) connections. Deity used a sessionID, screenID and actionID to simplify the design of multiple phase sessions."



The Sample Screenshot above shows a Windows 7 Desktop with:

- 1 A local Windows host with Python 3x session on left; and

2 A remote Mac OS X host with Python 2x session on right.

3 Each session consists of

- a) a wxPython-style Frame named "Dual ScrollBars" containing scrollable text with optional color markup.
- b) a wxPython-style Frame named "Redirected Output" containing date and time stamped event messages, with optional with color markup, that scroll up when new events are registered.
- c) a Host Desktop-style Frame named "Tasks @ Host Name" and Application Name with buttons to shift focus from background to foreground. There is also a spinner (to indicate the frequency or absence of idle time) and the current date and time (to indicate when the display was last updated).

Draft

2.5 APPENDIX D - SYSTEM INTERNAL INTERFACE REQUIREMENTS

This paragraph shall specify the requirements, if any, imposed on interfaces internal to the system. If all internal interfaces are left to the design or to requirement specifications for system components, this fact shall be so stated. If such requirements are to be imposed, paragraph 3.3 of this DID provides a list of topics to be considered.

1 *Toolkit Architecture* (on page 277)

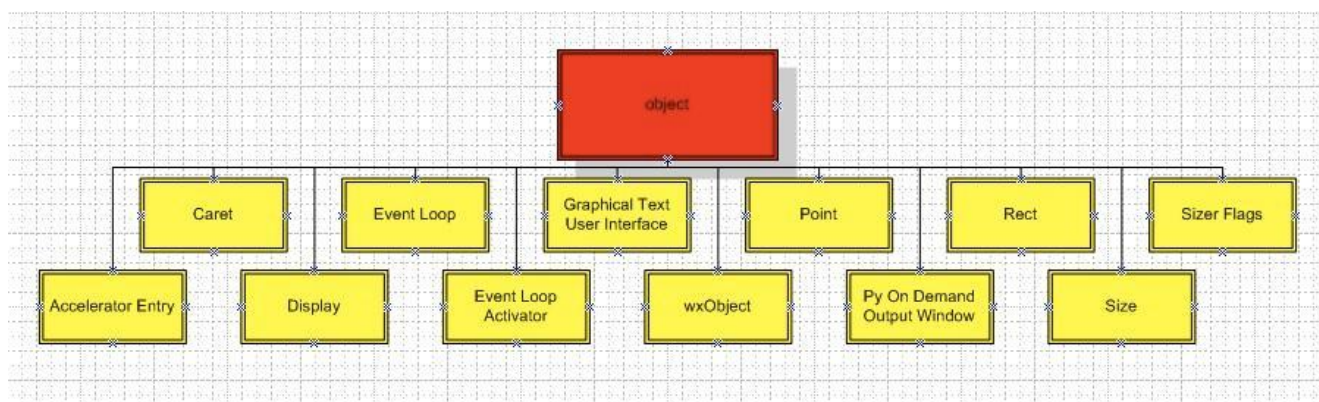
2.5.1 Toolkit Architecture

The following examples highlight the hierarchical dependency relationship between various tsWxGTUI objects:

- **"object" Dependents** (on page 277) - Python's "object" class is the parent class for the various wxPython classes.
- **"Object" Dependents** (see **"wxObject" Dependents** on page 278) - wxPython's "Object" class is a decendent of Python's "object" class and the parent class for the various wxPython classes, some of which have their own descendants.
- **"EvtHandler" Dependents** (see **"wxEvtHandler" Dependents** on page 279) - wxPython's "EvtHandler" class is a descendant of wxPython's "Object" class and the parent class for the various wxPython classes, some of which have their own descendants.
- **"Window" Dependents** (see **"wxWindow" Dependents** on page 280) - wxPython's "Window" class is a descendant of wxPython's "EvtHandler" class. It is the base class for all windows and represents any visible object on the screen.

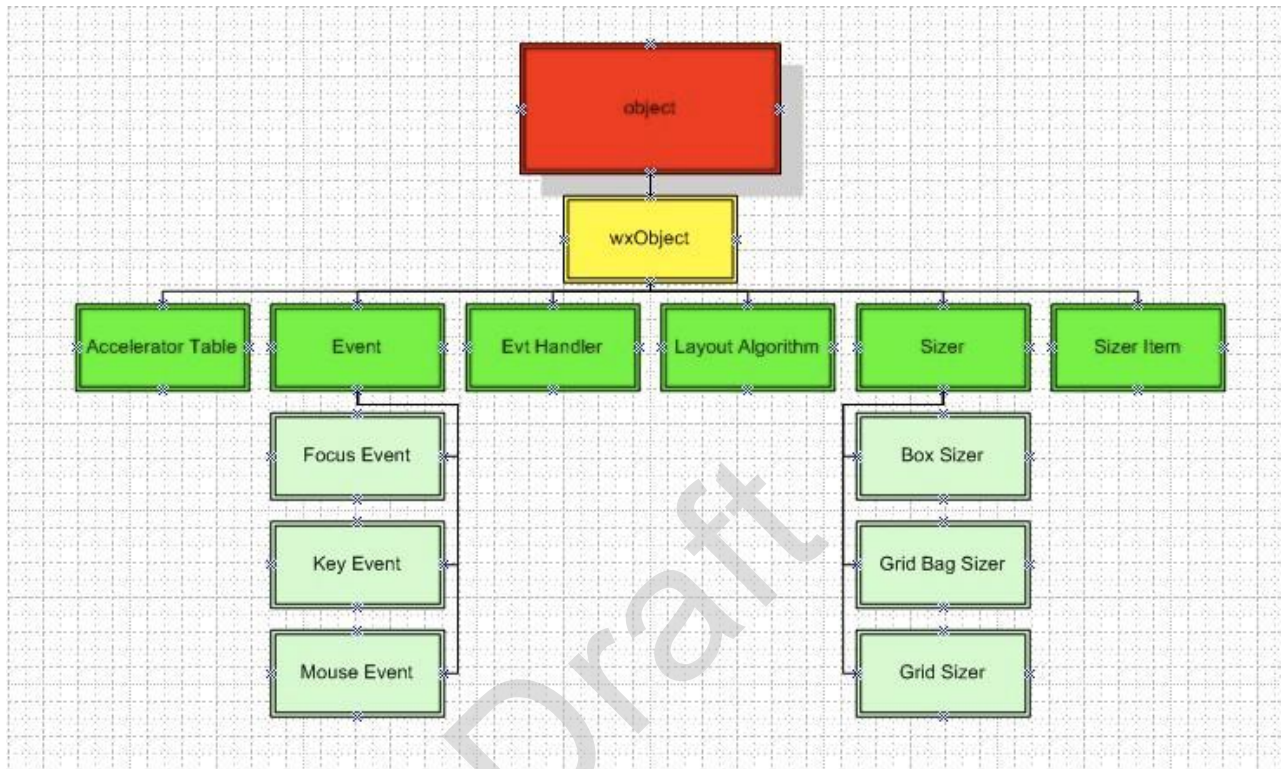
2.5.1.1 "object" Dependents

Python's "object" class is the parent class for the following wxPython classes.



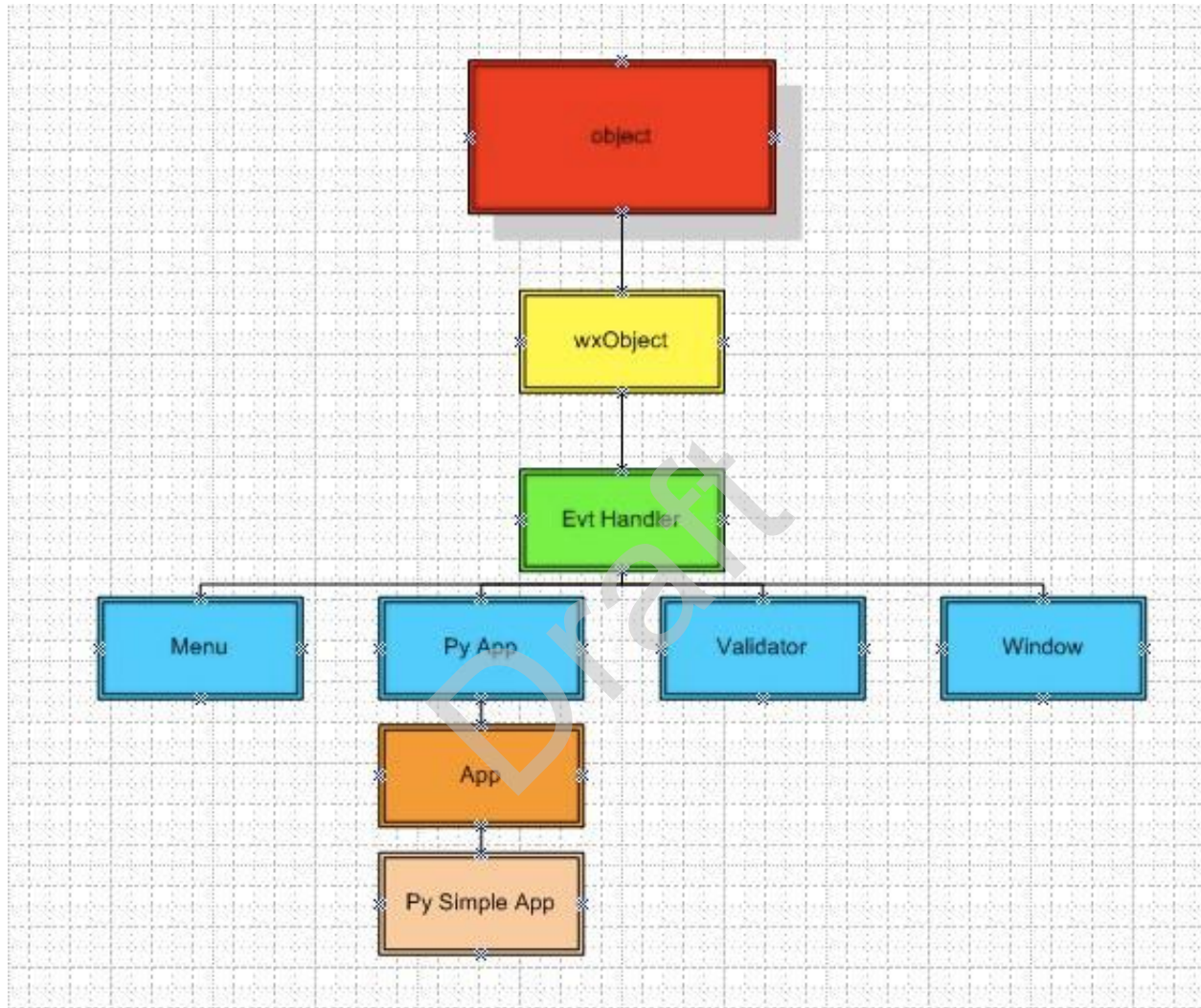
2.5.1.2 "wxObject" Dependents

wxPython's "Object" class is a descendant of Python's "object" class and the parent class for the following wxPython classes, some of which have their own descendants.



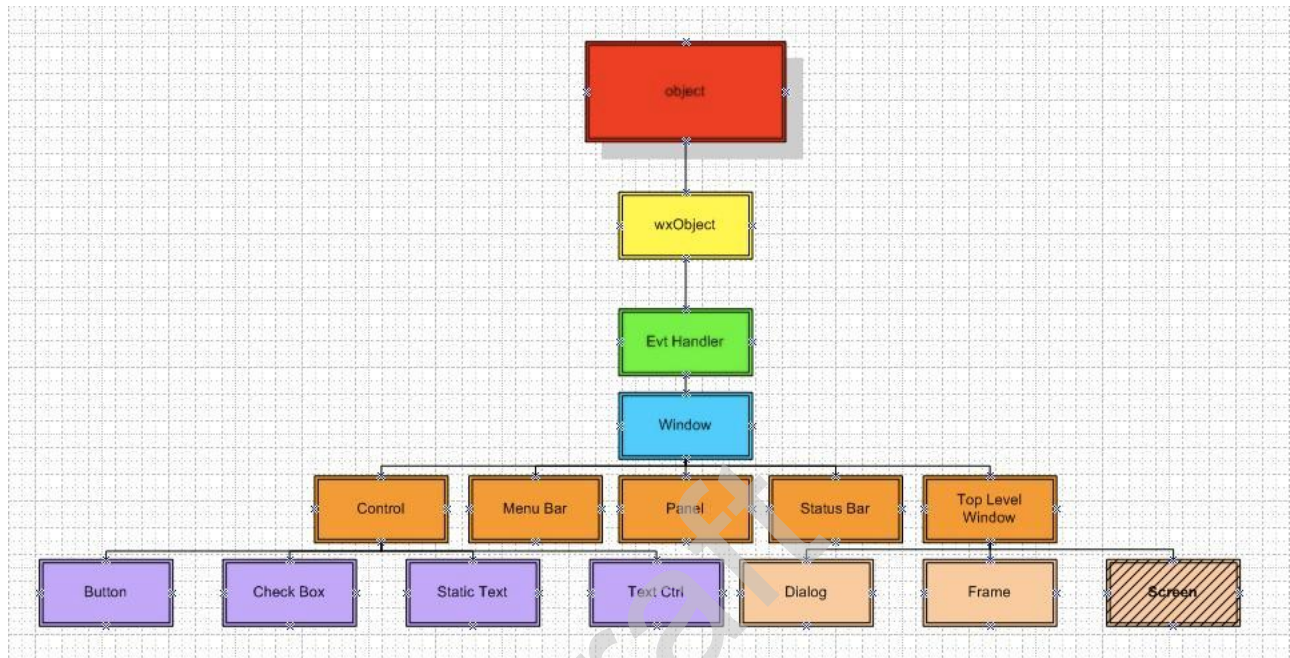
2.5.1.3 "wxEvtHandler" Dependents

wxPython's "EvtHandler" class is a descendant of wxPython's "Object" class and the parent class for the following wxPython classes, some of which have their own descendants.



2.5.1.4 "wxWindow" Dependents

wxPython's "Window" class is a descendant of wxPython's "EvtHandler" class. It is the base class for all windows and represents any visible object on the screen.



2.6 APPENDIX E - SYSTEM INTERNAL DATA REQUIREMENTS

This paragraph shall specify the requirements, if any, imposed on data internal to the system. Included shall be requirements, if any, on databases and data files to be included in the system. If all decisions about internal data are left to the design or to requirements specifications for system components, this fact shall be so stated. If such requirements are to be imposed, paragraphs 3.3.x.c and 3.3.x.d of this DID provide a list of topics to be considered.

Draft

Draft

2.7 APPENDIX F - ADAPTATION REQUIREMENTS

This paragraph shall specify the requirements, if any, concerning installation-dependent data that the system is required to provide (such as site-dependent latitude and longitude or site-dependent state tax codes) and operational parameters that the system is required to use that may vary according to operational needs (such as parameters indicating operation-dependent targeting constants or data recording).

Draft

Draft

2.8 APPENDIX G - SAFETY REQUIREMENTS

This paragraph shall specify the system requirements, if any, concerned with preventing or minimizing unintended hazards to personnel, property, and the physical environment. Examples include restricting the use of dangerous materials; classifying explosives for purposes of shipping, handling, and storing; abort/escape provisions from enclosures; gas detection and warning devices; grounding of electrical systems; decontamination; and explosion proofing. This paragraph shall include the system requirements, if any, for nuclear components, including, as applicable, requirements for component design, prevention of inadvertent detonation, and compliance with nuclear safety rules.

Draft

Draft

2.9 APPENDIX H - SECURITY AND PRIVACY REQUIREMENTS

This paragraph shall specify the system requirements, if any, concerned with maintaining security and privacy. The requirements shall include, as applicable, the security/privacy environment in which the system must operate, the type and degree of security or privacy to be provided, the security/privacy risks the system must withstand, required safeguards to reduce those risks, the security/privacy policy that must be met, the security/privacy accountability the system must provide, and the criteria that must be met for security/privacy certification/accreditation.

Draft

Draft

2.10 APPENDIX I - SYSTEM ENVIRONMENT REQUIREMENTS

This paragraph shall specify the requirements, if any, regarding the environment in which the system must operate. Examples for a software system are the computer hardware and operating system on which the software must run. (Additional requirements concerning computer resources are given in the next paragraph). Examples for a hardware-software system include the environmental conditions that the system must withstand during transportation, storage, and operation, such as conditions in the natural environment (wind, rain, temperature, geographic location), the induced environment (motion, shock, noise, electromagnetic radiation), and environments due to enemy action (explosions, radiation).

Draft

Draft

2.11 APPENDIX J - COMPUTER RESOURCE REQUIREMENTS

This paragraph shall be divided into the following subparagraphs. Depending upon the nature of the system, the computer resources covered in these subparagraphs may constitute the environment of the system (as for a software system) or components of the system (as for a hardware-software system).

To demonstrate computer resource requirements and usability considerations for various application-specific capabilities, the "tsWxGTUI_PyVx" Toolkit developers have used the following platform configurations:

Draft

Draft

Make & Model	Hardware	Software
Apple 27" iMac Desktop	<p>2013-model year, 3.5 GHz Intel Quad Core i7 processor-based desktop with 16 GB RAM, 2560x1440 pixel resolution display and sufficient performance, resources and expansion capabilities to serve as the high-level baseline development and operator platform.</p> <ul style="list-style-type: none"> Its 3 TB 7200 RPM internal hard drive had 128 GB Solid State Flash memory and was used to run Apple's Mac OS X 10.9 and the hypervisor virtualization applications (Parallels Desktop 9 and VMware Fusion 5) that supported various guest operating systems. Its 3 TB internal hard drive was also used to store and run configured versions of the Fedora Linux (20), Scientific Linux (6.4), Ubuntu Linux (12.04), Microsoft Windows (8.1/8 Pro, 7 Pro and XP Pro) and Unix (PC-BSD (9.2), OpenSolaris 11/OpenIndiana 151A8) guest operating systems. Hewlett-Packard Company P2015DN Series (265C8) LaserJet Printer connected via Ethernet Hewlett-Packard Company Officejet Pro 8500A (A910) e-All-in-One Series Printer connected via Wi-Fi or USB. 	<p>Host Operating System</p> <ul style="list-style-type: none"> Apple's Unix-like, Darwin-based Mac OS X 10.9 with Python 2.6.8, 2.7.5, 3.2.2 <p>Host Hypervisor Virtualization Applications</p> <ul style="list-style-type: none"> Parallels Desktop 9 VMware Fusion 5 <p>Concurrent or Interchangeable Guest Operating Systems (cloned from Apple 17" MacBook Pro Laptop and configured to share host drives and use 1440 x 900 display)</p> <ul style="list-style-type: none"> Linux (Fedora 20 64-bit, OpenSuSE 12.2 64-bit, Scientific (CentOS) 6.4-6.5 64-bit, Ubuntu 12.04 32-bit) with Python 2.7 and 3.2. Windows (8.1 Professional 32-bit, 8 Professional 32-bit, 7 Professional 32-bit with SP1, XP Professional 32-bit with SP3) with POSIX-like Cygwin plug-in and Python 2.6, 2.7, 3.2 and 3.3 UNIX (PC-BSD 9.2-10.0 64-bit without Parallels Tools, OpenIndiana 151A8 32-bit without Parallels Tools, a replacement for unreleased OpenSolaris 11/SunOS 5.11) with Python 2.6.4 running on Apple MacBook Pro
Apple 17" MacBook Pro Laptop	<p>2007-model year, 2.33 GHz Intel Core 2 Duo processor-based laptop with 4 GB RAM, 1920x1200 pixel resolution display and sufficient performance, resources and expansion capabilities to serve as the moderate-level baseline development and operator platform.</p> <ul style="list-style-type: none"> Its 160 GB internal hard drive was used to run Apple's Mac OS X 10.7 and the hypervisor virtualization applications (Parallels Desktop 8 and VMware Fusion 5) that supported various guest operating systems. Its 1.5 TB external hard drive was used to store and run configured versions of the Ubuntu Linux (12.04), Microsoft Windows (8 Pro, 7 Pro and XP Pro) and Unix (OpenSolaris 11/OpenIndiana 151) guest operating systems. Hewlett-Packard Company P2015DN Series (265C8) LaserJet Printer connected via Ethernet 	<p>Host Operating System</p> <ul style="list-style-type: none"> Apple's Unix-like, Darwin-based Mac OS X 10.7 with Python 2.6.8, 2.7.5, 3.2.2 <p>Host Hypervisor Virtualization Applications</p> <ul style="list-style-type: none"> Parallels Desktop 8 VMware Fusion 5 <p>Interchangeable Guest Operating Systems (configured to share host drives and use 1440 x 900 display)</p> <ul style="list-style-type: none"> Linux (Fedora 20 64-bit, OpenSuSE 12.2 64-bit, Scientific (CentOS) 6.4-6.5 64-bit, Ubuntu 12.04 32-bit) with Python 2.7 and 3.2. Windows (8.1 Professional 32-bit, 8 Professional 32-bit, 7 Professional 32-bit with SP1, XP Professional 32-bit with SP3) with POSIX-like Cygwin plug-in and Python 2.6, 2.7, 3.2 and 3.3 UNIX (PC-BSD 9.2-10.0 64-bit without Parallels Tools, OpenIndiana 151A8 32-bit

	<ul style="list-style-type: none"> ▪ Hewlett-Packard Company Officejet Pro 8500A (A910) e-All-in-One Series Printer connected via Wi-Fi or USB. 	without Parallels Tools, a replacement for unreleased OpenSolaris 11/SunOS 5.11) with Python 2.6.4 running on Apple MacBook Pro
Dell 15.6" Inspiron 7000 Laptop	<p>1998-model year, 366 MHz Intel Pentium II-based laptop with 384 MB RAM, 640x480/1024x768 pixel resolution display and expansion capabilities (marginal resources and performance) sufficient enough to serve as the low-level baseline development and operator platform.</p> <ul style="list-style-type: none"> ▪ Its interchangeable 32 GB (4200 RPM) ATA hard drives were used to run Windows XP Pro or to run Ubuntu 12.04 LTS Linux operating system kernel with GNU toolchain. ▪ The platform's limited memory and available PCMCIA network adapters were compatible with: (1) Windows XP Pro via Xircom RBEM56G-100, a RealPort CardBus Ethernet 10/100+Modem 56 or 3CPM 3CRWE62092A Wireless LAN PC Card; (2) Ubuntu 12.04 LTS Linux via LINKSYS WPC11 Instant Wireless Network PC Card. The platform was incompatible with later versions of Windows and with other Linux distributions. later versions of Windows or with other Linux distributions. ▪ Hewlett-Packard Company Photosmart C3180 All-in-One Printer, Scanner, Copier connected via USB. 	<p>Interchangeable Host Operating System</p> <ul style="list-style-type: none"> ▪ Windows XP Professional with SP3) with POSIX-like Cygwin plug-in and Python 2.6, 2.7 , 3.2 and 3.3 ▪ Linux (Ubuntu 12.04) with Python 2.7 and 3.3

2.12 APPENDIX K - SYSTEM QUALITY FACTORS

This paragraph shall specify the requirements, if any, pertaining to system quality factors. Examples include quantitative requirements concerning system functionality (the ability to perform all required functions), reliability (the ability to perform with correct, consistent results -- such as mean time between failure for equipment), maintainability (the ability to be easily serviced, repaired, or corrected), availability (the ability to be accessed and operated when needed), flexibility (the ability to be easily adapted to changing requirements), portability of software (the ability to be easily modified for a new environment), reusability (the ability to be used in multiple applications), testability (the ability to be easily and thoroughly tested), usability (the ability to be easily learned and used), and other attributes.

Draft

Draft

2.13 APPENDIX L - DESIGN AND CONSTRUCTION CONSTRAINTS

This paragraph shall specify the requirements, if any, that constrain the design and construction of the system. For hardware-software systems, this paragraph shall include the physical requirements imposed on the system. These requirements may be specified by reference to appropriate commercial or military standards and specifications. Examples include requirements concerning:

- a. Use of a particular system architecture or requirements on the architecture, such as required subsystems; use of standard, military, or existing components; or use of Government/acquirer-furnished property (equipment, information, or software)
- b. Use of particular design or construction standards; use of particular data standards; use of a particular programming language; workmanship requirements and production techniques
- c. Physical characteristics of the system (such as weight limits, dimensional limits, color, protective coatings); interchangeability of parts; ability to be transported from one location to another; ability to be carried or set up by one, or a given number of, persons
- d. Materials that can and cannot be used; requirements on the handling of toxic materials; limits on the electromagnetic radiation that the system is permitted to generate
- e. Use of nameplates, part marking, serial and lot number marking, and other identifying markings
- f. Flexibility and expandability that must be provided to support anticipated areas of growth or changes in technology, threat, or mission

Draft

2.14 APPENDIX M - PERSONNEL-RELATED REQUIREMENTS

This paragraph shall specify the system requirements, if any, included to accommodate the number, skill levels, duty cycles, training needs, or other information about the personnel who will use or support the system. Examples include requirements for the number of work stations to be provided and for built-in help and training features. Also included shall be the human factors engineering requirements, if any, imposed on the system. These requirements shall include, as applicable, considerations for the capabilities and limitations of humans, foreseeable human errors under both normal and extreme conditions, and specific areas where the effects of human error would be particularly serious. Examples include requirements for adjustable-height work stations, color and duration of error messages, physical placement of critical indicators or buttons, and use of auditory signals.

Draft

Draft

2.15 APPENDIX N - TRAINING- RELATED REQUIREMENTS

This paragraph shall specify the system requirements, if any, pertaining to training. Examples include training devices and training materials to be included in the system.

Draft

Draft

2.16 APPENDIX O - LOGISTICS-RELATED REQUIREMENTS

This paragraph shall specify the system requirements, if any, concerned with logistics considerations. These considerations may include: system maintenance, software support, system transportation modes, supply-system requirements, impact on existing facilities, and impact on existing equipment.

Draft

Draft

2.17 APPENDIX P - OTHER REQUIREMENTS

This paragraph shall specify additional system requirements, if any, not covered in the previous paragraphs. Examples include requirements for system documentation, such as specifications, drawings, technical manuals, test plans and procedures, and installation instruction data, if not covered in other contractual documents.

Draft

Draft

2.18 APPENDIX Q - PACKAGING REQUIREMENTS

This section shall specify the requirements, if any, for packaging, labeling, and handling the system and its components for delivery. Applicable military specifications and standards may be referenced if appropriate.

Draft

Draft

2.19 APPENDIX R - PRECEDENCE AND CRITICALITY OF REQUIREMENTS

This paragraph shall specify, if applicable, the order of precedence, criticality, or assigned weights indicating the relative importance of the requirements in this specification. Examples include identifying those requirements deemed critical to safety, to security, or to privacy for purposes of singling them out for special treatment. If all requirements have equal weight, this paragraph shall so state.

Draft

Draft

CHAPTER 3

3 INTERFACE REQUIREMENTS SPECIFICATION

Draft

Draft

CHAPTER 4

4 SOFTWARE REQUIREMENTS SPECIFICATION

4.1.1.1 In This Chapter

APPENDIX A - Nature of System and Software	314
APPENDIX B - Software Design Constraints.....	323

Draft

4.2 APPENDIX A - Nature of System and Software

The nature of the system and software depends on the observer's functional role:

- **System Administrator** (on page 318) - In this role, the perspective and activities are that of an individual who will specify, plan, supervise and/or perform the installation, configuration, maintenance, repair and support of the computer hardware, operating system, application software and network to be used by Software Engineers and System Operators.
- **Software Engineer** (on page 319) - In this role, the perspective and activities are that of an individual who will specify, plan, supervise and/or perform the design, development, coding, testing, debugging, maintenance and support of application programs, with Command Line Interface or Graphical User Interface, to be used by System Operators.
- **System Operator** (on page 319) - In this role, the perspective and activities are that of an individual who will use various application programs, with associated Command Line Interface or Graphical User Interface, to interactively supervise communication, control, data base, diagnostic, instrumentation or simulation activities.

Regardless of functional role, the general nature of user interfaces and systems are summarized in the following sections:

- **Command Line and Graphical User Interfaces** (on page 315) - Describes the two types of interfaces a human operator typically uses to interact with a computer system.
- **Hardware Components** - Describes typical computer processor, memory, storage, networking, input and output hardware.
- **Software Components** - Describes typical operating system, Python Virtual Machine and "tsWxGTUI_PyVx" Toolkit software

Typical hardware and software configurations are identified in the following section:

- **System Configurations** (on page 315) - Identifies various assemblies of computer products that would be suitable for use by Software Engineers or System Operators.

Typical "tsWxGTUI_PyVx" Toolkit scenarios for the System Administrator, Software Engineer and System Operator are summarized in the following sections:

- **Application Programming Interface** - The following description is taken from Wikipedia, the free encyclopedia, at http://en.wikipedia.org/wiki/Application_programming_interface:
 - An application programming interface (API) is a particular set of rules ('code') and specifications that software programs can follow to communicate with each other. It serves as an interface between different software programs and facilitates their interaction, similar to the way the user interface facilitates interaction between humans and computers. API can be created for applications, libraries, operating systems, etc., as a way of defining their "vocabularies" and resources request conventions (e.g. function-calling conventions).
 - It may include specifications for routines, data structures, object classes, and protocols used to communicate between the consumer program and the implementer program of the API.

- **Use Case(s)** (see "*Computer User Use Case(s)*" on page 317) - The following description is taken from Wikipedia, the free encyclopedia, at http://en.wikipedia.org/wiki/Use_case:
 - A use case in software engineering and systems engineering, is a description of steps or actions between a user (or "actor") and a software system which leads the user towards something useful.[1] The user or actor might be a person or something more abstract, such as an external software system or manual process.
 - Use cases are a software modeling technique that helps developers determine which features to implement and how to gracefully resolve errors.[2]
 - Within systems engineering, use cases are used at a higher level than within software engineering, often representing missions or stakeholder goals. The detailed requirements may then be captured in SysML requirement diagrams or similar mechanisms.

4.2.1 Command Line and Graphical User Interfaces

Humans interact with computers via those hardware and software components associated with the User Interface.

- **Command Line Interface:** Humans began interacting with computers via a mechanical terminal device, such as a teletype, and the computer's Command Line Interface (CLI). The computer would print out a single line of text to prompt the user for input. It would wait for the user to type in a command. In responding to the command, it would print out the appropriate data followed by a prompt for the next command. For additional details see: *Text Mode* (on page 188).
- **Graphical User Interface:** Over the years, human interactions have become more complex. Modern computer now concurrently wait for multiple user inputs while updating multiple output areas on an electronic terminal device, such as a laptop computer. Today, the Graphical User Interface (GUI) is more convenient to use than the Command Line Interface. It more fully displays the information and actions available to a user. Considerable time and effort goes into the design and implementation of a GUI. Technological advances have created toolkits that provide standard class, method and data components that enable the developer to design and implement only the application specific components. For additional details see: *Graphical Mode* (on page 192).

4.2.2 System Configurations

The following assemblies of computer hardware and software products would be suitable for use by:

- Software Engineers, who develop application programs with the "tsWxGTUI_PyVx" toolkit
- System Operators, who use the application programs

The "tsWxGTUI_PyVx" Toolkit and the application programs produced by it are designed for use on commonly available Laptop, Desktop or Workstation type platforms. It is only necessary that the platform be appropriate for the application and include the following:

- 1 keyboard input device
- 2 pointing input device (mouse, touchpad, touchscreen or trackball)

- 3 display output device (with at least 640 x 480 pixel size)
- 4 Unix-style "nCurses" program library
- 5 Python virtual machine with its Python programming language interpreter
- 6 Computer processor, memory, disk storage and other peripheral devices

Operating System Software	Add-On Software	Central Processing Unit Hardware
Apple Mac OS X (10.4.x-10.7.x etc.)	<ul style="list-style-type: none"> ▪ Python 2.6.x ▪ Python 2.7.x ▪ Python 3.1.x ▪ Python 3.2.x ▪ Parallels Desktop 5 / 6 / 7 for Mac (runs various Linux and Windows Virtual Machines on Mac OS X using shared computer processor, memory, peripheral, and network resources) 	<ul style="list-style-type: none"> ▪ Intel & AMD / PC Compatibles (32-bit & 64-bit) ▪ Freescale PowerPC Compatibles (32-bit & 64-bit)
Linux (Fedora, Mandriva, Red Hat, Susse, Ubuntu etc.)	<ul style="list-style-type: none"> ▪ Python 2.6.x ▪ Python 2.7.x ▪ Python 3.1.x ▪ Python 3.2.x 	<ul style="list-style-type: none"> ▪ Intel & AMD / PC Compatibles (32-bit & 64-bit) ▪ Freescale PowerPC Compatibles (32-bit & 64-bit)
Microsoft Windows (such as 2000, XP, Vista, 7 etc.)	<ul style="list-style-type: none"> ▪ Cygwin 1.7.x, Linux-style operating environment ▪ Python 2.6.x ▪ Python 2.7.x ▪ Python 3.1.x ▪ Python 3.2.x 	<ul style="list-style-type: none"> ▪ Intel & AMD / PC Compatibles (32-bit & 64-bit)
Oracle/Sun Solaris (such as Solaris 9, 10, 11 etc.)	<ul style="list-style-type: none"> ▪ Python 2.6.x ▪ Python 2.7.x ▪ Python 3.1.x ▪ Python 3.2.x 	<ul style="list-style-type: none"> ▪ Intel & AMD / PC Compatibles (32-bit & 64-bit) ▪ Oracle/Sun SPARC or compatibles
Unix (such as FreeBSD, HP-UX, IBM-AIX)	<ul style="list-style-type: none"> ▪ Python 2.6.x ▪ Python 2.7.x ▪ Python 3.1.x ▪ Python 3.2.x 	<ul style="list-style-type: none"> ▪ Intel & AMD / PC Compatibles (32-bit & 64-bit) ▪ Freescale PowerPC Compatibles (32-bit & 64-bit)

4.2.3 Computer User Use Case(s)

From Wikipedia, the free encyclopedia

"In software and systems engineering, a use case is a list of steps, typically defining interactions between a role (known in Unified Modeling Language (UML) as an "actor") and a system, to achieve a goal. The actor can be a human, an external system, or time.

In systems engineering, use cases are used at a higher level than within software engineering, often representing missions or stakeholder goals. The detailed requirements may then be captured in Systems Modeling Language (SysML) or as contractual statements.

As an important requirement technique, use cases have been widely used in modern software engineering over the last two decades. Use case driven development is a key characteristic of process models and frameworks like Unified Process (UP), Rational Unified Process (RUP), Oracle Unified Method (OUM), etc. With its iterative and evolutionary nature, use case is also a good fit for agile development."

On-line documentation for the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit is provided in the subdirectory named:

```

"./<Toolkit Recipient's Repository>/Documents".

<Your Working Repository>
(e.g. "tsWxGTUI_PyVx_Repository")
|
|
+-- ["Documents"]
|   |
|   |   This directory contains a collection of files
|   |   which provide the Toolkit recipient with an
|   |   understanding of the purpose, goals & capabil-
|   |   ities, non-goals & limitations, terms & condi-
|   |   tions and procedures for installing, operating,
|   |   modifying and redistributing the Toolkit.
|   |
+-- ["ManPages"]
|   |
|   |   Deliverable Toolkit manual pages are a
|   |   form of online software documentation
|   |   usually found on a Unix or Unix-like
|   |   operating system.
|   |
|   |   Topics covered include computer programs
|   |   (including library and system calls),
|   |   formal standards and conventions, and even
|   |   abstract concepts.
|   |
+-- ["Notebooks"]
|   |
|   |   Contains a collection of commentaries that
|   |   express opinions or offerings of explana-
|   |   tions about events or situations that might

```

```
|         | be useful to Toolkit installers, developers,  
|         | operators, troubleshooters and distributors.  
|         | The documents may be in Application-specific  
|         | formats (such as Adobe PDF, JPEG Bit-mapped  
|         | image, LibreOffice, Microsoft Office, plain  
|         | text).  
|  
+-- ["SourceDistributions"]  
|  
|         | Contains a collection of computer program  
|         | source code files that the Toolkit recip-  
|         | ient will need to install, operate, modify  
|         | and re-distribute the Toolkit.  
|  
+-- "README.txt"  
|  
|         | Contains information about other files in  
|         | a directory or archive and is commonly  
|         | distributed with computer software, forming  
|         | part of its documentation.  
|  
+-- "MANIFEST.txt"  
|  
|         | Contains a Tally List for deliverable items.
```

4.2.3.1 System Administrator

Based on: http://en.wikipedia.org/wiki/System_administrator

A System Administrator is typically responsible for supervising and/or performing the following:

- Installation, configuration, maintenance, repair and support of all system hardware, operating system and application software and network components.
- Scripting or light programming.
- Project management for systems-related projects.
- Supervising or training computer operators.
- Consultant for computer problems beyond the knowledge of technical support staff.
- To perform his or her job well, a System Administrator must demonstrate a blend of technical skills and responsibility.

The subject matter of System Administration includes computer systems and the ways people use them in an organization. This entails a knowledge of operating systems and applications, as well as hardware and software troubleshooting, but also knowledge of the purposes for which people in the organization use the computers. Perhaps the most important skill for a System Administrator is problem solving -- frequently under various sorts of constraints and stress. The System Administrator is on call when a computer system goes down or malfunctions, and must be able to quickly and correctly diagnose what is wrong and how best to fix it. System Administrators are not Software Engineers. It is not usually within their duties to design or write new application software. However, System Administrators must understand the behavior of software in order to deploy it and to troubleshoot problems, and generally know several programming languages used for scripting or automation of routine tasks.

4.2.3.2 Software Engineer

Derived from: <http://www.bls.gov/oco/ocos303.htm>

Computer Software Engineers design and develop software for use by System Operators. They perform the following:

- Begin by analyzing users' needs and then design, test, and develop software to meet those needs.
- Apply the theories and principles of computer science and mathematical analysis to create, test, and evaluate the systems, application software and command line or graphical user interface that make computers work.
- During this process they create flowcharts, diagrams, and other documentation, and may also create the detailed sets of instructions, called algorithms, that actually tell the computer what to do.
- They may also be responsible for converting these instructions into a computer language, a process called programming or coding, but this usually is the responsibility of computer programmers.

4.2.3.3 System Operator

Derived from <http://whatis.techtarget.com/definition/system-operator-sysop>

A System Operator is the person who runs the day-to-day operation of the computer system. The term suggests a person who is available when the system is.

The System Operator is ultimately the end-user of application programs, with associated command line or graphical user interfaces, that interactively supervise various communication, control, data base, diagnostic, instrumentation or simulation activities.

At earlier stages in the computer system's installation, configuration and software development process, the System Operator may temporarily be a **System Administrator** (on page 318) or **Software Engineer** (on page 319).

At later stages in the computer system's maintenance, the System Operator may temporarily be the **Field Service Engineer**.

The activities are application-specific, system-specific and role-specific.

The *TeamSTARS* "tsWxGTUI_PyVx" Toolkit application-specific use cases are detailed in the downloaded copy of the following document:

- https://GitHub.com/rigordo959/tsWxGTUI_PyVx_Repository/Documents/DEMO.txt

The following use cases illustrate the system specific similarities and differences:

- 1 **Linux Use Case** (on page 321)
- 2 **Mac OS X Use Case** (on page 320)
- 3 Microsoft Windows Use Case(s)
 - a) **Linux-Like (Cygwin) Use Case** (on page 320)

b) DOS-Like Use Case

4 Unix Use Case

4.2.3.3.1 Mac OS X Use Case

Mac OS X (10.3 -10.11) Use Case

Operation of a typical computer system advances through various stages:

- 1** Manual Power-ON of Host Computer Hardware (Booting)
 - 2** Automatic reset of Host Computer Hardware
 - 3** Automatic startup of Host Computer Operating System Software
 - 4** Automatic startup of Host Computer Pixel-mode Graphical User Interface Software (**login**)
 - 5** Optional installation of new or updated Host Computer Operating System Libraries and Software
-

See the file "**GETTING_STARTED.txt**" (in the **"./Documents"** directory) for platform preparation instructions.

- 6** Optional installation of new or updated "tsWxGTUI_PyVx" Toolkit Application Libraries and Software
 - 7** Manual startup of Host Computer Command Line Interface Software (**Applications -> iTerm** or **Applications -> Utilities -> terminal**)
-

For those application programs that require manual termination, it may be necessary for the System Operator to issue an **exit** command (via text from the Command Line Interface or via a BUTTON from the Graphical User Interface) or an **abort** command (via a CTRL-C signal from the Command Line Interface).

- 8** Manual shutdown of Host Computer Command Line Interface Software (**exit**)
- 9** Manual shutdown of Host Computer Pixel-mode Graphical User Interface Software (**logout**)
- 10** Manual shutdown of Host Computer Operating System Software (**shutdown**)
- 11** Manual Power-OFF of Host Computer Hardware (Shutdown)

4.2.3.3.2 Linux-Like (Cygwin) Use Case

Linux-Like (Cygwin 1.7.8 - 2.2.1) Use Case for Microsoft Windows (XP, 7, 8, 8.1, 10)

Use this procedure only AFTER the installation of "Cygwin".

Operation of a typical computer system advances through various stages:

- 1** Manual Power-ON of Host Computer Hardware (Booting)
- 2** Automatic reset of Host Computer Hardware
- 3** Automatic startup of Host Computer Operating System Software
- 4** Automatic startup of Host Computer Pixel-mode Graphical User Interface Software (**login**)
- 5** Optional installation of new or updated Host Computer Operating System Libraries and Software

See the file "**GETTING_STARTED.txt**" (in the "**./Documents**" directory) for platform preparation instructions.

