

Introduction

Vol. 2 - "tsWxGTUI" Toolkit

Rev. 0.0.0 (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) 2010-2015 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/Documents*.

Preface

Thank you for your interest in the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit.

Begun in July 2007, the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit development project is still a work in progress:

- 1 This set of documents detail the project planning, requirements analysis, architectural concept, design, test, qualification and release of a free and open source library of software:
 - a) **Building Blocks** - Basic units from which something of greater complexity is built up. For example, an application or operating system program may be constructed from one or more data structure definitions, functions, methods, classes, subroutines, threads, processes or building block libraries.
 - b) **Programming or Software Development Tools** - Computer programs that software developers use to create, debug, maintain, or otherwise support other programs and applications.
 - c) **System Administration Utilities** - Computer programs that computer system administrators use to install, debug, maintain, or otherwise support computer hardware and software.
- 2 As a convenience to those of you interested in developing, customizing, documenting, enhancing, maintaining, supporting, troubleshooting and using the capabilities of the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit, it describes the author's plans, experiments and evolutionary application of computer technology and software engineering principles in a manner suitable for individuals with a broad range of computer programming skills:
 - a) **New TeamSTARS "tsWxGTUI_PyVx" Toolkit Users** --- Refer to Vol. 0 - Announcement and Vol. 1 - Brochure.
 - b) **Student** --- For useful background information on computer hardware and software technology refer to Vol. 2 - Introduction.

For useful Python programming and software engineering techniques refer to source files of tsLibCLI, tsToolsCLI, tsUtilities and tsDemoArchive.
 - c) **Intermediate** --- For useful "Curses" / "nCurses" and "wxPython" programming techniques refer to tsLibGUI source files.
 - d) **Advanced** --- For useful project planning and engineering information refer to Vol. 3 - Usage Terms & Conditions, Vol. 4 - Development Plan, Vol. 5 - System Specification, Vol. 6 - Interface Requirements, Vol. 7 - Functional Requirements and Vol. 8 - Release Notes.
 - e) **Expert** --- For useful troubleshooting information refer to Vol. 9 - Software User's Manual.
- 3 The *TeamSTARS* "tsWxGTUI_PyVx" Toolkit software is being engineered to facilitate your development and use of application programs that:
 - a) Feature either:

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.

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).

- b) Will initially be installed in computer systems to be used for development of software and documentation.

Such general-purpose desktop, laptop and workstation computer systems typically have upgradable or at least sufficient processing, memory, communication, input/output and file storage resources.

They typically have computer terminal interface hardware suitable for a pixel-mode display that also supports character-mode.

- c) Will ultimately be installed in embedded systems to be used to monitor and control commercial, industrial, manufacturing, medical or military equipment.

Such application-specific computer systems typically have upgradable but limited processing, memory, communication, input/output and file storage resources. They typically have computer terminal interface hardware suitable only for a character-mode display, keyboard and pointing device.

What you can look forward to doing with the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit?

- 1 Once you've logged into your local computer:

One or more shells for the local operating system's command line interface, terminal interface and the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit's Python and "Curses" based character-mode user interfaces enable you to monitor and control one or more local application programs.

- 2 Then, if and when you login to a remote computer:

One or more shells for the local operating system's command line interface, terminal interface and the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit's Python and "Curses" based character-mode user interfaces communicate with their counterparts on the remote computer with greater speed and efficiency than possible with the larger communication traffic associated with pixel-mode.

This enables you to monitor and control one or more remote application programs from the convenience of your local computer.

What makes the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit possible?

- 1 The *TeamSTARS* "tsWxGUI" Toolkit reflects the author's multi-disciplinary background and extensive professional experience.

- a) **System Engineering**

Single, multiprocessor and distributed computer systems for commercial, industrial, manufacturing and military applications.

- b) **Software Engineering**

Modular, object oriented, portable, realtime, re-usable and scalable designs for application, automation, communication, controls, diagnostic, instrumentation, modeling, operating system, simulation and user interface software.

Software for computer processors, operating systems and peripheral equipment made by Digital Equipment Corporation, Fisher Provox, General Electric, Hewlett-Packard, IBM, Intel, Mercury Computer Systems, Motorola and Sun Microsystems.

"TeamSTARS" was the nickname for Frederick "Rick" A. Kier and Richard "Dick" S. Gordon when they developed diagnostic software at Mercury Computer Systems. "STARS" was the name for their Scalable Test and Resource Stressing diagnostic that originally ran on individual Intel 860 processors and later grew to become a System Test and Resource Stressing diagnostic that concurrently ran a pseudo-random test suite on mixed-processor systems ranging from tens to many hundreds of Intel 860, IBM PowerPC and Analog Device SHARC processors with host computers including Intel x86 running Microsoft Windows, Motorola 68020 processors running VxWorks, Silicon Graphics Inc. MIPS processors running IRIX and Sun Microsystems SPARC processors running SunOS/Solaris. The "STARS" diagnostic was implemented in the "c" programming language and used the "curses" terminal interface library and terminal emulator software, available on the host computer, for its graphical-style user interface.

c) **Programming Languages**

Ada, Algol, Assembler, Basic, C, FORTRAN, Pascal, PL/M, and Python

- 2 If this engineering effort has produced anything useful, it is because it incorporates free and open source technology and documentation that others made readily available on the internet. Their effort, community spirit and altruism are greatly appreciated.

The engineering documentation encompasses much of the traditional software development process and life cycle. It has been organized into an extended set of book volumes with the book titles and table of contents in the style of MIL-STD-498. Many, but not all topics have been addressed. It incorporates excerpts of material already available for fair use on the internet. The objectives being:

- a) To share the *TeamSTARS* "tsWxGUI" Toolkit author's insight (design requirements, applicable third-party technologies and building blocks, development strategy and implementation plan);
 - b) To enrich each reader's understanding of the design, implementation and usage concepts;
 - c) To provide a coherent snapshot of applicable third-party reference material which is subject to change over time; and
 - d) To combine a large, diverse set of building blocks into a coherent whole, giving appropriate Authorship attribution.
- 3 Hopefully, your comments, suggestions and enhancements will improve the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit for yourself and others.

Richard S. Gordon, a.k.a. Software Gadgetry
TeamSTARS "tsWxGTUI_PyVx" Toolkit
Principal Engineer, Developer,
Author and Publisher
July 3, 2015

Contents

1	SCOPE (System Specification)	5
1.1	Identification (System Specification)	5
1.2	Purpose (System Specification)	8
1.3	Document Overview (System Specification)	9
2	SYSTEM OVERVIEW (tsWxGTUI Introduction)	21
2.1	Purpose of System and Software	21
2.1.1	Executive Summary	22
2.1.2	Development Objectives	23
2.2	General Nature of the System and Software	26
2.2.1	System Block Diagrams	27
2.2.2	System Components	38
2.2.3	Platform Configurations (Release Notes)	52
2.3	History of System Development, Operation and Maintenance	61
2.3.1	Embedded Software Development Background	61
2.3.2	Python Programming Background	62
2.3.3	Documentation Tool Background	62
2.3.4	Evolution of the "tsWxGTUI_PyVx" Toolkit	64
3	OPERATOR INTERFACE	67
3.1	Command Line Interface (CLI)	67
3.1.1	Customizable CLI Features	67
3.1.2	Command Line Input Syntax	73
3.1.3	Command Line Input Parser	74
3.1.4	Command Line Output Examples	78
3.1.5	Log File Examples	89
3.2	Graphical User Interface (GUI)	123
3.2.1	Customizable GUI Features	123
3.2.2	Graphical User Interface Examples	127
3.2.3	Graphical Desktop Features	148
3.2.4	Graphical Component Features	158
4	APPLICATION PROGRAMMING INTERFACE	219
4.1	Command Line Interface API	220
4.1.1	Python Command Line Parsing API	220
4.1.2	Application Run Time Environment API	222
4.1.3	tsCommandLineInterface	229
4.1.4	tsConfigObject (Obsolete)	230
4.1.5	tsDataBase (Future)	230
4.1.6	tsDecorator (Future)	231
4.1.7	tsExceptions	231
4.1.8	tsLogger	240
4.1.9	tsOperatorSettingsBuilder API (Future)	243
4.1.10	tsOperatorSettingsParser API	243

4.1.11	tsPlatformRunTimeEnvironment	248
4.1.12	tsReportUtility	248
4.1.13	tsSysCommand	248
4.1.14	tsThreadPool	249
4.2	Graphical User Interface API	249
4.2.1	nCurses API	250
4.2.2	Python Curses API	252
4.2.3	wxPython API	255
4.2.4	tsWxGTUI API	259

5 TOOLS & UTILITIES 261

5.1	tsLinesOfCodeProjectMetrics	261
5.2	tsPlatformQuery	261
5.3	tsStripComments	262
5.4	tsStripLineNumbers	262
5.5	tsTreeCopy	262
5.6	tsTreeTrimLines	262

6 DEFINITIONS, ABBREVIATIONS, AND ACRONYMS 263

7 REFERENCED DOCUMENTS 287

7.1	Project Documents	287
7.2	Release Distribution Documents	290
7.3	External Documents	294

8 NOTES 297

8.1	Operator Interface Technology	297
8.1.1	Shells	298
8.1.2	Desktop Environments	308
8.1.3	Terminal Device Interface (TDI)	316

9 APPENDIXES 332

10 APPENDIX A - BASELINE HOST COMPUTER PLATFORMS 333

10.1	Currently Used Platforms (Release Notes)	334
10.2	Previously Used Platforms (Release Notes)	339
10.3	Designing Embedded Systems with Linux and Python	342
10.3.1	Embedded Python	349

11 APPENDIX B - API RELATIONSHIP 351

11.1	High, Low and Extended API Relationships	352
11.2	Comparison with “wxPython	353
11.3	High, Low and Extended API Relationships	357

12 APPENDIX C - DELIVERABLES 379

13 APPENDIX D - ACCOMPLISHMENTS (Draft) 385

13.1	Capabilities	385
13.1.1	Platform Hardware and Software	385
13.1.2	tsLibCLI Software	386
13.1.3	tsToolsCLI Capabilities	388
13.1.4	tsTestsCLI Capabilities	389
13.1.5	tsUtilities Capabilities	389
13.1.6	tsLibGUI Capabilities	389
13.1.7	tsToolsGUI Capabilities	397
13.1.8	tsTestsGUI Capabilities	397
13.2	Limitations	398
13.2.1	Platform Interface Limitations	398
13.2.2	tsLibGUI Functional Limitations	399
13.3	Comparison of wxPython with tsWxGTUI (Development Plan)	399
13.4	Benefits	402

14 APPENDIX E - INHERITED, FIELD-PROVEN COMPUTER TECHNOLOGY 405

14.1	VGA-compatible Text Mode	406
14.2	POSIX	414
14.3	Command Line Interface (CLI)	422
14.3.1	Operating System Shell	424
14.3.2	Text-based User Interface (TUI)	429
14.3.3	POSIX Terminal Device Interface (TDI)	433
14.4	Graphical User Interface (GUI)	435
14.4.1	Graphical Device Interface (GDI)	435
14.4.2	Widget Toolkit	436
14.5	Python Programming Language	445
14.5.1	Python Technology Overview	446
14.5.2	CLI Shell Mode Examples	451
14.5.3	TUI Low-Level Curses Mode Examples	473
14.5.4	GUI High-Level wxPython Mode Examples	485

15 APPENDIX F - HISTORY OF SYSTEM DEVELOPMENT, OPERATION, AND MAINTENANCE 499

15.1	System Development	499
15.1.1	Rationale	500
15.1.2	Requirements	500
15.1.3	Goals	501
15.1.4	Non-Goals	502
15.1.5	Assumptions	502
15.1.6	Deliverables	504

15.1.7	Stakeholders.....	505
15.1.8	Prerequisites.....	506
15.1.9	Dependents	506
15.2	System Maintenance	506
15.2.1	Developer Platform Maintenance	507
15.2.2	Operator Platform Maintenance.....	508
15.3	System Operation.....	509

Draft


1 SCOPE (System Specification)

Define the extent of the area or subject matter that something deals with or to which it is relevant.

- *Identification (System Specification)* (on page 5)
- *Purpose (System Specification)* (on page 8)
- *Document Overview (System Specification)* (on page 9)

1.1 Identification (System Specification)

This paragraph shall contain a full identification of the system and the software to which this document applies, including, as applicable, identification number(s), title(s), abbreviation(s), version number(s), and release number(s).

PRODUCT	IDENTIFICATION
Abbreviation	"tsWxGTUI"
Icon	
Name	<p>TeamSTARS "tsWxGTUI_PyVx" Toolkit</p> <p>Generic alias for:</p> <ul style="list-style-type: none"> ▪ TeamSTARS "tsWxGTUI_Py1x" Toolkit (reserved for Python 2x backport to legacy Python 1x) ▪ TeamSTARS "tsWxGTUI_Py2x" Toolkit (updated baseline for Python 2x) ▪ TeamSTARS "tsWxGTUI_Py3x" Toolkit (ports from Python 2x until Python 2x enters bug-fix only update stage then becoming updated baseline for Python 3x) ▪ TeamSTARS "tsWxGTUI_Py4x" Toolkit (reserved for Python 3x port to future Python 4x)
Title	Python 2.x & Python 3.x based Command Line Interface (CLI) Toolkit with "Curses" based, "wxPython"-style Graphical-Text User Interface (GUI) Toolkit
Identification Number	N/A
Version Number	0.0.0

Release Number	0.0.0 (pre-alpha)
Build Date	07/03/2015

Draft

<p>Usage Terms and Conditions --- Key Points</p>	<p>This is free and open source software. The <i>TeamSTARS</i> "tsWxGTUI_PyVx" Toolkit and its 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. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.</p> <p>Please note the following:</p> <ol style="list-style-type: none"> 1 Each <i>TeamSTARS</i> "tsWxGTUI_PyVx" Toolkit distribution includes a root directory whose name ("tsWxGTUI_PyVx") has a suffix "_PyVx" that reflects the associated Python language syntax version: <ol style="list-style-type: none"> a) "tsWxGTUI_Py1x" - <i>Reserved</i> for Root of first generation syntax files and subdirectories for Python 1.0.0-1.6.1. There is currently no compelling need to justify the substantial effort to Backport from Python 2.x syntax, semantics and libraries. There would be obsolete syntax and semantic issues to be resolved. For example, issues associated with the importing of modules and data. There would be unimplemented library issues to be resolved. For example, Python 1.x supported only a Command Line Interface. It would be necessary to Backport the Python 2.x curses library in order to support a character-mode Graphical-style User Interface. b) "tsWxGTUI_Py2x" - Root of second generation syntax files and subdirectories for Python 2.0.0-2.7.9. c) "tsWxGTUI_Py3x" - Root of third generation syntax files and subdirectories for Python 3.0.0-3.4.3. d) "tsWxGTUI_Py4x" - <i>Reserved</i> for future Root of next generation syntax files and subdirectories for Python 4.x. 2 You can use, modify and redistribute the distribution only under the <i>Terms and Conditions</i> files provided in the ".tsWxGTUI_PyVx/Documents" sub-directory: <ol style="list-style-type: none"> a) "Copyright.txt" - Attribution for the principal Author(s) of intellectual property created specifically for the <i>TeamSTARS</i> "tsWxGTUI_PyVx" Toolkit. b) "Credits.txt" - Attribution for those third-party Author(s) whose intellectual property has been adapted for use in the <i>TeamSTARS</i> "tsWxGTUI_PyVx" Toolkit. c) "License.txt" - Statements of rights, obligations and limitations stipulated by the Authors of intellectual property included in the <i>TeamSTARS</i> "tsWxGTUI_PyVx" Toolkit. d) "Notices.txt" - Announcement calling the <i>TeamSTARS</i> "tsWxGTUI_PyVx" Toolkit recipient's attention to the applicable "Copyright.txt", "Credits.txt" and "License.txt" files.
---	---

1.2 Purpose (System Specification)

The *TeamSTARS* "tsWxGTUI_PyVx" Toolkit software provides building blocks and tools for the rapid prototyping and development of application programs suitable for embedded systems.

It and the hardware/software platform on which it has been installed, provide the means for developing, documenting, enhancing, operating, troubleshooting, maintaining and supporting the embedded system. Those means include the following software components:

- 1 Release Distributions** - The Toolkit distribution in one or more Python language generation-specific form(s).
 - a) *TeamSTARS* "tsWxGTUI_Py1x" Toolkit only for the first generation Python language 1.0.0-1.6.1 (reserved for future Python 2x back-port use)
 - b) *TeamSTARS* "tsWxGTUI_Py2x" Toolkit only for the second generation Python language 2.0.0-2.7.9
 - c) *TeamSTARS* "tsWxGTUI_Py3x" Toolkit only for the third generation Python language 3.0.0-3.4.3
 - d) *TeamSTARS* "tsWxGTUI_Py4x" Toolkit only for the fourth generation Python language 4.0.0 (reserved for future Python 3x port use)
 - e) *TeamSTARS* "tsWxGTUI_PyVx" Toolkit containing two or more of the above single generation Python language releases
- 2 Toolkit Components** - Toolkit building-block components are general-purpose, re-usable and enable the application developer to focus on the application specific functionality and not waste effort re-inventing and re-implementing the functionality typical of Command Line and Graphical User Interfaces. Components include:
 - a) **tsToolkitCLI** - Python-based toolkit for development of applications featuring a Command Line Interface (CLI).
 - b) **tsToolkitGUI** - Python and Curses-based toolkit for development of applications featuring a character-mode Graphical-style User Interface (GUI).
- 3 Toolkit Applications** - Applications typically feature "user-friendly" Command Line and/or Graphical-style User Interfaces that can be controlled locally or remotely. Commercial, industrial, medical and military embedded systems are customized and optimized for a specific use. Unlike their general-purpose desktop, laptop and workstation counterparts, they typically have limited, application-specific processing, memory, communication, input/output and file storage resources. Typical applications include:
 - a) **Automation** - The use of various control systems for operating equipment such as machinery, processes in factories, telephone networks, steering and stabilization of ships, aircraft and other applications with minimal or reduced human intervention.
 - b) **Communication** - The application of telecommunications technology for the transmission of data to, from, or between computers over dedicated or time shared hardware.

- c) **Control** - The application of one or more devices, to manage, command, direct or regulate the behavior of other device(s) or system(s). Industrial control systems, as used in industrial production, control equipment or machinery. In open loop control systems output is generated based only on inputs. In closed loop (feedback) control systems current output is taken into consideration and corrections are made based on feedback.
- d) **Diagnostic** - The application of technology to locate problems with software, hardware, or any combination there of in a system, or a network of systems. Diagnostics typically provide guidance to the user to solve issues.
- e) **Instrumentation** - The application of technology for the measurement and control of process variables within a production or manufacturing area. An instrument is a device that measures a physical quantity such as flow, temperature, level, distance, angle, or pressure. Instruments may be as simple as direct reading thermometers or may be complex multi-variable process analyzers. Instruments are often part of a control system in refineries, factories, and vehicles.
- f) **Simulation** - The application of technology for the imitation of the operation of a real-world process or system over time. The act of simulating something first requires that a model be developed; this model represents the key characteristics or behaviors/functions of the selected physical or abstract system or process. The model represents the system itself, whereas the simulation represents the operation of the system over time.

1.3 Document Overview (System Specification)

This paragraph shall summarize the purpose and contents of this document and shall describe any security or privacy considerations associated with its use.

The Introduction is one of a set of reference document volumes for individuals installing, developing, maintaining, troubleshooting and using the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit and application programs developed with the Toolkit. It and the other documents are described below.

VOL	TITLE	CONTENTS
0	Announcement	This document alerts potential and existing users of the availability, capabilities, limitations and sources for the latest source code release and technical support.
1	Brochure	<p>From Wikipedia, the free encyclopedia:</p> <p>"A brochure (also referred to as a pamphlet) is a type of leaflet.</p> <p>Brochures are advertising pieces mainly used to introduce a company or organization, and inform about products and/or services to a target audience.</p> <p>Brochures are distributed by mail, handed personally or placed in brochure racks.</p> <p>The most common types of single-sheet brochures are the bi-fold (a single sheet printed on both sides and folded into halves) and the tri-</p>

		<p>fold (the same, but folded into thirds). A bi-fold brochure results in four panels (two panels on each side), while a tri-fold results in six panels (three panels on each side).</p> <p>Other folder arrangements are possible: the accordion or "Z-fold" method, the "C-fold" method, etc. Larger sheets, such as those with detailed maps or expansive photo spreads, are folded into four, five, or six panels. When two card fascia are affixed to the outer panels of the z-folded brochure, it is commonly known as a "Z-card".</p> <p>Booklet brochures are made of multiple sheets most often saddle stitched (stapled on the creased edge) or "perfect bound" like a paperback book, and result in eight panels or more.</p> <p>Brochures are often printed using four color process on thick gloss paper to give an initial impression of quality. Businesses may turn out small quantities of brochures on a computer printer or on a digital printer, but offset printing turns out higher quantities for less cost.</p> <p>Compared with a flyer or a handbill, a brochure usually uses higher-quality paper, more color, and is folded."</p> <p>Included are the following topics:</p> <ol style="list-style-type: none"> 1 Features <ol style="list-style-type: none"> a) Command Line Interface (CLI) b) Graphical-style User Interface (GUI) c) System Block Diagrams 2 Usage Applications <ol style="list-style-type: none"> a) Development Platforms b) Embedded Systems 3 Design Goals <p>What you can look forward to doing with the TeamSTARS "tsWxGTUI_PyVx" Toolkit?</p> <ol style="list-style-type: none"> a) Local Equipment Monitoring & Control b) Remote Equipment Monitoring & Control 4 Design Non-Goals <p>What the TeamSTARS"tsWxGTUI_PyVx" Toolkit is NOT ?</p> 5 Usage Terms & Conditions 6 SCREENSHOTS <ol style="list-style-type: none"> a) XTERM Terminal Emulator with 8-Color / 64-Color Pairs b) VT100 Terminal Emulator with 1-Color / 2-Color Pairs
2	Introduction	<p>This document orients the reader to the goals and non-goals for the "tsWxGTUI_PyVx" Toolkit. Included are the following topics:</p> <ol style="list-style-type: none"> 1 SCOPE (System Specification)

		<ul style="list-style-type: none"> a) Identification (System Specification) b) Purpose (System Specification) c) Document Overview (System Specification)
		2 SYSTEM OVERVIEW (tsWxGTUI Introduction) <ul style="list-style-type: none"> a) Purpose of System and Software b) General Nature of the System and Software c) History of System Development, Operation and Maintenance
		3 OPERATOR INTERFACE <ul style="list-style-type: none"> a) Command Line Interface (CLI) b) Graphical User Interface (GUI)
		4 APPLICATION PROGRAMMING INTERFACE <ul style="list-style-type: none"> a) Command Line Interface API b) Graphical User Interface API
		5 TOOLS & UTILITIES <ul style="list-style-type: none"> a) tsLinesOfCodeProjectMetrics b) tsPlatformQuery c) tStripComments d) tsStripLineNumbers e) tsTreeCopy f) tsTreeTrimLines
		6 DEFINITIONS, ABBREVIATIONS, AND ACRONYMS
		7 REFERENCED DOCUMENTS <ul style="list-style-type: none"> a) Project Documents b) Release Distribution Documents c) External Documents
		8 NOTES <ul style="list-style-type: none"> a) Operator Interface Technology
		9 APPENDIXES <ul style="list-style-type: none"> a) Appendix A - Baseline Host Computer Platforms b) Appendix B - API Relationship c) Appendix C - Deliverables d) Appendix D - Accomplishments (Draft) e) Appendix E - Inherited, Field-Proven Computer

		<p>Technology</p> <p>f) Appendix F - History of System Development. Operation, and Maintenance</p>
3	Terms & Conditions	<p>This document alerts potential users and reminds existing users to the rules which the user must agree to abide by in order to use, modify and redistribute the "tsWxGTUI_PyVx" Toolkit and/or any of its components.</p> <p>Included are the following topics:</p> <p>1 TERMS & CONDITIONS</p> <p>a) Copyright - Identifies the original author(s) of source code and documentation for building block libraries and application programs.</p> <p>b) License - Identifies the original author's rules for using, modifying and redistributing source code and documentation for building block libraries and application programs.</p> <p>c) Notices - Identifier placed on copies of the source code and documentation to inform the world of copyright ownership and the applicable license.</p> <p>d) Splash Screen Designer's Guide - Identifies the original author's personal guidelines for incorporating Copyright and License notices in Command Line Interface(s) and Graphical-style User Interface(s).</p>
4	Software Development Plan	<p>This document specifies a developer's plans for conducting a software development effort. The term "software development" is meant to include new development, modification, reuse, re-engineering, maintenance, and all other activities resulting in software products.</p> <p>The plan provides the acquirer insight into, and a tool for monitoring, the processes to be followed for software development, the methods to be used, the approach to be followed for each activity, and project schedules, organization, and resources.</p> <p>Included are the following topics:</p> <p>1 SCOPE (System Specification)</p> <p>2 DEFINITIONS, ABBREVIATIONS, AND ACRONYMS</p> <p>3 OVERVIEW OF REQUIRED WORK</p> <p>a) Purpose</p> <p>b) Requirements and Constraints</p> <p>c) Concept</p> <p>4 PLANS FOR PERFORMING GENERAL SOFTWARE DEVELOPMENT ACTIVITIES</p> <p>a) Software Development Process</p>

		<ul style="list-style-type: none"> b) General Plans for Software Development 5 PLANS FOR PERFORMING DETAILED SOFTWARE DEVELOPMENT ACTIVITIES <ul style="list-style-type: none"> a) Project Planning and Oversight b) Establishing A Software Development Environment c) System Requirements Analysis d) Software Design e) Software implementation And unit Test f) Unit integration And Testong g) CSCI Qualification Testing h) CSCI/HWCI Integration And Testing i) System Qualification Testing j) Preparing for Software Use k) Preparing for Software Transition l) Software Configuration Management m) Software Product Evaluation n) Software Quality Assurance o) Corrective Action p) Joint Technical and Management Reviews q) Other Software Development Activities 6 SCHEDULES AND ACTIVITY NETWORK 7 PROJECT ORGANIZATION AND RESOURCES <ul style="list-style-type: none"> a) Project organization b) Project Resources 8 NOTES 9 APPENDIXES 10 TO-DO-LIST <ul style="list-style-type: none"> a) New Features b) Modifications c) Troubleshooting d) Validation/Regression Test 11 SAMPLE SCREEN SHOTS
5	System Specification	This document specifies the required behavior of an engineering system. The documentation typically describes what is needed by the system user as well as requested properties of inputs and outputs (e.g.

	<p>of the software system).</p> <p>Included are the following topics:</p> <ol style="list-style-type: none"> 1 SCOPE (System Specification) 2 DEFINITIONS, ABBREVIATIONS, AND ACRONYMS 3 REQUIREMENTS 4 QUALIFICATION PROVISIONS <ol style="list-style-type: none"> a) Demonstration b) Test c) Analysis d) Inspection e) Special Qualification Methods 5 REQUIREMENTS TRACEABILITY 6 NOTES <ol style="list-style-type: none"> a) Partial List of Widget Toolkits b) Use Case(s) <ul style="list-style-type: none"> System Administrator Software Engineer System Operator 7 APPENDIXES 8 APPENDIX A - REQUIRED STATES AND MODES 9 APPENDIX B - SYSTEM CAPABILITY REQUIREMENTS 10 APPENDIX C - SYSTEM EXTERNAL INTERFACE REQUIREMENTS 11 APPENDIX D - SYSTEM INTERNAL INTERFACE REQUIREMENTS 12 APPENDIX E - SYSTEM INTERNAL DATA REQUIREMENTS 13 APPENDIX F - ADAPTATION REQUIREMENTS 14 APPENDIX G - SAFETY REQUIREMENTS 15 APPENDIX H - SECURITY AND PRIVACY REQUIREMENTS 16 APPENDIX I - SYSTEM ENVIRONMENT REQUIREMENTS 17 APPENDIX J - COMPUTER RESOURCE REQUIREMENTS
--	--

		<p>18 APPENDIX K - SYSTEM QUALITY FACTORS</p> <p>19 APPENDIX L - DESIGN AND CONSTRUCTION CONSTRAINTS</p> <p>20 APPENDIX M - PERSONNEL-RELATED REQUIREMENTS</p> <p>21 APPENDIX N - TRAINING-RELATED REQUIREMENTS</p> <p>22 APPENDIX O - LOGISTICS-RELATED REQUIREMENTS</p> <p>23 APPENDIX P - OTHER REQUIREMENTS</p> <p>24 APPENDIX Q - PACKAGING REQUIREMENTS</p> <p>25 APPENDIX R - PRECEDENCE AND CRITICALITY OF REQUIREMENTS</p>
6	Interface Requirements Specification	<p>This document specifies the requirements imposed on one or more systems, subsystems, Hardware Configuration Items (HWCIs), Computer Software Configuration Items (CSCIs), manual operations, or other system components to achieve one or more interfaces among these entities. An IRS can cover any number of interfaces.</p> <p>Included are the following topics:</p> <p>1 SCOPE (System Specification)</p> <p>2 DEFINITIONS, ABBREVIATIONS, AND ACRONYMS</p> <p>3 REQUIREMENTS</p> <p>a) Interface Identification and Diagrams</p> <p>b) (Project unique identifier of interface)</p> <p>c) Terminal Device Interface (TDI)</p> <p>d) Precedence and Criticality of Requirements</p> <p>4 QUALIFICATION PROVISIONS</p> <p>5 REQUIREMENTS TRACEABILITY</p> <p>6 NOTES</p> <p>7 APPENDIXES</p>
7	Software Requirements Specification	<p>This document specifies the requirements for a Computer Software Configuration Item (CSCI) and the methods to be used to ensure that each requirement has been met. Requirements pertaining to the CSCI's external interfaces may be presented in the SRS or in one or more Interface Requirements Specifications (IRSs).</p> <p>Included are the following topics:</p> <p>1 SCOPE (System Specification)</p> <p>2 DEFINITIONS, ABBREVIATIONS, AND ACRONYMS</p> <p>3 REQUIREMENTS</p>

		<ul style="list-style-type: none"> a) Required States and Modes b) CSCI Capability Requirements c) CSCI External Interface Requirements d) CSCI Internal Interface Requirements e) CSCI Internal Data Requirements f) Adaptation Requirements g) Safety Requirements h) Security and Privacy Requirements i) CSCI Environment Requirements j) Computer Resource Requirements k) Software Quality Factors l) Design and Construction Constraints m) Personnel-related Requirements n) Training-related Requirements o) Logistics-related Requirements p) Other Requirements q) Packaging Requirements r) Precedence and Criticality of Requirements <p>4 QUALIFICATION PROVISIONS</p> <p>5 REQUIREMENTS TRACEABILITY</p> <p>6 NOTES</p> <ul style="list-style-type: none"> a) TO-DO-LIST b) Command Line Interface Library c) Graphical Text User Interface Library <p>7 APPENDIXES</p> <ul style="list-style-type: none"> a) Appendix A - Nature of System and Software <p>8 TECHNICAL IMPACT ANALYSIS</p> <p>9 TEST REQUIREMENTS AND RESTRICTIONS</p> <p>10 USE CASES</p> <p>11 TRACEABILITY INFORMATION</p> <p>12 CLASS LIST BY CATEGORY</p>
8	Release Notes	<p>This document is distributed with software products, often when the product is still in the development or test state (e.g., a beta release). For products that have already been in use by clients, the release note is a supplementary document that is delivered to the customer when a bug</p>

		<p>is fixed or an enhancement is made to the product.</p> <p>Included are the following topics:</p> <ol style="list-style-type: none"> 1 SCOPE (System Specification) 2 DEFINITIONS, ABBREVIATIONS, AND ACRONYMS 3 SYSTEM REQUIREMENTS <ol style="list-style-type: none"> a) Platform Configurations (Release Notes) b) Network Configurations (Release Notes) 4 PACKAGE CONTENTS <ol style="list-style-type: none"> a) Command Line Interface Library b) Graphical Text User Interface Library 5 FIXED BUGS & ISSUES 6 KNOWN BUGS & ISSUES 7 NEW FEATURES 8 APPLICATION NOTES <ol style="list-style-type: none"> a) Installation Procedure b) User Interface Design c) Developer 9 PERFORMANCE TUNING 10 ACCEPTANCE TESTING 11 COMPONENT STATUS 12 APPENDIXES <ol style="list-style-type: none"> a) Appendix A - Baseline Host Computer Platforms b) Appendix B - API Relationship c) Appendix C - Deliverables
9	Software Users Manual	<p>This document tells a hands-on software user (developer and operator) how to install and use the associated computer software (<i>TeamSTARS</i> "tsWxGTUI_PyVx" Toolkit). It may also cover a particular aspect of software operation, such as instructions for a particular position or task.</p> <p>Included are the following topics:</p> <ol style="list-style-type: none"> 1 SCOPE (System Specification) 2 SOFTWARE SUMMARY <ol style="list-style-type: none"> a) Software Application b) Software Inventory c) Software Environment

		<ul style="list-style-type: none"> d) Software Organization and Overview of Operation e) Security and Privacy f) Assistance and Problem Reporting <p>3 ACCESS TO THE SOFTWARE</p> <ul style="list-style-type: none"> a) First-time User of the Software b) Initiating a Session c) Stopping and Suspending Work <p>4 PROCESSING REFERENCE GUIDE</p> <ul style="list-style-type: none"> a) Capabilities b) Conventions c) Processing Procedures d) Related Processing e) Data Backup f) Recovery from Errors, Malfunctions, and Emergencies g) Messages h) Quick Reference Guide <p>5 NOTES</p> <ul style="list-style-type: none"> a) Use Case(s) b) Baseline Toolkit Development Platforms <p>6 DEFINITIONS, ABBREVIATIONS, AND ACRONYMS</p> <p>7 APPENDIXES</p> <p>8 APPENDIX A - BASELINE HOST COMPUTER PLATFORMS</p> <p>9 APPENDIX B - API RELATIONSHIP</p> <p>10 APPENDIX C - DELIVERABLES</p> <p>11 APPENDIX D - LOG FILES</p>
10	Appendixes	<p>1 Introduction</p> <ul style="list-style-type: none"> a) Appendix A - Baseline Host Computer Platforms b) Appendix B - API Relationship c) Appendix C - Deliverables d) Appendix D - Accomplishments (Draft) e) Appendix E - Inherited, Field-Proven Computer Technology f) Appendix F - History of System Development. Operation, and Maintenance

		2 Software Development Plan
		3 System Specification
		a) APPENDIX A - REQUIRED STATES AND MODES
		b) APPENDIX B - SYSTEM CAPABILITY REQUIREMENTS
		c) APPENDIX C - SYSTEM EXTERNAL INTERFACE REQUIREMENTS
		d) APPENDIX D - SYSTEM INTERNAL INTERFACE REQUIREMENTS
		e) APPENDIX E - SYSTEM INTERNAL DATA REQUIREMENTS
		f) APPENDIX F - ADAPTATION REQUIREMENTS
		g) APPENDIX G - SAFETY REQUIREMENTS
		h) APPENDIX H - SECURITY AND PRIVACY REQUIREMENTS
		i) APPENDIX I - SYSTEM ENVIRONMENT REQUIREMENTS
		j) APPENDIX J - COMPUTER RESOURCE REQUIREMENTS
		k) APPENDIX K - SYSTEM QUALITY FACTORS
		l) APPENDIX L - DESIGN AND CONSTRUCTION CONSTRAINTS
		m) APPENDIX M - PERSONNEL-RELATED REQUIREMENTS
		n) APPENDIX N - TRAINING-RELATED REQUIREMENTS
		o) APPENDIX O - LOGISTICS-RELATED REQUIREMENTS
		p) APPENDIX P - OTHER REQUIREMENTS
		q) APPENDIX Q - PACKAGING REQUIREMENTS
		r) APPENDIX R - PRECEDENCE AND CRITICALITY OF REQUIREMENTS
		4 Interface Requirements Specification
		5 Software Requirements Specification
		a) APPENDIX A - Nature of System and Software
		6 Release Notes
		a) APPENDIX A - Baseline Host Computer Platforms

		<ul style="list-style-type: none">b) APPENDIX B - API Relationshipc) APPENDIX C - Deliverables 7 Software Users Manual
11	Dictionary	A reference document that contains words, phrases, abbreviations and acronyms that are listed in alphabetical order with information about the their meanings in the context of the <i>TeamSTARS</i> "tsWxGTUI_PyVx" Toolkit software.
12	To-Do	

Draft

2 SYSTEM OVERVIEW (tsWxGTUI Introduction)

This section shall briefly describe the purpose and nature of the computer system's hardware and software to which this document applies. It shall also briefly describe the history of system development, operation and maintenance.

- *Purpose of System and Software* (on page 21)
- *General Nature of the System and Software* (on page 26)
- *History of System Development, Operation and Maintenance* (on page 61)

2.1 Purpose of System and Software

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

- *Executive Summary* (on page 22)
- *Development Objectives* (on page 23)

2.1.1 Executive Summary

This paragraph shall state the essence of the purpose of the system and the software to which this document applies.

Software development systems typically have sufficient and upgradable resources including 32-bit/64-bit processors, random access memory, non-volatile 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. Such 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 *TeamSTARS* "tsWxGTUI_PyVx" 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

The *TeamSTARS* "tsWxGTUI_PyVx" Toolkit software is engineered to facilitate the development and use of application programs with the following features:

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).

From "<https://docs.python.org/3/howto/curses.html>" by A.M. Kuchling, Eric S. Raymond

"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 various programs. 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 graphical 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 niche is on small-footprint or embedded Unixes that don't run an X server. Another is tools such as OS installers and kernel configurators that may have to run before any graphical support is available.

The curses library provides fairly basic functionality, providing the programmer with an abstraction of a display containing multiple non-overlapping windows of text. The contents of a window can be changed in various ways—adding text, erasing it, changing its appearance—and the curses library will figure out what control codes need to be sent to the terminal to produce the right output. curses doesn't provide many user-interface concepts such as buttons, checkboxes, or dialogs; if you need such features, consider a user interface library such as Urwid.

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.

The Windows version of Python doesn't include the curses module. A ported version called UniCurses is available. You could also try the Console module written by Fredrik Lundh, which doesn't use the same API as curses but provides cursor-addressable text output and full support for mouse and keyboard input."

Building on the available curses capabilities, the TeamSTARS "tsWxGTUI_PyVx" Toolkit creates and manages multiple overlapping windows (frames, dialogs and panels) and such desktop features as a task bar, status bar, scroll bar, buttons, check boxes, radio buttons, horizontal/vertical box layout sizer, grid layout sizer, 68-color palette mapped into the available 8-/16-color xterm/non-color vt100 terminal palette, configuration property files and event message log files with event severity level indicators and with date and time stamps by emulating the Application Programming Interface (API) of the Python programming language wrapper ("wxPython") to the pixel-mode cross-platform "wxWidgets" GUI Toolkit which itself is implemented in C++.

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.

2.1.2 Development Objectives

This paragraph shall briefly state the development objectives for the software to which this document applies.

The TeamSTARS "tsWxGTUI_PyVx" Toolkit shall provide the following capabilities:

- 1** A means for one or more System Administrators, Software Engineers, System Operators and Field Service Personnel to interactively monitor and control local and remote computer equipment via either or both of the following, as appropriate to the application (such as automation, communication, control, diagnostic, instrumentation and simulation):

- a) Command Line Interface (CLI)
 - b) Graphical-style User Interface (GUI)
- 2** A means for one or more Software Engineers to develop applications and toolkit enhancements that 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, tools, utilities and tests.
 - c) Designs suitable for installation in computer systems to be used for software development and documentation. Such general-purpose desktop, laptop and workstation computer systems typically have upgradable or at least sufficient processing, memory, communication, input/output and file storage resources.
 - d) Designs suitable for installation and use in embedded systems 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.
 - e) CLI and GUI interfaces that are attractive and "user-friendly".
- 3** A means for the Software Engineer to re-use existing desktop, laptop, workstation and embedded system application programs 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 Software Engineer to troubleshoot design and user issues via date and time stamped operational, diagnostic and exception logs.
- 5** A means for the Software Engineer to develop application programs and toolkit components that co-ordinate the automated operation and failure recovery of multiple applications by use of 8-bit, POSIX-style exit codes and error messages.
- 6** A means for System Administrators, Software Engineers, System Operators and Field Service Personnel to use
- a) A mouse, trackball, touchpad or touchscreen to select GUI objects when the platform supports such a device.
 - b) A keyboard to select GUI objects, via hot-keys, when the platform does not support a mouse, trackball, touchpad or touchscreen.
- 7** A means (Site-Packages) for the application and toolkit developer to import application and library building-block components from a single-level (Global Module Index-style) directory file system. This is intended to extend the capabilities of the standard Python library packages and modules by the standard procedures for installing and using those third-party library packages and modules which are not released by the Python Software Foundation.
- 8** A means (Developer-Sandboxes) for the application and toolkit developer to import application and library building-block components from a nested, multilevel 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.

- 9 Sufficient system hardware and software reliability, availability and maintainability to continuously operate unattended and without failure for extended periods of time on an "AS IS" basis unless otherwise certified by suitably qualified system integrators.

Draft

2.2 General Nature of the System and Software

System Administrators, Software Engineers, System Operators and Field Service personnel use the following hardware and software configurations:

- 1 *System Block Diagrams* (on page 27)
- 2 *System Components* (on page 38)
- 3 System Configurations
- 4 *History of System Development, Operation and Maintenance* (on page 61)

NOTES:

Pre-alpha release of the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit:

- 1 "tsWxGTUI_Py2x-0.0.0"
 - a) Command Line Interface features are released for Python 2.0.1-2.7.9.
 - b) Graphical-style User Interface features are released for Python 2.6.4-2.7.9.
- 2 "tsWxGTUI_Py3x-0.0.0"
 - a) Command Line Interface features are released for Python 3.0.1-3.4.2.
 - b) Graphical-style User Interface features are released for Python 3.1.5-3.4.2.

Excerpt From <https://wiki.python.org/moin/Python2orPython3>
(<https://wiki.python.org/moin/Python2orPython3>):

"What are the differences?

Short version: Python 2.x is legacy, Python 3.x is the present and future of the language.

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. 3.x is under active development and has already seen over two years of stable releases, including version 3.3 in 2012. This means that all recent standard library improvements, for example, are only available 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).

The What's New in Python 3.0 document provides a good overview of the major language changes and likely sources of incompatibility with existing Python 2.x code.

However, the broader Python ecosystem has amassed a significant amount of quality software over the years. The

Pre-alpha release of the *Team*STARS "tsWxGTUI_PyVx" Toolkit:

1 "tsWxGTUI_Py2x-0.0.0"

- a) Command Line Interface features are released for Python 2.0.1-2.7.9.
- b) Graphical-style User Interface features are released for Python 2.6.4-2.7.9.

2 "tsWxGTUI_Py3x-0.0.0"

- a) Command Line Interface features are released for Python 3.0.1-3.4.2.
- b) Graphical-style User Interface features are released for Python 3.1.5-3.4.2.

downside of breaking backwards compatibility in 3.x is that some of that software (especially in-house software in companies) still doesn't work on 3.x yet."

2.2.1 System Block Diagrams

- 1** *Block Diagram* (on page 29) - High level depiction of the organizational, functional and interface relationship between the *Team*STARS "tsWxGTUI_PyVx" Toolkit components and users (System Administrator, Software Engineer, System Operator and Field Service personnel):

- a) A Command Line Interface ("tsToolkitCLI") - It provides the foundation for the Graphical User Interface ("tsToolkitGUI). Its components include:

tsApplicationPkg --- 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).

tsCommandLineEnvPkg --- 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 coordinate other application programs.

tsCommandLineInterfacePkg --- 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.

tsCxGlobalsPkg --- Module to establish configuration constants and macro-type functions for the Command Line Interface mode of the "tsWxGTUI_PyVx" Toolkit. It 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 manual searches to locate and modify or restore a constant appropriate to the current configuration. It also provides a theme-based mechanism for modifying/restoring those configuration constants as appropriate for various users and their activities.

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

tsExceptionPkg --- 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.

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.

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.

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

tsReportUtilityPkg --- 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.

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

- b) A Graphical-style User Interface ("tsToolkitGUI") - It uses the services of the "tsToolkitCLI" when appropriate.

tsWxPkg --- Collection of approximately 100 Classes that use the services of the Python Curses module to create a character-mode emulation of their pixel-mode "wxPython" Class counterparts.

tsWxGlobals --- Module 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 manual searches to locate and modify or restore a constant appropriate to the current configuration. It also provides a theme-based mechanism for modifying/restoring those configuration constants as appropriate for the character-mode emulation of the following pixel-mode "wxPython" features:

The collection includes widgets for frames, dialogs, panels, buttons, check boxes, radio boxes/buttons and scrollable text windows. It includes box and grid sizers.

It also includes classes to emulate the host operating system theme-based color palette management, task bar, scroll bar, mouse click and window focus control services used/expected by "wxPython".

The Application Programming Interface (API) retains those "wxPython" keyword-value pairs and positional arguments needed for pixel-mode application compatibility. It adds keyword-value pairs and positional arguments needed only for internal (non-application) Toolkit use.

- 2 **Stand Alone System Architecture** (on page 30) - High level depiction and description of the components of and relationship between components of an isolated system operating by itself.
- 3 **Stand Among System Architecture** (on page 34) - High level depiction and description of the components of and relationship between two or more networked systems operating in collaboration with each other.

2.2.1.1 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"

Draft

Graphical-style User Interface (tsToolkitGUI)	
<ul style="list-style-type: none"> The "tsToolsGUI" is a set of graphical-style user interface application programs for tracking software development metrics. 	<ul style="list-style-type: none"> The "tsTestsGUI" is a set of graphical-style user interface application programs for regression testing and tutorial demos.
<ul style="list-style-type: none"> 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)	
<ul style="list-style-type: none"> 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. 	<ul style="list-style-type: none"> The "tsTestsCLI" is a set of command line interface application programs and scripts for regression testing and tutorial demos.
<ul style="list-style-type: none"> 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. 	
<ul style="list-style-type: none"> 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

2.2.1.2 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").

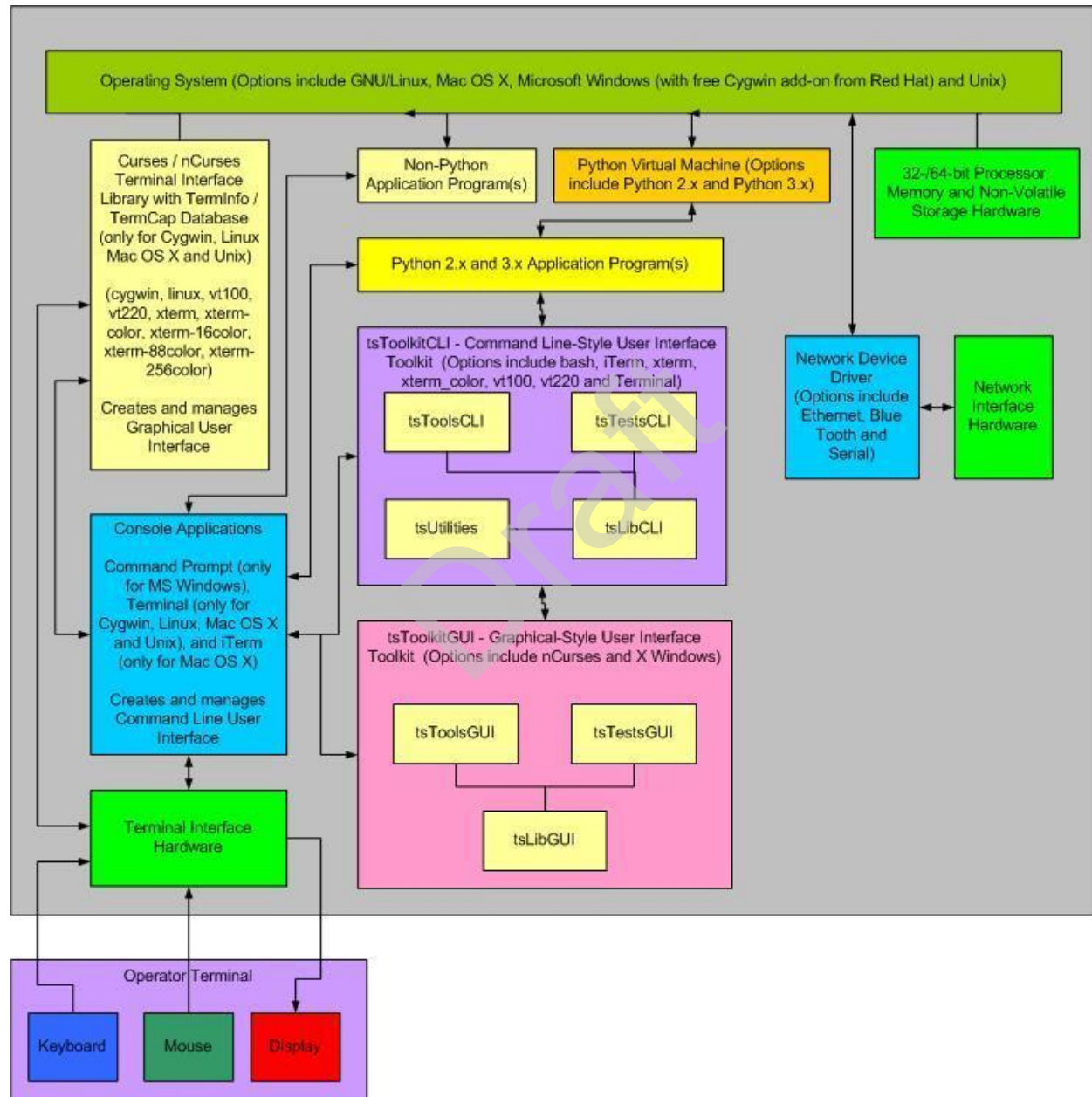
The following description uses the component names as depicted in the **Block Diagram** (on page 29)

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.

- a) An RJ-45 Ethernet port connecting to a local or wide area network. This port is capable of conducting simultaneous (full-duplex,) two-directional input and output at speeds over 100 gigabits per second. This port can also provide the interface to an optional printer shared with other network users.
- b) An RS-232C or RS-422 port for connecting a modem. Depending on the application, modems could be operated, at speeds over 56 kilobits per second, in either of two modes. In half duplex mode, each side alternated its sending and receiving roles. In full-duplex mode, each side could simultaneously send and receive.

12 Network Device Driver Interface - The optional platform specific software whose layered protocol suite (such as TCP/IP) enables the concurrent sharing of the physical connection between the local system and one or more remote systems.

NOTE: Concurrent local and remote 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.

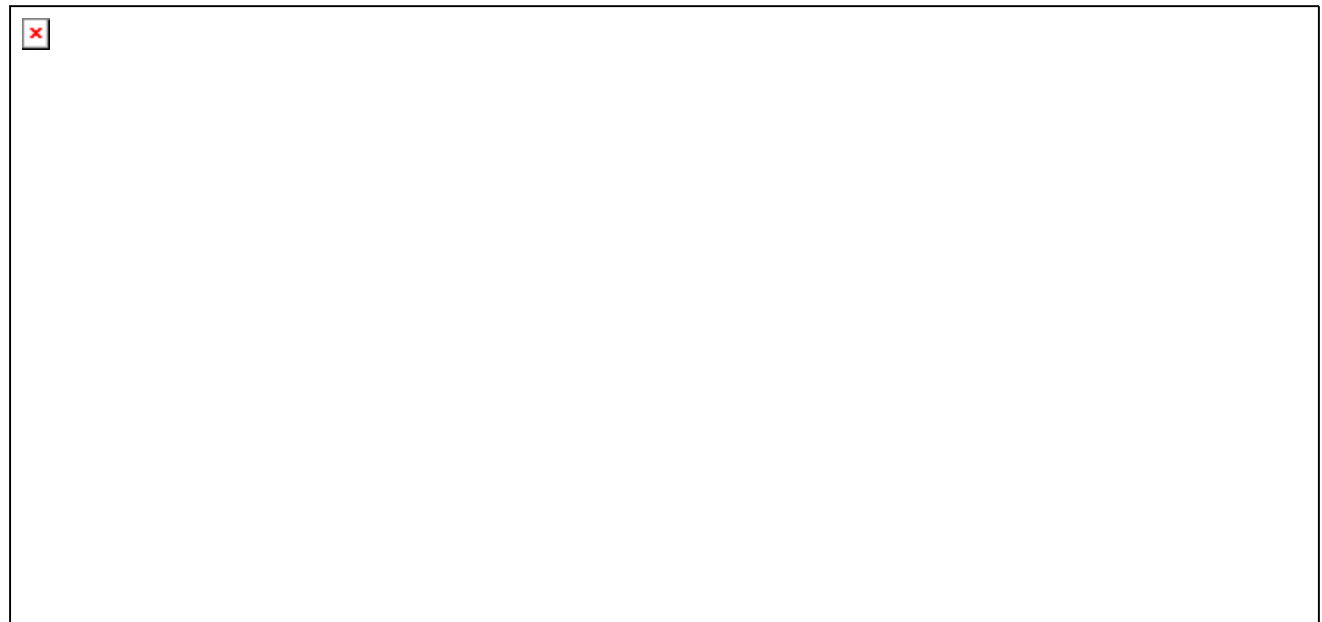
2.2.1.3 Stand Among System Architecture

The *TeamSTARS* "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*** (on page 29)

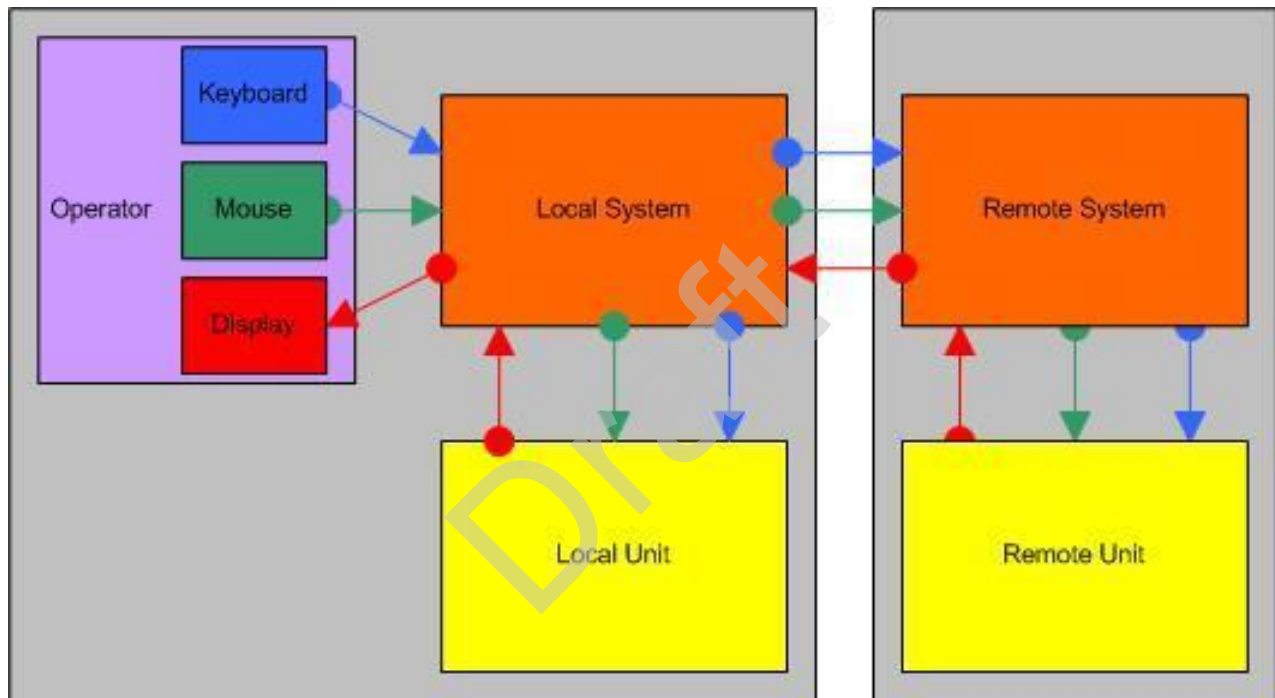
The ***Stand Alone System Architecture*** (on page 30), as previously described, may be extended to enable a single operator, working from a Local System, to interact with one or more Remote Systems.