- 6 Optional installation of new or updated "tsWxGTUI_PyVx" Toolkit Application Libraries and Software
 - 7 Manual startup of Host Computer Command Line Interface Software (**Start -> Programs -> Cygwin -> Cygwin Bash Shell**)
-

For those application programs that require manual termination, it may be necessary for the System Operator to issue an **exit** command (via text from the Command Line Interface or via a BUTTON from the Graphical User Interface) or an **abort** command (via a CTRL-C signal from the Command Line Interface).

- 8 Manual shutdown of Host Computer Command Line Interface Software (**exit**)
- 9 Manual shutdown of Host Computer Pixel-mode Graphical User Interface Software (**logout**)
- 10 Manual shutdown of Host Computer Operating System Software (**shutdown**)
- 11 Manual Power-OFF of Host Computer Hardware (Shutdown)

4.2.3.3.3 Linux Use Case

Linux (Debian, Fedora, SuSE, Ubuntu etc.) Use Case

Operation of a typical computer system advances through various stages:

- 1 Manual Power-ON of Host Computer Hardware (Booting)
 - 2 Automatic reset of Host Computer Hardware
 - 3 Automatic startup of Host Computer Operating System Software
 - 4 Automatic startup of Host Computer Pixel-mode Graphical User Interface Software (**login**)
 - 5 Optional installation of new or updated Host Computer Operating System Libraries and Software
-

See the file "**GETTING_STARTED.txt**" (in the "**./Documents**" directory) for platform preparation instructions.

- 6 Optional installation of new or updated "tsWxGTUI_PyVx" Toolkit Application Libraries and Software
 - 7 Manual startup of Host Computer Command Line Interface Software (**Applications -> Accessories -> terminal**)
-

For those application programs that require manual termination, it may be necessary for the System Operator to issue an **exit** command (via text from the Command Line Interface or via a BUTTON from the Graphical User Interface) or an **abort** command (via a CTRL-C signal from the Command Line Interface).

- 8 Manual shutdown of Host Computer Command Line Interface Software (**exit**)
- 9 Manual shutdown of Host Computer Pixel-mode Graphical User Interface Software (**System -> Logout**)
- 10 Manual shutdown of Host Computer Operating System Software (**System -> Shutdown**)

11 Manual Power-OFF of Host Computer Hardware (Shutdown)

Draft

4.3 APPENDIX B - Software Design Constraints

Since each software release shall provide the means to:

- 1** Install in developer-sandbox and/or site-package configurations; and
- 2** Use with Python 2x and/or Python 3x language generations

The design and implementation shall therefore be constrained as described in the following sections.

Draft

4.3.1 README

```
#-----
#"Time-stamp: <06/05/2015  6:31:31 AM rsg>
#-----

===== File: README.txt =====

+-----+-----+ TeamSTARS "tsWxGTUI_PyVx" Toolkit
| ts | Wx |      with Python 2x & Python 3x based
+-----+-----+      Command Line Interface (CLI)
| G T U I |      and "Curses"-based "wxPython"-style,
+-----+-----+      Graphical-Text User Interface (GUI)

Get that cross-platform, pixel-mode "wxPython" feeling
on character-mode 8-/16-color (xterm-family) & non-color
(vt100-family) terminals and terminal emulators.

You can find this and other files in the following sub-
directories:

<Your Working Repository>
(e.g. "Technical_Preview")
|
|   Working repository containing directories and
|   files to be packaged into downloadable "tarball"
|   and/or "zip" files via the setup shell scripts
|   at the bottom of this diagram.
|
+-- ["Documents"] (Original)
|
|   |   This directory contains a collection of files
|   |   which provide the Toolkit recipient with an
|   |   understanding of the purpose, goals & capabil-
|   |   ities, non-goals & limitations, terms & condi-
|   |   tions and procedures for installing, operating,
|   |   modifying and redistributing the Toolkit.
|   |
|   +-- "README.txt"
|   +-- "README1-Introduction.txt"
|   +-- "README2-Repository.txt"
|   +-- "README3-Documents.txt"
|   +-- "README4-ManPages.txt"
|   +-- "README5-Notebooks.txt"
|   +-- "README6-SourceDistributions.txt"
|   +-- "GETTING_STARTED.txt"
|
+-- ["ManPages"] (Original)
|
|   |   Deliverable Toolkit manual pages are a
|   |   form of online software documentation
|   |   usually found on a Unix or Unix-like
|   |   operating system.
|   |
```

```

|   | Topics covered include computer programs
|   | (including library and system calls),
|   | formal standards and conventions, and even
|   | abstract concepts.
|   |
|   | Unlike their Unix or Unix-like counterparts,
|   | a Toolkit user may NOT invoke a man page by
|   | issuing the "man command". Instead, a user
|   | must display a man page by issuing the
|   | "less <man document file>" command.
|   |
|   | +-- ["tsManPagesLibCLI"]
|   | +-- ["tsManPagesLibGUI"]
|   | +-- ["tsManPagesTestsLibCLI"]
|   | +-- ["tsManPagesTestsLibGUI"]
|   | +-- ["tsManPagesToolsCLI"]
|   | +-- ["tsManPagesToolsGUI"]      (Future)
|   | +-- ["tsManPagesToolsLibCLI"]
|   | +-- ["tsManPagesToolsLibGUI"]  (Future)
|   | +-- ["tsManPagesUtilitiesCLI"] (Future)
|   |
|   | +-- "README4-ManPages.txt"
|
+-- ["Notebooks"] (Pre-dates Documents)
|
|   | Contains a collection of commentaries that
|   | express opinions or offerings of explana-
|   | tions about events or situations that might
|   | be useful to Toolkit installers, developers,
|   | operators, troubleshooters and distributors.
|   | The documents may be in Application-specific
|   | formats (such as Adobe PDF, JPEG Bit-mapped
|   | image, LibreOffice, Microsoft Office, plain
|   | text).
|   |
|   | +-- ["DeveloperNotebook"]
|   |
|   |   | Contains a collection of:
|   |   |   API-References-Pixel-Mode-wxPython
|   |   |   and Developer-ReadMe-Files
|   |
|   | +-- ["EngineeringNotebook"]
|   |
|   |   | Contains a Toolkit Developer oriented collection of:
|   |   |   Project (purpose,
|   |   |   |   goals,
|   |   |   |   non-goals,
|   |   |   |   features,
|   |   |   |   capabilities,
|   |   |   |   limitations),
|   |   |   Plan (software life-cycle),
|   |   |   Requirements (purpose,
|   |   |   |   goals,
|   |   |   |   non-goals,
|   |   |   |   features,
|   |   |   |   capabilities,

```



```

|
|   Site-packages is the location where third-
|   party packages are installed (i.e., those
|   not part of the core Python distribution).
|   NOTE: That with Linux, Mac OS X and Unix
|   operating systems one must have root priv-
|   ileages to write to that location.
|
|   +-- ["tsWxGTUI_PyVx"] (Site-Package)
|       |
|       +-- ["Documents"] (Copy)
|       |
|       +-- ["ManPages"] (Copy)
|       |
|       +-- ["Python-2x"] (Site-Package)
|           |
|           +-- ["tsWxGTUI_Py2x"]
|           |
|       +-- ["Python-3x"] (Site-Package,
|                           Ported from Python-2x)
|           |
|           +-- ["tsWxGTUI_Py3x"]
|
+-- "MANIFEST.in"
|
|   Deliverable File inclusion criteria list.
|
+-- "MANIFEST_template.in"
|
|   Deliverable Generic file inclusion criteria list
|   template for any Python version-specific TeamSTARS
|   "tsWxGTUI_PyVx" Toolkit.
|
+-- "MANIFEST_TREE.html"
|
|   Non-Deliverable Diagram (Multi-Level Org Chart)
|   depicting the hierarchical relationship between files
|   in the release, in Hypertext Markup Language format.
|
|   Diagram created via Command "./MANIFEST_TREE.sh".
|
+-- "MANIFEST_TREE.sh"
|
|   Deliverable POSIX-style Command Line Interface shell
|   script to generate diagrams depicting the hierarchical
|   relationship between files in the release
|   ("MANIFEST_TREE.html" and "MANIFEST_TREE.txt").
|
+-- "MANIFEST_TREE.txt"
|
|   Non-Deliverable Diagram (Multi-Level Org Chart)
|   depicting the hierarchical relationship between
|   files in the release, in Plain Text format.
|
|   Diagram created via Command "./MANIFEST_TREE.sh".

```

```
+-- "setup_Technical_Preview_tar_file.sh"
|
|   Deliverable POSIX-style Command Line Interface shell
|   script to generate downloadable "tarball" file.
|
+-- "setup_Technical_Preview_zip_file.sh"
|
|   Deliverable POSIX-style Command Line Interface shell
|   script to generate downloadable "zip" file.
|
+-- "README.txt"
```

===== WELCOME =====

The "tsWxGTUI" Toolkit provides a collection of building-block components, tools, utilities, and tests for creating, enhancing, troubleshooting, maintaining and supporting application programs that are suitable for embedded systems.

1. Application Programs

Automation, communication, control, diagnostic, instrumentation and simulation application programs typically require an "operator-friendly" Command Line Interface (CLI) or a Graphical-style User Interface (GUI) that can be controlled locally or remotely.

2. Embedded Systems

Mission-critical systems for commercial, industrial, medical and military applications are typically customized and optimized for a specific use. Unlike their general-purpose desktop, laptop and workstation counterparts, embedded systems typically have limited, application-specific processing, memory, communication, input/output and file storage resources. Some may have character-mode hardware only suitable for their operating system's command line console.

3. What should you do to get started?

Browse through the following information to get an overview of the distribution and its contents:

- 3.1 `"./tsWxGTUI_PyVx/Documents/README1-Introduction.txt"`
- 3.2 `"./tsWxGTUI_PyVx/Documents/README2-Repository.txt"`
- 3.3 `"./tsWxGTUI_PyVx/Documents/README3-Documents.txt"`
- 3.4 `"./tsWxGTUI_PyVx/Documents/README4-ManPages.txt"`
- 3.5 `"./tsWxGTUI_PyVx/Documents/README5-Notebooks.txt"`
- 3.6 `"./tsWxGTUI_PyVx/Documents/README6-SourceDistribution"`
- 3.7 `"./tsWxGTUI_PyVx/Documents/GETTING_STARTED.txt"`
- 3.8 `"./tsWxGTUI_PyVx/Documents/DEMO.txt"`

4. Experience the features, look and feel of the toolkit by running through the scenarios presented in DEMO.txt file and browsing through the Python source code for the

associated application programs and building blocks.

--

Richard S. Gordon
SoftwareGadgetry@comcast.net

===== End-Of-File =====

Draft

4.3.2 README1-Introduction.txt

```
#-----
#"Time-stamp: <06/05/2015  5:09:51 AM rsg>
#-----

===== File: README1-Introduction.txt =====

+-----+-----+ TeamSTARS "tsWxGTUI_PyVx" Toolkit
| ts | Wx |      with Python 2x & Python 3x based
+-----+-----+      Command Line Interface (CLI)
| G T U I |      and "Curses"-based "wxPython"-style,
+-----+-----+      Graphical-Text User Interface (GUI)

Get that cross-platform, pixel-mode "wxPython" feeling
on character-mode 8-/16-color (xterm-family) & non-color
(vt100-family) terminals and terminal emulators.

You can find this and other plain-text files in the
Toolkit subdirectory named:

    <Your Working Repository>
    (e.g. "Technical_Preview")
    |
    |   Working repository containing directories and
    |   files to be packaged into downloadable "tarball"
    |   and/or "zip" files via the setup shell scripts
    |   at the bottom of this diagram.
    |
    +-- ["Documents"]

===== TABLE OF CONTENTS =====

1. What is it?

    1.1 In a Nutshell
    1.2 The Gory Details

2. How is it implemented?

    2.1 Python Programming Language
    2.2 Python-based Command Line Interface (CLI)
    2.3 "wxPython"-style Graphical-Text User Interface (GUI)

3. What are the System Requirements?

    3.1 Hardware
    3.2 Software

4. The Latest Versions

    4.1 "tsWxGTUI_Py2x-0.0.0" for Python 2.4.1 - 2.7.9
    4.2 "tsWxGTUI_Py3x-0.0.0" for Python 3.0.0 - 3.4.3
```

5. Deliverables

- 5.1 Documentation
- 5.2 Source Code

6. Installation

7. Licensing

8. Contacts

9. Acknowledgments

===== WHAT IS IT? =====

1. What is it?

The "tsWxGTUI" Toolkit provides a collection of building-block components, tools, utilities, and tests.

The Toolkit facilitates the effort of software developers creating enhanceing, troubleshooting, maintaining and supporting application programs with character-mode Command Line and Graphical-style User Interfaces that are suitable for the local and remote monitoring and control of computer systems that are embedded in mission-critical equipment.

1.1 In a Nutshell

The TeamSTARS "tsWxGTUI_PyVx" Toolkit software is engineered to facilitate the development and use of application programs with the following features:

1.1.1 User-friendly Interfaces:

a. Command Line Interface (CLI)

Output to the user of a chronological sequence of lines of text via a scrolling computer terminal display with input from the user via a computer terminal keyboard.

b. Graphical-style User Interface (GUI)

Output to the user of character strings to application-specified column and row (line) fields of a computer terminal display with input from the user via a computer terminal keyboard and pointing device (such as mouse, trackball, touchpad or touchscreen).

1.1.2 General-purpose, portability, maintainability, re-usability, scalability, deployability:

a. Toolkit Applications

Software development and installation toolkit for automation, communication, control, diagnostic, instrumentation and simulation applications.

b. Usage Applications

Computerized mainframe, workstation, desktop, laptop, tablet and embedded systems with 32-bit/64-bit processors from various manufacturers and popular operating systems including GNU/Linux, Mac OS X, Microsoft Windows and Unix.

1.2 The Gory Details

Software development systems typically have sufficient and upgradable resources including 32-bit/64-bit processors, random access memory, non-volatile memory (such as electro-mechanical hard drives and electronic flash memory), network interface devices and operator control consoles with keyboard, mouse and large high-cost pixel-mode/graphics-mode displays that can also operate in character-mode/text-mode.

Embedded systems typically are optimized for specific commercial, industrial, medical and military applications. The more capable systems have 32-bit/64-bit processors, random access memory, non-volatile memory, network interface devices and operator control consoles with keyboard, mouse and small low-cost character-mode/text-mode displays.

The Toolkit provides utilities, tools and a library of building blocks for you to create application programs that can raise the productivity and reduce the applied time of software developers and maintainers by its use of the high-level Python programming language.

It can also raise the productivity of system administrators, software developers, equipment operators and field service personnel by providing a suitable user-friendly interface:

1.2.1 A Command Line Interface (CLI)

Please see encyclopedia-based information at:

[http://en.wikipedia.org/wiki/
Command-line_interface](http://en.wikipedia.org/wiki/Command-line_interface)

1.2.2 A Text-based User Interface (TUI)

The Text-based User Interface (TUI) emulates a subset of the Application Programming Interface (API)

for wxPython, a popular cross-platform Graphical User Interface (GUI). The baseline API for the TUI is that of wxPython 2.8.9.2. Once development has been completed, the API can be updated to track the latest wxPython version.

Please see encyclopedia-based information at:

http://en.wikipedia.org/wiki/Text-based_user_interface

http://en.wikipedia.org/wiki/Graphical_user_interface

It can raise overall productivity through its ability to monitor remote computer systems and associated equipment from the convenience of a local centralized computer system.

The network communication speed of character data will be faster than that of pixel data because:

1. There would be more pixels than character cells involved in programmatic change. For example, on a 307,200 (640x480) pixel display, there are only 3,200 (80 cols x 40 rows) character cells when using a 96 (8x12) pixel font.
2. There would only be 1 byte involved in the following programmatic character cell changes:
 - a) Which of the 256 text characters, and pre-assigned 96 (8x12) pixel combinations, to apply to individual character cells.
 - b) Which of the 256 (16x16) color pairs, and pre-assigned 256x256x256 red-green-blue foreground color intensity levels, and 256x256x256 red-green-blue background color intensity levels to apply to individual strings of one or more character cells.
 - c) Which of the 8 text character attribute combinations (alternate character set, blink mode, bold mode, dim mode, normal attribute, reverse background and foreground colors, standout mode and underline mode) to apply to individual strings of one or more character cells.
3. Alternative Solutions
 - a) The Java programming language user community has access to a small footprint console interface.

See "<http://www.codeproject.com/Articles/328417/Java-Console-apps-made-easy>"

In the absence of any statement about character-mode, it is presumed that the Java console operates only in pixel-mode.

- b) The wxWidgets 3.0 C++ programming language user community has access to a set of console interface classes that were not in wxWidgets 2.8 or 2.9.

See "http://docs.wxwidgets.org/trunk/classwx_app_console.html"

In the absence of any statement about character-mode, it is presumed that the set of "wxWidgets 3.0" console interface classes operate only in standard pixel-mode.

- c) "Urwid"

Excerpt from "<http://urwid.org/manual/overview.html>"

"Urwid is a console user interface library for Python. Urwid offers an alternative to using Python's curses module directly and handles many of the difficult and tedious tasks for you.

../_images/introduction.png

Each Urwid component is loosely coupled and designed to be extended by the user.

Display modules are responsible for accepting user input and converting escape sequences to lists of keystrokes and mouse events. They also draw the screen contents and convert attributes used in the canvases rendered to the actual colors that appear on screen.

The included widgets are simple building blocks and examples that try not to impose a particular style of interface. It may be helpful to think of Urwid as a console widget construction set rather than a finished UI library like GTK or Qt. The Widget base class describes the widget interface and widget layout describes how widgets are nested and arranged on the screen.

Text is the bulk of what will be displayed in any console user interface. Urwid supports a number of text encodings and Urwid comes with a configurable text layout that handles the

most of the common alignment and wrapping modes. If you need more flexibility you can also write your own text layout classes.

Urwid supports a range of common display attributes, including 256-color foreground and background settings, bold, underline and standout settings for displaying text. Not all of these are supported by all terminals, so Urwid helps you write applications that support different color modes depending on what the user's terminal supports and what they choose to enable."

<additional info not reproduced>

d) "npyscreen"

Excerpts from "<https://code.google.com/p/npyscreen/>"

"Npyscreen is a python widget library and application framework for programming terminal or console applications. It is built on top of ncurses, which is part of the standard library."

<additional info not reproduced>

"Strengths

This framework should be powerful enough to create everything from quick, simple programs to complex, multi-screen applications. It is designed to make doing the simple tasks very quick and to take much of the pain out of writing larger applications.

There is a very wide variety of default widgets - everything from simple text fields to more complex tree and grid views.

The framework is easy to extend. That said, if you have a requirement for a widget that is not currently included you can try emailing me and I'll see whether I have time to help - no promises!"

===== HOW IS IT IMPLEMENTED? =====

2. How is it implemented?

2.1 Python Programming Language

From <https://docs.python.org/3/faq/general.html>
 From <https://docs.python.org/2/faq/general.html>

"What is Python:

Python is an interpreted, interactive, object-

oriented programming language. It incorporates modules, exceptions, dynamic typing, very high level dynamic data types, and classes. Python combines remarkable power with very clear syntax. It has interfaces to many system calls and libraries, as well as to various window systems, and is extensible in C or C++. It is also usable as an extension language for applications that need a programmable interface. Finally, Python is portable: it runs on many Unix variants, on the Mac, and on PCs ..."

Python 3x and 2x for 32-bit and 64-bit processors:

"... on Windows 2000 and later."

Python 2x for 16-bit and 32-bit processors:

"... under MS-DOS, Windows, Windows NT, and OS/2."

Python 1x for 16-bit and 32-bit processors

System requirements are no longer available.

From <http://ftp.python.org/download/releases/1.6.1/>

"Python 1.6 was the last of the versions developed at [the Corporation for National Research Initiatives] CNRI and the only version issued by CNRI with an open source license. Following the release of Python 1.6, and after Guido van Rossum left CNRI to work with commercial software developers, it became clear that the ability to use Python with software available under the GNU General Public License (GPL) was very desirable. CNRI and the Free Software Foundation (FSF) interacted to develop enabling wording changes to the Python license. Python 1.6.1 is essentially the same as Python 1.6, with a few minor bug fixes, and with a GPL-compatible license."

2.1.1 The high-level Python programming language is used to implement the TeamSTARS "tsWxGTUI_PyVx" Toolkit.

The TeamSTARS "tsWxGTUI_PyVx" Toolkit's building block and tool components import and use run time library components from the Python Global Module Index and from Python user-installed site-packages.

Python is a popular, field proven, portable, cross-platform programming language.

Please see encyclopedia-based Python information at:

<http://en.wikipedia.org/wiki>

/Python_(programming_language)

2.2 Python-based Command Line Interface (CLI)

2.2.1 The TeamSTARS "tsWxGTUI_PyVx" Toolkit's Command Line Interface building block components import and use the Keyword-Value Pair and Positional argument parsers:

Please see Python Global Module Index-based information at:

<http://docs.python.org/3/library/argparse.html>

<http://docs.python.org/2/library/optparse.html>

<http://docs.python.org/2/library/getopt.html>

Availability:

a) "argparse"

Introduced in Python 2.7 and Python 3.2.

b) "optparse"

Introduced in Python 2.3 (deprecated in Python 2.7) and Python 3.0 (deprecated in Python 3.2).

c) "getopt"

Available in Python 1.6, 2.0 and Python 3.0.

2.3 "wxPython"-style Graphical-Text User Interface (GUI)

1.3.1 The TeamSTARS "tsWxGTUI_PyVx" Toolkit's Text-based User Interface components import and use the Terminal handler ("curses") for character-cell displays available in Python's Global Module Index.

It uses "curses" to emulate a subset of the Application Programming Interface (API) of "wxPython", a wrapper to the popular "wxWidgets" pixel-mode, cross-platform Graphical User Interface Toolkit which is implemented in the C++ programming language.

The "wxPython" emulation retains the pixel-mode parameters of the "wxPython" API and mimics the look and feel of Microsoft "Windows XP" Displays which are similar to

those of Linux "GTK+" Displays:

- a) GUI container features such as frames, dialogs and panels and buttons to close, iconize and maximize/restore the container.
- b) GUI control features such as buttons, checkboxes, radio buttons, scroll bars, scroll lists and status bars.
- c) GUI layout features such as box sizer and grid sizer.
- d) GUI operator notification features such as a scrolling log of date and time stamped event notification messages.
- e) GUI desktop features such as task bar buttons to control GUI container focus.

Please see encyclopedia-based information at:

<http://en.wikipedia.org/wiki/WxPython>

<http://en.wikipedia.org/wiki/WxWidgets>

<http://en.wikipedia.org/wiki/GTK%2B>

Please see Python Global Module Index-based information at:

<http://docs.python.org/2/library/curses.html>

<http://docs.python.org/3/library/curses.html>

===== WHAT ARE THE SYSTEM REQUIREMENTS? =====

3. What are the System Requirements?

The design of the TeamSTARS "tsWxGTUI_PyVx" Toolkit supports a wide range of possible system configurations.

Cross-platform Python virtual machine technology is often available for Intel and non-Intel 32-bit and 64-bit processor based systems running proprietary and non-proprietary operating systems.

For example, Toolkit development and testing has involved the following system configurations:

3.1 Hardware

The TeamSTARS "tsWxGTUI_PyVx" Toolkit development and testing platforms have involved three classes of

equipment represented by the following:

a. Minimal Usability and Performance

1998-model year, 366 MHz Intel Pentium II-based Dell Inspiron 7000 laptop with 384 MB RAM, 640x480/1024x768 pixel resolution display and dual PCMCIA Card expansion capabilities for network and peripheral device interfaces (with marginal resources and performance) sufficient enough to serve as the low-end, single user baseline development and operator platform.

Its interchangeable 32 GB (4200 RPM) ATA hard drives were used to run either:

- * Microsoft Windows Desktop with Cygwin, the free GNU/Linux-like Plug-in from Red Hat (XP Pro); or
- * Ubuntu GNU/Linux Desktop (12.04)

The platform's limited memory and available PCMCIA network adapters were incompatible with later versions of Windows or with other Linux distributions. (Windows XP recognized Xircom Ethernet and 3Com Wireless adapters; Ubuntu Linux 12.04 recognized only Linksys Wireless adapter.

b. Moderate Usability and Performance

2007-model year, 2.33 GHz Intel Core 2 Duo processor-based Apple 17" MacBook Pro laptop with 4 GB RAM, 1920x1200 pixel resolution display and sufficient performance, resources and expansion capabilities to serve as the mid-range, single user baseline development and operator platform.

Its 160 GB (5400 RPM) SATA 1.5 Gb/s internal hard drive was used to run Apple's Mac OS X 10.7 and the hypervisor virtualization applications (Parallels Desktop 8 and VMware Fusion 5) that supported various guest operating systems.

Its 1.5 TB (7200 RPM) SATA 3 Gb/s external hard drive was used to store and concurrently run an assortment of up to two configured versions selected (for normal use rather than for stress-testing) from the following guest operating systems:

- * Fedora GNU/Linux Desktop (17)
- * Ubuntu GNU/Linux Desktop (12.04)
- * Microsoft Windows Desktop with Cygwin, the free GNU/Linux-like Plug-in from Red Hat

(XP Pro, 7 Pro, 8 Pro and 8.1 Pro)

- * OpenIndiana (OpenSolaris 11-based) Unix Desktop (151a6)

NOTES:

- 1) A Seagate ST31500341AS 1.5TB SATA 7200 RPM was used as the external hard drive. The wear and tear from using a consumer product (over 8-10 hours a day, 7 days each week for 7 years) for the Guest Operating System swapfile and data storage ultimately wore it out (unrecoverable disk head crash).
- 2) Subsequent research indicated that a more appropriate hard drive would have been be an Hitachi Ultrastar 7K3000 HUA723020ALA641 2TB 7200RPM 64MB Cache SATA 6.0Gb/s 3.5" Enterprise Hard Drive -OEM.

c. High Usability and Performance

2013-model year, 3.5 GHz Intel Quad Core i7 processor-based Apple 27" iMac desktop with 16 GB RAM, 2560x1440 pixel resolution display and sufficient performance, resources and expansion capabilities to serve as the high-end, multi-user baseline development and operator platform.

Its 3 TB (7200 RPM) SATA 6 Gb/s internal hard drive had 128 GB Solid State Flash memory and was used to run Apple's Mac OS X 10.9-10.10 and the hypervisor virtualization applications (Parallels Desktop 9-10 and VMware Fusion 5 and 7) that supported various guest operating systems.

Its 3 TB (7200 RPM) SATA 6 Gb/s internal hard drive was normally used to store and concurrently run an assortment of up to four configured versions selected (for normal use rather than for stress-testing) from the following guest operating systems:

- * CentOS GNU/Linux Desktop (7.0)
- * Debian GNU/Linux Desktop (8) and Server (Turnkey 13.0) Configured with Apache, Bugzilla and MySQL.
- * Fedora GNU/Linux Desktop (20-22)
- * OpenSUSE GNU/Linux Desktop (13.1)
- * Scientific GNU/Linux Desktop (6.5 & 7.0)
- * Ubuntu GNU/Linux Desktop (12.04 & 14.04)

- * Microsoft Windows Desktop with Cygwin, the free GNU/Linux-like Plug-in from Red Hat (XP Pro, 7 Pro, 8 Pro, 8.1 Pro and 10 Technical Preview)
- * PC-BSD (FreeBSD-based) Unix Desktop (9.2 & 10.0)
- * OpenIndiana (OpenSolaris 11-based) Unix Desktop (151a8)

3.2 Software

The TeamSTARS "tsWxGTUI_PyVx" Toolkit development and testing platforms have involved an assortment of single and multi-user, multi-process and multi-threaded POSIX-compatible operating system releases.

The following operating system information consists of annotated excerpts From Wikipedia, the free encyclopedia, in order to preserve a snapshot of relevant content that might otherwise be subject to change and lose relevance:

a. GNU/Linux

GNU is a Unix-like computer operating system developed by the GNU Project, ultimately aiming to be a "complete Unix-compatible software system" composed wholly of free software.

Linux is a Unix-like and mostly POSIX-compliant computer operating system assembled under the model of free and open-source software development and distribution. The defining component of Linux is the Linux kernel, an operating system kernel first released on 5 October 1991 by Linus Torvalds. The Free Software Foundation uses the name GNU/Linux to describe the operating system, which has led to some controversy.

Many computer users run a modified version of the GNU system every day, without realizing it. Through a peculiar turn of events, the version of GNU which is widely used today is often called "Linux", and many of its users are not aware that it is basically the GNU system, developed by the GNU Project.

Please see encyclopedia-based information at:

<http://en.wikipedia.org/wiki/GNU>
<http://en.wikipedia.org/wiki/Linux>
http://en.wikipedia.org/wiki/Linux_kernel

- * CentOS, a distribution derived from the same sources used by Red Hat, maintained by a dedicated volunteer community of developers with both 100% Red Hat-compatible versions and an

upgraded version that is not always 100% upstream compatible. (7.0)

- * Debian, a non-commercial distribution and one of the earliest, maintained by a volunteer developer community with a strong commitment to free software principles and democratic project management
- * Fedora, a community distribution sponsored by American company Red Hat. (17-22)
- * OpenSuSE, a community distribution mainly sponsored by German company SuSE. (13.1)
- * SuSE Linux Enterprise, derived from openSuSE, is maintained and commercially supported by SuSE.
- * Red Hat Enterprise Linux, a derivative of Fedora, maintained and commercially supported by Red Hat
- * Scientific, a Linux distribution produced by Fermi National Accelerator Laboratory. It is a free and open source operating system based on Red Hat Enterprise Linux and aims to be "as close to the commercial enterprise distribution as we can get it.". (6.5 and 7.0)

This product is derived from the free and open source software made available by Red Hat, Inc., but is not produced, maintained or supported by Red Hat. Specifically, this product is built from the source code for Red Hat Enterprise Linux versions, under the terms and conditions of Red Hat Enterprise Linux's EULA and the GNU General Public License.

- * Ubuntu, a popular desktop and server distribution derived from Debian, maintained by British company Canonical Ltd. (12.04 LTS and 14.04 LTS)
- b. Microsoft Windows, a metafamily of graphical operating systems developed, marketed, and sold by Microsoft. It consists of several families of operating systems, each of which cater to a certain sector of the computing industry. Active Windows families include Windows NT, Windows Embedded and Windows Phone; these may encompass subfamilies, e.g. Windows Embedded Compact (Windows CE) or Windows Server. Defunct Windows families include Windows 9x and Windows Mobile.

Microsoft Windows will only require "Cygwin", the free Linux-like plug-in from Red Hat for users of the "wxPython"-style, "Curses"-based Graphical-Text User Interface. (Home or Professional editions of Windows XP, 7, 8, 8.1, 10 Technical Preview)

Microsoft introduced an operating environment named Windows on November 20, 1985 as a graphical operating system shell for MS-DOS in response to the growing interest in graphical user interfaces (GUIs). Microsoft Windows came to dominate the world's personal computer market with over 90% market share, overtaking Mac OS, which had been introduced in 1984. However, it is outsold by Android on smartphones and tablets.

- c. OS X (formerly known as Mac OS X), a series of Unix-based graphical interface operating systems developed and marketed by Apple Inc. It is designed to run on Mac computers. (10.4 "Tiger"-10.10 "Yosemite")

NOTES: From http://en.wikipedia.org/wiki/OS_X

"The first releases of Mac OS X from 1999 to 2006 can run only on the PowerPC based Macs of the period. After Apple announced it would shift to using Intel x86 CPUs from 2006 onwards, Tiger and Leopard were released in versions for Intel and PowerPC processors. Snow Leopard is the first version released only for Intel Macs. Since the release of Mac OS X 10.7 "Lion", OS X has dropped support for 32-bit Intel processors as well. It now runs exclusively on 64-bit Intel CPUs."

Mac OS X Version	PowerPC Platform (NOT tested with Toolkit)
10.0: "Cheetah"	(32-bit PowerPC)
10.1: "Puma"	(32-bit PowerPC)
10.2: "Jaguar"	(32-bit PowerPC)
10.3: "Panther"	(32-bit PowerPC)
10.4: "Tiger"	(32-bit PowerPC and Intel)
10.5: "Leopard"	(32-bit PowerPC and Intel)

Mac OS X Version	Intel Platforms (tested with Toolkit)
10.4: "Tiger"	(32-bit PowerPC and Intel)
10.5: "Leopard"	(32-bit PowerPC and Intel)
10.6: "Snow Leopard"	(32-bit Intel)
10.7: "Lion"	(32-bit Intel)
10.8: "Mountain Lion"	(64-bit Intel)
10.9: "Mavericks"	(64-bit Intel)
10.10: "Yosemite"	(64-bit Intel)

Versions 10.5 "Leopard" running on Intel processors, 10.6 "Snow Leopard", 10.7 "Lion", 10.8 "Mountain Lion", 10.9 "Mavericks", and 10.10 "Yosemite" have obtained UNIX 03 certification.

iOS, which runs on the iPhone, iPod Touch, iPad, and the 2nd and 3rd generation Apple TV, shares the Darwin core and many frameworks with OS X.

- d. Unix, a multitasking, multiuser computer operating system that exists in many variants. The original Unix was developed at AT&T's Bell Labs research center by Ken Thompson, Dennis Ritchie, and others. From the power user's or programmer's perspective, Unix systems are characterized by a modular design that is sometimes called the "Unix philosophy," meaning the OS provides a set of simple tools that each perform a limited, well-defined function, with a unified filesystem as the main means of communication and a shell scripting and command language to combine the tools to perform complex workflows.
- * FreeBSD, a free Unix-like operating system descended from Research Unix via the Berkeley Software Distribution (BSD). Although for legal reasons FreeBSD cannot use the Unix trademark, it is a direct descendant of BSD, which was historically also called "BSD Unix" or "Berkeley Unix." (10.0)
- * OpenIndiana, a free and open-source, Unix operating system derived from OpenSolaris. Developers forked OpenSolaris after Oracle Corporation discontinued it, in order to continue development and distribution of the source code. The OpenIndiana project is stewarded by the illumos Foundation, which also stewards the illumos operating system. OpenIndiana's developers strive to make it "the defacto OpenSolaris distribution installed on production servers where security and bug fixes are required free of charge". (151a8)
- * OpenSolaris, a descendant of the UNIX System V Release 4 (SVR4) code base developed by Sun and AT&T in the late 1980s. It is the only version of the System V variant of UNIX available as open source.
- * PC-BSD, or PCBSD, a Unix-like, desktop-oriented operating system built upon the most recent releases of FreeBSD. It aims to be easy to install by using a graphical installation program, and easy and ready-to-use immediately by providing KDE SC, LXDE, Xfce, and MATE as the graphical user interface. (10.0)

===== THE LATEST VERSIONS =====

4. The Latest Versions

The latest TeamSTARS "tsWxGTUI_PyVx" Toolkit version is a pre-alpha stage, pre-production release identified as:

4.1 "tsWxGTUI_Py2x-0.0.0" for Python 2.4.1 - 2.7.9

4.2 "tsWxGTUI_Py3x-0.0.0" for Python 3.0.0 - 3.4.3

===== DELIVERABLES =====

5. Deliverables

Deliverables for the TeamSTARS "tsWxGTUI_PyVx" Toolkit include the following:

5.1 Documentation

5.1.1 Documentation in Plain Text Format

The TeamSTARS "tsWxGTUI_PyVx" Toolkit documentation is included, in plain text format, in the directory named:

./tsWxGTUI_PyVx/Documents

5.1.2 Documentation in HTML Format

Since the TeamSTARS "tsWxGTUI_PyVx" Toolkit emulates a character-mode compatible subset of the wxPython and wxWidgets pixel-mode GUI Application Programming Interface (API), the documentation includes archive copies of pixel-mode API in its Hypertext Markup Language format.

The archive copies are provided because the original On-Line versions are no longer available on the wxWidgets and wxPython web sites.

5.2 Source Code

Excerpt From Wikipedia, the free encyclopedia:

"Python is an open source programming language that was made to both look good and be easy to read. It was created by a programmer named Guido van Rossum in 1991. The language is named after the television show Monty Python's Flying Circus and many examples and tutorials include jokes from the show.

Python is an interpreted language. An interpreted language allows the programmer to give the source code to the computer and the computer runs the code right away. This means if the programmer needs to change the code they can quickly see the results. This makes Python a good programming language for beginners and for making programs rapidly because you do not have to compile the code to make it run,

and compiling takes a lot of time. But because the computer has to figure out what the code does every time the code runs, Python is a very slow language. Sometimes, it can be 200 times slower than C [programming language].

Python is also a high-level programming language. A high-level language has advanced features which let the programmer to tell the computer what to do without having to worry about how the computer is going to do that as much as low-level programming languages. This makes writing programs easier and faster. Some of the rules of how you write code in Python are taken from C, and Python can run some C code."

Since Python is not a conventional compiled language, its language syntax does not include compiler directives to conditionally compile language version specific features. Consequently there must be separate source code directories and files for Python 2x and Python 3x.

While many language features are common to Python 2x and Python 3x:

- a) obsolescent ones may be deprecated, available for a limited time (like the print statements and old-style classes introduced in Python 1x that were retained only in Python 2x) but not recommended;
- b) obsolete ones ultimately disappear (like the print statements and old-style classes syntax no longer in Python 3.x); and
- c) enhanced ones may be introduced (like the print function syntax introduced in Python 2x and the new except statement syntax introduced in Python 3.x).

That being said, cross-platform regression testing on various Linux, Mac OS X, Microsoft Windows (with and without the free, Linux-like Cygwin plug-in) and Unix has established the set of Python 2x and Python 3x versions which fully support the Toolkit's Command Line Interface and Graphical User Interface.

Release of the source code enables Toolkit users to customize the source code so as to support older or newer Python versions and platforms.

===== INSTALLATION =====

6. Installation

Please see the file named:

./Documents/INSTALL.txt

===== LICENSING =====

7. Licensing

Please see the file named:

./Documents/LICENSE.txt

===== CONTACTS =====

8. Contacts

Technical Support Requests should be directed via email sent to:

SoftwareGadgetry@comcast.net

===== ACKNOWLEDGMENTS =====

9. Acknowledgments

Please see files with the following names:

./Documents/AUTHORS.txt

./Documents/COPYRIGHT.txt

./Documents/CREDITS.txt

./Documents/THANKS.txt

===== End-Of-File =====

4.3.3 README2-Repository.txt

```
#-----
#"Time-stamp: <06/03/2015  6:46:09 AM rsg>
#-----

===== File: README2-Repository.txt =====

+-----+-----+ TeamSTARS "tsWxGTUI_PyVx" Toolkit
| ts | Wx |      with Python 2x & Python 3x based
+-----+-----+      Command Line Interface (CLI)
| G T U I |      and "Curses"-based "wxPython"-style,
+-----+-----+      Graphical-Text User Interface (GUI)

Get that cross-platform, pixel-mode "wxPython" feeling
on character-mode 8-/16-color (xterm-family) & non-color
(vt100-family) terminals and terminal emulators.

You can find this and other files in the following sub-
directories:

<Your Working Repository>
(e.g. "Technical_Preview")
|
|   Working repository containing directories and
|   files to be packaged into downloadable "tarball"
|   and/or "zip" files via the setup shell scripts
|   at the bottom of this diagram.
|
+-- ["Documents"]
|
|   |   This directory contains a collection of files
|   |   which provide the Toolkit recipient with an
|   |   understanding of the purpose, goals & capabil-
|   |   ities, non-goals & limitations, terms & condi-
|   |   tions and procedures for installing, operating,
|   |   modifying and redistributing the Toolkit.
|   |
|   +-- "README.txt"
|   +-- "README1-Introduction.txt"
|   +-- "README2-Repository.txt"
|   +-- "README3-Documents.txt"
|   +-- "README4-ManPages.txt"
|   +-- "README5-Notebooks.txt"
|   +-- "README6-SourceDistributions.txt"
|
+-- ["ManPages"]
|
|   |   Deliverable Toolkit manual pages are a
|   |   form of online software documentation
|   |   usually found on a Unix or Unix-like
|   |   operating system.
|   |
|   |   Topics covered include computer programs
```

```

|         | (including library and system calls),
|         | formal standards and conventions, and even
|         | abstract concepts.
|
|         | Unlike their Unix or Unix-like counterparts,
|         | a Toolkit user may NOT invoke a man page by
|         | issuing the "man command". Instead, a user
|         | must display a man page by issuing the
|         | "less <man document file>" command.
|
|         |
| --- ["tsManPagesLibCLI"]
| --- ["tsManPagesLibGUI"]
| --- ["tsManPagesTestsLibCLI"]
| --- ["tsManPagesTestsLibGUI"]
| --- ["tsManPagesToolsCLI"]
| --- ["tsManPagesToolsGUI"]          (Future)
| --- ["tsManPagesToolsLibCLI"]
| --- ["tsManPagesToolsLibGUI"]      (Future)
| --- ["tsManPagesUtilitiesCLI"]     (Future)
|
| --- "README4-ManPages.txt"
|
+-- ["Notebooks"]                      (Pre-dates Documents)
|
|         | Contains a collection of commentaries that
|         | express opinions or offerings of explana-
|         | tions about events or situations that might
|         | be useful to Toolkit installers, developers,
|         | operators, troubleshooters and distributors.
|         | The documents may be in Application-specific
|         | formats (such as Adobe PDF, JPEG Bit-mapped
|         | image, LibreOffice, Microsoft Office, plain
|         | text).
|
|         |
| --- ["DeveloperNotebook"]
|
|         | Contains a collection of:
|         |     API-References-Pixel-Mode-wxPython
|         |     and Developer-ReadMe-Files
|
|         |
| --- ["EngineeringNotebook"]
|
|         | Contains a Toolkit Developer oriented collection of:
|         |     Project (purpose,
|         |             goals,
|         |             non-goals,
|         |             features,
|         |             capabilities,
|         |             limitations),
|         |     Plan (software life-cycle),
|         |     Requirements (purpose,
|         |             goals,
|         |             non-goals,
|         |             features,
|         |             capabilities,
|         |             limitations,

```



```

|       |       | Topics covered include computer programs
|       |       | (including library and system calls),
|       |       | formal standards and conventions, and even
|       |       | abstract concepts.
|
|       |       | A user may NOT invoke a man page by issu-
|       |       | ing the man command. Instead, a user may
|       |       | display a man page by issuing the
|       |       | less <man document file> command.
|
|       |       | +-- ["tsManPagesLibCLI"]
|       |       | +-- ["tsManPagesLibGUI"]
|       |       | +-- ["tsManPagesTestsLibCLI"]
|       |       | +-- ["tsManPagesTestsLibGUI"]
|       |       | +-- ["tsManPagesToolsCLI"]
|       |       | +-- ["tsManPagesToolsGUI"] (Future)
|       |       | +-- ["tsManPagesToolsLibCLI"]
|       |       | +-- ["tsManPagesToolsLibGUI"] (Future)
|       |       | +-- ["tsManPagesUtilitiesCLI"] (Future)
|
| +-- ["Python-2x"]
|       |
|       | +-- ["tsWxGTUI_Py2x"]
|
| +-- ["Python-3x"] (Ported from Python-2x)
|       |
|       | +-- ["tsWxGTUI_Py3x"]
|
+-- ["Site-Packages"]
|
|   Site-packages is the location where third-
|   party packages are installed (i.e., those
|   not part of the core Python distribution).
|   NOTE: That with Linux, Mac OS X and Unix
|   operating systems one must have root priv-
|   ileages to write to that location.
|
+-- ["tsWxGTUI_PyVx"]
|
+-- ["Documents"]
|
|   This directory contains a collection of files
|   which provide the Toolkit recipient with an
|   understanding of the purpose, goals & capabil-
|   ities, non-goals & limitations, terms & condi-
|   tions and procedures for installing, operating,
|   modifying and redistributing the Toolkit.
|
+-- ["ManPages"]
|
|   Deliverable Toolkit manual pages are a
|   form of online software documentation
|   usually found on a Unix or Unix-like oper-
|   ating system.

```



```

|     depicting the hierarchical relationship between
|     files in the release, in Plain Text format.
|
|     Diagram created via Command "./MANIFEST_TREE.sh".
|
+-- "setup_Technical_Preview_tar_file.sh"
|
|     Deliverable POSIX-style Command Line Interface shell
|     script to generate downloadable "tarball" file.
|
+-- "setup_Technical_Preview_zip_file.sh"
|
|     Deliverable POSIX-style Command Line Interface shell
|     script to generate downloadable "zip" file.
|
+-- "README.txt"

```

===== TABLE OF CONTENTS =====

1. Repository

1.1 Documents

1.2 ManPages

1.3 Notebooks-Site-Packages

1.3.1 Preview-Project

1.3.2 Preview-Site-Packages

1.4 SourceDistributions-Site-Packages

1.4.1 tsWxGTUI_Py2x

1.4.2 tsWxGTUI_Py3x

1.5 Manifest

===== REPOSITORY =====

1. Repository

Excerpt From Wikipedia, the free encyclopedia:

"A software repository is a storage location from which software packages may be retrieved and installed on a computer."

This is the repository for the TeamSTARS "tsWxGTUI_PyVx" Toolkit. It is the collection of documentation and computer program source code files that is being distributed by its author in order to publish and share the intellectual property with others.

It contains the following subdirectories and files:

1.1 Documents

Contains introductory and other training information for installers, developers, operators, troubleshooters and distributors.

Toolkit software documentation is written in plain text that accompanies computer software. It either explains how it operates or how to use it, and may mean different things to people in different roles. Types of software documentation include:

- a) Requirements - Statements that identify attributes, capabilities, characteristics, or qualities of a system. This is the foundation for what shall be or has been implemented.
- b) Architecture/Design - Overview of software. Includes relations to an environment and construction principles to be used in design of software components.
- c) Technical - Documentation of code, algorithms, interfaces, and APIs.
- d) End user - Manuals for the end-user, system administrators and support staff.

1.2 ManPages

Contains a collection of man pages. A man page (short for manual page) is a form of online software documentation usually found on a Unix or Unix-like operating system. Topics covered include computer programs (including library and system calls), formal standards and conventions, and even abstract concepts.

Categories of man pages include:

- a) tsManPagesLibCLI
- b) tsManPagesLibGUI
- c) tsManPagesTestsLibCLI
- d) tsManPagesTestsLibGUI
- e) tsManPagesTestsToolsLibCLI
- f) tsManPagesTestsToolsLibGUI (Future)
- g) tsManPagesToolsCLI
- h) tsManPagesToolsGUI (Future)
- i) tsManPagesUtilities (Future)

1.3 Notebooks

Contains a collection of commentaries that express opinions or offerings of explanations about events or situations that might be useful to Toolkit installers, developers, operators, troubleshooters and distributors. The documents may be in Application-specific formats (Adobe PDF, JPEG Bit-mapped image,

Microsoft Office, Plain text etc.).

The collection includes:

- a) DeveloperDocuments (API-References-Pixel-Mode-wxPython and Developer-ReadMe-Files); and
- b) EngineeringDocuments (Product Marketing Documentation, Project Documentation and Technical Documentation).

1.3.1 Preview-Project

1.3.2 Preview-Site-Packages

1.4 SourceDistributions-Site-Packages

Contains a collection of computer program source code files that the Toolkit recipient will need to recreate the Toolkit.

The collection includes:

1.4.1 tsWxGTUI_Py2x

The source code files appropriate for use with Python 2.4-2.7.

1.4.2 tsWxGTUI_Py3x

The source code files appropriate for use with Python 3.0-3.4. These files are generated by converting their Python 2x counterparts with the 2to3 translation utility followed by debugging of unresolved syntax and type conversion issues.

A site-package is the location where third-party packages will be installed (i.e., those not part of the core Python distribution that includes os, sys, platform, curses, logging etc.).

----- NOTES:

- a) With Linux, Mac OS X and Unix operating systems one must have "root" or administrator privileges to write to the site-package location.
- b) If the user will not have permission to directly access the Repository but has a need to know specific contents, the system administrator should copy the appropriate contents of the Documents, ManPages and Notenook directories into each Site-Package.

The copies were not included in the distribution in order to:

- (1) avoid increasing the release development and qualification efforts; and
 - (2) minimize the size of the downloadable "tar" and "zip" files.
- c) Users of third-party site-packages must explicitly import via its path from top-level package through lower-level packages to module:

site-package.package.module

- d) Examples for Python 2.4-2.7 site-packages:

```
from tsWxGTUI_Py2x.tsLibCLI import tsCxGlobals
from tsWxGTUI_Py2x.tsLibCLI import tsPlatformRunTimeEnvironment
from tsWxGTUI_Py2x.tsLibCLI import tsExceptions as tse
from tsWxGTUI_Py2x.tsLibCLI import tsLogger

from tsWxGTUI_Py2x.tsLibGUI import tsWx as wx
```

- d) Examples for Python 3.0-3.4 site-packages:

```
from tsWxGTUI_Py3x.tsLibCLI import tsCxGlobals
from tsWxGTUI_Py3x.tsLibCLI import tsPlatformRunTimeEnvironment
from tsWxGTUI_Py3x.tsLibCLI import tsExceptions as tse
from tsWxGTUI_Py3x.tsLibCLI import tsLogger

from tsWxGTUI_Py3x.tsLibGUI import tsWx as wx
```

1.5 Manifest

For a listing of the repository contents (complete with last modified date, time, size and access permissions), see:

```
"./MANIFEST_TREE.html"
"./MANIFEST_TREE.txt"
```

For additional commentary on the repository contents see:

```
"./Documents/README-Manifest.txt"
```

===== End-Of-File =====

4.3.4 README3-Documents.txt

```
#-----
#"Time-stamp: <06/03/2015  7:05:59 AM rsg>"
#-----

===== File: README3-Documents.txt =====

+-----+-----+ TeamSTARS "tsWxGTUI_PyVx" Toolkit
| ts | Wx |      with Python 2x & Python 3x based
+-----+-----+      Command Line Interface (CLI)
| G T U I |      and "Curses"-based "wxPython"-style,
+-----+-----+      Graphical-Text User Interface (GUI)

Get that cross-platform, pixel-mode "wxPython" feeling
on character-mode 8-/16-color (xterm-family) & non-color
(vt100-family) terminals and terminal emulators.

You can find this and other files in the following
Toolkit subdirectories:

<Your Working Repository>
(e.g. "Technical_Preview")
|
|   Working repository containing directories and
|   files to be packaged into downloadable "tarball"
|   and/or "zip" files via the setup shell scripts
|   at the bottom of this diagram.
|
+-- ["Documents"]
|
|   |   This directory contains a collection of files
|   |   which provide the Toolkit recipient with an
|   |   understanding of the purpose, goals & capabil-
|   |   ities, non-goals & limitations, terms & condi-
|   |   tions and procedures for installing, operating,
|   |   modifying and redistributing the Toolkit.
|   |
|   +-- "AUTHORS.txt"
|   +-- "BUGS.txt"
|   +-- "CHANGE_LOG.txt"
|   +-- "CONFIGURE.txt"
|   +-- "COPYING.txt"
|   +-- "COPYRIGHT.txt"
|   +-- "CREDITS.txt"
|   +-- "DEMO.txt"
|   +-- "FAQ.txt"
|   +-- "GETTING_STARTED.txt"
|   +-- "INSTALL.txt"
|   +-- "LICENSE.txt"
|   +-- "NEWS.txt"
|   +-- "NOTICES.txt"
|   +-- "OPERATE.txt"
|   +-- "README.txt"
```

```

|      +--- "README1-Introduction.txt"
|      +--- "README2-Repository.txt"
|      +--- "README3-Documents.txt"
|      +--- "README4-ManPages.txt"
|      +--- "README5-Notebooks.txt"
|      +--- "README6-SourceDistributions.txt"
|      +--- "THANKS.txt"
|      +--- "TO-DO.txt"
|      +--- "TROUBLESHOOT.txt"
|
+--- ["ManPages"]
|
+--- ["Notebooks"]
|
+--- ["SourceDistributions"]
|
+--- "README.txt"

===== ["DOCUMENTS"] =====

This directory contains a collection of files which provide the Toolkit recipient with an understanding of the purpose, goals & capabilities, non-goals & limitation, terms & conditions and procedures and for installing, operating, modifying and redistributing the Toolkit.

"GETTING_STARTED.txt" --- Introduces new recipients to
                           the system requirements and
                           third-party resources available to new Toolkit users.

"README.txt" --- Introduces new recipients to the purpose, goals, non-goals, design and features of the computer software product. Supplements include the following:

                           "README1-Introduction.txt"
                           "README2-Repository.txt"
                           "README3-Documents.txt"
                           "README4-ManPages.txt"
                           "README5-Notebooks.txt"
                           "README6-SourceDistributions.txt"

"AUTHORS.txt" --- List of the principal "tsWxGTUI" Toolkit author(s) and authors credited for work covered by a prior copyright and license.

"BUGS.txt" --- List of Known Problems / Issues.

"CHANGE_LOG.txt" --- List of Additions, Modification and Deletions.

"CONFIGURE.txt" --- Instructions for applying factory and site-specific configurations.

```


"COPYING.txt" --- Instructions for copying all or a portion of the distribution.

"FAQ.txt" --- Answers to Frequently Asked Questions.

"INSTALL.txt" --- Describes steps to download, extract install and configure the "tsWxGUI" Toolkit.

"LICENSE.txt" --- General and special arrangements, provisions, rules, specifications and standards that form an integral part of the agreement or contract between the creator and recipient of Copyrighted and Licensed Work.

"MANIFEST.txt" --- Tally List for deliverable items.

"NEWS.txt" --- Announcements of new releases.

"NOTICES.txt" --- Details the copyright(s) and license(s).

"OPERATE.txt" --- Describes steps to use the "tsWxGUI" Toolkit.

"THANKS.txt" --- Acknowledgments to those otherwise unsung heroes who contributed time and effort to supporting the authors as planners, editors, designers, coders and testers.

"TO-DO.txt" --- A To-Do-List provides a roadmap for development and troubleshooting work.

"TROUBLESHOOT.txt" --- Provides a list of available reference resources and a guide for planning, developing and troubleshooting a cross-platform system of hundreds of files each containing a few, tens or hundred of class, data and method definitions. Its complexity becomes apparent in the recent software Lines-Of-Code metrics.

===== End-Of-File =====

4.3.5 README4-ManPages.txt

```
#-----
#"Time-stamp: <06/03/2015  7:40:26 AM rsg>"
#-----

===== File: README4-ManPages.txt =====

+-----+-----+ TeamSTARS "tsWxGTUI_PyVx" Toolkit
| ts | Wx |      with Python 2x & Python 3x based
+-----+-----+      Command Line Interface (CLI)
| G T U I |      and "Curses"-based "wxPython"-style,
+-----+-----+      Graphical-Text User Interface (GUI)

Get that cross-platform, pixel-mode "wxPython" feeling
on character-mode 8-/16-color (xterm-family) & non-color
(vt100-family) terminals and terminal emulators.

You can find this and other files in the following
Toolkit subdirectories:

<Your Working Repository>
(e.g. "Technical_Preview")
|
+-- ["Documents"]
|
+-- ["ManPages"]
|
|   Deliverable Toolkit manual pages are a
|   form of online software documentation
|   usually found on a Unix or Unix-like oper-
|   ating system.
|
|   Topics covered include computer programs
|   (including library and system calls),
|   formal standards and conventions, and even
|   abstract concepts.
|
|   A user may NOT invoke a man page by issu-
|   ing the man command. Instead, a user may
|   display a man page by issuing the
|   less <man document file> command.
|
+-- "README4-ManPages.txt"
|
+-- ["tsManPagesLibCLI"]
|
|   +-- "runPydoc_tsManPagesCLI.sh"
|   +-- "runPylint_tsManPagesCLI.sh"
|   +-- "tsApplication.man"
|   +-- "tsCommandLineEnv.man"
|   +-- "tsCommandLineInterface.man"
|   +-- "tsCxGlobals.man"
|   +-- "tsDoubleLinkedList.man"
```

```

|         |         | +-- "tsExceptions.man"
|         |         | +-- "tsGistGetTerminalSize.man"
|         |         | +-- "tsLogger.man"
|         |         | +-- "tsOperatorSettingsParser.man"
|         |         | +-- "tsPlatformRunTimeEnvironment.man"
|         |         | +-- "tsReportUtilities.man"
|         |         | +-- "tsSysCommands.man"
|         |         |
|         |         | +-- ["tsManPagesLibGUI"]
|         |         | |
|         |         | | +-- "runPydoc_tsManPagesGUI.sh"
|         |         | | +-- "runPylint_tsManPagesGUI.sh"
|         |         | | +-- "tsWx.man"
|         |         | | +-- "tsWxAcceleratorEntry.man"
|         |         | | +-- "tsWxAcceleratorTable.man"
|         |         | | +-- "tsWxApp.man"
|         |         | | +-- "tsWxBoxSizer.man"
|         |         | | +-- "tsWxButton.man"
|         |         | | +-- "tsWxCallLater.man"
|         |         | | +-- "tsWxCaret.man"
|         |         | | +-- "tsWxCheckBox.man"
|         |         | | +-- "tsWxChoice.man"
|         |         | | +-- "tsWxColor.man"
|         |         | | +-- "tsWxColorDatabase.man"
|         |         | | +-- "tsWxControl.man"
|         |         | | +-- "tsWxControlWithItems.man"
|         |         | | +-- "tsWxCursor.man"
|         |         | | +-- "tsWxDebugHandlers.man"
|         |         | | +-- "tsWxDialog.man"
|         |         | | +-- "tsWxDialogButton.man"
|         |         | | +-- "tsWxDisplay.man"
|         |         | | +-- "tsWxDoubleLinkedList.man"
|         |         | | +-- "tsWxEraseEvent.man"
|         |         | | +-- "tsWxEvent.man"
|         |         | | +-- "tsWxEventDaemon.man"
|         |         | | +-- "tsWxEventLoop.man"
|         |         | | +-- "tsWxEventLoopActivator.man"
|         |         | | +-- "tsWxEventQueueEntry.man"
|         |         | | +-- "tsWxEventTableEntry.man"
|         |         | | +-- "tsWxEvtHandler.man"
|         |         | | +-- "tsWxFlexGridSizer.man"
|         |         | | +-- "tsWxFocusEvent.man"
|         |         | | +-- "tsWxFrame.man"
|         |         | | +-- "tsWxFrameButton.man"
|         |         | | +-- "tsWxGauge.man"
|         |         | | +-- "tsWxGlobals.man"
|         |         | | +-- "tsWxGraphicalTextUserInterface.man"
|         |         | | +-- "tsWxGridBagSizer.man"
|         |         | | +-- "tsWxGridSizer.man"
|         |         | | +-- "tsWxItemContainer.man"
|         |         | | +-- "tsWxKeyboardState.man"
|         |         | | +-- "tsWxKeyEvent.man"
|         |         | | +-- "tsWxListBox.man"
|         |         | | +-- "tsWxLog.man"
|         |         | | +-- "tsWxMenu.man"
|         |         | | +-- "tsWxMenuBar.man"

```



```

|
|
|   +-- ["tsManPagesTestsLibGUI"]
|   |
|   |   +-- "buildManPagesTestsGUI.sh"
|   |   +-- "test_tsWxBoxSizer.man"
|   |   +-- "test_tsWxCheckBox.man"
|   |   +-- "test_tsWxDisplay.man"
|   |   +-- "test_tsWxDoubleLinkedList.man"
|   |   +-- "test_tsWxGlobals.man"
|   |   +-- "test_tsWxGraphicalTextUserInterface.man"
|   |   +-- "test_tsWxGridSizer.man"
|   |   +-- "test_tsWxMultiFrameEnv.man"
|   |   +-- "test_tsWxRSM.man"
|   |   +-- "test_tsWxScrolledWindow.man"
|   |   +-- "test_tsWxScrolledWindowDual.man"
|   |   +-- "test_tsWxSplashScreen.man"
|   |   +-- "test_tsWxWidgets.man"
|   |
|   +-- ["tsManPagesToolsCLI"]
|   |
|   |   +-- "buildManPagesToolsCLI.sh"
|   |   +-- "runPydoc_tsManPagesToolsCLI.sh"
|   |   +-- "runPylint_tsManPagesToolsCLI.sh"
|   |   +-- "tsLinesOfCodeProjectMetrics.man"
|   |   +-- "tsPlatformQuery.man"
|   |   +-- "tsStripComments.man"
|   |   +-- "tsStripLineNumbers.man"
|   |   +-- "tsTreeCopy.man"
|   |   +-- "tsTreeTrimLines.man"
|   |
|   +-- ["tsManPagesToolsGUI"] (Future)
|   |
|   |   +-- To-Be-Determined
|   |
|   +-- ["tsManPagesToolsLibCLI"]
|   |
|   |   +-- To-Be-Determined
|   |
|   +-- ["tsManPagesToolsLibGUI"] (Future)
|   |
|   |   +-- To-Be-Determined
|   |
|   +-- ["tsManPagesUtilitiesCLI"] (Future)
|   |
|   |   +-- To-Be-Determined
|   |
|
+-- ["Notebooks"]
|
+-- ["SourceDistributions"]
|
+-- "README.txt"

```

===== TABLE OF CONTENTS =====

1. ["ManPages"]

2. How to create and install a manpage (Future)

===== ["ManPages"] =====

1. ["ManPages"]

The following defines the purpose and use of a set of on-line reference documents. This Toolkit provides utility scripts that:

- a) create rudimentary ManPages from source code;
- b) do NOT yet merge Toolkit ManPages with ManPages installed by the host computer operating system or with the installation of other third-pary add-ons.

Excerpt From Wikipedia, the free encyclopedia:

"A man page (short for manual page) is a form of online software documentation usually found on a Unix or Unix-like operating system. Topics covered include computer programs (including library and system calls), formal standards and conventions, and even abstract concepts. A user may invoke a man page by issuing the man command.

By default, man typically uses a terminal pager program such as more or less to display its output.

Usage

To read a manual page for a Unix command, type:

```
man <command_name>
```

Pages are traditionally referred to using the notation "name(section)": for example, ftp(1). The same page name may appear in more than one section of the manual, such as when the names of system calls, user commands, or macro packages coincide. Examples are man(1) and man(7), or exit(2) and exit(3).

The syntax for accessing the non-default manual section varies between different man implementations. On Solaris, for example, the syntax for reading printf(3C) is:

```
man -s 3c printf
```

On Linux and BSD derivatives the same invocation would be:

```
man 3 printf
```

which searches for printf in section 3 of the man pages."

===== "How to create and install a manpage" =====

2. How to create and install a manpage (Future)

The following may be useful for a future Toolkit enhancement.

Excerpts from Google Search:

"HowTo: Linux / UNIX Create a Manpage - nixCraft
www.cyberciti.biz/faq/linux-unix-creating-a-manpage/
 May 6, 2010 - How do I create a man page for my shell
 or python script under Linux / UNIX ...
 install -g 0 -o 0 -m 0644 nuseradd.1 /usr/local/man/man8/ gzip ...

How to create a manpage? - Ask Ubuntu
askubuntu.com/questions/42923/how-to-create-a-manpage
 Ask Ubuntu
 May 15, 2011 - With the help of Gmanedit · Install
 gmanedit you are able to create manpages with a
 graphical GUI. Gtk+ Manpages Editor is an editor for man ...

Linux Man Page Howto - Who is Jens Schweikhardt?
www.schweikhardt.net/man_page_howto.html
 The next decision is the directory in which it will
 finally be installed (say, when the user runs `make
 install` for your package.) On Linux, all man pages
 are below ...

Linux Howtos: System -> Creating Your Own MAN Page
www.linuxhowtos.org/system/creatingman.htm
 We will be using groff macros to create our manual page.
 These macros always ... Normally you put the version
 number of your program here. [center header]

How can I add man page entries for my own power tools?
unix.stackexchange.com/.../how-can-i-add-man-page-ent...
 Stack Exchange
 Feb 4, 2011 - I have no idea about how I can make my
 home-grown specialist scripts ... and you can create
 a man page from the POD file with the pod2man ... You
 can then optionally (b|g)zip it and put it in the
 appropriate man directory.

How to add entry in Linux man page database - Stack ...
stackoverflow.com/.../how-to-add-entry-in-linux-man-page-database
 Dec 25, 2012 - I have a manual page for mongoose web
 server named as mongoose.1 as a result of doing make
 and make install command to install ...

How should a formatted man page look?
www.tldp.org/HOWTO/Man-Page/q3.html
 Linux Documentation Project
 Here comes the man page for the (hypothetical)
 foo program. ... However, if you install using
 'make prefix=/opt/gnu' the references in the man
 page change to ..."

===== End-Of-File =====

Draft

4.3.6 README5-Notebooks.txt

```
#-----
#"Time-stamp: <06/03/2015  6:50:16 AM rsg>"
#-----

===== File: README5-Notebooks.txt =====

+-----+-----+ TeamSTARS "tsWxGTUI_PyVx" Toolkit
| ts | Wx |      with Python 2x & Python 3x based
+-----+-----+      Command Line Interface (CLI)
| G T U I |      and "Curses"-based "wxPython"-style,
+-----+-----+      Graphical-Text User Interface (GUI)

Get that cross-platform, pixel-mode "wxPython" feeling
on character-mode 8-/16-color (xterm-family) & non-color
(vt100-family) terminals and terminal emulators.

<Your Working Repository>
(e.g. "Technical_Preview")
|
|   Working repository containing directories and
|   files to be packaged into downloadable "tarball"
|   and/or "zip" files via the setup shell scripts
|   at the bottom of this diagram.
|
+-- ["Documents"]
|
|   |   This directory contains a collection of files
|   |   which provide the Toolkit recipient with an
|   |   understanding of the purpose, goals & capabil-
|   |   ities, non-goals & limitations, terms & condi-
|   |   tions and procedures for installing, operating,
|   |   modifying and redistributing the Toolkit.
|   |
|   +-- "README.txt"
|   +-- "README1-Introduction.txt"
|   +-- "README2-Repository.txt"
|   +-- "README3-Documents.txt"
|   +-- "README4-ManPages.txt"
|   +-- "README5-Notebooks.txt"
|   +-- "README6-SourceDistributions.txt"
|
+-- ["ManPages"]
|
|   |   Deliverable Toolkit manual pages are a
|   |   form of online software documentation
|   |   usually found on a Unix or Unix-like
|   |   operating system.
|   |
|   |   Topics covered include computer programs
|   |   (including library and system calls),
|   |   formal standards and conventions, and even
|   |   abstract concepts.
```

```
|
|
|   Unlike their Unix or Unix-like counterparts,
|   a Toolkit user may NOT invoke a man page by
|   issuing the "man command". Instead, a user
|   must display a man page by issuing the
|   "less <man document file>" command.
|
|
|   +-- ["tsManPagesLibCLI"]
|   +-- ["tsManPagesLibGUI"]
|   +-- ["tsManPagesTestsLibCLI"]
|   +-- ["tsManPagesTestsLibGUI"]
|   +-- ["tsManPagesToolsCLI"]
|   +-- ["tsManPagesToolsGUI"]      (Future)
|   +-- ["tsManPagesToolsLibCLI"]
|   +-- ["tsManPagesToolsLibGUI"]  (Future)
|   +-- ["tsManPagesUtilitiesCLI"] (Future)
|
|   +-- "README4-ManPages.txt"
|
+-- ["Notebooks"]      (Pre-dates Documents)
|
|   Contains a collection of commentaries that
|   express opinions or offerings of explanations
|   about events or situations that might
|   be useful to Toolkit installers, developers,
|   operators, troubleshooters and distributors.
|   The documents may be in Application-specific
|   formats (such as Adobe PDF, JPEG Bit-mapped
|   image, LibreOffice, Microsoft Office, plain
|   text).
|
|   +-- ["DeveloperNotebook"]
|
|       Contains a collection of:
|       API-References-Pixel-Mode-wxPython
|       and Developer-ReadMe-Files
|
+-- ["EngineeringNotebook"]
|
|       Contains a Toolkit Developer oriented collection of:
|       Project (purpose,
|               goals,
|               non-goals,
|               features,
|               capabilities,
|               limitations),
|       Plan (software life-cycle),
|       Requirements (purpose,
|                     goals,
|                     non-goals,
|                     features,
|                     capabilities,
|                     limitations,
|                     file system configuration,
|                     hardware & software interface,
|                     software,
```

```

|         system,
|         user configuration options),
|         Design (API emulation strategy, architecture),
|         Implementation (developer-sandbox, site-package),
|         Test (unit, integration, system, acceptance),
|         Marketing (announcement, brochure),
|         Release (introduction,
|             release notes,
|             software user's manual,
|             terms & conditions,
|             dictionary),
|         Third-party Resources
|
+-- ["ProjectNotebook"]
|
|     Contains a Toolkit User oriented collection of:
|     ["EngineeringNotebook"] abstracts
|
+-- "README5-Notebooks.txt"
|
+-- ["SourceDistributions"]
|
|     Contains a collection of computer program
|     source code files that the Toolkit recip-
|     ient will need to install, operate, modify
|     and re-distribute the Toolkit.
|
+-- "README6-SourceDistributions.txt"
|
+-- ["Developer-Sandboxes"] (Pre-dates Site-Packages)
|
|     A sandbox is a testing environment that iso-
|     lates untested code changes and outright
|     experimentation from the production environ-
|     ment or repository.
|
+-- ["tsWxGTUI_PyVx"]
|
|     +-- ["Documents"]
|     |
|     |     This directory contains a collection of files
|     |     which provide the Toolkit recipient with an
|     |     understanding of the purpose, goals & capabil-
|     |     ities, non-goals & limitations, terms & condi-
|     |     tions and procedures for installing, operating,
|     |     modifying and redistributing the Toolkit.
|     |
+-- ["ManPages"]
|
|     |
|     |     Deliverable Toolkit manual pages are a
|     |     form of online software documentation
|     |     usually found on a Unix or Unix-like oper-
|     |     ating system.
|     |
|     |     Topics covered include computer programs
|     |     (including library and system calls),

```



```

|         |         | abstract concepts.
|         |         |
|         |         | A user may NOT invoke a man page by issu-
|         |         | ing the man command. Instead, a user may
|         |         | display a man page by issuing the
|         |         | less <man document file> command.
|         |         |
|         |         | +-- ["tsManPagesLibCLI"]
|         |         | +-- ["tsManPagesLibGUI"]
|         |         | +-- ["tsManPagesTestsLibCLI"]
|         |         | +-- ["tsManPagesTestsLibGUI"]
|         |         | +-- ["tsManPagesToolsCLI"]
|         |         | +-- ["tsManPagesToolsGUI"]          (Future)
|         |         | +-- ["tsManPagesToolsLibCLI"]
|         |         | +-- ["tsManPagesToolsLibGUI"]    (Future)
|         |         | +-- ["tsManPagesUtilitiesCLI"] (Future)
|         |         |
|         |         | +-- ["Python-2x"]
|         |         |         |
|         |         |         | +-- ["tsWxGTUI_Py2x"]
|         |         |         |
|         |         | +-- ["Python-3x"] (Ported from Python-2x)
|         |         |         |
|         |         |         | +-- ["tsWxGTUI_Py3x"]
|         |         |
| +-- "MANIFEST.in"
|
|     Deliverable File inclusion criteria list.
|
| +-- "MANIFEST_template.in"
|
|     Deliverable Generic file inclusion criteria list
|     template for any Python version-specific TeamSTARS
|     "tsWxGTUI_PyVx" Toolkit.
|
| +-- "MANIFEST_TREE.html"
|
|     Non-Deliverable Diagram (Multi-Level Org Chart)
|     depicting the hierarchical relationship between files
|     in the release, in Hypertext Markup Language format.
|
|     Diagram created via Command "./MANIFEST_TREE.sh".
|
| +-- "MANIFEST_TREE.sh"
|
|     Deliverable POSIX-style Command Line Interface shell
|     script to generate diagrams depicting the hierarchical
|     relationship between files in the release
|     ("MANIFEST_TREE.html" and "MANIFEST_TREE.txt").
|
| +-- "MANIFEST_TREE.txt"
|
|     Non-Deliverable Diagram (Multi-Level Org Chart)
|     depicting the hierarchical relationship between
|     files in the release, in Plain Text format.
|

```

```
|      Diagram created via Command "./MANIFEST_TREE.sh".
|
+-- "setup_Technical_Preview_tar_file.sh"
|
|      Deliverable POSIX-style Command Line Interface shell
|      script to generate downloadable "tarball" file.
|
+-- "setup_Technical_Preview_zip_file.sh"
|
|      Deliverable POSIX-style Command Line Interface shell
|      script to generate downloadable "zip" file.
|
+-- "README.txt"
```

===== TABLE OF CONTENTS =====

- 1. ["DeveloperNotebook"]
 - 1.1 ["API-References-Pixel-Mode-wxPython"] (Future)
 - 1.2 ["DeveloperReadMeFiles"] (Future)
- 2. ["EngineeringNotebook"]
 - 2.1 Product Marketing Documentation (Future)
 - 2.2 Project Documentation
 - 2.3 Technical Documentation
- 3. ["ProjectNotebook"]
 - 3.1 ["API-Preview"]
 - 3.2 ["API-References-Pixel-Mode-wxPython"] (Future)
 - 3.3 ["DeveloperReadMeFiles"] (Future)

===== ["DeveloperNotebook"] =====

- 1. ["DeveloperNotebook"]

This directory contains a collection of files which provide the Toolkit Building Block, Tool and Application programmer with an understanding of the design and usage of the Toolkit's CLI and GUI Application Programming Interface.

 - 1.1 ["API-References-Pixel-Mode-wxPython"]
 - 1.2 ["DeveloperReadMeFiles"]

===== ["EngineeringNotebook"] =====

- 2. ["EngineeringNotebook"]

This directory contains a collection of files, in application-specific formats, which provide the Toolkit architect, product manager, project engineer, system engineer, software engineer, test engineer and troubleshooter with:

2.1 Product Marketing Documentation

- a) announcement notice(s)
- b) brochure(s)
- c) introduction

2.2 Project Documentation

- a) goal and capability objectives
- b) non-goal and limitation constraints
- c) project plans

2.3 Technical Documentation

- a) architectural plans
- b) system specifications
- c) Interface specifications
- d) software specifications
- e) design specifications
- f) test specifications
- g) software user manual

===== ["ProjectNotebook"] =====

3. ["ProjectNotebook"]

===== End-Of-File =====

4.3.7 README6-SourceDistributions.txt

```
#-----
#"Time-stamp: <06/05/2015  4:25:31 AM rsg>"
#-----

===== File: README6-SourceDistributions.txt =====

+-----+-----+ TeamSTARS "tsWxGTUI_PyVx" Toolkit
| ts | Wx |      with Python 2x & Python 3x based
+-----+-----+      Command Line Interface (CLI)
| G T U I |      and "Curses"-based "wxPython"-style,
+-----+-----+      Graphical-Text User Interface (GUI)

Get that cross-platform, pixel-mode "wxPython" feeling
on character-mode 8-/16-color (xterm-family) & non-color
(vt100-family) terminals and terminal emulators.

You can find this and other files in the following sub-
directories:

<Your Working Repository>
(e.g. "Technical_Preview")
|
|   Working repository containing directories and
|   files to be packaged into downloadable "tarball"
|   and/or "zip" files via the setup shell scripts
|   at the bottom of this diagram.
|
+-- ["Documents"]
|   |
|   |   This directory contains a collection of files
|   |   which provide the Toolkit recipient with an
|   |   understanding of the purpose, goals & capabil-
|   |   ities, non-goals & limitations, terms & condi-
|   |   tions and procedures for installing, operating,
|   |   modifying and redistributing the Toolkit.
|   |
|   +-- "README3-Documents.txt"
|
+-- ["ManPages"]
|   |
|   |   Deliverable Toolkit manual pages are a
|   |   form of online software documentation
|   |   usually found on a Unix or Unix-like oper-
|   |   ating system.
|   |
|   +-- "README4-ManPages.txt"
|
+-- ["Notebooks"]
|   |
|   |   Contains a collection of commentaries that
|   |   express opinions or offerings of explana-
|   |   tions about events or situations that might
```



```

|         | be useful to Toolkit installers, developers,
|         | operators, troubleshooters and distributors.
|         | The documents may be in Application-specific
|         | formats (Adobe PDF, JPEG Bit-mapped image,
|         | Microsoft Office, Plain text etc.).
|
|         +--- "README5-Notebooks.txt"
|
+--- ["SourceDistributions"]
|
|         | Contains a collection of computer program
|         | source code files that the Toolkit recip-
|         | ient will need to install, operate, modify
|         | and re-distribute the Toolkit.
|
|         +--- "README6-SourceDistributions.txt"
|
+--- ["Developer-Sandboxes"] (Pre-dates Site-Packages)
|
|         | A sandbox is a testing environment that iso-
|         | lates untested code changes and outright experi-
|         | mentation from the production environment or
|         | repository.
|
|         +--- ["tsWxGTUI_PyVx"]
|         |
|         |         | Contains one or more Python language gener-
|         |         | ation-specific releases each sharing the same
|         |         | programmer (API) and user (CLI & GUI) inter-
|         |         | faces, documents and manual pages.
|         |
|         |         +--- ["Documents"] (copy)
|         |         |
|         |         +--- ["ManPages"] (copy)
|         |         |
|         |         +--- ["Python-2x"]
|         |         |
|         |         |         | Second generation Python programming
|         |         |         | language.
|         |         |
|         |         |         +--- ["tsWxGTUI_Py2x"]
|         |         |         |
|         |         |         |         +--- ["tsDemoArchive"]
|         |         |         |         |
|         |         |         |         +--- ["src"]
|         |         |         |         |
|         |         |         |         +--- "TermsAndConditions.txt"
|         |         |         |
|         |         |         +--- ["tsLibCLI"]
|         |         |         |
|         |         |         |         +--- ["tsApplicationPkg"]
|         |         |         |         |
|         |         |         |         |         +--- ["src"]
|         |         |         |         |         +--- ["test"]
|         |         |         |
|         |         |         +--- ["tsCommandLineEnvPkg"]

```


Draft

```
|
|   operating systems one must have root priv-
|   ileages to write to that location.
|
+-- ["tsWxGTUI_PyVx"]
|
|   +-- ["Documents"] (copy)
|   |
|   +-- ["ManPages"] (copy)
|   |
+-- ["Python-2x"]
|   |
|   |   Second generation Python programming
|   |   language.
|   |
|   +-- ["tsWxGTUI_Py2x"]
|   |   |
|   |   +-- ["tsDemoArchive"]
|   |   |   |
|   |   |   +-- ["tsTestsLibCLI"]
|   |   |   +-- ["tsTestsLibGUI"]
|   |   |   +-- ["tsTestsToolsCLI"]
|   |   |   +-- ["tsTestsToolsGUI"]
|   |   |   +-- ["tsTestsToolsLibCLI"]
|   |   |   +-- ["tsTestsToolsLibGUI"]
|   |   |
|   |   +-- ["tsLibCLI"]
|   |   |
|   |   +-- ["tsLibGUI"]
|   |   |
|   |   +-- ["tsToolsCLI"]
|   |   |
|   |   +-- ["tsToolsGUI"]
|   |   |
|   |   +-- ["tsUtilities"]
|   |
+-- ["Python-3x"] (Ported from Python-2x)
|   |
|   |   Third generation Python programming
|   |   language.
|   |
|   +-- ["tsWxGTUI_Py3x"]
|
+-- "README.txt"
```

===== TABLE OF CONTENTS =====

1. Source Code

2. Developer-Sandbox

2.1 tsLibCLI	(equivalent to 3.1.1 tsTestsLibCLI)
2.2 tsLibGUI	(equivalent to 3.1.2 tsTestsLibGUI)
2.3 tsToolsCLI	(equivalent to 3.1.3 tsTestsToolsCLI and 3.1.5 tsTestsToolsLibCLI)
2.4 tsToolsGUI	(equivalent to 3.1.4 tsTestsToolsGUI)
2.5 tsUtilities	(equivalent to 3.8 tsUtilities)

3. Site-Package

3.1 tsDemoArchive

- 3.1.1 tsTestsLibCLI
- 3.1.2 tsTestsLibGUI
- 3.1.3 tsTestsToolsCLI
- 3.1.4 tsTestsToolsGUI
- 3.1.5 tsTestsToolsLibCLI
- 3.1.6 tsTestsToolsLibGUI

3.2 tsLibCLI

3.3 tsLibGUI

3.4 tsToolsCLI

3.5 tsToolsGUI

3.6 tsToolsLibCLI

3.7 tsToolsLibGUI

3.8 tsUtilities

===== SourceDistribution =====

1. Source Code

Excerpt From Wikipedia, the free encyclopedia:

"In computing, source code is any collection of computer instructions (possibly with comments) written using some human-readable computer language, usually as text. The source code of a program is specially designed to facilitate the work of computer programmers, who specify the actions to be performed by a computer mostly by writing source code. The source code is often transformed by a compiler program into low-level machine code understood by the computer. The machine code might then be stored for execution at a later time. Alternatively, an interpreter can be used to analyze and perform the outcomes of the source code program directly on the fly.

Most computer applications are distributed in a form that includes executable files, but not their source code. If the source code were included, it would be useful to a user, programmer, or system administrator, who may wish to modify the program or to understand how it works.

Aside from its machine-readable forms, source code also appears in books and other media; often in the form of small code snippets, but occasionally complete code bases; a well-known case is the source code of PGP."

===== DEVELOPER-SANDBOX =====

2. Developer-Sandbox

Excerpt From Wikipedia, the free encyclopedia:

"A sandbox is a testing environment that isolates untested code changes and outright experimentation from the production environment or repository, in the context of software development including Web development and revision control. Sandboxing protects "live" servers and their data, vetted source code distributions, and other collections of code, data and/or content, proprietary or public, from changes that could be damaging (regardless of the intent of the author of those changes) to a mission-critical system or which could simply be difficult to revert. Sandboxes replicate at least the minimal functionality needed to accurately test the programs or other code under development (e.g. usage of the same environment variables as, or access to an identical database to that used by, the stable prior implementation intended to be modified; there are many other possibilities, as the specific functionality needs vary widely with the nature of the code and the application[s] for which it is intended.)

The concept of the sandbox (sometimes also called a working directory, a test server or development server) is typically built into revision control software such as CVS and Subversion (SVN), in which developers "check out" a copy of the source code tree, or a branch thereof, to examine and work on. Only after the developer has (hopefully) fully tested the code changes in their own sandbox should the changes be checked back into and merged with the repository and thereby made available to other developers or end users of the software.[1]

By further analogy, the term "sandbox" can also be applied in computing and networking to other temporary or indefinite isolation areas, such as security sandboxes and search engine sandboxes (both of which have highly specific meanings), that prevent incoming data from affecting a "live" system (or aspects thereof) unless/ until defined requirements or criteria have been met."

Unlike the contents of the installable site-package, this sandbox uses a multi-level tree of subdirectories and associated files whose topology is defined by a set of package "__init__.py" files which collaborate in performing dynamic path generation and importing of modules and subpackages. Applications import individual packages and individual modules simply by name (if module name is unique) or by package.module name (if module name is not unique).