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 36) 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.

2.2.1.3.1 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]

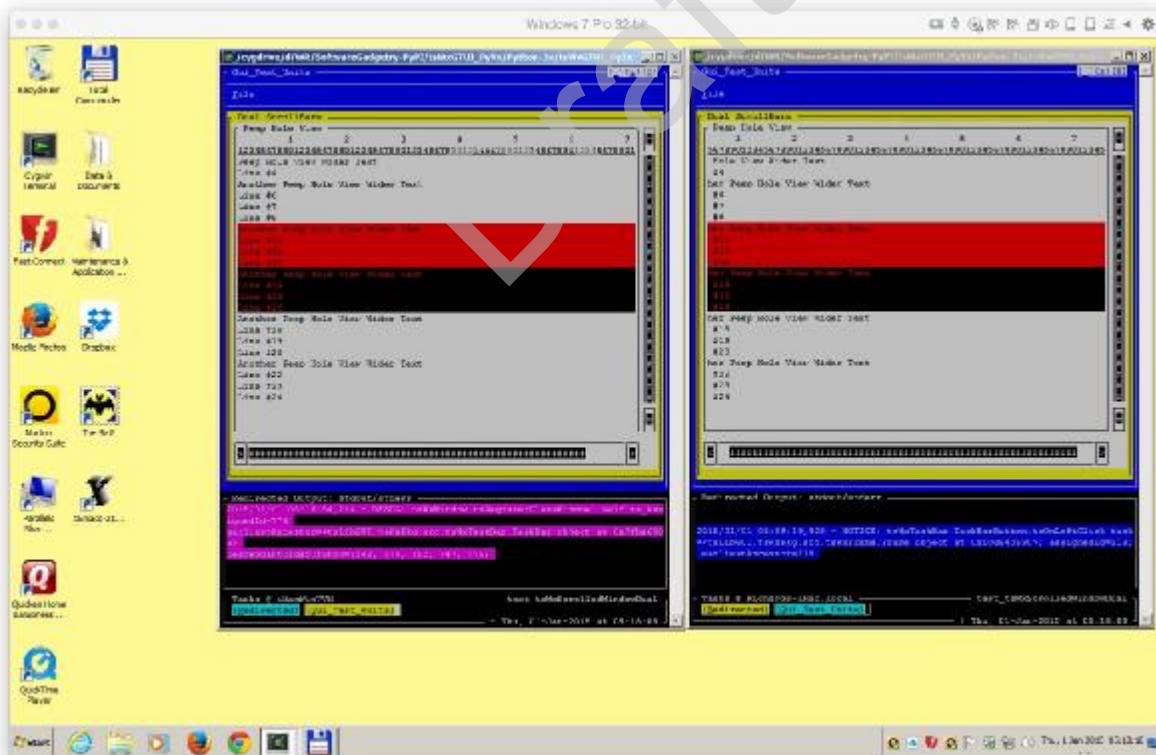
Communication sessions may be implemented as part of protocols and services at the application layer, at the session layer or at the transport layer in the OSI model.

- Application layer examples:
 - HTTP sessions, which allow associating information with individual visitors
 - A telnet remote login session
- Session layer example:
 - A Session Initiation Protocol (SIP) based Internet phone call
- Transport layer example:
 - 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).

2.2.2 System Components

- *Platform Hardware* (on page 38)
- *Platform Software* (on page 42)
- *Platform Configuration Release Notes* (see "*Platform Configurations (Release Notes)*" on page 52)

2.2.2.1 Platform Hardware

System Administrators, Software Engineers, System Operators and Field Service Personnel use the following Local and optional Remote Host Computer hardware components.

- *Processor Hardware* (on page 38)
- *Peripheral Hardware* (on page 41)

2.2.2.1.1 Processor Hardware

System Administrators, Software Engineers, System Operators and Field Service Personnel use the following Local and optional Remote Host Computer Processor hardware components.

- *Single or Multi-Core 32-bit/64-bit Central Processing Units (CPU)* (on page 39)
- *Random Access Memory (RAM)* (on page 39)
- *Non-Volatile Memory* (on page 39)

2.2.2.1.1.1 Single or Multi-Core 32-bit/64-bit Central Processing Units (CPU)

Excerpted From Wikipedia, the free encyclopedia:

"A central processing unit (CPU) (formerly also referred to as a central processor unit[1]) is the hardware within a computer that carries out the instructions of a computer program by performing the basic arithmetical, logical, and input/output operations of the system. The term has been in use in the computer industry at least since the early 1960s.[2] The form, design, and implementation of CPUs have changed over the course of their history, but their fundamental operation remains much the same.

A computer can have more than one CPU; this is called multiprocessing. All modern CPUs are microprocessors, meaning contained on a single chip. Some integrated circuits (ICs) can contain multiple CPUs on a single chip; those ICs are called multi-core processors. An IC containing a CPU can also contain peripheral devices, and other components of a computer system; this is called a system on a chip (SoC).

Two typical components of a CPU are the arithmetic logic unit (ALU), which performs arithmetic and logical operations, and the control unit (CU), which extracts instructions from memory and decodes and executes them, calling on the ALU when necessary.

Not all computational systems rely on a central processing unit. An array processor or vector processor has multiple parallel computing elements, with no one unit considered the "center". In the distributed computing model, problems are solved by a distributed interconnected set of processors."

2.2.2.1.1.2 Random Access Memory (RAM)

Excerpted From Wikipedia, the free encyclopedia:

"Random-access memory (RAM /ræm/) is a form of computer data storage. A random-access device allows stored data to be accessed directly in any random order. In contrast, other data storage media such as hard disks, CDs, DVDs and magnetic tape, as well as early primary memory types such as drum memory, read and write data only in a predetermined order, consecutively, because of mechanical design limitations. Therefore, the time to access a given data location varies significantly depending on its physical location.

Today, random-access memory takes the form of integrated circuits. Strictly speaking, modern types of DRAM are not random access, as data is read in bursts, although the name DRAM / RAM has stuck. However, many types of SRAM, ROM, OTP, and NOR flash are still random access even in a strict sense. RAM is normally associated with volatile types of memory (such as DRAM memory modules), where its stored information is lost if the power is removed. Many other types of non-volatile memory are RAM as well, including most types of ROM and a type of flash memory called NOR-Flash. The first RAM modules to come into the market were created in 1951 and were sold until the late 1960s and early 1970s."

2.2.2.1.1.3 Non-Volatile Memory

Hard Disk Drive(s) and/or optional Flash Memory (Solid State Drive(s)) are needed to store and retain the local or remote computer system bootstrap programs and configuration data when the computer system has been shut down by either an intentional or unexpected loss of electrical power.

1 Hard Disk Drive (Electro-Mechanical Memory)

Excerpted From Wikipedia, the free encyclopedia:

"A hard disk drive (HDD)[note 2] is a data storage device used for storing and retrieving digital information using rapidly rotating disks (platters) coated with magnetic material. An HDD retains its data even when powered off. Data is read in a random-access manner, meaning individual blocks of data can be stored or retrieved in any order rather than sequentially. An HDD consists of one or more rigid ("hard") rapidly rotating disks (platters) with magnetic heads arranged on a moving actuator arm to read and write data to the surfaces.

Introduced by IBM in 1956,[2] HDDs became the dominant secondary storage device for general-purpose computers by the early 1960s. Continuously improved, HDDs have maintained this position into the modern era of servers and personal computers. More than 200 companies have produced HDD units, though most current units are manufactured by Seagate, Toshiba and Western Digital. Worldwide revenues for HDD shipments are expected to reach \$33 billion in 2013, a decrease of approximately 12% from \$37.8 billion in 2012.

The primary characteristics of an HDD are its capacity and performance. Capacity is specified in unit prefixes corresponding to powers of 1000: a 1-terabyte (TB) drive has a capacity of 1,000 gigabytes (GB; where 1 gigabyte = 1 billion bytes). Typically, some of an HDD's capacity is unavailable to the user because it is used by the file system and the computer operating system, and possibly inbuilt redundancy for error correction and recovery. Performance is specified by the time to move the heads to a file (Average Access Time) plus the time it takes for the file to move under its head (average latency, a function of the physical rotational speed in revolutions per minute) and the speed at which the file is transmitted (data rate).

The two most common form factors for modern HDDs are 3.5-inch in desktop computers and 2.5-inch in laptops. HDDs are connected to systems by standard interface cables such as SATA (Serial ATA), USB or SAS (Serial attached SCSI) cables.

As of 2012, the primary competing technology for secondary storage is flash memory in the form of solid-state drives (SSDs). HDDs are expected to remain the dominant medium for secondary storage due to predicted continuing advantages in recording capacity and price per unit of storage;[3][4] but SSDs are replacing HDDs where speed, power consumption and durability are more important considerations than price and capacity.[5][6]"

2 Flash Memory (Electronic Memory)

Excerpted From Wikipedia, the free encyclopedia:

"Flash memory is an electronic non-volatile computer storage medium that can be electrically erased and reprogrammed.

Flash memory was developed from EEPROM (electrically erasable programmable read-only memory). There are two main types of flash memory, which are named after the NAND and NOR logic gates. The internal characteristics of the individual flash memory cells exhibit characteristics similar to those of the corresponding gates.

Whereas EPROMs had to be completely erased before being rewritten, NAND type flash memory may be written and read in blocks (or pages) which are generally much smaller than the entire device. NOR type flash allows a single machine word (byte) to be written—to an erased location—or read independently.

The NAND type is primarily used in main memory, memory cards, USB flash drives, solid-state drives, and similar products, for general storage and transfer of data. The NOR type, which allows true random access and therefore direct code execution, is used as a replacement for the older EPROM and as an alternative to certain kinds of ROM applications, whereas NOR flash memory may emulate ROM primarily at the machine code level; many digital designs need ROM (or PLA) structures for other uses, often at significantly higher speeds than (economical) flash memory may achieve.[citation needed] NAND or NOR flash memory is also often used to store configuration data in numerous digital products, a task previously made possible by EEPROMs or battery-powered static RAM.

Example applications of both types of flash memory include personal computers, PDAs, digital audio players, digital cameras, mobile phones, synthesizers, video games, scientific instrumentation, industrial robotics, medical electronics, and so on. In addition to being non-volatile, flash memory offers fast read access times, as fast as dynamic RAM, although not as fast as static RAM or ROM. Its mechanical shock resistance helps explain its popularity over hard disks in portable devices, as does its high durability, being able to withstand high pressure, temperature, immersion in water, etc.[1]

Although flash memory is technically a type of EEPROM, the term "EEPROM" is generally used to refer specifically to non-flash EEPROM which is erasable in small blocks, typically bytes. Because erase cycles are slow, the large block sizes used in flash memory erasing give it a significant speed advantage over non-flash EEPROM when writing large amounts of data. As of 2013 flash memory costs much less than byte-programmable EEPROM and has become the dominant memory type wherever a system requires a significant amount of non-volatile, solid state storage."

3 Removable, Non-volatile Memory

See Wikipedia, the free encyclopedia, for descriptions of DVDs, CDs, Floppy Disks and USB Flash Drives.

2.2.2.1.2 Peripheral Hardware

System Administrators, Software Engineers, System Operators and Field Service Personnel use the following Local and optional Remote Host Computer Peripheral hardware components.

- *Operator's Terminal(s)* (on page 41)
- *Network Communications Interface(s)* (on page 42)

2.2.2.1.2.1 Operator's Terminal(s)

System Administrators, Software Engineers, System Operators and Field Service Personnel use the following Local and optional Remote Host Computer Operator's Terminal components.

- 1 **Keyboard Input Device(s)** - Alphabetic, numeric, punctuation and control function buttons
- 2 **Display Output Device(s)** - Character strings and graphical-style objects, in two or more colors, that occupy one or more two-dimensional areas each defined by its width (in pixels or character columns) and its height (in pixels or character rows/lines)
- 3 **Optional Pointing Input Device(s)** - Mouse, TrackBall, TouchPad or TouchScreen
- 4 **Optional Printer Output Device(s)** - Laser or Ink (applied via impact or spray)

2.2.2.1.2.2 Network Communications Interface(s)

System Administrators, Software Engineers, System Operators and Field Service Personnel use the following Local and optional Remote Host Computer Network Communication components.

- 1 **Optional Wired and/or Wireless Device(s)** suitable for Local Area Network and/or Wide Area Network interconnect circuits.

From Wikipedia, the free encyclopedia:

"The network controller implements the electronic circuitry required to communicate using a specific physical layer and data link layer standard such as Ethernet, Wi-Fi or Token Ring. This provides a base for a full network protocol stack, allowing communication among small groups of computers on the same LAN and large-scale network communications through routable protocols, such as IP.

Although other network technologies exist (e.g. token ring), Ethernet has achieved near-ubiquity since the mid-1990s.

Every network controller for an IEEE 802 network such as Ethernet, Wi-Fi, or Token Ring, and every FDDI network controller, has a unique 48-bit serial number called a MAC address, which is stored in read-only memory. Every computer on an Ethernet network must have at least one controller. Normally it is safe to assume that no two network controllers will share the same address, because controller vendors purchase blocks of addresses from the Institute of Electrical and Electronics Engineers (IEEE) and assign a unique address to each controller at the time of manufacture.[2]

The NIC allows computers to communicate over a computer network. It is both an OSI layer 1 (physical layer) and layer 2 (data link layer) device, as it provides physical access to a networking medium and, for IEEE 802 networks and FDDI, provides a low-level addressing system through the use of MAC addresses. It allows users to connect to each other either by using cables or wirelessly."

- 2 **Optional Modem Device(s)** suitable for modulating an analog carrier signal to encode digital information, and also demodulating such a carrier signal to decode the transmitted information.

From Wikipedia, the free encyclopedia:

"The goal is to produce a signal that can be transmitted easily and decoded to reproduce the original digital data. Modems can be used with any means of transmitting analog signals, from light emitting diodes to radio. The most familiar example is a voice band modem that turns the digital data of a personal computer into modulated electrical signals in the voice frequency range of a telephone channel. These signals can be transmitted over telephone lines and demodulated by another modem at the receiver side to recover the digital data."

2.2.2.2 Platform Software

System Administrators, Software Engineers, System Operators and Field Service Personnel use the following Local and optional Remote Host Computer software components.

2.2.2.2.1 Operating System(s)

System Administrators, Software Engineers, System Operators and Field Service Personnel use the following Local and optional Remote Host Computer Operating System components.

- **Device Driver(s)** (on page 44)
- **Run Time Libraries** (on page 45)
- **File System Manager(s)** (on page 45)
- **Terminal Emulator(s)** (on page 46)
- **Command Line Shell(s)** (on page 47)

From Wikipedia, the free encyclopedia:

"An operating system (OS) is a collection of software that manages computer hardware resources and provides common services for computer programs. The operating system is an essential component of the system software in a computer system. Application programs usually require an operating system to function.

Time-sharing operating systems schedule tasks for efficient use of the system and may also include accounting software for cost allocation of processor time, mass storage, printing, and other resources.

For hardware functions such as input and output and memory allocation, the operating system acts as an intermediary between programs and the computer hardware,[1][2] although the application code is usually executed directly by the hardware and will frequently make a system call to an OS function or be interrupted by it. Operating systems can be found on almost any device that contains a computer—from cellular phones and video game consoles to supercomputers and web servers.

Examples of popular modern operating systems include Android, BSD, iOS, Linux, OS X, QNX, Microsoft Windows,[3] Windows Phone, and IBM z/OS. All these, except Windows, Windows Phone and z/OS, share roots in UNIX.

- **Real-time** - A real-time operating system is a multitasking operating system that aims at executing real-time applications. Real-time operating systems often use specialized scheduling algorithms so that they can achieve a deterministic nature of behavior. The main objective of real-time operating systems is their quick and predictable response to events. They have an event-driven or time-sharing design and often aspects of both. An event-driven system switches between tasks based on their priorities or external events while time-sharing operating systems switch tasks based on clock interrupts.
- **Multi-user** - A multi-user operating system allows multiple users to access a computer system at the same time. Time-sharing systems and Internet servers can be classified as multi-user systems as they enable multiple-user access to a computer through the sharing of time. Single-user operating systems have only one user but may allow multiple programs to run at the same time.

- **Multi-tasking vs. Single-tasking** - A multi-tasking operating system allows more than one program to be running at the same time, from the point of view of human time scales. A single-tasking system has only one running program. Multi-tasking can be of two types: pre-emptive and co-operative. In pre-emptive multitasking, the operating system slices the CPU time and dedicates one slot to each of the programs. Unix-like operating systems such as Solaris and Linux support pre-emptive multitasking, as does AmigaOS. Cooperative multitasking is achieved by relying on each process to give time to the other processes in a defined manner. 16-bit versions of Microsoft Windows used cooperative multi-tasking. 32-bit versions of both Windows NT and Win9x, used pre-emptive multi-tasking. Mac OS prior to OS X used to support cooperative multitasking.
- **Distributed** - A distributed operating system manages a group of independent computers and makes them appear to be a single computer. The development of networked computers that could be linked and communicate with each other gave rise to distributed computing. Distributed computations are carried out on more than one machine. When computers in a group work in cooperation, they make a distributed system.
- **Embedded** - Embedded operating systems are designed to be used in embedded computer systems. They are designed to operate on small machines like PDAs with less autonomy. They are able to operate with a limited number of resources. They are very compact and extremely efficient by design. Windows CE and Minix 3 are some examples of embedded operating systems."

2.2.2.2.1.1 Device Driver(s)

From Wikipedia, the free encyclopedia:

"In computing, a device driver is a computer program that operates or controls a particular type of device that is attached to a computer.[1] A driver typically communicates with the device through the computer bus or communications subsystem to which the hardware connects. When a calling program invokes a routine in the driver, the driver issues commands to the device. Once the device sends data back to the driver, the driver may invoke routines in the original calling program. Drivers are hardware-dependent and operating-system-specific. They usually provide the interrupt handling required for any necessary asynchronous time-dependent hardware interface.

A device driver simplifies programming by acting as translator between a hardware device and the applications or operating systems that use it.[1] Programmers can write the higher-level application code independently of whatever specific hardware the end-user is using. Physical layers communicate with specific device instances. For example, a serial port needs to handle standard communication protocols such as XON/XOFF that are common for all serial port hardware. This would be managed by a serial port logical layer. However, the physical layer needs to communicate with a particular serial port chip. 16550 UART hardware differs from PL-011. The physical layer addresses these chip-specific variations. Conventionally, OS requests go to the logical layer first. In turn, the logical layer calls upon the physical layer to implement OS requests in terms understandable by the hardware. Conversely, when a hardware device needs to respond to the OS, it uses the physical layer to speak to the logical layer.

In Linux environments, programmers can build device drivers either as parts of the kernel or separately as loadable modules. Makedev includes a list of the devices in Linux: ttyS (terminal), lp (parallel port), hd (disk), loop (loopback disk device), sound (these include mixer, sequencer, dsp, and audio)...[3]

The Microsoft Windows .sys files and Linux .ko modules contain loadable device drivers. The advantage of loadable device drivers is that they can be loaded only when necessary and then unloaded, thus saving kernel memory."

2.2.2.2.1.2 Run Time Libraries

From Wikipedia, the free encyclopedia:

"In computer programming, a runtime library is the API used by a compiler to invoke some of the behaviors of a runtime system. The runtime system implements the execution model and other fundamental behaviors of a programming language. The compiler inserts calls to the runtime library into the executable binary. During execution (run time) of that computer program, execution of those calls to the runtime library cause communication between the application and the runtime system. This often includes functions for input and output, or for memory management.

The runtime library may implement a portion of the runtime system's behavior, but if one reads the code of the calls available, they typically are thin wrappers that simply package information and send it to the runtime system. However, sometimes the term runtime library is meant to include the code of the runtime system itself, even though much of that code cannot be directly reached via a library call.

For example, some language features that can be performed only (or are more efficient or accurate) at runtime are implemented in the runtime system and may be invoked via the runtime library API, e.g. some logic errors, array bounds checking, dynamic type checking, exception handling and possibly debugging functionality. For this reason, some programming bugs are not discovered until the program is tested in a "live" environment with real data, despite sophisticated compile-time checking and pre-release testing. In this case, the end user may encounter a runtime error message.

The concept of a runtime library should not be confused with an ordinary program library like that created by an application programmer or delivered by a third party, nor with a dynamic library, meaning a program library linked at run time."

2.2.2.2.1.3 File System Manager(s)

From Wikipedia, the free encyclopedia:

"A file manager or file browser is a computer program that provides a user interface to work with file systems. The most common operations performed on files or groups of files are: create, open, edit, view, print, play, rename, move, copy, delete, search/find, and modify file attributes, properties and file permissions. Files are typically displayed in a hierarchy. Some file managers contain features inspired by web browsers, including forward and back navigational buttons.

Some file managers provide network connectivity via protocols, such as FTP, NFS, SMB or WebDAV. This is achieved by allowing the user to browse for a file server (connecting and accessing the server's file system like a local file system) or by providing its own full client implementations for file server protocols."

2.2.2.2.1.4 Terminal Emulator(s)

From Wikipedia, the free encyclopedia:

"A terminal emulator, terminal application, term, or tty for short, is a program that emulates a video terminal within some other display architecture. Though typically synonymous with a command line shell or text terminal, the term terminal covers all remote terminals, including graphical interfaces. A terminal emulator inside a graphical user interface is often called a terminal window.

A terminal window allows the user access to a text terminal and all its applications such as command line interfaces (CLI) and text user interface applications. These may be running either on the same machine or on a different one via telnet, ssh, or dial-up. On Unix-like operating systems, it is common to have one or more terminal windows connected to the local machine.

Terminals usually support a set of escape sequences for controlling color, cursor position, etc. Examples include the family of terminal control sequence standards known as ECMA-48, ANSI X3.64 or ISO/IEC 6429.

Early adopters of computer technology, such as banks, insurance companies, and governments, still make frequent use of terminal emulators. They typically have decades-old applications running on mainframe computers. The old "dumb" video terminals used to access the mainframe are long since obsolete; however, applications on the mainframe are still in use. Quite often, terminal emulators are the only way to access applications running on these older machines."

2.2.2.3 Common Terminal Emulators

- vt100 and vt220 (non-color devices that do NOT support a mouse)
- cygwin mintty, xterm, xterm-color, xterm-16color, xterm-88color and xterm-256color (multi-color devices that support a mouse)

2.2.2.3.1.1 Command Line Shell(s)

From Wikipedia, the free encyclopedia:

"A shell in computing provides a user interface for access to an operating system's kernel services. "Shell" is also used loosely to describe applications, including software that is "built around" a particular component, such as web browsers and email clients that are, in themselves, "shells" for HTML rendering engines. The term "shell" in computing, being the outer layer between the user and the operating system kernel, is synonymous with the general word "shell".

Generally, operating system shells use either a command-line interface (CLI) or graphical user interface (GUI). Mac OS xxx and Windows xxx are widely used operating systems with GUIs.[1][2][3]

The optimum choice of user interface depends on a computer's role and particular operation. CLIs allow some operations to be performed faster, rearranging large blocks of data for example. CLIs may be best for servers which are managed by experts: administrators, while GUIs offer simplicity and ease-of-use and would be more appropriate for image editing, CADD, and desktop publishing. In practice, many systems provide both user interfaces which can be called on a command-by-command basis. Windows xxx is the most obvious example with its "command prompt" and normal "windows" mode. It's no exaggeration to say that both Apple Macintosh OS xxx and Microsoft Windows xxx have revolutionised home computing by helping relatively inexperienced users to interface with a PC using a GUI.

In expert systems, a shell is a piece of software that is an "empty" expert system without the knowledge base for any particular application.[4]"

2.2.2.4 Common Shells

- bash (Bourne Again Shell)
- csh (C-Shell)
- sh (Bourne Shell)

2.2.2.4.1 Software Development Tools

System Administrators, Software Engineers, System Operators and Field Service Personnel use the following Local and optional Remote Host Computer Software Development Tool components.

- *Python Programming Language* (on page 49)
 - *Source Code Editors* (on page 50)
 - *Program Debuggers* (on page 51)
-

From Wikipedia, the free encyclopedia:

"A programming tool or software development tool is a program or application that software developers use to create, debug, maintain, or otherwise support other programs and applications. The term usually refers to relatively simple programs, that can be combined together to accomplish a task, much as one might use multiple hand tools to fix a physical object.

The distinction between tools and applications is murky. For example, developers use simple databases (such as a file containing a list of important values) all the time as tools.[dubious – discuss] However a full-blown database is usually thought of as an application or software in its own right.

For many years, computer-assisted software engineering (CASE) tools were sought after. Successful tools have proven elusive.[citation needed] In one sense, CASE tools emphasized design and architecture support, such as for UML. But the most successful of these tools are IDEs.

The ability to use a variety of tools productively is one hallmark of a skilled software engineer."

2.2.2.4.1.1 Python Programming Language

From Wikipedia, the free encyclopedia:

"Python is a widely used general-purpose, high-level programming language.[13][14][15] 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.[16][17] The language provides constructs intended to enable clear programs on both a small and large scale.[18]

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.[19]

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).[20] 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."

NOTES:

1. For those Operating System distributions that don't include Python, various release versions can be obtained directly from the Python Software Foundation at www.python.org.
 2. Many Linux and Unix Operating System distributions include a recent, but not the necessarily the latest, Python 2.x release. However, since the Operating System distribution may itself use a specific Python release, it is often necessary to follow special installation procedures when adding newer Python releases. For example, see the "*How to install Python 2.7 and Python 3.3 on CentOS 6*" article at <http://toomuchdata.com>.
 3. Microsoft Windows distributions do NOT include any Python release and do NOT include any "Curses" or "nCurses" Terminal Control Library for use with the terminal emulators (xterm, xterm-color, xterm-16color, xterm-88color, xterm-256color, vt100 and vt220) and terminals. However, if you install Cygwin, the free Linux-like Command Line Interface and GNU Toolkit from Red Hat at <http://cygwin.com>, you can then install a recent, but not necessarily the latest Python 2.x and/or Python 3.x release and the associated "Curses" or "nCurses" Terminal Control Library.
-

2.2.2.4.1.2 Source Code Editors

From Wikipedia, the free encyclopedia:

"A source code editor is a text editor program designed specifically for editing source code of computer programs by programmers. It may be a standalone application or it may be built into an integrated development environment (IDE).

Source code editors have features specifically designed to simplify and speed up input of source code, such as syntax highlighting, autocomplete and bracket matching functionality. These editors also provide a convenient way to run a compiler, interpreter, debugger, or other program relevant for software development process. So, while many text editors can be used to edit source code, if they don't enhance, automate or ease the editing of code, they are not source code editors, but simply text editors that can also be used to edit source code.

A few source code editors check syntax while users type, immediately warning of syntax problems. A few source code editors compress source code, typically converting common keywords into single-byte tokens, removing unnecessary whitespace, and converting numbers to a binary form. Such tokenizing editors later uncompress the source code when viewing it, prettyprinting it with consistent capitalizing and spacing. A few source code editors do both."

NOTE: The "tsWxGTUI_PyVx" Toolkit author(s) prefer to use the free, cross-platform, screen-oriented "xemacs" editor which supports multiple programming languages (and replaced "lemacs") rather than GNU's "emacs" editor.

2.2.2.4.1.3 Program Debuggers

From Wikipedia, the free encyclopedia:

"A debugger or debugging tool is a computer program that is used to test and debug other programs (the "target" program). The code to be examined might alternatively be running on an instruction set simulator (ISS), a technique that allows great power in its ability to halt when specific conditions are encountered but which will typically be somewhat slower than executing the code directly on the appropriate (or the same) processor. Some debuggers offer two modes of operation—full or partial simulation—to limit this impact.

A "crash" happens when the program cannot normally continue because of a programming bug. For example, the program might have tried to use an instruction not available on the current version of the CPU or attempted to access unavailable or protected memory. When the program "crashes" or reaches a preset condition, the debugger typically shows the location in the original code if it is a source-level debugger or symbolic debugger, commonly now seen in integrated development environments. If it is a low-level debugger or a machine-language debugger it shows the line in the disassembly (unless it also has online access to the original source code and can display the appropriate section of code from the assembly or compilation).

FEATURES

Typically, debuggers offer a query processor, symbol resolver, expression interpreter, and debug support interface at its top level.[1] Debuggers also offer more sophisticated functions such as running a program step by step (single-stepping or program animation), stopping (breaking) (pausing the program to examine the current state) at some event or specified instruction by means of a breakpoint, and tracking the values of variables.[2] Some debuggers have the ability to modify program state while it is running. It may also be possible to continue execution at a different location in the program to bypass a crash or logical error.

The same functionality which makes a debugger useful for eliminating bugs allows it to be used as a software cracking tool to evade copy protection, digital rights management, and other software protection features. It often also makes it useful as a general verification tool, fault coverage, and performance analyzer, especially if instruction path lengths are shown.[3]

Most mainstream debugging engines, such as gdb and dbx, provide console-based command line interfaces. Debugger front-ends are popular extensions to debugger engines that provide IDE integration, program animation, and visualization features. Some early mainframe debuggers such as Oliver and SIMON provided this same functionality for the IBM System/360 and later operating systems, as long ago as the 1970s."

NOTE: The "tsWxGTUI_PyVx" Toolkit author(s) prefer to use Wingware's commercial, cross-platform, "WingIDE Pro" debugger for Python code rather than Python's free, standard debugger "IDLE".

2.2.3 Platform Configurations (Release Notes)

System Administrators, Software Engineers, System Operators and Field Service Personnel may use a broad range of platform hardware and software configurations. Candidate configurations include or are equivalent to the following:

Operating System Software	Add-On Software	Central Processing Unit Hardware
GNU/Linux (CentOS 7.0, Fedora 21, OpenSUSE 13.1, Scientific 6.5, Ubuntu 12.04 and 14.04 etc.)	<ul style="list-style-type: none"> ▪ Python 2.6.x ▪ Python 2.7.x ▪ Python 3.2.x ▪ Python 3.3.x ▪ Python 3.4.x 	<ul style="list-style-type: none"> ▪ Intel & AMD / PC Compatible (32-bit & 64-bit) ▪ Freescale / IBM / Motorola PowerPC Compatible (32-bit & 64-bit) ▪ IBM Cell Broadband Engine (64-bit)
Mac OS X (10.3.x Panther - 10.10.x Yosemite etc.)	<ul style="list-style-type: none"> ▪ Python 2.6.x ▪ Python 2.7.x ▪ Python 3.2.x ▪ Python 3.3.x ▪ Python 3.4.x ▪ Parallels Desktop 5 / 6 / 7 / 8 / 9 / 10 for Mac (runs various Linux, Unix and Windows Virtual Machines on Mac OS X using shared computer processor, memory, peripheral, and network resources) ▪ VMware Fusion 3 / 4 / 5 (runs various Linux, Unix 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 Compatible (32-bit & 64-bit) ▪ Freescale / IBM / Motorola PowerPC Compatible (32-bit & 64-bit)
<p>Microsoft Windows (XP, Vista, 7, 8, 8.1 etc. which require Cygwin 1.7.x-1.8x, the free and open-source, Linux-like command line interface and GNU tool-chain plug-in from Red Hat.)</p> <p>Cygwin consists of two parts:</p> <ul style="list-style-type: none"> ▪ a dynamic-link library (DLL) as an API compatibility layer providing a substantial part of the POSIX API functionality, and 	<ul style="list-style-type: none"> ▪ Python 2.6.x ▪ Python 2.7.x ▪ Python 3.2.x ▪ Python 3.3.x ▪ Python 3.4.x 	<ul style="list-style-type: none"> ▪ Intel & AMD / PC Compatible (32-bit & 64-bit)

<ul style="list-style-type: none"> ▪ an extensive collection of software tools and applications that provide a Unix-like look and feel. 		
UNIX (FreeBSD 9.2, PC-BSD 10, OpenIndiana 151a8, Solaris 11, SunOS 5.11 etc.)	<ul style="list-style-type: none"> ▪ Python 2.6.x ▪ Python 2.7.x ▪ Python 3.2.x ▪ Python 3.3.x ▪ Python 3.4.x 	<ul style="list-style-type: none"> ▪ Intel & AMD / PC Compatible (32-bit & 64-bit) ▪ Freescale PowerPC Compatible (32-bit & 64-bit) ▪ Oracle/Sun SPARC or Compatible (32-bit & 64-bit) ▪ Silicon Graphics Incorporated MIPS or Compatible (32-bit & 64-bit)

See also: **Appendix A - Baseline Host Computer Platforms** (on page 333). It describes the development and test platforms used by the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit author(s).

Having access to engineering documentation and source code empowers the "tsWxGTUI_PyVx" Toolkit recipients to "port" the Toolkit to other platforms and Python versions by:

- 1) Resolving some compatibility issues simply by modifying its source code to apply Python's "__future__" module feature.
- 2) Resolving remaining compatibility issues by the non-trivial task of re-engineering various *TeamSTARS* "tsWxGTUI_PyVx" Toolkit components.

2.2.3.1 Currently Used Platforms (Release Notes)

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

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 10.9-10.10 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 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 (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 9-10 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 7.0 64-bit, Fedora 21 64-bit, OpenSUSE 12.2 & 13.1 64-bit, Scientific 6.4 & 6.5 64-bit, Ubuntu 12.04 & 14.04 32-bit with Python 2.7 and 3.2. Microsoft Windows (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 installation 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 support mouse even with xterm.) 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"	2007-model year, 2.33 GHz Intel Core 2 Duo	Host Operating System

MacBook Pro Laptop	<p>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. 	<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*
- *Freescall*
- *Intel*
- *IBM*
- *Marvell*
- *NexGen*
- *Nvidia Tegra*
- *Oracle (previously Sun)*
- *OWC*
- *Qualcomm*
- *Rise Technology*
- *Samsung*
- *SigmaTel*
- *Texas Instruments*
- *Transmeta*
- *tilera*
- *Via (Centaur Technology division)*
- *winchip*

2.2.3.2 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

<p>Notes - Baseline Host Computer Platform Configurations previously used by "tsWxGTUI_PyVx" Toolkit developers</p>	<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 Mac 2 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 Mac 3 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 Mac 4 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-SP3 6 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>

2.3 History of System Development, Operation and Maintenance

- *Embedded Software Development Background* (on page 61)
- *Python Programming Background* (on page 62)
- *Documentation Tool Background* (on page 62)
- *Evolution of the "tsWxGTUI_PyVx" Toolkit* (on page 64)

2.3.1 Embedded Software Development Background

In 1997, Richard S. Gordon ("Dick") joined Frederick A. Kier ("Rick") in developing, enhancing, maintaining, supporting and using a family of "Scalable Test and Resource Stressor" ("STARS") system level diagnostic tools at Mercury Computer Systems, in Chelmsford, MA.

Those system-level diagnostic tools were implemented in C and performed terminal-independent input/output, on the host system, via the "curses" programming library. The system-level diagnostic tools were used to detect and troubleshoot engineering design, manufacturing and interoperability defects in Mercury's products (hardware, software, compilers and embedded commercial, industrial and military systems).

- 1 The collaboration enhanced the capabilities of testing configurations with:
 - a) A Motorola 68020 host processor running VxWorks or a SPARC host processor running SunOS/Solaris.
 - b) One or more Intel 860 reduced instruction set processors running Mercury's Multi-Computer Operating System (MCOS).
- 2 Later enhanced capabilities precipitated a change in the "STARS" acronym to "System Test and Resource Stressor". The enhanced "STARS" tested single chassis, multi-chassis, multi-rack and multi-cabinet configurations with:
 - a) An Intel x86 host running Windows, a Motorola 68020 host running VxWorks, a SPARC running SunOS/Solaris or a MIPS host running IRIX.
 - b) Homogenous configurations of a handful or many hundreds of a single type of procesor running Mercury's Multi-Computer Operating System (MCOS)
 - Intel 860
 - IBM/Motorola PowerPC
 - Analog Device SHARC
 - c) Heterogeneous configurations of a handful or many hundreds of multiple types of procesors running Mercury's Multi-Computer Operating System (MCOS)
 - Intel 860
 - IBM/Motorola PowerPC

Analog Device SHARC

Custom Field Programmable Gate Arrays (FPGA)

- 3 Co-workers and management nicknamed the duo "*TeamSTARS*" for their highly effective collaboration and troubleshooting contributions.

2.3.2 Python Programming Background

- 1 At home, after work, the "*TeamSTARS*" duo researched the topic of Python on the internet.
 - a) "Rick" experimented with Python on his personal computers:
A Dell Inspiron 7500 laptop running Microsoft Windows 2000 and XP.
An Apple Mac desktop running Mac OS X.
 - b) "Dick" experimented with Python on his personal computer:
A Dell Inspiron 7000 laptop running Microsoft Windows 2000 and XP.
 - c) Convinced of the technology's functional capabilities and substantial productivity improvements, "*TeamSTARS*" proposed Python to Mercury management and received executive-level approval for the introduction of Python components for "STARS" and for new engineering and manufacturing diagnostic applications such as "SLEUTH" and "MCPDIAG".
- 2 In late 2006, Mercury's "MCPDIAG" project became an unexpected opportunity for "*TeamSTARS*" to create a Python-based command line interface which configured and installed system and diagnostic software before running engineering design verification tests and manufacturing qualification tests on the following:
 - a) Hardware - An Intel x86 host motherboard with sixteen daughter cards and their dual IBM Cell Broadband Engines (each with a PowerPC General Processing Unit and eight Digital Signal Processing Units interconnected via Gigabit Ethernet.
 - b) Software - Host and IBM Cell Broadband Engines ran Linux. Due to schedule and manpower constraints, it was not practical to develop an application program with a Graphical User Interface. Instead, the diagnostic application used a multi-layered Command Line Interface syntax which was programmed by "*TeamSTARS*" in Python. Python's extension capability was used to interface with the hardware-specific diagnostic tests that were programmed in C by a third software engineer.
 - c) Python exceeded our expectations for its veratility, capabilities and productivity gains. We planned to adopt a modern graphical-style user interface for our diagnostic tools.

2.3.3 Documentation Tool Background

- 1 While working for:
 - a) The General Electric Company, "Dick" authored engineering documents using MultiMate and Word Perfect desktop word processing software on his home and office IBM Personal Computers.
 - b) The Radio Corporation of America, "Dick" authored engineering documents using Interleaf desktop authoring and publishing software on Hewlett Packard Apollo Worstations.

- c) Mercury Computer Systems, "Dick" authored engineering documents using Adobe FrameMaker desktop authoring and publishing software on Sun Micro System workstations.

In 1999, the "*TeamSTARS*" duo purchased Dell Inspiron 7000/7500 model laptop computers, Microsoft Windows 2000 Professional (later upgraded to XP) and Microsoft Office 97 (later upgraded to XP) software for their personal and office use. To increase their productivity, they personally purchased and installed Interleaf software on their Dell laptops

In 2002, the "*TeamSTARS*" duo researched the topic of document authoring tools and evaluated Author-It because at the time, unlike Interleaf, it did not require a costly annual subscription to obtain maintenance updates and technical support. Each subsequently purchased a licensed copy for their personal laptop use.

- 2 In 2007, the "*TeamSTARS*" duo began the "tsWxGTUI_PyVx" Toolkit documentation Authoring and Publication activities on a Windows XP Professional platform. The limited performance and upgradability of their Dell laptops propted the duo to purchase Apple MacBook Pro laptops. A third-party hypervisor software product (Parallels Desktop) that runs on an Apple MacBook Pro laptop was used to create a virtual Windows XP Professional platform that ran on the Apple laptop. Such work continues on virtual Windows 7, 8 and 8.1 Professional platforms.

From Wikipedia, the free encyclopedia:

"A hypervisor or virtual machine monitor (VMM) is a piece of computer software, firmware or hardware that creates and runs virtual machines.

A computer on which a hypervisor is running one or more virtual machines is defined as a host machine. Each virtual machine is called a guest machine. The hypervisor presents the guest operating systems with a virtual operating platform and manages the execution of the guest operating systems. Multiple instances of a variety of operating systems may share the virtualized hardware resources."

The purpose of the documentation was to establish and maintain a focus for contributors to the "tsWxGTUI_PyVx" Toolkit development project. Based on the author's prior experience, the model for the document set was based on MIL-STD-498 (See Section 6 **DEFINITIONS, ABBREVIATIONS, AND ACRONYMS** (on page 263)). Annotations were included to focus the author and reader on the topic under discussion. To minimize duplication of authoring, editing and reading effort, topics were discussed once and then re-used (with identification of the document containing the master) as needed.

- 3 The document masters and derived HTML and Microsoft Word versions were produced using Author-it version 5. The latter is an authoring tool and content management system for creating, maintaining, and distributing single-sourced content. The Microsoft Word versions contain Author-it specific markups (such as outline numbered sections, page headings, page footers and hyper-links). Any chapter, section or paragraph component authored for a hierarchical location in one document (document 1 at a.b.c) may be re-used by reference at any other hierarchical location in any other document (document 2 at d.e, document 3 at f.g.h.i.j.k). Author-it automatically re-numbers new references as appropriate for their new hierarchical location.

- 4 The derived PDF document versions (with or without such water mark "Page Tags" as "Confidential" and "Draft") were produced using FinePrint version 7.10 and pdfFactory Pro version 4.70. To capture all bit-mapped graphic images it was simply a matter of: **a)** using the print preview feature on each Microsoft Word document and selecting output to FinePrint; **b)** when FinePrint had completed its somewhat lengthy activities, output to pdfFactory Pro was selected; **c)** when pdfFactory Pro had completed its somewhat lengthy activities, output to the disk file was initiated; **d)** the published Adobe PDF files and Microsoft Office directories and files were then copied to the associated directories in the "tsDocGUI" directory.
- 5 Microsoft Word (2002 and 2013 versions) can view and edit the Author-it produced Microsoft Word document. Due to Author-it's higher level abstraction and support of relocatable shared library documents, Microsoft Word may not properly recognize and handle Microsoft Word changes to the Author-it numbered sections, paragraphs, lists and hyper-links.
- 6 Microsoft Word can also save the Author-it produced Microsoft Word document as "Plain Text", "MS-DOS Text with Layout" and "Text with Layout". In an attempt to produce the README and associated release documents that might be examined during a login session to a local or remote computer via an xterm with only a 60-80 column, character-mode display, each of the "save as" options was tried. The results invariably required extensive editing to restore the stripped out white space that had separated and indented paragraphs and list entries. Editing was also needed to fold the white space expanded lines so as to stay within the 60-80 column display margin.
- 7 Apple Computer's Mac OS X offered an office suite ("iWork" '09 which contains "Pages" 4.0, "Numbers" 2.0 and "Keynote" 5.0) which are somewhat compatible with their Microsoft Office counterparts (Word, Excel and Powerpoint). Neither the 2009 or 2013 versions of Pages" can view and edit the Author-it produced Microsoft Word document. Though they can import Microsoft Word documents, they do not properly recognize and handle even the original Author-it markup. The 2013 version does NOT recognize several of the graphic bit-mapped image types.
- 8 A third party offers users of various Linux, Mac OS X and Microsoft Windows distributions, an optional document authoring tool, LibreOffice Version 4.x. It includes a Text Document component that can view and edit the Author-it produced Microsoft Word document. Though it can import Microsoft Word documents, it does not properly recognize and handle even the original Author-it markup. However, even with its limitations, it can be used as a Word Document viewer/reader on Linux, Unix and other non-Windows platforms.

2.3.4 Evolution of the "tsWxGTUI_PyVx" Toolkit

- 1 In early 2007, a major Mercury downsizing led "TeamSTARS" to become freelance developers and co-founders of a computer software development entity, naturally known as "TeamSTARS", in order to embark on the development of a prototype of the "tsWxGTUI_PyVx" Toolkit's Command Line Interface and Graphical-style User Interface.
 - a) "Rick" and "Dick" collaborated and created the prototype toolkit ("ts") for developing Python 2.x language application programs requiring a character-mode, Unix-style Command Line Interface (CLI).
 - b) "Dick" created the prototype toolkit ("tsWx") for developing Python 2.x language application programs requiring a character-mode, "wxPython"-style, "nCurses"-based, Graphical-style User Interface (GUI).
 - c) Diverging interests precipitated the end of the "TeamSTARS" collaboration in December 2009.

- 2 In January 2010, "Dick" started "Software Gadgetry" to continue development of the "tsWxGTUI_PyVx" Toolkit and release it as free and open source code.
 - a) "Dick" re-engineered, enhanced and documented the Python 2.x language "ts" prototype components, deprecated non-essential prototype components, added numerous new components, and re-packaged the productized components for release as "tsLibCLI", "tsTestsCLI", "tsToolsCLI" and "tsUtilities".
 - b) "Dick" re-engineered, enhanced and documented the Python 2.x language "tsWx" prototype components, added new components, and packaged the productized components for release as "tsLibGUI", "tsTestsGUI" and "tsToolsGUI".
 - c) "Dick" created the Python 2.x language package distribution tools, the install tools and the "tsWxGTUI_PyVx" Toolkit Documentation.
 - d) "Dick" ported the Python 2.x language source code for use with the Python 3.x language.
- 3 "Dick" developed, conducted and debugged native host platform-specific design verification tests.
 - a) Apple Mac OS X 10.3 ("Panther") - 10.7 ("Lion"), 32-bit on 2007 model year 17" MacBook Pro Laptop
 - b) Apple Mac OS X 10.8 ("Mountain Lion") - 10.10 ("Yosemite"), 64-bit on 2013 model year 27" iMac Desktop
 - c) Ubuntu Linux 12.04, 32-bit on 1999 model year Dell Inspiron 7000 Laptop
 - d) Microsoft Windows XP Professional SPK3, 32-bit on 1999 model year Dell Inspiron 7000 Laptop
- 4 "Dick" developed, conducted and debugged Parallels Desktop 7-9 for Mac GuestOS platform-specific design verification tests (usable initially on 2007 model year 17" MacBook Pro Laptop and later with Parallels Desktop 9-10 on 2013 model year 27" iMac Desktop). Each "operating system overview" has been excerpted "From Wikipedia, the free encyclopedia":
 - a) **CentOS** (abbreviated from Community Enterprise Operating System) Linux 7.0 64-bit --- *Is a Linux distribution that attempts to provide a free, enterprise-class, community-supported computing platform which aims to be 100% binary compatible with its upstream source, Red Hat Enterprise Linux (RHEL).*
 - b) **Fedora** Linux 20, 64-bit --- *Is an operating system based on the Linux kernel, developed by the community-supported Fedora Project and owned by Red Hat. Fedora contains software distributed under a free and open source license and aims to be on the leading edge of such technologies.*
 - c) **OpenSUSE** 13.1 64-bit --- *Is a general purpose operating system built on top of the Linux kernel, developed by the community-supported openSUSE Project and sponsored by SUSE and a number of other companies.*
 - d) **Scientific** Linux 6.4, 64-bit --- *Is 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."*
 - e) **Ubuntu** Linux 12.04 & 14.04, 32-bit --- *Is a Debian-based Linux operating system, with Unity as its default desktop environment (GNOME was the previous desktop environment). It is based on free software and named after the Southern African philosophy of ubuntu (literally, "human-ness"), which often is translated as "humanity towards others" or "the belief in a universal bond of sharing that connects all humanity".*

- f) **PC-BSD** Unix 9.2, 32-bit & 10.0, 64-bit --- *Is 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 [1] as the graphical user interface. It provides official binary nVidia and Intel drivers for hardware acceleration and an optional 3D desktop interface through Kwin, and Wine is ready-to-use in running Microsoft Windows software. PC-BSD is able to run Linux software,[2] in addition to FreeBSD ports, and it has its own package management system that allows users to graphically install pre-built software packages from a single downloaded executable file, which is unique for BSD operating systems.*
- g) **OpenIndiana** (151a8 / OpenSolaris) Unix 11, 32-bit --- *Is a free and open-source, Unix operating system derived from OpenSolaris. Developers forked OpenSolaris after Oracle Corporation discontinued it,[1] in order to continue development and distribution of the source code.[2] The OpenIndiana project is stewarded by the illumos Foundation, which also stewards the illumos operating system.[2] 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".*
- h) **Microsoft Windows XP** Professional SPK3, 32-bit --- *Is a personal computer operating system produced by Microsoft as part of the Windows NT family of operating systems. The operating system was released to manufacturing on August 24, 2001, and generally released for retail sale on October 25, 2001*
- i) **Microsoft Windows 7** Professional SPK1, 32-bit --- *Is a personal computer operating system developed by Microsoft, a version of Windows NT. Development of Windows 7 occurred as early as 2006 under the codename "Blackcomb." Windows 7 was released to manufacturing on July 22, 2009,[8] and became generally available on October 22, 2009,[9] less than three years after the release of its predecessor, Windows Vista. Windows 7's server counterpart, Windows Server 2008 R2, was released at the same time.*
- j) **Microsoft Windows 8** Professional, 32-bit --- *Is a personal computer operating system developed by Microsoft as part of the Windows NT family of operating systems. Development of Windows 8 started before the release of its predecessor, Windows 7, in 2009. It was announced at CES 2011, and followed by the release of three pre-release versions from September 2011 to May 2012. The operating system was released to manufacturing on August 1, 2012, and was released for general availability on October 26, 2012*
- k) **Microsoft Windows 8.1** Professional, 32-bit --- *Is a version of the Windows NT operating system and an upgrade for Windows 8. First unveiled and released as a public beta in June 2013, it was released to manufacturing on August 27, 2013, and reached general availability on October 17, 2013, almost a year after the retail release of its predecessor. Windows 8.1 is available free of charge for retail copies of Windows 8 and Windows RT users via Windows Store. Unlike service packs on previous versions of Windows, users who obtained 8 outside of retail copies or pre-loaded installations (i.e., volume licensing) must obtain 8.1 through new installation media from their respective subscription or enterprise channel. Microsoft's support lifecycle policy treats Windows 8.1 similar to previous service packs of Windows: It is part of Windows 8's support lifecycle, and installing 8.1 is required to maintain access to support and Windows updates after January 12, 2016. However, unlike previous service packs, Windows 8.1 cannot be acquired via Windows Update and only accepts 8.1-specific product keys.*

3 OPERATOR INTERFACE

- *Command Line Interface (CLI)* (on page 67)
- *Graphical User Interface (GUI)* (on page 123)

3.1 Command Line Interface (CLI)

Humans began interacting with computers via a mechanical terminal device, such as a teletype, and the computer's Command Line Interface (CLI). The user typically interacts with the computer's Operating System via a component commonly known as a *Shell* (see "*POSIX-style Bash Shell*" on page 306). When the user issues a command, the shell connects the user with the user designated Operating System Service, Application Program or Tool.

- 1 *Customizable CLI Features* (on page 67)
- 2 *Command Line Input Syntax* (on page 73)
- 3 *Command Line Input Parser* (on page 74)
- 4 *Command Line Output Examples* (on page 78)
- 5 *Log File Examples* (on page 89)

3.1.1 Customizable CLI Features

The "tsWxGTUI_PyVx" Toolkit provides its "tsCxGlobal.py" module to establish configuration constants and macro-type functions for the Command Line Interface mode. A Software Engineer can use the module as a template to derive a customized version more suited to the needs of a specific business or product family.

Capabilities

- 1 Provide 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.

- a) Product Identification

ProductName = TeamSTARS "tsWxGTUI_PyVx" Toolkit

SubSystemName = "tsToolkitCLI"

VendorName = 'Richard S. Gordon, a.k.a. Software Gadgetry'

ThemeDate = __date__

b) Log File Identification & Attributes

```
LOG_PATH = "./<application launch directory>"
```

```
LOG_NAME = "<application name>"
```

```
LOG_EXTENSION = ".log"
```

```
APPEND = "a"
```

```
TRUNCATE = "w"
```

```
DEFAULT_LOG_FILE_MODE = TRUNCATE
```

c) Logger Standard Targets

```
"StandardOutputFile" = " "
```

```
"StandardOutputDevice" = "stdout"
```

```
"StandardErrorDevice" = "stderr"
```

```
"StandardScreenDevice" = "stdscr"
```

```
"SystemLogDevice" = "syslog"
```

d) Logger Severity Levels

```
NOTSET = 0 # From Python Logging module
```

```
PRIVATE = NOTSET + 1 # Introduced by tsLogger package
```

```
DEBUG = 10 # From Python Logging module
```

```
INFO = 20 # From Python Logging module
```

```
NOTICE = INFO + 5 # Introduced by tsLogger package
```

```
WARNING = 30 # From Python Logging module
```

```
ALERT = WARNING + 5 # Introduced by tsLogger package
```

```
ERROR = 40 # From Python Logging module
```

```
CRITICAL = 50 # From Python Logging module
```

```
EMERGENCY = CRITICAL + 5 # Introduced by tsLogger package
```

e) Console Shell Screen Size

```
nColumns = <Width in Characters>
```

```
nRows = <Height in Characters>
```

2 Provide a theme-based mechanism for modifying/restoring those configuration constants as appropriate for the following users and their activities:

a) Supervisory Control and Data Acquisition (SCADA) System:

System Operator

Software Engineer

System Administrator

Field Service

b) Application ("tsWxGTUI_PyVx" CLI/GUI) Development System:

Software Engineer

System Administrator

Field Service

c) Toolkit ("tsWxGTUI_PyVx") Development System:

Software Engineer

System Administrator

Field Service

3 Provide text that highlights the Terms & Conditions notices to be displayed in Splash Screens of various sizes:

Large (60+ Col x 44+ Row)	Medium (60+ Col x 34+Row)	Small (60+ Col x 7+ Row)
<ul style="list-style-type: none"> theMasthead theCopyright theLicense 	<ul style="list-style-type: none"> theCopyright theLicense 	<ul style="list-style-type: none"> theNotices
<pre> theMasthead = ''' +----+----+ TeamSTARS "%s" Toolkit ts Wx with Python-based +----+----+ Command Line Interface (CLI) G T U I and "wxPython"-style, "Curses"-based +-----+ Graphical-Text User Interface (GUI) Get that cross-platform, pixel-mode "wxPython" feeling on character-mode color (xterm-family) & non-color (vt100- family) terminals. ''' % tsWxGTUI_PyVx </pre>		
<pre> theCopyright = ''' %s, v0.0.0 (pre-alpha build 01/13/2015) Author(s): Richard S. Gordon & Frederick A. Kier Copyright (c) 2007-2009 Frederick A. Kier & Richard S. Gordon, a.k.a TeamSTARS. All rights reserved. Copyright (c) 2010-2015 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 Each third-party component is subject to its copyright holder's designated copyright and license notices. ''' % tsWxGTUI_PyVx </pre>		

```
theLicense = '''
The "%s" Toolkit and its third-party components
are distributed as free and open source software. You may
use, modify and redistribute it only under the terms and
conditions set forth in the "COPYRIGHT.txt", "CREDITS.txt"
and "LICENSE.txt" files located in the directory
"./%s/Documents/tsDistributors".
```

```

The "%s" Toolkit and its 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.
''' % (tsWxGTUI_PyVx, 'tsWxGTUI_PyVx', tsWxGTUI_PyVx)
```

```
theNotices = '''
The Terms & Conditions which permit YOUR use, modification
and redistribution of the "%s" Toolkit may be
found in the "NOTICES.txt" file located in the directory
"./%s/Documents/tsDistributors".
''' % (tsWxGTUI_PyVx, 'tsWxGTUI_PyVx')
```

Draft

Feature	Excerpt Sample Theme Definition for "tsWxGTUI_PyVx" Toolkit Software Engineer
Common Feature Set	<pre> ThemeCxPython = { 'ProductName': ProductName, 'SubSystemName': SubSystemName, 'VendorName': VendorName, 'ThemeName': 'ThemeCxPython', 'ThemeDate': ThemeDate, 'name': __title__, 'tsLoggerLevels': { 'NotSet': NOTSET, # From Python Logging module 'Private': PRIVATE, # Introduced by tsLogger package 'Debug': DEBUG, # From Python Logging module 'Info': INFO, # From Python Logging module 'Notice': NOTICE, # Introduced by tsLogger package 'Warning': WARNING, # From Python Logging module 'Alert': ALERT, # Introduced by tsLogger package 'Error': ERROR, # From Python Logging module 'Critical': CRITICAL, # From Python Logging module 'Emergency': EMERGENCY, # Introduced by tsLogger package 'name': 'tsLoggerLevels' }, 'tsLoggerStandardTargets': { 'StandardOutputFile': '', 'StandardOutputDevice': 'stdout', 'StandardErrorDevice': 'stderr', 'StandardScreenDevice': 'stdscr', 'SystemLogDevice': 'syslog', 'name': 'tsLoggerStandardTargets' }, 'tsConsoleDisplaySize': { 'Cols': sizex, 'Rows': sizey, 'name': 'tsConsoleDisplaySize' } } </pre>
User Specific Feature Set	<pre> Theme_Toolkit_Engineer = copy.copy(ThemeCxPython) Theme_Toolkit_Engineer['ThemeUser'] = 'Theme_Toolkit_Engineer' Theme_Toolkit_Engineer['Troubleshooting'] = { 'Debug_CLI_Configuration': True, 'Debug_CLI_Exceptions': True, 'Debug_CLI_Launch': True, 'Debug_CLI_Progress': True, 'Debug_CLI_Termination': True, 'Debug_GUI_Configuration': True, 'Debug_GUI_Exceptions': True, 'Debug_GUI_Launch': True, 'Debug_GUI_Progress': True, </pre>

Feature	Excerpt Sample Theme Definition for "tsWxGTUI_PyVx" Toolkit Software Engineer
	<pre> 'Debug_GUI_Termination': True, 'name': 'Troubleshooting' } Theme_Toolkit_Engineer['tsLoggerThresholds'] = { 'StandardOutputFile': DEBUG, 'StandardOutputDevice': DEBUG, # WARNING, 'StandardErrorDevice': DEBUG, # ERROR, 'StandardScreenDevice': DEBUG, # INFO, 'SystemLogDevice': ERROR, 'name': 'tsLoggerThresholds' } </pre>
Theme Activation	ThemeToUse = Theme_Toolkit_Engineer

Limitations:

- 1 There are two copies of the "tsCxGlobals.py" file. The shared one must be just above the tsLibCLI directory. As a backup, it is advisable to also maintain a reference copy of it in ["tsCxGlobalsPkg/src"].
- 2 The "tsToolkitCLI" and the "tsCxGlobals" module are designed to launch and support Python executable Command Line Interface (CLI) applications, tools and tests via the "tsCommandLineEnv" module.
- 3 The "tsToolkitGUI" and the "tsWxGlobals" module are designed to launch and support Python executable Graphical-style User Interface (GUI) applications, tools and tests via the "tsWxMultiFrameEnv" module. The launch module is a customized version of "tsCommandLineEnv". After launching the Python application, it manages the startup, operation and shutdown of the "wxPython"-style, "nCurses"-based GUI emulation.