Source Code Manifest Checklist:

2.1 tsLibCLI	(equivalent to 3.1.1 tsTestsLibCLI)
2.2 tsLibGUI	(equivalent to 3.1.2 tsTestsLibGUI)
2.3 tsToolsCLI	(equivalent to 3.1.3 tsTestsToolsCLI and 3.1.5 tsTestsToolsLibCLI)
2.4 tsToolsGUI	(equivalent to 3.1.4 tsTestsToolsGUI)
2.5 tsUtilities	(equivalent to 3.8 tsUtilities)

===== SITE-PACKAGE =====

3. Site-Package

Site-packages is the location where third-party packages are installed (i.e., those not part of the core Python distribution). NOTE: That with Linux, Mac OS X and Unix operating systems one must have root privileges to write to that location.

Unlike the contents of the Developer-Sandbox, the third-party site-package and its users must explicitly import via the site-package.package.module path identifier.

```
<Your Working Repository>
(e.g. "Technical_Preview")
|
|   Working repository containing directories and
|   files to be packaged into downloadable "tarball"
|   and/or "zip" files via the setup shell scripts
|   at the bottom of this diagram.
|
+-- ["Documents"]
|   |
|   |   This directory contains a collection of files
|   |   which provide the Toolkit recipient with an
|   |   understanding of the purpose, goals & capabilities,
|   |   non-goals & limitations, terms & conditions and
|   |   procedures for installing, operating, modifying and
|   |   redistributing the Toolkit.
|   |
|   +-- "README3-Documents.txt"
+-- ["ManPages"]
|   |
|   |   Deliverable Toolkit manual pages are a form of online
|   |   software documentation usually found on a Unix or Unix-
|   |   like operating system.
|   |
|   +-- "README4-ManPages.txt"
+-- ["Notebooks"]
|   |
|   |   Contains a collection of commentaries that express
|   |   opinions or offerings of explanations about events or
|   |   situations that might be useful to Toolkit installers,
|   |   developers, operators, troubleshooters and distributors.
|   |   The documents may be in Application-specific formats
|   |   (Adobe PDF, JPEG Bit-mapped image, Microsoft Office,
|   |   Plain text etc.).
|   |
|   +-- "README5-Notebooks.txt"
```

```
+-- ["SourceDistributions"]
|
|   Contains a collection of computer program
|   source code files that the Toolkit recip-
|   ient will need to install, operate, modify
|   and re-distribute the Toolkit.
|
+-- "README6-SourceDistributions.txt"
|
+-- ["Developer-Sandboxes"] (Pre-dates Site-Packages)
|
|   A sandbox is a testing environment that iso-
|   lates untested code changes and outright
|   experimentation from the production environ-
|   ment or repository.
|
+-- ["tsWxGTUI_PyVx"]
|
|   +-- ["Documents"] (copy)
|   |
|   +-- ["ManPages"] (copy)
|   |
|   +-- ["Python-2x"]
|   |   |
|   |   +-- ["tsWxGTUI_Py2x"]
|   |   |
|   |   +-- ["Python-3x"] (Ported from Python-2x)
|   |   |
|   |   +-- ["tsWxGTUI_Py3x"]
|   |
+-- ["Site-Packages"]
|
|   Site-packages is the location where third-
|   party packages are installed (i.e., those
|   not part of the core Python distribution).
|   NOTE: That with Linux, Mac OS X and Unix
|   operating systems one must have root priv-
|   ileages to write to that location.
|
+-- ["tsWxGTUI_PyVx"]
|
|   +-- ["Documents"] (copy)
|   |
|   +-- ["ManPages"] (copy)
|   |
|   +-- ["Python-2x"]
|   |   |
|   |   +-- ["tsWxGTUI_Py2x"]
|   |   |
|   |   +-- ["tsDemoArchive"]
|   |   |
|   |   +-- ["tsTestsLibCLI"]
|   |   +-- ["tsTestsLibGUI"]
|   |   +-- ["tsTestsToolsCLI"]
|   |   +-- ["tsTestsToolsGUI"]
|   |   +-- ["tsTestsToolsLibCLI"]
```



```
| | | | +-- ["t  
| | | | +-- ["t  
| | | | +-- ["  
| | | | |  
| | | | +-- "Te  
| | | |  
+-- ["tsLibCD  
+-- ["tsLibGU  
+-- ["tsTools  
+-- ["tsTools  
+-- ["tsTools  
+-- ["tsTools  
+-- ["tsUtili
```

3.1 tsDemoArchive

3.1.1 tsTestsLibCLI

- ```
3.1.1.1 test_TermsAndConditions.py
3.1.1.2 test_tsApplication.py
3.1.1.3 test_tsCommandLineEnv.py
3.1.1.4 test_tsCommandLineInterface.py
3.1.1.5 test_tsCxGlobals.py
3.1.1.6 test_tsDoubleLinkedList.py
3.1.1.7 test_tsException.py
```

- 3.1.1.8 test\_tsLogger.py
- 3.1.1.9 test\_tsOperatorSettingsParser.py
- 3.1.1.10 test\_tsPlatformRunTimeEnvironment.py
- 3.1.1.11 test\_tsReportUtility.py
- 3.1.1.12 test\_tsSysCommands.py

### 3.1.2 tsTestsLibGUI

- 3.1.2.1 test\_cursesOverlappingPanels.py
- 3.1.2.2 test\_cursesPanels.py
- 3.1.2.3 test\_tsWxBoxSizer.py
- 3.1.2.4 test\_tsWxCheckBox.py
- 3.1.2.5 test\_tsWxColorPalette.py
- 3.1.2.6 test\_tsWxDisplay.py
- 3.1.2.7 test\_tsWxDoubleLinkedList.py
- 3.1.2.8 test\_tsWxGlobals.py
- 3.1.2.9 test\_tsWxGraphicalTextUserInterface.py
- 3.1.2.10 test\_tsWxGridSizer.py
- 3.1.2.11 test\_tsWxMultiFrameEnv.py
- 3.1.2.12 test\_tsWxPasswordEntryDialog.py
- 3.1.2.13 test\_tsWxRSM.py
- 3.1.2.14 test\_tsWxScrolledWindow.py
- 3.1.2.15 test\_tsWxScrolledWindowDual.py
- 3.1.2.16 test\_tsWxSplashScreen.py
- 3.1.2.17 test\_tsWxSystemSettings.py
- 3.1.2.17 test\_tsWxTextEntryDialog.py
- 3.1.2.18 test\_tsWxWidgets.py

### 3.1.3 tsTestsToolsCLI

- 3.1.3.1 ["basicFileTypes"]
- 3.1.3.2 ["regressionTestFileTypes"]
- 3.1.3.3 ["sampleHelp"]
- 3.1.3.4 "File\_Extensions.txt"
- 3.1.3.5 tsLinesOfCodeProjectMetrics.py
- 3.1.3.6 tsPlatformQuery.py
- 3.1.3.7 tsReVersion.py
- 3.1.3.8 tsStripComments.py
- 3.1.3.9 tsStripLineNumbers.py
- 3.1.3.10 tsTreeCopy.py
- 3.1.3.11 tsTreeTrimLines.py

### 3.1.4 tsTestsToolsGUI

<None>

### 3.1.5 tsTestsToolsLibCLI

- 3.1.5.1 ["basicFileTypes"]
- 3.1.5.2 ["regressionTestFileTypes"]
- 3.1.3.3 "File\_Extensions.txt"
- 3.1.5.4 test\_tsLinesOfCode.py
- 3.1.5.5 test\_tsStripComments.py
- 3.1.5.6 test\_tsStripSettingsParser.py

### 3.1.6 tsTestsToolsLibGUI

<None>

### 3.2 tsLibCLI

A collection of building-block components for Python application programs used during the Command Line Interface mode of operation.

- 3.2.1 tsApplication.py
- 3.2.2 tsCommandLineEnv.py
- 3.2.3 tsCommandLineInterface.py
- 3.2.4 tsCxGlobals.py
- 3.2.5 tsDoubleLinkedList.py
- 3.2.6 tsException.py
- 3.2.7 tsGistGetTerminalSize.py
- 3.2.8 tsLogger.py
- 3.2.9 tsOperatorSettingsParser.py
- 3.2.10 tsPlatformRunTimeEnvironment.py
- 3.2.11 tsReportUtility.py
- 3.2.12 tsSysCommands.py

### 3.3 tsLibGUI

A collection of building-block components for Python application programs used during the Graphical-style User Interface mode of operation.

#### 3.3.1 wxLibPython

A collection of building-blocks for the wxPython API Graphical-style User Interface mode of operation.

NOTE: "wxWidgets", used by software developers working in the C++ programming language, or an alternate programming language-specific wrapper (such as "wxPython") provides a pixel-mode cross-platform Application Programming Interface (API) which depends on the host platform's native pixel-mode GUI services (such as GNU/Linux, Mac OS X, Microsoft Windows or Unix).

- 3.3.1.1 tsWx.py
- 3.3.1.2 tsWxAcceleratorEntry.py
- 3.3.1.3 tsWxAcceleratorTable.py
- 3.3.1.4 tsWxApp.py
- 3.3.1.5 tsWxBoxSizer.py
- 3.3.1.6 tsWxButton.py
- 3.3.1.7 tsWxCallLater.py
- 3.3.1.8 tsWxCaret.py
- 3.3.1.9 tsWxCheckBox.py
- 3.3.1.10 tsWxChoice.py
- 3.3.1.11 tsWxColor.py
- 3.3.1.12 tsWxColorDatabase.py
- 3.3.1.13 tsWxControl.py
- 3.3.1.14 tsWxControlWithItems.py

3.3.1.15 tsWxCursor.py  
3.3.1.16 tsWxDebugHandlers.py  
3.3.1.17 tsWxDialog.py  
3.3.1.18 tsWxDialogButton.py  
3.3.1.19 tsWxDisplay.py  
3.3.1.20 tsWxEraseEvent.py  
3.3.1.21 tsWxEvent.py  
3.3.1.22 tsWxEventDaemon.py  
3.3.1.23 tsWxEventLoop.py  
3.3.1.24 tsWxEventLoopActivator.py  
3.3.1.25 tsWxEvtHandler.py  
3.3.1.26 tsWxFlexGridSizer.py  
3.3.1.27 tsWxFocusEvent.py  
3.3.1.28 tsWxFrame.py  
3.3.1.29 tsWxFrameButton.py  
3.3.1.30 tsWxGauge.py  
3.3.1.31 tsWxGridBagSizer.py  
3.3.1.32 tsWxGridSizer.py  
3.3.1.33 tsWxItemContainer.py  
3.3.1.34 tsWxKeyEvent.py  
3.3.1.35 tsWxListBox.py  
3.3.1.36 tsWxLog.py  
3.3.1.37 tsWxMenu.py  
3.3.1.38 tsWxMenuBar.py  
3.3.1.39 tsWxMouseEvent.py  
3.3.1.40 tsWxMouseState.py  
3.3.1.41 tsWxObject.py  
3.3.1.42 tsWxPanel.py  
3.3.1.43 tsWxPasswordEntryDialog.py  
3.3.1.44 tsWxPoint.py  
3.3.1.45 tsWxPyApp.py  
3.3.1.46 tsWxPyEventBinder.py  
3.3.1.47 tsWxPyOnDemandOutputWindow.py  
3.3.1.48 tsWxPySimpleApp.py  
3.3.1.49 tsWxPySizer.py  
3.3.1.50 tsWxRadioBox.py  
3.3.1.51 tsWxRadioButton.py  
3.3.1.52 tsWxRect.py  
3.3.1.53 tsWxScreen.py  
3.3.1.54 tsWxScrollBar.py  
3.3.1.55 tsWxScrolled.py  
3.3.1.56 tsWxScrolledText.py  
3.3.1.57 tsWxScrolledWindow.py  
3.3.1.58 tsWxShowEvent.py  
3.3.1.59 tsWxSize.py  
3.3.1.60 tsWxSizer.py  
3.3.1.61 tsWxSizerFlags.py  
3.3.1.62 tsWxSizerItem.py  
3.3.1.63 tsWxSizerItemList.py  
3.3.1.64 tsWxSlider.py  
3.3.1.65 tsWxSplashScreen.py  
3.3.1.66 tsWxStaticBox.py  
3.3.1.67 tsWxStaticBoxSizer.py  
3.3.1.68 tsWxStaticLine.py  
3.3.1.69 tsWxStaticText.py  
3.3.1.70 tsWxStatusBar.py

3.3.1.71 tsWxSystemSettings.py  
 3.3.1.72 tsWxTextCtrl.py  
 3.3.1.73 tsWxTextEditBox.py  
 3.3.1.74 tsWxTextEntry.py  
 3.3.1.75 tsWxTextEntryDialog.py  
 3.3.1.76 tsWxTimer.py  
 3.3.1.77 tsWxToggleButton.py  
 3.3.1.78 tsWxTopLevelWindow.py  
 3.3.1.79 tsWxValidator.py  
 3.3.1.80 tsWxWindow.py

### 3.3.2 wxLibCurses

A collection of building-blocks for the Curses API Graphical-style User Interface mode of operation.

NOTE: These building blocks provide a cross-platform character-mode emulation of the host platform's native pixel-mode GUI services (such as GNU/Linux, Mac OS X, Microsoft Windows or Unix):

- a) converting between pixel- and character-mode dimensions;
- b) detects and/or sets up the color palette and foreground/background color pair combinations;
- c) handles input from terminal keyboard and mouse;
- d) formats and outputs control (position, size, font, color) and data (borders, lines, labels and messages) to the terminal display.
- e) determines which pixel-mode GUI object triggered the mouse click; and which one should receive the event notification.
- f) manages the terminal display layout to provide an application desktop area for frames and dialogs; an operator notification area for date and time stamped event messages; and a task bar area.

3.3.2.1 tsWxCursesKeyCodesDataBase.py  
 3.3.2.2 tsWxCursesMouseButtonCodesDataBase.py  
 3.3.2.3 tsWxCursesServices.py  
 3.3.2.4 tsWxDoubleLinkedList.py  
 3.3.2.5 tsWxEventQueueEntry.py  
 3.3.2.6 tsWxEventTableEntry.py  
 3.3.2.7 tsWxGlobals.py  
 3.3.2.8 tsWxGraphicalTextUserInterface.py  
 3.3.2.9 tsWxKeyboardState.py  
 3.3.2.10 tsWxMultiFrameEnv.py  
 3.3.2.11 tsWxNonLinkedList.py  
 3.3.2.12 tsWxPythonColor16DataBase.py  
 3.3.2.13 tsWxPythonColor16SubstitutionMap.py  
 3.3.2.14 tsWxPythonColor256DataBase.py

- 3.3.2.15 tsWxPythonColor88DataBase.py
- 3.3.2.16 tsWxPythonColor8DataBase.py
- 3.3.2.17 tsWxPythonColor8SubstitutionMap.py
- 3.3.2.18 tsWxPythonColorDataBaseRGB.py
- 3.3.2.19 tsWxPythonColorNames.py
- 3.3.2.20 tsWxPythonColorRGBNames.py
- 3.3.2.21 tsWxPythonColorRGBValues.py
- 3.3.2.22 tsWxPythonMonochromeDataBase.py
- 3.3.2.23 tsWxPythonPrivateLogger.py
- 3.3.2.24 tsWxScrollBarButton.py
- 3.3.2.25 tsWxScrollBarGauge.py
- 3.3.2.26 tsWxSizerSpacer.py
- 3.3.2.27 tsWxTaskBar.py
- 3.3.2.28 tsWxWindowCurses.py

### 3.4 tsToolsCLI

A collection of Python application programs for the Command Line Interface mode of operation.

- 3.4.1 tsChange\_tsWxGTUI\_PyVx.py
- 3.4.2 tsChange\_wxPython\_API\_Version.py
- 3.4.3 tsLinesOfCodeProjectMetrics.py
- 3.4.4 tsPlatformQuery.py
- 3.4.5 tsStripComments.py
- 3.4.6 tsStripLineNumbers.py
- 3.4.7 tsTreeCopy.py
- 3.4.8 tsTreeTrimLines.py

### 3.5 tsToolsGUI

A collection of Python application programs for the Graphical-style User Interface mode of operation.

<none>

### 3.6 tsToolsLibCLI

A collection of building-block components for Python application program tools used during the Command Line Interface mode of operation.

### 3.7 tsToolsLibGUI

A collection of building-block components for Python application program tools used during the Graphical-style User Interface mode of operation.

<none>

### 3.8 tsUtilities

A collection of operating system shell scripts and Python scripts used for hardware and software administration during the Command Line Interface mode of operation.

===== End-Of-File =====

Draft

## 4.3.8 README7-DeveloperSandboxes.txt

```
#-----
#"Time-stamp: <06/05/2015 4:25:31 AM rsg>"
#-----

===== File: README6-SourceDistributions.txt =====

+-----+-----+ TeamSTARS "tsWxGTUI_PyVx" Toolkit
| ts | Wx | with Python 2x & Python 3x based
+-----+-----+ Command Line Interface (CLI)
| G T U I | and "Curses"-based "wxPython"-style,
+-----+-----+ Graphical-Text User Interface (GUI)

Get that cross-platform, pixel-mode "wxPython" feeling
on character-mode 8-/16-color (xterm-family) & non-color
(vt100-family) terminals and terminal emulators.

You can find this and other files in the following sub-
directories:

<Your Working Repository>
(e.g. "Technical_Preview")
|
| Working repository containing directories and
| files to be packaged into downloadable "tarball"
| and/or "zip" files via the setup shell scripts
| at the bottom of this diagram.
|
+-- ["Documents"]
| |
| | This directory contains a collection of files
| | which provide the Toolkit recipient with an
| | understanding of the purpose, goals & capabil-
| | ities, non-goals & limitations, terms & condi-
| | tions and procedures for installing, operating,
| | modifying and redistributing the Toolkit.
| |
| +-- "README3-Documents.txt"
|
+-- ["ManPages"]
| |
| | Deliverable Toolkit manual pages are a
| | form of online software documentation
| | usually found on a Unix or Unix-like oper-
| | ating system.
| |
| +-- "README4-ManPages.txt"
|
+-- ["Notebooks"]
| |
| | Contains a collection of commentaries that
| | express opinions or offerings of explana-
| | tions about events or situations that might
```



```

| | be useful to Toolkit installers, developers,
| | operators, troubleshooters and distributors.
| | The documents may be in Application-specific
| | formats (Adobe PDF, JPEG Bit-mapped image,
| | Microsoft Office, Plain text etc.).
|
| +--- "README5-Notebooks.txt"
|
+--- ["SourceDistributions"]
|
| | Contains a collection of computer program
| | source code files that the Toolkit recip-
| | ient will need to install, operate, modify
| | and re-distribute the Toolkit.
|
| +--- "README6-SourceDistributions.txt"
|
+--- ["Developer-Sandboxes"] (Pre-dates Site-Packages)
|
| | A sandbox is a testing environment that iso-
| | lates untested code changes and outright experi-
| | mentation from the production environment or
| | repository.
|
| +--- ["tsWxGTUI_PyVx"]
|
| | Contains one or more Python language gener-
| | ation-specific releases each sharing the same
| | programmer (API) and user (CLI & GUI) inter-
| | faces, documents and manual pages.
|
| +--- ["Documents"] (copy)
|
| +--- ["ManPages"] (copy)
|
| +--- ["Python-2x"]
|
| | | Second generation Python programming
| | | language.
| |
| | +--- ["tsWxGTUI_Py2x"]
| |
| | | +--- ["tsDemoArchive"]
| | | |
| | | +--- ["src"]
| | | |
| | | +--- "TermsAndConditions.txt"
| | |
| | +--- ["tsLibCLI"]
| | |
| | | +--- ["tsApplicationPkg"]
| | | |
| | | | +--- ["src"]
| | | | +--- ["test"]
| | |
| | +--- ["tsCommandLineEnvPkg"]

```



```
["tsLinesOfCodeProjectMetricsPkg"]
|
| +-- ["src"]
| +-- ["test"]
|
+-- ["tsPlatformQueryPkg"]
|
| +-- ["src"]
| +-- ["test"]
|
+-- ["tsStripCommentsPkg"]
|
| +-- ["src"]
| +-- ["test"]
|
+-- ["tsStripLineNumbersPkg"]
|
| +-- ["src"]
| +-- ["test"]
|
+-- ["tsTreeCopyPkg"]
|
| +-- ["src"]
| +-- ["test"]
|
+-- ["tsTreeTrimLinesPkg"]
|
| +-- ["src"]
| +-- ["test"]
|
+-- ["tsToolsGUI"]
|
+-- ["tsUtilities"]
|
+-- ["Python-3x"] (Ported from Python-2x)
|
| Third generation Python programming
| language.
|
+-- ["tsWxGTUI_Py3x"]

+-- ["Site-Packages"]
|
| A site-packages is the location where third-
| party packages are installed (i.e., those
| not part of the core Python distribution).
| NOTE: That with Linux, Mac OS X and Unix
```

```
|
| operating systems one must have root priv-
| ileages to write to that location.
|
+-- ["tsWxGTUI_PyVx"]
|
| +-- ["Documents"] (copy)
| |
| +-- ["ManPages"] (copy)
| |
+-- ["Python-2x"]
| |
| | Second generation Python programming
| | language.
| |
| +-- ["tsWxGTUI_Py2x"]
| | |
| | +-- ["tsDemoArchive"]
| | | |
| | | +-- ["tsTestsLibCLI"]
| | | +-- ["tsTestsLibGUI"]
| | | +-- ["tsTestsToolsCLI"]
| | | +-- ["tsTestsToolsGUI"]
| | | +-- ["tsTestsToolsLibCLI"]
| | | +-- ["tsTestsToolsLibGUI"]
| | |
| | +-- ["tsLibCLI"]
| | |
| | +-- ["tsLibGUI"]
| | |
| | +-- ["tsToolsCLI"]
| | |
| | +-- ["tsToolsGUI"]
| | |
| | +-- ["tsUtilities"]
| |
+-- ["Python-3x"] (Ported from Python-2x)
| |
| | Third generation Python programming
| | language.
| |
| +-- ["tsWxGTUI_Py3x"]
|
+-- "README.txt"
```

===== TABLE OF CONTENTS =====

1. Source Code

2. Developer-Sandbox

2.1 tsLibCLI	(equivalent to 3.1.1 tsTestsLibCLI)
2.2 tsLibGUI	(equivalent to 3.1.2 tsTestsLibGUI)
2.3 tsToolsCLI	(equivalent to 3.1.3 tsTestsToolsCLI and 3.1.5 tsTestsToolsLibCLI)
2.4 tsToolsGUI	(equivalent to 3.1.4 tsTestsToolsGUI)
2.5 tsUtilities	(equivalent to 3.8 tsUtilities)

### 3. Site-Package

#### 3.1 tsDemoArchive

- 3.1.1 tsTestsLibCLI
- 3.1.2 tsTestsLibGUI
- 3.1.3 tsTestsToolsCLI
- 3.1.4 tsTestsToolsGUI
- 3.1.5 tsTestsToolsLibCLI
- 3.1.6 tsTestsToolsLibGUI

#### 3.2 tsLibCLI

#### 3.3 tsLibGUI

#### 3.4 tsToolsCLI

#### 3.5 tsToolsGUI

#### 3.6 tsToolsLibCLI

#### 3.7 tsToolsLibGUI

#### 3.8 tsUtilities

===== SourceDistribution =====

### 1. Source Code

Excerpt From Wikipedia, the free encyclopedia:

"In computing, source code is any collection of computer instructions (possibly with comments) written using some human-readable computer language, usually as text. The source code of a program is specially designed to facilitate the work of computer programmers, who specify the actions to be performed by a computer mostly by writing source code. The source code is often transformed by a compiler program into low-level machine code understood by the computer. The machine code might then be stored for execution at a later time. Alternatively, an interpreter can be used to analyze and perform the outcomes of the source code program directly on the fly.

Most computer applications are distributed in a form that includes executable files, but not their source code. If the source code were included, it would be useful to a user, programmer, or system administrator, who may wish to modify the program or to understand how it works.

Aside from its machine-readable forms, source code also appears in books and other media; often in the form of small code snippets, but occasionally complete code bases; a well-known case is the source code of PGP."

===== DEVELOPER-SANDBOX =====

### 2. Developer-Sandbox

Excerpt From Wikipedia, the free encyclopedia:



===== SITE-PACKAGE =====

### 3. Site-Package

Site-packages is the location where third-party packages are installed (i.e., those not part of the core Python distribution). NOTE: That with Linux, Mac OS X and Unix operating systems one must have root privileges to write to that location.

Unlike the contents of the Developer-Sandbox, the third-party site-package and its users must explicitly import via the site-package.package.module path identifier.

```
<Your Working Repository>
(e.g. "Technical_Preview")
|
| Working repository containing directories and
| files to be packaged into downloadable "tarball"
| and/or "zip" files via the setup shell scripts
| at the bottom of this diagram.
|
+-- ["Documents"]
| |
| | This directory contains a collection of files
| | which provide the Toolkit recipient with an
| | understanding of the purpose, goals & capabilities,
| | non-goals & limitations, terms & conditions and
| | procedures for installing, operating, modifying and
| | redistributing the Toolkit.
| |
| +-- "README3-Documents.txt"
|
+-- ["ManPages"]
| |
| | Deliverable Toolkit manual pages are a form of online
| | software documentation usually found on a Unix or Unix-
| | like operating system.
| |
| +-- "README4-ManPages.txt"
|
+-- ["Notebooks"]
| |
| | Contains a collection of commentaries that express
| | opinions or offerings of explanations about events or
| | situations that might be useful to Toolkit installers,
| | developers, operators, troubleshooters and distributors.
| | The documents may be in Application-specific formats
| | (Adobe PDF, JPEG Bit-mapped image, Microsoft Office,
| | Plain text etc.).
| |
| +-- "README5-Notebooks.txt"
```

```
+-- ["SourceDistributions"]
|
| Contains a collection of computer program
| source code files that the Toolkit recip-
| ient will need to install, operate, modify
| and re-distribute the Toolkit.
|
+-- "README6-SourceDistributions.txt"
|
+-- ["Developer-Sandboxes"] (Pre-dates Site-Packages)
|
| A sandbox is a testing environment that iso-
| lates untested code changes and outright
| experimentation from the production environ-
| ment or repository.
|
+-- ["tsWxGTUI_PyVx"]
|
| +-- ["Documents"] (copy)
| |
| +-- ["ManPages"] (copy)
| |
| +-- ["Python-2x"]
| | |
| | +-- ["tsWxGTUI_Py2x"]
| | |
| | +-- ["Python-3x"] (Ported from Python-2x)
| | |
| | +-- ["tsWxGTUI_Py3x"]
| |
+-- ["Site-Packages"]
|
| Site-packages is the location where third-
| party packages are installed (i.e., those
| not part of the core Python distribution).
| NOTE: That with Linux, Mac OS X and Unix
| operating systems one must have root priv-
| ileages to write to that location.
|
+-- ["tsWxGTUI_PyVx"]
|
| +-- ["Documents"] (copy)
| |
| +-- ["ManPages"] (copy)
| |
| +-- ["Python-2x"]
| | |
| | +-- ["tsWxGTUI_Py2x"]
| | |
| | +-- ["tsDemoArchive"]
| | |
| | | +-- ["tsTestsLibCLI"]
| | | +-- ["tsTestsLibGUI"]
| | | +-- ["tsTestsToolsCLI"]
| | | +-- ["tsTestsToolsGUI"]
| | | +-- ["tsTestsToolsLibCLI"]
```



```
| | | | +-- ["t
| | | | +-- ["t
| | | | +-- ["
| | | | |
| | | | +-- "Te
| | | |
+-- ["tsLibCD
+-- ["tsLibGU
+-- ["tsTools
+-- ["tsTools
+-- ["tsTools
+-- ["tsTools
+-- ["tsUtili
```

### 3.1 tsDemoArchive

### 3.1.1 tsTestsLibCLI

- ```
3.1.1.1 test_TermsAndConditions.py
3.1.1.2 test_tsApplication.py
3.1.1.3 test_tsCommandLineEnv.py
3.1.1.4 test_tsCommandLineInterface.py
3.1.1.5 test_tsCxGlobals.py
3.1.1.6 test_tsDoubleLinkedList.py
3.1.1.7 test_tsException.py
```

- 3.1.1.8 test_tsLogger.py
- 3.1.1.9 test_tsOperatorSettingsParser.py
- 3.1.1.10 test_tsPlatformRunTimeEnvironment.py
- 3.1.1.11 test_tsReportUtility.py
- 3.1.1.12 test_tsSysCommands.py

3.1.2 tsTestsLibGUI

- 3.1.2.1 test_cursesOverlappingPanels.py
- 3.1.2.2 test_cursesPanels.py
- 3.1.2.3 test_tsWxBoxSizer.py
- 3.1.2.4 test_tsWxCheckBox.py
- 3.1.2.5 test_tsWxColorPalette.py
- 3.1.2.6 test_tsWxDisplay.py
- 3.1.2.7 test_tsWxDoubleLinkedList.py
- 3.1.2.8 test_tsWxGlobals.py
- 3.1.2.9 test_tsWxGraphicalTextUserInterface.py
- 3.1.2.10 test_tsWxGridSizer.py
- 3.1.2.11 test_tsWxMultiFrameEnv.py
- 3.1.2.12 test_tsWxPasswordEntryDialog.py
- 3.1.2.13 test_tsWxRSM.py
- 3.1.2.14 test_tsWxScrolledWindow.py
- 3.1.2.15 test_tsWxScrolledWindowDual.py
- 3.1.2.16 test_tsWxSplashScreen.py
- 3.1.2.17 test_tsWxSystemSettings.py
- 3.1.2.17 test_tsWxTextEntryDialog.py
- 3.1.2.18 test_tsWxWidgets.py

3.1.3 tsTestsToolsCLI

- 3.1.3.1 ["basicFileTypes"]
- 3.1.3.2 ["regressionTestFileTypes"]
- 3.1.3.3 ["sampleHelp"]
- 3.1.3.4 "File_Extensions.txt"
- 3.1.3.5 tsLinesOfCodeProjectMetrics.py
- 3.1.3.6 tsPlatformQuery.py
- 3.1.3.7 tsReVersion.py
- 3.1.3.8 tsStripComments.py
- 3.1.3.9 tsStripLineNumbers.py
- 3.1.3.10 tsTreeCopy.py
- 3.1.3.11 tsTreeTrimLines.py

3.1.4 tsTestsToolsGUI

<None>

3.1.5 tsTestsToolsLibCLI

- 3.1.5.1 ["basicFileTypes"]
- 3.1.5.2 ["regressionTestFileTypes"]
- 3.1.3.3 "File_Extensions.txt"
- 3.1.5.4 test_tsLinesOfCode.py
- 3.1.5.5 test_tsStripComments.py
- 3.1.5.6 test_tsStripSettingsParser.py

3.1.6 tsTestsToolsLibGUI

<None>

3.2 tsLibCLI

A collection of building-block components for Python application programs used during the Command Line Interface mode of operation.

- 3.2.1 tsApplication.py
- 3.2.2 tsCommandLineEnv.py
- 3.2.3 tsCommandLineInterface.py
- 3.2.4 tsCxGlobals.py
- 3.2.5 tsDoubleLinkedList.py
- 3.2.6 tsException.py
- 3.2.7 tsGistGetTerminalSize.py
- 3.2.8 tsLogger.py
- 3.2.9 tsOperatorSettingsParser.py
- 3.2.10 tsPlatformRunTimeEnvironment.py
- 3.2.11 tsReportUtility.py
- 3.2.12 tsSysCommands.py

3.3 tsLibGUI

A collection of building-block components for Python application programs used during the Graphical-style User Interface mode of operation.

3.3.1 wxLibPython

A collection of building-blocks for the wxPython API Graphical-style User Interface mode of operation.

NOTE: "wxWidgets", used by software developers working in the C++ programming language, or an alternate programming language-specific wrapper (such as "wxPython") provides a pixel-mode cross-platform Application Programming Interface (API) which depends on the host platform's native pixel-mode GUI services (such as GNU/Linux, Mac OS X, Microsoft Windows or Unix).

- 3.3.1.1 tsWx.py
- 3.3.1.2 tsWxAcceleratorEntry.py
- 3.3.1.3 tsWxAcceleratorTable.py
- 3.3.1.4 tsWxApp.py
- 3.3.1.5 tsWxBoxSizer.py
- 3.3.1.6 tsWxButton.py
- 3.3.1.7 tsWxCallLater.py
- 3.3.1.8 tsWxCaret.py
- 3.3.1.9 tsWxCheckBox.py
- 3.3.1.10 tsWxChoice.py
- 3.3.1.11 tsWxColor.py
- 3.3.1.12 tsWxColorDatabase.py
- 3.3.1.13 tsWxControl.py
- 3.3.1.14 tsWxControlWithItems.py

3.3.1.15 tsWxCursor.py
3.3.1.16 tsWxDebugHandlers.py
3.3.1.17 tsWxDialog.py
3.3.1.18 tsWxDialogButton.py
3.3.1.19 tsWxDisplay.py
3.3.1.20 tsWxEraseEvent.py
3.3.1.21 tsWxEvent.py
3.3.1.22 tsWxEventDaemon.py
3.3.1.23 tsWxEventLoop.py
3.3.1.24 tsWxEventLoopActivator.py
3.3.1.25 tsWxEvtHandler.py
3.3.1.26 tsWxFlexGridSizer.py
3.3.1.27 tsWxFocusEvent.py
3.3.1.28 tsWxFrame.py
3.3.1.29 tsWxFrameButton.py
3.3.1.30 tsWxGauge.py
3.3.1.31 tsWxGridBagSizer.py
3.3.1.32 tsWxGridSizer.py
3.3.1.33 tsWxItemContainer.py
3.3.1.34 tsWxKeyEvent.py
3.3.1.35 tsWxListBox.py
3.3.1.36 tsWxLog.py
3.3.1.37 tsWxMenu.py
3.3.1.38 tsWxMenuBar.py
3.3.1.39 tsWxMouseEvent.py
3.3.1.40 tsWxMouseState.py
3.3.1.41 tsWxObject.py
3.3.1.42 tsWxPanel.py
3.3.1.43 tsWxPasswordEntryDialog.py
3.3.1.44 tsWxPoint.py
3.3.1.45 tsWxPyApp.py
3.3.1.46 tsWxPyEventBinder.py
3.3.1.47 tsWxPyOnDemandOutputWindow.py
3.3.1.48 tsWxPySimpleApp.py
3.3.1.49 tsWxPySizer.py
3.3.1.50 tsWxRadioBox.py
3.3.1.51 tsWxRadioButton.py
3.3.1.52 tsWxRect.py
3.3.1.53 tsWxScreen.py
3.3.1.54 tsWxScrollBar.py
3.3.1.55 tsWxScrolled.py
3.3.1.56 tsWxScrolledText.py
3.3.1.57 tsWxScrolledWindow.py
3.3.1.58 tsWxShowEvent.py
3.3.1.59 tsWxSize.py
3.3.1.60 tsWxSizer.py
3.3.1.61 tsWxSizerFlags.py
3.3.1.62 tsWxSizerItem.py
3.3.1.63 tsWxSizerItemList.py
3.3.1.64 tsWxSlider.py
3.3.1.65 tsWxSplashScreen.py
3.3.1.66 tsWxStaticBox.py
3.3.1.67 tsWxStaticBoxSizer.py
3.3.1.68 tsWxStaticLine.py
3.3.1.69 tsWxStaticText.py
3.3.1.70 tsWxStatusBar.py

- 3.3.1.71 tsWxSystemSettings.py
- 3.3.1.72 tsWxTextCtrl.py
- 3.3.1.73 tsWxTextEditBox.py
- 3.3.1.74 tsWxTextEntry.py
- 3.3.1.75 tsWxTextEntryDialog.py
- 3.3.1.76 tsWxTimer.py
- 3.3.1.77 tsWxToggleButton.py
- 3.3.1.78 tsWxTopLevelWindow.py
- 3.3.1.79 tsWxValidator.py
- 3.3.1.80 tsWxWindow.py

3.3.2 wxLibCurses

A collection of building-blocks for the Curses API Graphical-style User Interface mode of operation.

NOTE: These building blocks provide a cross-platform character-mode emulation of the host platform's native pixel-mode GUI services (such as GNU/Linux, Mac OS X, Microsoft Windows or Unix):

- a) converting between pixel- and character-mode dimensions;
- b) detects and/or sets up the color palette and foreground/background color pair combinations;
- c) handles input from terminal keyboard and mouse;
- d) formats and outputs control (position, size, font, color) and data (borders, lines, labels and messages) to the terminal display.
- e) determines which pixel-mode GUI object triggered the mouse click; and which one should receive the event notification.
- f) manages the terminal display layout to provide an application desktop area for frames and dialogs; an operator notification area for date and time stamped event messages; and a task bar area.

- 3.3.2.1 tsWxCursesKeyCodesDataBase.py
- 3.3.2.2 tsWxCursesMouseButtonCodesDataBase.py
- 3.3.2.3 tsWxCursesServices.py
- 3.3.2.4 tsWxDoubleLinkedList.py
- 3.3.2.5 tsWxEventQueueEntry.py
- 3.3.2.6 tsWxEventTableEntry.py
- 3.3.2.7 tsWxGlobals.py
- 3.3.2.8 tsWxGraphicalTextUserInterface.py
- 3.3.2.9 tsWxKeyboardState.py
- 3.3.2.10 tsWxMultiFrameEnv.py
- 3.3.2.11 tsWxNonLinkedList.py
- 3.3.2.12 tsWxPythonColor16DataBase.py
- 3.3.2.13 tsWxPythonColor16SubstitutionMap.py
- 3.3.2.14 tsWxPythonColor256DataBase.py

- 3.3.2.15 tsWxPythonColor88DataBase.py
- 3.3.2.16 tsWxPythonColor8DataBase.py
- 3.3.2.17 tsWxPythonColor8SubstitutionMap.py
- 3.3.2.18 tsWxPythonColorDataBaseRGB.py
- 3.3.2.19 tsWxPythonColorNames.py
- 3.3.2.20 tsWxPythonColorRGBNames.py
- 3.3.2.21 tsWxPythonColorRGBValues.py
- 3.3.2.22 tsWxPythonMonochromeDataBase.py
- 3.3.2.23 tsWxPythonPrivateLogger.py
- 3.3.2.24 tsWxScrollBarButton.py
- 3.3.2.25 tsWxScrollBarGauge.py
- 3.3.2.26 tsWxSizerSpacer.py
- 3.3.2.27 tsWxTaskBar.py
- 3.3.2.28 tsWxWindowCurses.py

3.4 tsToolsCLI

A collection of Python application programs for the Command Line Interface mode of operation.

- 3.4.1 tsChange_tsWxGTUI_PyVx.py
- 3.4.2 tsChange_wxPython_API_Version.py
- 3.4.3 tsLinesOfCodeProjectMetrics.py
- 3.4.4 tsPlatformQuery.py
- 3.4.5 tsStripComments.py
- 3.4.6 tsStripLineNumbers.py
- 3.4.7 tsTreeCopy.py
- 3.4.8 tsTreeTrimLines.py

3.5 tsToolsGUI

A collection of Python application programs for the Graphical-style User Interface mode of operation.

<none>

3.6 tsToolsLibCLI

A collection of building-block components for Python application program tools used during the Command Line Interface mode of operation.

3.7 tsToolsLibGUI

A collection of building-block components for Python application program tools used during the Graphical-style User Interface mode of operation.

<none>

3.8 tsUtilities

A collection of operating system shell scripts and Python scripts used for hardware and software administration during the Command Line Interface mode of operation.