Notes:

- 1 The "tsToolkitCLI" is designed to be independent of the "tsToolkitGUI". However, its "tsApplication" module, collects and distributes all the keyword-value pair options and positional arguments entered by the operator and those launch parameters established by the application itself. This independence ensures that there is a common launch mechanism ("tsApplication") for both CLI and GUI applications.
- 2 The "tsCxGlobals" module establishes a common area for defining system-wide CLI configuration parameters whose values may be then communicated within and across a distributed system.

Themes are customized sets of configuration parameters that are appropriate for a desired look and feel (there are separate themes for System Administrator, Software Engineer, System Operator and Field Service personnel).

- a) The System Administrator expects the display to show a chronology of only messages related to software installation and troubleshooting.
- b) The Software Engineer expects the display to show a chronology of all messages related to software design verification, quality assurance and troubleshooting.
- c) The System Operator expects the display to show a chronology of only messages related to the operator's monitoring and control activities.

- d) Field Service personnel expect the display to show only messages related to capturing the chronology of hardware and software malfunctions.

3.1.2 Command Line Input Syntax

The syntax of user input may have any of the following variations:

- 1 <name of Operating System Service Command, Application Program or Tool>
- 2 <name of Operating System Service Command, Application Program or Tool> <positional argument list>
- 3 <name of Operating System Service Command, Application Program or Tool> <keyword option list>
- 4 <name of Operating System Service Command, Application Program or Tool> <keyword option list> <positional argument list>

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 the text of a command. Upon receiving the user's input, the operating system shell, or application program if appropriate, would examine, parse and validate or reject the user's input (using the *tsOperatorSettingsParser* (see "*Command Line Input Parser*" on page 74) or an equivalent).

The operating system shell, or application program if appropriate, in responding to:

- An invalid command, would print out the help message, followed by a prompt for the next command.
- A valid command, would print out the appropriate data followed by a prompt for the next command.

3.1.3 Command Line Input Parser

Excerpt From Wikipedia, the free encyclopedia:

"Different Command-line argument parsing methods are used by different programming languages to parse command-line arguments.

Programming languages

C

C uses `argv` to process command-line arguments.[1]

Java

An example of Java argument parsing would be:

```
public class Echo {  
    public static void main (String[] args) {  
        for (String s: args) {  
            System.out.println(s);  
        }  
    }  
}
```

Perl

Perl uses `$ARGV`.

Python

Python uses `sys.argv`, e.g.:

```
import sys  
for arg in sys.argv:  
    print arg
```

Python also has a module called `argparse` in the standard library for parsing command-line arguments.[5]

3.1.3.1 tsOperatorSettingsParser

The "tsWxGTUI_PyVx" Toolkit provides its "tsOperatorSettingsParser" module as the default for parsing the command line input entered by the operator of an application program. A Software Engineer can use the module as a template to derive a customized version more suited to the specific needs of each application.

When used with Python 2.6-2.7 or Python 3.1-3.4, the "tsOperatorSettings.py" module (a "tsLibCLI" component of the "tsWxGTUI_PyVx" Toolkit) supports the available Python version specific parser module(s) as an experimental and educational opportunity.

However, when one seeks to backport applications to Python 2.0-2.5 or Python 3.0, the "tsOperatorSettings.py" module ignores unsupported Python version specific parser modules in order to prevent a program trap which will block the application from running.

Parsing extracts the Keyword-Value pair Options and Positional Arguments that will configure and control the application during its execution. It validates the syntax of the input and generates the on-line help output when requested or appropriate for error recovery.

- 1** 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)
- 2** Provides keyword-value pair definitions needed for debugging non-application-specific Command Line and Graphical-style User Interfaces:
 - a) -h, --help
 - b) -v, --version
 - c) -a, --about
 - d) -d, --debug
 - e) -V, --Verbose
- 3** Provides positional argument definitions needed for debugging Python version-specific library modules:
 - a) module {choice of "argparse", "optparse", "getopt"}

Draft


```
usageHelp = '''
```

Summary

Interpreter	program app.	operator settings
-----	-----	-----
python	%prog	
python2.7	%prog	[-h] [-v] [-a] [-d] [-V]
python3.3	%prog	[--help] [--version] [--about] [--debug] [--Verbose] {argparse, optparse, getopt}

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 and positional arguments

Details

```
python %prog \\  
  
    [-h | --help] \\  
    [-v | --version] \\  
    [-a | --about] \\  
    [-d | --debug] \\  
    [-V | --Verbose] \\  
  
    {argparse, optparse, getopt}
```

positional arguments:

```
{argparse, optparse, getopt}  
    command line parser module is Python version dependent  
    unless the operator uses a positional argument to  
    designate one of the following choices: {"argparse",  
    "optparse", "getopt"}
```

optional arguments:

-h, --help	show this help message and exit
-v, --version	exit after displaying program title, version and date (default = False)
-a, --about	exit after displaying the software release identification (build title, version and date) and applicable usage terms and conditions (stipulated by primary and third-party author(s), copyright(s) and

-d, --debug	license(s) . log/display application program progress and diagnostic messages useful in debugging and troubleshooting. (default = False).
-V, --Verbose	enable display/log of verbose troubleshooting details for application program activity tracking diagnostic messages (default = False)

3.1.3.2 tsOperatorSettingsBuilder (Future)

Class to enable software developers to define an application-specific set of command line keyword-value pair options and positional arguments, in the conventional GNU/POSIX syntax. Those application-specific definitions are then translated, by the builder, into the executable format(s) appropriate for installation on a set of Python-versions. An operator may then enter valid combinations of keyword-value pair options and positional arguments when launching the installed application.

3.1.4 Command Line Output Examples

1 Debug_CLI_Launch (See "tsCxGlobals.py"):

Debug mode enabled for Python module import activity

2 Command Line:

```
python tsLinesOfCodeProjectMetrics.py -i ./basicFileTypes/ -o basicFileTypesProjectMetrics.txt >  
basicFileTypesProjectMetrics.log
```

3.1.4.1 basicFileTypesProjectMetrics.log

When the Debug_CLI_Launch is enabled in the "tsCxGlobals.py" file, the "tsWxGTUI_PyVx" Toolkit will log activity associated with the importing of Python modules. The activity is recorded by redirecting console output a log file as recorded below. Once a module has been imported, subsequent duplicated import information is deleted.

```

init in tsLibCLI
  tsLibCLI = /cygdrive/d/WR/SoftwareGadgetry-Dev/Python-2x/tsWxGTUI/tsLibCLI
  Importing tsLibCLI.tsCommandLineInterfacePkg.src
    init in tsCommandLineInterfacePkg
      Importing
tsLibCLI.tsCommandLineInterfacePkg.src.tsCommandLineInterface
  Importing tsLibCLI.tsConfigObjectPkg.src
    init in tsConfigObjectPkg
      Importing tsLibCLI.tsConfigObjectPkg.src.configobj
      Importing tsLibCLI.tsConfigObjectPkg.src.validate
      Importing tsLibCLI.tsConfigObjectPkg.src.tsConfig
  Importing tsLibCLI.tsDecoratorPkg.src
    init in tsDecoratorPkg
      Importing tsLibCLI.tsDecoratorPkg.src.decorator
      Importing tsLibCLI.tsDecoratorPkg.src.tsDecorators
  Importing tsLibCLI.tsExceptionPkg.src
    init in tsExceptionPkg
      Importing tsLibCLI.tsExceptionPkg.src.tsExceptions
  Importing tsLibCLI.tsReportUtilityPkg.src
    init in tsReportUtilityPkg
      Importing tsLibCLI.tsReportUtilityPkg.src.tsReportUtilities
  Importing tsLibCLI.tsLoggerPkg.src
    init in tsLoggerPkg
      Importing tsLibCLI.tsLoggerPkg.src.tsLogger
<open file './logs/2013-12-13-at-07-33-24/tsLinesOfCodeProjectMetrics.log',
mode 'w' at 0x7fd30288>
  Importing tsLibCLI.tsDoubleLinkedListPkg.src
    init in tsDoubleLinkedListPkg
      Importing tsLibCLI.tsDoubleLinkedListPkg.src.tsDoubleLinkedList
  Importing tsLibCLI.tsPlatformRunTimeEnvironmentPkg.src
    init in tsPlatformRunTimeEnvironmentPkg
      Importing
tsLibCLI.tsPlatformRunTimeEnvironmentPkg.src.tsGistGetTerminalSize
      Importing
tsLibCLI.tsPlatformRunTimeEnvironmentPkg.src.tsPlatformRunTimeEnvironment
      Importing tsLibCLI.tsApplicationPkg.src
        init in tsApplicationPkg
          Importing tsLibCLI.tsApplicationPkg.src.tsApplication
      Importing tsLibCLI.tsCommandLineEnvPkg.src
        init in tsCommandLineEnvPkg
          Importing tsLibCLI.tsCommandLineEnvPkg.src.tsCommandLineEnv
tsCommandLineEnv: ImportError (tsLibCLI); importCode=No module named
tsOperatorSettingsParser
  Importing tsLibCLI.tsOperatorSettingsParserPkg.src
    init in tsOperatorSettingsParserPkg

```

```
    Importing
tsLibCLI.tsOperatorSettingsParserPkg.src.tsOperatorSettingsParser
    Importing tsLibCLI.tsSysCommandsPkg.src
        init in tsSysCommandsPkg
            Importing tsLibCLI.tsSysCommandsPkg.src.tsSysCommands
    Importing tsLibCLI.tsThreadPoolPkg.src
        init in tsThreadPoolPkg
            Importing tsLibCLI.tsThreadPoolPkg.src.threadpool
            Importing tsLibCLI.tsThreadPoolPkg.src.tsThreadPool
Begin Cleanup tsLibCLI.tsThreadPoolPkg.src
    Deleting tsLibCLI.tsPlatformRunTimeEnvironmentPkg
    Deleting tsLibCLI.tsLoggerPkg.src.thread
    Deleting tsLibCLI.tsOperatorSettingsParserPkg.src.tsCommandLineEnv
    Deleting tsLibCLI.tsCommandLineEnvPkg.src.tsLogger
    Deleting tsLibCLI.tsExceptionPkg.src
    Deleting tsLibCLI.tsConfigObjectPkg
    Deleting tsLibCLI.tsCommandLineInterfacePkg.src.os
    Deleting tsLibCLI.tsDecoratorPkg.src.imp
    Deleting tsLibCLI.sys
    Deleting tsLibCLI.tsDoubleLinkedListPkg.src
    Deleting tsLibCLI.tsSysCommandsPkg.src.subprocess
    Deleting tsLibCLI.tsDecoratorPkg.src.sys
    Deleting tsLibCLI.tsLoggerPkg.src.imp
    Deleting tsLibCLI.tsPlatformRunTimeEnvironmentPkg.src.sys
    Deleting tsLibCLI.tsCommandLineEnvPkg
    Deleting tsLibCLI.tsPlatformRunTimeEnvironmentPkg.src.socket
    Deleting tsLibCLI.tsLoggerPkg.src.threading
    Deleting tsLibCLI.tsPlatformRunTimeEnvironmentPkg.src.optparse
    Deleting tsLibCLI.tsApplicationPkg.src.tsExceptions
    Deleting tsLibCLI.tsCommandLineEnvPkg.src.imp
    Deleting tsLibCLI.tsThreadPoolPkg.src.imp
    Deleting tsLibCLI.tsApplicationPkg.src.imp
    Deleting tsLibCLI.tsCommandLineInterfacePkg
    Deleting tsLibCLI.imp
    Deleting tsLibCLI.tsOperatorSettingsParserPkg.src
    Deleting tsLibCLI.tsThreadPoolPkg.src.tsReportUtilities
    Deleting tsLibCLI.tsLoggerPkg.src.errno
    Deleting tsLibCLI.tsOperatorSettingsParserPkg.src.platform
    Deleting tsLibCLI.tsDecoratorPkg.src.tsCxGlobals
    Deleting tsLibCLI.tsLoggerPkg.src.tsReportUtilities
    Deleting tsLibCLI.tsCommandLineEnvPkg.src.tsCxGlobals
    Deleting tsLibCLI.tsReportUtilityPkg.src.glob
    Deleting tsLibCLI.tsSysCommandsPkg
    Deleting tsLibCLI.tsSysCommandsPkg.src.tsLogger
    Deleting tsLibCLI.tsThreadPoolPkg.src.tsCxGlobals
    Deleting tsLibCLI.tsThreadPoolPkg.src.traceback
    Deleting tsLibCLI.tsExceptionPkg.src.os
    Deleting tsLibCLI.tsSysCommandsPkg.src.sys
    Deleting tsLibCLI.tsThreadPoolPkg.src.Queue
    Deleting tsLibCLI.tsOperatorSettingsParserPkg.src.string
    Deleting tsLibCLI.tsApplicationPkg.src.time
    Deleting tsLibCLI.tsConfigObjectPkg.src.configobj
    Deleting tsLibCLI.tsConfigObjectPkg.src.tsCxGlobals
    Deleting tsLibCLI.tsApplicationPkg.src.tsCxGlobals
    Deleting tsLibCLI.tsApplicationPkg
    Deleting tsLibCLI.tsLoggerPkg
```

```
Deleting tsLibCLI.tsConfigObjectPkg.src.codecs
Deleting tsLibCLI.tsSysCommandsPkg.src.tsCxGlobals
Deleting tsLibCLI.tsExceptionPkg.src.math
Deleting tsLibCLI.tsLoggerPkg.src.tsExceptions
Deleting tsLibCLI.tsPlatformRunTimeEnvironmentPkg.src.os
Deleting tsLibCLI.tsApplicationPkg.src.tsPlatformRunTimeEnvironment
Deleting tsLibCLI.tsCommandLineInterfacePkg.src.imp
Deleting tsLibCLI.tsOperatorSettingsParserPkg.src.textwrap
Deleting tsLibCLI.tsReportUtilityPkg.src.sys
Deleting tsLibCLI.tsSysCommandsPkg.src.optparse
Deleting tsLibCLI.tsCommandLineEnvPkg.src.tsLibCLI
Deleting tsLibCLI.tsPlatformRunTimeEnvironmentPkg.src.struct
Deleting tsLibCLI.tsReportUtilityPkg.src.imp
Deleting tsLibCLI.tsDoubleLinkedListPkg.src.imp
Deleting tsLibCLI.tsOperatorSettingsParserPkg.src.tsLogger
Deleting tsLibCLI.tsThreadPoolPkg.src.threading
Deleting tsLibCLI.tsThreadPoolPkg.src.tsExceptions
Deleting tsLibCLI.tsOperatorSettingsParserPkg.src.tsLibCLI
Deleting tsLibCLI.tsApplicationPkg.src.tsLibCLI
Deleting tsLibCLI.tsDecoratorPkg.src.functools
Deleting tsLibCLI.tsPlatformRunTimeEnvironmentPkg.src.textwrap
Deleting tsLibCLI.tsReportUtilityPkg.src.tsLibCLI
Deleting tsLibCLI.tsOperatorSettingsParserPkg.src.imp
Deleting tsLibCLI.tsApplicationPkg.src.sys
Deleting tsLibCLI.tsReportUtilityPkg.src
Deleting tsLibCLI.tsExceptionPkg.src.tsLibCLI
Deleting tsLibCLI.tsDecoratorPkg.src.contextlib
Deleting tsLibCLI.tsApplicationPkg.src.os
Deleting tsLibCLI.tsPlatformRunTimeEnvironmentPkg.src.tsCxGlobals
Deleting tsLibCLI.tsConfigObjectPkg.src.os
Deleting tsLibCLI.tsThreadPoolPkg.src.sys
Deleting tsLibCLI.tsSysCommandsPkg.src
Deleting tsLibCLI.tsThreadPoolPkg.src.tsApplication
Deleting tsLibCLI.tsThreadPoolPkg.src
Deleting tsLibCLI.tsThreadPoolPkg.src.tsLibCLI
Deleting tsLibCLI.tsExceptionPkg
Deleting tsLibCLI.os
Deleting tsLibCLI.tsCommandLineInterfacePkg.src.tsCxGlobals
Deleting tsLibCLI.tsDoubleLinkedListPkg.src.os
Deleting tsLibCLI.tsConfigObjectPkg.src.sys
Deleting tsLibCLI.tsConfigObjectPkg.src
Deleting tsLibCLI.tsDecoratorPkg.src
Deleting tsLibCLI.tsThreadPoolPkg.src.optparse
Deleting tsLibCLI.tsDecoratorPkg.src.linecache
Deleting tsLibCLI.tsDoubleLinkedListPkg.src.tsCxGlobals
Deleting tsLibCLI.tsConfigObjectPkg.src.re
Deleting tsLibCLI.tsPlatformRunTimeEnvironmentPkg.src.subprocess
Deleting tsLibCLI.tsLoggerPkg.src.textwrap
Deleting tsLibCLI.tsReportUtilityPkg.src.tsCxGlobals
Deleting tsLibCLI.tsCommandLineEnvPkg.src.sys
Deleting tsLibCLI.tsDoubleLinkedListPkg.src.sys
Deleting tsLibCLI.tsConfigObjectPkg.src.__future__
Deleting tsLibCLI.tsThreadPoolPkg.src.os
Deleting tsLibCLI.tsApplicationPkg.src
Deleting tsLibCLI.tsDecoratorPkg.src.os
Deleting tsLibCLI.tsLoggerPkg.src.tsCxGlobals
```

```
Deleting tsLibCLI.tsExceptionPkg.src.imp
Deleting tsLibCLI.tsPlatformRunTimeEnvironmentPkg.src.traceback
Deleting tsLibCLI.tsPlatformRunTimeEnvironmentPkg.src
Deleting tsLibCLI.tsLoggerPkg.src.tsLibCLI
Deleting tsLibCLI.tsPlatformRunTimeEnvironmentPkg.src.tsLibCLI
Deleting tsLibCLI.tsCommandLineInterfacePkg.src.sys
Deleting tsLibCLI.tsSysCommandsPkg.src.os
Deleting tsLibCLI.tsPlatformRunTimeEnvironmentPkg.src.imp
Deleting tsLibCLI.tsDecoratorPkg
Deleting tsLibCLI.tsLoggerPkg.src.sys
Deleting tsLibCLI.tsPlatformRunTimeEnvironmentPkg.src.platform
Deleting tsLibCLI.tsReportUtilityPkg.src.types
Deleting tsLibCLI.tsLoggerPkg.src
Deleting tsLibCLI.tsPlatformRunTimeEnvironmentPkg.src.shlex
Deleting tsLibCLI.tsPlatformRunTimeEnvironmentPkg.src.tsReportUtilities
Deleting tsLibCLI.tsExceptionPkg.src.traceback
Deleting tsLibCLI.tsLoggerPkg.src.syslog
Deleting tsLibCLI.tsSysCommandsPkg.src.tsLibCLI
Deleting tsLibCLI.tsCommandLineEnvPkg.src.traceback
Deleting tsLibCLI.tsDecoratorPkg.src.decorator
Deleting tsLibCLI.tsLoggerPkg.src.os
Deleting tsLibCLI.tsDecoratorPkg.src.threading
Deleting tsLibCLI.tsLoggerPkg.src.logging
Deleting tsLibCLI.tsOperatorSettingsParserPkg.src.tsCxGlobals
Deleting tsLibCLI.tsExceptionPkg.src.sys
Deleting tsLibCLI.tsThreadPoolPkg.src.time
Deleting tsLibCLI.tsDecoratorPkg.src.inspect
Deleting tsLibCLI.tsCommandLineEnvPkg.src.tsApplication
Deleting tsLibCLI.tsLoggerPkg.src.time
Deleting
tsLibCLI.tsPlatformRunTimeEnvironmentPkg.src.tsGistGetTerminalSize
Deleting tsLibCLI.tsThreadPoolPkg.src.tsLogger
Deleting tsLibCLI.tsReportUtilityPkg.src.time
Deleting tsLibCLI.tsPlatformRunTimeEnvironmentPkg.src.termios
Deleting tsLibCLI.tsPlatformRunTimeEnvironmentPkg.src.tsLogger
Deleting tsLibCLI.tsCommandLineEnvPkg.src.tsExceptions
Deleting tsLibCLI.tsConfigObjectPkg.src.imp
Deleting tsLibCLI.tsCommandLineEnvPkg.src.os
Deleting tsLibCLI.tsReportUtilityPkg.src.tsExceptions
Deleting tsLibCLI.tsSysCommandsPkg.src.tsReportUtilities
Deleting tsLibCLI.tsCommandLineInterfacePkg.src
Deleting tsLibCLI.tsReportUtilityPkg.src.os
Deleting tsLibCLI.tsReportUtilityPkg
Deleting tsLibCLI.tsSysCommandsPkg.src.tsExceptions
Deleting tsLibCLI.tsSysCommandsPkg.src.imp
Deleting tsLibCLI.tsApplicationPkg.src.tsLogger
Deleting tsLibCLI.tsDecoratorPkg.src.re
Deleting tsLibCLI.tsPlatformRunTimeEnvironmentPkg.src.tsExceptions
Deleting tsLibCLI.tsOperatorSettingsParserPkg.src.sys
Deleting tsLibCLI.tsDoubleLinkedListPkg
Deleting tsLibCLI.tsDecoratorPkg.src.itertools
Deleting tsLibCLI.tsCommandLineEnvPkg.src
Deleting tsLibCLI.tsExceptionPkg.src.tsCxGlobals
Deleting tsLibCLI.tsThreadPoolPkg.src.threadpool
Deleting tsLibCLI.tsPlatformRunTimeEnvironmentPkg.src.time
Deleting tsLibCLI.tsOperatorSettingsParserPkg.src.os
```

```

Deleting tsLibCLI.tsThreadPoolPkg
Deleting tsLibCLI.tsOperatorSettingsParserPkg
Deleting tsLibCLI.tsPlatformRunTimeEnvironmentPkg.src.fcntl
Deleting tsLibCLI.tsCxGlobals
End Cleanup tsLibCLI.tsThreadPoolPkg.src
init in tsToolsCLI
    tsToolsCLI = /cygdrive/d/WR/SoftwareGadgetry-Dev/Python-
2x/tsWxGTUI/tsToolsCLI
    Importing tsToolsCLI.tsLinesOfCodeProjectMetricsPkg.src
    init in tsLinesOfCodeProjectMetricsPkg
    Importing
tsToolsCLI.tsLinesOfCodeProjectMetricsPkg.src.tsLOCPMOperatorSettingsParser
    Importing
tsToolsCLI.tsLinesOfCodeProjectMetricsPkg.src.tsSoftwareParser
    Importing
tsToolsCLI.tsLinesOfCodeProjectMetricsPkg.src.tsSoftwareMetrics
    Importing
tsToolsCLI.tsLinesOfCodeProjectMetricsPkg.src.tsProjectMetrics
    Importing tsToolsCLI.tsLinesOfCodeProjectMetricsPkg.src.tsLinesOfCode
Importing tsToolsCLI.tsPlatformQueryPkg.src
Importing tsToolsCLI.tsStripCommentsPkg.src
    init in tsStripCommentsPkg
    Importing tsToolsCLI.tsStripCommentsPkg.src.tsStripSettingsParser
    Importing tsToolsCLI.tsStripCommentsPkg.src.tsStripComments
Importing tsToolsCLI.tsStripLineNumbersPkg.src
Importing tsToolsCLI.tsTreeCopyPkg.src
Importing tsToolsCLI.tsTreeTrimLinesPkg.src
    init in tsTreeTrimLinesPkg
    Importing tsToolsCLI.tsTreeTrimLinesPkg.src.tsTreeTrimLines
    Importing tsToolsCLI.tsTreeTrimLinesPkg.src.tsTreeTrimShutil
Begin Cleanup tsToolsCLI.tsTreeTrimLinesPkg.src
Deleting tsToolsCLI.tsStripLineNumbersPkg
Deleting tsToolsCLI.tsTreeTrimLinesPkg.src.tsToolsCLI
Deleting tsToolsCLI.tsTreeTrimLinesPkg.src.tsCxGlobals
Deleting tsToolsCLI.tsPlatformQueryPkg
Deleting tsToolsCLI.tsStripCommentsPkg.src.stat
Deleting tsToolsCLI.tsPlatformQueryPkg.src
Deleting tsToolsCLI.tsStripCommentsPkg.src.getopt
Deleting tsToolsCLI.tsCxGlobals
Deleting tsToolsCLI.tsLinesOfCodeProjectMetricsPkg.src.platform
Deleting tsToolsCLI.tsStripCommentsPkg.src.sys
Deleting tsToolsCLI.tsLinesOfCodeProjectMetricsPkg
Deleting tsToolsCLI.tsLinesOfCodeProjectMetricsPkg.src.math
Deleting tsToolsCLI.tsStripCommentsPkg.src.cStringIO
Deleting tsToolsCLI.tsStripCommentsPkg.src.tsToolsCLI
Deleting tsToolsCLI.tsStripCommentsPkg.src.grp
Deleting tsToolsCLI.tsTreeTrimLinesPkg.src.tsCommandLineEnv
Deleting tsToolsCLI.tsStripLineNumbersPkg.src
Deleting tsToolsCLI.tsStripCommentsPkg.src.tsCxGlobals
Deleting tsToolsCLI.tsStripCommentsPkg.src.optparse
Deleting tsToolsCLI.tsStripCommentsPkg.src.tsStripSettingsParser
Deleting tsToolsCLI.tsStripCommentsPkg.src.string
Deleting tsToolsCLI.tsLinesOfCodeProjectMetricsPkg.src.tsSoftwareMetrics
Deleting tsToolsCLI.tsStripCommentsPkg.src
Deleting tsToolsCLI.tsTreeTrimLinesPkg.src.imp
Deleting tsToolsCLI.tsTreeTrimLinesPkg.src.os

```

```
Deleting tsToolsCLI.tsTreeTrimLinesPkg.src.tsLibCLI
Deleting tsToolsCLI.tsLinesOfCodeProjectMetricsPkg.src.textwrap
Deleting tsToolsCLI.tsTreeCopyPkg.src
Deleting tsToolsCLI.tsLinesOfCodeProjectMetricsPkg.src.tsSoftwareParser
Deleting tsToolsCLI.tsTreeTrimLinesPkg.src
Deleting tsToolsCLI.tsLinesOfCodeProjectMetricsPkg.src.tsProjectMetrics
Deleting tsToolsCLI.tsStripCommentsPkg.src.tsLogger
Deleting tsToolsCLI.tsStripCommentsPkg.src.os
Deleting tsToolsCLI.tsStripCommentsPkg.src.tokenize
Deleting tsToolsCLI.tsStripCommentsPkg.src.tsExceptions
Deleting tsToolsCLI.tsLinesOfCodeProjectMetricsPkg.src.string
Deleting tsToolsCLI.tsStripCommentsPkg.src.fnmatch
Deleting tsToolsCLI.tsTreeCopyPkg
Deleting tsToolsCLI.tsStripCommentsPkg.src.math
Deleting tsToolsCLI.tsLinesOfCodeProjectMetricsPkg.src
Deleting tsToolsCLI.tsLinesOfCodeProjectMetricsPkg.src.tsLogger
Deleting tsToolsCLI.os
Deleting tsToolsCLI.sys
Deleting
tsToolsCLI.tsLinesOfCodeProjectMetricsPkg.src.tsLOCPMOperatorSettingsParser
Deleting tsToolsCLI.tsTreeTrimLinesPkg.src.stat
Deleting tsToolsCLI.tsLinesOfCodeProjectMetricsPkg.src.tsCommandLineEnv
Deleting tsToolsCLI.imp
Deleting tsToolsCLI.tsLinesOfCodeProjectMetricsPkg.src.sys
Deleting tsToolsCLI.tsLinesOfCodeProjectMetricsPkg.src.imp
Deleting tsToolsCLI.tsTreeTrimLinesPkg.src.optparse
Deleting tsToolsCLI.tsStripCommentsPkg.src.platform
Deleting tsToolsCLI.tsStripCommentsPkg
Deleting tsToolsCLI.tsLinesOfCodeProjectMetricsPkg.src.os
Deleting tsToolsCLI.tsStripCommentsPkg.src.collections
Deleting tsToolsCLI.tsStripCommentsPkg.src.imp
Deleting tsToolsCLI.tsTreeTrimLinesPkg.src.fnmatch
Deleting tsToolsCLI.tsTreeTrimLinesPkg.src.sys
Deleting tsToolsCLI.tsStripCommentsPkg.src.tsCommandLineEnv
Deleting tsToolsCLI.tsTreeTrimLinesPkg.src.tsExceptions
Deleting tsToolsCLI.tsStripCommentsPkg.src.tsLibCLI
Deleting tsToolsCLI.tsTreeTrimLinesPkg.src.tsLogger
Deleting tsToolsCLI.tsStripCommentsPkg.src.errno
Deleting tsToolsCLI.tsLinesOfCodeProjectMetricsPkg.src.tsCxGlobals
Deleting tsToolsCLI.tsTreeTrimLinesPkg
Deleting tsToolsCLI.tsStripCommentsPkg.src.pwd
Deleting tsToolsCLI.tsStripCommentsPkg.src.argparse
Deleting tsToolsCLI.tsStripCommentsPkg.src.textwrap
End Cleanup tsToolsCLI.tsTreeTrimLinesPkg.src
```

tsSoftwareParser, v2.5.0 (build 11/15/2013)

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.

File Extension Features of SLOCCount 2.26:
 Copyright (c) 2001-2004 David A. Wheeler.
 All rights reserved.

Algorithm Features of COCOMO(R) 81:
 Copyright (c) 1981 Dr. Barry W. Boehm.
 All rights reserved.

Python 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

 Input Path: <./basicFileTypes/>.
 Results are available in "/cygdrive/d/WR/SoftwareGadgetry-Dev/Python-
 2x/tsWxGTUI/basicFileTypesProjectMetrics.txt".

Overall Source Code Feature Statistics

	FILES	CODE	CMNTS	LINES	WORDS	CHARS
Pct:		13.35%	86.65%	100.00%		
Totals:	7	432	2805	3237	10840	310795
Std:	6	126	811	813	2849	106743
Avg:	1	62	401	463	1549	44399

Distribution of Source Code Feature Statistics by File Types

TYPES	FILES	CODE	CMNTS	LINES	WORDS	CHARS	%-LINES
.asm	1	25	40	65	252	1414	2.01%
.c	1	344	384	728	1944	16538	22.49%
.f90	1	59	58	117	422	2253	3.61%
.py	2	4	20	24	56	571	0.74%
.txt	2	0	2303	2303	8166	290019	71.15%

Definition of Source Code by File Types

TYPES	DEFINITION
.asm	Assembler Source (Intel ASM80 standard)
.c	C Source
.f90	FORTTRAN Source (90 standard)
.py	Python Script Source

.txt Plain Text (ASCII)

"Organic" Software Project Estimate
Constructive Cost Model (COCOMO(R) 81)

Total Physical Source Lines of Code (KSLOC) = 432 (0.43)

Estimated Development Effort in Person-Years (Person-Months) = 0.08
(0.99)

(Basic COCOMO model, Person-Months = $2.40 * (KSLOC^{1.05})$)

Estimated Schedule in Years (Months) = 0.21 (2.49)

(Basic COCOMO model, Months = $2.50 * (person-months^{0.38})$)

Estimated Average Number of Developers = 0.40
(Effort/Schedule)

Total Estimated Cost to Develop = \$ 11192
(Average Salary = \$ 56286.00/year, Overhead = 2.40).

Estimated Productivity in Source Lines of Code per Day = 52

Results are available in "/cygdrive/d/WR/SoftwareGadgetry-Dev/Python-
2x/tsWxGTUI/basicFileTypesProjectMetrics.txt".

No Error

3.1.4.2 basicFileTypesProjectMetrics.txt

tsLinesOfCode, v2.4.0 (build 07/31/2013)

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.

File Extension Features of SLOCCount 2.26:
 Copyright (c) 2001-2004 David A. Wheeler.
 All rights reserved.

Algorithm Features of COCOMO(R) 81:
 Copyright (c) 1981 Dr. Barry W. Boehm.
 All rights reserved.

Python 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

Individual Source Code Feature Statistics

CODE	CMNTS	LINES	WORDS	CHARS	PATH/FILE
344	384	728	1944	16538	/cygdrive/d/WR/SoftwareGadgetry-Dev/Python-2x/tsWxGTUI/basicFileTypes/GeneralCmds.c
59	58	117	422	2253	/cygdrive/d/WR/SoftwareGadgetry-Dev/Python-2x/tsWxGTUI/basicFileTypes/hello.f90
4	20	24	56	571	/cygdrive/d/WR/SoftwareGadgetry-Dev/Python-2x/tsWxGTUI/basicFileTypes/tsConfig.py
0	2216	2216	7835	286125	/cygdrive/d/WR/SoftwareGadgetry-Dev/Python-2x/tsWxGTUI/basicFileTypes/tsLinesOfCodeStatistics-Python-2.x.txt
0	87	87	331	3894	/cygdrive/d/WR/SoftwareGadgetry-Dev/Python-2x/tsWxGTUI/basicFileTypes/tsLinesOfCodeStatistics.txt
25	40	65	252	1414	/cygdrive/d/WR/SoftwareGadgetry-Dev/Python-2x/tsWxGTUI/basicFileTypes/xmitr.asm
0	0	0	0	0	/cygdrive/d/WR/SoftwareGadgetry-Dev/Python-2x/tsWxGTUI/basicFileTypes/__init__.py

Input Path: <./basicFileTypes/>.
 Results are available in "/cygdrive/d/WR/SoftwareGadgetry-Dev/Python-2x/tsWxGTUI/basicFileTypesProjectMetrics.txt".

Overall Source Code Feature Statistics

	FILES	CODE	CMNTS	LINES	WORDS	CHARS
Pct:		13.35%	86.65%	100.00%		
Totals:	7	432	2805	3237	10840	310795
Std:	6	126	811	813	2849	106743
Avg:	1	62	401	463	1549	44399

Distribution of Source Code Feature Statistics by File Types

TYPES	FILES	CODE	CMNTS	LINES	WORDS	CHARS	%-LINES
.asm	1	25	40	65	252	1414	2.01%
.c	1	344	384	728	1944	16538	22.49%
.f90	1	59	58	117	422	2253	3.61%
.py	2	4	20	24	56	571	0.74%
.txt	2	0	2303	2303	8166	290019	71.15%

Definition of Source Code by File Types

TYPES	DEFINITION
.asm	Assembler Source (Intel ASM80 standard)
.c	C Source
.f90	FORTRAN Source (90 standard)
.py	Python Script Source
.txt	Plain Text (ASCII)

"Organic" Software Project Estimate
Constructive Cost Model (COCOMO(R) 81)

Total Physical Source Lines of Code (KSLOC) = 432 (0.43)

Estimated Development Effort in Person-Years (Person-Months) = 0.08
(0.99)
(Basic COCOMO model, Person-Months = $2.40 * (KSLOC^{1.05})$)

Estimated Schedule in Years (Months) = 0.21 (2.49)
(Basic COCOMO model, Months = $2.50 * (person-months^{0.38})$)

Estimated Average Number of Developers = 0.40
(Effort/Schedule)

Total Estimated Cost to Develop = \$ 11192
(Average Salary = \$ 56286.00/year, Overhead = 2.40).

Estimated Productivity in Source Lines of Code per Day = 52

3.1.5 Log File Examples

- *Bit-Mapped Image Tree (Splash Screen Files for Graphical Applications)* (on page 90)
- *Logs Tree (Files for Non-Graphical Applications)* (on page 92)
- *Logs Tree (Files for Graphical Applications)* (on page 101)

Draft

3.1.5.1 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.

3.1.5.2 Logs Tree (Files for Non-Graphical Applications)

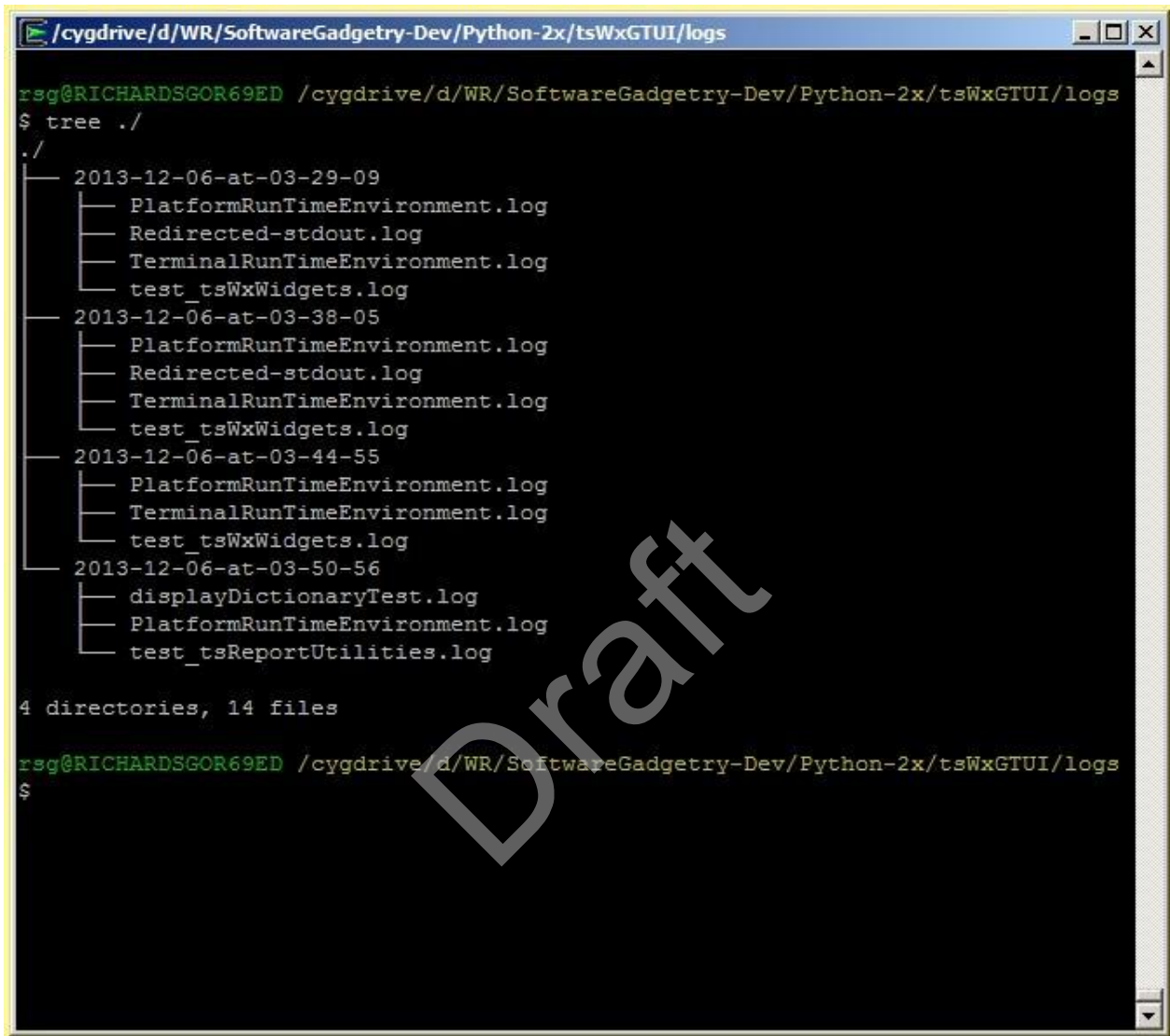
The "tsWxGTUI_PyVx" Toolkit registers, via log files, the startup, configuration and chronology of activities needed for design verification and troubleshooting.

- 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* (on page 67)).

- 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
$
```

3.1.5.2.1 2013-12-06-at-03-50-56**3.1.5.2.1.1 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 =====

3.1.5.2.1.2 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', '0x40000000000000')
2013/12/06 03:50:58,885      INFO:  1024^6 ('1.00 EBytes',
'1152921504606846976', '0x1000000000000000')
2013/12/06 03:50:58,885      INFO:  1024^7 ('1.00 ZBytes',
'1180591620717411303424', '0x400000000000000000')
2013/12/06 03:50:58,885      INFO:  1024^8 ('1.00 YBytes',
'1208925819614629174706176', '0x10000000000000000000')
2013/12/06 03:50:58,885      INFO:
*** DISPLAY DICTIONARY TEST ***

```

3.1.5.2.1.3 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 -----
```


3.1.5.3 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* (on page 67)).

"<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* (on page 67)).

- 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
$
```

3.1.5.3.1 2013-12-06-at-03-44-55**3.1.5.3.1.1 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 =====

3.1.5.3.1.2 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.

```

3.1.5.3.1.3 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"

```

3.2 Graphical User Interface (GUI)

Over the years, human interactions with computers 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.

- 1 **Customizable GUI Features** (see "*Graphical User Interface (GUI)*" on page 123) -
- 2 **Graphical Component Features** (on page 158) - Describes the purpose and appearance of graphical components.

3.2.1 Customizable GUI Features

The "tsWxGTUI_PyVx" Toolkit provides the "tsWxGlobals.py" module to establish configuration constants and macro-type functions for the Graphical-style User Interface mode. A Software Engineer can use the module as a template to derive a customized version more suited to the needs of a specific business or product family.

Capabilities

- 1 Provide 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.
 - a) Product Identification


```

ProductName = TeamSTARS "tsWxGTUI_PyVx" Toolkit
SubSystemName = "tsToolkitGUI"
VendorName = 'Richard S. Gordon, a.k.a. Software Gadgetry'
ThemeDate = __date__
          
```
- 2 Provide a theme-based mechanism for modifying/restoring those configuration constants as appropriate for the following users and their activities:
 - a) Supervisory Control and Data Acquisition (SCADA) System:

System Operator

Software Engineer

System Administrator

Field Service

b) Application ("tsWxGTUI_PyVx" CLI/GUI) Development System:

Software Engineer

System Administrator

Field Service

c) Toolkit ("tsWxGTUI_PyVx") Development System:

Software Engineer

System Administrator

Field Service

3 Sample Theme Definitions and Theme Selection

Draft

Feature	Excerpt Sample Theme Definition for "tsWxGTUI_PyVx" Toolkit Software Engineer
wxPython Feature Set	<pre> ThemeWxPython = { 'ProductName': ProductName, 'VendorName': VendorName, 'SubSystemName': SubSystemName, 'ThemeName': 'ThemeWxPython', 'ThemeDate': ThemeDate, 'name': 'tsWxPython', 'BackgroundColour': COLOR_BLUE, 'ForegroundColour': COLOR_WHITE, 'CharacterCellAlignment': True, # Supports ScrollBars 'Dialog': { 'name': 'Dialog', 'ButtonBarDefault': '[?][X]', 'CloseButtonLabel': '&X\tCtrl-X', 'HelpButtonLabel': '&?\tCtrl-?', 'IconizeButtonLabel': '&\tCtrl-', 'MaximizeButtonLabel': '&Z\tCtrl-Z', 'RestoreDownButtonLabel': '&z\tCtrl-z', 'BackgroundColour': COLOR_WHITE, 'ForegroundColour': COLOR_BLUE }, 'Frame': { 'name': 'Frame', 'ButtonBarDefault': '[_][Z][X]', 'CloseButtonLabel': '&X\tCtrl-X', 'HelpButtonLabel': '&?\tCtrl-?', 'IconizeButtonLabel': '&\tCtrl-', 'MaximizeButtonLabel': '&Z\tCtrl-Z', 'RestoreDownButtonLabel': '&z\tCtrl-z' }, </pre>
TeamSTARS Feature Set	<pre> ThemeTeamSTARS = { 'ProductName': ProductName, 'SubSystemName': SubSystemName, 'VendorName': VendorName, 'ThemeName': 'ThemeTeamSTARS', 'ThemeDate': ThemeDate, 'name': 'ThemeTeamSTARS', 'BackgroundColour': COLOR_BLUE, 'ForegroundColour': COLOR_WHITE, 'CharacterCellAlignment': True, # Supports ScrollBars 'Dialog': { 'name': 'Dialog', 'ButtonBarDefault': '[?][X]', </pre>

Feature	Excerpt Sample Theme Definition for "tsWxGTUI_PyVx" Toolkit Software Engineer
	<pre> 'CloseButtonLabel': '&X\tCtrl-X', 'HelpButtonLabel': '&?\tCtrl-?', 'IconizeButtonLabel': '&_\tCtrl-_', 'MaximizeButtonLabel': '&Z\tCtrl-Z', 'RestoreDownButtonLabel': '&z\tCtrl-z', 'BackgroundColour': COLOR_WHITE, 'ForegroundColour': COLOR_BLUE }, 'Frame': { 'name': 'Frame', 'ButtonBarDefault': '[_][Z][X]', 'CloseButtonLabel': '&X\tCtrl-X', 'HelpButtonLabel': '&?\tCtrl-?', 'IconizeButtonLabel': '&_\tCtrl-_', 'MaximizeButtonLabel': '&Z\tCtrl-Z', 'RestoreDownButtonLabel': '&z\tCtrl-z' }, </pre>
Theme Activation	ThemeToUse = ThemeTeamSTARS

Limitations:

- 1 The "tsWxGTUI_PyVx" Toolkit is an "nCurses"-based, character-mode emulation of the pixel-mode "wxPython" Toolkit.
- 2 Each character-mode "graphical" element occupies a cell that is 8-pixels wide and 12-pixels high.
- 3 The available set of alphabetic, numeric, punctuation and line-drawing characters is quite limited to less than 256.
- 4 Overlapping the border character cells of "graphical" elements is frequently used to conserve display screen real estate.

Notes:

- 1 The "tsToolkitCLI" is designed to be independent of the "tsToolkitGUI". However, its "tsApplication" module, collects and distributes all the keyword-value pair options and positional arguments entered by the operator and those launch parameters established by the application itself. This independence ensures that there is a common launch mechanism ("tsApplication") for both CLI and GUI applications.
- 2 The "tsCxGlobals" module establishes a common area for defining system-wide CLI configuration parameters whose values may be then communicated within and across a distributed system.
- 3 The "tsWxGlobals" module establishes a common area for defining system-wide GUI configuration parameters whose values may be then communicated within and across a distributed system.
- 4 Themes are customized sets of configuration parameters that are appropriate for a desired look and feel (there is one theme to emulate the colorless look of wxPython applications and another to create the colorful look favored by the TeamSTARS Toolkit Developers).

3.2.2 Graphical User Interface Examples

The "tsWxGTUI_PyVx" Toolkit can create displays that include

- 1 *Splash Screen* (on page 128)
- 2 *Application-Specific Screen(s)* (on page 131)

Draft

3.2.2.1 Splash Screen

The "tsWxGTUI_PyVx" Toolkit can create application-specific Splash Screens, bitmap images, that are briefly displayed during startup.

The text contents, font-style and font-color attributes are defined in "tsWxGlobals.py" file. Depending on the run time screen size, a Splash Screen may include none, one or more of the following:

Name	Default Content
theMasthead	<pre> +-----+-----+ TeamSTARS "tsWxGTUI_PyVx" 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 local/remote character-mode terminal emulators ("xterm-256color", "xterm-88color", "xterm-16color", "xterm-color", "xterm", "cygwin", "vt220 and "vt100") and terminals.</pre>
theCopyright	<pre> tsWxGTUI, v0.0.0 (pre-alpha build 02/16/2014) Author(s): Richard S. Gordon Copyright (c) 2010-2014 Richard S. Gordon, a.k.a Software Gadgetry (formerly TeamSTARS), 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 components are subject to their author's designated copyright and license notices.</pre>
theLicense	<pre> This is free and open source software. You can use, redis- tribute and modify it only under the terms and conditions set forth in the accompanying files: "COPYRIGHT.txt" and "LICENSE.txt". The "tsWxGTUI_PyVx" 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.</pre>
theNotices	<pre> Copies of the Copyright, License and Contract agreements which permit YOUR use, modification & redistribution of the "tsWxGTUI_PyVx" Toolkit are in the "NOTICES.txt" file.</pre>

For large screens exceeding 60 Columns x 35 Rows, it can included the following:

- 1 theMasthead
- 2 theCopyright

3 theLicense

For medium screens exceeding 60 Columns x 28 Rows, it can included the following:

1 theCopyright**2** theLicense

For small screens exceeding 60 Columns x 11 Rows, it can included the following:

1 theNotices

Draft

For example:



3.2.2.2 Application-Specific Screen(s)

3.2.2.2.1 test_tsWxWidgets



1 Frame

- a) Border: Thin Line
- b) Foreground Color: White
- c) Background Color: Blue
- d) Title: "Rotor Stress Monitor"
- e) Control Buttons for operator input:
 - ☐ denotes Iconize;
 - ☐[Z] denotes Maximize;
 - ☐[X] denotes Close.
- f) TextCtrl for mockup of proposed SCADA Application Monitor Screen

2 Dialog

- a) Border: Thin Line
- b) Foreground Color: Blue
- c) Background Color: White
- d) Title: "Diagnostics"
- e) Control Buttons for operator input:
 - ☐ denotes Iconize;
 - ☐[Z] denotes Maximize;
 - ☐[X] denotes Close.
- f) TextCtrl for mockup of proposed On-Line Diagnostic & Maintenance Command Screen

3 Frame

- a) Border: Thin Line
- b) Foreground Color: White
- c) Background Color: Black
- d) Title: "Redirected Output: stdout/stderr"
- e) Widgets Used:
 - TextCtrl (list scrolls upwards and shows bottom-most messages prefixed with date and time stamps)

4 Frame

- a) Border: Thin Line
- b) Foreground Color: Black
- c) Background Color: White
- d) Title: "Tasks"

Control Buttons for operator input: [Redirected], [Rotor Stress Monitor] and [Diagnostics] denotes associated Dialog and Frames (excludes Tasks)

Widgets Used: StaticText (displays run time name of application "test_tsWxRSM") TextCtrl (displays activity with the current date and time prefixed by a rotating "icon" denoted by the sequence "-", "\", "|", "/")

3.2.2.2.2 test_tsRSM

The layout and appearance is dependant of the shell window size (and any adjustment to the default by the System Operator).

The display mimics that of a 1980-model year supervisory control system in order to demonstrate how much information can be organized and presented within a limited display area (80 columns by 25 rows/lines).

Whereas the supervisory control system asynchronously used escape sequences to position, highlight (blink, bold and underline) and a later version colorized (8-color) selected display areas with new data when updated values became available, this "tsWxGTUI_PyVx" Toolkit mockup simply filled the frame and dialog windows with the equivalent non-colored plain text.