===== End-Of-File =====

Draft

4.3.9 README8-SitePackages.txt

```
#-----
#"Time-stamp: <06/04/2015  9:24:01 PM rsg>"
#-----

===== Title Page for File: README8-Site-Packages.txt =====

+-----+-----+ TeamSTARS "tsWxGTUI_PyVx" Toolkit
| ts | Wx |      with Python 2x & Python 3x based
+-----+-----+      Command Line Interface (CLI)
| G T U I |      and "Curses"-based "wxPython"-style,
+-----+-----+      Graphical-Text User Interface (GUI)

Get that cross-platform, pixel-mode "wxPython" feeling
on character-mode 8-/16-color (xterm-family) & non-color
(vt100-family) terminals and terminal emulators.

You can find this and other files in the following sub-
directories:

<Your Working Repository>
(e.g. "Technical_Preview")
|
|   Working repository containing directories and
|   files to be packaged into downloadable "tarball"
|   and/or "zip" files via the setup shell scripts
|   at the bottom of this diagram.
|
+-- ["Documents"]
|
|   |   This directory contains a collection of files
|   |   which provide the Toolkit recipient with an
|   |   understanding of the purpose, goals & capabil-
|   |   ities, non-goals & limitations, terms & condi-
|   |   tions and procedures for installing, operating,
|   |   modifying and redistributing the Toolkit.
|   |
|   |   Deliverable Toolkit manual pages are a
|   |   form of online software documentation
|   |   usually found on a Unix or Unix-like oper-
|   |   ating system.
|   |
+-- "README3-Documents.txt"
|
+-- ["ManPages"]
|
|   |   Deliverable Toolkit manual pages are a
|   |   form of online software documentation
|   |   usually found on a Unix or Unix-like oper-
|   |   ating system.
|   |
+-- "README4-ManPages.txt"
|
```



```

+-- ["Notebooks"]
|
| Contains a collection of commentaries that
| express opinions or offerings of explana-
| tions about events or situations that might
| be useful to Toolkit installers, developers,
| operators, troubleshooters and distributors.
| The documents may be in Application-specific
| formats (Adobe PDF, JPEG Bit-mapped image,
| Microsoft Office, Plain text etc.).
|
+-- "README5-Notebooks.txt"
+-- ["SourceDistributions"]
|
| Contains a collection of computer program
| source code files that the Toolkit recip-
| ient will need to install, operate, modify
| and re-distribute the Toolkit.
|
+-- "README6-SourceDistributions.txt"
+-- ["Developer-Sandboxes"] (Pre-dates Site-Packages)
|
| A sandbox is a testing environment that iso-
| lates untested code changes and outright experi-
| mentation from the production environment or
| repository.
|
+-- ["tsWxGTUI_PyVx"]
|
| Contains one or more Python language gener-
| ation-specific releases each sharing the same
| programmer (API) and user (CLI & GUI) inter-
| faces, documents and manual pages.
|
+-- ["Documents"]
|
+-- ["ManPages"]
|
+-- ["Python-2x"]
|
| Second generation Python programming
| language.
|
+-- ["tsWxGTUI_Py2x"]
|
| +-- ["tsDemoArchive"]
|
| +-- ["src"]
|
| +-- "TermsAndConditions.txt"
|
+-- ["tsLibCLI"]
|
+-- ["tsApplicationPkg"]

```


			+
			+
		+--	["t
			+
			+
		+--	["t
			+
			+

```
|
|
|   A site-packages is the location where third-
|   party packages are installed (i.e., those
|   not part of the core Python distribution).
|   NOTE: That with Linux, Mac OS X and Unix
|   operating systems one must have root priv-
|   ileages to write to that location.
|
+-- ["tsWxGTUI_PyVx"]
|
|   +-- ["Documents"]
|   |
|   +-- ["ManPages"]
|   |
|   +-- ["Python-2x"]
|   |
|   |   Second generation Python programming
|   |   language.
|   |
|   |   +-- ["tsWxGTUI_Py2x"]
|   |   |
|   |   |   +-- ["tsDemoArchive"]
|   |   |   |
|   |   |   |   +-- ["tsTestsLibCLI"]
|   |   |   |   +-- ["tsTestsLibGUI"]
|   |   |   |   +-- ["tsTestsToolsCLI"]
|   |   |   |   +-- ["tsTestsToolsGUI"]
|   |   |   |   +-- ["tsTestsToolsLibCLI"]
|   |   |   |   +-- ["tsTestsToolsLibGUI"]
|   |   |   |
|   |   |   +-- ["tsLibCLI"]
|   |   |   |
|   |   |   +-- ["tsLibGUI"]
|   |   |   |
|   |   |   +-- ["tsToolsCLI"]
|   |   |   |
|   |   |   +-- ["tsToolsGUI"]
|   |   |   |
|   |   |   +-- ["tsUtilities"]
|   |   |
|   |   +-- ["Python-3x"] (Ported from Python-2x)
|   |   |
|   |   |   Third generation Python programming
|   |   |   language.
|   |   |
|   |   |   +-- ["tsWxGTUI_Py3x"]
|   |
+-- "README.txt"
```

===== TABLE OF CONTENTS =====

1. Source Code
2. Developer-Sandbox
3. Site-Package

===== SourceDistribution =====

1. Source Code

Excerpt From Wikipedia, the free encyclopedia:

"In computing, source code is any collection of computer instructions (possibly with comments) written using some human-readable computer language, usually as text. The source code of a program is specially designed to facilitate the work of computer programmers, who specify the actions to be performed by a computer mostly by writing source code. The source code is often transformed by a compiler program into low-level machine code understood by the computer. The machine code might then be stored for execution at a later time. Alternatively, an interpreter can be used to analyze and perform the outcomes of the source code program directly on the fly.

Most computer applications are distributed in a form that includes executable files, but not their source code. If the source code were included, it would be useful to a user, programmer, or system administrator, who may wish to modify the program or to understand how it works.

Aside from its machine-readable forms, source code also appears in books and other media; often in the form of small code snippets, but occasionally complete code bases; a well-known case is the source code of PGP."

===== DEVELOPER-SANDBOX =====

2. Developer-Sandbox

Excerpt From Wikipedia, the free encyclopedia:

"A sandbox is a testing environment that isolates untested code changes and outright experimentation from the production environment or repository, in the context of software development including Web development and revision control. Sandboxing protects "live" servers and their data, vetted source code distributions, and other collections of code, data and/or content, proprietary or public, from changes that could be damaging (regardless of the intent of the author of those changes) to a mission-critical system or which could simply be difficult to revert. Sandboxes replicate at least the minimal functionality needed to accurately test the programs or other code under development (e.g. usage of the same environment variables as, or access to an identical database to that used by, the stable prior implementation intended to be modified; there are many other possibilities, as the specific functionality needs vary widely with the nature of the code and the application[s] for which it is intended.)

The concept of the sandbox (sometimes also called a working directory, a test server or development server) is typically built into revision control software such as CVS and Subversion (SVN), in which developers "check out" a copy of the source code tree, or a branch thereof, to examine and work on. Only after the developer has (hopefully) fully tested the code changes in their own sandbox should the changes be checked back into and merged with the repository and thereby made available to other developers or end users of the software.[1]

By further analogy, the term "sandbox" can also be applied in computing and networking to other temporary or indefinite isolation areas, such as security sandboxes and search engine sandboxes (both of which have highly specific meanings), that prevent incoming data from affecting a "live" system (or aspects thereof) unless/until defined requirements or criteria have been met."

Unlike the contents of the installable site-package, this sandbox uses a multi-level tree of subdirectories and associated files whose topology is defined by a set of package "__init__.py" files which collaborate in performing dynamic path generation and importing of modules and subpackages. Applications import individual packages and individual modules simply by name (if module name is unique) or by package.module name (if module name is not unique).

===== SITE-PACKAGE =====

3. Site-Package

Site-packages is the location where third-party packages are installed (i.e., those not part of the core Python distribution). NOTE: That with Linux, Mac OS X and Unix operating systems one must have root privileges to write to that location.

Unlike the contents of the Developer-Sandbox, the third-party site-package and its users must explicitly import via the site-package.package.module path identifier.

===== End-Of-File =====

CHAPTER 5

5 RELEASE NOTES

5.1.1.1 In This Chapter

APPENDIX A - BASELINE HOST COMPUTER PLATFORMS	415
APPENDIX B - API RELATIONSHIP	425
APPENDIX C - DELIVERABLES	427

Draft

Draft

5.2 APPENDIX A - BASELINE HOST COMPUTER PLATFORMS

This section shall identify those host computer platforms actually used to plan, implement, test, document, deploy and maintain the "tsWxGTUI_PyVx" Toolkit.

- 1** *Currently Used Platforms* (see "*Currently Used Platforms (Release Notes)*" on page 8) - Linux (Fedora 22 / OpenSUSE 13.2 / Scientific (CentOS) 7 / Ubuntu 12.04, 14.04 & 15.04), Mac OS X (10.7 / 10.11), Microsoft Windows (7 Pro. 32-bit with Cygwin 1.7.x / 10 Pro. 64-bit with Cygwin 2.2) / and Unix (PC-BSD 10 & OpenIndiana 151a8 / Solaris 11 / SunOS 5.11).
-

The Toolkit's Python source code has been developed and tested only with Intel x86 and x64 processors and representative GNU/Linux, Mac OS X, Microsoft Windows and Unix operating system releases.

Its source code has been compiled, interpreted and executed by Python 2x and 3x Virtual Machines developed by the Python Software Foundation (PSF).

The PSF also distributes equivalent Python 2x and 3x Virtual Machines (and the source code to build them) for other processor types and operating systems, some of which are listed below.

See file `"/tsWxGTUI_PyVx_Repository/Notebooks/EngineeringNotebook/MS-Excel-Files/Platform_Configuration.xls"` for Python version specific platform configuration details (Sheet 1 --- Host Configurations; Sheet 2 --- Terminal Configurations):

- a) 2013-model year, 3.5 GHz Intel Quad Core i7 processor-based desktop with 16 GB RAM and sufficient performance, resources and expansion capabilities to serve as the high-level baseline development and operator platform.
 - b) 2007-model year, 2.33 GHz Intel Core 2 Duo processor-based laptop with 4 GB RAM and sufficient performance, resources and expansion capabilities to serve as the moderate-level baseline development and operator platform.
 - c) 1998-model year, 366 MHz Intel Pentium II-based laptop with 384 MB RAM and expansion capabilities (marginal resources and performance) sufficient enough to serve as the low-level baseline development and operator platform.
- 2** *Previously Used Platforms* (see "*Previously Used Platforms (Release Notes)*" on page 13) - Linux (Fedora 15 & 16 / Open SuSE 11 / Ubuntu 10.04), Mac OS X (10.4 / 10.5 / 10.6 / 10.7), Microsoft Windows (8 Professional / 7 Professional / XP Professional each with Cygwin 1.7.x) and Unix (OpenIndiana 151a6 / Solaris 11 / SunOS 5.11).
- a) 2007-model year, 2.33 GHz Intel Core 2 Duo processor-based laptop with 4 GB RAM and sufficient performance, resources and expansion capabilities to serve as the moderate-level baseline development and operator platform.
 - b) 1998-model year, 366 MHz Intel Pentium II-based laptop with 384 MB RAM and expansion capabilities (marginal resources and performance) sufficient enough to serve as the low-level baseline development and operator platform.

- 3** *Designing Embedded Systems with Linux and Python* (on page 16) - Link to YouTube Video by Mark Kohler.

5.2.1 Currently Used Platforms (Release Notes)

Host Computer Platform Configurations currently used by "tsWxGTUI_PyVx" Toolkit developers:

Draft

Draft

Make & Model	Hardware	Software
Apple 27" iMac Desktop	<p>2013-model year, 3.5 GHz Intel Quad Core i7 processor-based desktop with 16 GB RAM, 2560x1440 pixel resolution display and sufficient performance, resources and expansion capabilities to serve as the high-level baseline development and operator platform.</p> <ul style="list-style-type: none"> ▪ Its 3 TB (7200 RPM) SATA 6 Gb/s internal hard drive had 128 GB Solid State Flash memory and was used to run Apple's Mac OS X 10.9-19.10 and the hypervisor virtualization applications (Parallels Desktop 9-10 and VMware Fusion 5 and 7.1) that supported various guest operating systems. ▪ Its 3 TB (7200 RPM) SATA 6 Gb/s internal hard drive was also used to store and run configured versions of the eComStation Demo CD version 2.2 BETA (IBM OS/2 4.5 Warp descendant), CentOS Linux (7.0), Fedora Linux (21), OpenSUSE Linux (13.1), Scientific Linux (6.5), Ubuntu Linux (12.04, 14.04), Microsoft Windows (8.1 Pro, 8 Pro, 7 Pro, XP Pro and 2000 Pro), and Unix (PC-BSD (9.2, 10.0), OpenIndiana 151A8) guest operating systems. ▪ Hewlett-Packard Company P2015DN Series (26A5C8) LaserJet Printer conected via Ethernet ▪ Hewlett-Packard Company Officejet Pro 8500A (A910) e-All-in-One Series Printer connected viia Wi-Fi or USB. 	<p>Host Operating System</p> <ul style="list-style-type: none"> ▪ Apple's Mac OS X (initially Mavericks releases 10.9.x and currently Yosemite releases 10.10.x) with Python 2.6.8, 2.7.5, 3.2.2 <p>Host Hypervisor Virtualization Applications</p> <ul style="list-style-type: none"> ▪ Parallels Desktop 9-11 ▪ VMware Fusion 5 & 7.1 <p>Concurrent or Interchangeable Guest Operating Systems (cloned from Apple 17" MacBook Pro Laptop and configured to share host drives and use 1440 x 900 display)</p> <ul style="list-style-type: none"> ▪ eComStation Demo CD version 2.2 BETA (IBM OS/2 4.5 Warp descendant) runs only what was shipped on the live CD and does not include Python. Most recent Python port for OS/2 & eComStations is Python 2.7.5. For details, see the following: "http://os2ports.smedley.id.au/index.php?page=python" and "http://blog.python.org/2011/05/python-33-to-drop-support-for-os2.html" ▪ Linux (CentOS 7.1 64-bit, Debian 8.2 64-bit, Fedora 22 & 23 64-bit, LXLE 14.02 32-bit, OpenSUSE 12.2 & 13.2 64-bit, Scientific 6.4 & 7.1 64-bit, Ubuntu 12.04, 14.04 & 15.10 32-/64-bit with Python 2.7 and 3.4. ▪ Microsoft Windows (10.0 Professional 64-bit, 10.0 Professional 32-bit, 8.1 Professional 32-bit, 8 Professional 32-bit, 7 Professional 32-bit with SP1, XP Professional 32-bit with SP3 and 2000 Professional 32-bit with SP2) with POSIX-like Cygwin plug-in and Python 2.6, 2.7, 3.2, 3.3 and 3.4. (NOTE: Windows 2000 came with obsolete Web Browser that precluded Windows 2000 updates and installaion of Cygwin. After copying Windows 2000 updates and Cygwin directory from XP, Windows 2000 ran the TeamSTARS "tsWxGTUI_PyVx" Toolkit CLI and GUI tests but did not suport mouse even with xterm.) ▪ UNIX (PC-BSD 9.2 & 11.0 64-bit without Parallels Tools, OpenIndiana 151a8 32-bit without Parallels Tools, a replacement for unreleased OpenSolaris 11/SunOS 5.11)

		with Python 2.6.4 running on Apple MacBook Pro
Apple 17" MacBook Pro Laptop	<p>2007-model year, 2.33 GHz Intel Core 2 Duo processor-based laptop with 4 GB RAM, 1920x1200 pixel resolution display and sufficient performance, resources and expansion capabilities to serve as the moderate-level baseline development and operator platform.</p> <ul style="list-style-type: none"> Its 160 GB (5400 RPM) SATA 1.5 Gb/s internal hard drive was used to run Apple's Mac OS X 10.7 and the hypervisor virtualization applications (Parallels Desktop 8 and VMware Fusion 5) that supported various guest operating systems. Its 1.5 TB (7200 RPM) SATA 3 Gb/s external hard drive was used to store and run configured versions of the Ubuntu Linux (12.04), Microsoft Windows (8 Pro, 7 Pro and XP Pro) and Unix (OpenSolaris 11/OpenIndiana 151) guest operating systems. Hewlett-Packard Company P2015DN Series (26A5C8) LaserJet Printer connected via Ethernet Hewlett-Packard Company Officejet Pro 8500A (A910) e-All-in-One Series Printer connected via Wi-Fi or USB. 	<p>Host Operating System</p> <ul style="list-style-type: none"> Apple's Mac OS X 10.7.5 with Python 2.6.8, 2.7.5, 3.2.2 <p>Host Hypervisor Virtualization Applications</p> <ul style="list-style-type: none"> Parallels Desktop 8 VMware Fusion 5 <p>Interchangeable Guest Operating Systems (configured to share host drives and use 1440 x 900 display)</p> <ul style="list-style-type: none"> Linux (Fedora 20 64-bit, OpenSuSE 12.2 64-bit, Scientific (CentOS) 6.4-6.5 64-bit, Ubuntu 12.04 32-bit) with Python 2.7 and 3.2. Windows (8.1 Professional 32-bit, 8 Professional 32-bit, 7 Professional 32-bit with SP1, XP Professional 32-bit with SP3) with POSIX-like Cygwin plug-in and Python 2.6, 2.7, 3.2 and 3.3 UNIX (PC-BSD 9.2-10.0 64-bit without Parallels Tools, OpenIndiana 151A8 32-bit without Parallels Tools, a replacement for unreleased OpenSolaris 11/SunOS 5.11) with Python 2.6.4 running on Apple MacBook Pro
Dell 15.6" Inspiron 7000 Laptop	<p>1998-model year, 366 MHz Intel Pentium II-based laptop with 384 MB RAM, 640x480/1024x768 pixel resolution display and expansion capabilities (marginal resources and performance) sufficient enough to serve as the low-level baseline development and operator platform.</p> <ul style="list-style-type: none"> Its interchangeable 32 GB (4200 RPM) ATA hard drives were used to run Windows XP Pro or Ubuntu 12.04 Linux. The platform's limited memory and available PCMCIA network adapters were incompatible with later versions of Windows or with other Linux distributions. (Windows XP recognized Xircom Ethernet and 3Com Wireless adapters; Ubuntu Linux 12.04 recognized only Linksys Wireless adapter.) Hewlett-Packard Company Photosmart C3180 All-in-One Printer, Scanner, Copier connected via USB. 	<p>Interchangeable Host Operating System</p> <ul style="list-style-type: none"> Windows XP Professional with SP3) with POSIX-like Cygwin plug-in and Python 2.6, 2.7, 3.2 and 3.3 Linux (Ubuntu 12.04) with Python 2.7 and 3.3

NOTE: Since cross-platform operating system and Python virtual machine technology is also available for

non-Intel based systems, it is likely that the TeamSTARS "tsWxGTUI_PyVx" toolkit will also work on those systems which use the equivalent operating systems and Python virtual machines with 32-bit and 64-bit microprocessors from other manufacturers including:

- *AMD*
- *ARM Holdings*
- *Cyrix*
- *Freescape*
- *Intel*
- *IBM*
- *Marvell*
- *NexGen*
- *Nvidia Tegra*
- *Oracle (previously Sun)*
- *OWC*
- *Qualcomm*
- *Rise Technology*
- *Samsung*
- *SigmaTel*
- *Texas Instruments*
- *Transmeta*
- *tilera*
- *Via (Centaur Technology division)*
- *winchip*

5.2.2 Previously Used Platforms (Release Notes)

Host Computer Platform Configurations previously used by "tsWxGTUI_PyVx" Toolkit developers:

Make & Model	Hardware	Software
Apple 17" MacBook Pro Laptop	<p>2007-model year, 2.33 GHz Intel Core 2 Duo processor-based laptop with 4 GB RAM and sufficient performance, resources and expansion capabilities to serve as the moderate-level baseline development and operator platform.</p> <ul style="list-style-type: none"> Its 160 GB (5400 RPM) SATA 1.5 Gb/s internal hard drive was used to run Apple's Mac OS X 10.7 and the hypervisor virtualization applications (Parallels Desktop 4-7 and VMware Fusion 4-5) that supported various guest operating systems. Its 1.5 TB (7200 RPM) SATA 3 Gb/s external hard drive was used to store and run configured versions of the Ubuntu Linux (10.04), Microsoft Windows (7 Pro and XP Pro) and Unix (OpenSolaris 11/OpenIndiana 151) guest operating systems. 	<p>Host Operating System</p> <ul style="list-style-type: none"> Apple's Mac OS X 10.4-10.7 with Python 2.6.8 <p>Host Hypervisor Virtualization Applications</p> <ul style="list-style-type: none"> Parallels Desktop 4-7 VMware Fusion 4-5 <p>Interchangeable Guest Operating Systems (configured to share host drives and use 1440 x 900 display)</p> <ul style="list-style-type: none"> Linux (Ubuntu 10.04) with Python 2.7 and 3.2 Windows (7 Professional / XP Professional with SP3) with POSIX-like Cygwin plug-in and Python 2.6, 2.7 and 3.2 UNIX (OpenIndiana 151a5, a replacement for unreleased OpenSolaris 11/SunOS 5.11) with Python 2.6.4 running on Apple MacBook Pro
Dell 15.6" Inspiron 7000 Laptop	<p>1998-model year, 366 MHz Intel Pentium II-based laptop with 384 MB RAM and expansion capabilities (marginal resources and performance) sufficient enough to serve as the low-level baseline development and operator platform.</p> <ul style="list-style-type: none"> Its interchangeable 32 GB (4200 RPM) ATA hard drives were used to run Windows XP Pro or Ubuntu 12.04 Linux. The platform's limited memory and available PCMCIA network adapters were incompatible with later versions of Windows or with other Linux distributions. (Windows XP recognized Xircom Ethernet and 3Com Wireless adapters; Ubuntu Linux 12.04 recognized only Linksys Wireless adapter.) 	<p>Interchangeable Host Operating System</p> <ul style="list-style-type: none"> Windows XP Professional with SP3) with POSIX-like Cygwin plug-in and Python 2.6, 2.7 and 3.2 Linux (Ubuntu 12.04) with Python 2.7 and 3.2

Notes - Baseline Host Computer Platform Configurations previously used by "tsWxGTUI_PyVx" Toolkit developers	<ol style="list-style-type: none">1 Linux (Fedora 15-16) with Python 2.6, 2.7 and 3.2 running on Apple MacBook Pro - 32-/64-bit Intel Core 2 Duo Processors running with 4 GB RAM Linux Virtual Machine created and managed by Parallels Desktop 6 for Mac2 Linux (openSuSE 11) with Python 2.6, 2.7 and 3.2 running on Apple MacBook Pro - 32-/64-bit Intel Core 2 Duo Processors with 4 GB RAM running Virtual Machine created and managed by Parallels Desktop 6 for Mac3 Linux (Ubuntu 10.04) with Python 2.7 and 3.2 running on Apple MacBook Pro - 32-/64-bit Intel Core 2 Duo Processors with 4 GB RAM running Linux Virtual Machine created and managed by Parallels Desktop 6 for Mac4 Mac OS X (Tiger 10.4.0-Snow Leopard 10.6.8) with Python 2.6, 2.7 and 3.2 running on Apple MacBook Pro - 32-/64-bit Intel Core 2 Duo Processors with 4 GB RAM running Mac OS 10.4-10.6.5 Windows (XP-SP3) with UNIX-like Cygwin plug-in and Python 2.6 running on Dell Laptop - 32-bit Intel Pentium 2 Processor with 384 MB RAM running Windows XP-SP36 Windows (XP-SP3 and Release 8 Preview) with UNIX-like Cygwin plug-in and Python 2.6 running on Apple MacBook Pro - 32-/64-bit Intel Core 2 Duo Processors with 4 GB RAM running Windows Virtual Machine created and managed by Parallels Desktop 6 for Mac
---	---

<p>Notes - Experimental Host Computer Platform Configurations previously used by "tsWxGTUI_PyVx" Toolkit developers</p>	<p>1 Windows (7 Professional) with built-in "Command Prompt" accessory shell and Python 2.7 running on Apple MacBook Pro - 32-/64-bit Intel Core 2 Duo Processors with 4 GB RAM running Windows Virtual Machine created and managed by Parallels Desktop (8) for Mac.</p>
	<p>This configuration uses the Windows built-in "Command Prompt" accessory shell which can run Command Line Interface components ("tsLibCLI", "tsTestsCLI" and "tsToolsCLI") of the "tsWxGTUI_PyVx" toolkit.</p> <p>It does NOT support the low-level, "nCurses" library, a pre-requisite for running the Graphical-style User Interface components ("tsLibGUI", "tsTestsGUI" and "tsToolsGUI") of the "tsWxGTUI_PyVx" toolkit.</p>
	<p>2 Windows (7 Professional) with UNIX-like Git Bash plug-in and Python 2.7 running on Apple MacBook Pro - 32-/64-bit Intel Core 2 Duo Processors with 4 GB RAM running Windows Virtual Machine created and managed by Parallels Desktop (8) for Mac.</p> <p>This configuration, like those using Cygwin, includes a Bash shell which can run Command Line Interface components ("tsLibCLI", "tsTestsCLI" and "tsToolsCLI") of the "tsWxGTUI_PyVx" toolkit.</p> <p>Unlike those configurations using Cygwin, it does NOT support the low-level, "nCurses" library, a pre-requisite for running the Graphical-style User Interface components ("tsLibGUI", "tsTestsGUI" and "tsToolsGUI") of the "tsWxGTUI_PyVx" toolkit.</p>
	<p>3 Windows (7 Professional) and Python 2.7 with the GNUwin32 and PDCurses Version 2.6 plug-ins running on Apple MacBook Pro - 32-/64-bit Intel Core 2 Duo Processors with 4 GB RAM running Windows Virtual Machine created and managed by Parallels Desktop (8) for Mac.</p> <p>This configuration, unlike those using Cygwin, includes a DOS-like Command Prompt shell which can run Command Line Interface components ("tsLibCLI", "tsTestsCLI" and "tsToolsCLI") of the "tsWxGTUI_PyVx" toolkit.</p> <p>Like those configurations using Cygwin, PDCurses Version 2.6 does support the low-level, Python "Curses" library, a pre-requisite for running the Graphical-style User Interface components ("tsLibGUI", "tsTestsGUI" and "tsToolsGUI") of the "tsWxGTUI_PyVx" toolkit. It runs the test_PDCurses (renamed test_tsWxCurses) application. However, PDCurses Version 2.6 traps because it lacks the mouse button definitions needed by the "tsWxGraphicalTextUserInterface" module to emulate "wxPython" in order to run such applications as "test_tsWxWidgets.py".</p>

Draft

5.3 APPENDIX B - API RELATIONSHIP

An Application Programming Interface (API) is a source code based specification intended to be used as an interface by software components to communicate with each other. An API may include specifications for routines, data structures, object classes, and variables.

- 1 High Level API** - A programming interface that is the least detailed but provides more functionality within one statement than a lower-level interface. High-level interfaces are designed to enable the programmer to write code in a shorter amount of time and to be less involved with the details of the software module or hardware device that is performing the required services. For example, the programmer can invoke a high level procedure to append one or more text strings to an internal buffer.
- 2 Low Level API** - A programming interface that is the most detailed, allowing the programmer to manipulate functions within a software module or within hardware at a very granular level. To continue with the afore mentioned example, a high level procedure can then invoke one or more low level procedures that will sequentially get one text string at a time from the internal buffer, sequentially scroll the top most string off the display, then scroll up each remaining displayed string and finally outputting the new string to the bottom row of the screen.
- 3 Extended API** - A customized version of an existing programming interface that supports additional keyword value pairs and positional arguments. For example, the "tsWxGTUI_PyVx" Toolkit is a character-mode emulation of the pixel-mode "wxPython" API. It internally may require optional parameters to distinguish such features as character-mode dimensions from their pixel-mode counterparts. It may also include functionality that "wxPython" and its underlying "wxWidgets" toolkit otherwise obtain from the host operating system and its programming libraries, such as the installation of the color palette and the implementation of scrollable windows.

5.3.1 High, Low and Extended API Relationships

The following table describes the initial conceptual purpose, scope, capabilities, limitations and relationship between the High-level, Low-Level and Extended APIs associated with the "tsWxGTUI_PyVx" Toolkit and its third-party components.

An Application Programming Interface (API) is a source code based specification intended to be used as an interface by software components to communicate with each other. An API may include specifications for routines, data structures, object classes, and variables.

- 1 High Level API** - A programming interface that is the least detailed but provides more functionality within one statement than a lower-level interface. High-level interfaces are designed to enable the programmer to write code in a shorter amount of time and to be less involved with the details of the software module or hardware device that is performing the required services. For example, the programmer can invoke a high level procedure to append one or more text strings to an internal buffer.

- 2 Low Level API** - A programming interface that is the most detailed, allowing the programmer to manipulate functions within a software module or within hardware at a very granular level. To continue with the afore mentioned example, a high level procedure can then invoke one or more low level procedures that will sequentially get one text string at a time from the internal buffer, sequentially scroll the top most string off the display, then scroll up each remaining displayed string and finally outputting the new string to the bottom row of the screen.
- 3 Extended API** - A customized version of an existing programming interface that supports additional keyword value pairs and positional arguments. For example, the "tsWxGTUI_PyVx" Toolkit is a character-mode emulation of the pixel-mode "wxPython" API. It internally may require optional parameters to distinguish such features as character-mode dimensions from their pixel-mode counterparts. It may also include functionality that "wxPython" and its underlying "wxWidgets" toolkit otherwise obtain from the host operating system and its programming libraries, such as the installation of the color palette and the implementation of scrollable windows.

Draft

5.4 APPENDIX C - DELIVERABLES

The following table describes the work product(s) to be delivered by the developer:

DELIVERABLE	DESCRIPTION
Software Package(s)	<p>The "tsWxGTUI_PyVx" Toolkit software:</p> <ul style="list-style-type: none"> ▪ Building Block Library Packages ("tsLibCLI" & "tsLibGUI") in Python 2x & Python 3x source code formats ▪ Tool Application Programs ("tsToolsCLI" & "tsToolsGUI") in Python 2x & Python 3x source code formats ▪ Test Application Programs ("tsTestsCLI" & "tsTestsGUI") in Python 2x & Python 3x source code formats ▪ Utility Modules ("tsUtilities") in Python 2x and Python 3x and POSIX-style shell script formats
Engineering Documentation ("tsEngineering") (Optional, Fee-based subscription)	<p>While the <i>TeamSTARS</i> "tsWxGTUI_PyVx" Toolkit is released as free, open source software, the engineering documentation establishes proprietary Copyright ownership by the original Toolkit Author(s).</p> <p>Toolkit recipients can obtain a license for the engineering documentation, if their intent is to assemble a large team to commercialize the software and want to kickstart the effort.</p> <p>The "tsEngineering" subdirectory contains a collection of Toolit Author written large, complex documents, including structured documents, featuring multiple fonts and graphic images.</p> <p>The documents are authored using the following products:</p> <ul style="list-style-type: none"> ▪ Author-it --- A help authoring tool (http://www.author-it.com) and content management system for creating, maintaining, and distributing single-sourced content. It generates Microsoft Word files with the appropriate outline numbering (for table of contents, sections, paragraphs, lists and figures) regardless of where the single-source content originated. ▪ Microsoft Office --- A collection of software applications (http://www.microsoftstore.com/store/msusa/en_US/cat/Office/categoryID.62684700) intended to be used by knowledge workers. The components are generally distributed together, have a consistent user interface and usually can interact with each other, sometimes in ways that the operating system would not normally allow. <ul style="list-style-type: none"> Access (data base) Excel (spreadsheet) PowerPoint (presentation) Word (word processor) ▪ Microsoft Visio --- A software application (http://www.microsoftstore.com/store/msusa/en_US/list/Visio/categoryID.62687700) intended to be used by knowledge workers.

DELIVERABLE	DESCRIPTION
Software Package(s)	<p>The "tsWxGTUI_PyVx" Toolkit software:</p> <ul style="list-style-type: none"> ▪ Building Block Library Packages ("tsLibCLI" & "tsLibGUI") in Python 2x & Python 3x source code formats ▪ Tool Application Programs ("tsToolsCLI" & "tsToolsGUI") in Python 2x & Python 3x source code formats ▪ Test Application Programs ("tsTestsCLI" & "tsTestsGUI") in Python 2x & Python 3x source code formats ▪ Utility Modules ("tsUtilities") in Python 2x and Python 3x and POSIX-style shell script formats
	<p>Visio (diagrams)</p> <ul style="list-style-type: none"> ▪ Fineprint Software (http://fineprint.com) <p>FinePrint (A print driver, for Microsoft Windows, which allows you to manage the printing process. It helps you to reduce printing costs while enhancing and managing complex print jobs.)</p> <p>pdfFactory Pro (An application, for Microsoft Windows, that provides Adobe PDF compatible output.)</p> <p>The documents accompany the computer software for the training and reference use of the software developer:</p> <ul style="list-style-type: none"> ▪ It explains the purpose, goals, non-goals, system requirements, interface requirements, software requirements, qualification requirements, development plan, software design, how it operates and how to install, use and troubleshoot it. ▪ It is provided in various application specific formats (such as Adobe PDF and Microsoft Office & Visio formats which may be read and edited by the free, open source, cross-platform LibreOffice Suite). <p>The "tsWxGTUI_PyVx" Toolkit software development and release documents are deliverable in Adobe's Portable Document Format and Microsoft's Word Format (with associated bit-mapped images in JPG or PNG Format):</p> <ul style="list-style-type: none"> ▪ Vol. 0 --- SDIST Announcement ▪ Vol. 1 --- SDIST Brochure ▪ Vol. 2 --- SDIST Introduction ▪ Vol. 3 --- SDIST Terms & Conditions ▪ Vol. 4 --- SDIST Software Development Plans ▪ Vol. 5 --- SDIST System Specification ▪ Vol. 6 --- SDIST Interface Requirements Specification ▪ Vol. 7 --- SDIST Software Requirements Specification ▪ Vol. 8 --- SDIST Release Notes ▪ Vol. 9 --- SDIST Software User's Manual ▪ Vol. 10 --- SDIST Appendixes ▪ Vol. 11 --- SDIST Dictionary

DELIVERABLE	DESCRIPTION
Software Package(s)	<p>The "tsWxGTUI_PyVx" Toolkit software:</p> <ul style="list-style-type: none"> ▪ Building Block Library Packages ("tsLibCLI" & "tsLibGUI") in Python 2x & Python 3x source code formats ▪ Tool Application Programs ("tsToolsCLI" & "tsToolsGUI") in Python 2x & Python 3x source code formats ▪ Test Application Programs ("tsTestsCLI" & "tsTestsGUI") in Python 2x & Python 3x source code formats ▪ Utility Modules ("tsUtilities") in Python 2x and Python 3x and POSIX-style shell script formats
	<ul style="list-style-type: none"> ▪ Vol. 12 --- SDIST To-Do List
Engineering Notes	<p>The "tsWxGTUI_PyVx" Toolkit software development notes in Microsoft's Word Format:</p> <ul style="list-style-type: none"> ▪ Class List.doc ▪ Relationships_Between_tsWxGTUI_Toolkit_APIs.doc ▪ RGB to Color Name Mapping(Triplet and Hex).doc ▪ Writing a Python Package by Tarek Ziadé.doc <p>The "tsWxGTUI_PyVx" Toolkit software development notes in Microsoft's Access Format:</p> <ul style="list-style-type: none"> ▪ tsWxPython.mdb <p>The "tsWxGTUI_PyVx" Toolkit software development diagram notes in Microsoft's Visio Format:</p> <ul style="list-style-type: none"> ▪ Distributed System.vsd ▪ Networked Architecture.vsd ▪ System Architecture.vsd ▪ tsWxPython Org Chart.vsd <p>The "tsWxGTUI_PyVx" Toolkit software development notes in Microsoft's Excel Format:</p> <ul style="list-style-type: none"> ▪ Platform_Configuration.xls <p>The "tsWxGTUI_PyVx" Toolkit software development notes in Plain Text Format:</p> <ul style="list-style-type: none"> ▪ README_BMP.txt
Equipment Operator Documentation ("tsDocuments")	<p>The "tsWxGTUI_PyVx" Toolkit installation, configuration, operation and troubleshooting documents in Plain Text Format:</p> <ul style="list-style-type: none"> ▪ tsDocCLI-1-GettingStartedFiles ▪ tsDocCLI-2-DistributionReadMeFiles ▪ tsDocCLI-3-DeveloperReadMeFiles ▪ tsDocCLI-5-UsageTermsAndConditions ▪ tsManPages ("tsLibCLI", "tsLibGUI", "tsToolsCLI", "tsToolsGUI", "tsTestsCLI", "tsTestsGUI") <p>1. Operator Documentation</p> <p>User documentation for each "tsWxGTUI" Toolkit distribution is contained, in plain text format, in the subdirectory named "/tsDocCLI". The subdirectory is organized, by topic, to contain the following:</p>

DELIVERABLE	DESCRIPTION
Software Package(s)	<p>The "tsWxGTUI_PyVx" Toolkit software:</p> <ul style="list-style-type: none"> ▪ Building Block Library Packages ("tsLibCLI" & "tsLibGUI") in Python 2x & Python 3x source code formats ▪ Tool Application Programs ("tsToolsCLI" & "tsToolsGUI") in Python 2x & Python 3x source code formats ▪ Test Application Programs ("tsTestsCLI" & "tsTestsGUI") in Python 2x & Python 3x source code formats ▪ Utility Modules ("tsUtilities") in Python 2x and Python 3x and POSIX-style shell script formats
	<p>a. How to Prepare Platform to Get Started</p> <p>The "./GettingStartedFiles" subdirectory contains the following file(s):</p> <p>"README-GettingStarted.txt" ---</p> <p>b. How to Install and Redistribute</p> <p>The "./DistributionReadMeFiles" subdirectory contains the following file(s):</p> <ul style="list-style-type: none"> ▪ "AUTHORS.txt" --- List of the principal "tsWxGTUI" Toolkit author(s) and authors credited for work covered by a prior copyright and license. ▪ "BUGS.txt" --- List of Known Problems / Issues. ▪ "CHANGE_LOG.txt" --- List of Additions, Modification and Deletions. ▪ "CONFIGURE.txt" --- Instructions for applying factory and site-specific configurations. ▪ "COPYING.txt" --- Instructions for copying all or a portion of the distribution. ▪ "FAQ.txt" --- Answers to Frequently Asked Questions. ▪ "INSTALL.txt" --- Describes steps to download, extract install and configure the "tsWxGUI" Toolkit. ▪ "LICENSE.txt" --- General and special arrangements, provisions, rules, specifications and standards that form an integral part the agreement or contract between the creator and recipient of Copyrighted and Licensed Work. ▪ "MANIFEST.txt" --- Tally List for deliverable items. ▪ "NEWS.txt" --- Announcements of new releases. ▪ "NOTICES.txt" --- Details the copyright(s) and license(s). ▪ "OPERATE.txt" --- Describes steps to use the "tsWxGUI" Toolkit. ▪ "README.txt" --- Introduces new recipients to the purpose, goals, non-goals, design and features of the computer software product. ▪ "THANKS.txt" --- Acknowledgments to those otherwise unsung heros who contributed time and effort to supporting the authors as planners, editors, designers, coders and testers. ▪ "TO-DO.txt" --- A To-Do-List provides a roadmap for development and troubleshooting work. ▪ "TROUBLE SHOOT.txt" --- Provides a list of available reference resources and a guide for planning, developing and troubleshooting a cross-platform system of hundreds of files each containing a few, tens or hundred of class, data and method definitions. Its complexity becomes apparent in the recent software Lines-Of-Code

DELIVERABLE	DESCRIPTION
Software Package(s)	<p>The "tsWxGTUI_PyVx" Toolkit software:</p> <ul style="list-style-type: none"> ▪ Building Block Library Packages ("tsLibCLI" & "tsLibGUI") in Python 2x & Python 3x source code formats ▪ Tool Application Programs ("tsToolsCLI" & "tsToolsGUI") in Python 2x & Python 3x source code formats ▪ Test Application Programs ("tsTestsCLI" & "tsTestsGUI") in Python 2x & Python 3x source code formats ▪ Utility Modules ("tsUtilities") in Python 2x and Python 3x and POSIX-style shell script formats
	<p>metrics.</p> <p>c. How to Use and Modify</p> <p>The "./DeveloperReadMeFiles" subdirectory contains the following file(s):</p> <ul style="list-style-type: none"> ▪ "README.txt" ▪ "README_1st.txt" ▪ "README_1st-PyPI-Dev-tsWxGTUI.txt" ▪ "README-01-Title_Page.txt" ▪ "README-02-Table_Of_Contents.txt" ▪ "README-03-Purpose.txt" ▪ "README-04-Goals.txt" ▪ "README-05-Non-Goals.txt" ▪ "README-06-Design_Strategy.txt" ▪ "README-07-Design_Architecture.txt" ▪ "README-08-Software_Configuration_Management.txt" ▪ "README-09-tsWxGTUI_Directories.txt" ▪ "README-10-tsToolkitCLI_Directories.txt" ▪ "README-11-tsToolkitGUI_Directories.txt" ▪ "README-12-Features.txt" ▪ "README-13-Current_Capabilities.txt" ▪ "README-14-Current_Limitations.txt" ▪ "README-15-Reference_Documents.txt" <p>d. How to Learn "tsWxGTUI" Toolkit Software Development</p> <p>The "./DeveloperTutorialFiles" subdirectory contains the following file(s):</p> <ul style="list-style-type: none"> ▪ "CLI_0_hello_world_print_statement.py" ▪ "CLI_1_hello_world_print_function.py" ▪ "CLI_2_hello_world_script_environment.py" ▪ "CLI_3_hello_world_main_module_application.py" ▪ "GUI_4_Curses_LowLevel_WidgetApi_application.py" ▪ "GUI_5_tsWxGTUI_HighLevel_WidgetApi_application.py"

DELIVERABLE	DESCRIPTION
Software Package(s)	<p>The "tsWxGTUI_PyVx" Toolkit software:</p> <ul style="list-style-type: none">▪ Building Block Library Packages ("tsLibCLI" & "tsLibGUI") in Python 2x & Python 3x source code formats▪ Tool Application Programs ("tsToolsCLI" & "tsToolsGUI") in Python 2x & Python 3x source code formats▪ Test Application Programs ("tsTestsCLI" & "tsTestsGUI") in Python 2x & Python 3x source code formats▪ Utility Modules ("tsUtilities") in Python 2x and Python 3x and POSIX-style shell script formats
	<ul style="list-style-type: none">▪ "GUI_6_tsWxGTUI_HighLevel_BoxSizerApi_application.py"▪ "ReadMe_Developer_Tutorial_Files.txt"▪ "test_tsCxGlobals.py"▪ "test_tsWxGlobals.py"▪ "tsCxGlobals.py" <p>e. Terms & Conditions</p> <p>The "./UsageTermsAndConditions" subdirectory contains the following file(s):</p> <ul style="list-style-type: none">▪ "COPYRIGHT.txt"▪ "LICENSE.txt"▪ "NOTICES.txt"▪ "SplashScreenDesignersGuide.txt"

CHAPTER 6

6 SOFTWARE USERS MANUAL

6.1.1.1 In This Chapter

APPENDIX A - BASELINE HOST COMPUTER PLATFORMS	435
APPENDIX B - API RELATIONSHIP	445
APPENDIX C - DELIVERABLES	449
APPENDIX D - LOG FILES	455

Draft

Draft

6.2 APPENDIX A - BASELINE HOST COMPUTER PLATFORMS

This section shall identify those host computer platforms actually used to plan, implement, test, document, deploy and maintain the "tsWxGTUI_PyVx" Toolkit.