3.2.2.2.1 Dialog Overlapping Frame Bottom Half

For an 80 Column x 47 Row configuration:



Draft

1 Frame

- a) Border: Thin Line
- b) Foreground Color: White
- c) Background Color: Blue
- d) Title: "Rotor Stress Monitor"
- e) Control Buttons for operator input:
 - ☐ denotes Iconize;
 - ☐ [Z] denotes Maximize;
 - ☐ [X] denotes Close.
- f) TextCtrl for mockup of proposed SCADA Application Monitor Screen

2 Dialog

- a) Border: Thin Line
- b) Foreground Color: Blue
- c) Background Color: White
- d) Title: "Diagnostics"
- e) Control Buttons for operator input:
 - ☐ denotes Iconize;
 - ☐ [Z] denotes Maximize;
 - ☐ [X] denotes Close.
- f) TextCtrl for mockup of proposed On-Line Diagnostic & Maintenance Command Screen

3 Frame

- a) Border: Thin Line
- b) Foreground Color: White
- c) Background Color: Black
- d) Title: "Redirected Output: stdout/stderr"
- e) Widgets Used:
 - TextCtrl (list scrolls upwards and shows bottom-most messages prefixed with date and time stamps)

4 Frame

- a) Border: Thin Line
- b) Foreground Color: Black
- c) Background Color: White
- d) Title: "Tasks"

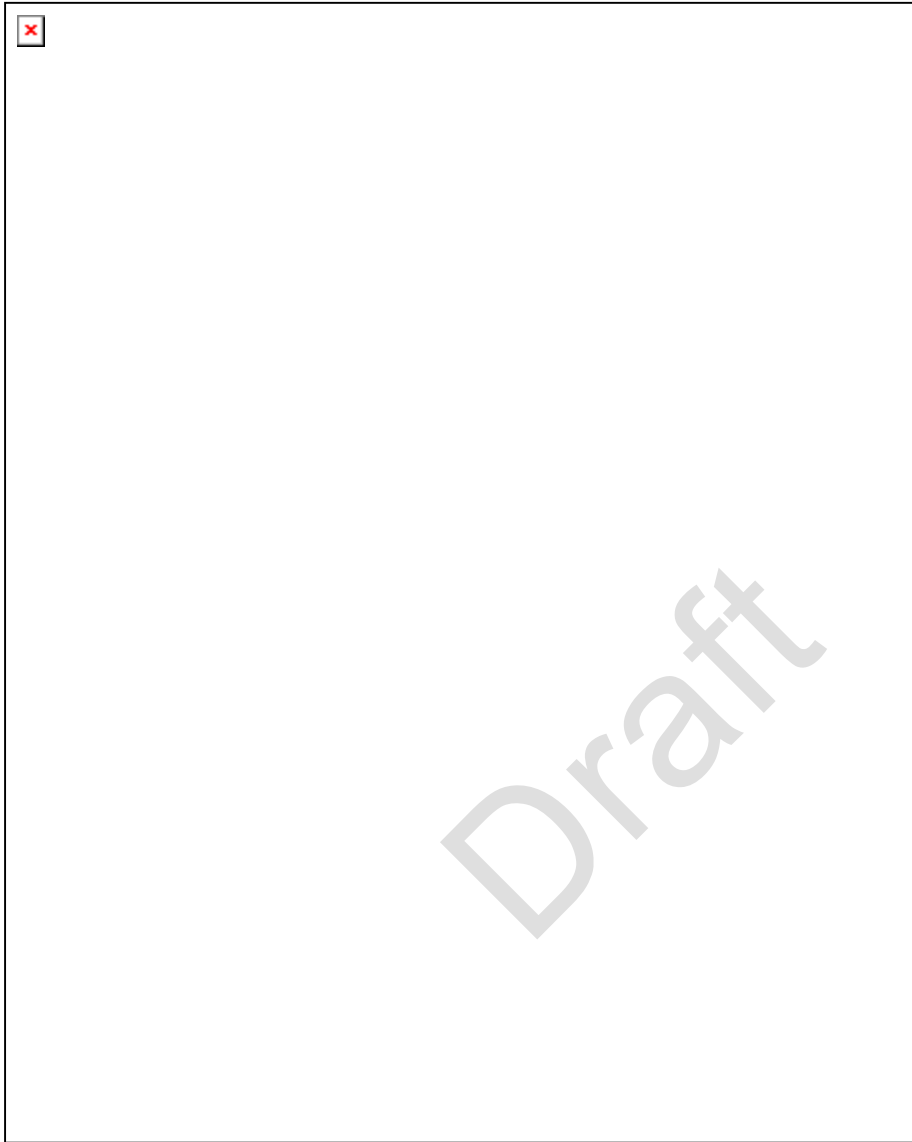
Control Buttons for operator input: [Redirected], [Rotor Stress Monitor] and [Diagnostics] denotes associated Dialog and Frames (excludes Tasks)

Widgets Used: StaticText (displays run time name of application "test_tsWxRSM") TextCtrl (displays activity with the current date and time prefixed by a rotating "icon" denoted by the sequence "-", "\", "|", "/")

3.2.2.2.2.2 Frame Vertically Stacked Above Dialog

Draft

80 Column x 100 Row using 6pt font



1 Frame

- a) Border: Thin Line
- b) Foreground Color: White
- c) Background Color: Blue
- d) Title: "Rotor Stress Monitor"
- e) Control Buttons for operator input:
 - ☐ denotes Iconize;
 - ☐ [Z] denotes Maximize;
 - ☐ [X] denotes Close.
- f) TextCtrl for mockup of proposed SCADA Application Monitor Screen

2 Dialog

- a) Border: Thin Line
- b) Foreground Color: Blue
- c) Background Color: White
- d) Title: "Diagnostics"
- e) Control Buttons for operator input:
 - ☐ denotes Iconize;
 - ☐ [Z] denotes Maximize;
 - ☐ [X] denotes Close.
- f) TextCtrl for mockup of proposed On-Line Diagnostic & Maintenance Command Screen

3 Frame

- a) Border: Thin Line
- b) Foreground Color: White
- c) Background Color: Black
- d) Title: "Redirected Output: stdout/stderr"
- e) Widgets Used:
 - TextCtrl (list scrolls upwards and shows bottom-most messages prefixed with date and time stamps)

4 Frame

- a) Border: Thin Line
- b) Foreground Color: Black
- c) Background Color: White
- d) Title: "Tasks"

Control Buttons for operator input: [Redirected], [Rotor Stress Monitor] and [Diagnostics] denotes associated Dialog and Frames (excludes Tasks)

Widgets Used: StaticText (displays run time name of application "test_tsWxRSM") TextCtrl (displays activity with the current date and time prefixed by a rotating "icon" denoted by the sequence "-", "\", "|", "/")

3.2.2.2.2.3 Dialog Overlapping Frame Lower Right Corner

Draft

200 Column x 47 Row using 8pt font



1 Frame

- a) Border: Thin Line
- b) Foreground Color: White
- c) Background Color: Blue
- d) Title: "Rotor Stress Monitor"
- e) Control Buttons for operator input:
 - ☐ denotes Iconize;
 - ☐ [Z] denotes Maximize;
 - ☐ [X] denotes Close.
- f) TextCtrl for mockup of proposed SCADA Application Monitor Screen

2 Dialog

- a) Border: Thin Line
- b) Foreground Color: Blue
- c) Background Color: White
- d) Title: "Diagnostics"
- e) Control Buttons for operator input:
 - ☐ denotes Iconize;
 - ☐ [Z] denotes Maximize;
 - ☐ [X] denotes Close.
- f) TextCtrl for mockup of proposed On-Line Diagnostic & Maintenance Command Screen

3 Frame

- a) Border: Thin Line
- b) Foreground Color: White
- c) Background Color: Black
- d) Title: "Redirected Output: stdout/stderr"
- e) Widgets Used:
 - TextCtrl (list scrolls upwards and shows bottom-most messages prefixed with date and time stamps)

4 Frame

- a) Border: Thin Line
- b) Foreground Color: Black
- c) Background Color: White
- d) Title: "Tasks"

Control Buttons for operator input: [Redirected], [Rotor Stress Monitor] and [Diagnostics] denotes associated Dialog and Frames (excludes Tasks)

Widgets Used: StaticText (displays run time name of application "test_tsWxRSM") TextCtrl (displays activity with the current date and time prefixed by a rotating "icon" denoted by the sequence "-", "\", "|", "/")

3.2.2.2.2.4 Left Aligned Frame Stacked Above Center Aligned Dialog

Draft

Draft

96 Column x 100 Row using 10 pt font

Draft



Draft

1 Frame

- a) Border: Thin Line
- b) Foreground Color: White
- c) Background Color: Blue
- d) Title: "SCADA"
- e) Control Buttons for operator input:
 - ☐ denotes Iconize;
 - ☐[Z] denotes Maximize;
 - ☐[X] denotes Close.
- f) TextCtl for mockup of proposed SCADA Application Monitor Screen

2 Dialog

- a) Border: Thin Line
- b) Foreground Color: Blue
- c) Background Color: White
- d) Title: "Diagnostics"
- e) Control Buttons for operator input:
 - ☐[?] denotes Help;
 - ☐[X] denotes Close
- f) TextCtl for mockup of proposed On-Line Diagnostic & Maintenance Command Screen

3 Frame

- a) Border: Thin Line
- b) Foreground Color: White
- c) Background Color: Green
- d) Title: "Redirected Output: stdout/stderr"
- e) Optional Control Buttons for operator input:
 - ☐ denotes Iconize;
 - ☐[Z] denotes Maximize;
 - ☐[X] denotes Close
- f) Widgets Used:
 - TextCtrl (list scrolls upwards and shows bottom-most messages prefixed with date and time stamps)

4 Frame

- a) Border: Thin Line
- b) Foreground Color: Black

c) Background Color: White

d) Title: "Tasks"

Control Buttons for operator input: [Redirected], [SCADA] and [Diagnostics] denotes associated Dialog and Frames (excludes Tasks)

Widgets Used: StaticText (displays run time name of application "test_tsWxRSM") TextCtrl (displays activity with the current date and time prefixed by a rotating "icon" denoted by the sequence "-", "\", "|", "/")

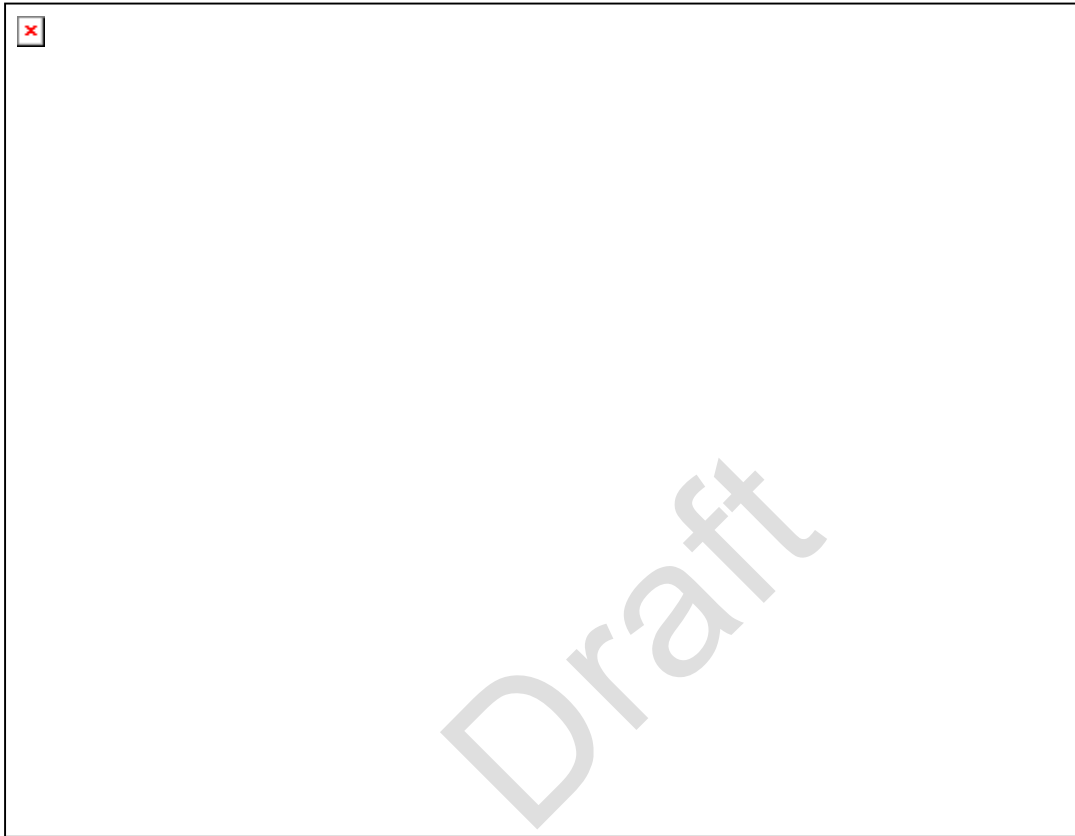
3.2.3 Graphical Desktop Features

The System Operator uses the platform's display as a workstation. The display may be organized into one or more desktops.

- *Application-specific Input and Output* (on page 149)
- *Redirected Output* (on page 151)
- *Task Bar* (on page 152)
- *Splash Screen* (on page 153)

3.2.3.1 Application-specific Input and Output

Graphical User Interface toolkit building blocks are typically selected and arranged to output desktop displays that are suitable to the application (see the following figure and description).



1 Frame

- a) Border: Thin Line
- b) Foreground Color: White
- c) Background Color: Blue
- d) Title: "metrics_communicate"
- e) Control Buttons for operator input:
 - [] denotes Iconize;
 - [Z] denotes Maximize;
 - [X] denotes Close.
- f) Widgets Used:
 - Menu Bar ("Help" and "File"),
 - Gauges ("Vertical" and "Horizontal");
 - Check Boxes (Right and Left (not shown) Aligned) and

Status Bar (with one of three panes not shown)

2 Dialog

- Border: Thin Line
- Foreground Color: Blue
- Background Color: White
- Title: "Dialog Non-Modal"
- Control Buttons for operator input:
 - [?] denotes Help;
 - [X] denotes Close
- Widgets Used:
 - Radio Box (Vertical and Horizontal);
 - Radio Buttons (Left Aligned)

3 Frame

- Border: Thin Line
- Foreground Color: White
- Background Color: Red
- Title: "Get_Metrics"
- Control Buttons for operator input:
 - [] denotes Iconize;
 - [Z] denotes Maximize;
 - [X] denotes Close
- Widgets Used:
 - TextCtrl (list shows identification numbers, names and values)

3.2.3.2 Redirected Output

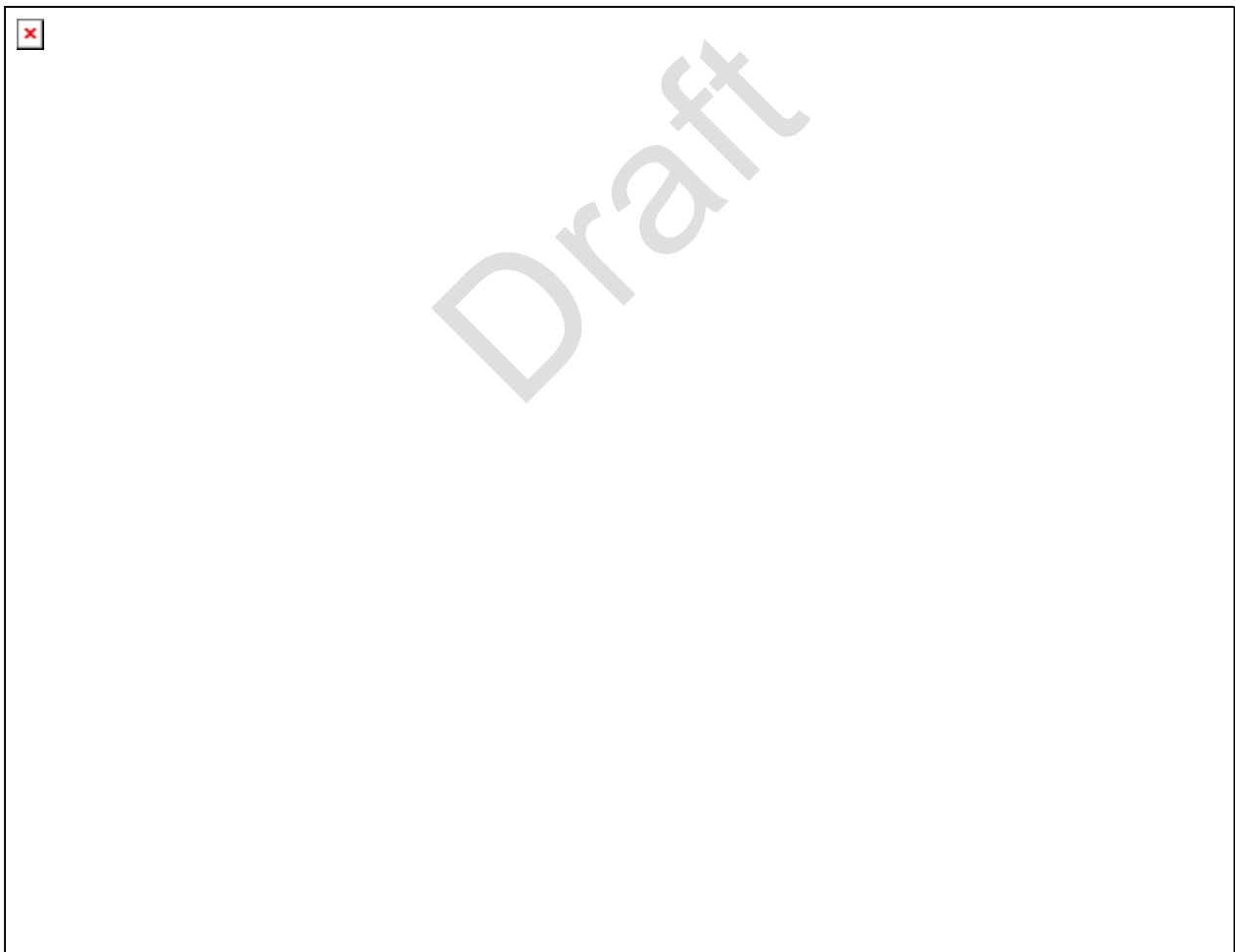
Optional window to display the UNIX-style "stdout" (progress) and "stderr" (error) messages output by an application task via a "print" statement. All output is automatically prefixed with:

- The date (year/month/day)
- The time (hour:minute:second.millisecond such as "2011/01/23 12:34:56.789")
- A separator (" - ").

When the application designer wants to draw attention to special messages, the messages may be enhanced by:

- Reversed or custom foreground and background colors
- Blink, bold, dim, normal, standout and underline special effects

In the following figure, for example, a background color change makes the line with "NOTICE" stand out from the line with the "INFO" key word.



1 Frame

- Border: Thin Line

- Foreground Color: White
- Background Color: Green
- Title: "Redirected Output: stdout/stderr"
- Control Buttons for operator input:
 - [] denotes Iconize;
 - [Z] denotes Maximize;
 - [X] denotes Close
- Widgets Used:
 - TextCtrl (list scrolls upwards and shows bottom-most messages prefixed with date and time stamps)

3.2.3.3 Task Bar

Optional window to display a Button associated with each Frame and Dialog. Upon a triggering event, if the Frame and all of its lower level windows will "gain" focus and their features will obscure those of GUI objects that "lost" focus.



1 Frame

- Border: Thin Line

- Foreground Color: Black
- Background Color: White
- Title: "Tasks"
- Control Buttons for operator input: [Redirected], [Metrics_Communicate], [Dialog] and [GetMmetrics] denotes associated Dialog and Frames (excludes Tasks)
- Widgets Used: StaticText (displays run time name of application "test_tsWxMetrics") TextCtrl (displays activity with the current date and time prefixed by a rotating "icon" denoted by the sequence "-", "\", "|", "/")

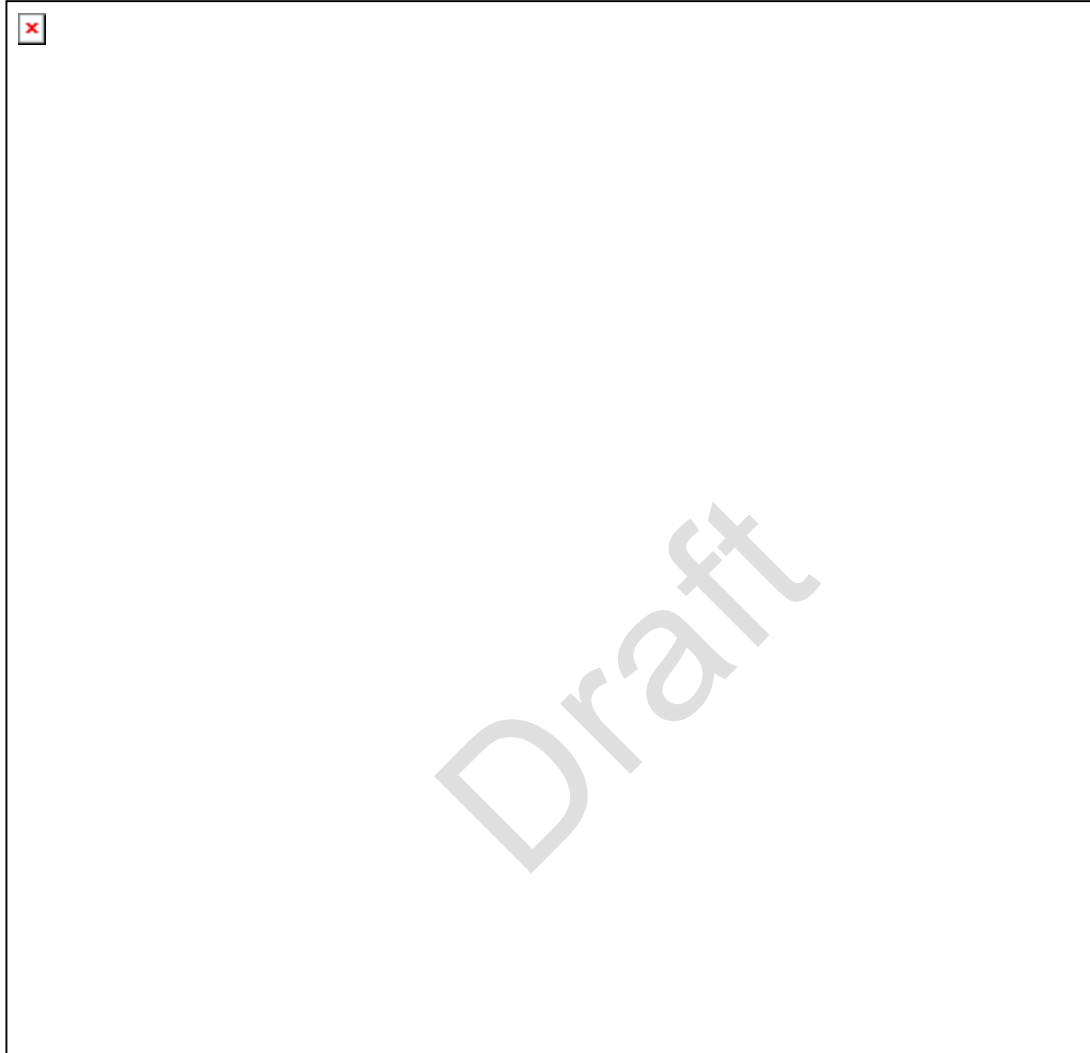
3.2.3.4 Splash Screen

Optional window to display a text-based, bitmap-like description of the application. It briefly appears before any top-level wxPython-style application Frames.

The following is a character-mode screen shot of the display constructed by the "test_tsWxSplashScreen.py" using a boxsizer layout with a multi-color scheme in an 60 column x 14 line bash shell terminal. It was taken during the bitmap image capture process.

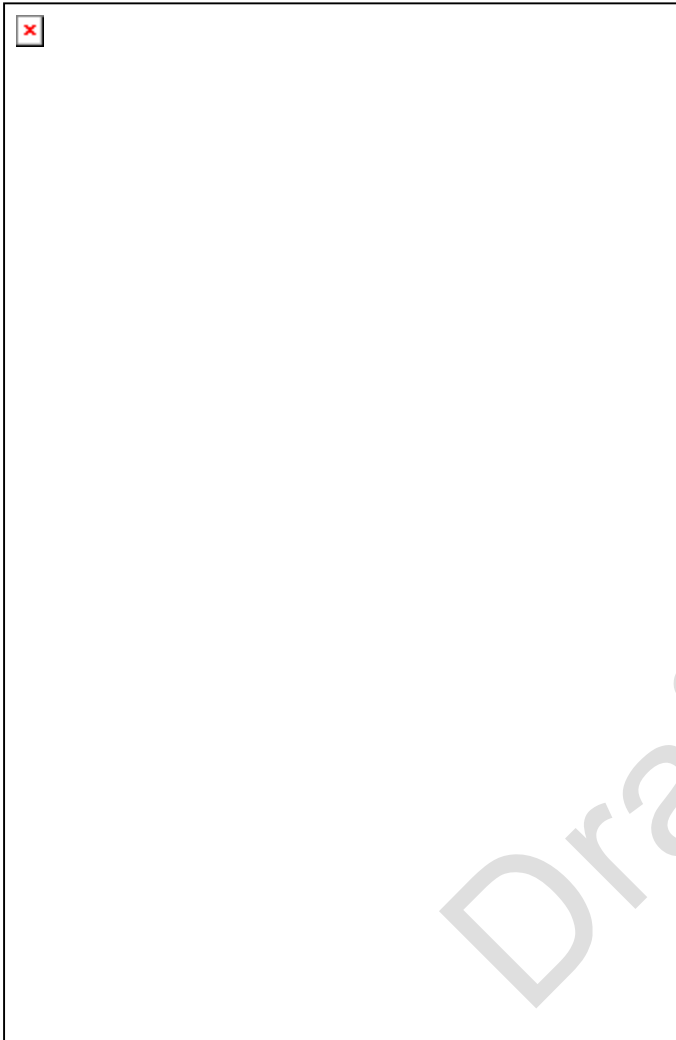


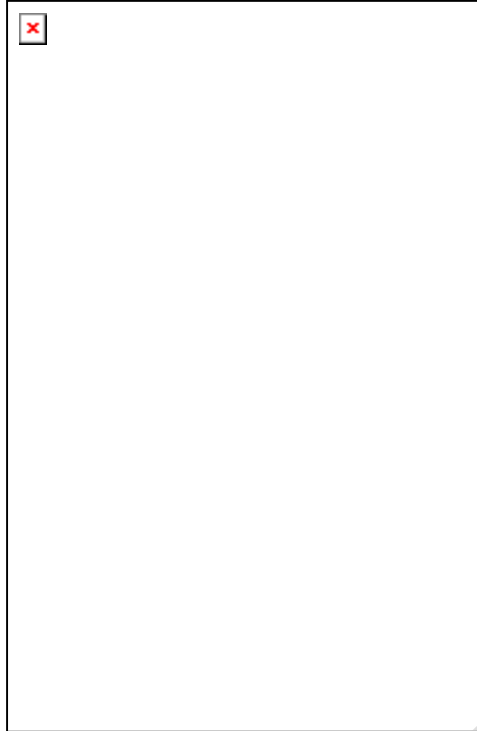
The following is a character-mode screen shot of the display constructed by the "test_tsWxSplashScreen.py" using a boxsizer layout with a multi-color scheme in an 60 column x 47 line bash shell terminal. It was taken during the bitmap image capture process.



A refinement of the "test_tsWxSplashScreen.py" became the tool "Sample_tsWxSplashScreenCreator.py". Whereas "test_tsWxSplashScreen.py" captures only one of boxsizer panel box, The "Sample_tsWxSplashScreenCreator.py" concatenated the text for the multiple panel boxes and applied the BLACK background and WHITE foreground color scheme before capturing the bitmap image. To make the bitmapped image a splash screen, it is moved to the "tsWxGTUIPython-2.x/tsLibGUI/tsWxPkg/src" directory.

The following bitmap image is displayed during startup of the "tsWxGraphicalTextUserInterface.py" prior to the startup of the main application Frame.





The above sample lacks the border lines and flexible color scheme of the original character-mode, wxPython-style splash screen mockup, taken before the image capture process.

NOTES:

1. The Splash Screen is an "nCurses" bitmap image file. It is created via the "curses.putwin(fileID)" function. The file is named "theSplashScreen.img". It must be located in the directory that contains the "tsWxGraphicalTextUserInterface.py" module,
 2. The "curses.putwin(fileID)" function can only capture the text and curses graphic-style symbols associated with a single curses window. The function cannot capture the overlapping content of multiple wxPython-style GUI Objects.
 3. The "tsWxGraphicalTextUserInterface.py" module uses the "curses.getwin(fileID)" function to retrieve the bit map image. It flashes the text and curses graphic-style symbol output on the terminal screen (stdscr) after it has initialized "curses". It pauses for 15 seconds before clearing the screen and yielding control back to the wxPython-style application program.
 4. The bitmap image file creation process is illustrated in "test_tsWxSplashScreen.py". Application programmers may modify a copy of this file to compose and save customized splash screens.
 5. The bitmap images appear to be platform specific. The Cygwin console-type or Cygwin-X xterm-type images did not display on the Mac OS X xterm-type iTerm or on the Ubuntu GNU/Linux Linux xterm-type Terminal.
-

1 Frame

- Border: Thin Line
- Foreground Color: White
- Background Color: Black
- Title: None
- Control Buttons for operator input: None
- Widgets Used: BoxSizer (subdivides Frame into four vertically oriented panes, Panel (container for TextCtrl) and TextCtrl (displays copyright of designated components))

3.2.4 Graphical Component Features

The following sections describe the purpose, appearance and operation of various graphical components.

- *Top Level GUI Objects* (on page 161)
- *Lower Level GUI Objects* (on page 173)
- *GUI Object Attributes* (on page 211)
- *GUI Object Layout Services* (on page 205)

REGION	DESCRIPTION	FEATURES
Top Of Desktop	Application Windows (Frames and Dialogs) are constrained to occupy all columns of the designated Screen Client Area which is directly above Redirected Stdio (Frame).	<p>Top Level Windows (Frames and Dialogs) are constrained to be located within Screen Border at the appropriate character cell coordinates (i.e., spaced every 8 pixels horizontally and 12 pixels vertically). Centering is subject to the same Screen Border and character cell coordinate constraints.</p> <ul style="list-style-type: none"> ▪ Border is an optional line surrounding the Client Area of the GUI object ▪ Top Border Title is optional text identifying the GUI object to the operator ▪ Top Border Buttons are optional controls available to the operator for Frames and Dialogs ▪ Client Area Menu Bar are optional activities available to the operator ▪ Client Area Tool Bar are optional utilities available to the operator ▪ Client Area Buttons, CheckBoxes and Radio Boxes & Buttons are application controls available to the operator ▪ Client Area Status Bar are application activity and progress indicators available to the operator. The Status Bar may be subdivided into two or more panels (fields) that are constrained to be located within the parent Frame border. The wx.ThemeToUse provides the means to better utilize narrow screens (by overlapping Status Bar panel borders) and to indicate hidden text (by the use of ellipses (i.e., "...")).

Middle of Desktop	Redirected Stdio (Frame) is constrained to occupy all columns of the designated rows immediately above the Task Bar.	<ul style="list-style-type: none"> ▪ Border is a line surrounding the Client Area of the GUI object. ▪ Top Border Title identifies the Redirected Stdio GUI object. ▪ Client Area displays the most recent event messages of interest to the operator. The messages are date and time stamped. The messages are labeled with the severity level associated with the event and any information describes the issue. The area is automatically sized to be proportional to the initial screen size.
Bottom Of Desktop	Task Bar (Frame) is constrained to occupy all columns of the designated bottom rows of the screen.	<ul style="list-style-type: none"> ▪ Border is a line surrounding the Client Area of the GUI object ▪ Top Border Title identifies the Task Bar GUI object and the runtime name of the application. ▪ Client Area Buttons are GUI object controls available to the operator for changing which GUI object reappears after having been iconized and which appears in front of all others. ▪ Client Area Status Bar are application activity and progress indicators available to the operator. The normally twirling baton field updates when the application is idle or non-twirling when busy or stuck. The date and time field allows the operator to assess the timeliness of the display.

Draft

3.2.4.1 Top Level GUI Objects

3.2.4.1.1 wxFrame

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.

3.2.4.1.2 wxPyOnDemandOutputWindow

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 automaticall prefixed with the date and time (such as "2011/01/23 12:34:56.789 - "

- ***Stdio Key Words and Configuration Parameters*** (on page 162) - Lists and describes each key word and configuration parameter.
- ***Severity Key Word Use Cases*** (on page 164) - Provides guidelines and examples of application programming mechanisms for outputing message text to the Stdio Redirected Output Frame.

3.2.4.1.2.1 Stdio Key Words and Configuration Parameters

The application developer can control the look and feel of the Redirected Output by adjusting Stdio Key Words and Configuration Parameters which are part of the tsWxGlobals.py file.

- The tsWxGlobals.py file contains configuration parameters that control the appearance of the text sent to the Redirected Output Frame
- The StdioMargin dictionary key word parameters control the spacing between the Frame's border and the text. To conserve space, the margin can be set to 0. For improved readability, the margin can be set to the width and height of a single character.
- The StdioMarkup dictionary key word parameters control the font attributes ("blink", "bold", "dim", "normal", "reverse", "standout" and "underline") and foreground and background colors of text lines when the text includes one of the key words followed by a colon ":" and without an intervening space. For the markup to be recognized, the message text must be terminated with a new line ("\n") such as:
`print("WARNING: Water Temperature over 130 degrees.\n")`

```

▪
▪      'StdioMargin': (0 * pixelWidthPerCharacter,
▪                    0 * pixelHeightPerCharacter)
▪
▪
▪      'StdioMarkup': {
▪          'DEBUG': {
▪              'Background': COLOR_MAGENTA, # Debug background
▪              'Foreground': COLOR_WHITE,  # Most visible color
▪              'Attributes': [DISPLAY_NORMAL]},
▪
▪          'INFO': {
▪              'Background': COLOR_BLUE,   # Comment background
▪              'Foreground': COLOR_CYAN,   # Lower visible color
▪              'Attributes': [DISPLAY_NORMAL]},
▪
▪          'NOTICE': {
▪              'Background': COLOR_BLUE,   # Comment background
▪              'Foreground': COLOR_WHITE,   # Most visible color
▪              'Attributes': [DISPLAY_NORMAL]},
▪
▪          'WARNING': {
▪              'Background': COLOR_BLACK,   # Advice background
▪              'Foreground': COLOR_YELLOW,  # A visible color
▪              'Attributes': [DISPLAY_NORMAL, DISPLAY_BLINK]},
▪
▪          'ALERT': {
▪              'Background': COLOR_BLACK,   # Advice background
▪              'Foreground': COLOR_CYAN,    # A visible color
▪              'Attributes': [DISPLAY_NORMAL]},
▪
▪          'ERROR': {
▪              'Background': COLOR_BLACK,   # Advice background
▪              'Foreground': COLOR_RED,     # Most visible color
▪              'Attributes': [DISPLAY_NORMAL]},
▪
▪          'CRITICAL': {
▪              'Background': COLOR_BLACK,   # Advice background
▪              'Foreground': COLOR_YELLOW,  # A visible color
▪              'Attributes': [DISPLAY_NORMAL, DISPLAY_BLINK]},
▪
▪          'EMERGENCY': {
▪              'Background': COLOR_BLACK,   # Advise background
▪              'Foreground': COLOR_WHITE,   # Most visible color
▪              'Attributes': [DISPLAY_NORMAL, DISPLAY_BLINK]}

```

3.2.4.1.2.2 Severity Key Word Use Cases

The Severity Key Words and their usage is described in the following table.

SEVERITY	PRIORITY	APPLICATION	EXAMPLES
NOTSET NOTE: This is NOT a key word.	0 (Min) No Details	Indication that severity has not been specified by the application. NOTE: Output for stdout.	<pre>print("Water Temperature reached 130 degrees.")</pre> <p>NOTE: Severity associated color and font attributes will NOT be applied to displayed message because the print statement did not include the required three markups:</p> <ul style="list-style-type: none"> ▪ A severity key word ▪ The colon (":") suffix ▪ The terminating new line (\n)
PRIVATE NOTE: This is NOT a key word.	1 Unlimited Details	Used to suppress console output of data intended only to be archived in a file. NOTE: Output for file only via "logger.private" statement.	<ul style="list-style-type: none"> ▪ NOTE: Output NOT available via "print" statement;
DEBUG NOTE: This is a key word.	10 Lowest usable details.	Option used to communicate troubleshooting information to the system operator. NOTE: Output for stdout.	<pre>Print("DEBUG: Should Water Temperature be over 130 degrees?.\n")</pre> <p>NOTE: Severity associated color and font attributes will be applied to displayed message because the print statement included the required three markups:</p> <ul style="list-style-type: none"> ▪ A severity key word ▪ The colon (":") suffix ▪ The terminating new line (\n)
INFO NOTE: This is a key word.	20 Progress Details	Option used to communicate progress information to the system operator. NOTE: Output for stdout.	<pre>print("INFO: Water Temperature between 120-129 degrees.\n")</pre>
NOTICE NOTE: This is a key word.	25 Milestone Details	Option used to communicate milestone information to the system operator. NOTE: Output for stdout.	<pre>print("NOTICE: Water Temperature reached 130 degrees.\n")</pre>
WARNING	30	Used by default for	<pre>print("WARNING: Water</pre>

NOTE: This is a key word.	Pre-Alarm Details	anticipated, recoverable pre-alarm conditions which ought to be brought to attention of the system operator. NOTE: Output for stdout.	Temperature over 130 degrees.\n")
ALERT NOTE: This is a key word.	35 Alarm Details	Used by default for anticipated, recoverable alarm conditions which ought to be brought to attention of the system operator. NOTE: Output for stdout.	print("ALERT: Water Temperature between 131-139 degrees.\n")
ERROR NOTE: This is a key word.	40 Failure Details	Used for unexpected, non-recoverable failures which need to be brought to the attention of the system operator. NOTE: Output for stderr.	print("ERROR: Sensor Failed. Water Temperature not in range of 32-212 degrees.\n").
CRITICAL NOTE: This is a key word.	50 Damage Related Details	Used for urgent, non-recoverable operating conditions (coolant leak) which need to be brought to the attention of the system operator. NOTE: Output for stderr.	print("CRITICAL: Thermostat failure. Temperature over 140 degrees.\n")
EMERGENCY NOTE: This is a key word.	55 Safety Related Details	Used for urgent, non-recoverable operating conditions (primary power loss) which need to be brought to the attention of the system operator. NOTE: Output for stderr.	print("EMERGENCY: Primary AC Power Supply failed. Shutdown system before Backup Battery runs out in 10 minutes.\n")

3.2.4.1.3 wxDialog

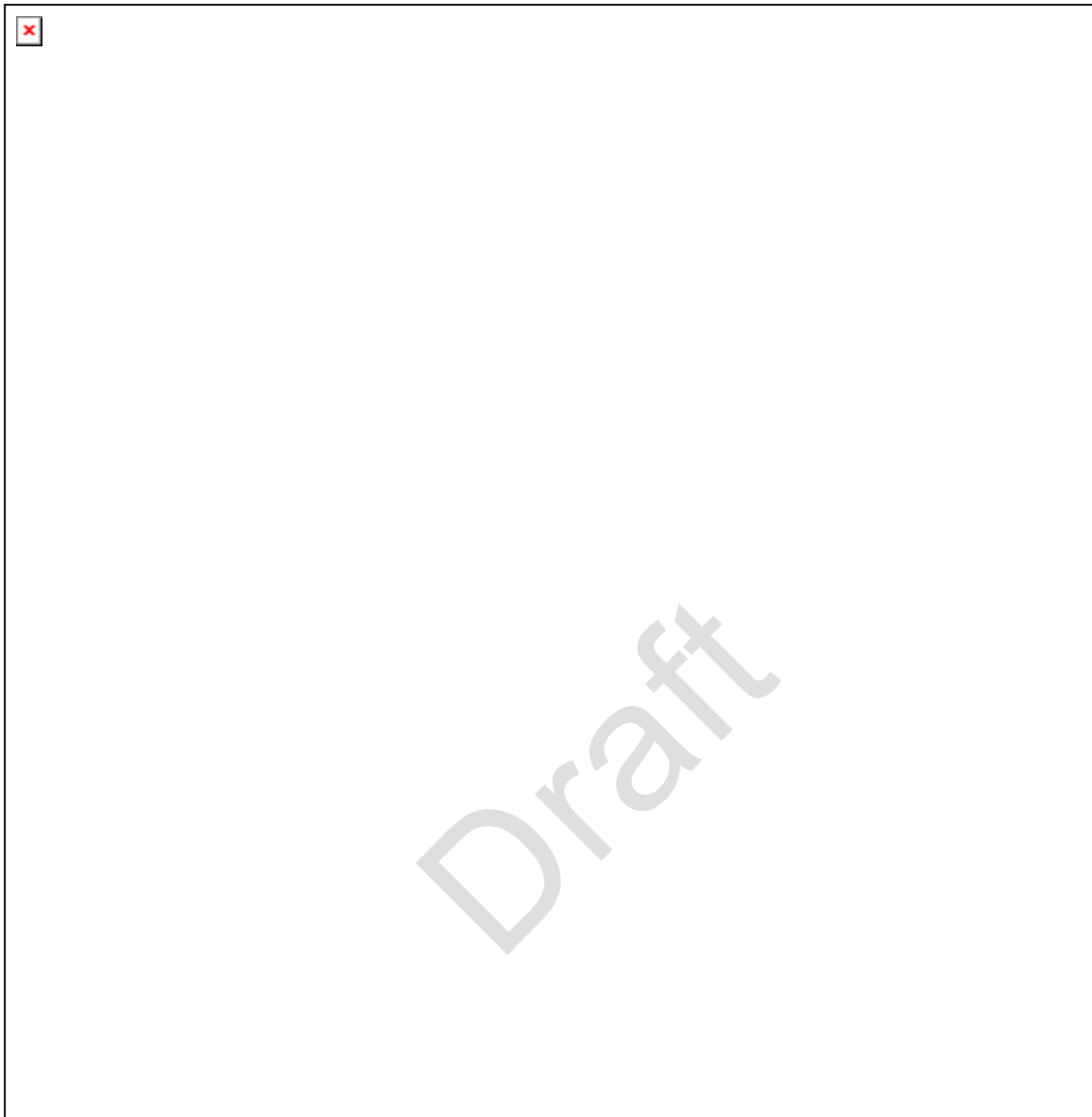
A GUI object with a title bar and sometimes a system menu, which can be moved around the screen. It can contain controls and other GUI objects and is often used to allow the user to make some choice or to answer a question. Dialogs are pop-up windows associated with an application task's menu bar (such 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.

3.2.4.1.4 wxPasswordEntryDialog (Under Construction with Usability Issues)

This class represents a dialog that requests a one-line password string from the user.

NOTES:

- 1) The TextEntryDialog is the base class for the PasswordEntryDialog class.
 - 2) The Title of this Top Level Window ("Dialog Modal") is used to display an identifying application-specific "Caption". Like other Dialog based classes, the window provides buttons to request help ("[?]") and closure ("[X]") of the dialog.
 - 3) The Label ("What password do you want to use?") of an internal Panel is used to display an identifying Message to prompt the operator for keyboard input.
 - 4) The event driven tsWxTextEditBox class (replaced the non-event driven nCurses Textpad utilities) to provide an Emacs-style edit box for operator keyboard input. It recognizes control keys for moving (forward/backward and upward/downward) and for deleting (one or more) characters in the line(s) of text.
 - 5) The OK and Cancel buttons are used to signal completion or discarding of the operator keyboard input.
 - 6) The application programmer optionally establishes the position and/or size and colors of the dialog.
 - 7) The "tsWxGTUI_PyVx" Toolkit programmer invokes sizers to layout the internal dialog features.
 - 8) Preliminary design provides a non-event driven capability in which the Operator must terminate each text entry via the console's "enter" key because mouse clicks on "OK" or "Cancel" buttons are ignored. A screenshot (not available) would show preemption of Task Bar creation, after operator typed in "Guest" text but before "enter" key has been pressed (it would display a white block cursor to right of text). The screenshot shows completion of Task Bar creation after "enter" key pressed by operator (note absence of white block cursor to right of text and its re-appearance to right of Task Bar button "Password").
-



3.2.4.1.5 **wxProgressDialog (TBD)**

This class represents a dialog that shows a short message and a progress bar

3.2.4.1.6 **wxSingleChoiceDialog (TBD)**

This class represents a dialog that shows a list of strings, and allows the user to select one

3.2.4.1.7 wxTextEditBox (Under Construction with Usability Issues)

NOTES:

- 1) This class represents that portion of the wxPasswordDialog and wxTextEntryDialog that requests a single or multi-line text string from the user.
 - 2) The design receives Keyboard event notifications (and the associated inputTuple from the tsWxEventLoop.
 - 3) The input tuple includes the following items: (ch, theCharacter, theKeyname, theFlags) where ch is the key scan code, theCharacter is the ASCII character, theKeyName is the nCurses identifier and theFlags is the applicable Shift and Alt key state.
 - 4) The design establishes and maintains a list of theCharacters. Items in the list may be deleted upon receipt of the appropriate operator actionCommand (such as Control-B, Control-D and Control-E.
 - 5) The design establishes and maintains a list of character strings, one string for each line of the display. It also establishes and maintains mappings between each displayed character and its offset into the list of theCharacters. The offset to cursor map is used to identify the column and row associated with the displayed character. The cursor to offset map is used to identify the beginning of a line for the Control-A, the end of the line for Control-E and the range of characters to be deleted for Control-K.
 - 6) Usability issue(s): a) The display flashes as each character is written to the screen. b) The display flashes as the display is refreshed. c) The operator may initiate a new line by entering Control-J (or via the KEY-ENTER). d) Consecutive multiple new lines are ignored. e). The Control-A, Control-E and Control-K are disfunctional.
-

The wxTextEditBox class is an extension to wxWidgets and wxPython. It establishes an event driven wxTextCtrl for handling single-line or multi-line text input and editing.

It provides the following methods:

- **edit([validator])** - This is the entry point you will normally use. It accepts editing keystrokes until one of the termination keystrokes is entered. If validator is supplied, it must be a function. It will be called for each keystroke entered with the keystroke as a parameter; command dispatch is done on the result. This method returns the window contents as a string; whether blanks in the window are included is affected by the stripspaces member.
- **do_command(ch)** - Processes a single command keystroke. Here are the supported special keystrokes (key codes and names) and action summary:

1 : tsOnControl_A	# Move Cursor to Left Side (<- Backward)
2 : tsOnControl_B	# Move Cursor to Previous Character(<- Backward)
4 : tsOnControl_D	# Delete Next Character (-> Forward)
5 : tsOnControl_E	# Move Cursor to Right Side (-> Forward)
6 : tsOnControl_F	# Move Cursor to Next Character (-> Forward)
7 : tsOnControl_G	# Terminate, return contents
8 : tsOnControl_H	# Delete Previous Character (<- Backward)

9 : tsOnControl_I	# Horizontal Tab Character (-> Tab)
10 : tsOnControl_J	# Terminate or Insert New Line
11 : tsOnControl_K	# Delete Empty Line or String to End of Line
12 : tsOnControl_L	# Refresh Screen
14 : tsOnControl_N	# Move Cursor to Next Line (v Downward)
15 : tsOnControl_O	# Insert Blank Line
16 : tsOnControl_P	# Move Cursor to Previous Line (^ Upward)
258: tsOnKeyDown	# Move Cursor to Next Line (v Downward)
259: tsOnKeyUp	# Move Cursor to Previous Line (^ Upward)
260: tsOnKeyLeft	# Move Cursor to Previous Character (<- Backward)
261: tsOnKeyRight	# Move Cursor to Next Character (-> Forward)
262: tsOnKeyHome	# Move Cursor to Left Side (<- Backward)
263: tsOnKeyBackSpace	# Delete Previous Character (<- Backward)
330: tsOnKeyDC	# Delete Next Character (-> Forward)
338: tsOnKeyNpage	# Move Cursor Page Bottom Right (v Downward)
339: tsOnKeyPpage	# Move Cursor Page Top Left (^ Upward)
343: self.tsOnKeyEnter	# Terminate or Insert New Line
360: tsOnKeyEnd	# Move Cursor Bot. Right (v-> Forward)

- **gather()** – This method returns the window contents as a string; whether blanks in the window are included is affected by the `stripspaces` member.

stripspaces – This data member is a flag which controls the interpretation of blanks in the window. When it is on, trailing blanks on each line are ignored; any cursor motion that would land the cursor on a trailing blank goes to the end of that line instead, and trailing blanks are stripped when the window contents are gathered.

Completion – This data member is a flag which controls the release of the edit buffer contents to the requestor.

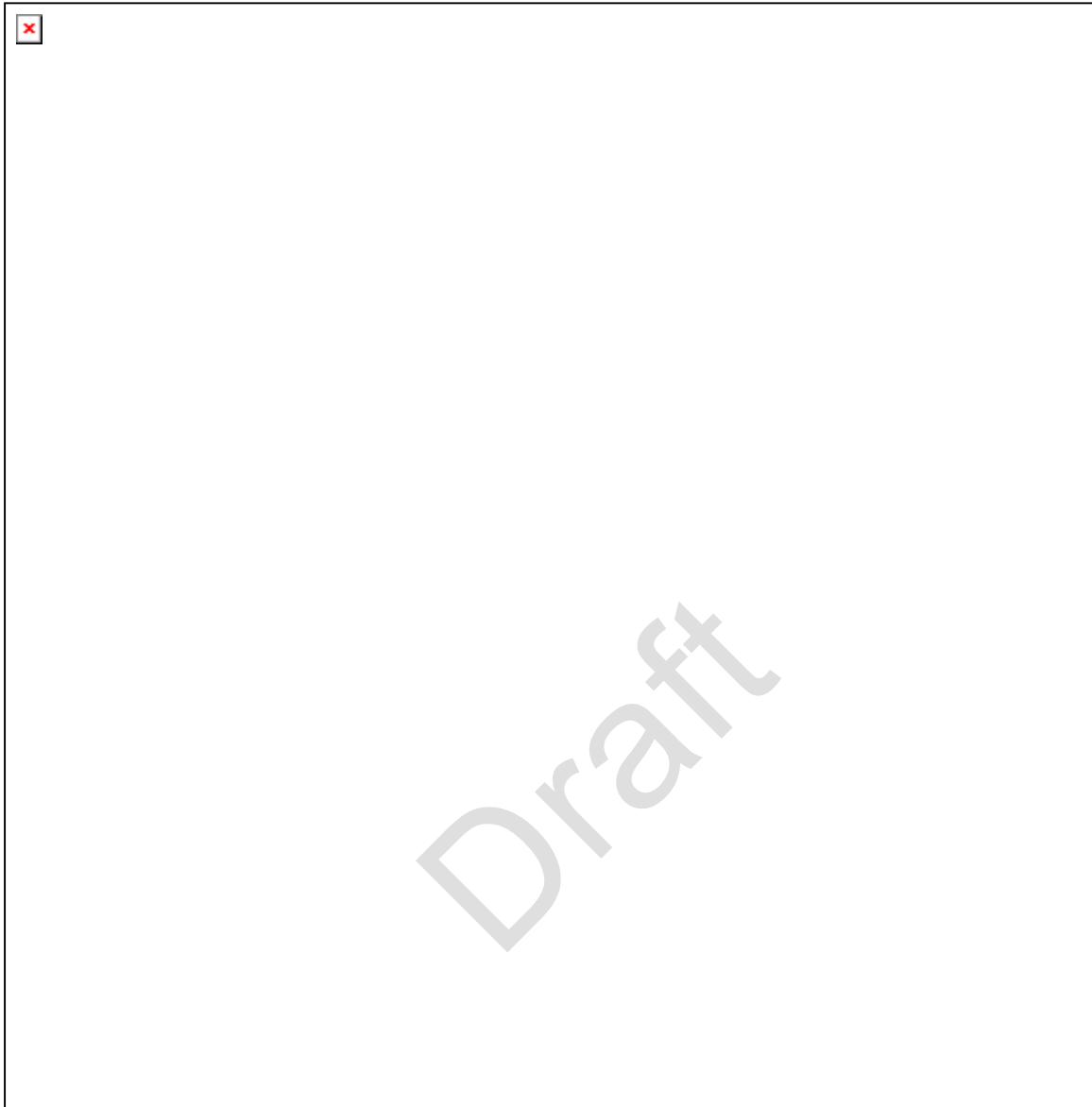
PasswordMode - This data member controls the concealing or echoing of operator input. When True, each character is replaced by an asterisk ("*") character. When False, each non-control character is echoed.

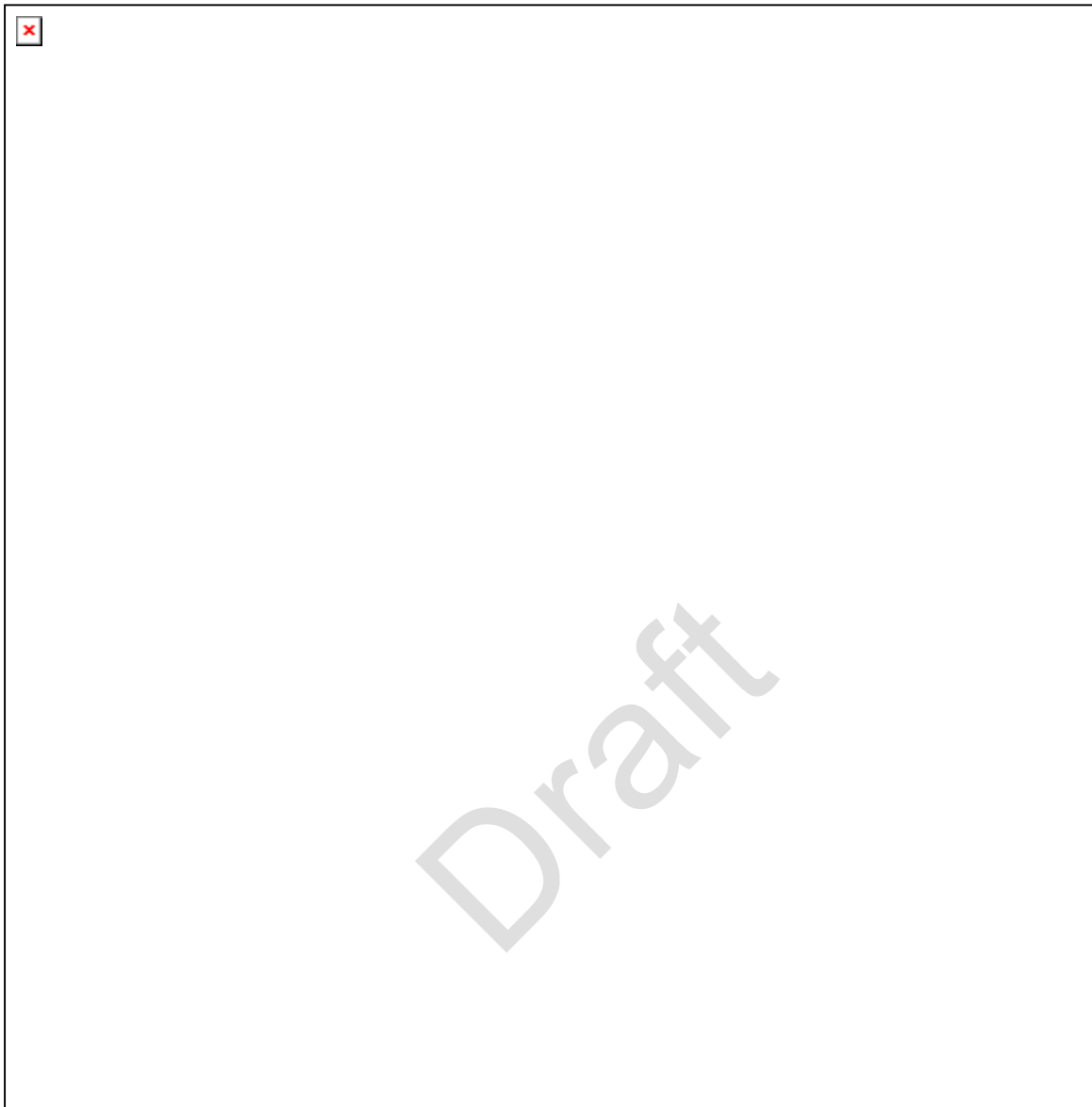
3.2.4.1.8 wxTextEntryDialog (Under Construction with Usability Issues)

This class represents a dialog that requests a one-line text string from the user.

NOTES:

- 1) The Dialog is the base class for the TextEntryDialog class.
 - 2) The Title of this Top Level Window ("Dialog Non-Modal") is used to display an identifying application-specific "Caption". Like other Dialog based classes, the window provides buttons to request help ("[?]") and closure ("[X]") of the dialog.
 - 3) The Label ("What is your favorite programming language?") of an internal Panel is used to display an identifying Message to prompt the operator for keyboard input.
 - 4) The event driven tsWxTextEditBox class (replaced the non-event driven nCurses Textpad utilities) to provide an Emacs-style edit box for operator keyboard input. It recognizes control keys for moving (forward/backward and upward/downward) and for deleting (one or more) characters in the line(s) of text.
 - 5) The OK and Cancel buttons are used to signal completion or discarding of the operator keyboard input.
 - 6) The application programmer optionally establishes the position and/or size and colors of the dialog.
 - 7) The "tsWxGTUI_PyVx" Toolkit programmer invokes sizers to layout the internal dialog features.
 - 8) Preliminary design provided a non-event driven capability, via the nCurses Textpad utilities, in which the Operator must terminate each text entry via the console's "enter" key because mouse clicks on "OK" or "Cancel" buttons are ignored. First screenshot shows preemption of Task Bar creation, after operator typed in "Python is perfect!" text but before "enter" key has been pressed (note white block cursor to right of text). Second screenshot shows completion of Task Bar creation after "enter" key pressed by operator (note absence of white block cursor to right of text and its re-appearance to right of Task Bar button "Eh??").
-





3.2.4.1.9 wxTopLevelWindow

The wxTopLevelWindow is a common base class for wxDialog and wxFrame (and their derivatives).

3.2.4.2 Lower Level GUI Objects

3.2.4.2.1 wxAboutDialogInfo (TBD)

WxAboutDialogInfo contains information shown in the standard About dialog displayed by the wxAboutBox() function

3.2.4.2.2 wxAny (TBD)

Container for any type

3.2.4.2.3 wxAnyButton (TBD)

A class for common button functionality used as the base for the various button classes

3.2.4.2.4 wxAnyValueBuffer (TBD)

Type for buffer within wxAny for holding data

3.2.4.2.5 wxApp

Application itself when wxUSE_GUI=1

3.2.4.2.6 wxAppConsole (TBD)

This class is essential for writing console-only or hybrid apps without having to define wxUSE_GUI=0

3.2.4.2.7 wxAppTraits (TBD)

Defines various configurable aspects of a wxApp

3.2.4.2.8 wxBannerWindow (TBD)

A simple banner window showing either a bitmap or text

3.2.4.2.9 wxBufferedInputStream (TBD)

This stream acts as a cache

3.2.4.2.10 wxBufferedOutputStream (TBD)

This stream acts as a cache

3.2.4.2.11 wxBusyCursor (TBD)

This class makes it easy to tell your user that the program is temporarily busy

3.2.4.2.12 wxBusyInfo (TBD)

This class makes it easy to tell your user that the program is temporarily busy

3.2.4.2.13 wxButton

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).