- 1** *Currently Used Platforms* (see "*Currently Used Platforms (Release Notes)*" on page 8) - Linux (Fedora 22 / OpenSUSE 13.2 / Scientific (CentOS) 7 / Ubuntu 12.04, 14.04 & 15.04), Mac OS X (10.7 / 10.11), Microsoft Windows (7 Pro. 32-bit with Cygwin 1.7.x / 10 Pro. 64-bit with Cygwin 2.2) / and Unix (PC-BSD 10 & OpenIndiana 151a8 / Solaris 11 / SunOS 5.11).
-

The Toolkit's Python source code has been developed and tested only with Intel x86 and x64 processors and representative GNU/Linux, Mac OS X, Microsoft Windows and Unix operating system releases.

Its source code has been compiled, interpreted and executed by Python 2x and 3x Virtual Machines developed by the Python Software Foundation (PSF).

The PSF also distributes equivalent Python 2x and 3x Virtual Machines (and the source code to build them) for other processor types and operating systems, some of which are listed below.

See file `"/tsWxGTUI_PyVx_Repository/Notebooks/EngineeringNotebook/MS-Excel-Files/Platform_Configuration.xls"` for Python version specific platform configuration details (Sheet 1 --- Host Configurations; Sheet 2 --- Terminal Configurations):

- a) 2013-model year, 3.5 GHz Intel Quad Core i7 processor-based desktop with 16 GB RAM and sufficient performance, resources and expansion capabilities to serve as the high-level baseline development and operator platform.
 - b) 2007-model year, 2.33 GHz Intel Core 2 Duo processor-based laptop with 4 GB RAM and sufficient performance, resources and expansion capabilities to serve as the moderate-level baseline development and operator platform.
 - c) 1998-model year, 366 MHz Intel Pentium II-based laptop with 384 MB RAM and expansion capabilities (marginal resources and performance) sufficient enough to serve as the low-level baseline development and operator platform.
- 2** *Previously Used Platforms* (see "*Previously Used Platforms (Release Notes)*" on page 13) - Linux (Fedora 15 & 16 / Open SuSE 11 / Ubuntu 10.04), Mac OS X (10.4 / 10.5 / 10.6 / 10.7), Microsoft Windows (8 Professional / 7 Professional / XP Professional each with Cygwin 1.7.x) and Unix (OpenIndiana 151a6 / Solaris 11 / SunOS 5.11).
- a) 2007-model year, 2.33 GHz Intel Core 2 Duo processor-based laptop with 4 GB RAM and sufficient performance, resources and expansion capabilities to serve as the moderate-level baseline development and operator platform.
 - b) 1998-model year, 366 MHz Intel Pentium II-based laptop with 384 MB RAM and expansion capabilities (marginal resources and performance) sufficient enough to serve as the low-level baseline development and operator platform.

- 3** *Designing Embedded Systems with Linux and Python* (on page 16) - Link to YouTube Video by Mark Kohler.

6.2.1 Currently Used Platforms (Release Notes)

Host Computer Platform Configurations currently used by "tsWxGTUI_PyVx" Toolkit developers:

Draft

Draft

Make & Model	Hardware	Software
Apple 27" iMac Desktop	<p>2013-model year, 3.5 GHz Intel Quad Core i7 processor-based desktop with 16 GB RAM, 2560x1440 pixel resolution display and sufficient performance, resources and expansion capabilities to serve as the high-level baseline development and operator platform.</p> <ul style="list-style-type: none"> ▪ Its 3 TB (7200 RPM) SATA 6 Gb/s internal hard drive had 128 GB Solid State Flash memory and was used to run Apple's Mac OS X 10.9-19.10 and the hypervisor virtualization applications (Parallels Desktop 9-10 and VMware Fusion 5 and 7.1) that supported various guest operating systems. ▪ Its 3 TB (7200 RPM) SATA 6 Gb/s internal hard drive was also used to store and run configured versions of the eComStation Demo CD version 2.2 BETA (IBM OS/2 4.5 Warp descendant), CentOS Linux (7.0), Fedora Linux (21), OpenSUSE Linux (13.1), Scientific Linux (6.5), Ubuntu Linux (12.04, 14.04), Microsoft Windows (8.1 Pro, 8 Pro, 7 Pro, XP Pro and 2000 Pro), and Unix (PC-BSD (9.2, 10.0), OpenIndiana 151A8) guest operating systems. ▪ Hewlett-Packard Company P2015DN Series (26A5C8) LaserJet Printer conected via Ethernet ▪ Hewlett-Packard Company Officejet Pro 8500A (A910) e-All-in-One Series Printer connected viia Wi-Fi or USB. 	<p>Host Operating System</p> <ul style="list-style-type: none"> ▪ Apple's Mac OS X (initially Mavericks releases 10.9.x and currently Yosemite releases 10.10.x) with Python 2.6.8, 2.7.5, 3.2.2 <p>Host Hypervisor Virtualization Applications</p> <ul style="list-style-type: none"> ▪ Parallels Desktop 9-11 ▪ VMware Fusion 5 & 7.1 <p>Concurrent or Interchangeable Guest Operating Systems (cloned from Apple 17" MacBook Pro Laptop and configured to share host drives and use 1440 x 900 display)</p> <ul style="list-style-type: none"> ▪ eComStation Demo CD version 2.2 BETA (IBM OS/2 4.5 Warp descendant) runs only what was shipped on the live CD and does not include Python. Most recent Python port for OS/2 & eComStations is Python 2.7.5. For details, see the following: "http://os2ports.smedley.id.au/index.php?page=python" and "http://blog.python.org/2011/05/python-33-to-drop-support-for-os2.html" ▪ Linux (CentOS 7.1 64-bit, Debian 8.2 64-bit, Fedora 22 & 23 64-bit, LXLE 14.02 32-bit, OpenSUSE 12.2 & 13.2 64-bit, Scientific 6.4 & 7.1 64-bit, Ubuntu 12.04, 14.04 & 15.10 32-/64-bit with Python 2.7 and 3.4. ▪ Microsoft Windows (10.0 Professional 64-bit, 10.0 Professional 32-bit, 8.1 Professional 32-bit, 8 Professional 32-bit, 7 Professional 32-bit with SP1, XP Professional 32-bit with SP3 and 2000 Professional 32-bit with SP2) with POSIX-like Cygwin plug-in and Python 2.6, 2.7, 3.2, 3.3 and 3.4. (NOTE: Windows 2000 came with obsolete Web Browser that precluded Windows 2000 updates and installaion of Cygwin. After copying Windows 2000 updates and Cygwin directory from XP, Windows 2000 ran the TeamSTARS "tsWxGTUI_PyVx" Toolkit CLI and GUI tests but did not suport mouse even with xterm.) ▪ UNIX (PC-BSD 9.2 & 11.0 64-bit without Parallels Tools, OpenIndiana 151a8 32-bit without Parallels Tools, a replacement for unreleased OpenSolaris 11/SunOS 5.11)

		with Python 2.6.4 running on Apple MacBook Pro
Apple 17" MacBook Pro Laptop	<p>2007-model year, 2.33 GHz Intel Core 2 Duo processor-based laptop with 4 GB RAM, 1920x1200 pixel resolution display and sufficient performance, resources and expansion capabilities to serve as the moderate-level baseline development and operator platform.</p> <ul style="list-style-type: none"> Its 160 GB (5400 RPM) SATA 1.5 Gb/s internal hard drive was used to run Apple's Mac OS X 10.7 and the hypervisor virtualization applications (Parallels Desktop 8 and VMware Fusion 5) that supported various guest operating systems. Its 1.5 TB (7200 RPM) SATA 3 Gb/s external hard drive was used to store and run configured versions of the Ubuntu Linux (12.04), Microsoft Windows (8 Pro, 7 Pro and XP Pro) and Unix (OpenSolaris 11/OpenIndiana 151) guest operating systems. Hewlett-Packard Company P2015DN Series (26A5C8) LaserJet Printer connected via Ethernet Hewlett-Packard Company Officejet Pro 8500A (A910) e-All-in-One Series Printer connected via Wi-Fi or USB. 	<p>Host Operating System</p> <ul style="list-style-type: none"> Apple's Mac OS X 10.7.5 with Python 2.6.8, 2.7.5, 3.2.2 <p>Host Hypervisor Virtualization Applications</p> <ul style="list-style-type: none"> Parallels Desktop 8 VMware Fusion 5 <p>Interchangeable Guest Operating Systems (configured to share host drives and use 1440 x 900 display)</p> <ul style="list-style-type: none"> Linux (Fedora 20 64-bit, OpenSuSE 12.2 64-bit, Scientific (CentOS) 6.4-6.5 64-bit, Ubuntu 12.04 32-bit) with Python 2.7 and 3.2. Windows (8.1 Professional 32-bit, 8 Professional 32-bit, 7 Professional 32-bit with SP1, XP Professional 32-bit with SP3) with POSIX-like Cygwin plug-in and Python 2.6, 2.7, 3.2 and 3.3 UNIX (PC-BSD 9.2-10.0 64-bit without Parallels Tools, OpenIndiana 151A8 32-bit without Parallels Tools, a replacement for unreleased OpenSolaris 11/SunOS 5.11) with Python 2.6.4 running on Apple MacBook Pro
Dell 15.6" Inspiron 7000 Laptop	<p>1998-model year, 366 MHz Intel Pentium II-based laptop with 384 MB RAM, 640x480/1024x768 pixel resolution display and expansion capabilities (marginal resources and performance) sufficient enough to serve as the low-level baseline development and operator platform.</p> <ul style="list-style-type: none"> Its interchangeable 32 GB (4200 RPM) ATA hard drives were used to run Windows XP Pro or Ubuntu 12.04 Linux. The platform's limited memory and available PCMCIA network adapters were incompatible with later versions of Windows or with other Linux distributions. (Windows XP recognized Xircom Ethernet and 3Com Wireless adapters; Ubuntu Linux 12.04 recognized only Linksys Wireless adapter.) Hewlett-Packard Company Photosmart C3180 All-in-One Printer, Scanner, Copier connected via USB. 	<p>Interchangeable Host Operating System</p> <ul style="list-style-type: none"> Windows XP Professional with SP3) with POSIX-like Cygwin plug-in and Python 2.6, 2.7, 3.2 and 3.3 Linux (Ubuntu 12.04) with Python 2.7 and 3.3

NOTE: Since cross-platform operating system and Python virtual machine technology is also available for

non-Intel based systems, it is likely that the TeamSTARS "tsWxGTUI_PyVx" toolkit will also work on those systems which use the equivalent operating systems and Python virtual machines with 32-bit and 64-bit microprocessors from other manufacturers including:

- *AMD*
- *ARM Holdings*
- *Cyrix*
- *Freescape*
- *Intel*
- *IBM*
- *Marvell*
- *NexGen*
- *Nvidia Tegra*
- *Oracle (previously Sun)*
- *OWC*
- *Qualcomm*
- *Rise Technology*
- *Samsung*
- *SigmaTel*
- *Texas Instruments*
- *Transmeta*
- *tilera*
- *Via (Centaur Technology division)*
- *winchip*

6.2.2 Previously Used Platforms (Release Notes)

Host Computer Platform Configurations previously used by "tsWxGTUI_PyVx" Toolkit developers:

Make & Model	Hardware	Software
Apple 17" MacBook Pro Laptop	<p>2007-model year, 2.33 GHz Intel Core 2 Duo processor-based laptop with 4 GB RAM and sufficient performance, resources and expansion capabilities to serve as the moderate-level baseline development and operator platform.</p> <ul style="list-style-type: none"> Its 160 GB (5400 RPM) SATA 1.5 Gb/s internal hard drive was used to run Apple's Mac OS X 10.7 and the hypervisor virtualization applications (Parallels Desktop 4-7 and VMware Fusion 4-5) that supported various guest operating systems. Its 1.5 TB (7200 RPM) SATA 3 Gb/s external hard drive was used to store and run configured versions of the Ubuntu Linux (10.04), Microsoft Windows (7 Pro and XP Pro) and Unix (OpenSolaris 11/OpenIndiana 151) guest operating systems. 	<p>Host Operating System</p> <ul style="list-style-type: none"> Apple's Mac OS X 10.4-10.7 with Python 2.6.8 <p>Host Hypervisor Virtualization Applications</p> <ul style="list-style-type: none"> Parallels Desktop 4-7 VMware Fusion 4-5 <p>Interchangeable Guest Operating Systems (configured to share host drives and use 1440 x 900 display)</p> <ul style="list-style-type: none"> Linux (Ubuntu 10.04) with Python 2.7 and 3.2 Windows (7 Professional / XP Professional with SP3) with POSIX-like Cygwin plug-in and Python 2.6, 2.7 and 3.2 UNIX (OpenIndiana 151a5, a replacement for unreleased OpenSolaris 11/SunOS 5.11) with Python 2.6.4 running on Apple MacBook Pro
Dell 15.6" Inspiron 7000 Laptop	<p>1998-model year, 366 MHz Intel Pentium II-based laptop with 384 MB RAM and expansion capabilities (marginal resources and performance) sufficient enough to serve as the low-level baseline development and operator platform.</p> <ul style="list-style-type: none"> Its interchangeable 32 GB (4200 RPM) ATA hard drives were used to run Windows XP Pro or Ubuntu 12.04 Linux. The platform's limited memory and available PCMCIA network adapters were incompatible with later versions of Windows or with other Linux distributions. (Windows XP recognized Xircom Ethernet and 3Com Wireless adapters; Ubuntu Linux 12.04 recognized only Linksys Wireless adapter.) 	<p>Interchangeable Host Operating System</p> <ul style="list-style-type: none"> Windows XP Professional with SP3) with POSIX-like Cygwin plug-in and Python 2.6, 2.7 and 3.2 Linux (Ubuntu 12.04) with Python 2.7 and 3.2

Notes - Baseline Host Computer Platform Configurations previously used by "tsWxGTUI_PyVx" Toolkit developers	<ol style="list-style-type: none">1 Linux (Fedora 15-16) with Python 2.6, 2.7 and 3.2 running on Apple MacBook Pro - 32-/64-bit Intel Core 2 Duo Processors running with 4 GB RAM Linux Virtual Machine created and managed by Parallels Desktop 6 for Mac2 Linux (openSuSE 11) with Python 2.6, 2.7 and 3.2 running on Apple MacBook Pro - 32-/64-bit Intel Core 2 Duo Processors with 4 GB RAM running Virtual Machine created and managed by Parallels Desktop 6 for Mac3 Linux (Ubuntu 10.04) with Python 2.7 and 3.2 running on Apple MacBook Pro - 32-/64-bit Intel Core 2 Duo Processors with 4 GB RAM running Linux Virtual Machine created and managed by Parallels Desktop 6 for Mac4 Mac OS X (Tiger 10.4.0-Snow Leopard 10.6.8) with Python 2.6, 2.7 and 3.2 running on Apple MacBook Pro - 32-/64-bit Intel Core 2 Duo Processors with 4 GB RAM running Mac OS 10.4-10.6.5 Windows (XP-SP3) with UNIX-like Cygwin plug-in and Python 2.6 running on Dell Laptop - 32-bit Intel Pentium 2 Processor with 384 MB RAM running Windows XP-SP36 Windows (XP-SP3 and Release 8 Preview) with UNIX-like Cygwin plug-in and Python 2.6 running on Apple MacBook Pro - 32-/64-bit Intel Core 2 Duo Processors with 4 GB RAM running Windows Virtual Machine created and managed by Parallels Desktop 6 for Mac
---	---

<p>Notes - Experimental Host Computer Platform Configurations previously used by "tsWxGTUI_PyVx" Toolkit developers</p>	<p>1 Windows (7 Professional) with built-in "Command Prompt" accessory shell and Python 2.7 running on Apple MacBook Pro - 32-/64-bit Intel Core 2 Duo Processors with 4 GB RAM running Windows Virtual Machine created and managed by Parallels Desktop (8) for Mac.</p> <hr/> <p>This configuration uses the Windows built-in "Command Prompt" accessory shell which can run Command Line Interface components ("tsLibCLI", "tsTestsCLI" and "tsToolsCLI") of the "tsWxGTUI_PyVx" toolkit.</p> <p>It does NOT support the low-level, "nCurses" library, a pre-requisite for running the Graphical-style User Interface components ("tsLibGUI", "tsTestsGUI" and "tsToolsGUI") of the "tsWxGTUI_PyVx" toolkit.</p>
	<p>2 Windows (7 Professional) with UNIX-like Git Bash plug-in and Python 2.7 running on Apple MacBook Pro - 32-/64-bit Intel Core 2 Duo Processors with 4 GB RAM running Windows Virtual Machine created and managed by Parallels Desktop (8) for Mac.</p> <hr/> <p>This configuration, like those using Cygwin, includes a Bash shell which can run Command Line Interface components ("tsLibCLI", "tsTestsCLI" and "tsToolsCLI") of the "tsWxGTUI_PyVx" toolkit.</p> <p>Unlike those configurations using Cygwin, it does NOT support the low-level, "nCurses" library, a pre-requisite for running the Graphical-style User Interface components ("tsLibGUI", "tsTestsGUI" and "tsToolsGUI") of the "tsWxGTUI_PyVx" toolkit.</p>
	<p>3 Windows (7 Professional) and Python 2.7 with the GNUwin32 and PDCurses Version 2.6 plug-ins running on Apple MacBook Pro - 32-/64-bit Intel Core 2 Duo Processors with 4 GB RAM running Windows Virtual Machine created and managed by Parallels Desktop (8) for Mac.</p> <hr/> <p>This configuration, unlike those using Cygwin, includes a DOS-like Command Prompt shell which can run Command Line Interface components ("tsLibCLI", "tsTestsCLI" and "tsToolsCLI") of the "tsWxGTUI_PyVx" toolkit.</p> <p>Like those configurations using Cygwin, PDCurses Version 2.6 does support the low-level, Python "Curses" library, a pre-requisite for running the Graphical-style User Interface components ("tsLibGUI", "tsTestsGUI" and "tsToolsGUI") of the "tsWxGTUI_PyVx" toolkit. It runs the test_PDCurses (renamed test_tsWxCurses) application. However, PDCurses Version 2.6 traps because it lacks the mouse button definitions needed by the "tsWxGraphicalTextUserInterface" module to emulate "wxPython" in order to run such applications as "test_tsWxWidgets.py".</p>

Draft

6.3 APPENDIX B - API RELATIONSHIP

An Application Programming Interface (API) is a source code based specification intended to be used as an interface by software components to communicate with each other. An API may include specifications for routines, data structures, object classes, and variables.

- 1 High Level API** - A programming interface that is the least detailed but provides more functionality within one statement than a lower-level interface. High-level interfaces are designed to enable the programmer to write code in a shorter amount of time and to be less involved with the details of the software module or hardware device that is performing the required services. For example, the programmer can invoke a high level procedure to append one or more text strings to an internal buffer.
- 2 Low Level API** - A programming interface that is the most detailed, allowing the programmer to manipulate functions within a software module or within hardware at a very granular level. To continue with the afore mentioned example, a high level procedure can then invoke one or more low level procedures that will sequentially get one text string at a time from the internal buffer, sequentially scroll the top most string off the display, then scroll up each remaining displayed string and finally outputting the new string to the bottom row of the screen.
- 3 Extended API** - A customized version of an existing programming interface that supports additional keyword value pairs and positional arguments. For example, the "tsWxGTUI_PyVx" Toolkit is a character-mode emulation of the pixel-mode "wxPython" API. It internally may require optional parameters to distinguish such features as character-mode dimensions from their pixel-mode counterparts. It may also include functionality that "wxPython" and its underlying "wxWidgets" toolkit otherwise obtain from the host operating system and its programming libraries, such as the installation of the color palette and the implementation of scrollable windows.

6.3.1 Comparison of wxPython with tsWxGTUI (Development Plan)

This tabulation shall briefly compare the features, capabilities and limitations of the pixel-mode "wxPython" / "wxWidgets" / "wxEmbedded" Toolkit with those of its character-mode emulator, the "tsWxGTUI_PyVx" Toolkit.

Feature	"wxPython" / "wxWidgets" / "wxEmbedded"	"tsWxGTUI_PyVx"
Platform	Locally (via system console or xterm) and remotely (via vnc) accessible systems: <ul style="list-style-type: none"> ▪ Desktop ▪ Laptop ▪ Workstation ▪ Embedded (via KOAN's "wxEmbedded") 	Locally (via system console or xterm) and remotely (via xterm) accessible systems: <ul style="list-style-type: none"> ▪ Desktop ▪ Laptop ▪ Workstation ▪ Embedded
Operator Desktop Interface Hardware	<ul style="list-style-type: none"> ▪ Keyboard ▪ Mouse, trackball, touchpad or touchscreen ▪ Optional Mouseless 	<ul style="list-style-type: none"> ▪ Keyboard ▪ Mouse, trackball, touchpad or touchscreen ▪ Keyboard Shortcut Key with/without Optional Mouse (Future)

Feature	"wxPython" / "wxWidgets" / "wxEmbedded"	"tsWxGTUI_PyVx"
	<ul style="list-style-type: none"> ▪ Display (pixel mode) 	<ul style="list-style-type: none"> ▪ Display (character mode)
Operating System & Device Driver Software	<ul style="list-style-type: none"> ▪ Linux ▪ Mac OS X ▪ Microsoft Windows ▪ Unix 	<ul style="list-style-type: none"> ▪ Linux ▪ Mac OS X ▪ Microsoft Windows (with free add-on of POSIX-like Cygwin Command Line Interface and GNU tools from Red Hat) ▪ Unix
Terminal Device Interface Software	<ul style="list-style-type: none"> ▪ "wxWidgets" internally uses host Operating System specific API for its Terminal Device Interface. ▪ It translates host Operating System specific events into "wxWidget" specific published API events. ▪ Events include keyboard key press / release, mouse button press / release and single / double click with mouse position at every clocked sample. ▪ Events also include window resizing. 	<ul style="list-style-type: none"> ▪ "nCurses" internally uses host Operating System specific API for its Terminal Device Interface. ▪ It translates host Operating System specific events into "nCurses" specific published API events. ▪ Events include keyboard key press / release, mouse button press / release and single / double / triple click with mouse position ONLY available at time of button state change. ▪ Events also include window resizing. However, event handling is currently limited to trapping the unsupported event. Though re-initializing "nCurses" to detect and use the new size is feasible and fast (50 milliseconds or more depending on host processor and terminal color palette (and associated definition of or mapping of wxPython color palette), it does not address the time consuming (20 seconds or more on host processor) complexities associated with re-initializing the "wxPython" emulation and the System Operator designated application program.
Programming Language	<ul style="list-style-type: none"> ▪ "wxWidgets" is implemented in C++ ▪ "wxPython" binding/wrapper for "wxWidgets" is implemented with SWIG in Python 2.x and Python 3.x 	<ul style="list-style-type: none"> ▪ "tsWxGTUI_PyVx" is implemented in Python 2.x ▪ After debugging, "tsWxGTUI_PyVx" is ported from Python 2.x to Python 3.x via the standard Python "2to3" utility with minor manual debugging when appropriate
Terminal Interface Software	<ul style="list-style-type: none"> ▪ "wxWidgets" internally uses Operating System or X window system specific API for Graphical User Interface. 	<ul style="list-style-type: none"> ▪ "tsWxGTUI_PyVx" internally uses Python Curses ("nCurses") module API for Graphical-style User Interface. The Python Curses module only implements the most commonly used subset features of the "nCurses" API. ▪ The "tsWxGTUI_PyVx" Toolkit emulates a typical Operating System or X window system specific API for Graphical User Interface.

Feature	"wxPython" / "wxWidgets" / "wxEmbedded"	"tsWxGTUI_PyVx"
Operator Desktop Interface Software	<ul style="list-style-type: none"> Host Operating System specific Command Line Interface is a POSIX compatible shell (such as bash) Host Operating System specific Graphical User Interface features support multiple independently re-sizable and repositionable Frames and Dialogs Host Operating System specific Graphical User Interface features reflect proprietary placement of labels and buttons to iconize, maximize/restore and close frames and dialogs Host Operating System specific Graphical User Interface task bar features support the closing of individual Frames and Dialogs Host Operating System specific task bar features support the shifting of foreground focus from one Frame or Dialog to another 	<ul style="list-style-type: none"> Host Operating System specific Command Line Interface is a POSIX compatible shell (such as bash) Host Operating System specific Graphical User Interface shell features support multiple independently re-sizable and repositionable Command Line Interface shell windows "tsWxGTUI_PyVx" emulated Graphical User Interface features DO NOT support multiple independently re-sizable and repositionable Frames and Dialogs WITHOUT the System Operator first creating separate Command Line Interface shell windows "tsWxGTUI_PyVx" emulated Graphical User Interface features reflect Microsoft Windows-style placement of labels and buttons to iconize, maximize/restore and close frames and dialogs "tsWxGTUI_PyVx" emulated Graphical User Interface task bar features DO NOT currently support the closing of individual Frames and Dialogs "tsWxGTUI_PyVx" Toolkit task bar features (future enhancement) support the shifting of foreground focus from one Frame or Dialog to another
Application Programming Interface	<ul style="list-style-type: none"> "wxWidgets" / "wxPython" API for Graphical User Interface supports bit-mapped images. It supports fixed and variable sized fonts and at least the 68 most commonly used colors. 	<ul style="list-style-type: none"> "tsWxGTUI_PyVx" emulated subset of "wxWidgets" / "wxPython" API for Graphical-style User Interface DOES NOT support bitmapped images except for the single, predefined bitmapped image used as a splash screen at startup It supports the 1-color (single terminal-dependant green, orange or white phosphor) that is "ON" or "OFF" (black) associated with a non-color, VT100/VT220 terminal hardware and terminal emulator software. It supports a single fixed sized font and at least the 68 most commonly used colors (which are either internally mapped into the "nCurses" standard 8-color, 16-color or defined for optional 88-color and 256-color terminal hardware and terminal emulator software). It supports a 71-color palette by augmenting the 68-color wxPython palette with the 3 colors that must be supported for the xterm-16color palette.

Feature	"wxPython" / "wxWidgets" / "wxEmbedded"	"tsWxGTUI_PyVx"
		<ul style="list-style-type: none"> It supports a 140-color palette by augmenting the 68-color wxPython palette with the 3 colors unique to the xterm-16color palette plus 69 other colors that demonstrate the customization potential. However, the constraint on the number of color pairs (32767) constrains the number of colors to be no more than 181 (square root of 32768) unless the design is re-designed to use a sparse matrix to avoid impractical color combinations.

6.3.2 High, Low and Extended API Relationships

The following table describes the initial conceptual purpose, scope, capabilities, limitations and relationship between the High-level, Low-Level and Extended APIs associated with the "tsWxGTUI_PyVx" Toolkit and its third-party components.

An Application Programming Interface (API) is a source code based specification intended to be used as an interface by software components to communicate with each other. An API may include specifications for routines, data structures, object classes, and variables.

- 1 High Level API** - A programming interface that is the least detailed but provides more functionality within one statement than a lower-level interface. High-level interfaces are designed to enable the programmer to write code in a shorter amount of time and to be less involved with the details of the software module or hardware device that is performing the required services. For example, the programmer can invoke a high level procedure to append one or more text strings to an internal buffer.
- 2 Low Level API** - A programming interface that is the most detailed, allowing the programmer to manipulate functions within a software module or within hardware at a very granular level. To continue with the afore mentioned example, a high level procedure can then invoke one or more low level procedures that will sequentially get one text string at a time from the internal buffer, sequentially scroll the top most string off the display, then scroll up each remaining displayed string and finally outputting the new string to the bottom row of the screen.
- 3 Extended API** - A customized version of an existing programming interface that supports additional keyword value pairs and positional arguments. For example, the "tsWxGTUI_PyVx" Toolkit is a character-mode emulation of the pixel-mode "wxPython" API. It internally may require optional parameters to distinguish such features as character-mode dimensions from their pixel-mode counterparts. It may also include functionality that "wxPython" and its underlying "wxWidgets" toolkit otherwise obtain from the host operating system and its programming libraries, such as the installation of the color palette and the implementation of scrollable windows.

6.4 APPENDIX C - DELIVERABLES

The following table describes the work product(s) to be delivered by the developer:

DELIVERABLE	DESCRIPTION
Software Package(s)	<p>The "tsWxGTUI_PyVx" Toolkit software:</p> <ul style="list-style-type: none"> ▪ Building Block Library Packages ("tsLibCLI" & "tsLibGUI") in Python 2x & Python 3x source code formats ▪ Tool Application Programs ("tsToolsCLI" & "tsToolsGUI") in Python 2x & Python 3x source code formats ▪ Test Application Programs ("tsTestsCLI" & "tsTestsGUI") in Python 2x & Python 3x source code formats ▪ Utility Modules ("tsUtilities") in Python 2x and Python 3x and POSIX-style shell script formats
Engineering Documentation ("tsEngineering") (Optional, Fee-based subscription)	<p>While the <i>TeamSTARS</i> "tsWxGTUI_PyVx" Toolkit is released as free, open source software, the engineering documentation establishes proprietary Copyright ownership by the original Toolkit Author(s).</p> <p>Toolkit recipients can obtain a license for the engineering documentation, if their intent is to assemble a large team to commercialize the software and want to kickstart the effort.</p> <p>The "tsEngineering" subdirectory contains a collection of Toolit Author written large, complex documents, including structured documents, featuring multiple fonts and graphic images.</p> <p>The documents are authored using the following products:</p> <ul style="list-style-type: none"> ▪ Author-it --- A help authoring tool (http://www.author-it.com) and content management system for creating, maintaining, and distributing single-sourced content. It generates Microsoft Word files with the appropriate outline numbering (for table of contents, sections, paragraphs, lists and figures) regardless of where the single-source content originated. ▪ Microsoft Office --- A collection of software applications (http://www.microsoftstore.com/store/msusa/en_US/cat/Office/categoryID.62684700) intended to be used by knowledge workers. The components are generally distributed together, have a consistent user interface and usually can interact with each other, sometimes in ways that the operating system would not normally allow. <ul style="list-style-type: none"> Access (data base) Excel (spreadsheet) PowerPoint (presentation) Word (word processor) ▪ Microsoft Visio --- A software application (http://www.microsoftstore.com/store/msusa/en_US/list/Visio/categoryID.62687700) intended to be used by knowledge workers.

DELIVERABLE	DESCRIPTION
Software Package(s)	<p>The "tsWxGTUI_PyVx" Toolkit software:</p> <ul style="list-style-type: none"> ▪ Building Block Library Packages ("tsLibCLI" & "tsLibGUI") in Python 2x & Python 3x source code formats ▪ Tool Application Programs ("tsToolsCLI" & "tsToolsGUI") in Python 2x & Python 3x source code formats ▪ Test Application Programs ("tsTestsCLI" & "tsTestsGUI") in Python 2x & Python 3x source code formats ▪ Utility Modules ("tsUtilities") in Python 2x and Python 3x and POSIX-style shell script formats
	<p>Visio (diagrams)</p> <ul style="list-style-type: none"> ▪ Fineprint Software (http://fineprint.com) <p>FinePrint (A print driver, for Microsoft Windows, which allows you to manage the printing process. It helps you to reduce printing costs while enhancing and managing complex print jobs.)</p> <p>pdfFactory Pro (An application, for Microsoft Windows, that provides Adobe PDF compatible output.)</p> <p>The documents accompany the computer software for the training and reference use of the software developer:</p> <ul style="list-style-type: none"> ▪ It explains the purpose, goals, non-goals, system requirements, interface requirements, software requirements, qualification requirements, development plan, software design, how it operates and how to install, use and troubleshoot it. ▪ It is provided in various application specific formats (such as Adobe PDF and Microsoft Office & Visio formats which may be read and edited by the free, open source, cross-platform LibreOffice Suite). <p>The "tsWxGTUI_PyVx" Toolkit software development and release documents are deliverable in Adobe's Portable Document Format and Microsoft's Word Format (with associated bit-mapped images in JPG or PNG Format):</p> <ul style="list-style-type: none"> ▪ Vol. 0 --- SDIST Announcement ▪ Vol. 1 --- SDIST Brochure ▪ Vol. 2 --- SDIST Introduction ▪ Vol. 3 --- SDIST Terms & Conditions ▪ Vol. 4 --- SDIST Software Development Plans ▪ Vol. 5 --- SDIST System Specification ▪ Vol. 6 --- SDIST Interface Requirements Specification ▪ Vol. 7 --- SDIST Software Requirements Specification ▪ Vol. 8 --- SDIST Release Notes ▪ Vol. 9 --- SDIST Software User's Manual ▪ Vol. 10 --- SDIST Appendixes ▪ Vol. 11 --- SDIST Dictionary

DELIVERABLE	DESCRIPTION
Software Package(s)	<p>The "tsWxGTUI_PyVx" Toolkit software:</p> <ul style="list-style-type: none"> ▪ Building Block Library Packages ("tsLibCLI" & "tsLibGUI") in Python 2x & Python 3x source code formats ▪ Tool Application Programs ("tsToolsCLI" & "tsToolsGUI") in Python 2x & Python 3x source code formats ▪ Test Application Programs ("tsTestsCLI" & "tsTestsGUI") in Python 2x & Python 3x source code formats ▪ Utility Modules ("tsUtilities") in Python 2x and Python 3x and POSIX-style shell script formats
	<ul style="list-style-type: none"> ▪ Vol. 12 --- SDIST To-Do List
Engineering Notes	<p>The "tsWxGTUI_PyVx" Toolkit software development notes in Microsoft's Word Format:</p> <ul style="list-style-type: none"> ▪ Class List.doc ▪ Relationships_Between_tsWxGTUI_Toolkit_APIs.doc ▪ RGB to Color Name Mapping(Triplet and Hex).doc ▪ Writing a Python Package by Tarek Ziadé.doc <p>The "tsWxGTUI_PyVx" Toolkit software development notes in Microsoft's Access Format:</p> <ul style="list-style-type: none"> ▪ tsWxPython.mdb <p>The "tsWxGTUI_PyVx" Toolkit software development diagram notes in Microsoft's Visio Format:</p> <ul style="list-style-type: none"> ▪ Distributed System.vsd ▪ Networked Architecture.vsd ▪ System Architecture.vsd ▪ tsWxPython Org Chart.vsd <p>The "tsWxGTUI_PyVx" Toolkit software development notes in Microsoft's Excel Format:</p> <ul style="list-style-type: none"> ▪ Platform_Configuration.xls <p>The "tsWxGTUI_PyVx" Toolkit software development notes in Plain Text Format:</p> <ul style="list-style-type: none"> ▪ README_BMP.txt
Equipment Operator Documentation ("tsDocuments")	<p>The "tsWxGTUI_PyVx" Toolkit installation, configuration, operation and troubleshooting documents in Plain Text Format:</p> <ul style="list-style-type: none"> ▪ tsDocCLI-1-GettingStartedFiles ▪ tsDocCLI-2-DistributionReadMeFiles ▪ tsDocCLI-3-DeveloperReadMeFiles ▪ tsDocCLI-5-UsageTermsAndConditions ▪ tsManPages ("tsLibCLI", "tsLibGUI", "tsToolsCLI", "tsToolsGUI", "tsTestsCLI", "tsTestsGUI") <p>1. Operator Documentation</p> <p>User documentation for each "tsWxGTUI" Toolkit distribution is contained, in plain text format, in the subdirectory named ".tsDocCLI". The subdirectory is organized, by topic, to contain the following:</p>

DELIVERABLE	DESCRIPTION
Software Package(s)	<p>The "tsWxGTUI_PyVx" Toolkit software:</p> <ul style="list-style-type: none"> ▪ Building Block Library Packages ("tsLibCLI" & "tsLibGUI") in Python 2x & Python 3x source code formats ▪ Tool Application Programs ("tsToolsCLI" & "tsToolsGUI") in Python 2x & Python 3x source code formats ▪ Test Application Programs ("tsTestsCLI" & "tsTestsGUI") in Python 2x & Python 3x source code formats ▪ Utility Modules ("tsUtilities") in Python 2x and Python 3x and POSIX-style shell script formats
	<p>a. How to Prepare Platform to Get Started</p> <p>The "./GettingStartedFiles" subdirectory contains the following file(s):</p> <p>"README-GettingStarted.txt" ---</p> <p>b. How to Install and Redistribute</p> <p>The "./DistributionReadMeFiles" subdirectory contains the following file(s):</p> <ul style="list-style-type: none"> ▪ "AUTHORS.txt" --- List of the principal "tsWxGTUI" Toolkit author(s) and authors credited for work covered by a prior copyright and license. ▪ "BUGS.txt" --- List of Known Problems / Issues. ▪ "CHANGE_LOG.txt" --- List of Additions, Modification and Deletions. ▪ "CONFIGURE.txt" --- Instructions for applying factory and site-specific configurations. ▪ "COPYING.txt" --- Instructions for copying all or a portion of the distribution. ▪ "FAQ.txt" --- Answers to Frequently Asked Questions. ▪ "INSTALL.txt" --- Describes steps to download, extract install and configure the "tsWxGUI" Toolkit. ▪ "LICENSE.txt" --- General and special arrangements, provisions, rules, specifications and standards that form an integral part the agreement or contract between the creator and recipient of Copyrighted and Licensed Work. ▪ "MANIFEST.txt" --- Tally List for deliverable items. ▪ "NEWS.txt" --- Announcements of new releases. ▪ "NOTICES.txt" --- Details the copyright(s) and license(s). ▪ "OPERATE.txt" --- Describes steps to use the "tsWxGUI" Toolkit. ▪ "README.txt" --- Introduces new recipients to the purpose, goals, non-goals, design and features of the computer software product. ▪ "THANKS.txt" --- Acknowledgments to those otherwise unsung heros who contributed time and effort to supporting the authors as planners, editors, designers, coders and testers. ▪ "TO-DO.txt" --- A To-Do-List provides a roadmap for development and troubleshooting work. ▪ "TROUBLE SHOOT.txt" --- Provides a list of available reference resources and a guide for planning, developing and troubleshooting a cross-platform system of hundreds of files each containing a few, tens or hundred of class, data and method definitions. Its complexity becomes apparent in the recent software Lines-Of-Code

DELIVERABLE	DESCRIPTION
Software Package(s)	<p>The "tsWxGTUI_PyVx" Toolkit software:</p> <ul style="list-style-type: none"> ▪ Building Block Library Packages ("tsLibCLI" & "tsLibGUI") in Python 2x & Python 3x source code formats ▪ Tool Application Programs ("tsToolsCLI" & "tsToolsGUI") in Python 2x & Python 3x source code formats ▪ Test Application Programs ("tsTestsCLI" & "tsTestsGUI") in Python 2x & Python 3x source code formats ▪ Utility Modules ("tsUtilities") in Python 2x and Python 3x and POSIX-style shell script formats
	<p>metrics.</p> <p>c. How to Use and Modify</p> <p>The "./DeveloperReadMeFiles" subdirectory contains the following file(s):</p> <ul style="list-style-type: none"> ▪ "README.txt" ▪ "README_1st.txt" ▪ "README_1st-PyPI-Dev-tsWxGTUI.txt" ▪ "README-01-Title_Page.txt" ▪ "README-02-Table_Of_Contents.txt" ▪ "README-03-Purpose.txt" ▪ "README-04-Goals.txt" ▪ "README-05-Non-Goals.txt" ▪ "README-06-Design_Strategy.txt" ▪ "README-07-Design_Architecture.txt" ▪ "README-08-Software_Configuration_Management.txt" ▪ "README-09-tsWxGTUI_Directories.txt" ▪ "README-10-tsToolkitCLI_Directories.txt" ▪ "README-11-tsToolkitGUI_Directories.txt" ▪ "README-12-Features.txt" ▪ "README-13-Current_Capabilities.txt" ▪ "README-14-Current_Limitations.txt" ▪ "README-15-Reference_Documents.txt" <p>d. How to Learn "tsWxGTUI" Toolkit Software Development</p> <p>The "./DeveloperTutorialFiles" subdirectory contains the following file(s):</p> <ul style="list-style-type: none"> ▪ "CLI_0_hello_world_print_statement.py" ▪ "CLI_1_hello_world_print_function.py" ▪ "CLI_2_hello_world_script_environment.py" ▪ "CLI_3_hello_world_main_module_application.py" ▪ "GUI_4_Curses_LowLevel_WidgetApi_application.py" ▪ "GUI_5_tsWxGTUI_HighLevel_WidgetApi_application.py"

DELIVERABLE	DESCRIPTION
Software Package(s)	<p>The "tsWxGTUI_PyVx" Toolkit software:</p> <ul style="list-style-type: none">▪ Building Block Library Packages ("tsLibCLI" & "tsLibGUI") in Python 2x & Python 3x source code formats▪ Tool Application Programs ("tsToolsCLI" & "tsToolsGUI") in Python 2x & Python 3x source code formats▪ Test Application Programs ("tsTestsCLI" & "tsTestsGUI") in Python 2x & Python 3x source code formats▪ Utility Modules ("tsUtilities") in Python 2x and Python 3x and POSIX-style shell script formats
	<ul style="list-style-type: none">▪ "GUI_6_tsWxGTUI_HighLevel_BoxSizerApi_application.py"▪ "ReadMe_Developer_Tutorial_Files.txt"▪ "test_tsCxGlobals.py"▪ "test_tsWxGlobals.py"▪ "tsCxGlobals.py" <p>e. Terms & Conditions</p> <p>The "./UsageTermsAndConditions" subdirectory contains the following file(s):</p> <ul style="list-style-type: none">▪ "COPYRIGHT.txt"▪ "LICENSE.txt"▪ "NOTICES.txt"▪ "SplashScreenDesignersGuide.txt"

6.5 APPENDIX D - LOG FILES

6.5.1 Log File Examples

- *Bit-Mapped Image Tree (Splash Screen Files for Graphical Applications)* (on page 456)
- *Logs Tree (Files for Non-Graphical Applications)* (on page 458)
- *Logs Tree (Files for Graphical Applications)* (on page 467)

Draft

6.5.1.1 Bit-Mapped Image Tree (Splash Screen Files for Graphical Applications)

```
# File: ".logs/bmp/README_BMP.txt"
# "Time-stamp: <01/05/2015  2:14:20 AM rsg>"
```

This "bmp" directory contains those Splash Screen(s) generated by:

```
"/tsWxGTUI_Py2x/tsLibGUI/tsWxPkg/src/tsWxGraphicalTextUserInterface.py."
```

A Splash Screen is displayed during the launch of a GUI-style application program. It identifies the application's author(s), copyright(s) and licence(s) using information defined in:

```
"/tsWxGTUI_Py2x/tsLibGUI/tsWxPkg/src/tsWxGlobals.py"
```

Splash Screens are named for the display size (in character columns and rows/lines), terminal/emulator type, and host operating system.

Examples:

Basic Multi-Color ("wxPython" transformation maps
68-color palette into 8 or 16 built-in colors)

Base Name	Size	Type	Host OS	File Ext.	Notes
"SplashScreen-[60x15_CYGWIN]-cygwin_nt-5.0.bmp"					(Placeholder for Windows 2000)
"SplashScreen-[60x15_CYGWIN]-cygwin_nt-5.1.bmp"					(Windows XP)
"SplashScreen-[60x15_CYGWIN]-cygwin_nt-5.2.bmp"					(Placeholder for Windows XP x64)
"SplashScreen-[60x15_CYGWIN]-cygwin_nt-6.0.bmp"					(Placeholder for Windows Vista)
"SplashScreen-[60x15_CYGWIN]-cygwin_nt-6.1.bmp"					(Windows 7)
"SplashScreen-[60x15_LINUX]-cygwin_nt-6.1.bmp"					(Windows 7)
"SplashScreen-[60x15_XTERM]-cygwin_nt-6.1.bmp"					(Windows 7)
"SplashScreen-[80x25_XTERM]-darwin.bmp"					(Mac OS 10.9.1)
"SplashScreen-[96x40_XTERM]-linux.bmp"					(Fedora 21)
"SplashScreen-[96x40_XTERM]-linux.bmp"					(Ubuntu 12.04)
"SplashScreen-[128x50_XTERM]-freebsd.bmp"					(PC-BSD 9.2)
"SplashScreen-[128x50_XTERM]-sunos.bmp"					(OpenIndiana 151a8)
"SplashScreen-[60x15_XTERM-COLOR]-cygwin_nt-6.1.bmp"					(Windows 7)
"SplashScreen-[80x15_XTERM-16COLOR]-cygwin_nt-6.3.bmp"					(Windows 8.1)
"SplashScreen-[80x15_XTERM-88COLOR]-cygwin_nt-6.3.bmp"					(16x16 color pair limit for Windows 8.1)
"SplashScreen-[80x15_XTERM-256COLOR]-cygwin_nt-6.3.bmp"					(16x16 color pair limit for Windows 8.1)
"SplashScreen-[80x15_XTERM-256COLOR]-cygwin_nt-6.4.bmp"					(16x16 color pair limit for Windows 10)

Non-Color ("wxPython" transformation maps 68-color palette into black with one shade of the default color for displays having only a white, orange or green phosphor).

Base Name	Size	Type	Host OS	File Ext.	Notes
"SplashScreen-[80x40_VT100]-cygwin_nt-5.0.bmp"					(Placeholder for Windows 2000)
"SplashScreen-[80x40_VT100]-cygwin_nt-5.1.bmp"					(Windows XP)
"SplashScreen-[80x40_VT100]-cygwin_nt-5.2.bmp"					(Placeholder for Windows XP x64)
"SplashScreen-[80x40_VT100]-cygwin_nt-6.0.bmp"					(Placeholder for Windows Vista)
"SplashScreen-[80x15_VT100]-cygwin_nt-6.1.bmp"					(Windows 7)
"SplashScreen-[80x15_VT100]-cygwin_nt-6.2.bmp"					(Windows 8)
"SplashScreen-[80x15_VT220]-cygwin_nt-6.3.bmp"					(Windows 8.1)
"SplashScreen-[80x15_VT220]-cygwin_nt-6.4.bmp"					(Windows 10)

```
#####
# Advanced Multi-Color support is still evolving.... #
# # #
# Terminal/Emulator standards support an undocumented #
# maximum of 256 color pairs. This inference resulted #
# from the following observations. #
# # #
# Under Python 2.x and Python 3.x, most xterm-88color #
# and xterm-256color terminal emulators produced the #
# wrong colors marred by spurious underline artifacts. #
# # #
# Under Python 3.3.0 with a Yosemite Mac OS X Terminal #
# utility, there were 256 color pairs and a 16-color #
# palette that worked. #
# # #
# Also, under the Python 2.7.6 with the GNOME Desktop #
# for CentOS 7.0 Linux, there were 256 color pairs and #
# a 16-color palette that worked . #
# # #
# For the color pair matrix used by the "tsWxGTUI" #
# Toolkit, the Advanced Multi-Color support emulates #
# xterm-16color and associated mapping of the wxPython #
# 68-color palette into the built-in 16 colors. #
#####
```

```
if (COLOR_PAIRS > 256) and (USE_256_COLOR_PAIR_LIMIT): # As set in
tsWxGlobals.py
```

```
    Advanced Multi-Color ("wxPython" emulation maps
    68-color palette into 16 of 88 or 16 of 256 colors)
```

```
elif (COLOR_PAIRS == 256) # As reported by curses
```

```
    Advanced Multi-Color ("wxPython" emulation maps
    68-color palette into 16 of 88 or 16 of 256 colors)
```

```
else:
```

```
    Advanced Multi-Color ("wxPython" emulation maps
    68-color palette into 71 of 88 or 140 of 256 colors)
```

Base Name	Size	Type	Host OS	File Ext.	Notes
"SplashScreen-[80x15_XTERM-88COLOR]-cygwin_nt-6.3.bmp"					(Windows 8.1)
"SplashScreen-[80x25_XTERM-88COLOR]-Darwin.bmp"					(Mac OS 10.9.1)
"SplashScreen-[96x40_XTERM-88COLOR]-linux.bmp"					(Fedora 20)
"SplashScreen-[80x15_XTERM-256COLOR]-cygwin_nt-6.3.bmp"					(Windows 8.1)
"SplashScreen-[80x25_XTERM-256COLOR]-Darwin.bmp"					(Mac OS 10.9.1)
"SplashScreen-[96x40_XTERM-256COLOR]-linux.bmp"					(Fedora 20)

A new Splash Screen is built upon the first use of a uniquely sized command line interface display by those host operating systems that share this directory.

NOTE:

Previous terminal emulator used in a command line interface shell can alter the built-in color palette for subsequent terminal emulators.
Examples:

- 1) The final xterm sees no color palette change if the first one is xterm followed by xterm-color, vt100 and xterm.
- 2) The final xterm sees a dim color palette change if the first one is xterm followed by by xterm-16color, vt100 and xterm.

Keeping a copy here avoids spending time to rebuild the same Splash Screen each time a GUI-style application program is launched.

You may recover disk space by deleting those Splash Screens that have outlived their usefulness.

6.5.1.2 Logs Tree (Files for Non-Graphical Applications)

The "tsWxGTUI_PyVx" Toolkit registers, via log files, the startup, configuration and chronology of activities needed for design verification and troubleshooting.

- 1 The root directory is named "logs" and is always placed in the directory from which the application has been launched. The root directory is created, if it does not already exist.

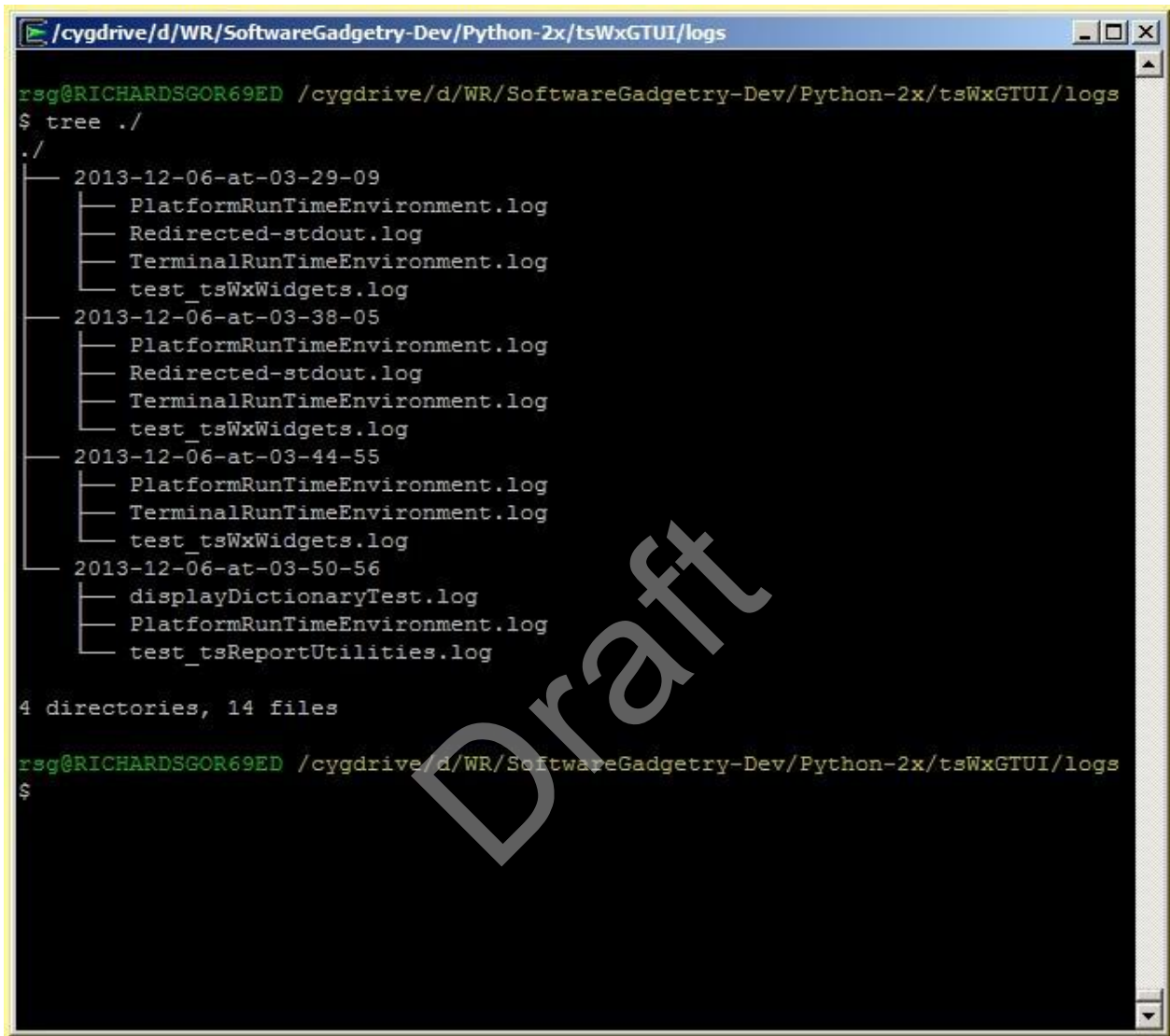
- a) Within the "logs" directory is a date and time stamped directory for each application startup. The startup directory is always created.
- b) Within the startup directory are the application-specific log files.

"PlatformRunTimeEnvironment.log" - Captures current hardware, software and network information about the run time environment for the user process.

"<application>.log(s)" - Captures date and time stamped messages associated with application designated normal and abnormal situations and activities. The contents and verbosity of

"<application>.log(s)" are controlled by configuration settings (for details see Customizable CLI Features).

- 2 It is left to the System Administrator, Software Engineer, System Operator and Field Service personnel to do the housekeeping that removes those log file when they are no longer useful.



```
/cygdrive/d/WR/SoftwareGadgetry-Dev/Python-2x/tsWxGTUI/logs
rsg@RICHARDSGOR69ED /cygdrive/d/WR/SoftwareGadgetry-Dev/Python-2x/tsWxGTUI/logs
$ tree ./
./
├── 2013-12-06-at-03-29-09
│   ├── PlatformRunTimeEnvironment.log
│   ├── Redirected-stdout.log
│   ├── TerminalRunTimeEnvironment.log
│   └── test_tsWxWidgets.log
├── 2013-12-06-at-03-38-05
│   ├── PlatformRunTimeEnvironment.log
│   ├── Redirected-stdout.log
│   ├── TerminalRunTimeEnvironment.log
│   └── test_tsWxWidgets.log
├── 2013-12-06-at-03-44-55
│   ├── PlatformRunTimeEnvironment.log
│   ├── TerminalRunTimeEnvironment.log
│   └── test_tsWxWidgets.log
└── 2013-12-06-at-03-50-56
    ├── displayDictionaryTest.log
    ├── PlatformRunTimeEnvironment.log
    └── test_tsReportUtilities.log

4 directories, 14 files
rsg@RICHARDSGOR69ED /cygdrive/d/WR/SoftwareGadgetry-Dev/Python-2x/tsWxGTUI/logs
$
```

6.5.1.2.1 2013-12-06-at-03-50-56**6.5.1.2.1.1 PlatformRunTimeEnvironment.log**

tsPlatformRunTimeEnvironment.py, v2.1.0 (build 10/19/2013)

Author(s): Richard S. Gordon
Copyright (c) 2007-2013 Richard S. Gordon.
All rights reserved.
GNU General Public License, Version 3, 29 June 2007

Credits:

tsLibCLI Import & Application Launch Features:
Copyright (c) 2007-2009 Frederick A. Kier.
All rights reserved.
GNU General Public License, Version 3, 29 June 2007

terminalsize (<https://gist.github.com/jtriley/1108174>) Features:
Copyright (c) 2011 Justin T. Riley.
All rights reserved.
GNU General Public License, Version 3, 29 June 2007

Python Platform & Logging Module API Features:
Copyright (c) 2001-2013 Python Software Foundation.
All rights reserved.
PSF License Agreement for Python 2.7.3 & 3.3.0

===== Begin Platform Run Time Environment =====

Reported Fri, 06-Dec-2013 at 03:50:58

Network Identification

hostname = <RICHARDSGOR69ED>
aliaslist = <[]> # NOTE: List of Values NOT available.
ipaddrlist = <['fe80::5079:ab53:c361:23a4']>

Host Central Processing Unit

machine = <i686>
processor = <> # NOTE: Value NOT available.
architecture = <32> bits; <> # NOTE: Value NOT available. linkage
byteorder = <little>

Host Operating System

api = <posix>
system = <CYGWIN_NT-6.1>
release = <1.7.25(0.270/5/3)>

```
version = <2013-08-31 20:39>
```

Host Console Display Size

```
-----
characters/line = <80>
lines/display = <35>
```

Python Platform

```
-----
branch = <> # NOTE: Value NOT available.
build = <default> number; <Dec 18 2012 13:50:09> date
compiler = <GCC 4.5.3>
implementation = <CPython>
revision = <> # NOTE: Value NOT available.
version = <2.7.3>
```

Process Parameters

```
-----
pid = <900496> / <0xDBD90>
getppid = <891332> / <0xD99C4>
getegid = <513>
geteuid = <1000>
getgid = <513>
getgroups = <[545, 1001, 513]>
getpgid = <900496>
getuid = <1000>
getlogin = <rsg>
ctermid = </dev/pty0>
cwd = </cygdrive/d/WR/SoftwareGadgetry-Dev/Python-2x/tsWxGTUI>
```

Environment Variables

```
-----
!:: = <::\>
ALLUSERSPROFILE = <C:\ProgramData>
APPDATA = <C:\Users\rsg\AppData\Roaming>
COMMONPROGRAMFILES = <C:\Program Files\Common Files>
COMPUTERNAME = <RICHARDSGOR69ED>
COMSPEC = <C:\Windows\system32\cmd.exe>
EMAIL = <D:\Mail\The Bat!>
FP_NO_HOST_CHECK = <NO>
HOME = </home/rsg>
HOMEDRIVE = <d:>
HOMEPATH = <\Home\rsg>
HOSTNAME = <RICHARDSGOR69ED>
INFOPATH = </usr/local/info:/usr/share/info:/usr/info:>
LANG = <en_US.UTF-8>
LOCALAPPDATA = <C:\Users\rsg\AppData\Local>
LOGONSERVER = <\\RICHARDSGOR69ED>
MANPATH =
    </usr/local/man:/usr/share/man:/usr/man::/usr/ssl
    /man>
NUMBER_OF_PROCESSORS = <1>
```

```
OLDPWD = </home/rsg>
OS = <Windows_NT>
PATH = </usr/local/bin:/usr/bin:/cygdrive/c/Program
Files/Parallels/Parallels Tools/Applications:/cyg
drive/c/Windows/system32:/cygdrive/c/Windows:/cyg
drive/c/Windows/System32/Wbem:/cygdrive/c/Windows
/System32/WindowsPowerShell/v1.0:/cygdrive/c/PROG
RA~1/CONDUS~1/DISKEE~1:/cygdrive/c/Program
Files/Microsoft SQL
Server/110/Tools/Binn:/cygdrive/c/Program
Files/Microsoft SQL
Server/110/DTS/Binn:/cygdrive/c/Program
Files/Microsoft SQL Server/110/Tools/Binn/Managem
entStudio:/cygdrive/c/Borland/Bcc55/Bin:/cygdrive
/c/Python27:/cygdrive/c/Program Files/TortoiseHg:
/cygdrive/c/gnuwin32/bin:/cygdrive/c/Bazaar:/cygd
rive/c/Program Files/Mercurial:/usr/lib/lapack>
PATHEXT =
<.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH
;.MSC>
PRINTER = <HP LaserJet P2015 Series (26A5C8)>
PROCESSOR_ARCHITECTURE = <x86>
PROCESSOR_IDENTIFIER = <x86 Family 6 Model 60 Stepping 3, GenuineIntel>
PROCESSOR_LEVEL = <6>
PROCESSOR_REVISION = <3c03>
PROGRAMFILES = <C:\Program Files>
PS1 = <[\[e]0;\w[a]\n\[e[32m]\u@h
\[e[33m]\w\[e[0m]\n$ >
PSModulePath =
<C:\Windows\system32\WindowsPowerShell\v1.0\Modul
es;c:\Program Files\Microsoft SQL
Server\110\Tools\PowerShell\Modules>
PUBLIC = <C:\Users\Public>
PWD = </cygdrive/d/WR/SoftwareGadgetry-Dev/Python-
2x/tsWxGTUI>
ProgramData = <C:\ProgramData>
SESSIONNAME = <Console>
SHELL = </bin/bash>
SHLVL = <1>
SYSTEMDRIVE = <C:>
SYSTEMROOT = <C:\Windows>
TERM = <xterm>
TZ = <America/New_York>
USER = <rsg>
USERDOMAIN = <RICHARDSGOR69ED>
USERNAME = <rsg>
USERPROFILE = <C:\Users\rsg>
WINDIR = <C:\Windows>
_ = </usr/bin/python>
temp = <C:\Users\rsg\AppData\Local\Temp>
tmp = <C:\Users\rsg\AppData\Local\Temp>
windows_tracing_flags = <3>
windows_tracing_logfile = <C:\BVTBin\Tests\installpackage\csilogfile.log>
```

===== End Platform Run Time Environment =====

6.5.1.2.1.2 test_tsReportUtilities.log

```

2013/12/06 03:50:58,881      INFO: tsApplication.TsApplication:
    "logs": <"[]">
2013/12/06 03:50:58,881      INFO: tsApplication.TsApplication:
    "Command Line": <"['test_tsReportUtilities.py']">
2013/12/06 03:50:58,881      INFO: tsApplication.TsApplication:
    "buildTitle" = <"test_tsReportUtilities.py">.
2013/12/06 03:50:58,881      INFO: tsApplication.TsApplication:
    "buildVersion" = <"2.0.0">.
2013/12/06 03:50:58,881      INFO: tsApplication.TsApplication:
    "buildDate" = <"05/24/2013">.
2013/12/06 03:50:58,881      INFO: tsApplication.TsApplication:
    "buildAuthors" = <"Richard S. Gordon">.
2013/12/06 03:50:58,881      INFO: tsApplication.TsApplication:
    "buildCopyright" = <"Copyright (c) 2007-2013 Richard S. Gordon.
        All rights reserved.">.
2013/12/06 03:50:58,881      INFO: tsApplication.TsApplication:
    "buildLicense" = <"GNU General Public License, Version 3, 29 June
2007">.
2013/12/06 03:50:58,881      INFO: tsApplication.TsApplication:
    "buildCredits" = <"

```

Credits:

```

    tsLibCLI Import & Application Launch Features:
    Copyright (c) 2007-2009 Frederick A. Kier.
        All rights reserved.">.
2013/12/06 03:50:58,881      INFO: tsApplication.TsApplication:
    "buildTitleVersionDate" = <"test_tsReportUtilities.py, v2.0.0 (build
05/24/2013)">.
2013/12/06 03:50:58,881      INFO: tsApplication.TsApplication:
    "buildHeader" = <"

```

test_tsReportUtilities.py, v2.0.0 (build 05/24/2013)

```

Author(s): Richard S. Gordon
Copyright (c) 2007-2013 Richard S. Gordon.
    All rights reserved.
GNU General Public License, Version 3, 29 June 2007

```

Credits:

```

    tsLibCLI Import & Application Launch Features:
    Copyright (c) 2007-2009 Frederick A. Kier.
        All rights reserved.
">.
2013/12/06 03:50:58,881      INFO: tsApplication.TsApplication:
    "purpose" not defined, using <" ">
2013/12/06 03:50:58,881      INFO: tsApplication.TsApplication:
    "runTimeEntryPoint" = <"<function theMainApplication at 0x7fd4179c>">.
2013/12/06 03:50:58,881      INFO: tsApplication.TsApplication: "guiRequired"
not found, using <"False">
2013/12/06 03:50:58,881      INFO: tsApplication.TsApplication:
    "buildTitleVersionDate" = <"test_tsReportUtilities.py, v2.0.0 (build
05/24/2013)">.
2013/12/06 03:50:58,881      INFO: tsApplication.TsApplication:

```

```
"buildHeader" = <"

test_tsReportUtilities.py, v2.0.0 (build 05/24/2013)

Author(s): Richard S. Gordon
Copyright (c) 2007-2013 Richard S. Gordon.
    All rights reserved.
GNU General Public License, Version 3, 29 June 2007

Credits:

    tsLibCLI Import & Application Launch Features:
    Copyright (c) 2007-2009 Frederick A. Kier.
    All rights reserved.
">.
2013/12/06 03:50:58,881      INFO: tsApplication.TsApplication:
    "runTimeEntryPoint" = <"<function theMainApplication at 0x7fd4179c>">.
2013/12/06 03:50:58,881      INFO:

test_tsReportUtilities.py, v2.0.0 (build 05/24/2013)

Author(s): Richard S. Gordon
Copyright (c) 2007-2013 Richard S. Gordon.
    All rights reserved.
GNU General Public License, Version 3, 29 June 2007

Credits:

    tsLibCLI Import & Application Launch Features:
    Copyright (c) 2007-2009 Frederick A. Kier.
    All rights reserved.

2013/12/06 03:50:58,881      INFO:
*** DISPLAY DICTIONARY TEST ***

2013/12/06 03:50:58,885      INFO:
*** GET NEXT PATH NAME TEST ***

2013/12/06 03:50:58,885      INFO:      ./ junk ./junk_0001.txt
2013/12/06 03:50:58,885      INFO:
*** GET STATISTICS TIME TEST ***

2013/12/06 03:50:58,885      INFO:      1386319858.89 1386323458.89
    Started 03:50:58
    Ended   04:50:58
    Elapsed 00-01:00:00 (days-hrs:min:sec)

2013/12/06 03:50:58,885      INFO:
*** GET STATISTICS LIST TEST ***

2013/12/06 03:50:58,885      INFO:      1386319858.89 1386323458.89 100 80 20
Test Summary: FAILED after 00-01:00:00 (days-hrs:min:sec)
    Details: Runs 100, Passes 80, Failures 20
    Started 03:50:58
    Ended   04:50:58
    Elapsed 00-01:00:00 (days-hrs:min:sec)
```

2013/12/06 03:50:58,885 INFO:
 *** GET DAY HOUR MINUTE SECOND STRING TEST ***

2013/12/06 03:50:58,885 INFO: 0 00-00:00:00
 2013/12/06 03:50:58,885 INFO: 1 00-00:00:01
 2013/12/06 03:50:58,885 INFO: 59 00-00:00:59
 2013/12/06 03:50:58,885 INFO: 60 00-00:01:00
 2013/12/06 03:50:58,885 INFO: 120 00-00:02:00
 2013/12/06 03:50:58,885 INFO: 3599 00-00:59:59
 2013/12/06 03:50:58,885 INFO: 3600 00-01:00:00
 2013/12/06 03:50:58,885 INFO: 86400 01-00:00:00
 2013/12/06 03:50:58,885 INFO: 90000 01-01:00:00
 2013/12/06 03:50:58,885 INFO:

*** GET SECONDS TIME FROM HOURS MINUTES SECONDS STRING TEST ***

2013/12/06 03:50:58,885 INFO: None 0
 2013/12/06 03:50:58,885 INFO: 00:00:00 0
 2013/12/06 03:50:58,885 INFO: 01:02:03 3723
 2013/12/06 03:50:58,885 INFO: 12:34:56 45296
 2013/12/06 03:50:58,885 INFO: 23:59:59 86399
 2013/12/06 03:50:58,885 INFO: 24:00:00 86400
 2013/12/06 03:50:58,885 INFO:

*** GET BYTE COUNT STRINGS TEST ***

2013/12/06 03:50:58,885 INFO: 1024^0 ('1 Bytes', '1', '0x1')
 2013/12/06 03:50:58,885 INFO: 1024^1 ('1.00 KBytes', '1024', '0x400')
 2013/12/06 03:50:58,885 INFO: 1024^2 ('1.00 MBytes', '1048576',
 '0x100000')
 2013/12/06 03:50:58,885 INFO: 1024^3 ('1.00 GBytes', '1073741824',
 '0x40000000')
 2013/12/06 03:50:58,885 INFO: 1024^4 ('1.00 TBytes', '1099511627776',
 '0x10000000000')
 2013/12/06 03:50:58,885 INFO: 1024^5 ('1.00 PBytes',
 '1125899906842624', '0x400000000000')
 2013/12/06 03:50:58,885 INFO: 1024^6 ('1.00 EBytes',
 '1152921504606846976', '0x100000000000000')
 2013/12/06 03:50:58,885 INFO: 1024^7 ('1.00 ZBytes',
 '1180591620717411303424', '0x4000000000000000')
 2013/12/06 03:50:58,885 INFO: 1024^8 ('1.00 YBytes',
 '1208925819614629174706176', '0x100000000000000000')
 2013/12/06 03:50:58,885 INFO:

*** DISPLAY DICTIONARY TEST ***

6.5.1.2.1.3 displayDictionaryText.log

```
----- Begin "myDictionary" at level 0 -----

        ----- Begin "contents" at level 1 -----

                name = contents

                ----- Begin "programDictionary" at level 2 -----

                        list2 = ['ab', 'cd', 'ef']
                        main = __main__
                        mainTitleVersionDate = test_tsReportUtilities.py, v2.0.0
(build 05/24/2013)
                        name = programDictionary

                ----- End   "programDictionary" at level 2 -----

                ----- Begin "releaseDictionary" at level 2 -----

                        date = 05/24/2013
                        list1 = [12, 34, 56]
                        name = releaseDictionary
                        title = test_tsReportUtilities.py
                        version = 2.0.0

                ----- End   "releaseDictionary" at level 2 -----

                ----- Begin "sequentialDictionary" at level 2 -----

                        -1 = pear
                        15 = 1.234
                        25 = 1234 (0x4D2)
                        50 = orange
                        100 = apple
                        name = sequentialDictionary

                ----- End   "sequentialDictionary" at level 2 -----

        ----- End   "contents" at level 1 -----

        name = myDictionary

----- End   "myDictionary" at level 0 -----
```

6.5.1.3 Logs Tree (Files for Graphical Applications)

The "tsWxGTUI_PyVx" Toolkit registers, via log files, the startup, configuration and chronology of activities needed for design verification and troubleshooting.

- 1 The root directory is named "logs" and is always placed in the directory from which the application has been launched. The root directory is created, if it does not already exist.

- a) Within the "logs" directory is a date and time stamped directory for each application startup. The startup directory is always created.

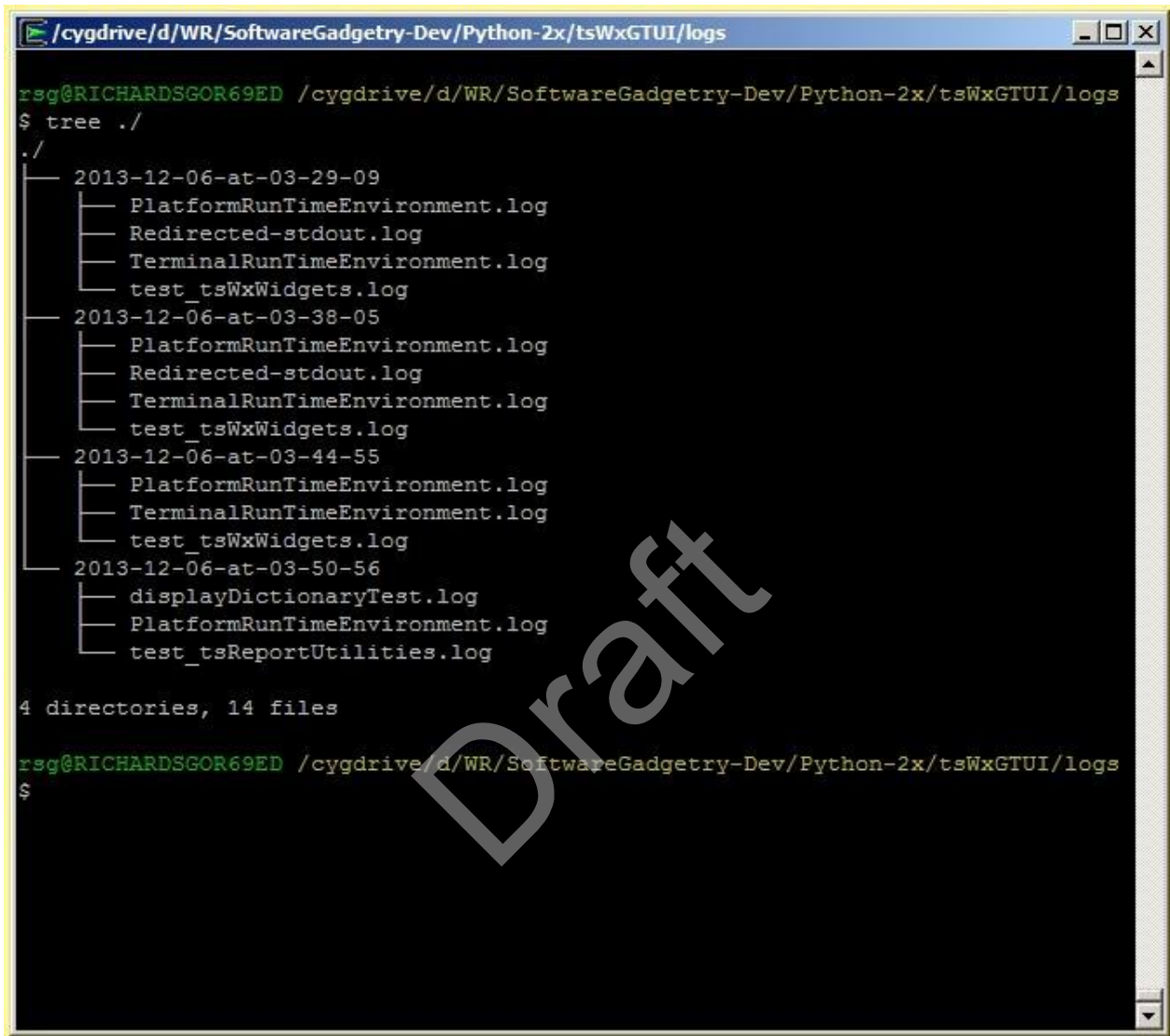
- b) Within the startup directory are the application-specific log files.

"PlatformRunTimeEnvironment.log" - Captures current hardware, software and network information about the run time environment for the user process.

"TerminalRunTimeEnvironment.log" - Captures the current hardware, software and emulated configuration of the "wxPython"-style, "nCurses"-based Graphical-Text User Interface subsystem. The contents and verbosity of "TerminalRunTimeEnvironment.log(s)" are controlled by configuration settings (for details see Customizable CLI Features).

"<application>.log(s)" - Captures date and time stamped messages associated with application designated normal and abnormal situations and activities. The contents and verbosity of "<application>.log(s)" are controlled by configuration settings (for details see Customizable CLI Features).

- 2 It is left to the System Administrator, Software Engineer, System Operator and Field Service personnel to do the housekeeping that removes those log file when they are no longer useful.



```
/cygdrive/d/WR/SoftwareGadgetry-Dev/Python-2x/tsWxGTUI/logs
rsg@RICHARDSGOR69ED /cygdrive/d/WR/SoftwareGadgetry-Dev/Python-2x/tsWxGTUI/logs
$ tree ./
./
├── 2013-12-06-at-03-29-09
│   ├── PlatformRunTimeEnvironment.log
│   ├── Redirected-stdout.log
│   ├── TerminalRunTimeEnvironment.log
│   └── test_tsWxWidgets.log
├── 2013-12-06-at-03-38-05
│   ├── PlatformRunTimeEnvironment.log
│   ├── Redirected-stdout.log
│   ├── TerminalRunTimeEnvironment.log
│   └── test_tsWxWidgets.log
├── 2013-12-06-at-03-44-55
│   ├── PlatformRunTimeEnvironment.log
│   ├── TerminalRunTimeEnvironment.log
│   └── test_tsWxWidgets.log
└── 2013-12-06-at-03-50-56
    ├── displayDictionaryTest.log
    ├── PlatformRunTimeEnvironment.log
    └── test_tsReportUtilities.log

4 directories, 14 files
rsg@RICHARDSGOR69ED /cygdrive/d/WR/SoftwareGadgetry-Dev/Python-2x/tsWxGTUI/logs
$
```

6.5.1.3.1 2013-12-06-at-03-44-55**6.5.1.3.1.1 PlatformRunTimeEnvironment.log**

tsPlatformRunTimeEnvironment.py, v2.1.0 (build 10/19/2013)

Author(s): Richard S. Gordon
 Copyright (c) 2007-2013 Richard S. Gordon.
 All rights reserved.
 GNU General Public License, Version 3, 29 June 2007

Credits:

tsLibCLI Import & Application Launch Features:
 Copyright (c) 2007-2009 Frederick A. Kier.
 All rights reserved.
 GNU General Public License, Version 3, 29 June 2007

terminalsize (<https://gist.github.com/jtriley/1108174>) Features:
 Copyright (c) 2011 Justin T. Riley.
 All rights reserved.
 GNU General Public License, Version 3, 29 June 2007

Python Platform & Logging Module API Features:
 Copyright (c) 2001-2013 Python Software Foundation.
 All rights reserved.
 PSF License Agreement for Python 2.7.3 & 3.3.0

===== Begin Platform Run Time Environment =====

Reported Fri, 06-Dec-2013 at 03:44:57

Network Identification

hostname = <RICHARDSGOR69ED>
 aliaslist = <[]> # NOTE: List of Values NOT available.
 ipaddrlist = <['fe80::5079:ab53:c361:23a4']>

Host Central Processing Unit

machine = <i686>
 processor = <> # NOTE: Value NOT available.
 architecture = <32> bits; <> # NOTE: Value NOT available. linkage
 byteorder = <little>

Host Operating System

api = <posix>
 system = <CYGWIN_NT-6.1>
 release = <1.7.25(0.270/5/3)>

```
version = <2013-08-31 20:39>
```

Host Console Display Size

```
-----  
characters/line = <80>  
lines/display = <35>
```

Python Platform

```
-----  
branch = <> # NOTE: Value NOT available.  
build = <default> number; <Dec 18 2012 13:50:09> date  
compiler = <GCC 4.5.3>  
implementation = <CPython>  
revision = <> # NOTE: Value NOT available.  
version = <2.7.3>
```

Process Parameters

```
-----  
pid = <898588> / <0xDB61C>  
getppid = <891332> / <0xD99C4>  
getegid = <513>  
geteuid = <1000>  
getgid = <513>  
getgroups = <[545, 1001, 513]>  
getpgid = <898588>  
getuid = <1000>  
getlogin = <rsg>  
ctermid = </dev/pty0>  
cwd = </cygdrive/d/WR/SoftwareGadgetry-Dev/Python-2x/tsWxGTUI>
```

Environment Variables

```
-----  
!:: = <::\>  
ALLUSERSPROFILE = <C:\ProgramData>  
APPDATA = <C:\Users\rsg\AppData\Roaming>  
COMMONPROGRAMFILES = <C:\Program Files\Common Files>  
COMPUTERNAME = <RICHARDSGOR69ED>  
COMSPEC = <C:\Windows\system32\cmd.exe>  
EMAIL = <D:\Mail\The Bat!>  
FP_NO_HOST_CHECK = <NO>  
HOME = </home/rsg>  
HOMEDRIVE = <d:>  
HOMEPATH = <\Home\rsg>  
HOSTNAME = <RICHARDSGOR69ED>  
INFOPATH = </usr/local/info:/usr/share/info:/usr/info:>  
LANG = <en_US.UTF-8>  
LOCALAPPDATA = <C:\Users\rsg\AppData\Local>  
LOGONSERVER = <\\RICHARDSGOR69ED>  
MANPATH =  
          </usr/local/man:/usr/share/man:/usr/man::/usr/ssl  
          /man>  
NUMBER_OF_PROCESSORS = <1>
```



```

OLDPWD = </home/rsg>
OS = <Windows_NT>
PATH = </usr/local/bin:/usr/bin:/cygdrive/c/Program
Files/Parallels/Parallels Tools/Applications:/cyg
drive/c/Windows/system32:/cygdrive/c/Windows:/cyg
drive/c/Windows/System32/Wbem:/cygdrive/c/Windows
/System32/WindowsPowerShell/v1.0:/cygdrive/c/PROG
RA~1/CONDUS~1/DISKEE~1:/cygdrive/c/Program
Files/Microsoft SQL
Server/110/Tools/Binn:/cygdrive/c/Program
Files/Microsoft SQL
Server/110/DTS/Binn:/cygdrive/c/Program
Files/Microsoft SQL Server/110/Tools/Binn/Managem
entStudio:/cygdrive/c/Borland/Bcc55/Bin:/cygdrive
/c/Python27:/cygdrive/c/Program Files/TortoiseHg:
/cygdrive/c/gnuwin32/bin:/cygdrive/c/Bazaar:/cygd
rive/c/Program Files/Mercurial:/usr/lib/lapack>
PATHEXT =
<.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH
;.MSC>
PRINTER = <HP LaserJet P2015 Series (26A5C8)>
PROCESSOR_ARCHITECTURE = <x86>
PROCESSOR_IDENTIFIER = <x86 Family 6 Model 60 Stepping 3, GenuineIntel>
PROCESSOR_LEVEL = <6>
PROCESSOR_REVISION = <3c03>
PROGRAMFILES = <C:\Program Files>
PS1 = <[\[e]0;\w[a]\n\[e[32m]\u@h
\[e[33m]\w\[e[0m]\n\$ >
PSModulePath =
<C:\Windows\system32\WindowsPowerShell\v1.0\Modul
es;c:\Program Files\Microsoft SQL
Server\110\Tools\PowerShell\Modules\>
PUBLIC = <C:\Users\Public>
PWD = </cygdrive/d/WR/SoftwareGadgetry-Dev/Python-
2x/tsWxGTUI>
ProgramData = <C:\ProgramData>
SESSIONNAME = <Console>
SHELL = </bin/bash>
SHLVL = <1>
SYSTEMDRIVE = <C:>
SYSTEMROOT = <C:\Windows>
TERM = <xterm>
TZ = <America/New_York>
USER = <rsg>
USERDOMAIN = <RICHARDSGOR69ED>
USERNAME = <rsg>
USERPROFILE = <C:\Users\rsg>
WINDIR = <C:\Windows>
_ = </usr/bin/python>
temp = <C:\Users\rsg\AppData\Local\Temp>
tmp = <C:\Users\rsg\AppData\Local\Temp>
windows_tracing_flags = <3>
windows_tracing_logfile = <C:\BVTBin\Tests\installpackage\csilogfile.log>

```

===== End Platform Run Time Environment =====

6.5.1.3.1.2 test_tsWxWidgets.log

```
2013/12/06 03:44:58,407      INFO: tsApplication.TsApplication:
    "logs": <"['']">
2013/12/06 03:44:58,407      INFO: tsApplication.TsApplication:
    "Command Line": <"['test_tsWxWidgets.py']">
2013/12/06 03:44:58,407      INFO: tsApplication.TsApplication:
    "buildTitle" = <"test_tsWxWidgets">.
2013/12/06 03:44:58,407      INFO: tsApplication.TsApplication:
    "buildVersion" = <"2.3.0">.
2013/12/06 03:44:58,407      INFO: tsApplication.TsApplication:
    "buildDate" = <"06/04/2013">.
2013/12/06 03:44:58,407      INFO: tsApplication.TsApplication:
    "buildAuthors" = <"Richard S. Gordon">.
2013/12/06 03:44:58,407      INFO: tsApplication.TsApplication:
    "buildCopyright" = <"Copyright (c) 2007-2013 Richard S. Gordon.
        All rights reserved.">.
2013/12/06 03:44:58,407      INFO: tsApplication.TsApplication:
    "buildLicense" = <"GNU General Public License, Version 3, 29 June
2007">.
2013/12/06 03:44:58,407      INFO: tsApplication.TsApplication:
    "buildCredits" = <"
```

Credits:

```
    tsLibGUI Import & Application Launch Features:
    Copyright (c) 2007-2009 Frederick A. Kier.
        All rights reserved.
```

```
    Python Curses Module API & Run Time Library Features:
    Copyright (c) 2001-2013 Python Software Foundation.
        All rights reserved.
    PSF License Agreement for Python 2.7.3 & 3.3.0
```

```
    wxWidgets (formerly wxWindows) & wxPython API Features:
    Copyright (c) 1992-2008 Julian Smart, Robert Roebling,
        Vadim Zeitlin and other members of the
        wxWidgets team.
        All rights reserved.
    wxWindows Library License
```

```
    nCurses API & Run Time Library Features:
    Copyright (c) 1998-2011 Free Software Foundation, Inc.
        All rights reserved.
```

```
    GNU General Public License, Version 3, 29 June 2007">.
2013/12/06 03:44:58,407      INFO: tsApplication.TsApplication:
    "buildTitleVersionDate" = <"test_tsWxWidgets, v2.3.0 (build
06/04/2013)">.
2013/12/06 03:44:58,407      INFO: tsApplication.TsApplication:
    "buildHeader" = <"
```

test_tsWxWidgets, v2.3.0 (build 06/04/2013)

```
    Author(s): Richard S. Gordon
    Copyright (c) 2007-2013 Richard S. Gordon.
        All rights reserved.
    GNU General Public License, Version 3, 29 June 2007
```

Credits:

tsLibGUI Import & Application Launch Features:
 Copyright (c) 2007-2009 Frederick A. Kier.
 All rights reserved.