3.2.4.2.14 wxCaret

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.

3.2.4.2.15 wxCharBuffer (TBD)

This is a specialization of `wxCharTypeBuffer<T>` for char type

3.2.4.2.16 wxCharTypeBuffer (TBD)

`WxCharTypeBuffer<T>` is a template class for storing characters

3.2.4.2.17 wxCheckBox

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)

3.2.4.2.18 wxCheckListBox (YBD)

A `wxCheckListBox` is like a `wxListBox`, but allows items to be checked or unchecked

3.2.4.2.19 wxCmdLineParser (TBD)

`WxCmdLineParser` is a class for parsing the command line

3.2.4.2.20 wxComboBox (TBD)

A combobox is like a combination of an edit control and a listbox

3.2.4.2.21 wxComboCtrl (TBD)

A combo control is a generic combobox that allows totally custom popup

3.2.4.2.22 wxComboCtrlFeatures (TBD)

Features enabled for `wxComboCtrl`

3.2.4.2.23 wxCommand (TBD)

`WxCommand` is a base class for modelling an application command, which is an action usually performed by selecting a menu item, pressing a toolbar button or any other means provided by the application to change the data or view

3.2.4.2.24 wxCommandEvent (TBD)

This event class contains information about command events, which originate from a variety of simple controls

3.2.4.2.25 wxCommandProcessor (TBD)

`WxCommandProcessor` is a class that maintains a history of `wxCommands`, with undo/redo functionality built-in

3.2.4.2.26 wxCursor

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.

3.2.4.2.27 wxDateTime (TBD)

Contains broken down date-time representation

3.2.4.2.28 wxDialogButton

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).

"tsWxGTUI_PyVx" specific sub-class of wxButton tailored for Dialogs (Top Level Windows). It ensures that a button is directly coupled with its associated event handler. It provides the following Title Line Buttons:

- 1 Help button ("?",) that pops up a temporary window containing operator instructions associated with the Dialog.
- 2 Close button ("X") that closes (deletes or at least hides) a Dialog so that not even the Dialog's focus control button remains visible in the Task Bar.

As of 07/12/2013, the mechanism for closing (deleting) curses windows, within the wxPython emulation framework, has yet to be determined. The thought is that the "nCurses" handles must be deleted before each wxPython emulation child window and must be done before deleting the child's parent window.

3.2.4.2.29 wxDir (TBD)

WxDir is a portable equivalent of Unix open/read/closedir functions which allow enumerating of the files in a directory

3.2.4.2.30 wxDirDialog (TBD)

This class represents the directory chooser dialog

3.2.4.2.31 wxDirPickerCtrl (TBD)

This control allows the user to select a directory

3.2.4.2.32 wxDocChildFrame (TBD)

Default frame for displaying documents on separate windows

3.2.4.2.33 wxDocManager (TBD)

Part of the document/view framework supported by wxWidgets, and cooperates with the wxView, wxDocument and wxDocTemplate classes

3.2.4.2.34 wxDocParentFrame (TBD)

Default top-level frame for applications using the document/view framework

3.2.4.2.35 wxDocument (TBD)

The document class can be used to model an application's file-based data

3.2.4.2.36 wxFFFile (TBD)

WxFFFile implements buffered file I/O

3.2.4.2.37 wxFont (TBD)

A font is an object which determines the appearance of text

3.2.4.2.38 wxFontMapper (TBD)

WxFontMapper manages user-definable correspondence between logical font names and the fonts present on the machine

3.2.4.2.39 wxFrameButton

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).

"tsWxGTUI_PyVx" specific sub-class of wxButton tailored for Frames (Top Level Windows). It ensures that a button is directly coupled with its associated event handler. It provides the following Title Line Buttons:

- 1 Iconize (minimize) button ("_") that hides a Frame so that only the Frame's focus control button remains visible in the Task Bar. (

As of 07/12/2013, the mechanism for hiding curses windows, within the wxPython emulation framework, has yet to be determined. The understanding is that the "nCurses" panel module, which provides the only hiding mechanism, will require that window.show use panel_update instead of curses.doupdate.

- 2 Maximize button ("Z") or RestoreDown button ("z") that resizes a frame so that it fills the available desktop area or resizes a frame to its pre-Maximize size.

As of 07/12/2013, the mechanism for resizing curses windows, within the wxPython emulation framework, has yet to be determined

- 3 Close button ("X") that closes (deletes or at least hides) a Frame so that not even the Frame's focus control button remains visible in the Task Bar.

As of 07/12/2013, the mechanism for closing (deleting) curses windows, within the wxPython emulation framework, has yet to be determined. The thought is that the "nCurses" handles must be deleted before each wxPython emulation child window and must be done before deleting the child's parent window.

3.2.4.2.40 wxGauge

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%).

3.2.4.2.41 wxKeyboardState

Provides methods for testing the state of the keyboard modifier keys

3.2.4.2.42 wxKeyEvent

This event class contains information about key press and release events

3.2.4.2.43 wxListBox (TBD)

A listbox is used to select one or more of a list of strings

3.2.4.2.44 wxLog (TBD)

WxLog class defines the interface for the log targets used by wxWidgets logging functions as explained in the wxLog Classes Overview

3.2.4.2.45 wxMenu (Under Construction)

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.

3.2.4.2.46 wxMenuBar (Under Construction)

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).

3.2.4.2.47 wxMouseEvent

This event class contains information about the events generated by the mouse: they include mouse buttons press and release events and mouse move events

3.2.4.2.48 wxMouseState

Represents the mouse state

3.2.4.2.49 wxPanel

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.

3.2.4.2.50 wxPlatformInfo (TBD)

This class holds informations about the operating system, the toolkit and the basic architecture of the machine where the application is currently running

3.2.4.2.51 wxPopupTransientWindow (TBD)

A wxPopupWindow which disappears automatically when the user clicks mouse outside it or if it loses focus in any other way

3.2.4.2.52 wxPopupWindow (TBD)

A special kind of top level window used for popup menus, combobox popups and such

3.2.4.2.53 wxRadioBox

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.

3.2.4.2.54 wxRadioButton

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.

3.2.4.2.55 wxScrollBar

GUI object, used by tsWxScrolled, whose horizontal or vertical bar graph has controls to re-position (pan) the contents of the associated window. The position controls include symbols (arrows) at each end of the bar graph. An additional symbol, located between the end points, indicates the relative position setting.

- When the operator left clicks on one of the control symbols, the contents of the window moves one or more character positions horizontally for a horizontal scroll bar or vertically for a vertical scroll bar.
- When the operator left clicks on a point between the two arrows, the contents move in proportion to the difference between the click and the relative position setting.
- When the operator double left clicks on one of the control symbols, the contents of the window moves one page position either horizontally or vertically. For example, either the width of the horizontal scroll bar or the height of the vertical scroll bar.

3.2.4.2.56 wxScrollBarButton

GUI object, used by tsWxScrolled, that may contain text ("<", ">", "^", "v") or equivalent 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 scroll the associated displayed text left, right, up or down).

3.2.4.2.57 tsWxScrolBarGauge

GUI object, used by tsWxScrolled, whose horizontal or vertical bar shows the relative position (via a single non-blank fill character) and size (via additional non-blank fill characters) of the view area within the associated scrolled text. ScrollBarGauges are also clickable windows that trigger an associated event (such as scroll the associated displayed text left, right, up or down). A single Left-Click on a ScrollBarGauge moves the text toward or away from the top (first) or bottom (last) horizontal or vertical character in proportion to how close the mouse caret (pointer) was to the arrow buttons at either end of the associated gauge. The relative position and size of the highlighted area in the ScrollBarGauge alerts the operator to the availability of text currently hidden from view. It eliminates any justification for wasting screen real estate by alerting the operator to hidden text via the tilde ("~") character.

1 Position

- 0 % - When there is no scrolled text, the horizontal or vertical bar is entirely filled with blank characters. Otherwise, the first non-blank fill character occupies the left-most horizontal or top-most vertical position.
- 25% - The first non-blank fill character occupies the left quarter mark horizontal or top quarter mark vertical position.
- 75 % - The first non-blank fill character occupies the right quartermark horizontal or bottom quarter mark vertical position.
- 100% - The first non-blank fill character occupies the right-most horizontal or bottom-most vertical position.

2 Size

- 0 % - When there is no scrolled text to be displayed, the horizontal or vertical bar is entirely filled with blank characters.
- 25 % - When the left or top quarter of the scrolled text is displayed, the left quarter of the horizontal or top quarter of the vertical bar is entirely filled with non-blank characters.
- 75 % - When the right or bottom three quarters of the scrolled text is displayed, the the right three quarters of the horizontal or bottom three quarters of the vertical bar is entirely filled with non-blank characters.
- 100 % - When all of the scrolled text is displayed, the horizontal or vertical bar is entirely filled with non-blank characters.

3 Controls - Unlike host operating system-based GUI's, nCurses-based GUIs can only sense mouse position at the time of each mouse button click. The Curses-based GUIs cannot sense the mouse dragging needed to provide a "thumb" by which an operator can reposition ScrolledText. As a substitute, this implementation permits an operator to click on the ScrollBarGauge and have the relative position between the ScrollBarButtons (Up/Down Arrow) determine the updated ScrolledText position (scaled between Left/Right columns or Top/Bottom rows). This Curses-based GUI uses mouse clicks to emulate the "thumb" position equivalent to that achieved when "dragging" ends and the mouse button is released. (NOTE: The fine control typically available from dragging a mouse over a pixel-mode ScrollBarGauge is NOT achievable with a character-mode ScrollBarGauge and the associated application of relative ScrollBarGauge character cell position scaling.)

- Clicking the mouse button at a point midway between the horizontal arrows will begin the horizontal page with the character in the middle column while the vertical position does not change.

- Clicking the mouse button at a point midway between the vertical arrows will begin the vertical page with the character in the middle row while the horizontal position does not change.
- A two-step process (horizontal followed by vertical or vertical followed by horizontal) must be used to adjust those ScrolledWindows that provide Dual ScrollBars.

3.2.4.2.58 wScrolled

Template for GUI object whose horizontal and/or vertical bar graph has controls to re-position (pan) the contents of the associated window. The position controls include symbols (arrows) at each end of the bar graph. An additional symbol, located between the end points, indicates the relative position setting and size of the displayed area relative to the total data space.

- When the operator left clicks on one of the control symbols, the contents of the window moves one character position horizontally for a horizontal scroll bar or vertically for a vertical scroll bar.
- When the operator left clicks on a point between the two arrows, the contents move in proportion to the difference between the click and the relative position setting.
- When the operator double left clicks on one of the control symbols, the contents of the window moves one page position either horizontally or vertically. For example, either the width of the horizontal scroll bar or the height of the vertical scroll bar.

3.2.4.2.58.1 tsWxScrolled Implementation

wxPython Style - Parent Rectangle		
+--Optional Label-----		++
Client Rectangle - Top Left	scrollbar vertical	^
		++
		#
ScrolledText		#
		#
		++
scrollbar horizontal	Client Rectangle - Bottom Right	v
+-----		++
< ##### >		
+-----		++

1 tsWxScrolled - Creates multiple GUI objects within the client area of its parent GUI object to provide a peep hole and control buttons which enable an operator to scroll through text that could not otherwise be displayed all at once.

- Depending on application needs, there may horizontal and/or vertical scrollbars and their associated buttons and "thumb" gauges.
- Each instance of this top level module initiates and supervises the following sub-components.

2 tsWxScrolledText - Creates and operates a GUI object that provides the scrollable text window. It displays only the text that fits within the space assigned by tsWxScrolled.

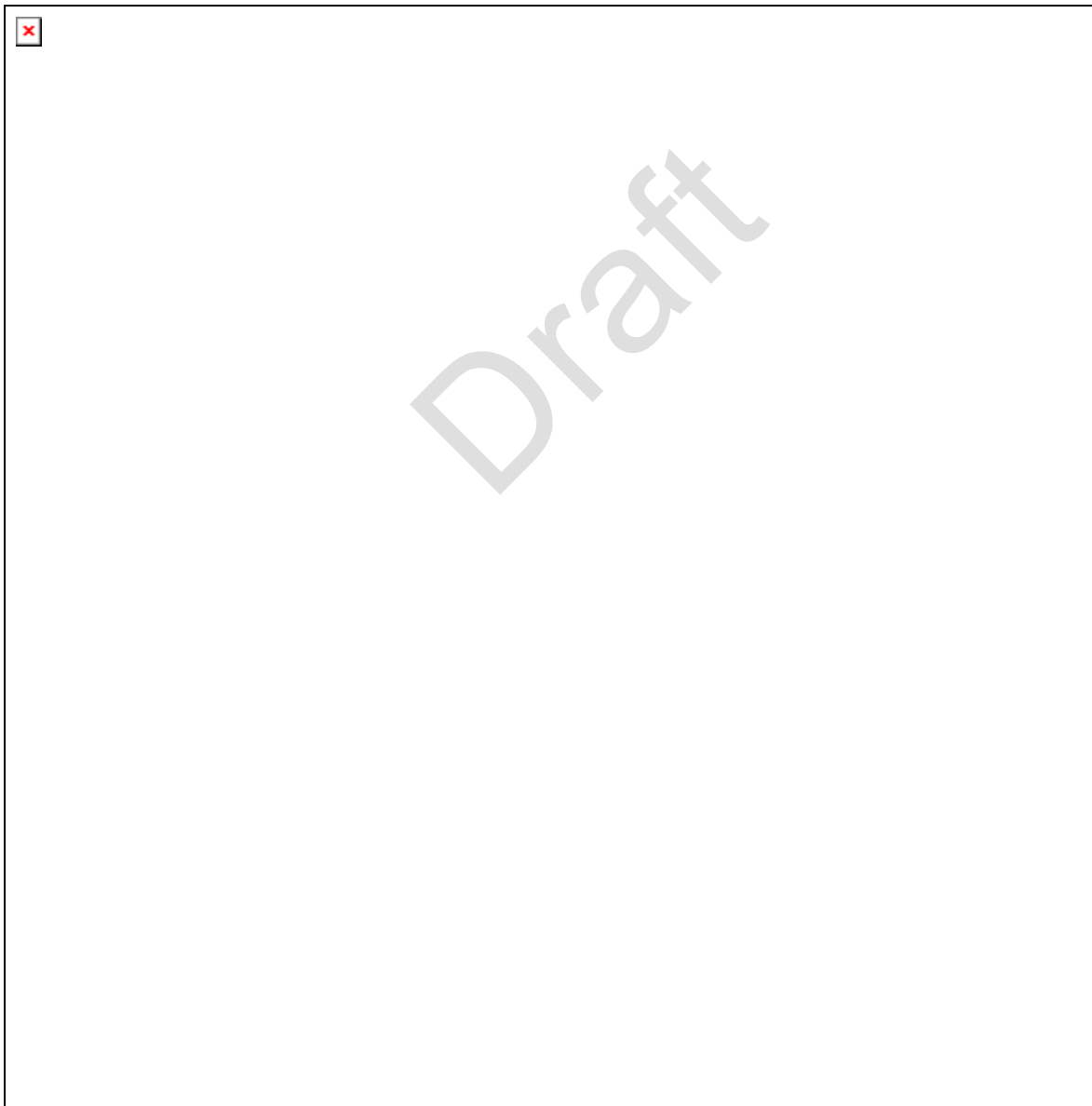
- When the text is too short to fill the displayed line, The text is appended with blanks.
- When the text is too long for the displayed line, the displayed text is truncated and appended with a tilde ("~") character to alert the operator.
- Depending on application needs, the starting column and/or row of the text portion to be display can be set upon receipt of an event notification from the associated horizontal or vertical scrollbar button.

- Upon each text display update, it will issue the appropriate horizontal or vertical event notification to the associated `tsWxScrollBarGauge` instance.
- 3 `tsWxScrollBar`** - Creates and supervises a multi-component GUI object with the position and size established by `tsWxScrolled`.
- When the `tsWxScrolled` specified a horizontal scrollbar, this instance will initiate creation of two scrollbarbuttons, one for the left arrow ("`<`") and one for the right arrow ("`>`").
 - When the `tsWxScrolled` specified a vertical scrollbar, this instance will initiate creation of two scrollbarbuttons, one for the up arrow ("`^`") and one for the down arrow ("`v`").
- 4 `tsWxScrollBarButton`** - Creates and operates the GUI object that is displayed and will respond to the event notifications of the operator's mouse button clicks.
- Depending on the button's purpose, it will forward the appropriate horizontal or vertical event notification to the associated `tsWxScrolledText` instance.
- 5 `tsWxScrollBarGauge`** - Creates and operates the GUI object for a horizontal or vertical bar graph whose highlighted and non-highlighted areas display the position and size of text displayed in the associated `tsWxScrolledText` window relative to the undisplayed text.

3.2.4.2.58.2 80-col x 40-row Cygwin-xterm (mintty) Scrollbar Layouts

NOTES:

1. A BoxSizer was used to subdivide the client area of the parent frame into three panels (blue for horizontal-only scrollbar, green for vertical-only scrollbar and red for the joint horizontal-vertical scrollbars). The resulting fractional character width and height dimensions of the panels makes overlapping of the blue, green and red panels likely.
 2. A border was used to surround the blue, green and red panels. This prevents the afore mentioned panel overlap from effecting the appearance of the scrollbar and scrollable text areas.
 3. The layout of GUI objects within the scrollbars and associated scrollable text areas was aligned to non-fractional character cell width and height dimensions.
-

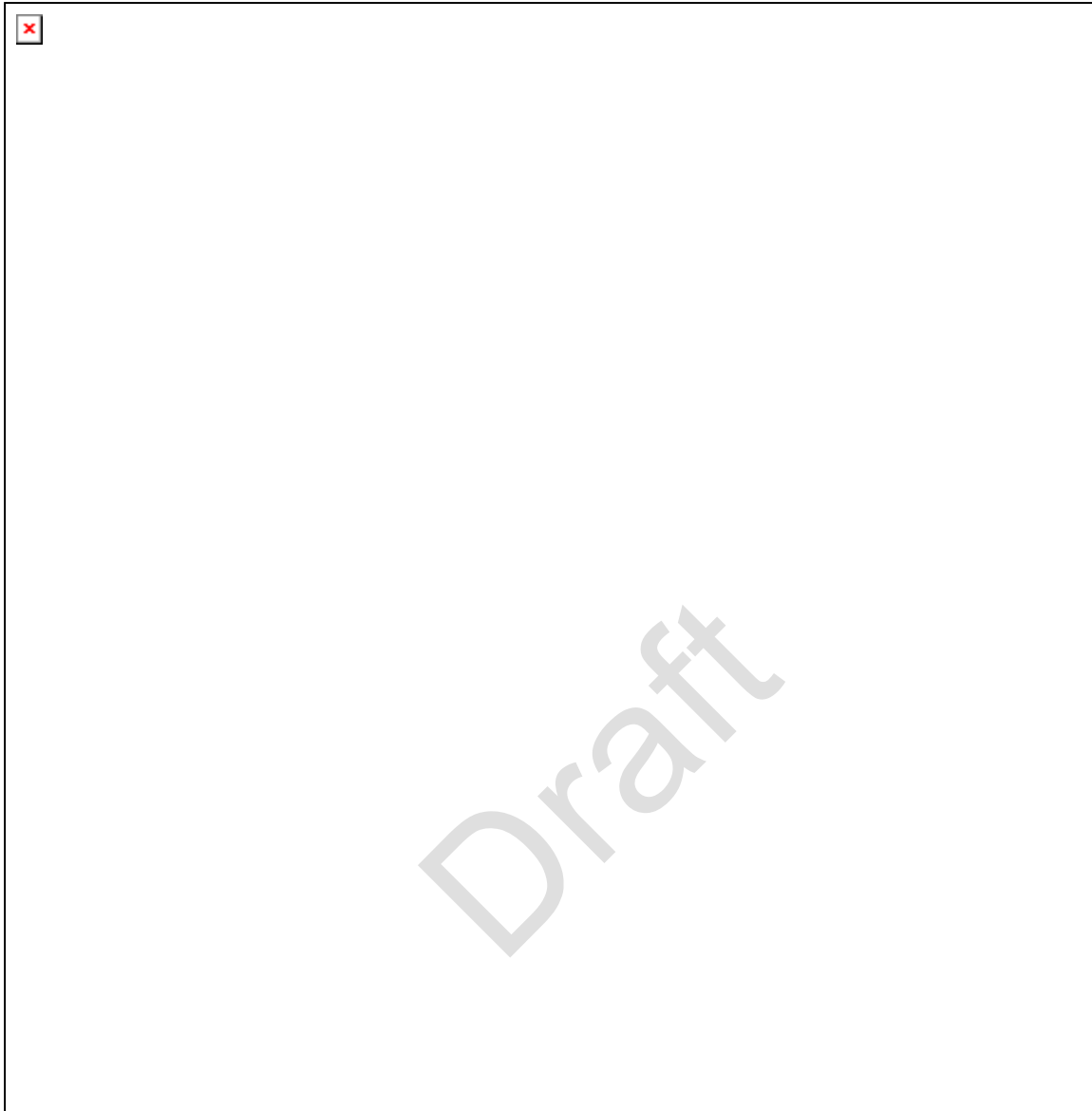


Draft

3.2.4.2.58.3 80-col x 40-row Cygwin-xterm (mintty) with improved button appearance

NOTES:

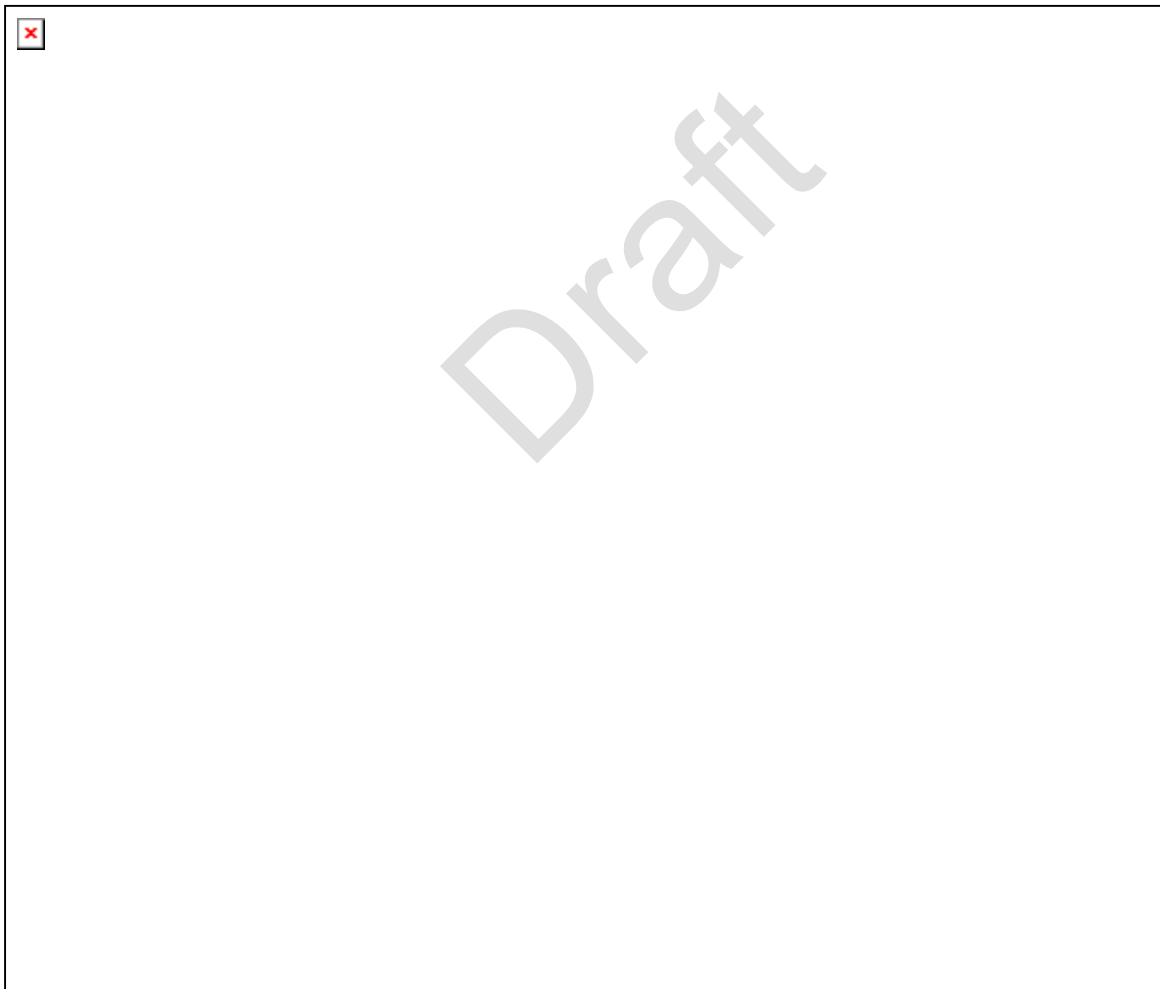
1. The application program `test_tsWxScrolled.py` demonstrates the `tsWxScrolled` (an internally used template for publically used classes such as `tsWxScrolledWindow`), `tsWxScrolledText`, `tsWxScrollBar`, `tsWxScrolBarButtons` and `tsWxScrollBarGauge` features of the "tsWxGTUI_PyVx" Toolkit's Graphical Text User Interface.
 2. The application program `test_tsWxScrolledWindow.py` demonstrates the `tsWxScrolledWindow` (an publically used base class) features of the "tsWxGTUI_PyVx" Toolkit's Graphical Text User Interface.
 3. The application program uses a `BoxSizer` to subdivide the client area of the parent frame into three panels. On the left is a cyan panel for a horizontal-only scrollbar. In the middle is a green panel for a vertical-only scrollbar. On the right is a yellow panel for both a horizontal and a vertical scrollbar. Each panel features an optional border and an optional label. The resulting fractional character width and height dimensions of the panels makes overlapping of the cyan, green and yellow panels dependent of the position and size of the parent frame. A panel border prevents the afore mentioned panel overlap from effecting the appearance of the scrollbar and scrollable text fetures.
 4. The `tsWxScrolled` module sizes, positions and iniates the required `tsWxScrolledText` and `tsWxScrollBar` components. The layout is aligned to non-fractional character cell width and height dimensions. Each instance features an optional border and optional descriptive label.
 5. The `tsWxScrolledText` module implements the size, position and style as specified by the `tsWxScrolled` module. The scrollable text area features a black foreground and white background color scheme for optimum readability. It also features an optional descriptive label.
 6. The `tsWxScrollBar` module implements the size, position and style as specified by the `tsWxScrolled` module. Each instance sizes, positions and iniates the required `tsWxScrollBarButtons` and `tsWxScrollBarGauge`. Each instance also features a border. Each instance also features a black foreground and white background color scheme for optimum readability. In order to achieve border integrity for the arrow buttons, they must be created after their associated `tsWxScrollBarGauge`.
 7. The `tsWxScrollBarButton` module implements the size, position and style as specified by the `tsWxScrollBar` module based on `tsWxScrolled` module specifications. Arrow-style buttons include left "<", right ">", up "^" and down "v".
 8. The `tsWxScrollBarGauge` module implements the size, position and style as specified by the `tsWxScrollBar` module based on `tsWxScrolled` module specifications. Each `ScrollBarGauge` "thumb" instance depicts the position and size of its associated displayed text relative to non-displayed text.
 9. While under development, there may be no correlation between the `tsWxScrolledText` and `tsWxScrollBarGauge` displays, as noted in the following "DEBUG message in the "Redirected Output" display.
-



3.2.4.2.58.4 160-col x 80-row Cygwin-style ScrollBar Layouts

NOTES:

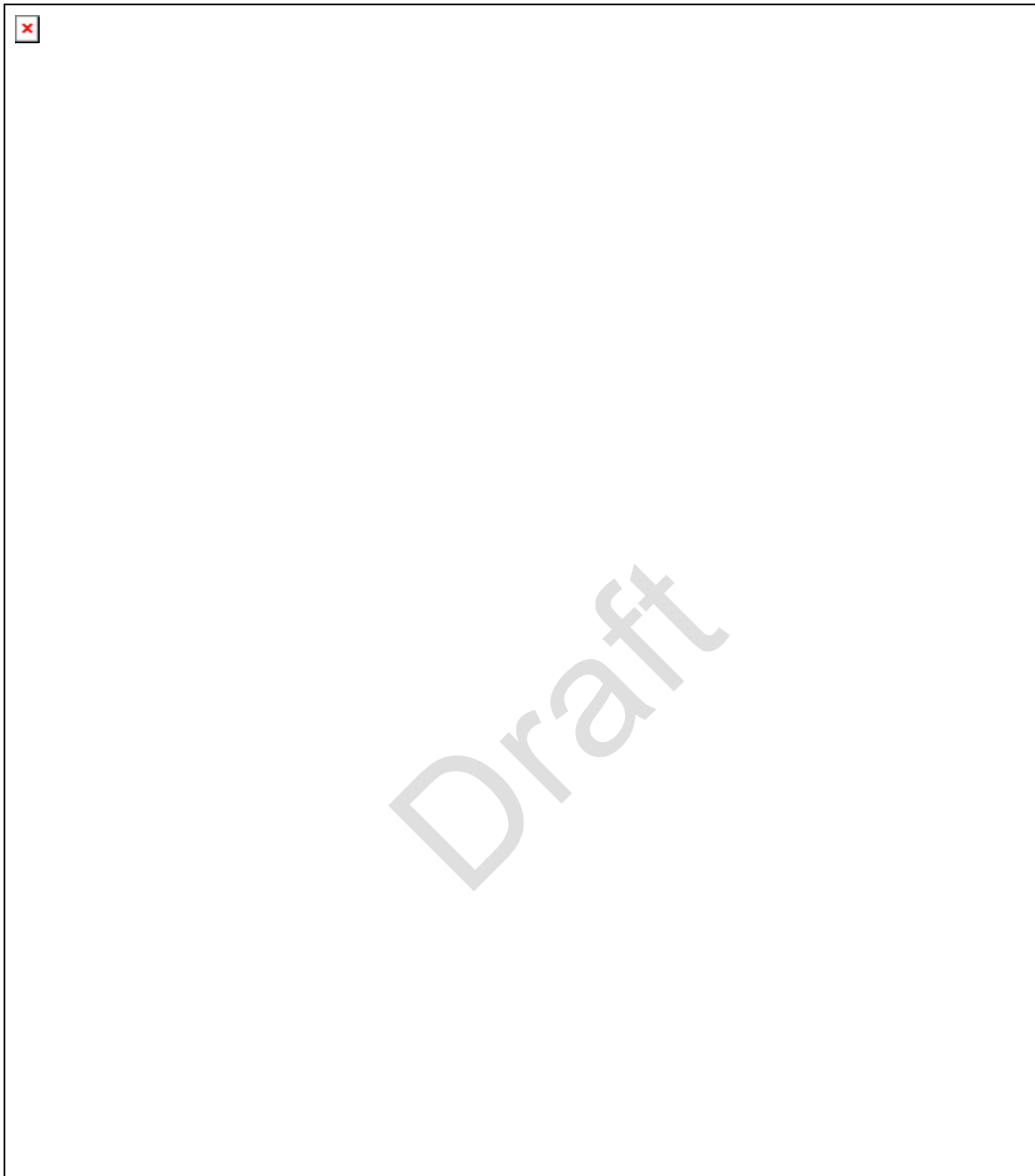
1. A BoxSizer was used to subdivide the client area of the parent frame into three panels (blue for horizontal-only scrollbar, green for vertical-only scrollbar and red for the joint horizontal-vertical scrollbars). The resulting fractional character width and height dimensions of the panels makes overlapping of the blue, green and red panels likely.
 2. A border was used to surround the blue, green and red panels. This prevents the afore mentioned panel overlap from effecting the appearance of the scrollbar and scrollable text areas.
 3. The layout of GUI objects within the scrollbars and associated scrollable text areas was aligned to non-fractional character cell width and height dimensions. It did NOT prevent a white area from "showing" in the fractional dimensioned area between the right side of the green and red scrollbars and the right side of their associasted panels.
-



3.2.4.2.58.5 96-col x 55-row xterm (Terminal) ScrolledText Layout

NOTES:

1. A BoxSizer was used to subdivide the client area of the parent frame into three panels (cyan for horizontal-only scrollbar, green for vertical-only scrollbar and yellow for the joint horizontal-vertical scrollbars). The resulting fractional character width and height dimensions of the panels makes overlapping of the blue, green and red panels likely.
 2. A border was used to surround the cyan, green and yellow panels. This prevents the afore mentioned panel overlap from effecting the appearance of the scrollbar and scrollable text areas. Each panel was labeled to identify its purpose.
 3. The layout of GUI objects within the scrollbars and associated scrollable text areas was aligned to non-fractional character cell width and height dimensions. It did NOT prevent a white area from "showing" in the fractional dimensioned area between the right side of the cyan and green scrollbars and between the right side of the green and yellow scrollbars.
 4. ScrollBarButton (Left Arrow "<", Right Arrow ">", Up Arrow "^" and Down Arrow "v") are displayed with a black background color and a white foreground color (the reverse of the ScrollBarText colors).
 5. ScrollBarGauge (a.k.a. the Thumb) typically displays the relative position and size of the scrolled page relative to the registered lines of text using a string of blank (" ") and non-blank ("X") fill characters. This prototype version substituted diagnostic text for the blank and non-blank fill characters.
 6. ScrollBarText is displayed in a black foreground color on a white background color. The cyan and green ScrollBars display two lines of text to identify column positions and text to identify line positions. A tilde ("~") character in the right most column identifies that the line has been truncated. Display of the truncated line will require the operator to scroll by clicking the mouse over the right arrow (">"). The yellow ScrollBar displays text that has been previously scrolled down five lines after the operator clicked on the down arrow ("v").
 7. The layout algorithms overlap borders to optimize use of display real estate: a) the left border of the green panel is overlapped by that of the "Short View" ScrolledText, b) the right border of the green panel is overlapped by the left border of the yellow panel; c) the right border of the left arrows ("<") are overlapped by the adjacent Horizontal ScrollBarGauge; d) the right border of the Horizontal ScrollBarGauge is overlapped by the left border of the right arrow (">"); e) the bottom border of the up arrow ("^") is overlapped by the top border of the adjacent Vertical ScrollBarGauge; and f) the bottom border the Vertical ScrollBarGauge is overlapped by the top border of the down arrow ("v").
-



3.2.4.2.59 tsWxScrolledText

GUI object, used by `tsWxScrolled`, that features a text control which allows lines of text to be registered in a list and then displayed with horizontal and/or vertical scrolling so as to fit page-sized portions (x-Columns by y-Rows) of the text within an associated scrolled window.

1 Mouse Left-Click

- Left Arrow - Scrolls text left by one (Column) increment
- Right Arrow - Scrolls text right by one (Column) increment

- Up Arrow - Scrolls text up by one (Row) increment
- Down Arrow - Scrolls text down by one (Row) increment

2 Mouse Left-DClick

- Left Arrow - Scrolls text left by one (x-Column page) increment
- Right Arrow - Scrolls text right by one (x-Column page) increment
- Up Arrow - Scrolls text up by one (y-Row page) increment
- Down Arrow - Scrolls text down by one (y-Row page) increment

3 Mouse Right-Click

- Left Arrow - Scrolls text to left most Column
- Right Arrow - Scrolls text to right most Column
- Up Arrow - Scrolls text to top most Row
- Down Arrow - Scrolls text to bottom most Row

3.2.4.2.60 wxScrolledWindow

GUI object whose horizontal and/or vertical bar graph has controls to re-position (pan) the contents of the associated window. The position controls include symbols (arrows) at each end of the bar graph. An additional symbol, located between the end points, indicates the relative position setting and size of the displayed area relative to the total data space.

- When the operator left clicks on one of the control symbols, the contents of the window moves one character position horizontally for a horizontal scroll bar or vertically for a vertical scroll bar.
- When the operator left clicks on a point between the two arrows, the contents move in proportion to the difference between the click and the relative position setting.
- When the operator double left clicks on one of the control symbols, the contents of the window moves one page position either horizontally or vertically. For example, either the width of the horizontal scroll bar or the height of the vertical scroll bar.

3.2.4.2.61 wxSlider (TBD)

A slider is a control with a handle which can be pulled back and forth to change the value

3.2.4.2.62 wxSpinButton (TBD)

A wxSpinButton has two small up and down (or left and right) arrow buttons

3.2.4.2.63 wxSpinCtrl (TBD)

WxSpinCtrl combines wxTextCtrl and wxSpinButton in one control

3.2.4.2.64 wxSplashScreen (Deprecated)

The "wxSplashScreen" module has been deprecated because a simpler, more direct and automated re-sizing mechanism has been built into the "tsWxGrapcalTextUerInterface" module.

GUI object that creates and saves an application-specific pixel-mode bitmap image file which will be displayed upon application startup. It may contain text or character-mode icons. It features a window with or without a thin border and text describing the application. It is created and shown during application initialization, before the application's own window. It disappears after a 15 second time-out.

NOTES:

- 1) A copy of the test_tsWxSplashScreen application can be modified to create and save an application-specific pixel-mode bitmap image file.
 - 2) To maintain compatibility with the pixel-mode bitmap image file loader, separate image must be created and installed for each specific terminal display type (cygwin, xterm etc.) and size (pixel width and height) configuration.
-

3.2.4.2.65 wxStaticBox

GUI objects with a rectangular shape that are drawn around other GUI objects to denote a logical grouping of items. The grouping is Each StaticBox is named to uniquely identify the logical grouping of items. Unlike a RadioBox, the selection of one StaticBox item does not implicitly cancel the function associated with the other items in its group.

3.2.4.2.66 wxStaticLine

A static line is just a line which may be used in a dialog to separate the groups of controls.

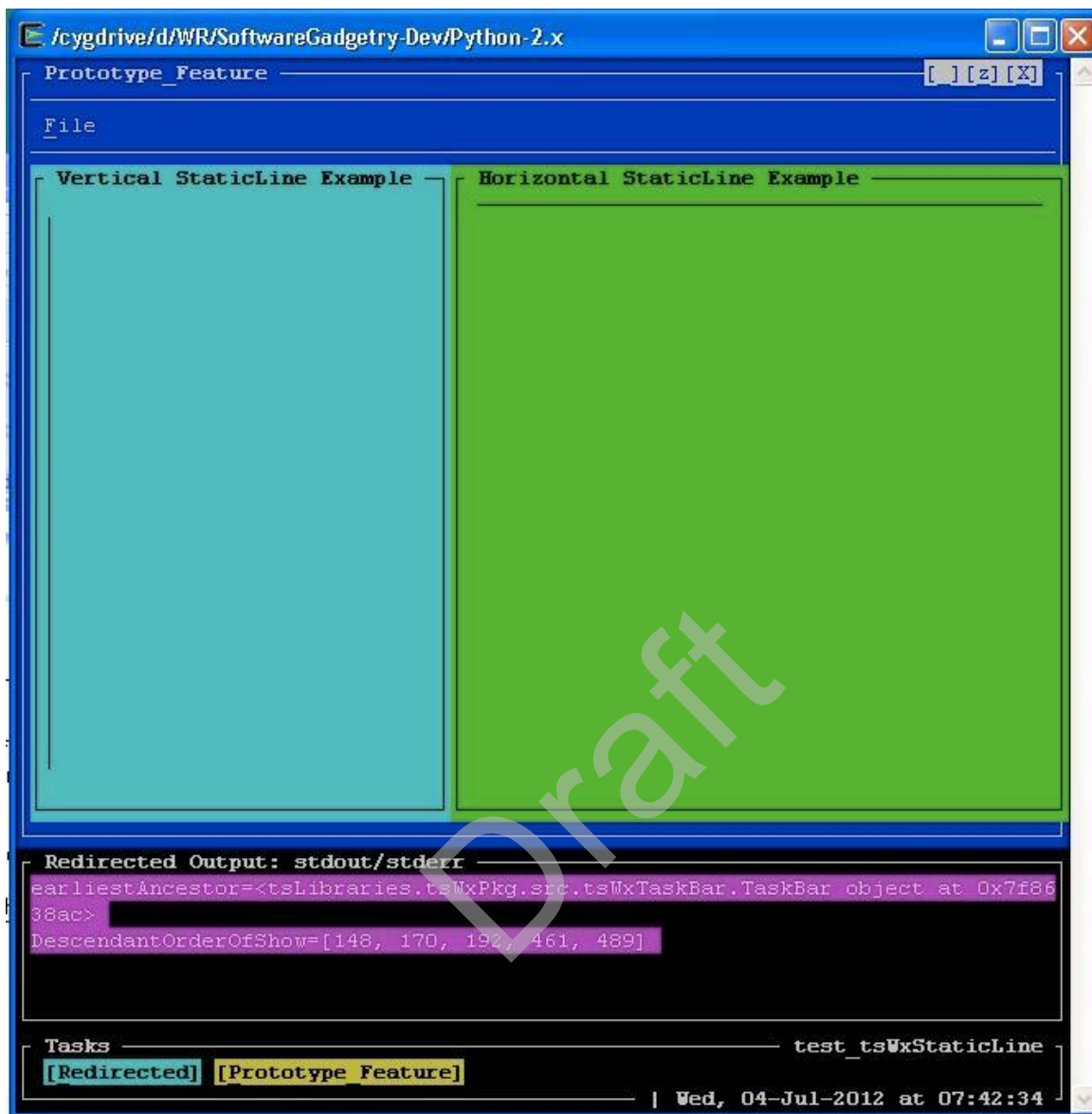
NOTES:

1) While a vertical line can fill its parent's client area height, a horizontal line can fill its parent's client area width except for right-most column which must be reserved for cursor in order to not wrap into next row. (The white block associated with the blinking cursor is not visible in the Horizontal StaticLine Example because the cursor had been moved beyond the xterm screen border in order to select and trigger the screenshot.)

2) To ensure symmetry, the design blank fills the left-most horizontal column.

3) To ensure consistency, the design blank fills the top-most and bottom-most vertical rows.

Draft



3.2.4.2.67 wxStaticText

GUI object that features a text control that displays one or more lines of read-only text.

3.2.4.2.68 wxStatusBar

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.

3.2.4.2.69 wxStatusBarPane

A status bar pane data container used by `wxStatusBar`

3.2.4.2.70 wxStopWatch (TBD)

Allow you to measure time intervals

3.2.4.2.71 wxSystemSettings

WxSystemSettings allows the application to ask for details about the system

3.2.4.2.72 wxTaskBarIcon (TBD)

This class represents a taskbar icon

3.2.4.2.73 wxTextAttr (TBD)

WxTextAttr represents the character and paragraph attributes, or style, for a range of text in a wxTextCtrl or wxRichTextCtrl

NOTE: The wxTextAttr might replace optional Markup arguments

3.2.4.2.74 wxTextEntry (TBD)

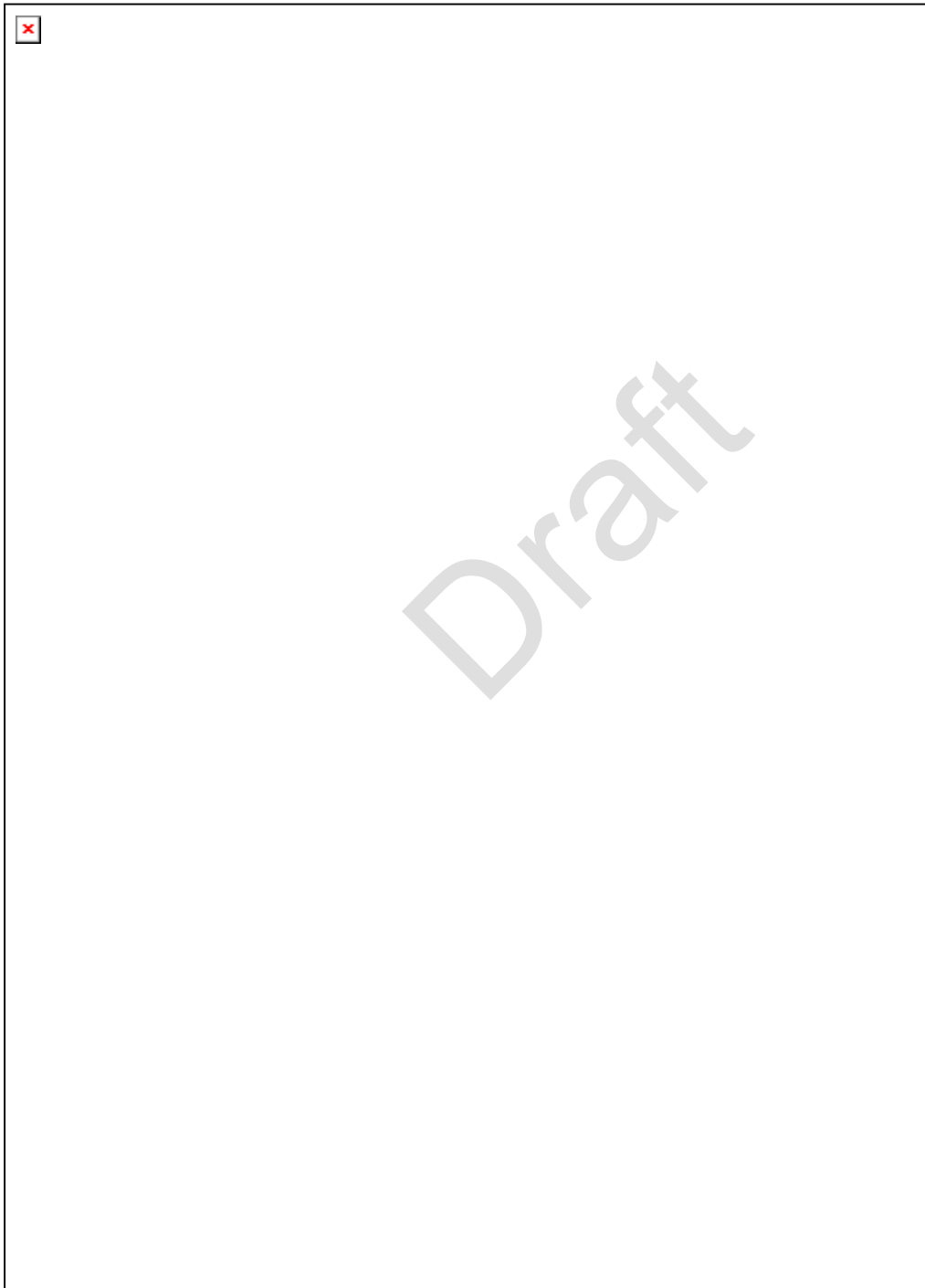
Common base class for single line text entry fields

Draft

3.2.4.2.75 wxTextCtrl

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.

The following figure uses the TextCtrl widget to simulate the appearance of the Operator's Control Panel for a Supervisory Control and Data Acquisition (SCADA) System. A description follows the figure.



- 1 Operating personnel require the display of mode, state and advisory messages. The following toolkit widgets could be applied:
 - Buttons to initiate application actions
 - Radio Boxes and Radio Buttons to set or clear application modes and settings
 - StaticText widgets to create column headings
 - TextCtrl widgets to create and update application mode and status information
- 2 Operating and Maintenance personnel require the dialogs for examining or modifying various administrative and maintenance options and settings. The following toolkit widgets could be applied:
 - Buttons to initiate administrative and maintenance actions
 - CheckBoxes to set or clear administrative and maintenance options
 - Radio Boxes and Radio Buttons to set or clear administrative and maintenance modes and settings
 - StaticText widgets to create column headings
 - TextCtrl widgets to create and update administrative and maintenance mode and status information

3.2.4.2.75.1 Positional & Key Word Argument Definitions

GUI Objects can also be created with Markup argument enhancements to the wxPython TextCtrl.

The positional arguments include:

- 1 **text** - One or more strings of characters:
 - a) **Single line mode** - Cannot contain embedded new lines ("\n")
 - b) **Multi-line mode** - Can contain embedded new lines ("\n"). A list of strings, embedded in brackets ("[]"), but without embedded new lines, may be used to enhance readability of source code.

The key word arguments include:

- 1 **markup** - When the argument is "None" the markup mode is "OFF" and the associated arguments are ignored.
 - a) **Attributes** - The example sequentially specifies that "wx.DISPLAY_BLINK", "wx.DISPLAY_BOLD", "wx.DISPLAY_DIM", "wx.DISPLAY_NORMAL", "wx.DISPLAY_REVERSE", "wx.DISPLAY_STANDOUT" and "wx.DISPLAY_UNDERLINE" mode be applied to the appropriate text message.
 - b) **Foreground** - Specifies the foreground color.
 - c) **Background** - Specifies the background color.
 - d) **newLine** - Specifies new line mode: When the argument is "True", the position of the text insertion pointer is set to Column 1 of the next Row inside the TextCtrl pane. NOTE: Any associated text argument will be ignored. When the argument is "False", any associated text is then appended to the previously inserted text inside the TextCtrl pane. The Column and Row settings are updated as appropriate to establish the next insertion position.
 - e) **topOfForm** - When the argument is "True", the position of the text insertion pointer is set to Column 1, Row 1 inside the TextCtrl pane. NOTE: Any associated text argument will be ignored.

2 wrap - When the argument is "False" the line wrap mode is "OFF" and the associated kew word arguments are ignored. When the argument is "True" the line wrap mode is "ON" and any associated text is then subjected to line and word wrapping using parameters associated with the following key word arguments:

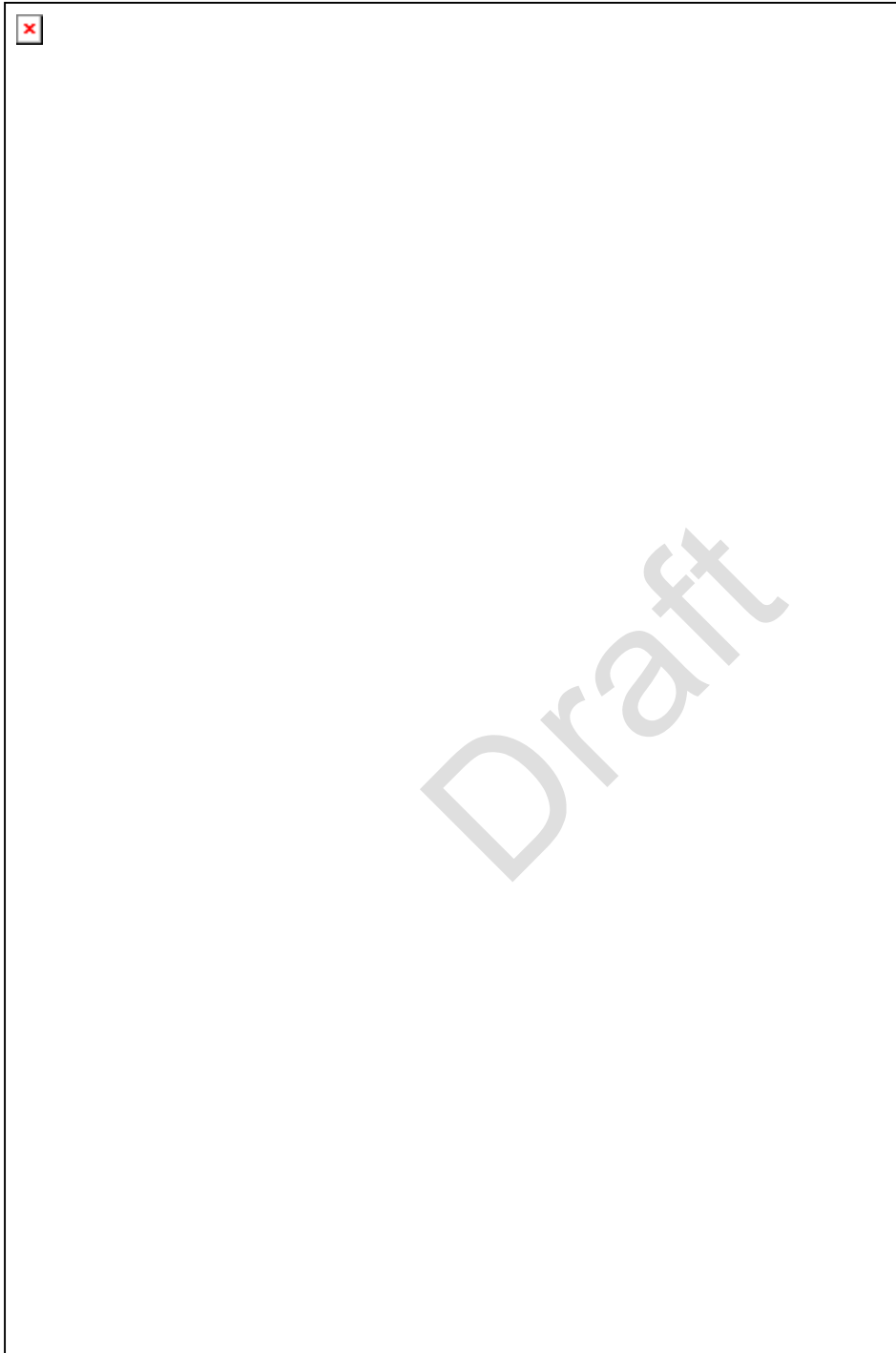
- a) **breakLongWords**=True
- b) **expandTabs**=True
- c) **fixSentenceEndings**=False
- d) **initialIndent**=""
- e) **replaceWhitespace**=False
- f) **subsequentIndent**=""
- g) **theLineNumberWidth**=0

3.2.4.2.75.1.1 TextCtrl Markup via Cygwin Terminal

The Cygwin Terminal support some attrdoes NOT support blinking. It substitutes a color for the underline. It substitutes reversed color for standout.

- 1 Foreground** - Attribute visibly changed foreground color.
- 2 Background** - Attribute visibly changed the background color.
- 3 Blink** - Attribute visibly does NOT blink but the background was changed from the specified "Black" to a another highlighted ("White") color.
- 4 Bold** - Attribute visibly brighter and bolder than Normal.
- 5 Dim** - Attribute visibly dimmer than Normal.
- 6 Normal** - Attribute visible.
- 7 Reverse** - Attribute visibly interchanged Foreground and Background colors.
- 8 Standout** - Attribute visibly interchanged Foreground and Background colors.

- 9 **Underline** - Attribute visibly does NOT underline but the foreground was changed from the specified "Red" to another color ("Cyan") colors.



3.2.4.2.75.1.2 TextCtrl Markup via Cygwin-X Terminal

The Cygwin-X Terminal supports some attributes. It does NOT support blinking. It substitutes reversed color for standout.

- 1 Foreground** - Attribute visibly changed foreground color.
- 2 Background** - Attribute visibly changed the background color.
- 3 Blink** - Attribute visibly blinks ON and OFF.
- 4 Bold** - Attribute visibly brighter and bolder than Normal.
- 5 Dim** - Attribute visibly dimmer than Normal.
- 6 Normal** - Attribute visible.
- 7 Reverse** - Attribute visibly interchanged Foreground and Background colors.
- 8 Standout** - Attribute visibly substituted the Reverse attribute.