Python Curses Module API & Run Time Library Features:
 Copyright (c) 2001-2013 Python Software Foundation.
 All rights reserved.
 PSF License Agreement for Python 2.7.3 & 3.3.0

wxWidgets (formerly wxWindows) & wxPython API Features:
 Copyright (c) 1992-2008 Julian Smart, Robert Roebling,
 Vadim Zeitlin and other members of the
 wxWidgets team.
 All rights reserved.
 wxWindows Library License

nCurses API & Run Time Library Features:
 Copyright (c) 1998-2011 Free Software Foundation, Inc.
 All rights reserved.
 GNU General Public License, Version 3, 29 June 2007

```
">.
2013/12/06 03:44:58,407      INFO: tsApplication.TsApplication:
      "buildPurpose" = <"
test_tsWxWidgets.py - Test application program. It demonstrates
features and operation of the tsWxGTUI toolkit and associated
building block components of tsLibCLI and tsLibGUI.
">.
2013/12/06 03:44:58,407      INFO: tsApplication.TsApplication:
      "runTimeEntryPoint" = <"<function EntryPoint at 0x7fcf464c>">.
2013/12/06 03:44:58,407      INFO: tsApplication.TsApplication:
      "guiRequired" = <"True">.
2013/12/06 03:44:58,407      INFO: tsApplication.TsApplication:
      "guiTopLevelObjectId" = <"-1">.
2013/12/06 03:44:58,407      INFO: tsApplication.TsApplication:
      "guiMessageRedirect" = <"True">.
2013/12/06 03:44:58,407      INFO: tsApplication.TsApplication:
      "guiMessageFilename" = <"None">.
2013/12/06 03:44:58,407      INFO: tsApplication.TsApplication:
      "guiTopLevelObject" = <"<class '__main___.Communicate'>">.
2013/12/06 03:44:58,407      INFO: tsApplication.TsApplication:
      "guiTopLevelObjectName" = <"frame">.
2013/12/06 03:44:58,407      INFO: tsApplication.TsApplication:
      "guiTopLevelObjectParent" = <"None">.
2013/12/06 03:44:58,407      INFO: tsApplication.TsApplication:
      "guiTopLevelObjectPosition" = <"(-1, -1)">.
2013/12/06 03:44:58,407      INFO: tsApplication.TsApplication:
      "guiTopLevelObjectSize" = <"(-1, -1)">.
2013/12/06 03:44:58,407      INFO: tsApplication.TsApplication:
      "guiTopLevelObjectStatusBar" = <"None">.
2013/12/06 03:44:58,407      INFO: tsApplication.TsApplication:
      "guiTopLevelObjectStyle" = <"570433088">.
2013/12/06 03:44:58,407      INFO: tsApplication.TsApplication:
      "guiTopLevelObjectTitle" = <"Gui_Test_Units">.
```

```
2013/12/06 03:44:58,407      INFO: tsApplication.TsApplication:
    "buildTitleVersionDate" = <"test_tsWxWidgets, v2.3.0 (build
06/04/2013)">.
2013/12/06 03:44:58,407      INFO: tsApplication.TsApplication:
    "buildHeader" = <"

test_tsWxWidgets, v2.3.0 (build 06/04/2013)

Author(s): Richard S. Gordon
Copyright (c) 2007-2013 Richard S. Gordon.
    All rights reserved.
GNU General Public License, Version 3, 29 June 2007

Credits:

    tsLibGUI Import & Application Launch Features:
    Copyright (c) 2007-2009 Frederick A. Kier.
        All rights reserved.

    Python Curses Module API & Run Time Library Features:
    Copyright (c) 2001-2013 Python Software Foundation.
        All rights reserved.
    PSF License Agreement for Python 2.7.3 & 3.3.0

    wxWidgets (formerly wxWindows) & wxPython API Features:
    Copyright (c) 1992-2008 Julian Smart, Robert Roebling,
        Vadim Zeitlin and other members of the
        wxWidgets team.
        All rights reserved.
    wxWindows Library License

    nCurses API & Run Time Library Features:
    Copyright (c) 1998-2011 Free Software Foundation, Inc.
        All rights reserved.
    GNU General Public License, Version 3, 29 June 2007
">.
2013/12/06 03:44:58,407      INFO: tsApplication.TsApplication:
    "runTimeEntryPoint" = <"<function EntryPoint at 0x7fcf464c>">.
```

```

2013/12/06 03:44:58,407      INFO: dir(self.parent)=[ 'Wrapper', '_App',
'_Logs', '_TheAssignedLogger', '_TheAssignedLogger', '__class__',
'__delattr__', '__dict__', '__doc__', '__format__', '__getattr__',
'__hash__', '__init__', '__module__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', '__weakref__', 'applicationStyle', 'args', 'buildAuthors',
'buildCopyright', 'buildCredits', 'buildDate', 'buildHeader', 'buildLicense',
'buildPurpose', 'buildTitle', 'buildTitleVersionDate', 'buildVersion',
'callersExceptionHandler', 'createLog', 'currentTime',
'enableDefaultCommandLineParser', 'getApp', 'getAssignedLogger',
'getLaunchSettings', 'getLog', 'getRunTimeTitle',
'getRunTimeTitleVersionDate', 'guiMessageFilename', 'guiMessageRedirect',
'guiModeLauncher', 'guiRequired', 'guiTopLevelObject', 'guiTopLevelObjectId',
'guiTopLevelObjectName', 'guiTopLevelObjectParent',
'guiTopLevelObjectPosition', 'guiTopLevelObjectSize',
'guiTopLevelObjectStatusBar', 'guiTopLevelObjectStyle',
'guiTopLevelObjectTitle', 'logger', 'logs', 'options', 'registerBuildAuthors',
'registerBuildCopyright', 'registerBuildCredits', 'registerBuildDate',
'registerBuildHeader', 'registerBuildLicense', 'registerBuildPurpose',
'registerBuildTitle', 'registerBuildTitleVersionDate', 'registerBuildVersion',
'registerGuiMessageFilename', 'registerGuiMessageRedirect',
'registerGuiRequired', 'registerGuiTopLevelObject',
'registerGuiTopLevelObjectId', 'registerGuiTopLevelObjectName',
'registerGuiTopLevelObjectParent', 'registerGuiTopLevelObjectPosition',
'registerGuiTopLevelObjectSize', 'registerGuiTopLevelObjectStatusBar',
'registerGuiTopLevelObjectStyle', 'registerGuiTopLevelObjectTitle',
'registerInstantiationSettings', 'registerLogs', 'registerRunTimeEntryPoint',
'runMainApplication', 'runTimeEntryPoint', 'runTimeTitle',
'runTimeTitleVersionDate', 'runTimeTrap', 'shutdownTime', 'startupTime',
'sysArgv', 'tsAppArgs', 'tsAppKw']
2013/12/06 03:44:58,417      DEBUG: Begin GraphicalTextUserInterface
(0x7F6C60AC) for Display.
2013/12/06 03:44:58,417      DEBUG:      Begin Curses Start.
2013/12/06 03:44:58,417      DEBUG:      stdscr = <_curses.curses window object
at 0x7ff921f0>
2013/12/06 03:44:58,417      DEBUG:      stdscrGeometry = (0, 0, 80, 35)
2013/12/06 03:44:58,417      DEBUG:      stdscrGeometryPixels = (0, 0, 640, 420)
2013/12/06 03:44:58,417      DEBUG:      termname = xterm
2013/12/06 03:44:58,417      DEBUG:      longname =
2013/12/06 03:44:58,417      DEBUG:      foreground = black
2013/12/06 03:44:58,417      DEBUG:      background = white
2013/12/06 03:44:58,417      DEBUG:      mmask = 536870911
2013/12/06 03:44:58,417      DEBUG:      has_mouse = True
2013/12/06 03:44:58,417      DEBUG:      MouseButtonCodes = {4096: 'button 3
clicked', 1: 'button 1 released', 2: 'button 1 pressed', 4: 'button 1
clicked', 524288: 'button 4 triple clicked', 8: 'button 1 double clicked',
128: 'button 2 clicked', 134217728: 'button alt', 256: 'button 2 double
clicked', 16: 'button 1 triple clicked', 512: 'button 2 triple clicked',
33554432: 'button ctrl', 67108864: 'button shift', 32: 'button 2 released',
131072: 'button 4 clicked', 262144: 'button 4 double clicked', 1024: 'button 3
released', 8192: 'button 3 double clicked', 16384: 'button 3 triple clicked',
32768: 'button 4 released', 64: 'button 2 pressed', 2048: 'button 3 pressed',
65536: 'button 4 pressed', 'name': 'MouseButtonCodes'}
2013/12/06 03:44:58,417      DEBUG:      has_colors = True
2013/12/06 03:44:58,417      DEBUG:      curses_colors = 8
2013/12/06 03:44:58,417      DEBUG:      curses_color_pairs = 64

```

```
2013/12/06 03:44:58,417      DEBUG:      can_change_color = False
2013/12/06 03:44:58,417      DEBUG:      Begin tsInstallDefaultColorDataBase
2013/12/06 03:44:58,417      DEBUG: installed standard ColorDataBase = {'blue':
4, 'name': 'ColorDataBase', 'yellow': 3, 'green': 2, 'cyan': 6, 'black': 0,
'magenta': 5, 'white': 7, 'red': 1}
2013/12/06 03:44:58,417      DEBUG: installed ColorDataBaseID = {0: 'black', 1:
'red', 2: 'green', 3: 'yellow', 4: 'blue', 'name': 'ColorDataBaseID', 6:
'cyan', 7: 'white', 5: 'magenta'}
2013/12/06 03:44:58,417      DEBUG: installed ColorDataBaseRGB = {'blue': (0,
0, 202), 'name': 'ColorDataBaseRGB', 'yellow': (202, 202, 0), 'green': (0,
202, 0), 'cyan': (0, 202, 202), 'black': (0, 0, 0), 'magenta': (202, 0, 202),
'white': (202, 202, 202), 'red': (202, 0, 0)}
2013/12/06 03:44:58,417      DEBUG: installed ColorSubstitutionDataBase =
{'cadet blue': 'blue', 'sea green': 'green', 'gold': 'yellow', 'firebrick':
'red', 'medium goldenrod': 'yellow', 'violet': 'magenta', 'steel blue':
'blue', 'maroon': 'red', 'sienna': 'red', 'dark slate blue': 'blue', 'khaki':
'yellow', 'medium turquoise': 'cyan', 'sky blue': 'cyan', 'navy': 'blue',
'light blue': 'blue', 'lime green': 'green', 'magenta': 'magenta', 'blue
violet': 'blue', 'orchid': 'magenta', 'blue': 'blue', 'violet red': 'red',
'medium aquamarine': 'cyan', 'medium violet red': 'red', 'medium slate blue':
'blue', 'purple': 'red', 'dark turquoise': 'cyan', 'thistle': 'black', 'light
steel blue': 'blue', 'black': 'black', 'medium spring green': 'green', 'indian
red': 'red', 'aquamarine': 'cyan', 'white': 'white', 'medium sea green':
'green', 'red': 'red', 'brown': 'yellow', 'turquoise': 'cyan', 'dim gray':
'black', 'wheat': 'yellow', 'yellow green': 'yellow', 'medium orchid':
'magenta', 'salmon': 'red', 'dark gray': 'black', 'orange': 'red', 'yellow':
'yellow', 'spring green': 'green', 'dark slate gray': 'black', 'dark olive
green': 'green', 'cyan': 'cyan', 'green yellow': 'green', 'orange red': 'red',
'tan': 'yellow', 'midnight blue': 'blue', 'gray': 'black', 'cornflower blue':
'blue', 'goldenrod': 'cyan', 'pink': 'red', 'name': 'colorSubstitutionMap',
'coral': 'red', 'medium forest green': 'green', 'medium blue': 'blue', 'forest
green': 'green', 'dark orchid': 'magenta', 'slate blue': 'blue', 'pale green':
'green', 'green': 'green', 'light gray': 'black', 'plum': 'magenta', 'dark
green': 'green'}
```

```

2013/12/06 03:44:58,417      DEBUG: installed standard ColorDataBasePairID =
{'ColorNumbersFromPairNumbers': {0: (0, 0), 1: (1, 0), 2: (2, 0), 3: (3, 0),
4: (4, 0), 5: (5, 0), 6: (6, 0), 7: (7, 0), 8: (0, 1), 9: (1, 1), 10: (2, 1),
11: (3, 1), 12: (4, 1), 13: (5, 1), 14: (6, 1), 15: (7, 1), 16: (0, 2), 17:
(1, 2), 18: (2, 2), 19: (3, 2), 20: (4, 2), 21: (5, 2), 22: (6, 2), 23: (7,
2), 24: (0, 3), 25: (1, 3), 26: (2, 3), 27: (3, 3), 28: (4, 3), 29: (5, 3),
30: (6, 3), 31: (7, 3), 32: (0, 4), 33: (1, 4), 34: (2, 4), 35: (3, 4), 36:
(4, 4), 37: (5, 4), 38: (6, 4), 39: (7, 4), 40: (0, 5), 41: (1, 5), 42: (2,
5), 43: (3, 5), 44: (4, 5), 45: (5, 5), 46: (6, 5), 47: (7, 5), 48: (0, 6),
49: (1, 6), 50: (2, 6), 51: (3, 6), 52: (4, 6), 53: (5, 6), 54: (6, 6), 55:
(7, 6), 56: (0, 7), 57: (1, 7), 58: (2, 7), 59: (3, 7), 60: (4, 7), 61: (5,
7), 62: (6, 7), 63: (7, 7), 'name': 'ColorNumbersFromPairNumbers'},
'PairNumbersFromColorNumbers': {(7, 3): 31, (4, 7): 60, (1, 3): 25, (6, 6):
54, (3, 0): 3, (5, 4): 37, (2, 1): 10, (4, 6): 52, (5, 6): 53, (6, 2): 22, (1,
6): 49, (3, 7): 59, (5, 1): 13, (0, 3): 24, (2, 5): 42, (7, 2): 23, (4, 0): 4,
(1, 2): 17, (6, 7): 62, (3, 3): 27, (0, 6): 48, (7, 6): 55, (4, 4): 36, (6,
3): 30, (1, 5): 41, (5, 0): 5, (2, 2): 18, (5, 7): 61, (3, 5): 43, (4, 1): 12,
(1, 1): 9, (6, 4): 38, (3, 2): 19, (0, 0): 0, (3, 6): 51, (2, 7): 58, (7, 1):
15, (4, 5): 44, (0, 4): 32, (6, 0): 6, (1, 4): 33, (7, 7): 63, (5, 5): 45, (7,
5): 47, (2, 3): 26, (0, 7): 56, (4, 2): 20, (1, 0): 1, (6, 5): 46, (5, 3): 29,
(0, 1): 8, (7, 0): 7, 'name': 'PairNumbersFromColorNumbers', (3, 4): 35, (6,
1): 14, (3, 1): 11, (2, 6): 50, (2, 4): 34, (7, 4): 39, (2, 0): 2, (4, 3): 28,
(1, 7): 57, (0, 5): 40, (5, 2): 21, (0, 2): 16}, 'name':
'ColorDataBasePairID'}
2013/12/06 03:44:58,417      DEBUG:      End tsInstallDefaultColorDataBase
2013/12/06 03:44:58,427      DEBUG:      End Curses Start.
2013/12/06 03:44:58,427      NOTICE: start: wxThemeToUse:
currentDirectory=/cygdrive/d/WR/SoftwareGadgetry-Dev/Python-2x/tsWxGTUI;
theSplashScreenPath=./tsLibGUI/tsWxPkg/src/;
theSplashScreenFileName=theSplashScreen.img
2013/12/06 03:44:58,427      NOTICE: start: platformSuffix=_80_x_35_cygwin_nt-
6.1.img
2013/12/06 03:44:58,427      NOTICE: start:
bitmapImageFileName=./tsLibGUI/tsWxPkg/src/theSplashScreen_80_x_35_cygwin_nt-
6.1.img
2013/12/06 03:44:58,427      DEBUG: type(bitmapID)=<type '_curses.curses
window'>
2013/12/06 03:45:38,257      WARNING: Received SIGINT
2013/12/06 03:45:38,257      DEBUG:      Begin Curses Stop.
2013/12/06 03:45:38,347      ERROR: Splash Screen Other Error: 'User Interface
Exception: Command Line Operation Not Valid; Command: "stty sane". [ExitCode
#133]'
2013/12/06 03:45:38,347      DEBUG: End GraphicalTextUserInterface (0x7F6C60AC)
for Display.
2013/12/06 03:45:38,347      DEBUG: Begin GraphicalTextUserInterface
(0x7F6B1ACC) for PyApp.
2013/12/06 03:45:38,347      DEBUG:      Begin Curses Start.
2013/12/06 03:45:38,347      DEBUG:      stdscr = <_curses.curses window object
at 0x7ff92290>
2013/12/06 03:45:38,347      DEBUG:      stdscrGeometry = (0, 0, 80, 35)
2013/12/06 03:45:38,347      DEBUG:      stdscrGeometryPixels = (0, 0, 640, 420)
2013/12/06 03:45:38,347      DEBUG:      termname = xterm
2013/12/06 03:45:38,347      DEBUG:      longname =
2013/12/06 03:45:38,347      DEBUG:      foreground = black
2013/12/06 03:45:38,347      DEBUG:      background = white
2013/12/06 03:45:38,347      DEBUG:      mmask = 536870911

```

```
2013/12/06 03:45:38,347      DEBUG:      has_mouse = True
2013/12/06 03:45:38,347      DEBUG:      MouseButtonCodes = {4096: 'button 3
clicked', 1: 'button 1 released', 2: 'button 1 pressed', 4: 'button 1
clicked', 524288: 'button 4 triple clicked', 8: 'button 1 double clicked',
128: 'button 2 clicked', 134217728: 'button alt', 256: 'button 2 double
clicked', 16: 'button 1 triple clicked', 512: 'button 2 triple clicked',
33554432: 'button ctrl', 67108864: 'button shift', 32: 'button 2 released',
131072: 'button 4 clicked', 262144: 'button 4 double clicked', 1024: 'button 3
released', 8192: 'button 3 double clicked', 16384: 'button 3 triple clicked',
32768: 'button 4 released', 64: 'button 2 pressed', 2048: 'button 3 pressed',
65536: 'button 4 pressed', 'name': 'MouseButtonCodes'}
2013/12/06 03:45:38,347      DEBUG:      has_colors = True
2013/12/06 03:45:38,347      DEBUG:      curses_colors = 8
2013/12/06 03:45:38,347      DEBUG:      curses_color_pairs = 64
2013/12/06 03:45:38,347      DEBUG:      can_change_color = False
2013/12/06 03:45:38,347      DEBUG:      Begin tsInstallDefaultColorDataBase
2013/12/06 03:45:38,347      DEBUG: installed standard ColorDataBase = {'blue':
4, 'name': 'ColorDataBase', 'yellow': 3, 'green': 2, 'cyan': 6, 'black': 0,
'magenta': 5, 'white': 7, 'red': 1}
2013/12/06 03:45:38,347      DEBUG: installed ColorDataBaseID = {0: 'black', 1:
'red', 2: 'green', 3: 'yellow', 4: 'blue', 'name': 'ColorDataBaseID', 6:
'cyan', 7: 'white', 5: 'magenta'}
2013/12/06 03:45:38,347      DEBUG: installed ColorDataBaseRGB = {'blue': (0,
0, 202), 'name': 'ColorDataBaseRGB', 'yellow': (202, 202, 0), 'green': (0,
202, 0), 'cyan': (0, 202, 202), 'black': (0, 0, 0), 'magenta': (202, 0, 202),
'white': (202, 202, 202), 'red': (202, 0, 0)}
2013/12/06 03:45:38,347      DEBUG: installed ColorSubstitutionDataBase =
{'cadet blue': 'blue', 'sea green': 'green', 'gold': 'yellow', 'firebrick':
'red', 'medium goldenrod': 'yellow', 'violet': 'magenta', 'steel blue':
'blue', 'maroon': 'red', 'sienna': 'red', 'dark slate blue': 'blue', 'khaki':
'yellow', 'medium turquoise': 'cyan', 'sky blue': 'cyan', 'navy': 'blue',
'light blue': 'blue', 'lime green': 'green', 'magenta': 'magenta', 'blue
violet': 'blue', 'orchid': 'magenta', 'blue': 'blue', 'violet red': 'red',
'medium aquamarine': 'cyan', 'medium violet red': 'red', 'medium slate blue':
'blue', 'purple': 'red', 'dark turquoise': 'cyan', 'thistle': 'black', 'light
steel blue': 'blue', 'black': 'black', 'medium spring green': 'green', 'indian
red': 'red', 'aquamarine': 'cyan', 'white': 'white', 'medium sea green':
'green', 'red': 'red', 'brown': 'yellow', 'turquoise': 'cyan', 'dim gray':
'black', 'wheat': 'yellow', 'yellow green': 'yellow', 'medium orchid':
'magenta', 'salmon': 'red', 'dark gray': 'black', 'orange': 'red', 'yellow':
'yellow', 'spring green': 'green', 'dark slate gray': 'black', 'dark olive
green': 'green', 'cyan': 'cyan', 'green yellow': 'green', 'orange red': 'red',
'tan': 'yellow', 'midnight blue': 'blue', 'gray': 'black', 'cornflower blue':
'blue', 'goldenrod': 'cyan', 'pink': 'red', 'name': 'colorSubstitutionMap',
'coral': 'red', 'medium forest green': 'green', 'medium blue': 'blue', 'forest
green': 'green', 'dark orchid': 'magenta', 'slate blue': 'blue', 'pale green':
'green', 'green': 'green', 'light gray': 'black', 'plum': 'magenta', 'dark
green': 'green'}
```



```

2013/12/06 03:45:38,347      DEBUG: installed standard ColorDataBasePairID =
{'ColorNumbersFromPairNumbers': {0: (0, 0), 1: (1, 0), 2: (2, 0), 3: (3, 0),
4: (4, 0), 5: (5, 0), 6: (6, 0), 7: (7, 0), 8: (0, 1), 9: (1, 1), 10: (2, 1),
11: (3, 1), 12: (4, 1), 13: (5, 1), 14: (6, 1), 15: (7, 1), 16: (0, 2), 17:
(1, 2), 18: (2, 2), 19: (3, 2), 20: (4, 2), 21: (5, 2), 22: (6, 2), 23: (7,
2), 24: (0, 3), 25: (1, 3), 26: (2, 3), 27: (3, 3), 28: (4, 3), 29: (5, 3),
30: (6, 3), 31: (7, 3), 32: (0, 4), 33: (1, 4), 34: (2, 4), 35: (3, 4), 36:
(4, 4), 37: (5, 4), 38: (6, 4), 39: (7, 4), 40: (0, 5), 41: (1, 5), 42: (2,
5), 43: (3, 5), 44: (4, 5), 45: (5, 5), 46: (6, 5), 47: (7, 5), 48: (0, 6),
49: (1, 6), 50: (2, 6), 51: (3, 6), 52: (4, 6), 53: (5, 6), 54: (6, 6), 55:
(7, 6), 56: (0, 7), 57: (1, 7), 58: (2, 7), 59: (3, 7), 60: (4, 7), 61: (5,
7), 62: (6, 7), 63: (7, 7), 'name': 'ColorNumbersFromPairNumbers'},
'PairNumbersFromColorNumbers': {(7, 3): 31, (4, 7): 60, (1, 3): 25, (6, 6):
54, (3, 0): 3, (5, 4): 37, (2, 1): 10, (4, 6): 52, (5, 6): 53, (6, 2): 22, (1,
6): 49, (3, 7): 59, (5, 1): 13, (0, 3): 24, (2, 5): 42, (7, 2): 23, (4, 0): 4,
(1, 2): 17, (6, 7): 62, (3, 3): 27, (0, 6): 48, (7, 6): 55, (4, 4): 36, (6,
3): 30, (1, 5): 41, (5, 0): 5, (2, 2): 18, (5, 7): 61, (3, 5): 43, (4, 1): 12,
(1, 1): 9, (6, 4): 38, (3, 2): 19, (0, 0): 0, (3, 6): 51, (2, 7): 58, (7, 1):
15, (4, 5): 44, (0, 4): 32, (6, 0): 6, (1, 4): 33, (7, 7): 63, (5, 5): 45, (7,
5): 47, (2, 3): 26, (0, 7): 56, (4, 2): 20, (1, 0): 1, (6, 5): 46, (5, 3): 29,
(0, 1): 8, (7, 0): 7, 'name': 'PairNumbersFromColorNumbers', (3, 4): 35, (6,
1): 14, (3, 1): 11, (2, 6): 50, (2, 4): 34, (7, 4): 39, (2, 0): 2, (4, 3): 28,
(1, 7): 57, (0, 5): 40, (5, 2): 21, (0, 2): 16}, 'name':
'ColorDataBasePairID'}
2013/12/06 03:45:38,347      DEBUG:      End tsInstallDefaultColorDataBase
2013/12/06 03:45:38,357      DEBUG:      End Curses Start.
2013/12/06 03:45:38,357      NOTICE: start: wxThemeToUse:
currentDirectory=/cygdrive/d/WR/SoftwareGadgetry-Dev/Python-2x/tsWxGTUI;
theSplashScreenPath=./tsLibGUI/tsWxPkg/src/;
theSplashScreenFileName=theSplashScreen.img
2013/12/06 03:45:38,357      NOTICE: start: platformSuffix=_80_x_35_cygwin_nt-
6.1.img
2013/12/06 03:45:38,357      NOTICE: start:
bitmapImageFileName=./tsLibGUI/tsWxPkg/src/theSplashScreen_80_x_35_cygwin_nt-
6.1.img
2013/12/06 03:45:38,357      DEBUG: type(bitmapID)=<type '_curses.curses
window'>
2013/12/06 03:45:39,617      WARNING: Received SIGINT
2013/12/06 03:45:39,617      DEBUG:      Begin Curses Stop.
2013/12/06 03:45:39,617      ERROR: Splash Screen Other Error: 'User Interface
Exception: Command Line Operation Not Valid; Command: "stty sane". [ExitCode
#133]'
2013/12/06 03:45:39,617      DEBUG: End GraphicalTextUserInterface (0x7F6B1ACC)
for PyApp.

```

6.5.1.3.1.3 TerminalRunTimeEnvironment.log

tsWxGraphicalTextUserInterface, v2.8.0 (build 12/02/2013)

Author(s): Richard S. Gordon
Copyright (c) 2007-2013 Richard S. Gordon.
All rights reserved.
GNU General Public License, Version 3, 29 June 2007

Credits:

tsLibGUI Import & Application Launch Features:
Copyright (c) 2007-2009 Frederick A. Kier.
All rights reserved.

Python Curses Module API & Run Time Library Features:
Copyright (c) 2001-2013 Python Software Foundation.
All rights reserved.
PSF License Agreement for Python 2.7.3 & 3.3.0

wxWidgets (formerly wxWindows) & wxPython API Features:
Copyright (c) 1992-2008 Julian Smart, Robert Roebling,
Vadim Zeitlin and other members of the
wxWidgets team.
All rights reserved.
wxWindows Library License

nCurses character-mode Terminal Control Library
for Unix-like systems and API Features:
Copyright (c) 1998-2004, 2006 Free Software
Foundation, Inc.
All rights reserved.
nCurses README,v 1.23 2006/04/22

2013/12/06 03:45:38,347 - Started logging to file "./logs/2013-12-06-at-03-44-55/TerminalRunTimeEnvironment.log"

----- Begin "CursesDataBase" at level 0 -----

 BackgroundColor = white

----- Begin "BuiltinPaletteRGB" at level 1 -----

 black = (0, 0, 0)
 blue = (0, 0, 202)
 cyan = (0, 202, 202)
 green = (0, 202, 0)
 magenta = (202, 0, 202)
 name = BuiltinPaletteRGB
 red = (202, 0, 0)
 white = (202, 202, 202)
 yellow = (202, 202, 0)

----- End "BuiltinPaletteRGB" at level 1 -----

```
CanChangeColor = 0 (0x0)
```

```
----- Begin "ColorDataBase" at level 1 -----
```

```
    black = 0 (0x0)
    blue = 4 (0x4)
    cyan = 6 (0x6)
    green = 2 (0x2)
    magenta = 5 (0x5)
    name = ColorDataBase
    red = 1 (0x1)
    white = 7 (0x7)
    yellow = 3 (0x3)
```

```
----- End   "ColorDataBase" at level 1 -----
```

```
----- Begin "ColorDataBaseID" at level 1 -----
```

```
    0 = black
    1 = red
    2 = green
    3 = yellow
    4 = blue
    5 = magenta
    6 = cyan
    7 = white
    name = ColorDataBaseID
```

```
----- End   "ColorDataBaseID" at level 1 -----
```

```
----- Begin "ColorDataBasePairID" at level 1 -----
```

```
----- Begin "ColorNumbersFromPairNumbers" at level 2 -----
```

```
    0 = (0, 0)
    1 = (1, 0)
    2 = (2, 0)
    3 = (3, 0)
    4 = (4, 0)
    5 = (5, 0)
    6 = (6, 0)
    7 = (7, 0)
    8 = (0, 1)
    9 = (1, 1)
    10 = (2, 1)
    11 = (3, 1)
    12 = (4, 1)
    13 = (5, 1)
    14 = (6, 1)
    15 = (7, 1)
    16 = (0, 2)
    17 = (1, 2)
```

```
18 = (2, 2)
19 = (3, 2)
20 = (4, 2)
21 = (5, 2)
22 = (6, 2)
23 = (7, 2)
24 = (0, 3)
25 = (1, 3)
26 = (2, 3)
27 = (3, 3)
28 = (4, 3)
29 = (5, 3)
30 = (6, 3)
31 = (7, 3)
32 = (0, 4)
33 = (1, 4)
34 = (2, 4)
35 = (3, 4)
36 = (4, 4)
37 = (5, 4)
38 = (6, 4)
39 = (7, 4)
40 = (0, 5)
41 = (1, 5)
42 = (2, 5)
43 = (3, 5)
44 = (4, 5)
45 = (5, 5)
46 = (6, 5)
47 = (7, 5)
48 = (0, 6)
49 = (1, 6)
50 = (2, 6)
51 = (3, 6)
52 = (4, 6)
53 = (5, 6)
54 = (6, 6)
55 = (7, 6)
56 = (0, 7)
57 = (1, 7)
58 = (2, 7)
59 = (3, 7)
60 = (4, 7)
61 = (5, 7)
62 = (6, 7)
63 = (7, 7)
name = ColorNumbersFromPairNumbers

----- End    "ColorNumbersFromPairNumbers" at level 2 -----

----- Begin "PairNumbersFromColorNumbers" at level 2 -----

(0, 0) = 0 (0x0)
(0, 1) = 8 (0x8)
(0, 2) = 16 (0x10)
```

(0, 3) = 24 (0x18)
(0, 4) = 32 (0x20)
(0, 5) = 40 (0x28)
(0, 6) = 48 (0x30)
(0, 7) = 56 (0x38)
(1, 0) = 1 (0x1)
(1, 1) = 9 (0x9)
(1, 2) = 17 (0x11)
(1, 3) = 25 (0x19)
(1, 4) = 33 (0x21)
(1, 5) = 41 (0x29)
(1, 6) = 49 (0x31)
(1, 7) = 57 (0x39)
(2, 0) = 2 (0x2)
(2, 1) = 10 (0xA)
(2, 2) = 18 (0x12)
(2, 3) = 26 (0x1A)
(2, 4) = 34 (0x22)
(2, 5) = 42 (0x2A)
(2, 6) = 50 (0x32)
(2, 7) = 58 (0x3A)
(3, 0) = 3 (0x3)
(3, 1) = 11 (0xB)
(3, 2) = 19 (0x13)
(3, 3) = 27 (0x1B)
(3, 4) = 35 (0x23)
(3, 5) = 43 (0x2B)
(3, 6) = 51 (0x33)
(3, 7) = 59 (0x3B)
(4, 0) = 4 (0x4)
(4, 1) = 12 (0xC)
(4, 2) = 20 (0x14)
(4, 3) = 28 (0x1C)
(4, 4) = 36 (0x24)
(4, 5) = 44 (0x2C)
(4, 6) = 52 (0x34)
(4, 7) = 60 (0x3C)
(5, 0) = 5 (0x5)
(5, 1) = 13 (0xD)
(5, 2) = 21 (0x15)
(5, 3) = 29 (0x1D)
(5, 4) = 37 (0x25)
(5, 5) = 45 (0x2D)
(5, 6) = 53 (0x35)
(5, 7) = 61 (0x3D)
(6, 0) = 6 (0x6)
(6, 1) = 14 (0xE)
(6, 2) = 22 (0x16)
(6, 3) = 30 (0x1E)
(6, 4) = 38 (0x26)
(6, 5) = 46 (0x2E)
(6, 6) = 54 (0x36)
(6, 7) = 62 (0x3E)
(7, 0) = 7 (0x7)
(7, 1) = 15 (0xF)
(7, 2) = 23 (0x17)

```
(7, 3) = 31 (0x1F)
(7, 4) = 39 (0x27)
(7, 5) = 47 (0x2F)
(7, 6) = 55 (0x37)
(7, 7) = 63 (0x3F)
    name = PairNumbersFromColorNumbers

----- End    "PairNumbersFromColorNumbers" at level 2 -----

        name = ColorDataBasePairID

----- End    "ColorDataBasePairID" at level 1 -----

----- Begin "ColorDataBaseRGB" at level 1 -----

    black = (0, 0, 0)
    blue = (0, 0, 202)
    cyan = (0, 202, 202)
    green = (0, 202, 0)
    magenta = (202, 0, 202)
    name = ColorDataBaseRGB
    red = (202, 0, 0)
    white = (202, 202, 202)
    yellow = (202, 202, 0)

----- End    "ColorDataBaseRGB" at level 1 -----

----- Begin "colorSubstitutionMap" at level 1 -----

    aquamarine = cyan
    black = black
    blue = blue
    blue violet = blue
    brown = yellow
    cadet blue = blue
    coral = red
    cornflower blue = blue
    cyan = cyan
    dark gray = black
    dark green = green
    dark olive green = green
    dark orchid = magenta
    dark slate blue = blue
    dark slate gray = black
    dark turquoise = cyan
    dim gray = black
    firebrick = red
    forest green = green
    gold = yellow
    goldenrod = cyan
    gray = black
    green = green
    green yellow = green
    indian red = red
```

```

        khaki = yellow
        light blue = blue
        light gray = black
    light steel blue = blue
        lime green = green
        magenta = magenta
        maroon = red
    medium aquamarine = cyan
        medium blue = blue
    medium forest green = green
        medium goldenrod = yellow
        medium orchid = magenta
        medium sea green = green
        medium slate blue = blue
    medium spring green = green
        medium turquoise = cyan
    medium violet red = red
        midnight blue = blue
            name = colorSubstitutionMap
            navy = blue
            orange = red
        orange red = red
            orchid = magenta
    pale green = green
        pink = red
        plum = magenta
        purple = red
        red = red
        salmon = red
        sea green = green
        sienna = red
        sky blue = cyan
        slate blue = blue
    spring green = green
        steel blue = blue
        tan = yellow
        thistle = black
        turquoise = cyan
        violet = magenta
    violet red = red
        wheat = yellow
        white = white
        yellow = yellow
    yellow green = yellow

```

```
----- End    "colorSubstitutionMap" at level 1 -----
```

```

    CursesColorPairs = 64 (0x40)
    CursesColors = 8 (0x8)

```

```
----- Begin "CursesPanels" at level 1 -----
```

```
    name = CursesPanels
```

```
----- End    "CursesPanels" at level 1 -----
```

```
ForegroundColor = black
  HasColors = 1 (0x1)
  HasDisplay = 1 (0x1)
  HasKeyboard = 1 (0x1)
  HasLogger = 1 (0x1)
  HasMouse = 1 (0x1)
  HostOS = CYGWIN_NT-6.1
  LongName =
  Mmask = 536870911 (0x1FFFFFFF)
```

----- Begin "MouseButtonCodes" at level 1 -----

```
    1 = button 1 released
    2 = button 1 pressed
    4 = button 1 clicked
    8 = button 1 double clicked
   16 = button 1 triple clicked
   32 = button 2 released
   64 = button 2 pressed
  128 = button 2 clicked
  256 = button 2 double clicked
  512 = button 2 triple clicked
 1024 = button 3 released
 2048 = button 3 pressed
 4096 = button 3 clicked
 8192 = button 3 double clicked
16384 = button 3 triple clicked
32768 = button 4 released
65536 = button 4 pressed
131072 = button 4 clicked
262144 = button 4 double clicked
524288 = button 4 triple clicked
33554432 = button ctrl
67108864 = button shift
134217728 = button alt
    name = MouseButtonCodes
```

----- End "MouseButtonCodes" at level 1 -----

```
    PythonVersion = Python-2.7.3
      Stdscr = <_curses.curses window object at 0x7ff92290>
      StdscrGeometry = (0, 0, 80, 35)
      StdscrGeometryPixels = (0, 0, 640, 420)
      TermName = xterm
      name = CursesDataBase
```

----- End "CursesDataBase" at level 0 -----

----- Begin "WindowDataBase" at level 0 -----

----- Begin "AcceleratorKeysByEarliestAssignedId" at level 1 -----

```
    name = AcceleratorKeysByEarliestAssignedId
```



```

----- End    "AcceleratorKeysByEarliestAssignedId" at level 1 -----

----- Begin "AcceleratorTableByAssignedId" at level 1 -----
    name = AcceleratorTableByAssignedId
----- End    "AcceleratorTableByAssignedId" at level 1 -----

----- Begin "EventAssociationsByEarliestAssignedId" at level 1 -----
    name = EventAssociationsByEarliestAssignedId
----- End    "EventAssociationsByEarliestAssignedId" at level 1 -----

----- Begin "KeyboardInputRecipients" at level 1 -----
    lifoList = []
        name = KeyboardInputRecipients
----- End    "KeyboardInputRecipients" at level 1 -----

----- Begin "TheWindows" at level 1 -----
        name = TheWindows
        windowIndex = -1 (0x-1)
----- End    "TheWindows" at level 1 -----

----- Begin "TopLevelWindows" at level 1 -----
    name = TopLevelWindows
----- End    "TopLevelWindows" at level 1 -----

----- Begin "WindowHandles" at level 1 -----
    name = WindowHandles
----- End    "WindowHandles" at level 1 -----

----- Begin "WindowTopLevelAncestors" at level 1 -----
    name = WindowTopLevelAncestors
----- End    "WindowTopLevelAncestors" at level 1 -----
        WindowTopLevelTasks = []
----- Begin "WindowsByAssignedId" at level 1 -----

```

```
    name = WindowsByAssignedId
----- End    "WindowsByAssignedId" at level 1 -----

----- Begin "WindowsByHandle" at level 1 -----
    name = WindowsByHandle
----- End    "WindowsByHandle" at level 1 -----

----- Begin "WindowsById" at level 1 -----
    name = WindowsById
----- End    "WindowsById" at level 1 -----

----- Begin "WindowsByName" at level 1 -----
    name = WindowsByName
----- End    "WindowsByName" at level 1 -----

----- Begin "WindowsByPanelLayer" at level 1 -----
    name = WindowsByPanelLayer
----- End    "WindowsByPanelLayer" at level 1 -----

----- Begin "WindowsByShowOrder" at level 1 -----

        ----- Begin "AssignedIdByPanelLayer" at level 2 -----
            name = AssignedIdByPanelLayer
        ----- End    "AssignedIdByPanelLayer" at level 2 -----

            OrderOfShow = []
            OrderOfShowPanelStack = []
            PanelLayer = []

        ----- Begin "PanelStack" at level 2 -----
            name = PanelStack
        ----- End    "PanelStack" at level 2 -----

            name = WindowsByShowOrder
----- End    "WindowsByShowOrder" at level 1 -----
```

```
name = WindowDataBase
```

```
----- End    "WindowDataBase" at level 0 -----
```

```
2013/12/06 03:45:38,347 - Ended logging to file "./logs/2013-12-06-at-03-44-55/TerminalRunTimeEnvironment.log"
```

Draft