Draft

9 Underline - Attribute visibly underlined text characters.



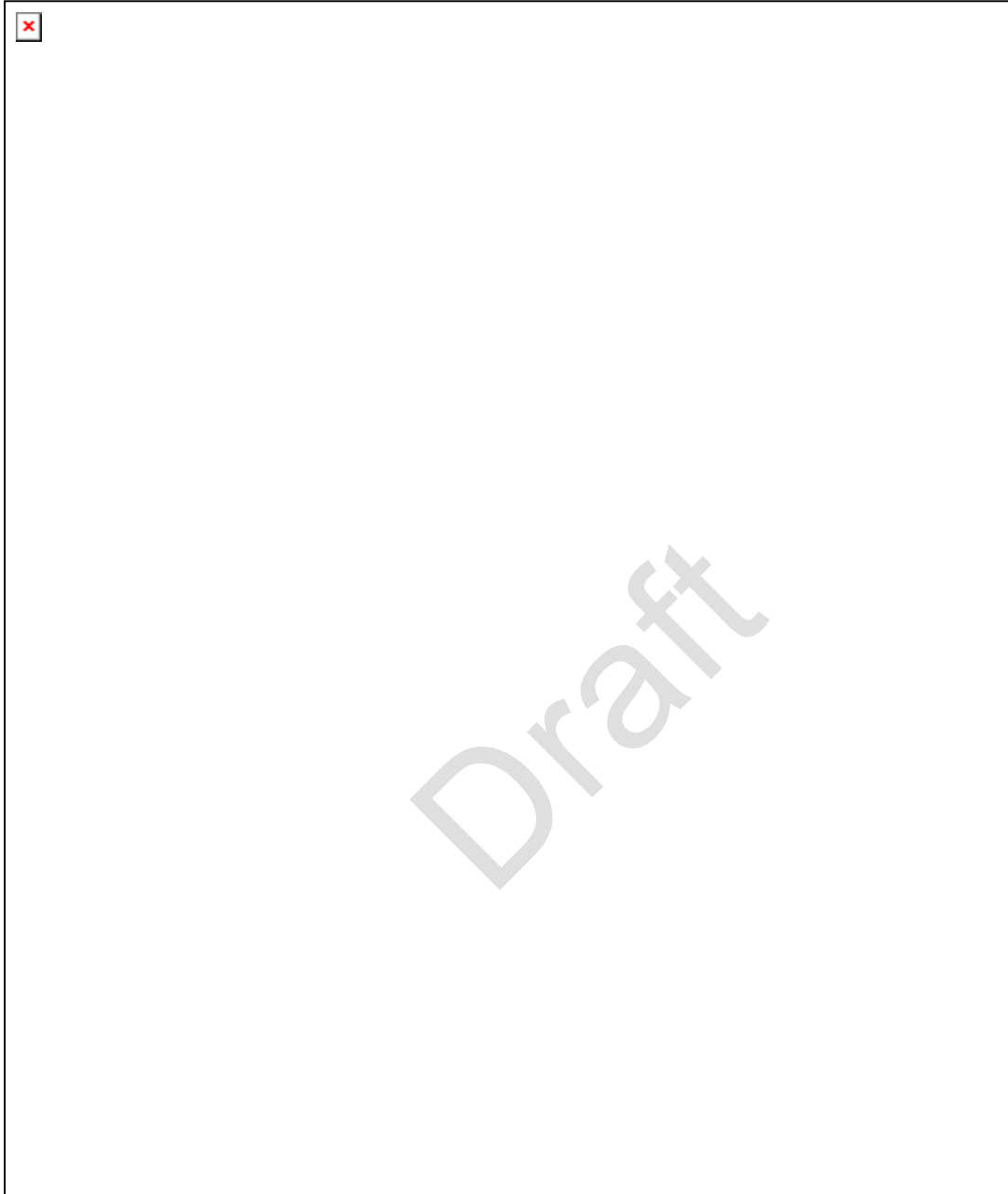
3.2.4.2.75.1.3 TextCtrl Markup via Cygwin-Mintty Terminal

The Cygwin-Mintty Terminal supports some attributes. It does NOT support blinking. It substitutes reversed color for standout.

- 1 Foreground** - Attribute visibly changed foreground color.
- 2 Background** - Attribute visibly changed the background color.
- 3 Blink** - Attribute visibly blinks ON and OFF.
- 4 Bold** - Attribute visibly brighter and bolder than Normal.
- 5 Dim** - Attribute visibly dimmer than Normal.
- 6 Normal** - Attribute visible.
- 7 Reverse** - Attribute visibly interchanged Foreground and Background colors.
- 8 Standout** - Attribute visibly substituted the Reverse attribute.

Draft

9 Underline - Attribute visibly underlined text characters.



3.2.4.2.75.2 Markup Key Word Argument Example

The following is an example of an application code fragment that defines and applies markup key word arguments with each invocation of the TextCtrl class's AppendText method:

Draft

```

▪ testCases = [(" [ BLINK   ] ", wx.DISPLAY_BLINK),
▪             (" [ BOLD    ] ", wx.DISPLAY_BOLD),
▪             (" [ DIM     ] ", wx.DISPLAY_DIM),
▪             (" [ NORMAL  ] ", wx.DISPLAY_NORMAL),
▪             (" [ STANDOUT ] ", wx.DISPLAY_STANDOUT),
▪             (" [ REVERSE ] ", wx.DISPLAY_REVERSE),
▪             (" [ UNDERLINE ] ", wx.DISPLAY_UNDERLINE)]
▪
▪ foreground = theText.GetForegroundColour() # wx.COLOR_CYAN
▪ background = theText.GetBackgroundColour()# wx.COLOR_MAGENTA
▪
▪ attrColor = theText.GetAttributeValueFromColorPair(
▪             foreground, background)
▪
▪ attrMarkupColor = theText.GetAttributeValueFromColorPair(
▪                 wx.COLOR_RED, wx.COLOR_BLACK)
▪
▪ i = 0
▪ for (msg, attr) in testCases:
▪
▪     markup = {"Foreground":wx.COLOR_RED,
▪               "Background":wx.COLOR_BLACK,
▪               "Attributes": [attr]}
▪
▪     comment = 'markup[%d]: msg=%s; attr=0x%X; attrColor=0x%X' % (
▪                 i, msg, attr, attrMarkupColor)
▪
▪     print(comment)
▪
▪     if (msg == " [ BLINK   ] "):
▪         theText.AppendText("", topOfForm=True, markup=markup)
▪         theText.AppendText(msg, newLine=False, markup=markup)
▪     elif (msg == " [ STANDOUT ] "):
▪         theText.AppendText("", newLine=True, markup=markup)
▪         theText.AppendText("", newLine=True, markup=markup)
▪         theText.AppendText(msg, newLine=False, markup=markup)
▪     else:
▪         theText.AppendText(msg, newLine=False, markup=markup)
▪
▪
▪ i += 1

```

3.2.4.2.76 wxToolBar (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.

3.2.4.2.77 wxTreeCtrl (TBD)

A control combining wxTreeCtrl and wxListCtrl features

3.2.4.2.78 wxVersionInfo (TBD)

WxVersionInfo contains version information

Draft

3.2.4.3 GUI Object Layout Services

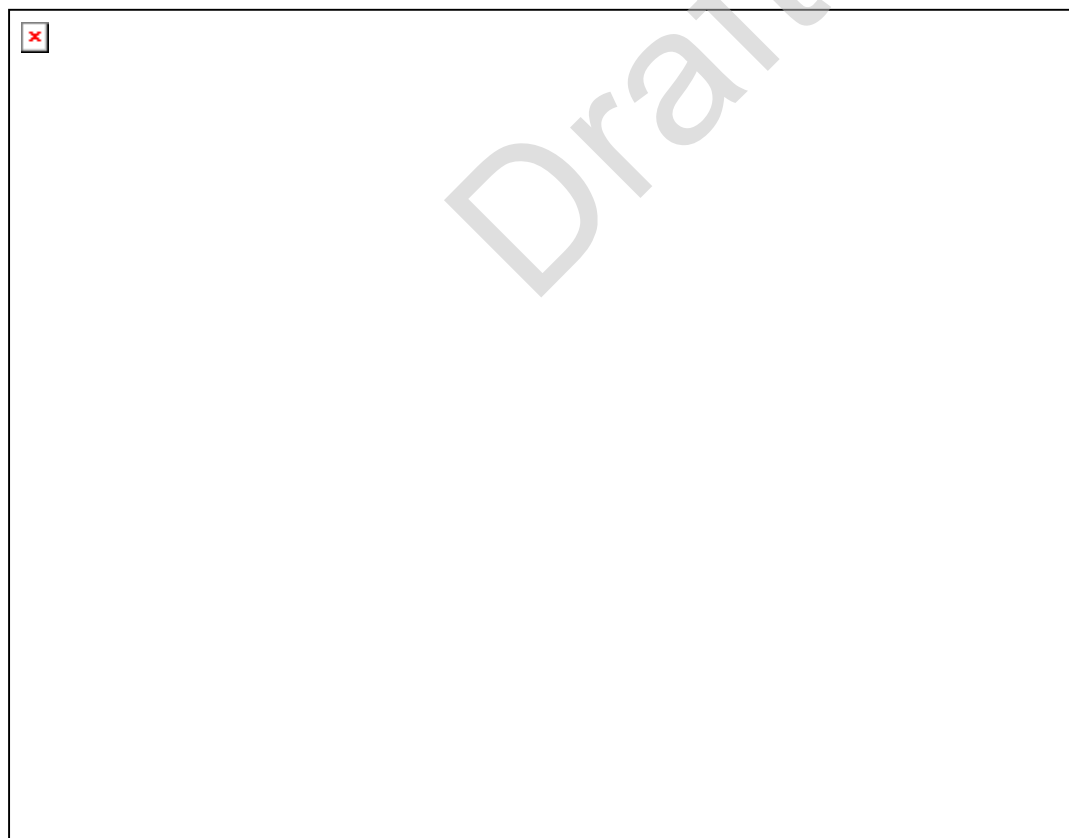
- *wxBoxSizer* (on page 205)
- *wxGridSizer* (on page 206)

3.2.4.3.1 wxBoxSizer

Automatically subdivides the containing GUI Object's client area into the number of rows (for vertical direction placement) or columns (for horizontal direction placement) specified by the application programmer. It then positions and scales the size of one or more associated GUI Objects to fit the assigned row or column location.

The following figure illustrates the two step process for subdividing a Frame's client area.

- 1 A Frame's client area is subdivided horizontally into two panes (two-thirds of the width assigned as the magenta pane on the left and the remaining one third assigned as the red pane on the right).
- 2 The magenta pane is subdivided vertically into three panes (one-third of the height assigned as the cyan pane on the top, one-third of the height assigned as the green area in the middle and one-third of the height assigned as the yellow on the bottom).



3.2.4.3.2 **wxFlexGridSizer** (TBD)

A flex grid sizer is a sizer which lays out its children in a two-dimensional table with all table fields in one row having the same height and all fields in one column having the same width, but all rows or all columns are not necessarily the same height or width as in the `wxGridSizer`

3.2.4.3.3 **wxGBPosition** (TBD)

This class represents the position of an item in a virtual grid of rows and columns managed by a `wxGridBagSizer`

3.2.4.3.4 **wxGBSizerItem** (TBD)

Used by the `wxGridBagSizer` for tracking the items in the sizer

3.2.4.3.5 **wxGBSpan** (TBD)

This class is used to hold the row and column spanning attributes of items in a `wxGridBagSizer`

3.2.4.3.6 **wxGridBagSizer** (TBD)

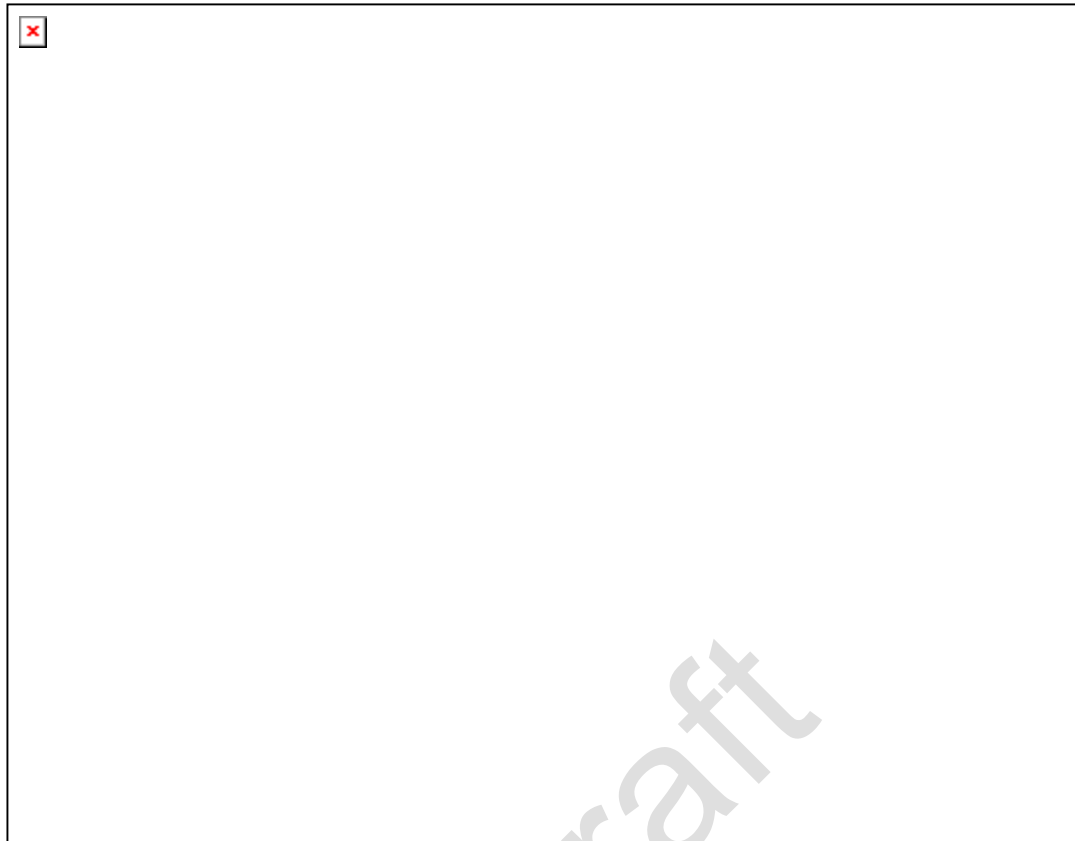
A `wxSizer` that can lay out items in a virtual grid like a `wxFlexGridSizer` but in this case explicit positioning of the items is allowed using `wxGBPosition`, and items can optionally span more than one row and/or column using `wxGBSpan`

3.2.4.3.7 **wxGridSizer**

Automatically subdivides the containing GUI Object's client area into the number of rows and columns of a grid specified by the application programmer. It then positions and scales the size of one or more associated GUI Objects to fit the assigned grid location.

The following figure illustrates the two step process for subdividing a Frame's client area.

- 1 A Frame's client area is assigned as the magenta pane to distinguish the area from the rest of the Frame.
- 2 The magenta pane is subdivided horizontally into four columns and vertically into four rows which are filled row by row and column by column within each row. The appearance is similar to that of a calculator keypad.



3.2.4.3.8 wxSizerFlags

Container for sizer items flags providing readable names for them

3.2.4.3.9 wxSizerItem

Used to track the position, size and other attributes of each item managed by a wxSizer

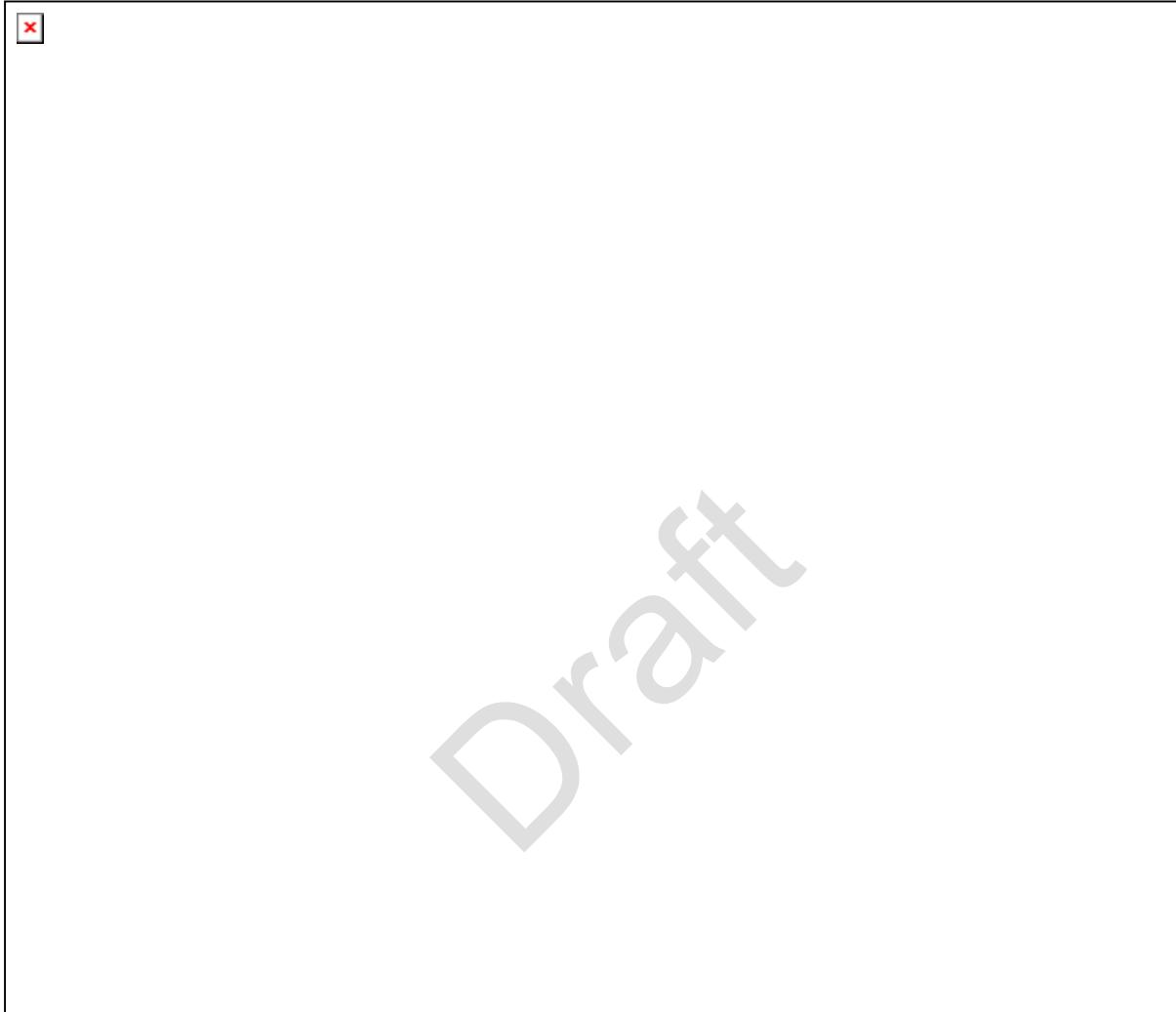
3.2.4.3.10 wxStaticBoxSizer

Automatically subdivides the containing GUI Object's client area into the number of rows (for vertical direction placement) or columns (for horizontal direction placement) specified by the application programmer. It then positions and scales the size of one or more associated GUI Objects to fit the assigned row or column location.

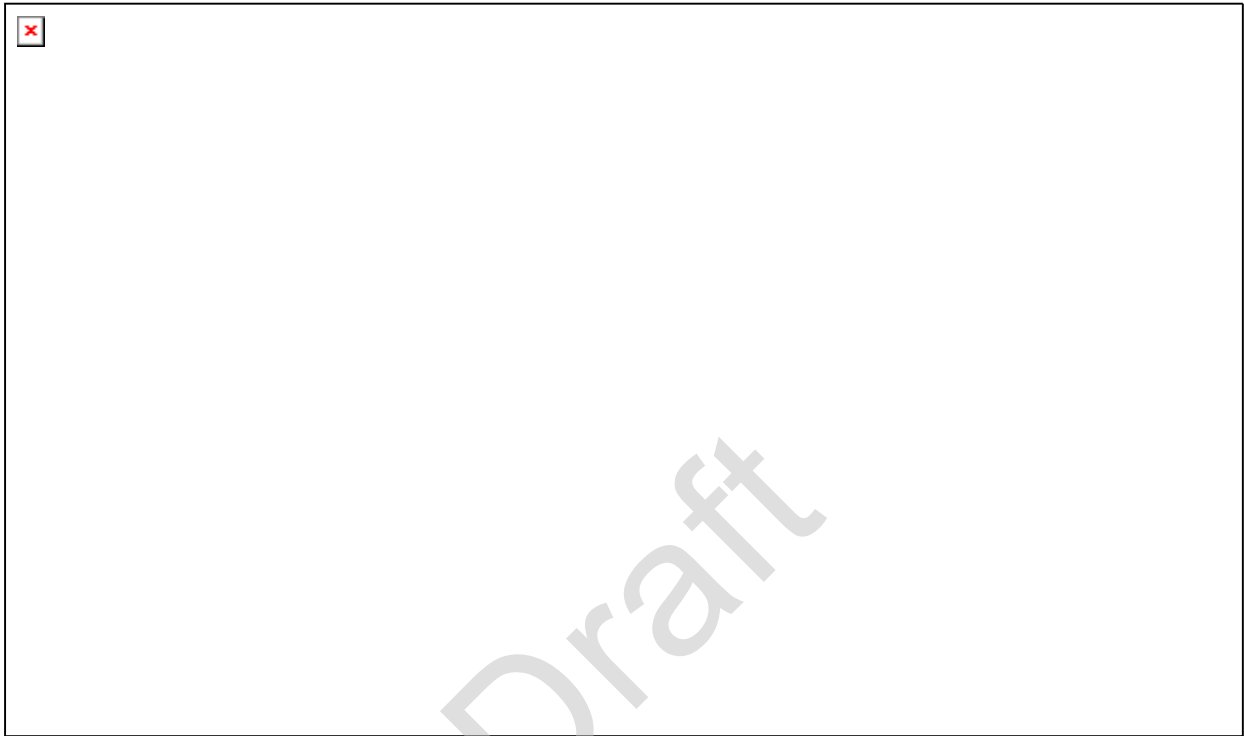
The following figure illustrates the two step process for subdividing a Frame's client area.

- 1 A Frame's client area is filled with a white panel named "myPanel".
- 2 The white panel is subdivided horizontally into three static box panes (one-third of the width assigned as the cyan pane, "Box 1", on the left, the green pane, "Box 2", in the middle and the yellow pane "Box 3" on the right).
- 3 Each of the static boxes are subdivided vertically into three red panes named "one", "two", "three", "four", "five", "six", "seven", "eight" and "nine". Each red pane is one-third of the height assigned the client area of its associated static box.

The following figure depicts compromises to the size of items within the static box in order to preserve the consistency of the borders by not overlapping any of them. The algorithm forces the width and height of each item to be integer multiples of a fixed size font character (8-w x 12-h pixels).



The following figure depicts the maximizing of the size of items within the static box by sacrificing the consistency of the borders through the overlapping of some of them. The algorithm does NOT force the width and height of each item to be integer multiples of a fixed size font character (8-w x 12-h pixels). Note: the nCurses-screen size is larger in this figure than in the previous one and the figure has been scaled down to fit the printed page.



3.2.4.3.11 wxStdDialogButtonSizer (TBD)

This class creates button layouts which conform to the standard button spacing and ordering defined by the platform or toolkit's user interface guidelines (if such things exist)

3.2.4.3.12 wxWrapSizer (TBD)

A wrap sizer lays out its items in a single line, like a box sizer -- as long as there is space available in that direction

Draft

3.2.4.4 GUI Object Attributes

The following "nCurses" character cell attributes are available:

- **Border** (on page 211) - Graphic-like symbols that surround the perimeter of a GUI Object.
- **Brightness** (on page 212) - Relative level of illumination from a GUI Object.
 - **Dim** (on page 212) - Selected by `DISPLAY_DIM = curses.A_DIM`
 - **Normal** (on page 212) - Selected by `DISPLAY_NORMAL = curses.A_NORMAL`
 - **Standout** (on page 212) - Selected by `DISPLAY_STANDOUT = curses.A_STANDOUT`
- **Color** (on page 213) - Hue, brightness and saturation of illumination from a GUI Object
 - **Reverse** (on page 215) - Selected by `DISPLAY_REVERSE = curses.A_REVERSE`
- **Font** (on page 216) - Set of type of one size and face of a GUI Object.
 - **Blink** (on page 217) - Selected by `DISPLAY_BLINK = curses.A_BLINK`
 - **Bold** (on page 217) - Selected by `DISPLAY_BOLD = curses.A_BOLD`
 - **Underline** (on page 217) - Selected by `DISPLAY_UNDERLINE = curses.A_UNDERLINE`

NOTE; These attributes are internally used to construct Frame titles and Button labels. However, it is not known how the wxPython API could be extended to provide applications with access to this capability. For example, the `TextCtrl.Append` method tasks its multi-line text input as a list or text with embedded new lines (`\n`). There currently is no provision for text markup such as HTML. An idea, perhaps the list of text strings might use tuples to pass attribute information for the associated text string.

3.2.4.4.1 Border

Graphic-like horizontal, vertical and corner symbols that optionally surround the perimeter of a GUI Object. A border conveys to the System Operator that the enclosed collection of GUI Objects function as an operationally related group.

3.2.4.4.2 Brightness

Relative level of illumination from a GUI Object.

The following GUI text object attributes are available:

- **Bold** (on page 217) - DISPLAY_BOLD (curses.A_BOLD)
- **Dim** (on page 212) - DISPLAY_DIM (curses.A_DIM)
- **Normal** (on page 212) - DISPLAY_NORMAL (curses.A_NORMAL)
- **Reverse** (on page 215) - DISPLAY_REVERSE (curses.A_REVERSE)
- **Standout** (on page 212) - DISPLAY_STANDOUT (curses.A_STANDOUT)

NOTE; These attributes are internally used to construct Frame titles and Button labels. However, it is not known how the wxPython API could be extended to provide applications with access to this capability. For example, the TextCtrl.Append method takes its multi-line text input as a list or text with embedded new lines (\n). There currently is no provision for text markup such as HTML. An idea, perhaps the list of text strings might use tuples to pass attribute information for the associated text string.

3.2.4.4.2.1 Dim

This attribute causes the display to reduce the brightness of the designated GUI object to less than Normal.

3.2.4.4.2.2 Normal

This attribute causes the display to set the brightness of the designated GUI object to Normal. It may also set the font thickness to Normal.

3.2.4.4.2.3 Standout

This attribute causes the display to increase the brightness of the designated GUI object to more than Normal. It may also increase the font thickness to Bold.

3.2.4.4.3 Color

Hue, brightness and saturation of illumination from a GUI Object.

- **Foreground** (see "**Color**" on page 213) - Color of "ink" in which the text is displayed.
- **Background** (see "**Color**" on page 213) - Color of "paper" on which the text is displayed.
- **Reverse** (on page 215) - Interchanged Foreground and Background Colors to highlight selected text.

Foreground and background color settings (such as yellow text on blue background) of the GUI object. The default curses 8-color set palette includes Black, Blue, Cyan, Green, Magenta, Red, Yellow and White.

The following dictionary maps the sixty-eight wxPython color set palette, guaranteed to be recognized, into the default curses 8-color set palette.

Note: This tabulation will not apply when the terminal type is xterm-256color. Needless to say, it will not apply for monochrome terminals such as ansi, vt100 or vt220.

- colorSubstitutionMap = {
- 'name': 'colorSubstitutionMap',
- COLOR_AQUAMARINE: COLOR_CYAN,
- COLOR_BLACK: COLOR_BLACK,
- COLOR_BLUE: COLOR_BLUE,
- COLOR_BLUE_VIOLET: COLOR_BLUE,
- COLOR_BROWN: COLOR_YELLOW,
- COLOR_CADET_BLUE: COLOR_BLUE,
- COLOR_CORAL: COLOR_RED,
- COLOR_CORNFLOWER_BLUE: COLOR_BLUE,
- COLOR_CYAN: COLOR_CYAN,
- COLOR_DARK_GRAY: COLOR_BLACK,
- COLOR_DARK_GREEN: COLOR_GREEN,
- COLOR_DARK_OLIVE_GREEN: COLOR_GREEN,
- COLOR_DARK_ORCHID: COLOR_MAGENTA,
- COLOR_DARK_SLATE_BLUE: COLOR_BLUE,
- COLOR_DARK_SLATE_GRAY: COLOR_BLACK,
- COLOR_DARK_TURQUOISE: COLOR_CYAN,
- COLOR_DIM_GRAY: COLOR_BLACK,
- COLOR_FIREBRICK: COLOR_RED,
- COLOR_FOREST_GREEN: COLOR_GREEN,
- COLOR_GOLD: COLOR_YELLOW,
- COLOR_GOLDENROD: COLOR_CYAN,
- COLOR_GRAY: COLOR_BLACK,
- COLOR_GREEN: COLOR_GREEN,

- COLOR_GREEN_YELLOW: COLOR_GREEN,
- COLOR_INDIAN_RED: COLOR_RED,
- COLOR_KHAKI: COLOR_YELLOW,
- COLOR_LIGHT_BLUE: COLOR_BLUE,
- COLOR_LIGHT_GRAY: COLOR_BLACK,
- COLOR_LIGHT_STEEL_BLUE: COLOR_BLUE,
- COLOR_LIME_GREEN: COLOR_GREEN,
- COLOR_MAGENTA: COLOR_MAGENTA,
- COLOR_MAROON: COLOR_RED,
- COLOR_MEDIUM_AQUAMARINE: COLOR_CYAN,
- COLOR_MEDIUM_BLUE: COLOR_BLUE,
- COLOR_MEDIUM_FOREST_GREEN: COLOR_GREEN,
- COLOR_MEDIUM_GOLDENROD: COLOR_YELLOW,
- COLOR_MEDIUM_ORCHID: COLOR_MAGENTA,
- COLOR_MEDIUM_SEA_GREEN: COLOR_GREEN,
- COLOR_MEDIUM_SLATE_BLUE: COLOR_BLUE,
- COLOR_MEDIUM_SPRING_GREEN: COLOR_GREEN,
- COLOR_MEDIUM_TURQUOISE: COLOR_CYAN,
- COLOR_MEDIUM_VIOLET_RED: COLOR_RED,
- COLOR_MIDNIGHT_BLUE: COLOR_BLUE,
- COLOR_NAVY: COLOR_BLUE,
- COLOR_ORANGE: COLOR_RED,
- COLOR_ORANGE_RED: COLOR_RED,
- COLOR_ORCHID: COLOR_MAGENTA,
- COLOR_PALE_GREEN: COLOR_GREEN,
- COLOR_PINK: COLOR_RED,
- COLOR_PLUM: COLOR_MAGENTA,
- COLOR_PURPLE: COLOR_RED,
- COLOR_RED: COLOR_RED,
- COLOR_SALMON: COLOR_RED,
- COLOR_SEA_GREEN: COLOR_GREEN,
- COLOR_SIENNA: COLOR_RED,
- COLOR_SKY_BLUE: COLOR_CYAN,
- COLOR_SLATE_BLUE: COLOR_BLUE,
- COLOR_SPRING_GREEN: COLOR_GREEN,
- COLOR_STEEL_BLUE: COLOR_BLUE,
- COLOR_TAN: COLOR_YELLOW,
- COLOR_THISTLE: COLOR_BLACK,

- COLOR_TURQUOISE: COLOR_CYAN,
- COLOR_VIOLET: COLOR_MAGENTA,
- COLOR_VIOLET_RED: COLOR_RED,
- COLOR_WHEAT: COLOR_YELLOW,
- COLOR_WHITE: COLOR_WHITE,
- COLOR_YELLOW: COLOR_YELLOW,
- COLOR_YELLOW_GREEN: COLOR_YELLOW
- }

3.2.4.4.3.1 Reverse

This attribute causes the display to exchanges Foreground and Background color settings of the selected GUI object from their original values.

3.2.4.4.4 Font

A font is a set of type that is of one size (e.g., 12 pt height) and face (e.g., Courier) of a GUI Object with the following display features:

- **Blink** (on page 217) - Visibility of the font turns OFF and ON in synchronization with each tick of the display's clock.
- **Bold** (on page 217) - Width of the font line segments are thicker than normal.
- **Underline** (on page 217) - Visibility of a reserved line of pixels used to generate the font are turned OFF or ON. The reserved row of pixels would typically be located below lower case characters such as "c" but above the descender part of lower case characters such as "g".

Early terminals had small display screens. The font, like the ones used on mechanical teletypes had characters of uniform width and height. A VT100 terminal, for example, supported 80 columns and 25 rows.

The original VGA Terminal, was designed to support text and graphical images. Its screen resolution of 640 x 480 pixels, would support 80 columns and 40 rows of text in its Courier 12pt fixed font. The fixed width and height for those characters and symbols was equivalent to 8 x 12 pixels.

The higher resolution terminals available today permit more information to be displayed and organized into multiple windows. The operator typically has control over the font type and size used in each window.

When a Terminal Emulator is used for a character-mode Graphical-style User Interface, it is necessary that the operator select a font (such a Courier) with the same fixed width and height for all characters and symbols.

Upon startup, the nCurses GUI-like Toolkit detects the screen's width and height in characters. Unable to determine the actual font, the wxPython emulation converts the screen size from character units to pixel users via the 8 x 12 pixels factor.

ATTRIBUTE	OPTIONS	AVAILABILITY	DESCRIPTION
font-size	<ul style="list-style-type: none"> ▪ 8 width x 12 height in pixels 	<ul style="list-style-type: none"> ▪ Yes 	The font size. Analogous to a 12pt Courier font on a VGA display.
font-style	<ul style="list-style-type: none"> ▪ Normal ▪ Italic ▪ Oblique 	<ul style="list-style-type: none"> ▪ Yes ▪ No ▪ No 	Style in which letters are rendered.
font-variant	<ul style="list-style-type: none"> ▪ Normal ▪ Small-caps 	<ul style="list-style-type: none"> ▪ Yes ▪ No 	The 'small-caps' makes all lower case letters render as small capitals.
font-weight	<ul style="list-style-type: none"> ▪ Thin ▪ Normal ▪ Bold 	<ul style="list-style-type: none"> ▪ No ▪ Yes ▪ Yes 	The thickness of the font.

3.2.4.4.1 Blink

This attribute causes the display to periodically turn the visibility of the designated GUI object OFF and ON in synchronization with each tick of the display's clock.

- When visibility is OFF, the display uses the GUI object's original Background Color for both Foreground and Background.
- When visibility is ON, the display use the GUI object's original Foreground and Background Colors.

3.2.4.4.2 Bold

3.2.4.4.3 Underline

This attribute causes the display to change the font for the selected GUI Object to the one having the row of pixels used to underscore the text.

Visibility of the reserved line of pixels used to generate the font are turned OFF or ON. The reserved row of pixels ("_") would typically be located below lower case characters such as "_c_" but above the descender part of lower case characters such as "_g_".

Draft

Draft

4 APPLICATION PROGRAMMING INTERFACE

- 1** *Command Line Interface API* (on page 220)
 - a) *Python Command Line Parsing API* (on page 220)
 - b) *Application Run Time Environment API* (on page 222)
 - c) *Command Line Interface API* (on page 220)
 - d) *Graphical User Interface API* (on page 249)
- 2** *Graphical User Interface API* (on page 249)
 - a) *nCurses API* (on page 250)
 - b) *Python Curses API* (on page 252)
 - c) *wxPython-Style API* (see "*wxPython API*" on page 255)
 - d) *tsWxGTUI API* (on page 259)

4.1 Command Line Interface API

From Wikipedia, the free encyclopedia

A command-line interface (CLI) 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.

This portion of the toolkit provides 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.

4.1.1 Python Command Line Parsing API

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.

Supports one or more of the parser module(s) available in the Python version(s) supported by the application:

- 1 ***argparse API*** (on page 221) (introduced with Python 2.7.0)
- 2 ***optparse API*** (on page 221) (introduced with Python 2.3.0)
- 3 ***getopt API*** (on page 221) (introduced with Python 1.6.0)

When used with Python 2.7, the "tsOperatorSettings.py" module (a "tsLibCLI" component of the "tsWxGTUI_PyVx" Toolkit) supports all three of the above mentioned parser module(s) as an experimental and educational opportunity.

However, when one seeks to backport applications to Python 2.0-2.6, the "tsOperatorSettings.py" module must be stripped of unsupported parser modules in order to prevent a program trap which will block the application from running.

4.1.1.1 argparse API

From Python Documentation for argparse:

The argparse module makes it easy to write user-friendly command-line interfaces. The program defines what arguments it requires, and argparse will figure out how to parse those out of `sys.argv`. The argparse module also automatically generates help and usage messages and issues errors when users give the program invalid arguments.

4.1.1.2 optparse API

From Python Documentation for optparse:

optparse is a more convenient, flexible, and powerful library for parsing command-line options than the old getopt module. optparse uses a more declarative style of command-line parsing: you create an instance of `OptionParser`, populate it with options, and parse the command line. optparse allows users to specify options in the conventional GNU/POSIX syntax, and additionally generates usage and help messages for you.

4.1.1.3 getopt API

From Python Documentation for getopt:

NOTE: The getopt module is a parser for command line options whose API is designed to be familiar to users of the C `getopt()` function. Users who are unfamiliar with the C `getopt()` function or who would like to write less code and get better help and error messages should consider using the argparse module instead.

This module helps scripts to parse the command line arguments in `sys.argv`. It supports the same conventions as the Unix `getopt()` function (including the special meanings of arguments of the form `'-'` and `'--'`). Long options similar to those supported by GNU software may be used as well via an optional third argument.

4.1.2 Application Run Time Environment API

The Application Run Time Environment API establishes and maintains a collection of classes, methods and resources for handling the following application program activities:

- ***tsCommandLineEnv*** (see "***tsCommandLineEnv API***" on page 225) - Establishes and maintains the pre- and post run time environment for those applications with Command Line Interfaces.
- ***tsWxMultiFrameEnv*** (see "***tsWxMultiFrameEnv API***" on page 227) - Establishes and maintains the pre- and post run time environment for those applications with Graphical User Interfaces that are launched via a Command Line Interface.

Draft

4.1.2.1 tsApplication API

From Wikipedia, the free encyclopedia

Application software is all the computer software that causes a computer to perform useful tasks (compare with Computer viruses) beyond the running of the computer itself. A specific instance of such software is called a software application, application or app.

The term is used to contrast such software with system software, which manages and integrates a computer's capabilities but does not directly perform tasks that benefit the user. The system software serves the application, which in turn serves the user.

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).

```
# #####
#
# myApp = tsAPP.TsApplication(
#
# #####
# # 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 # True = Enable
```

```

#####
#
#   # When appropriate, some applications also use the
#   # following Graphical-style User Interface Launch
#   # configuration item list:
#
#   guiMessageFilename=None,
#   guiMessageRedirect=True,
#   guiRequired=True,
#   guiTopLevelObject=_Communicate,
#   guiTopLevelObjectId=-1,
#   guiTopLevelObjectName='Sample',
#   guiTopLevelObjectParent='Sample',
#   guiTopLevelObjectStatusBar=None,
#   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.runMainApplication()

```

4.1.2.2 tsCommandLineEnv API

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.

The Command Line Environment API establishes and maintains a collection of classes, methods and resources for handling the following application program activities:

- 1 Command Line Argument Parsing** - Enables the application program to define what command line arguments it requires. It then figures out how to parse those out of "sys.argv". It automatically generates help and usage messages and issues errors when users give the program invalid arguments. It uses Python's platform.python_version method to determine whether to use Python's "argparse" module (for Python 2.7.0 or later) , "optparse" module for Python 2.32.0 or later) or "getopt" (for Python 1.6.0 or later) to ensure application portability across:

- Python 2.6.5 - Default on Cygwin 1.7.9-2 with Microsoft Windows XP
- Python 2.7.1 - Default on Mac OS 10.7.2 (Lion)
- Python-2.7.2 - Default on Ubuntu Linux 11.10 (Oneiric Ocelot)
- Python 3.2.2 - Option on Mac OS 10.7.2 (Lion)
- Python-3.2.2 - Option on Ubuntu Linux 11.10

Notes: With the introduction of "argparse", as of Python 2.7, "optparse" became deprecated. If backwards compatibility is NOT an issue, Python's new "argparse" module could be used instead for Python 2.7 through 3.2.

- 2 Command Line Launching** - Enables the application program to define its entry point for the builtin launcher.
- 3 Command Line Logging** - Enables the application program to define the devices and priority threshold for outputting messages to the operator and file archiver. It automatically creates archive directories and files. The loggers automatically date and time stamp each output record.
- 4 Command Line Exception Handling** - Enables the application program to define and handle foreseeable problems. Non-recoverable ones are handled by the environment so as to restore, when possible, the platform for subsequent use, output a descriptive error message and return the appropriate exit code which can be used to automate other application programs.
- 5 Command Line Terminating** - Enables the application program to automatically shutdown.

```

#####
#
# 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 # True = Enable
#
# #####
#
# # 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()
```

4.1.2.3 tsWxMultiFrameEnv API

The Multi-Frame Environment API establishes and maintains a collection of classes, methods and resources for handling the following application program activities:

- 1 Task Bar Frame Launching** - An automatically generated frame that enables the System Operator to select which application frame receives focus as the top-level window. This is particularly useful when the terminal display size and resolution are severely limited and windows overlap each other.
- 2 Redirected Output Frame Launching** - An automatically generated frame that enables the System Operator to view the most recent progress and status messages.
- 3 Application Frame(s) Launching** - One or more application generated frames and dialogs that provide the System Operator with a user friendly, graphical interface.
- 4 Redirected Output Logging** - An automatically or application generated log file that enables the System Operator to view the chronological record of progress and status messages.

```

# #####
#
# 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 # True = Enable
#
# #####
#
# # When appropriate, some applications also use the
# # following Graphical-style User Interface Launch
# # configuration item list:
#
# guiMessageFilename=None,
# guiMessageRedirect=True,
# guiRequired=True,
# guiTopLevelObject=_Communicate,
# guiTopLevelObjectId=-1,
# guiTopLevelObjectName='Sample',
# guiTopLevelObjectParent='Sample',
# guiTopLevelObjectStatusBar=None,
# 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()

```

4.1.3 tsCommandLineInterface

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.

4.1.4 tsConfigObject (Obsolete)

From Wikipedia, the free encyclopedia

In computing, configuration files, or config files configure the initial settings for some computer programs. They are used for user applications, server processes and operating system settings. The files are often written in ASCII (rarely UTF-8) and line-oriented, with lines terminated by a newline or carriage return/line feed pair, depending on the operating system. They may be considered a simple database.

Some applications provide tools to create, modify, and verify the syntax of their configuration files; these sometimes have graphical interfaces. For other programs, system administrators may be expected to create and modify files by hand using a text editor. For server processes and operating-system settings, there is often no standard tool, but operating systems may provide their own graphical interfaces such as YaST or debconf.

Some computer programs only read their configuration files at startup. Others periodically check the configuration files for changes. Users can instruct some programs to re-read the configuration files and apply the changes to the current process, or indeed to read arbitrary files as a configuration file. There are no definitive standards or strong conventions.

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

4.1.5 tsDataBase (Future)

From Wikipedia, the free encyclopedia

A database is an organized collection of data. The data is typically organized to model relevant aspects of reality (for example, the availability of rooms in hotels), in a way that supports processes requiring this information (for example, finding a hotel with vacancies).

Database management systems (DBMSs) are specially designed applications that interact with the user, other applications, and the database itself to capture and analyze data. A general-purpose database management system (DBMS) is a software system designed to allow the definition, creation, querying, update, and administration of databases. Well-known DBMSs include MySQL, PostgreSQL, SQLite, Microsoft SQL Server, Microsoft Access, Oracle, SAP, dBASE, FoxPro, IBM DB2 and FilemakerPro. A database is not generally portable across different DBMS, but different DBMSs can inter-operate by using standards such as SQL and ODBC or JDBC to allow a single application to work with more than one database.

Class candidates include buzrug and PyDbLite.

4.1.6 tsDecorator (Future)

From Wikipedia, the free encyclopedia:

A decorator is any callable Python object that is used to modify a function, method or class definition.

A decorator is passed the original object being defined and returns a modified object, which is then bound to the name in the definition. Python decorators were inspired in part by Java annotations, and have a similar syntax; the decorator syntax is pure syntactic sugar, using `@` as the keyword.

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.

4.1.7 tsExceptions

From Python Documentation:

Even if a statement or expression is syntactically correct, it may cause an error when an attempt is made to execute it. Errors detected during execution are called "exceptions" and are not unconditionally fatal.... Most exceptions are not handled by programs, however, and result in error messages....

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

- 1** The initial configuration allocates the upper two-bits for the following four exception classes:
 - a) Program Exceptions - Handles application software logic, syntax and run time errors
 - b) Input Output Exceptions - Handles application file and device access, data transfer and capacity errors.
 - c) User Interface Exceptions - Handles application command line interface and graphical user interface operator errors.
 - d) Diagnostic Exceptions - Handles hardware and software troubleshooting errors.
 - e) The numeric value of the exception ID is not significant. Changing the exception name would require testing and debugging the "tsWxGTUI_PyVx" Toolkit.
- 2** The lower six-bits afford each exception class the opportunity to define up to 64 unique error messages.
 - a) The "tsWxGTUI_PyVx" Toolkit has predefined a number of error messages that may be shared by the application.
 - b) Those error messages identified as "UNUSED_xx" are available for exclusive use of the application.
 - c) The numeric value of the error ID is not significant. Changing the error name would require testing and debugging the "tsWxGTUI_PyVx" Toolkit.
- 3** The numeric value of the exit code must range from 0 (No Error) to 255 (Unknown Error).

4 The Software User's Manual must identify and describe each exit code.

4.1.7.1 Exception Classes, Error Messages & Exit Codes

Exception (Class ID)	Error (Message ID)	Exit Code (ID)
Program Exception (0);	No Error (0);	ExitCode (0).
Program Exception (0);	Abort By Operator (1);	ExitCode (1).
Program Exception (0);	Application Trap (2);	ExitCode (2).
Program Exception (0);	Argument Key Not Valid (3);	ExitCode (3).
Program Exception (0);	Argument Type Not Valid (4);	ExitCode (4).
Program Exception (0);	Argument Value Not Valid (5);	ExitCode (5).
Program Exception (0);	Arithmetic Error (6);	ExitCode (6).
Program Exception (0);	Executable Image Does Not Exist (7);	ExitCode (7).
Program Exception (0);	Executable Image Not Valid (8);	ExitCode (8).
Program Exception (0);	Executable Instruction Not Valid (9);	ExitCode (9).
Program Exception (0);	Format Error (10);	ExitCode (10).
Program Exception (0);	Logic Error (11);	ExitCode (11).
Program Exception (0);	OS Error (12);	ExitCode (12).
Program Exception (0);	Syntax Error (13);	ExitCode (13).
Program Exception (0);	Timeout Error (14);	ExitCode (14).
Program Exception (0);	UNUSED_15 (15);	ExitCode (15).
Program Exception (0);	UNUSED_16 (16);	ExitCode (16).
Program Exception (0);	UNUSED_17 (17);	ExitCode (17).
Program Exception (0);	UNUSED_18 (18);	ExitCode (18).
Program Exception (0);	UNUSED_19 (19);	ExitCode (19).
Program Exception (0);	UNUSED_20 (20);	ExitCode (20).
Program Exception (0);	UNUSED_21 (21);	ExitCode (21).
Program Exception (0);	UNUSED_22 (22);	ExitCode (22).
Program Exception (0);	UNUSED_23 (23);	ExitCode (23).
Program Exception (0);	UNUSED_24 (24);	ExitCode (24).
Program Exception (0);	UNUSED_25 (25);	ExitCode (25).
Program Exception (0);	UNUSED_26 (26);	ExitCode (26).
Program Exception (0);	UNUSED_27 (27);	ExitCode (27).
Program Exception (0);	UNUSED_28 (28);	ExitCode (28).
Program Exception (0);	UNUSED_29 (29);	ExitCode (29).

Program Exception (0);	UNUSED_30 (30);	ExitCode (30).
Program Exception (0);	UNUSED_31 (31);	ExitCode (31).
Program Exception (0);	UNUSED_32 (32);	ExitCode (32).
Program Exception (0);	UNUSED_33 (33);	ExitCode (33).
Program Exception (0);	UNUSED_34 (34);	ExitCode (34).
Program Exception (0);	UNUSED_35 (35);	ExitCode (35).
Program Exception (0);	UNUSED_36 (36);	ExitCode (36).
Program Exception (0);	UNUSED_37 (37);	ExitCode (37).
Program Exception (0);	UNUSED_38 (38);	ExitCode (38).
Program Exception (0);	UNUSED_39 (39);	ExitCode (39).
Program Exception (0);	UNUSED_40 (40);	ExitCode (40).
Program Exception (0);	UNUSED_41 (41);	ExitCode (41).
Program Exception (0);	UNUSED_42 (42);	ExitCode (42).
Program Exception (0);	UNUSED_43 (43);	ExitCode (43).
Program Exception (0);	UNUSED_44 (44);	ExitCode (44).
Program Exception (0);	UNUSED_45 (45);	ExitCode (45).
Program Exception (0);	UNUSED_46 (46);	ExitCode (46).
Program Exception (0);	UNUSED_47 (47);	ExitCode (47).
Program Exception (0);	UNUSED_48 (48);	ExitCode (48).
Program Exception (0);	UNUSED_49 (49);	ExitCode (49).
Program Exception (0);	UNUSED_50 (50);	ExitCode (50).
Program Exception (0);	UNUSED_51 (51);	ExitCode (51).
Program Exception (0);	UNUSED_52 (52);	ExitCode (52).
Program Exception (0);	UNUSED_53 (53);	ExitCode (53).
Program Exception (0);	UNUSED_54 (54);	ExitCode (54).
Program Exception (0);	UNUSED_55 (55);	ExitCode (55).
Program Exception (0);	UNUSED_56 (56);	ExitCode (56).
Program Exception (0);	UNUSED_57 (57);	ExitCode (57).
Program Exception (0);	UNUSED_58 (58);	ExitCode (58).
Program Exception (0);	UNUSED_59 (59);	ExitCode (59).
Program Exception (0);	UNUSED_60 (60);	ExitCode (60).
Program Exception (0);	UNUSED_61 (61);	ExitCode (61).
Program Exception (0);	UNUSED_62 (62);	ExitCode (62).
Program Exception (0);	UNUSED_63 (63);	ExitCode (63).

Input Output Exception (1);	File Create Access Not Permitted (0);	ExitCode (64).
Input Output Exception (1);	File Delete Access Not Permitted (1);	ExitCode (65).
Input Output Exception (1);	File Directory Not Found (2);	ExitCode (66).
Input Output Exception (1);	File Execute Access Not Permitted (3);	ExitCode (67).
Input Output Exception (1);	File Not Found (4);	ExitCode (68).
Input Output Exception (1);	File Read Access Not Permitted (5);	ExitCode (69).
Input Output Exception (1);	File Storage Media Full (6);	ExitCode (70).
Input Output Exception (1);	File Write Access Not Permitted (7);	ExitCode (71).
Input Output Exception (1);	Interrupt Already In Use (8);	ExitCode (72).
Input Output Exception (1);	Interrupt Does Not Exist (9);	ExitCode (73).
Input Output Exception (1);	Interrupt Receiver Not Ready (10);	ExitCode (74).
Input Output Exception (1);	Interrupt Transmitter Not Ready (11);	ExitCode (75).
Input Output Exception (1);	IO Error (12);	ExitCode (76).
Input Output Exception (1);	Signal Already In Use (13);	ExitCode (77).
Input Output Exception (1);	Signal Does Not Exist (14);	ExitCode (78).
Input Output Exception (1);	Signal Receiver Not Ready (15);	ExitCode (79).
Input Output Exception (1);	Signal Transmitter Not Ready (16);	ExitCode (80).
Input Output Exception (1);	Spurious Interrupt Received (17);	ExitCode (81).
Input Output Exception (1);	Spurious Signal Received (18);	ExitCode (82).
Input Output Exception (1);	UNUSED_19 (19);	ExitCode (83).
Input Output Exception (1);	UNUSED_20 (20);	ExitCode (84).
Input Output Exception (1);	UNUSED_21 (21);	ExitCode (85).
Input Output Exception (1);	UNUSED_22 (22);	ExitCode (86).
Input Output Exception (1);	UNUSED_23 (23);	ExitCode (87).
Input Output Exception (1);	UNUSED_24 (24);	ExitCode (88).
Input Output Exception (1);	UNUSED_25 (25);	ExitCode (89).
Input Output Exception (1);	UNUSED_26 (26);	ExitCode (90).
Input Output Exception (1);	UNUSED_27 (27);	ExitCode (91).
Input Output Exception (1);	UNUSED_28 (28);	ExitCode (92).
Input Output Exception (1);	UNUSED_29 (29);	ExitCode (93).
Input Output Exception (1);	UNUSED_30 (30);	ExitCode (94).
Input Output Exception (1);	UNUSED_31 (31);	ExitCode (95).
Input Output Exception (1);	UNUSED_32 (32);	ExitCode (96).
Input Output Exception (1);	UNUSED_33 (33);	ExitCode (97).

Input Output Exception (1);	UNUSED_34 (34);	ExitCode (98).
Input Output Exception (1);	UNUSED_35 (35);	ExitCode (99).
Input Output Exception (1);	UNUSED_36 (36);	ExitCode (100).
Input Output Exception (1);	UNUSED_37 (37);	ExitCode (101).
Input Output Exception (1);	UNUSED_38 (38);	ExitCode (102).
Input Output Exception (1);	UNUSED_39 (39);	ExitCode (103).
Input Output Exception (1);	UNUSED_40 (40);	ExitCode (104).
Input Output Exception (1);	UNUSED_41 (41);	ExitCode (105).
Input Output Exception (1);	UNUSED_42 (42);	ExitCode (106).
Input Output Exception (1);	UNUSED_43 (43);	ExitCode (107).
Input Output Exception (1);	UNUSED_44 (44);	ExitCode (108).
Input Output Exception (1);	UNUSED_45 (45);	ExitCode (109).
Input Output Exception (1);	UNUSED_46 (46);	ExitCode (110).
Input Output Exception (1);	UNUSED_47 (47);	ExitCode (111).
Input Output Exception (1);	UNUSED_48 (48);	ExitCode (112).
Input Output Exception (1);	UNUSED_49 (49);	ExitCode (113).
Input Output Exception (1);	UNUSED_50 (50);	ExitCode (114).
Input Output Exception (1);	UNUSED_51 (51);	ExitCode (115).
Input Output Exception (1);	UNUSED_52 (52);	ExitCode (116).
Input Output Exception (1);	UNUSED_53 (53);	ExitCode (117).
Input Output Exception (1);	UNUSED_54 (54);	ExitCode (118).
Input Output Exception (1);	UNUSED_55 (55);	ExitCode (119).
Input Output Exception (1);	UNUSED_56 (56);	ExitCode (120).
Input Output Exception (1);	UNUSED_57 (57);	ExitCode (121).
Input Output Exception (1);	UNUSED_58 (58);	ExitCode (122).
Input Output Exception (1);	UNUSED_59 (59);	ExitCode (123).
Input Output Exception (1);	UNUSED_60 (60);	ExitCode (124).
Input Output Exception (1);	UNUSED_61 (61);	ExitCode (125).
Input Output Exception (1);	UNUSED_62 (62);	ExitCode (126).
Input Output Exception (1);	UNUSED_63 (63);	ExitCode (127).
User Interface Exception (2);	Character Graphics Not Available (0);	ExitCode (128).
User Interface Exception (2);	Command Line Argument Does Not Exist (1);	ExitCode (129).
User Interface Exception (2);	Command Line Argument Not Unique (2);	ExitCode (130).
User Interface Exception (2);	Command Line Argument Not Valid (3);	ExitCode (131).

User Interface Exception (2);	Command Line Operation Does Not Exist (4);	ExitCode (132).
User Interface Exception (2);	Command Line Operation Not Valid (5);	ExitCode (133).
User Interface Exception (2);	Command Line Option Does Not Exist (6);	ExitCode (134).
User Interface Exception (2);	Command Line Option Not Valid (7);	ExitCode (135).
User Interface Exception (2);	Command Line Switch Does Not Exist (8);	ExitCode (136).
User Interface Exception (2);	Command Line Switch Not Valid (9);	ExitCode (137).
User Interface Exception (2);	Command Line Syntax Not Valid (10);	ExitCode (138).
User Interface Exception (2);	Graphical Button Not Valid (11);	ExitCode (139).
User Interface Exception (2);	Graphical Dialog Not Valid (12);	ExitCode (140).
User Interface Exception (2);	Graphical Menu Not Valid (13);	ExitCode (141).
User Interface Exception (2);	Graphical Window Resized (14);	ExitCode (142).
User Interface Exception (2);	Graphical Window Does Not Exist (15);	ExitCode (143).
User Interface Exception (2);	Graphical Window Not Valid (16);	ExitCode (144).
User Interface Exception (2);	UNUSED_17 (17);	ExitCode (145).
User Interface Exception (2);	UNUSED_18 (18);	ExitCode (146).
User Interface Exception (2);	UNUSED_19 (19);	ExitCode (147).
User Interface Exception (2);	UNUSED_20 (20);	ExitCode (148).
User Interface Exception (2);	UNUSED_21 (21);	ExitCode (149).
User Interface Exception (2);	UNUSED_22 (22);	ExitCode (150).
User Interface Exception (2);	UNUSED_23 (23);	ExitCode (151).
User Interface Exception (2);	UNUSED_24 (24);	ExitCode (152).
User Interface Exception (2);	UNUSED_25 (25);	ExitCode (153).
User Interface Exception (2);	UNUSED_26 (26);	ExitCode (154).
User Interface Exception (2);	UNUSED_27 (27);	ExitCode (155).
User Interface Exception (2);	UNUSED_28 (28);	ExitCode (156).
User Interface Exception (2);	UNUSED_29 (29);	ExitCode (157).
User Interface Exception (2);	UNUSED_30 (30);	ExitCode (158).
User Interface Exception (2);	UNUSED_31 (31);	ExitCode (159).
User Interface Exception (2);	UNUSED_32 (32);	ExitCode (160).
User Interface Exception (2);	UNUSED_33 (33);	ExitCode (161).
User Interface Exception (2);	UNUSED_34 (34);	ExitCode (162).
User Interface Exception (2);	UNUSED_35 (35);	ExitCode (163).
User Interface Exception (2);	UNUSED_36 (36);	ExitCode (164).
User Interface Exception (2);	UNUSED_37 (37);	ExitCode (165).

User Interface Exception (2);	UNUSED_38 (38);	ExitCode (166).
User Interface Exception (2);	UNUSED_39 (39);	ExitCode (167).
User Interface Exception (2);	UNUSED_40 (40);	ExitCode (168).
User Interface Exception (2);	UNUSED_41 (41);	ExitCode (169).
User Interface Exception (2);	UNUSED_42 (42);	ExitCode (170).
User Interface Exception (2);	UNUSED_43 (43);	ExitCode (171).
User Interface Exception (2);	UNUSED_44 (44);	ExitCode (172).
User Interface Exception (2);	UNUSED_45 (45);	ExitCode (173).
User Interface Exception (2);	UNUSED_46 (46);	ExitCode (174).
User Interface Exception (2);	UNUSED_47 (47);	ExitCode (175).
User Interface Exception (2);	UNUSED_48 (48);	ExitCode (176).
User Interface Exception (2);	UNUSED_49 (49);	ExitCode (177).
User Interface Exception (2);	UNUSED_50 (50);	ExitCode (178).
User Interface Exception (2);	UNUSED_51 (51);	ExitCode (179).
User Interface Exception (2);	UNUSED_52 (52);	ExitCode (180).
User Interface Exception (2);	UNUSED_53 (53);	ExitCode (181).
User Interface Exception (2);	UNUSED_54 (54);	ExitCode (182).
User Interface Exception (2);	UNUSED_55 (55);	ExitCode (183).
User Interface Exception (2);	UNUSED_56 (56);	ExitCode (184).
User Interface Exception (2);	UNUSED_57 (57);	ExitCode (185).
User Interface Exception (2);	UNUSED_58 (58);	ExitCode (186).
User Interface Exception (2);	UNUSED_59 (59);	ExitCode (187).
User Interface Exception (2);	UNUSED_60 (60);	ExitCode (188).
User Interface Exception (2);	UNUSED_61 (61);	ExitCode (189).
User Interface Exception (2);	UNUSED_62 (62);	ExitCode (190).
User Interface Exception (2);	UNUSED_63 (63);	ExitCode (191).
Diagnostic Exception (3);	UNUSED_00 (0);	ExitCode (192).
Diagnostic Exception (3);	UNUSED_01 (1);	ExitCode (193).
Diagnostic Exception (3);	UNUSED_02 (2);	ExitCode (194).
Diagnostic Exception (3);	UNUSED_03 (3);	ExitCode (195).
Diagnostic Exception (3);	UNUSED_04 (4);	ExitCode (196).
Diagnostic Exception (3);	UNUSED_05 (5);	ExitCode (197).
Diagnostic Exception (3);	UNUSED_06 (6);	ExitCode (198).
Diagnostic Exception (3);	UNUSED_07 (7);	ExitCode (199).

Diagnostic Exception (3);	UNUSED_08 (8);	ExitCode (200).
Diagnostic Exception (3);	UNUSED_09 (9);	ExitCode (201).
Diagnostic Exception (3);	UNUSED_10 (10);	ExitCode (202).
Diagnostic Exception (3);	UNUSED_11 (11);	ExitCode (203).
Diagnostic Exception (3);	UNUSED_12 (12);	ExitCode (204).
Diagnostic Exception (3);	UNUSED_13 (13);	ExitCode (205).
Diagnostic Exception (3);	UNUSED_14 (14);	ExitCode (206).
Diagnostic Exception (3);	UNUSED_15 (15);	ExitCode (207).
Diagnostic Exception (3);	UNUSED_16 (16);	ExitCode (208).
Diagnostic Exception (3);	UNUSED_17 (17);	ExitCode (209).
Diagnostic Exception (3);	UNUSED_18 (18);	ExitCode (210).
Diagnostic Exception (3);	UNUSED_19 (19);	ExitCode (211).
Diagnostic Exception (3);	UNUSED_20 (20);	ExitCode (212).
Diagnostic Exception (3);	UNUSED_21 (21);	ExitCode (213).
Diagnostic Exception (3);	UNUSED_22 (22);	ExitCode (214).
Diagnostic Exception (3);	UNUSED_23 (23);	ExitCode (215).
Diagnostic Exception (3);	UNUSED_24 (24);	ExitCode (216).
Diagnostic Exception (3);	UNUSED_25 (25);	ExitCode (217).
Diagnostic Exception (3);	UNUSED_26 (26);	ExitCode (218).
Diagnostic Exception (3);	UNUSED_27 (27);	ExitCode (219).
Diagnostic Exception (3);	UNUSED_28 (28);	ExitCode (220).
Diagnostic Exception (3);	UNUSED_29 (29);	ExitCode (221).
Diagnostic Exception (3);	UNUSED_30 (30);	ExitCode (222).
Diagnostic Exception (3);	UNUSED_31 (31);	ExitCode (223).
Diagnostic Exception (3);	UNUSED_32 (32);	ExitCode (224).
Diagnostic Exception (3);	UNUSED_33 (33);	ExitCode (225).
Diagnostic Exception (3);	UNUSED_34 (34);	ExitCode (226).
Diagnostic Exception (3);	UNUSED_35 (35);	ExitCode (227).
Diagnostic Exception (3);	UNUSED_36 (36);	ExitCode (228).
Diagnostic Exception (3);	UNUSED_37 (37);	ExitCode (229).
Diagnostic Exception (3);	UNUSED_38 (38);	ExitCode (230).
Diagnostic Exception (3);	UNUSED_39 (39);	ExitCode (231).
Diagnostic Exception (3);	UNUSED_40 (40);	ExitCode (232).
Diagnostic Exception (3);	UNUSED_41 (41);	ExitCode (233).

Diagnostic Exception (3);	UNUSED_42 (42);	ExitCode (234).
Diagnostic Exception (3);	UNUSED_43 (43);	ExitCode (235).
Diagnostic Exception (3);	UNUSED_44 (44);	ExitCode (236).
Diagnostic Exception (3);	UNUSED_45 (45);	ExitCode (237).
Diagnostic Exception (3);	UNUSED_46 (46);	ExitCode (238).
Diagnostic Exception (3);	UNUSED_47 (47);	ExitCode (239).
Diagnostic Exception (3);	UNUSED_48 (48);	ExitCode (240).
Diagnostic Exception (3);	UNUSED_49 (49);	ExitCode (241).
Diagnostic Exception (3);	UNUSED_50 (50);	ExitCode (242).
Diagnostic Exception (3);	UNUSED_51 (51);	ExitCode (243).
Diagnostic Exception (3);	UNUSED_52 (52);	ExitCode (244).
Diagnostic Exception (3);	UNUSED_53 (53);	ExitCode (245).
Diagnostic Exception (3);	UNUSED_54 (54);	ExitCode (246).
Diagnostic Exception (3);	UNUSED_55 (55);	ExitCode (247).
Diagnostic Exception (3);	UNUSED_56 (56);	ExitCode (248).
Diagnostic Exception (3);	UNUSED_57 (57);	ExitCode (249).
Diagnostic Exception (3);	UNUSED_58 (58);	ExitCode (250).
Diagnostic Exception (3);	UNUSED_59 (59);	ExitCode (251).
Diagnostic Exception (3);	UNUSED_60 (60);	ExitCode (252).
Diagnostic Exception (3);	UNUSED_61 (61);	ExitCode (253).
Diagnostic Exception (3);	UNUSED_62 (62);	ExitCode (254).
Diagnostic Exception (3);	Unknown Error (63);	ExitCode (255).

4.1.8 tsLogger

From Python Documentation (for Python logging facility):

This module defines functions and classes which implement a flexible event logging system for applications and libraries.

The key benefit of having the logging API provided by a standard library module is that all Python modules can participate in logging, so your application log can include your own messages integrated with messages from third-party modules.

- 1 Class emulates a subset of Python logging API, to establish methods that receive commands from application or system components requesting output of messages to a designated logger (such as stdout, stderr, syslog, GUI object, CLI file) with the requested priority (such as DEBUG, INFO, NOTICE, WARNING, ALERT, ERROR, CRITICAL, EMERGENCY) and format (such as date and time stamped, indented, line wrapped).

Run-time message logging utility defines and handles event message time stamping, formatting and output. See: **Logger Severity Levels** (on page 240)

- 2 Class establishes methods for assert type operations (wxASSERT, wxASSERT_MSG, wxCHECK, wxCHECK_MSG, wxCHECK_RET, wxCHECK2, wxCHECK2_MSG, wxFAIL, wxFAIL_COND_MSG, wxFAIL_MSG and wxTRAP).

Run-time message logging utility evaluates and handles assert and check case results. See: **Logger Assert and Check Cases** (on page 243)

4.1.8.1 Logger Severity Levels

The tsLogger run-time message logging utility defines and handles event message time stamping, formatting and output.

SEVERITY	PRIORITY	APPLICATION
NOTSET	0 (None) <ul style="list-style-type: none"> No Details 	<ul style="list-style-type: none"> Indication that severity has not been specified by application. logger.notset is NOT available. Example: <pre>msg = 'NOT Available' logger.notset(msg)</pre>

PRIVATE	1 (Min) ▪ Unlimited Details	<ul style="list-style-type: none"> ▪ Option used to suppress console output of data intended only to be archived in a file. ▪ Example: <pre>msg = 'memory dump:' + \ '%s' % str(dump) logger.private(msg)</pre>
DEBUG	10 ▪ Lowest Usable Details	<ul style="list-style-type: none"> ▪ Option used to communicate troubleshooting information to the system operator. ▪ Detailed information, typically of interest only when diagnosing problems. ▪ Example: <pre>msg = 'Should Water Temperature ' + \ 'be over 130 degrees?' logger.debug(msg)</pre>
INFO	20 ▪ Progress Details	<ul style="list-style-type: none"> ▪ Option used to communicate progress information to the system operator. ▪ Confirmation that things are working as expected. ▪ Example: <pre>msg = 'Water Temperature ' + \ 'between 120-129 degrees.' logger.info(msg)</pre>
NOTICE	25 ▪ Milestone Details	<ul style="list-style-type: none"> ▪ Option used to communicate milestone information to the system operator. ▪ Example: <pre>msg = 'Water Temperature ' + \ 'reached 130 degrees.' logger.notice(msg)</pre>
WARNING	30 ▪ Pre-Alarm Details	<ul style="list-style-type: none"> ▪ Used by default for anticipated, recoverable pre-alarm conditions which ought to be brought to the attention of the system operator. ▪ 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. ▪ Example: <pre>msg = 'Water Temperature ' + \ 'over 130 degrees.' logger.warning(msg)</pre>
ALERT	35 ▪ Alarm Details	<ul style="list-style-type: none"> ▪ Used by default for anticipated, recoverable alarm conditions which ought to be brought to the attention of the system operator. ▪ Example:

		<pre>msg = 'Water Temperature ' + \ 'between 131-139 degrees.' logger.alert(msg)</pre>
ERROR	40 <ul style="list-style-type: none"> Failure Details 	<ul style="list-style-type: none"> Used for unexpected, non-recoverable failures which need to be communicated to the system operator. Due to a more serious problem, the software has not been able to perform some function. Example: <pre>msg = 'Sensor Failed. ' + \ 'Water Temperature not ' + \ 'in range of 32-212 degrees' logger.error(msg)</pre>
CRITICAL	50 <ul style="list-style-type: none"> Damage Related Details 	<ul style="list-style-type: none"> Used for urgent, non-recoverable operating situations (coolant leak) which need to be communicated to the system operator. A serious error, indicating that the program itself may be unable to continue running. Example: <pre>msg = 'Thermostat failed. ' + \ 'Water Temperature over ' + \ '140 degrees.' logger.critical(msg)</pre>
EMERGENCY	55 <ul style="list-style-type: none"> Safety Related Details 	<ul style="list-style-type: none"> Used for urgent, non-recoverable operating situations (primary power loss) which need to be communicated to the system operator. Example: <pre>msg = 'Primary AC Power ' + \ 'Supply failed. Shut' + \ 'down system before ' + \ 'Backup Battery runs out ' + \ 'in 10 minutes.' logger.emergency(msg)</pre>

4.1.8.2 Logger Assert and Check Cases

The tsLogger run-time message logging utility evaluates and handles assert and check case results.

Assertion handlers: check if the condition is true and call assert handler (which will by default notify the user about failure) if it is not.

- wxASSERT and wxFAIL handlers as well as wxTrap() function do nothing at all if wxDEBUG_LEVEL is 0 however they do check their conditions at default debug level 1, unlike the previous wxWidgets versions.
- wxASSERT_LEVEL_2 is meant to be used for "expensive" asserts which should normally be disabled because they have a big impact on performance and so this macro only does anything if wxDEBUG_LEVEL >= 2.
- wxCHECK handlers always check their conditions, setting debug level to 0 only makes them silent in case of failure, otherwise -- including at default debug level 1 -- they call the assert handler if the condition is false

They are supposed to be used only in invalid situation: for example, an invalid parameter (e.g. a NULL pointer) is passed to a function. Instead of dereferencing it and causing core dump the function might use:

```
wxCHECK_RET( p != NULL, "pointer ca not be NULL" )
```

4.1.9 tsOperatorSettingsBuilder API (Future)

Class for building command-line option parser. It uses a declarative style of command-line parsing definitions: you create an instance of tsOperatorSettingsParser, populate it with options, and parse the command line. It allows users to specify options in the conventional GNU/POSIX syntax, and additionally generates usage and help messages for you.

4.1.10 tsOperatorSettingsParser API

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.

- 1 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.

- 2** Provides keyword-value pair definitions needed for debugging non-application-specific Command Line and Graphical-style User Interfaces:
 - a) -h, --help
 - b) -v, --version
 - c) -a, --about
 - d) -d, --debug
 - e) -V, --Verbose
- 3** Provides positional argument definitions needed for debugging Python version-specific library modules:
 - a) module {choice of "argparse", "optparse", "getopt"}

4.1.10.1 Methods

Methods defined here:

```

__init__(self, *args, **kw)
    Class constructor.

argsFormatter(self, initialIndent, argsText)
    Format a list of positional arguments so that each one will be
    displayed on a line by itself.

dedentFormatter(self, inputBuffer)
    Unwrap contents of buffer.

getAbout(self)
    Return False, by default, after initialization and True after
    operator entered "-a"/"--about" key-word option.

getCommandLineParserModule(self, rawArgsOptions)
    Return the lower case module name of the Command Line Parser as
    appropriate to the Python version unless specified by the operator
    via a case-insensitive positional argument (e.g. "argparse",
    "OptParse" or "GETOPT").

getDebug(self)
    Return False, by default, after initialization and True after
    operator entered "-d"/"--debug" key-word option.

getHelp(self)
    Return False, by default, after initialization and True after
    operator entered "-ha"/"--help" key-word option.

getProperties(self)
    Return lookup table of keyword-get-property pairs.

getRunTimeHeader(self)
    Return Run Time Header, or Build Header, whichever was actually
    used in command line, stripping it of any file path.

getRunTimeTitle(self)
    Return Run Time Title, or Build Title, whichever was actually
    used in command line, stripping it of any file path.

getRunTimeTitleVersionDate(self)
    Return Run Time Title, or Build Title, whichever was actually
    used in command line, stripping it of any file path.

getVerbose(self)
    Return False, by default, after initialization and True after
    operator entered "-V"/"--Verbose" key-word option.

getVersion(self)
    Return False, by default, after initialization and True after
    operator entered "-v"/"--version" key-word option.

```

```
helpAbout(self)
    helpAbout

helpDebug(self)
    helpDebug

helpHelp(self)

helpPurpose(self)
    helpPurpose

helpVerbose(self)
    helpVerbose

helpVersion(self)
    helpVersion

optionsFormatter(self, initialIndent, optionsText)
    Format a dictionary of keyword-value pair options so that each
    pair will be displayed on a line by itself.

parseCommandLineDispatch(self)
    Stub to substitute for non-existent parseCommandLine. Returns
    non-application specific behavior and results appropriate for
    the latest Python parser module ("argparse", "optparse" or
    "getopt") available to the application.

parseCommandLineUsageViaArgParse(self)
    Create sample command line usage help, for use when application
    specific command line parser is under development or otherwise
    not available. For example:

    "
    Usage:  tsApplication.py <Keyword Value Pair Option(s)> \
            <Positional Argument(s)>

    Example(s):      tsApplication.py
                     tsApplication.py <Keyword Value Pair Option(s)>
                     tsApplication.py <Positional Argument(s)>

    Purpose:         demonstrate use of python "argparse" module.
    "

parseCommandLineUsageViaGetOpt(self)
    Create sample command line usage help, for use when application
    specific command line parser is under development or otherwise
    not available. For example:

    "
    Usage:  tsApplication.py <Keyword Value Pair Option(s)> \
            <Positional Argument(s)>

    Example(s):      tsApplication.py
                     tsApplication.py <Keyword Value Pair Option(s)>
                     tsApplication.py <Positional Argument(s)>
```

```
Purpose:      demonstrate use of python "getopt" module.
"
```

```
parseCommandLineUsageViaOptParse(self)
```

Create sample command line usage help, for use when application specific command line parser is under development or otherwise not available. For example:

11

```
Usage:  tsApplication.py <Keyword Value Pair Option(s)> \
      <Positional Argument(s)>
```

```
Example(s):      tsApplication.py
                  tsApplication.py <Keyword Value Pair Option(s)>
                  tsApplication.py <Positional Argument(s)>
```

```
tsApplication.py <Positional Argument(s)>
```

```
Purpose:      demonstrate use of python "optparse" module.
"
```

```
parseCommandLineViaArgParse(self)
```

```
Parse the command line and extract any keyword-value pair and
positional arguments.
```

```
parseCommandLineViaGetOpt (self)
```

Parse the command line and extract any keyword-value pair and positional arguments.

```
parseCommandLineViaOptParse(self)
```

Parse the command line and extract any keyword-value pair and positional arguments.

```
setAbout(self, value)
```

Set True after operator entered "-a"/"--about" key-word option.

```
setDebug(self, value)
```

Set True after operator entered "-d"/"--debug" key-word option.

```
setHelp(self, value)
```

Set True after operator entered "-h"/"--help" key-word option.

```
setProperties(self, about, debug, version, Verbose)
```

Return lookup table of keyword-set-property pairs.

```
setVerbose(self, value)
```

Set True after operator entered "-V"/"--Verbose" key-word option.

```
setVersion(self, value)
```

```
Set True after operator entered "-v"/"--version" key-word option.
```

```
unknown(self, opts, resultsOptions)
```

4.1.11 **tsPlatformRunTimeEnvironment**

Class to establish methods to capture, format and output current hardware and software information about the run time environment for the running application process.

- 1 Network Identification (hostname, aliaslist, ipaddrlist)
- 2 Host Central Processing Unit (machine, processor, architecture, byteorder)
- 3 Host Operating System (api, system, release, version)
- 4 Python Platform (branch, build, compiler, implementation, revision, version)
- 5 Host Operating System Library (distname, version, id, lib, lib version)
- 6 Process Parameters (pid, getppid, getegid, geteuid, getgid, getgroups, getpgid, getuid, getlogin, ctermid, cwd)
- 7 Environment Variables

4.1.12 **tsReportUtility**

Class to establish methods used to format information (displayDictionary, getByteCountStrings (Bytes, Kilo, Mega, Giga, Tera, Peta, Exa and Zetta), getDateAndTimeString (Oct. 31 2011 at 12:31:59), getDateTimeStrng (2011/05/13-04:17:49.123), getDayHourMinuteSecondString, getElapsedTimeString, getHourMinuteSecondString, etc.

4.1.13 **tsSysCommand**

Class to establish methods used for issuing shell commands to the host operating system.

4.1.14 `tsThreadPool`

4.2 Graphical User Interface API

This portion of the toolkit provides a library of building blocks and application programs that support the development of graphical-style applications. Such applications are invoked via mouse clicks on GUI objects and by textual commands to dialog objects entered via the console keyboard. Output is viewed on the console display.

Application programs ultimately interface with the operator via input/output services provided by the host computer operating system.

- *nCurses API* (on page 250)
- *Python Curses API* (on page 252)
- *wxPython-Style API* (see "*wxPython API*" on page 255)
- *tsWxGTUI API* (on page 259)

4.2.1 nCurses API

NOTE: This is an internal API used by the "tsWxGTUI_PyVx" Text Style, wxPython GUI Toolkit developers.

The UNIX-style "nCurses" (new curses) programming library provides an API, allowing 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. There are two "nCurses" library versions:

- libncurses - Supports the default, normal library that works properly only in 8-bit locales.
- libncursesw - Supports the special wide-character library that is usable in both multibyte and traditional 8-bit locales. Building this library requires use of the "--enable-widec" configuration option.
- Wide-character and normal libraries are source-compatible, but not binary-compatible.
- The "nCurses" API for C programmers is available at <http://invisible-island.net/ncurses/man/ncurses.3x.html> (<http://invisible-island.net/ncurses/man/ncurses.3x.html>)

The "nCurses" programming library is one of those input/output services provided by the host computer operating system. By its design, it shields application programs from device specific interface details. The following output-only display was produced by **905 lines code** using the Python programming language and its "nCurses" library module. Adding input capability requires creating a substantial amount of additional code equivalent to that created for the emulated *wxPython API* (on page 255).



Draft

4.2.2 Python Curses API

The Python 2.x "Curses" API for programmers, a subset of the "nCurses" API for C Programmers, is available at <http://docs.python.org/2/library/curses.html> (<http://docs.python.org/2/library/curses.html>).

The Python 3.x "Curses" API for programmers, a subset of the "nCurses" API for C Programmers, is available at <http://docs.python.org/3/library/curses.html> (<http://docs.python.org/3/library/curses.html>).

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

"What is curses?"

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>.

The Python curses module

The Python module is a fairly simple wrapper over the C functions provided by curses; if you're already familiar with curses programming in C, it's really easy to transfer that knowledge to Python. The biggest difference is that the Python interface makes things simpler, by merging different C functions such as `addstr()`, `mvaddstr()`, `mvwaddstr()`, into a single `addstr()` method. You'll see this covered in more detail later.

This HOWTO is simply an introduction to writing text-mode programs with curses and Python. It doesn't attempt to be a complete guide to the curses API; for that, see the Python library guide's section on ncurses, and the C manual pages for ncurses. It will, however, give you the basic ideas."

Unless Python 3.x's (a backporting to 2.6 and 2.7) integrated Unicode support changed things . . .

Python Curses Module Limitations as found on-line by the "tsWxGTUI_PyVx" Toolkit Author:

- It does not use the "nCurses" wide-character library and therefore cannot support wide characters.
- It interfaces only to a subset of the most commonly used "nCurses" API features.
- The Python Software Foundation encourages users to submit contributions which extend the subset.

The "tsWxGTUI_PyVx" Toolkit has been designed to handle text of a uniform size. It has not been designed to handle Unicode characters. This does not preclude a future design change.

4.2.2.1 Python Curses Module Evolution (Draft)

4.2.2.1.1 Python 1.5 Curses

From <https://docs.python.org/release/1.5.2p2/lib/module-curses.html>:

"6.12 curses -- Terminal independant console handling

The curses module provides an interface to the curses Unix 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.

The extension module has not been tested with all available versions of curses."

See Also:

"Tutorial material on using curses with Python is available on the Python Web site as Andrew Kuchling's Curses Programming with Python, at <http://www.python.org/doc/howto/curses/curses.html>."

6.12.1 Constants and Functions

6.12.2 Window Objects

4.2.2.1.2 Python 2.2.1 Curses

From <https://docs.python.org/release/2.2.1/lib/module-curses.html>

"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."

See Also:

Module `curses.ascii`:

Utilities for working with ASCII characters, regardless of your locale settings.

Module `curses.panel`:

A panel stack extension that adds depth to curses windows.

Module `curses.textpad`:

Editable text widget for curses supporting Emacs-like bindings.

Module `curses.wrapper`:

Convenience function to ensure proper terminal setup and resetting on application entry and exit.

Curses Programming with Python

Tutorial material on using curses with Python, by Andrew Kuchling and Eric Raymond, is available on the Python Web site.

The `Demo/curses/` directory in the Python source distribution contains some example programs using the curses bindings provided by this module.

6.13.1 Functions

6.13.2 Window Objects

6.13.3 Constants

4.2.2.1.3 Python 3.0 Curses

From <https://docs.python.org/release/3.4.3/library/curses.html#module-curses>:

"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 Windows, DOS, 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:

```
import locale
locale.setlocale(locale.LC_ALL, "")
code = locale.getpreferredencoding()
Then use code as the encoding for str.encode() calls."
```

See also

Module `curses.ascii`

Utilities for working with ASCII characters, regardless of your locale settings.

Module `curses.panel`

A panel stack extension that adds depth to curses windows.

Module `curses.textpad`

Editable text widget for curses supporting Emacs-like bindings.

Curses Programming with Python

Tutorial material on using curses with Python, by Andrew Kuchling and Eric Raymond.

The `Tools/demo/` directory in the Python source distribution contains some example programs using the curses bindings provided by this module.

4.2.3 wxPython API

"wxPython" is one of several programming language bindings for the "wxWidgets" Graphical User Interface Toolkit.

1 "wxWidgets" is the popular Pixel-mode Graphical User Interface toolkit for C++ programmers.

- It is implemented in the C++ programming language and is designed for Linux, Mac OS X, Windows and various other platforms having displays that operate in pixel-mode.

- It interfaces with and maps its operations to each platform-specific Graphical User Interface API. This preserve the native look and feel established by the underlying operating system without sacrificing the platform-independence sought by developers of cross-platform applications.
 - it provides popular language binding options for Python, Perl, Ruby and many other languages, in order to extend the "wxWidgets" capabilities to users of other programming languages.
 - It is currently at Release 2.8.12, for its old API, and 2.9.4 for its new API.
- 2** "wxPython" is the popular Pixel-mode Graphical User Interface toolkit for Python programmers.
- It is implemented in the Python programming language and supports current Python 2.x releases.
 - It supports a subset of the "wxWidgets" API.
 - It is currently at Release 2.8.12, for its old API, without any mention of 2.9.4 for its new API.
- 3** "tsWxGTUI_PyVx" is the Character-Mode Graphical User Interface toolkit for Python programmers.
- It emulates a "wxPython-style" subset of the "wxWidgets" API suitable for displays operated in character-mode.
 - It interfaces with the platform-specific "nCurses" Graphical User Interface API, in order to create and manage GUI objects on a Local or Remote terminal.
 - It interfaces with and maps its "wxPython-style" operations to the platform-specific "nCurses" Graphical User Interface API, in order to offer terminal-independence while preserving the interface and functional capability the "wxWidgets" API.
 - It is implemented in the Python programming language and supports current Python 2.x and 3.x releases.

The "tsWxGTUI_PyVx" Toolkit provides an "nCurses" wrapper that maps "wxWidget" style pixel-mode GUI-object types, colors, dimensions, attributes and methods into their "nCurses" style character-mode equivalents. It also identifies the GUI-object associated with the current mouse "click" by analyzing the hierarchical, overlapping and focus relationship between the layered GUI-objects. For example a pop-up dialog could obscure portions of a top level frame and a dialog button could obscure other portions of the dialog. Nothing that is obscured can be associated with a mouse "click". The following display was produced by **414 lines code** using the Python programming language and the "tsWxGTUI_PyVx" Text Style, wxPython GUI Toolkit, which itself used Python's "nCurses" library module. The toolkit improves:

- the productivity of the application developer through its higher level building blocks such as sizers and textctrl widgets which dynamically position, size and scroll output to fit the operator's desktop preferences.
- the maintainability of the application code through the maturity of the toolkit's object-oriented API and its built-in support of exception handling and diagnosing message reporting.

The following figure is visually similar to the output-only one in the ***N**Curses API* (on page 250). Adding input capability only requires adding a relatively small amount of event handling code to define and connect (bind) clickable GUI Objects (such as buttons, checkboxes etc) to the appropriate event handler.

Draft



Draft

4.2.4 tsWxGTUI API

NOTE: This is an internal API used by the `tsWxGTUI_PyVx` Text Style, wxPython GUI Toolkit developers.

This module establishes and maintains the Graphical Text User Interface. It performs the following activities:

- Initializes and starts Python's "curses" module.
- Defines the 8-colors and identifiers for the the "nCurses" palette and its 64-foreground/background combinations.
- Defines the 88-colors and identifiers for the the "wxPython" palette and its mapping into the 8-color "nCurses" palette.
- Defines the line-draw and text symbols available for creating window borders.
- Defines methods to convert between "wxPython" pixel-mode and "nCurses" character-mode dimensions.
- Defines methods to monitor and convert "nCurses" mouse button and position into "wxPython" triggering events.
- Defines methods for making association of "nCurses" mouse button and position with "wxPython" GUI object using the hierarchical layer relationships and testing of which non-overlaid GUI object contains the mouse position.
- Establishes and maintains a data base of the "wxPython" and "nCurses" run time configurations.
- Provides methods that interface "wxPython" to their associated "nCurses" ones.
- Restore the pre-startup terminal configuration and shutdown Python's "curses" module.

Draft

5 TOOLS & UTILITIES

5.1 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.

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).

5.2 tsPlatformQuery

Python application program, with a Command Line Interface (CLI), that generates reports of current hardware and software information about the run time environment available to computer programs:

- Network host information including hostname, list of aliases and list of IP addresses.
- Host CPU machine and processor information
- Console window width and height
- Host Operating System API, system, release and version
- Python Platform branch, build, compiler, implementation, revision and version
- Java Platform release, vendor, vm_name, vm_vendor, vm_release, os arch, os name and os version
- Mac Platform release, version, dev stage, non-release ver, machine
- Linux Platform distname, version, id, lib, lib version
- Windows rPlatform release, version, csd service pack and core & build type
- Process pid, getppid, geteuid, getid, getgroups, getpgid, getuid and environment variables.

5.3 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.

5.4 tsStripLineNumbers

Python application program, with a Command Line Interface (CLI), that strips line numbers from source code (such as annotated listings) that do not reference line numbers for conditional branching.

5.5 tsTreeCopy

Python application program, with a Command Line Interface (CLI), that copies the contents of a source directory to a target directory.

5.6 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.

6 DEFINITIONS, ABBREVIATIONS, AND ACRONYMS

Trademarks and copyrights (http://en.wikipedia.org/wiki/Wikipedia:About#Trademarks_and_copyrights)

Wikipedia is a registered trademark of the not-for-profit *Wikimedia Foundation*, which has created a family of free-content projects that are built by user contributions.

Most of Wikipedia's text and many of its images are dual-licensed under the *Creative Commons Attribution-Sharealike 3.0 Unported License* (CC-BY-SA) and the *GNU Free Documentation License* (GFDL) (unversioned, with no invariant sections, front-cover texts, or back-cover texts). Some text has been imported only under CC-BY-SA and CC-BY-SA-compatible license and cannot be reused under GFDL; such text is identified either on the page footer, in the page history or on the discussion page of the article that utilizes the text. Every image has a description page which indicates the license under which it is released or, if it is non-free, the rationale under which it is used.

Contributions remain the property of their creators, while the CC-BY-SA and GFDL licenses ensure the content is freely distributable and reproducible. (See the **copyright notice** (<http://en.wikipedia.org/wiki/Wikipedia:Copyrights>) and the **content disclaimer** (<http://en.wikipedia.org/wiki/Wikipedia:Disclaimers>) for more information.)

The following terms are used throughout this document:

TERM	DEFINITION
API	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.
AuthorIT	<p>Excerpts From Wikipedia, the free encyclopedia:</p> <p>"Author-it is a help authoring tool and content management system for creating, maintaining, and distributing single-sourced content.</p> <p>Author-it can produce documentation in the following formats:</p> <ul style="list-style-type: none"> ▪ RTF, PDF, or Microsoft Word format for printed documentation ▪ Microsoft WinHelp (Windows Help) ▪ Microsoft HTML Help ▪ JavaHelp ▪ Oracle Help for Java ▪ Web and browser based help ▪ XML ▪ DITA

TERM	DEFINITION
	<p>Author-it stores all information as objects in a central database, called a library. Object types include books, topics, file objects, hyperlinks, styles, glossaries, tables of contents, indexes, and publishing profiles. The library supports multiple users. Author-it Base User module includes importing, authoring, and publishing capability. Further modules that can be licensed...."</p> <p>NOTES:</p> <ul style="list-style-type: none"> ▪ Author-it is a product of the Author-it Software Corporation, Ltd. ▪ Any chapter, section or paragraph component authored for a hierarchical location in one document (document 1 at a.b.c) may be re-used by reference at any other hierarchical location in any other document (document 2 at d.e, document 3 at f.g.h.i.j.k). Author-it automatically re-numbers new references as appropriate for their new hierarchical location. ▪ Author-it 4.5 is the last stand-alone Author-it Workgroup Edition. It uses the Microsoft JET Database Engine and is compatible with Windows XP Professional, Windows Vista and Windows 7 Professional. ▪ Author-it 5.5 is the last stand-alone Edition to use the Microsoft JET Database Engine. It is compatible with Windows XP Professional, Windows Vista, Windows 7 Professional and Windows 8 Professional. Have yet to discover how to export Microsoft JET Database to either the Microsoft SQL Server or the free Microsoft SQL Express Database. ▪ Author-it iCloud Professional and Enterprise Editions use either the Microsoft SQL Server or the free Microsoft SQL Express Database engine.
Automation	The use of various control systems for operating equipment such as machinery, processes in factories, telephone networks, steering and stabilization of ships, aircraft and other applications with minimal or reduced human intervention.
Berkley Software Distribution License	<p>From Wikipedia, the free encyclopedia</p> <p>"* * *. BSD licenses are a family of permissive free software licenses, imposing minimal restrictions on the redistribution of covered software. This is in contrast to copyleft licenses, which have reciprocity share-alike requirements. The original BSD license was used for its namesake, the Berkeley Software Distribution (BSD), a Unix-like operating system. The original version has since been revised and its descendants are more properly termed modified BSD licenses.</p> <p>Two variants of the license, the New BSD License/Modified BSD License (3-clause),[1] and the Simplified BSD License/FreeBSD License (2-clause)[2] have been verified as GPL-compatible free software licenses by the Free Software Foundation, and have been vetted as open source licenses by the Open Source Initiative,[3] while the original, 4-clause license has not been accepted as an open source license and, although the original is considered to be a free software license by the FSF, the FSF does not consider it to be compatible with the GPL due to the advertising clause.[4]"</p>
Building Blocks	A Building Block is a basic unit from which something of greater complexity is built up. For example, an application or operating system program may be constructed from one or more data structure definitions, functions, methods, classes, subroutines, threads, processes or building block libraries.
CLI	<p>Acronym for Command Line Interface.</p> <p>From Wikipedia, the free encyclopedia:</p> <p>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.</p>
Communication	The application of telecommunications technology for the transmission of data to, from, or between computers over dedicated or time shared hardware.

TERM	DEFINITION
Control	<p>The application of one or more devices, to manage, command, direct or regulate the behavior of other device(s) or system(s). Industrial control systems, as used in industrial production, control equipment or machinery. In open loop control systems output is generated based only on inputs. In closed loop (feedback) control systems current output is taken into consideration and corrections are made based on feedback.</p>
Curses	<p>From Wikipedia, the free encyclopedia:</p> <p>"curses is a terminal control library for Unix-like systems, enabling the construction of text user interface (TUI) applications.name is a pun on the term "cursor optimization". It is a library of functions that manage an application's display on character-cell terminals (e.g., VT100).[1]</p> <p>Overview</p> <p>The curses API is described in several places.[2] Most implementations of curses use a database that can describe the capabilities of thousands of different terminals. There are a few implementations, such as PDCurses, which use specialized device drivers rather than a terminal database. Most implementations use terminfo; some use termcap. Curses has the advantage of back-portability to character-cell terminals and simplicity. For an application that does not require bit-mapped graphics or multiple fonts, an interface implementation using curses will usually be much simpler and faster than one using an X toolkit.</p> <p>Using curses, programmers are able to write text-based applications without writing directly for any specific terminal type. The curses library on the executing system sends the correct control characters based on the terminal type. It provides an abstraction of one or more windows that maps onto the terminal screen. Each window is represented by a character matrix. The programmer sets up each window to look as they want the display to look, and then tells the curses package to update the screen. The library determines a minimal set of changes needed to update the display and then executes these using the terminal's specific capabilities and control sequences.</p> <p>In short, this means that the programmer simply creates a character matrix of how the screen should look and lets curses handle the work."</p>

TERM	DEFINITION
Cygwin	<p>From Wikipedia, the free encyclopedia:</p> <p>"Cygwin[2] is a Unix-like environment and command-line interface for Microsoft Windows. Cygwin provides native integration of Windows-based applications, data, and other system resources with applications, software tools, and data of the Unix-like environment. Thus it is possible to launch Windows applications from the Cygwin environment, as well as to use Cygwin tools and applications within the Windows operating context.</p> <p>Cygwin consists of two parts: a dynamic-link library (DLL) as an API compatibility layer providing a substantial part of the POSIX API functionality, and an extensive collection of software tools and applications that provide a Unix-like look and feel.</p> <p>Cygwin was originally developed by Cygnus Solutions, which was later acquired by Red Hat. It is free and open source software, released under the GNU General Public License version 3. Today it is maintained by employees of Red Hat, NetApp and many other volunteers."</p> <p>See also:</p> <ul style="list-style-type: none"> ▪ Cooperative Linux ▪ Cygwin/X (X11 for Cygwin) ▪ GnuWin32 ▪ Interix ▪ MinGW (Minimalist GNU for Windows) ▪ mintty (Cygwin terminal) ▪ UWIN
Darwin	<p>From Wikipedia, the free encyclopedia:</p> <p>"Darwin is an open source Unix-like computer operating system released by Apple Inc. in 2000. It is composed of code developed by Apple, as well as code derived from NeXTSTEP, BSD, and other free software projects.</p> <p>Darwin forms the core set of components upon which OS X and iOS are based. It is mostly POSIX compatible, but has never, by itself, been certified as being compatible with any version of POSIX. (OS X, since Leopard, has been certified as compatible with the Single UNIX Specification version 3 (SUSv3).[2][3][4])</p> <p>HISTORY</p> <p>Darwin's heritage began with NeXT's NeXTSTEP operating system (later known as OpenStep), first released in 1989. After Apple bought NeXT in 1997, it announced it would base its next operating system on OpenStep. This was developed into Rhapsody in 1997, Mac OS X Server 1.0 in 1999, Mac OS X Public Beta in 2000, and Mac OS X 10.0 in 2001. In 2000, the core operating system components of Mac OS X were released as open-source software under the Apple Public Source License (APSL) as Darwin; the higher-level components, such as the Cocoa and Carbon frameworks, remained closed-source.</p> <p>Up to Darwin 8.0.1, Apple released a binary installer (as an ISO image) after each major Mac OS X release that allowed one to install Darwin on PowerPC and Intel x86 computers as a standalone operating system. Minor updates were released as packages that were installed separately. Darwin is now only available as source code,[5] except for the ARM variant, which has not been released in any form separately from iOS. However, the older versions of Darwin are still available in binary form,[6] and a hobbyist developer winocm took the official Darwin source code and ported it to ARM.[7]"</p>

TERM	DEFINITION
Debian	<p>From Wikipedia, the free encyclopedia</p> <p>"Debian (/ˈdɛbiən/) is an operating system composed primarily of free and open-source software, most of which is under the GNU General Public License, and developed by a group of individuals known as the Debian project. Debian is one of the most popular Linux distributions for personal computers and network servers, and has been used as a base for several other Linux distributions.</p> <p>Debian was first announced in 1993 by Ian Murdock, and the first stable release was made in 1996. The development is carried out over the Internet by a team of volunteers guided by a project leader and three foundational documents. New distributions are updated continually, and the next candidate is released after a time-based freeze.</p> <p>As one of the earliest Linux distributions, it was envisioned that Debian was to be developed openly in the spirit of Linux and GNU. This vision drew the attention and support of the Free Software Foundation, which sponsored the project for the first part of its life."</p>
Developer Sandbox	<p>Excerpt From Wikipedia, the free encyclopedia:</p> <p>"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.)</p> <p>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]</p> <p>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."</p>
Diagnostic	<p>The application of technology to locate problems with software, hardware, or any combination thereof in a system, or a network of systems. Diagnostics typically provide guidance to the user to solve issues.</p>
Docstring	<p>Excerpt from http://www.python.org/dev/peps/pep-0257/:</p> <p>"A docstring is a string literal that occurs as the first statement in a module, function, class, or method definition. Such a docstring becomes the <code>__doc__</code> special attribute of that object.</p> <p>All modules should normally have docstrings, and all functions and classes exported by a module should also have docstrings. Public methods (including the <code>__init__</code> constructor) should also have docstrings. A package may be documented in the module docstring of the <code>__init__.py</code> file in the package directory.</p> <p>String literals occurring elsewhere in Python code may also act as documentation. They are not recognized by the Python bytecode compiler and are not accessible as runtime object attributes (i.e.</p>

TERM	DEFINITION
	<p>not assigned to <code>__doc__</code>), but two types of extra docstrings may be extracted by software tools:</p> <ul style="list-style-type: none"> ▪ String literals occurring immediately after a simple assignment at the top level of a module, class, or <code>__init__</code> method are called "attribute docstrings". ▪ String literals occurring immediately after another docstring are called "additional docstrings". <p>Please see PEP 258, "Docutils Design Specification" [2], for a detailed description of attribute and additional docstrings."</p>
Dropbox	<p>From www.dropbox.com:</p> <p>"Dropbox is a free service that lets you bring all your photos, docs, and videos anywhere. Any file you save to your Dropbox will also automatically save to all your computers, phones, and even the Dropbox website. This means that you can start working on your computer at school or the office, and finish on your home computer. Never email yourself a file again!"</p>
Extension Module	<p>A module written in the low-level language of the Python implementation: C/C++ for Python, Java for Jython. Typically contained in a single dynamically loadable pre-compiled file, e.g. a shared object (.so) file for Python extensions on Unix, a DLL (given the .pyd extension) for Python extensions on Windows, or a Java class file for Jython extensions. (Note that currently, the Distutils only handles C/C++ extensions for Python.)</p>
FreeBSD	<p>From Wikipedia, the free encyclopedia:</p> <p>"FreeBSD is a free Unix-like operating system descended from Research Unix via 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". The first version of FreeBSD was released in 1993, and today FreeBSD is the most widely used open-source BSD distribution, accounting for more than three-quarters of all installed systems running open-source BSD derivatives.[3]</p> <p>FreeBSD has similarities with Linux, with two major differences in scope and licensing: FreeBSD maintains a complete operating system, i.e. the project delivers kernel, device drivers, userland utilities and documentation, as opposed to a kernel only;[4] and FreeBSD source code is generally released under a permissive BSD license as opposed to the more restrictive GPL.</p> <p>The FreeBSD project includes a security team overlooking all software shipped in the base distribution. A wide range of additional third-party applications may be installed via two package managers, "pkgng" and the FreeBSD Ports, or by directly compiling source code. Due to its permissive licensing terms, much of FreeBSD's code base has become an integral part of other operating systems such as Juniper JUNOS and Apple's OS X."</p>

TERM	DEFINITION
Functional Requirements	<p>From Wikipedia, the free encyclopedia:</p> <p>"In software engineering (and System Engineering), a functional requirement defines a function of a system or its component. A function is described as a set of inputs, the behavior, and outputs (see also software). Functional requirements may be calculations, technical details, data manipulation and processing and other specific functionality that define what a system is supposed to accomplish. Behavioral requirements describing all the cases where the system uses the functional requirements are captured in use cases. Functional requirements are supported by non-functional requirements (also known as quality requirements), which impose constraints on the design or implementation (such as performance requirements, security, or reliability). Generally, functional requirements are expressed in the form "system must do <requirement>", while non-functional requirements are "system shall be <requirement>". The plan for implementing functional requirements is detailed in the system design. The plan for implementing non-functional requirements is detailed in the system architecture.</p> <p>As defined in requirements engineering, functional requirements specify particular results of a system. This should be contrasted with non-functional requirements which specify overall characteristics such as cost and reliability. Functional requirements drive the application architecture of a system, while non-functional requirements drive the technical architecture of a system.</p> <p>In some cases a requirements analyst generates use cases after gathering and validating a set of functional requirements. The hierarchy of functional requirements is: user/stakeholder request → feature → use case → business rule. Each use case illustrates behavioral scenarios through one or more functional requirements. Often, though, an analyst will begin by eliciting a set of use cases, from which the analyst can derive the functional requirements that must be implemented to allow a user to perform each use case."</p>
GNU	<p>From Wikipedia, the free encyclopedia:</p> <p>"GNU [2][3] is a Unix-like computer operating system developed by the GNU Project, ultimately aiming to be a "complete Unix-compatible software system" [1][4][5] composed wholly of free software. Development of GNU was initiated by Richard Stallman in 1983 [1][6] and was the original focus of the Free Software Foundation (FSF). [1][7][8][9] Non-GNU kernels, most famously the Linux kernel, can also be used with GNU. [10][11][12] The FSF maintains that Linux, when used with GNU tools and utilities, should be considered a variant of GNU, and promotes the term Linux for such systems (leading to the Linux naming controversy). [13][14][15]</p> <p>GNU is a recursive acronym for "GNU's Not Unix!", [1][16] chosen because GNU's design is Unix-like, but differs from Unix by being free software and containing no Unix code. [17] Programs released under the auspices of the GNU Project are called GNU packages or GNU programs. The system's basic components include the GNU Compiler Collection (GCC), the GNU C library (glibc), and GNU Core Utilities (coreutils), [1] but also the GNU Debugger (GDB), GNU Binary Utilities (binutils), and the bash shell. [18] GNU developers have contributed GNU/Linux Linux ports of GNU applications and utilities, which are now also widely used on other operating systems such as BSD variants, Solaris and Mac OS X. [19]</p> <p>The GNU General Public License (GPL), the GNU Lesser General Public License (LGPL), and the GNU Free Documentation License (GFDL) were written for GNU and the GNU Affero General Public License was written as an extended version of GPL version 3 for programs run over a network, but the GNU Project's licenses are also used by many unrelated projects. A minority of the software used by GNU, such as the X Window System, is licensed under permissive free software licenses.</p> <p>Richard Stallman views GNU as a "technical means to a social end".[20]"</p>
GNU/Linux	Excerpted From https://www.gnu.org/gnu/linux-and-gnu.html :

TERM	DEFINITION
	<p>"Linux and the GNU System by Richard Stallman</p> <p>For more information see also the GNU/Linux FAQ, and Why GNU/Linux?</p> <p>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.</p> <p>There really is a Linux, and these people are using it, but it is just a part of the system they use. Linux is the kernel: the program in the system that allocates the machine's resources to the other programs that you run. The kernel is an essential part of an operating system, but useless by itself; it can only function in the context of a complete operating system. Linux is normally used in combination with the GNU operating system: the whole system is basically GNU with Linux added, or GNU/Linux. All the so-called "Linux" distributions are really distributions of GNU/Linux.</p> <p>Many users do not understand the difference between the kernel, which is Linux, and the whole system, which they also call "Linux". The ambiguous use of the name doesn't help people understand. These users often think that Linus Torvalds developed the whole operating system in 1991, with a bit of help.</p> <p>Programmers generally know that Linux is a kernel. But since they have generally heard the whole system called "Linux" as well, they often envisage a history that would justify naming the whole system after the kernel. For example, many believe that once Linus Torvalds finished writing Linux, the kernel, its users looked around for other free software to go with it, and found that (for no particular reason) most everything necessary to make a Unix-like system was already available.</p> <p>What they found was no accident: it was the not-quite-complete GNU system. The available free software added up to a complete system because the GNU Project had been working since 1984 to make one. In the The GNU Manifesto we set forth the goal of developing a free Unix-like system, called GNU. The Initial Announcement of the GNU Project also outlines some of the original plans for the GNU system. By the time Linux was started, GNU was almost finished.</p> <p>Most free software projects have the goal of developing a particular program for a particular job. For example, Linus Torvalds set out to write a Unix-like kernel (Linux); Donald Knuth set out to write a text formatter (TeX); Bob Scheifler set out to develop a window system (the X Window System). It's natural to measure the contribution of this kind of project by specific programs that came from the project.</p> <p>If we tried to measure the GNU Project's contribution in this way, what would we conclude? One CD-ROM vendor found that in their "Linux distribution", GNU software was the largest single contingent, around 28% of the total source code, and this included some of the essential major components without which there could be no system. Linux itself was about 3%. (The proportions in 2008 are similar: in the "main" repository of gNewSense, Linux is 1.5% and GNU packages are 15%.) So if you were going to pick a name for the system based on who wrote the programs in the system, the most appropriate single choice would be "GNU".</p> <p>But that is not the deepest way to consider the question. The GNU Project was not, is not, a project to develop specific software packages. It was not a project to develop a C compiler, although we did that. It was not a project to develop a text editor, although we developed one. The GNU Project set out to develop a complete free Unix-like system: GNU."</p> <p><i>The complete article is available at the aforementioned link.</i></p>
gnuwin32	<p>From Wikipedia, the free encyclopedia:</p> <p>"The GnuWin32 project provides native ports in the form of runnable computer programs, patches, and source code for various GNU and open source tools and software, much of it modified to run on</p>

TERM	DEFINITION
	<p>the 32-bit Windows platform. The ports included in the GnuWin32 packages are:</p> <ul style="list-style-type: none"> ▪ GNU utilities such as bc, bison, chess, Coreutils, diffutils, ed, Flex, gawk, gettext, grep, Groff, gzip, iconv, less, m4, patch, readline, rx, sharutils, sed, tar, texinfo, units, Wget, which ▪ Archive management and compression tools, such as: arc, arj, bzip2, gzip, lha, zip, zlib. ▪ Non-GNU utilities such as: cygutils, file, ntfsprogs, OpenSSL, PCRE. ▪ Graphics tools. ▪ PDCurses ▪ Tools for processing text. ▪ Mathematical software and statistics Software." <p>See also:</p> <ul style="list-style-type: none"> ▪ Cygwin ▪ DJGPP ▪ GNUWin II ▪ Microsoft Windows Services for UNIX ▪ MinGW, MSYS ▪ UnxUtils ▪ UWIN
GUI	<p>Acronym for Graphical User Interface.</p> <p>From Wikipedia, the free encyclopedia:</p> <p>"In computing, a graphical user interface (GUI, commonly pronounced gooey[1]) is a type of user interface that allows users to interact with electronic devices with images rather than text commands. GUIs can be used in computers, hand-held devices such as MP3 players, portable media players or gaming devices, household appliances and office equipment. A GUI represents the information and actions available to a user through graphical icons and visual indicators such as secondary notation, as opposed to text-based interfaces, typed command labels or text navigation. The actions are usually performed through direct manipulation of the graphical elements.[2]"</p>
High Level API	<p>A programming interface 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.</p>
Hypervisor	<p>From Wikipedia, the free encyclopedia</p> <p>"In computing, a hypervisor, also called virtual machine manager (VMM), is one of many hardware virtualization techniques allowing multiple operating systems, termed guests, to run concurrently on a host computer. It is so named because it is conceptually one level higher than a supervisory program. The hypervisor presents to the guest operating systems a virtual operating platform and manages the execution of the guest operating systems. Multiple instances of a variety of operating systems may share the virtualized hardware resources. Hypervisors are very commonly installed on server hardware, with the function of running guest operating systems, that themselves act as servers.</p> <p>The term can be used to describe the interface provided by the specific cloud computing functionality infrastructure as a service (IaaS).[1][2]</p> <p>The term "hypervisor" was first used in 1965, referring to software that accompanied an IBM RPQ for the IBM 360/65. It allowed the model IBM 360/65 to share its memory: half acting as an IBM</p>

TERM	DEFINITION
	360 and half as an emulated IBM 7080. The software, labeled "hypervisor," did the switching between the two modes on split-time basis. The term hypervisor was coined as an evolution of the term "supervisor," the software that provided control on earlier hardware.[3][4]"
Instrumentation	The application of technology for the measurement and control of process variables within a production or manufacturing area. An instrument is a device that measures a physical quantity such as flow, temperature, level, distance, angle, or pressure. Instruments may be as simple as direct reading thermometers or may be complex multi-variable process analyzers. Instruments are often part of a control system in refineries, factories, and vehicles.
Interface Requirements Specification	The Interface Requirements Specification (IRS) specifies the requirements imposed on one or more systems, subsystems, Hardware Configuration Items (HWCIs), Computer Software Configuration Items (CSCIs), manual operations, or other system components to achieve one or more interfaces among these entities. An IRS can cover any number of interfaces.
Linux	<p>From Wikipedia, the free encyclopedia</p> <p>"This article is about the operating system. For the kernel, see Linux kernel.</p> <p>Linux [8][9][10] is a Unix-like computer operating system assembled under the model of free and open source software development and distribution. The defining component of GNU/Linux Linux is the Linux kernel, an operating system kernel first released 5 October 1991 by Linus Torvalds.[11][12]"</p> <p>NOTE:</p> <ul style="list-style-type: none"> ▪ The Free Software Foundation (FSF) is responsible for the GNU Project and maintains that Linux Linux, when used with GNU tools and utilities, should be considered a variant of GNU, and promotes the term Linux for such systems (leading to the Linux naming controversy).
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.
MAC OS X	<p>From Wikipedia, the free encyclopedia</p> <p>"Mac OS is a series of graphical user interface-based operating systems developed by Apple Inc. (formerly Apple Computer, Inc.) for their Macintosh line of computer systems. Mac OS is credited with popularizing the graphical user interface. The original form of what Apple would later name the "Mac OS" (currently OSX) was the integral and unnamed system software first introduced in 1984 with the original Macintosh, usually referred to simply as the System software."</p>
ManPage	A manpage (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.
Massachusetts Institute of Technology License	<p>From Wikipedia, the free encyclopedia</p> <p>"The MIT License is a free software license originating at the Massachusetts Institute of Technology (MIT). It is a permissive free software license, meaning that it permits reuse within proprietary software provided all copies of the licensed software include a copy of the MIT License terms. Such proprietary software retains its proprietary nature even though it incorporates software under the MIT License. The license is also GPL-compatible, meaning that the GPL permits combination and redistribution with software that uses the MIT License.[1]</p> <p>Notable software packages that use one of the versions of the MIT License include Expat, PuTTY, the Mono development platform class libraries, Ruby on Rails, Lua (from version 5.0 onwards), Wayland and the X Window System, for which the license was written."</p>

TERM	DEFINITION
Microsoft Windows	<p>From Wikipedia, the free encyclopedia</p> <p>"Microsoft Windows is a series of graphical interface operating systems developed, marketed, and sold by Microsoft.</p> <p>Microsoft introduced an operating environment named Windows on November 20, 1985 as an add-on to MS-DOS in response to the growing interest in graphical user interfaces (GUIs).[2] 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."</p>
MIL-STD-498	<p>From Wikipedia, the free encyclopedia</p> <p>"MIL-STD-498 (Military-Standard-498) was a United States military standard whose purpose was to "establish uniform requirements for software development and documentation." It was released Nov. 8, 1994, and replaced DOD-STD-2167A, DOD-STD-7935A, and DOD-STD-1703. It was meant as an interim standard, to be in effect for about two years until a commercial standard was developed.</p> <p>Unlike previous efforts like the seminal "2167A" which was mainly focused on the risky new area of software development, "498" was the first attempt at a truly comprehensive description of the systems development life-cycle. It was the baseline that all of the ISO, IEEE, and related efforts after it replaced. It also contains much of the material that the subsequent professionalization of project management covered in the Project Management Body of Knowledge (PMBOK). The document "MIL-STD-498 Overview and Tailoring Guidebook" is 98 pages. The "MIL-STD-498 Application and Reference Guidebook" is 516 pages. Associated to these were document templates, or Data Item Descriptions, described below, bringing documentation and process order that could scale to projects of the size humans were then conducting (aircraft, battleships, canals, dams, factories, satellites, submarines, etcetera).</p> <p>It was one of the few military standards that survived the "Perry Memo", then U.S. Secretary of Defense William Perry's 1994 memorandum commanding the discontinuation of defense standards. However, it was canceled on May 27, 1998 and replaced by the essentially identical demilitarized version EIA J-STD-016[1] [2] as a process example guide for IEEE 12207. Several programs outside of the U.S. military continued to use the standard due to familiarity and perceived advantages over alternative standards, such as free availability of the standards documents and presence of process detail including contractually-usable Data Item Descriptions."</p> <hr/> <p>From http://everyspec.com/MIL-STD/MIL-STD-0300-0499/MIL-STD-498_25500/</p> <p>"MIL-STD-498, MILITARY STANDARD: SOFTWARE DEVELOPMENT AND DOCUMENTATION (05 DEC 1994) [SUPERSEDING MIL-STD-2167A, DOD-STD-7935A & DOD-STD-1703] [S/S BY IEEE/EIA 12207.0, IEEE/EIA 12207.1 & IEEE/EIA 12207.2]., The purpose of this standard is to establish uniform requirements for software development and documentation. This standard merges DOD-STD-2167A and DOD-STD-7935A to define a set of activities and documentation suitable for the development of both weapon systems and Automated Information Systems. A conversion guide from these standards to MIL-STD-498 is provided in Appendix I. Other changes include improved compatibility with incremental and evolutionary development models; improved compatibility with non-hierarchical design methods; improved compatibility with computer-aided software engineering (CASE) tools; alternatives to, and more flexibility in, preparing documents; clearer requirements for incorporating reusable software; introduction of software management indicators; added emphasis on software supportability; and improved links to systems engineering. This standard supersedes DOD-STD-2167A, DOD-STD- 7935A, and DOD-STD-1703</p>

TERM	DEFINITION
	(NS)." <u>A copy of the specification is available via a download link.</u>
Module	The basic unit of code reusability in Python: a block of code imported by some other code. Three types of modules concern us here: pure Python modules, extension modules, and packages.
nCurses	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."
Non-Functional Requirements	From Wikipedia, the free encyclopedia: See Functional Requirements for definition and relationship.
Notebooks	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.).
OpenIndiana	From Wikipedia, the free encyclopedia "OpenIndiana is a Unix-like computer operating system released as free and open source software. It forked from OpenSolaris after the discontinuation of that project by Oracle[1] and aims to continue development and distribution of the OpenSolaris codebase.[2] The project operates under the umbrella of the Illumos Foundation.[2] The stated aim of the project is "[...] to become the defacto OpenSolaris distribution installed on production servers where security and bug fixes are required free of charge".[3]"
OpenSolaris	From Wikipedia, the free encyclopedia "OpenSolaris (...) was an open source computer operating system based on Solaris created by Sun Microsystems. It was also the name of the project initiated by Sun to build a developer and user community around the software. After the acquisition of Sun Microsystems in 2010, Oracle decided to discontinue open development of the core software, and replaced the OpenSolaris distribution model with the proprietary Solaris Express. Prior to Oracle's moving of core development "behind closed doors", a group of former OpenSolaris developers decided to "fork" the core software under the name OpenIndiana. The project, a part of the Illumos Foundation, aims to continue the development and distribution of the OpenSolaris codebase.[5] OpenSolaris is 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.[6] OpenSolaris is developed as a combination of several software consolidations that were open sourced subsequent to Solaris 10. It includes a variety of free software, including popular desktop and server software.[7][8] On Friday, August 13, 2010, details started to emerge relating to the restructuring of the OpenSolaris project, the pending release of the new future commercial version of Solaris, Solaris 11, and how open source community interactions are being adjusted.[9]"
Package	A module that contains other modules; typically contained in a directory in the filesystem and distinguished from other directories by the presence of a file <code>__init__.py</code> .
Parallels Desktop for Mac	From Wikipedia, the free encyclopedia "Parallels Desktop for Mac by Parallels, Inc., is software providing hardware virtualization for

TERM	DEFINITION
	<p>Macintosh computers with Intel processors.</p> <p>Technical</p> <p>Parallels Desktop for Mac is a hardware emulation virtualization software, using hypervisor technology that works by mapping the host computer's hardware resources directly to the virtual machine's resources. Each virtual machine thus operates identically to a standalone computer, with virtually all the resources of a physical computer.[3] Because all guest virtual machines use the same hardware drivers irrespective of the actual hardware on the host computer, virtual machine instances are highly portable between computers. For example, a running virtual machine can be stopped, copied to another physical computer, and restarted."</p> <p>Parallels Tools</p> <p>Parallels Tools are a suite of behind-the-scenes tools that allow seamless operating between Mac OS X and Windows or another guest operating system.</p> <p>Each Parallels release includes its own Parallels Tools. Those tools interface the Guest Operating System's Virtual Machine to the Parallels Desktop installed on the host computer. The Tools enable the Host and Guest OS to share resources.</p> <p>After upgrading the Parallels Desktop from 9 to 10 on one host computer it was still possible to run recent copies of the Paralels 10 Guest OS virtual machines on a host computer that could not be upgraded to Parallels 10 simply by:</p> <ul style="list-style-type: none"> ▪ Attaching a hard drive with the Parallels 10 Guest OS virtual machine. ▪ Manually changing the Parallels Guest OS configuration to suit the available host memory and devices. ▪ Allowing Parallels to automatically (or manually initiating) re-install of the available older Parallels (8 or 9) Tools. ▪ Launching the re-configured Parallels Guest OS.
PC-BSD or PCBSD	<p>From Wikipedia, the free encyclopedia</p> <p>"PC-BSD, or PCBSD, is 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 [1] as the graphical user interface. It provides official binary nVidia and Intel drivers for hardware acceleration and an optional 3D desktop interface through Kwin, and Wine is ready-to-use in running Microsoft Windows software. PC-BSD is able to run Linux software,[2] in addition to FreeBSD ports, and it has its own package management system that allows users to graphically install pre-built software packages from a single downloaded executable file, which is unique for BSD operating systems.</p> <p>PC-BSD supports ZFS, and the installer offers disk encryption with geli so the system will require a passphrase before booting."</p>
PDCurses	<p>PDCurses is an implementation of the curses library for X11. It provides the ability for existing text-mode curses programs to be re-built as native X11 applications with very little modification. PDCurses for X11 is also known as XCURSES.</p> <p>It is available from http://pdcurses.sourceforge.net. Version 2.6 enables Python applications launched within the Windows Command Prompt shell to import and use the Python Curses module.</p> <p>However, Version 2.6 cannot be used by the "tsWxGraphicalTextUserInterface" module to emulate "wxPython" because it lacks the mouse button definitions and interface features available with Unix-like shells such as bash.</p>
POSIX	<p>From Wikipedia, the free encyclopedia</p>

TERM	DEFINITION
	<p>"Not to be confused with Unix, Unix-like, or Linux.</p> <p>An acronym for "Portable Operating System Interface", 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.[1][2]"</p>
Programming Tools or Software Development Tools	<p>From Wikipedia, the free encyclopedia</p> <p>"A programming tool or software development tool is a computer program that software developers use to create, debug, maintain, or otherwise support other programs and applications. The term usually refers to relatively simple programs, that can be combined together to accomplish a task, much as one might use multiple hand tools to fix a physical object. The ability to use a variety of tools productively is one hallmark of a skilled software engineer.</p> <p>The most basic tools are a source code editor and a compiler or interpreter, which are used ubiquitously and continuously. Other tools are used more or less depending on the language, development methodology, and individual engineer, and are often used for a discrete task, like a debugger or profiler. Tools may be discrete programs, executed separately – often from the command line – or may be parts of a single large program, called an integrated development environment (IDE). In many cases, particularly for simpler use, simple ad hoc techniques are used instead of a tool, such as print debugging instead of using a debugger, manual timing (of overall program or section of code) instead of a profiler, or tracking bugs in a text file or spreadsheet instead of a bug tracking system.</p> <p>The distinction between tools and applications is murky. For example, developers use simple databases (such as a file containing a list of important values) all the time as tools.[dubious – discuss] However a full-blown database is usually thought of as an application or software in its own right. For many years, computer-assisted software engineering (CASE) tools were sought after. Successful tools have proven elusive.[citation needed] In one sense, CASE tools emphasized design and architecture support, such as for UML. But the most successful of these tools are IDEs."</p>
Pure Python Module	<p>A module written in Python and contained in a single .py file (and possibly associated .pyc and/or .pyo files). Sometimes referred to as a "pure module."</p>
Python	<p>From Wikipedia, the free encyclopedia:</p> <p>"Python is a general-purpose, high-level programming language[11] whose design philosophy emphasizes code readability.[12] Python claims to combine "remarkable power with very clear syntax",[13] and its standard library is large and comprehensive.</p> <p>Python supports multiple programming paradigms, primarily but not limited to object-oriented, imperative and, to a lesser extent, functional programming styles. It features a fully dynamic type system and automatic memory management, similar to that of Scheme, Ruby, Perl, and Tcl. 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. Python interpreters are available for many operating systems.</p> <p>The reference implementation of Python (CPython) is free and open source software and has a community-based development model, as do all or nearly all of its alternative implementations. CPython is managed by the non-profit Python Software Foundation."</p>
Python Software Foundation License	<p>From Wikipedia, the free encyclopedia:</p> <ul style="list-style-type: none"> ▪ "Python Software Foundation License FSF approved Yes [1] ▪ OSI approved Yes ▪ GPL compatible Yes [1]

TERM	DEFINITION
	<p>▪ Copyleft No</p> <p>The Python Software Foundation License (PSFL) is a BSD-style, permissive free software license which is compatible with the GNU General Public License (GPL).[1] Its primary use is for distribution of the Python project software. Unlike the GPL the Python license is not a copyleft license, and allows modifications to the source code, as well as the construction of derivative works, without making the code open-source. The PSFL is listed as approved on both FSF's approved licenses list,[1] and OSI's approved licenses list.</p> <p>Earlier versions of Python were under the so-called Python License, which is incompatible with the GPL. The reason given for this incompatibility by Free Software Foundation was that "this Python license is governed by the laws of the 'State of Virginia', in the USA", and the GPL does not permit this.[2]</p> <p>The year that Python's creator Guido van Rossum changed the license to fix this incompatibility, he was awarded the Free Software Foundation Award for the Advancement of Free Software.[3]"</p>
Python Virtual Machine	<p>A virtual machine designed to interpret and execute programs implemented in the Python programming language.</p> <p>From Wikipedia, the free encyclopedia:</p> <p>"A virtual machine (VM) is a software implementation of a machine (i.e. a computer) that executes programs like a physical machine. Virtual machines are separated into two major categories, based on their use and degree of correspondence to any real machine. A system virtual machine provides a complete system platform which supports the execution of a complete operating system (OS). In contrast, a process virtual machine is designed to run a single program, which means that it supports a single process. An essential characteristic of a virtual machine is that the software running inside is limited to the resources and abstractions provided by the virtual machine—it cannot break out of its virtual world.</p> <p>A virtual machine was originally defined by Popek and Goldberg as "an efficient, isolated duplicate of a real machine". Current use includes virtual machines which have no direct correspondence to any real hardware.[2]"</p>
Repository	<p>Excerpt From Wikipedia, the free encyclopedia:</p> <p>"A software repository is a storage location from which software packages may be retrieved and installed on a computer."</p> <p>The TeamSTARS "tsWxGTUI_PyVx" Toolkit repository 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.</p>
Root Package	<p>The root of the hierarchy of packages. (This isn't really a package, since it doesn't have an <code>__init__.py</code> file. But we have to call it something.) The vast majority of the standard library is in the root package, as are many small, standalone third-party modules that don't belong to a larger module collection. Unlike regular packages, modules in the root package can be found in many directories: in fact, every directory listed in <code>sys.path</code> contributes modules to the root package.</p>
Simulation	<p>The application of technology for the imitation of the operation of a real-world process or system over time. The act of simulating something first requires that a model be developed; this model represents the key characteristics or behaviors/functions of the selected physical or abstract system or process. The model represents the system itself, whereas the simulation represents the operation of the system over time.</p>

TERM	DEFINITION
Site-Package	<p>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.</p> <p>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.</p>
Software Engineer	<p>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.</p> <p>The term also applies to those individuals who perform similar engineering activities associated with communication, data base, simulation, operating system, device driver, test and diagnostic components.</p>
Software Requirements Specification	<p>The Software Requirements Specification (SRS) specifies the requirements for a Computer Software Configuration Item (CSCI) and the methods to be used to ensure that each requirement has been met. Requirements pertaining to the CSCI's external interfaces may be presented in the SRS or in one or more Interface Requirements Specifications (IRSs) (DI-IPSC-81434) referenced from the SRS.</p>
Solaris	<p>From Wikipedia, the free encyclopedia</p> <p>"Solaris is a Unix operating system originally developed by Sun Microsystems. It superseded their earlier SunOS in 1993. Oracle Solaris, as it is now known, has been owned by Oracle Corporation since Oracle's acquisition of Sun in January 2010.[2]</p> <p>Solaris is known for its scalability, especially on SPARC systems, and for originating many innovative features such as DTrace, ZFS and Time Slider.[3][4] Solaris supports SPARC-based and x86-based workstations and servers from Sun and other vendors, with efforts underway to port to additional platforms. Solaris is registered as compliant with the Single Unix Specification.</p> <p>Solaris was historically developed as proprietary software, then in June 2005 Sun Microsystems released most of the codebase under the CDDL license, and founded the OpenSolaris open source project.[5] With OpenSolaris, Sun wanted to build a developer and user community around the software. After the acquisition of Sun Microsystems in January 2010, Oracle decided to discontinue the OpenSolaris distribution and the development model.[6][7] Just ten days before the internal Oracle memo announcing this decision to employees was "leaked", Garrett D'Amore had announced[8] the illumos project, creating a fork of the Solaris kernel and launching what has since become a thriving alternative to Oracle Solaris.</p> <p>In August 2010, Oracle discontinued providing public updates to the source code of the Solaris Kernel, effectively turning Solaris 11 into a closed source proprietary operating system. However, through the Oracle Technology Network (OTN), industry partners can still gain access to the in-development Solaris source code.[7] The Open source portion of Solaris 11 is available for download from Oracle.[9]"</p>

TERM	DEFINITION
Source Code	<p>Excerpt From Wikipedia, the free encyclopedia:</p> <p>"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.</p> <p>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.</p> <p>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."</p>
Stakeholder	Those persons, groups or organizations with an interest in a project.
System Administration Utilities	Computer programs that computer system administrators use to install, debug, maintain, or otherwise support computer hardware and software.
System Administrator	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 communication network to be used by Software Engineers and System Operators.
System Operator	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.
TDD	<p>Acronym for Test-Driven Development</p> <p>from: http://encyclopedia.thefreedictionary.com/Test+Driven+Development</p> <p>"Test-driven development (TDD) is a software development process that relies on the repetition of a very short development cycle: first the developer writes a failing automated test case that defines a desired improvement or new function, then produces code to pass that test and finally refactors the new code to acceptable standards. Kent Beck, who is credited with having developed or 'rediscovered' the technique, stated in 2003 that TDD encourages simple designs and inspires confidence.[1]</p> <p>Test-driven development is related to the test-first programming concepts of extreme programming, begun in 1999,[2] but more recently has created more general interest in its own right.[3]</p> <p>Programmers also apply the concept to improving and debugging legacy code developed with older techniques.[4]"</p>
tsToolKitCLI	<p>The identifier for a package of toolkit components for a Python-based Command Line Interface. The library is further subdivided into the following components:</p> <ul style="list-style-type: none"> ▪ tsLibCLI - library of command line building blocks that establishes the 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. ▪ tsTestsCLI - a set of command line interface application programs and scripts for regression testing and tutorial demos.

TERM	DEFINITION
	<ul style="list-style-type: none"> ▪ 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. ▪ tsUtilities - a library of computer system configuration, diagnostic, installation, maintenance and performance tool components for various host hardware and software platforms.
Terminal Emulator	<p>Excerpt from Wikipedia, the free encyclopedia</p> <p>"A program that emulates a video terminal within some other display architecture. Though typically synonymous with a shell or text terminal, the term terminal covers all remote terminals, including graphical interfaces. A terminal emulator inside a graphical user interface is often called a terminal window."</p>
tsToolkitGUI	<p>The identifier for a package of toolkit components for a "wxPython"-style, "nCurses"-based Graphical-style User Interface. The library is further subdivided into the following components:</p> <ul style="list-style-type: none"> ▪ 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 ▪ tsTestsGUI - a set of graphical-style user interface application programs for regression testing and tutorial demos. ▪ tsToolsGUI - a set of graphical-style user interface application programs for tracking software development metrics.
tsLibCLI	The identifier for a library of building-block components for a Python-based Command Line Interface.
tsLibGUI	The identifier for a library of building-block components for a "wxPython"-style, "nCurses"-based Graphical-style User Interface.
tsTestsCLI	The identifier for a library of qualification test components for a Python-based Command Line Interface.
tsTestsGUI	The identifier for a library of qualification test components for a "wxPython"-style, "nCurses"-based Graphical-style User Interface.
tsToolsCLI	The identifier for a library of software development, diagnostic, maintenance and project management components for a Python-based Command Line Interface.
tsToolsGUI	The identifier for a library of software development, diagnostic, maintenance and project management components for a "wxPython"-style, "nCurses"-based Graphical-style User Interface.
tsUtilities	The identifier for a library of computer system configuration, diagnostic, installation, maintenance and performance tool components for various host hardware and software platforms.
tsWxGTUI	<p>The identifier for a library of building-block components for a "wxPython"-style, "nCurses"-based Graphical-style User Interface (tsToolkitGUI) and Python-based Command Line Interface (tsToolkitCLI).</p> <p>The tsToolkitGUI and tsToolkitCLI component organization keeps the Command Line Interface components independent of the Graphical-style User Interface ones. However, it does not preclude the Graphical-style User Interface components from using the services of the Command Line Interface components as appropriate.</p>
tsWxGTUI_PyVx	The generic identifier for the following Python language generation specific <i>TeamSTARS</i> "tsWxGTUI" Toolkit releases:

TERM	DEFINITION
	<ul style="list-style-type: none"> ▪ tsWxGTUI_Py2x --- Python 2.0.0 - 2.7.9 (Toolkit for 2nd generation Python language) ▪ tsWxGTUI_Py3x --- Python 3.0.0 - 3.4.2 (Toolkit for 3rd generation Python language)
tsWxGTUI_PyVx -Major .Minor .Maintenance	<p>The Major-Minor-Maintenance identifier for a <i>TeamSTARS</i> "tsWxGTUI" Toolkit release where:</p> <ul style="list-style-type: none"> ▪ Major --- A version number that denotes the introduction of a new design which cannot be expected to be backward compatible to a previous version. Zero (0) denotes the initial release for a product preview during its pre-alpha or beta stage. ▪ Minor --- A version number that denotes a product update that selectively introduces new features but otherwise can be expected to be backward compatible to a previous version. Zero (0) denotes the initial release. ▪ Maintenance --- A version number that denotes a product update that corrects defects found after the release of a previous version and otherwise can be expected to be backward compatible to that previous version. Zero (0) denotes a product release in its pre-alpha stage.
UNIX	<p>From Wikipedia, the free encyclopedia</p> <p>"Unix (officially trademarked as UNIX, sometimes also written as Unix) is a multitasking, multi-user computer operating system originally developed in 1969 by a group of AT&T employees at Bell Labs, including Ken Thompson, Dennis Ritchie, Brian Kernighan, Douglas McIlroy, Michael Lesk and Joe Ossanna. The Unix operating system was first developed in assembly language, but by 1973 had been almost entirely recoded in C, greatly facilitating its further development and porting to other hardware. Today's Unix system evolution is split into various branches, developed over time by AT&T as well as various commercial vendors, universities (such as University of California, Berkeley's BSD), and non-profit organizations."</p>
UWIN	<p>From Wikipedia, the free encyclopedia</p> <p>"UWIN is a computer software package created by David Korn which allows programs written for the operating system Unix be built and run on Microsoft Windows with few, if any, changes. Some of the software development was subcontracted to Wipro, India. References, correct or not, to the software as U/Win and AT&T Unix for Windows can be found in some cases, especially from the early days of its existence.</p> <p>UWIN source and binaries are available under the Open Source Eclipse Public License 1.0 at AT&T AST/UWIN open source downloads."</p> <p>See also:</p> <ul style="list-style-type: none"> ▪ Cygwin, a similar project started at about the same time[2] ▪ Interix, the Microsoft product in this area ▪ Windows Services for Unix ▪ MKS Toolkit, a third-party proprietary product in this area ▪ DJGPP (DJ's GNU Programming Platform), a Windows port of Gnu programming tools ▪ Xming ▪ UnxUtils ▪ GnuWin32 ▪ GNUWin II ▪ MinGW ▪ LBW: Linux Binaries on Windows requires Interix to be installed first."
VMware Fusion	<p>From Wikipedia, the free encyclopedia</p>

TERM	DEFINITION
	<p>"VMware Fusion is a software hypervisor developed by VMware for Macintosh computers with Intel processors. Fusion allows Intel-based Macs to run operating systems, such as Microsoft Windows, Linux, NetWare or Solaris on virtual machines, along with their Mac OS X operating system using a combination of paravirtualization, hardware virtualization and dynamic recompilation."</p>
VNC	<p>From Wikipedia, the free encyclopedia:</p> <p>"In computing, Virtual Network Computing (VNC) is a graphical desktop sharing system that uses the Remote Frame Buffer protocol (RFB) to remotely control another computer. It transmits the keyboard and mouse events from one computer to another, relaying the graphical screen updates back in the other direction, over a network.[1]</p> <p>VNC is platform-independent – a VNC viewer on one operating system may connect to a VNC server on the same or any other operating system. There are clients and servers for many GUI-based operating systems and for Java. Multiple clients may connect to a VNC server at the same time. Popular uses for this technology include remote technical support and accessing files on one's work computer from one's home computer, or vice versa.</p> <p>VNC was originally developed at the Olivetti & Oracle Research Lab in Cambridge, United Kingdom. The original VNC source code and many modern derivatives are open source under the GNU General Public License.</p> <p>VNC in KDE 3.1</p> <p>There are a number of variants of VNC[2] which offer their own particular functionality; e.g., some optimised for Microsoft Windows, or offering file transfer (not part of VNC proper), etc. Many are compatible (without their added features) with VNC proper in the sense that a viewer of one flavour can connect with a server of another; others are based on VNC code but not compatible with standard VNC.</p> <p>VNC and RFB are registered trademarks of RealVNC Ltd. in the U.S. and in other countries."</p>
vt100	<p>From Wikipedia, the free encyclopedia:</p> <p>"The VT100 is a video terminal that was made by Digital Equipment Corporation (DEC). Its detailed attributes became the de facto standard for terminal emulators to emulate.</p> <p>History</p> <p>It was introduced in August 1978, following its predecessor, the VT52, and communicated with its host system over serial lines using the ASCII character set and control sequences (a.k.a. escape sequences) standardized by ANSI. The VT100 was also the first Digital mass-market terminal to incorporate "graphic renditions" (blinking, bolding, reverse video, and underlining) as well as a selectable 80 or 132 column display. All setup of the VT100 was accomplished using interactive displays presented on the screen; the setup data was stored in non-volatile memory within the terminal. The VT100 also introduced an additional character set that allowed the drawing of on-screen forms.</p> <p>The control sequences used by the VT100 family are based on the ANSI X3.64 standard, also known as ECMA-48 and ISO/IEC 6429. These are sometimes referred to as ANSI escape codes. The VT100 was not the first terminal to be based on X3.64—The Heath Company had a microprocessor-based video terminal, the Heathkit H-19 (H19), that implemented a subset of the standard proposed by ANSI in X3.64.[1] In addition, the VT100 provided backwards compatibility for VT52 users, with support for the VT52 control sequences.[2]</p> <p>In 1983, the VT100 was replaced by the more-powerful VT200 series terminals such as the VT220. In August 1995 the terminal business of Digital was sold to Boundless Technologies.[3]"</p>

TERM	DEFINITION
vt220	<p>From Wikipedia, the free encyclopedia:</p> <p>"DEC VT220 terminal with LK201 keyboard.</p> <p>The VT220 was a terminal produced by Digital Equipment Corporation from 1983 to 1987.[1][2]</p> <p>Hardware</p> <p>The VT220 improved on the earlier VT100 series of terminals with a redesigned keyboard, much smaller physical packaging, and a much faster microprocessor. To meet the needs of various national regulatory agencies[citation needed], the VT220 was available with CRTs that used white, green, or amber phosphors.</p> <p>Several of the VT2xx models were pyramid shaped, allowing them to sit on a table top, leaving the surface of the display at an angle to the user; this angle could be adjusted. Because it was lower than head height, the result was an especially ergonomic terminal. The LK201 keyboard supplied with the VT220 was one of the first full length low profile keyboards available; it was developed at DEC's Roxbury, Massachusetts facility.</p> <p>The VT240 and VT241 were variants of the VT220, both capable of displaying vector graphics using the ReGIS instruction set. The VT241 was equipped with a color screen. The successor of the VT220 was the VT320, itself followed by the VT420.</p> <p>DEC VT220 connected to the serial port of a modern computer.</p> <p>Software</p> <p>The VT220 was designed to be compatible with the VT100, but added features to make it more suitable for an international market. This was accomplished with the National Replacement Character Set feature (e.g., Multinational Character Set) and support for 8-bit downloadable character sets."</p>
wxEmbedded	<p>From http://www.koansoftware.com/en/content/wxembedded:</p> <p>"wxEmbedded</p> <p>What is wxEmbedded®</p> <p>wxEmbedded® name and logo are registered trademarks of KOAN Software</p> <p>wxEmbedded - Embedded crossplatform GUI Library</p> <p>On March 2002 Koan software announced that a new project has been started by our company with wx-developers group.</p> <p>"After years of wishing, several recent projects have brought this closer to being a reality thanks to KOAN who has given the motivation behind all wxEmbedded project"</p> <p>-- Julian Smart, wxWidgets founder</p> <p>Here are the current strands in the wxEmbedded strategy, some points are already working, some are works in progress</p> <ul style="list-style-type: none"> ▪ wxWidgets for X11 (wxX11) ▪ wxWidgets for GTK+ (wxGTK) ▪ wxWidgets for Nano-X (wxNano-X) ▪ wxWidgets for Microwindows (wxMicrowindows) ▪ wxWidgets for SciTech MGL (wxMGL) ▪ wxWidgets for MS Windows CE (wxWinCE)

TERM	DEFINITION
	<ul style="list-style-type: none"> Host tools, such as wxEmulator <p>Platforms supported are : x86 and ARM"</p>
wxPython	A popular Python language binding for the cross-platform, "wxWidgets" GUI Toolkit.
wxWidgets	<p>A C++ library that lets developers create applications for Windows, OS X, Linux 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>It has popular language bindings for Python, Perl, Ruby and many other languages.</p> <p>"wxWidgets" (formerly "wxWindows") is a widget toolkit 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, Mac OS X (Carbon and Cocoa), iOS (Cocoa Touch), GNU/Linux Linux/Unix (X11, Motif, and GTK+), OpenVMS, OS/2 and AmigaOS. A version for embedded systems is under development.</p>
wxWindows License	<p>From Wikipedia, the free encyclopedia</p> <p>'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]"</p>
xterm	<p>From Wikipedia, the free encyclopedia:</p> <p>"Not to be confused with X terminal display hardware.</p> <p>In computing, xterm is the standard terminal emulator for the X Window System. A user can have many different invocations of xterm running at once on the same display, each of which provides independent input/output for the process running in it (normally the process is a Unix shell).</p> <p>xterm originated prior to the X Window System. It was originally written as a stand-alone terminal emulator for the VAXStation 100 (VS100) by Mark Vandevoorde, a student of Jim Gettys, in the summer of 1984, when work on X started. It rapidly became clear that it would be more useful as part of X than as a standalone program, so it was retargeted to X. As Gettys tells the story, "part of why xterm's internals are so horrifying is that it was originally intended that a single process be able to drive multiple VS100 displays." [2]</p> <p>After many years as part of the X reference implementation, around 1996 the main line of development then shifted to XFree86 (which itself forked from X11R6.3), and it is presently actively maintained by Thomas Dickey.</p> <p>Many xterm variants are also available.[3] Most terminal emulators for X started as variations on xterm."</p> <p>NOTES:</p> <p>The "tsWxGTUI_PyVx" Toolkit supports the xterm, xterm-color, xterm-16color and xterm-256color terminal emulators with the following limitations:</p> <ul style="list-style-type: none"> The "tsWxGTUI_PyVx" Toolkit supports the xterm, xterm-color and xterm-16color terminal emulators. The inability to change the curses palette made it necessary to map the 68-color wxPython palette into the default 8-color curses palette. The "tsWxGTUI_PyVx" Toolkit does NOT yet support the xterm-256color terminal emulator. The inability to recreate the 68-color wxPython palette results in inappropriate colors and the

TERM	DEFINITION
	appearance of spurious lines streaking across the display.

Draft

Draft

7 REFERENCED DOCUMENTS

This section shall list the number, title, revision, and date of all documents referenced in this plan. This section shall also identify the source for all documents not available through normal Government stocking activities.

- **Project Documents** (on page 287)
- **Release Distribution Documents** (on page 290)
- **External Documents** (on page 294)

7.1 Project Documents

NOTES:

- 1) The set of printed or printable non-proprietary pages must carry a "DRAFT" watermark, if not approved for publication.
- 2) The set of printed or printable proprietary pages must carry a "CONFIDENTIAL" watermark, if intended for external use under a non-disclosure agreement whether or not approved for publication.
- 3) The set of printed or printable documents must all carry the same revision number and date.

NUMBER	TITLE (tsWxGTUI Toolkit)	REVISION	DATE	SOURCE
0	Vol. __0__ - _SDIST_Announcement <ul style="list-style-type: none"> ▪ This document alerts potential and existing users of the availability, capabilities, limitations and sources for the latest source code release and technical support. ▪ It is an advertising piece, typically mailed to the Python-announce-list mailing list. ▪ It introduces potential users to the goals, non-goals, known issues and disclaimer associated with the software. ▪ It describes how to get the software and technical support. 	Draft 0.0.0 (Pre-Appha)	07/19/2014	Software Gadgetry
1	Vol. __1__ - _SDIST_Brochure <ul style="list-style-type: none"> ▪ This document is an advertising piece, 	ditto	ditto	ditto

NUMBER	TITLE (tsWxGTUI Toolkit)	REVISION	DATE	SOURCE
	typically mailed or handed out, used to introduce a company or organization, and inform about products and/or services to a target audience.			
2	Vol. __2 - _SDIST_Introduction <ul style="list-style-type: none"> This document orients the reader to the goals and non-goals for the "tsWxGTUI_PyVx" Toolkit. 	ditto	ditto	ditto
3	Vol. __3 - _SDIST_Terms & Conditions <ul style="list-style-type: none"> This document alerts potential users and reminds existing users to the rules which the user must agree to abide by in order to reproduce or use the "tsWxGTUI_PyVx" Toolkit and/or any of its components. 	ditto	ditto	ditto
4	Vol. __4 - _SDIST_Development_Plan <ul style="list-style-type: none"> This document (SDP) describes the developer's plans for conducting a software development effort. The effort includes new development, modification, reuse, reengineering, maintenance, and all other activities resulting in software products. It provides the acquirer insight into, and a tool for monitoring the processes to be followed for software development, the methods to be used, the approach to be followed for each activity, and project schedules, organization, and resources. 	ditto	ditto	ditto
5	Vol. __5 - _SDIST_System_Specification <ul style="list-style-type: none"> The System/Subsystem Specification (SSS) specifies the requirements for a system or subsystem and the methods to be used to ensure that each requirement has been met. Requirements pertaining to the system or subsystem's external interfaces may be presented in the SSS or in one or more Interface Requirements Specifications (IRSs) (DI-IPSC-81434) referenced from the SSS. The SSS, possibly supplemented by IRSs, is used as the basis for design and qualification testing of a system or subsystem. Throughout this DID, the term "system" may be interpreted to mean "subsystem" as applicable. The resulting document should be titled System Specification or Subsystem 	ditto	ditto	ditto

NUMBER	TITLE (tsWxGTUI Toolkit)	REVISION	DATE	SOURCE
	Specification (SSS).			
6	Vol. 6 - SDIST_Interface_Requirements_Specification <ul style="list-style-type: none"> This document (IRS) specifies the required interface features of an engineering system. It typically describes the required inputs, outputs and any protocols for transferring information between external and internal system components. 	ditto	ditto	ditto
7	Vol. 7 - SDIST_Software_Requirements_Specification <ul style="list-style-type: none"> This document (SRS) specifies the requirements for a Computer Software Configuration Item (CSCI) and the methods to be used to ensure that each requirement has been met. Requirements pertaining to the CSCI's external interfaces may be presented in the SRS or in one or more Interface Requirements Specifications (IRSs) (DI-IPSC-81434) referenced from the SRS. The SRS, possibly supplemented by IRSs, is used as the basis for design and qualification testing of a CSCI. 	ditto	ditto	ditto
8	Vol. 8 - SDIST_Release Notes <ul style="list-style-type: none"> This document is distributed with software products, often when the product is still in the development or test state (e.g., a beta release). For products that have already been in use by clients, the release note is a supplementary document that is delivered to the customer when a bug is fixed or an enhancement is made to the product. 	ditto	ditto	ditto

NUMBER	TITLE (tsWxGTUI Toolkit)	REVISION	DATE	SOURCE
9	Vol. 9 - SDIST Software User Manual <ul style="list-style-type: none"> This document tells a hands-on software user (developoer and operator) how to install and use the associated computer software ("tsWxGTUI_PyVx"). It may also cover a particular aspect of software operation, such as instructions for a particular position or task. It is developed for software that is run by the user (developer and operator) and has a user interface requiring on-line user input or interpretation of displayed output. If the software is embedded in a hardware-software system, user manuals or operating procedures for that system may make separate Software User Manuals unnecessary. 	ditto	ditto	ditto
10	Vol. 10 - SDIST TO-DO <ul style="list-style-type: none"> A To-Do-List provides a roadmap for development and troubleshooting work. 	ditto	ditto	ditto

7.2 Release Distribution Documents

NOTES:

- 1) The set of printed or printable non-proprietary pages must carry a "DRAFT" watermark, if intended for internal use and not approved for publication.
- 2) The set of printed or printable proprietary pages must carry a "CONFIDENTIAL" watermark, if intended for external use under a non-disclosure agreement whether or not approved for publication.
- 3) The published set of printed or printable documents must all carry the same revision number and date as approved for publication.

The "tsWxGTUI_PyVx" Toolkit is a computer software product. Its reproduction and use is governed by one or more copyright and license notices.

It is incumbent on each recipient of the "tsWxGTUI_PyVx" Toolkit to become familiar with the following documents:

NUMBER	TITLE (tsWxGTUI Toolkit)	REVISION	DATE	SOURCE
1	AUTHORS.txt <ul style="list-style-type: none"> List of the principal 	1.0.0	07/19/2014	Software Gadgetry

NUMBER	TITLE (tsWxGTUI Toolkit)	REVISION	DATE	SOURCE
	"tsWxGTUI_PyVx" Toolkit author(s) and authors credited for work covered by a prior copyright and license.			
2	BUGS.txt ▪ List of Known Problems / Issues.	ditto	ditto	Software Gadgetry
3	CHANGE_LOG.txt ▪ List of Additions, Modification and Deletions.	ditto	ditto	Software Gadgetry
4	CONFIGURE.txt ▪ Instructions for applying factory and site-specific configurations.	ditto	ditto	Software Gadgetry
5	COPYING.txt ▪ Instructions for copying all or a portion of the distribution.	ditto	ditto	Software Gadgetry
6	FAQ.txt ▪ Answers to Frequently Asked Questions.	ditto	ditto	Software Gadgetry
7	INSTALL.txt ▪ Describes steps to download, extract install and configure the "tsWxGUI" Toolkit.	ditto	ditto	Software Gadgetry
8	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.	ditto	ditto	Software Gadgetry
9	MANIFEST.txt ▪ Tally List for deliverable items.	ditto	ditto	Software Gadgetry
10	NEWS.txt ▪ Announcements of new releases.	ditto	ditto	Software Gadgetry
11	NOTICES.txt ▪ Details the copyright(s) and license(s).	ditto	ditto	Software Gadgetry
12	OPERATE.txt ▪ Describes steps to use the "tsWxGUI" Toolkit.	ditto	ditto	Software Gadgetry

NUMBER	TITLE (tsWxGTUI Toolkit)	REVISION	DATE	SOURCE
13	README.txt <ul style="list-style-type: none"> Introduces new recipients to the purpose, goals, non-goals, design and features of the computer software product. It may be appropriate to subdivide this file into separate files for the convenience of users working on platforms lacking facilities to view files in HTML and PDF format. See enclosed note on suggested README.txt file partitioning and naming convention. 	ditto	ditto	Software Gadgetry
14	THANKS.txt <ul style="list-style-type: none"> Acknowledgments to those otherwise unsung heros who contributed time and effort to supporting the authors as planners, editors, designers, coders and testers. 	ditto	ditto	Software Gadgetry
15	TO-DO.txt <ul style="list-style-type: none"> A To-Do-List provides a roadmap for development and troubleshooting work. 	ditto	ditto	Software Gadgetry
16	TROUBLESHOOT.txt <ul style="list-style-type: none"> 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 statistics. 	ditto	ditto	Software Gadgetry

NOTE:

Suggested README.txt file partitioning and naming convention.

README.txt File Table of Contents (README-<Number>-<Title>.txt:

Num. Title Description

-
- 1** Title Page - Title, logo and catch phrase of the "tsWxGTUI_PyVx" Toolkit.
 - 2** Table Of Contents - Topic document index for the "tsWxGTUI_PyVx" Toolkit User Guide.
 - 3** Copyright - Identifies Author(s) and Copyrighted Work(s).
 - 4** Credits - Acknowledgments to those whose Work has been used in accordance with each Work's original Author's Copyright and License.
 - 5** License - General and special arrangements, provisions, rules, specifications and standards that form an integral part of the agreement or contract between the "tsWxGTUI_PyVx" Toolkit Author(s) and its recipients.
 - 6** Purpose - Role in development of Application Programs and Embedded Systems.
 - 7** Goals - High-level view of functional, interface, design, implementation, operation, quality and reliability requirements that are within the work scope.
 - 8** Non-Goals - High-level view of functional, interface, design, implementation, operation, quality and reliability requirements that are beyond the work scope.
 - 9** Design Strategy - Describes the utilization of popular, field-proven, free and open source technology including Python, wxWidgets/wxPython and nCurses whose track record of performance in development, enhancement, maintenance and support is comparable to their Commercial-Off-The-Shelf (COTS) product counterparts.
 - 10** Design Architecture - Describes strategy for goal achievement beginning with a Block Diagram that depicts interfaces between components of the "tsWxGTUI_PyVx" Toolkit and its Operator(s).
 - 11** Software Configuration - Identifies the tool(s) chosen to register, store and retrieve evolutionary changes Management in various software components.
 - 12** "tsWxGTUI_PyVx" Toolkit 2.x Directories - Outlines the manner by which debugged Python source code components, and package defining "__init__.py" files, are initially organized and stored. It outlines the process for converting and storing a package copy for use with Python 3.x.
 - 13** "tsToolkitCLI" Directories - Identifies the contents of the "tsLibCLI", "tsToolsCLI" "tsTestsCLI", and "tsUtilities" directories.
 - 14** "tsToolkitGUI" Directories - Identifies the contents of the "tsLibGUI", "tsToolsGUI" and "tsTestsGUI" directories.
 - 15** Features - Describes functional, interface, design, implementation, operation, quality and reliability characteristics.
 - 16** Current Capabilities - Details the achievements of the functional, interface, design, implementation, operation, quality and reliability goals.

17 Current Limitations - Details the limitations of the functional, interface, design, implementation, operation, quality and reliability goals.

18 Reference Documents - Disclosure document index for the "tsWxGTUI_PyVx" Toolkit Distribution.

7.3 External Documents

NUMBER	TITLE	REVISION	DATE	SOURCE
1	Apple Mac OS X	<ul style="list-style-type: none"> ▪ 10.4 (Tiger) ▪ 10.5 (Leopard) ▪ 10.6 (Snow Leopard) ▪ 10.7 (Lion) ▪ 10.8 (Mountain Lion) ▪ 10.9 (Mavericks) 		www.apple.com
2	Cygwin	<ul style="list-style-type: none"> ▪ 1.7.11-1.7.28 		www.cygwin.com
3	Microsoft Windows	<ul style="list-style-type: none"> ▪ XP (SP3) ▪ 7 (SP1) ▪ 8 ▪ 8.1 		www.microsoft.com
4	nCurses	<ul style="list-style-type: none"> ▪ 5.9 		www.gnu.org/s/nCurses
5	Python	<ul style="list-style-type: none"> ▪ 2.6.4-2.7.9 ▪ 3.0.0-3.2.5 ▪ 3.3.0-3.4.1 		www.python.org
6	Ubuntu Linux	<ul style="list-style-type: none"> ▪ 11.10 ▪ 12.04 ▪ 14.04 		www.ubuntu.com
7	wxPython	<ul style="list-style-type: none"> ▪ 2.8.12 ▪ 2.9.3 		www.wxpython.org
8	wxWidgets	<ul style="list-style-type: none"> ▪ 2.8.12 ▪ 2.9.2.3-2.9.5.0 ▪ 3.0.0.0 		www.wxwidgets.org
9	"wxPython In Action" by Noel Rappin and Robin Dunn		March 2006	www.manning.com/rappin
10	"Programmer's Guide to NCurses" by Dan Gookin		Feb. 20, 2007	www.wiley.com

NUMBER	TITLE	REVISION	DATE	SOURCE
11	Fedora (Red Hat) Linux	<ul style="list-style-type: none"> 17-20 		http://fedoraproject.org
12	Scientific (Red Hat) Linux	<ul style="list-style-type: none"> 6.5 		http://www.scientificlinux.org
13	SuSE Linux	<ul style="list-style-type: none"> 12.2 13.1 		http://www.openSuSE.org
14	Solaris Unix (Oracle / Sun Microsystems)	<ul style="list-style-type: none"> 9 10 11 		http://oracle.com
15	OpenIndiana Unix (replacement for unreleased OpenSolaris 11)	<ul style="list-style-type: none"> 151a5-151a8 		http://openindiana.org
16	MIL-STD-498		Dec. 5, 1994	<p>Specification canceled on May 27, 1998 and replaced by the essentially identical demilitarized version EIA J-STD-016 as a process example guide for IEEE 12207.</p> <p>Free MIL-STD-498 PDF available from:</p> <p>http://www.everyspec.com/MIL-STD/MIL-STD-0300-0499/MIL-STD-498_25500/</p>
17	Git			http://git-scm.com/about
18	Mercurial	<ul style="list-style-type: none"> 2.9.1 		http://mercurial.selenic.com/
19	Parallels	<ul style="list-style-type: none"> 5-10 		http://www.parallels.com/
20	VMware Fusion	<ul style="list-style-type: none"> 4-5 		http://www.vmware.com/products/fusion/overview.html

Draft

8 NOTES

This section shall contain any general information that aids in understanding this document (e.g., background information, glossary, rationale).

8.1 Operator Interface Technology

Humans interact with computers via those hardware and software components associated with the User Interface:

- 1 **Shells** (on page 298) - Describes Microsoft DOS-style and POSIX-style user interfaces for accessing an operating system's kernel services.
- 2 **Desktop Environments** (on page 308) - Describes platform-specific features of the "tsWxGTUI_PyVx" Toolkit's user interfaces:
 - a) Development Systems
 - b) Embedded Systems
 - c) Its non-graphical Command Line Interface
 - d) Its Graphical User Interface
- 3 **Terminal Device Interface** (see "**Terminal Device Interface (TDI)**" on page 316)
- 4 **Appendix E - Inherited, Field-Proven Computer Technology** (on page 405) - Provides introductory and background information suitable for individuals with a broad range of computer programming skills who are or become interested in developing, customizing, documenting, enhancing, maintaining, supporting, troubleshooting and using the internal capabilities of the "tsWxGUT" Toolkit.

8.1.1 Shells

From Wikipedia, the free encyclopedia:

"A shell in computing provides a user interface for access to an operating system's kernel services. "Shell" is also used loosely to describe applications, including software that is "built around" a particular component, such as web browsers and email clients that are, in themselves, "shells" for HTML rendering engines. The term "shell" in computing, being the outer layer between the user and the operating system kernel, is synonymous with the general word "shell".

Generally, operating system shells use either a command-line interface (CLI) or graphical user interface (GUI). Mac OS xxx and Windows xxx are widely used operating systems with GUIs.[1][2][3]

The optimum choice of user interface depends on a computer's role and particular operation. CLIs allow some operations to be performed faster, rearranging large blocks of data for example. CLIs may be best for servers which are managed by experts: administrators, while GUIs offer simplicity and ease-of-use and would be more appropriate for image editing, CADD, and desktop publishing. In practice, many systems provide both user interfaces which can be called on a command-by-command basis. Windows xxx is the most obvious example with its "command prompt" and normal "windows" mode. It's no exaggeration to say that both Apple Macintosh OS xxx and Microsoft Windows xxx have revolutionised home computing by helping relatively inexperienced users to interface with a PC using a GUI.

In expert systems, a shell is a piece of software that is an "empty" expert system without the knowledge base for any particular application.[4]

History

The first Unix shell, Ken Thompson's sh,[5] was modeled after the Multics shell,[6] itself modeled after the RUNCOM[7] program Louis Pouzin showed to the Multics Team. The 'rc' suffix on some Unix configuration files (e.g. ".vimrc"), is a remnant of the RUNCOM ancestry of Unix shells.[citation needed]

Practically all modern operating system shells can be used in both interactive and batch mode, the latter usually by specifying the name of a text file with commands listed therein. Batch mode use of shells usually involves structures, conditionals, variables, and other elements of programming languages; some have the bare essentials needed for such a purpose, others are very sophisticated programming languages in and of themselves. Conversely, some programming languages can be used interactively from an operating system shell or in a purpose-built program.

Text (CLI) shells

A command-line interface (CLI) is an operating system shell that uses alphanumeric characters typed on a keyboard to provide instructions and data to the operating system, interactively. For example, a teletypewriter can send codes representing keystrokes to a command interpreter program running on the computer; the command interpreter parses the sequence of keystrokes and responds with an error message if it cannot recognize the sequence of characters, or it may carry out some other program action such as loading an application program, listing files, logging in a user and many others. Operating systems such as UNIX have a large variety of shell programs with different commands, syntax and capabilities. Some operating systems had only a single style of command interface; commodity operating systems such as MS-DOS came with a standard command interface but third-party interfaces were also often available, providing additional features or functions such as menuing or remote program execution.

Application programs may also implement a command-line interface. For example, in Unix-like systems, the telnet program has a number of commands for controlling a link to a remote computer system. Since the commands to the program are made of the same keystrokes as the data being sent to a remote computer, some means of distinguishing the two are required. An escape sequence can be defined, using either a special local keystroke that is never passed on but always interpreted by the local system. The program becomes modal, switching between interpreting commands from the keyboard or passing keystrokes on as data to be processed.

A feature of many command-line shells is the ability to save sequences of commands for re-use. A data file can contain sequences of commands which the CLI can be made to follow as if typed in by a user. Special features in the CLI may apply when it is carrying out these stored instructions. Such batch files (script files) can be used repeatedly to automate routine operations such as initializing a set of programs when a system is restarted. The command-line shell may offer features such as Command-line completion, where the interpreter expands commands based on a few characters input by the user. A command-line interpreter may offer a history function, so that the user can recall earlier commands issued to the system and repeat them, possibly with some editing.

Since all commands to the operating system had to be typed by the user, short command names and compact systems for representing program options were common. Short names were sometimes hard for a user to recall, and early systems lacked the storage resources to provide a detailed on-line user instruction guide.

Draft

Graphical shells

A graphical user interface (GUI) represents programs and data using visual symbols instead of text. Graphical user interfaces became feasible as the cost of interactive computer graphic hardware declined. Most graphical user interfaces develop the metaphor of an "electronic desktop", where data files are represented as if paper documents on a desk, and application programs similarly have graphical representations instead of being invoked by command names.

On Microsoft Windows

Modern versions of the Microsoft Windows operating system use the Windows shell as their shell. Windows Shell provides the familiar desktop environment, start menu, and task bar, as well as a graphical user interface for accessing 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, and which in fact ships with later versions of Windows of both the 95 and NT types at least through Windows XP. The interfaces of Windows versions 1 and 2 were markedly different.

Desktop applications, like iTVmediaPlayer, are also considered shells, as long as they use a third-party engine. Likewise, 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, SharpE and Emerge Desktop are good examples of the latter.

Interoperability programmes and purpose-designed software lets Windows users use equivalents of many of the various Unix-based GUIs discussed below, as well as Macintosh. An equivalent of the OS/2 Presentation Manager for version 3.0 can run some OS/2 programmes under some conditions using the OS/2 environmental subsystem in versions of Windows NT. For an example of the first, X Window-type environments can be run using combinations of Windows/Unix interoperability packages, communications suites such as Hummingbird Connectivity, and/or X server programmes for Windows such as WinAxe and others.

On Unix-like

Graphical (GUI) shells typically build on top of a windowing system and at least in the case of the X Window System or of Wayland, the shell consist of the X window manager respectively the wayland compositor and one or multiple programs giving the functionality to start programs that are available on the operating system, to manage open windows, to manage virtual desktops and often a widget engine."

```
GNOME
  Mutter
    GNOME Panel
    adesklets or gDesklets or Screenlets
    Avant Window Navigator
    Docky
    GNOME Shell
    Cinnamon
KDE SC
  Kwin
    KDesktop
    Kicker
    SuperKaramba
```

```
Kwin
    KDE Plasma
Xfce
    Xfwm, the Xfce window manager
    Xfce-panel gives the ability to start applications, to dock
minimized windows, to show a clock, etc.
LXDE
    Openbox
    LXPanel gives the ability to start applications, to dock minimized
windows, to show a clock, etc.
Enlightenment
    Enlightenment window manager
    Enlightenment DR17 desktop shell
Ubuntu
    Compiz
    Unity
CDE
Other: There are maybe about 105 different window managers available for
Unix-like operating systems. A dekstop shell can consist of any one of them,
plus the additional programs for starting applications and other utilities.
    Blackbox
    Fluxbox
    ratpoison
    xmonad
    dwm
```

On other platforms

```
Amiga environments:
    Ambient (for MorphOS)
    Directory Opus
    ScalOS
    Wanderer (for AROS)
    Workbench - GUI
    AmigaCLI, AmigaShell (AmigaOS) - command line

DOS Shell
Finder (for Mac OS X)
Doors CS, MirageOS, Ion, and CrunchyOS (for TI-83 and TI-84 series graphing
calculators)
OS/2 environments:
    Presentation Manager (for OS/2 1.1 and greater and eComStation)
    Workplace Shell (for OS/2 2.0 and greater and eComStation)
```

8.1.1.1 DOS-style Microsoft Windows Shell

Upon logging into a Desktop, Laptop or Workstation computer running Microsoft Windows, a shell (such as "Command Prompt" or "PowerShell") is the piece of software that provides an interface for users of the computer's operating system. The shell provides access to the services of the operating system. It accepts user input via the local keyboard. It presents operating system output via the local display.

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

d:\Home\rsg>help
For more information on a specific command, type HELP command-name
ASSOC          Displays or modifies file extension associations.
ATTRIB         Displays or changes file attributes.
BREAK          Sets or clears extended CTRL+C checking.
BCDEDIT        Sets properties in boot database to control boot loading.
CACLS          Displays or modifies access control lists (ACLs) of files.
CALL           Calls one batch program from another.
CD             Displays the name of or changes the current directory.
CHCP           Displays or sets the active code page number.
CHDIR          Displays the name of or changes the current directory.
CHKDSK         Checks a disk and displays a status report.
CHKNTFS        Displays or modifies the checking of disk at boot time.
CLS            Clears the screen.
CMD            Starts a new instance of the Windows command interpreter.
COLOR          Sets the default console foreground and background colors.
COMP           Compares the contents of two files or sets of files.
COMPACT        Displays or alters the compression of files on NTFS partitions.
CONVERT        Converts FAT volumes to NTFS. You cannot convert the
               current drive.
COPY           Copies one or more files to another location.
DATE           Displays or sets the date.
DEL            Deletes one or more files.
DIR            Displays a list of files and subdirectories in a directory.
DISKCOMP       Compares the contents of two floppy disks.
DISKCOPY       Copies the contents of one floppy disk to another.
DISKPART       Displays or configures Disk Partition properties.
DOSKEY         Edits command lines, recalls Windows commands, and
               creates macros.
DRIVERQUERY    Displays current device driver status and properties.
ECHO           Displays messages, or turns command echoing on or off.
ENDLOCAL       Ends localization of environment changes in a batch file.
ERASE          Deletes one or more files.
EXIT           Quits the CMD.EXE program (command interpreter).
FC             Compares two files or sets of files, and displays the
               differences between them.
FIND           Searches for a text string in a file or files.
FINDSTR        Searches for strings in files.
FOR            Runs a specified command for each file in a set of files.
FORMAT         Formats a disk for use with Windows.
FSUTIL         Displays or configures the file system properties.
FTYPE          Displays or modifies file types used in file extension
               associations.
GOTO           Directs the Windows command interpreter to a labeled line in
```

	a batch program.
GPRESULT	Displays Group Policy information for machine or user.
GRAFTABL	Enables Windows to display an extended character set in graphics mode.
HELP	Provides Help information for Windows commands.
ICACLS	Display, modify, backup, or restore ACLs for files and directories.
IF	Performs conditional processing in batch programs.
LABEL	Creates, changes, or deletes the volume label of a disk.
MD	Creates a directory.
MKDIR	Creates a directory.
MKLINK	Creates Symbolic Links and Hard Links
MODE	Configures a system device.
MORE	Displays output one screen at a time.
MOVE	Moves one or more files from one directory to another directory.
OPENFILES	Displays files opened by remote users for a file share.
PATH	Displays or sets a search path for executable files.
PAUSE	Suspends processing of a batch file and displays a message.
POPD	Restores the previous value of the current directory saved by PUSHHD.
PRINT	Prints a text file.
PROMPT	Changes the Windows command prompt.
PUSHHD	Saves the current directory then changes it.
RD	Removes a directory.
RECOVER	Recovers readable information from a bad or defective disk.
REM	Records comments (remarks) in batch files or CONFIG.SYS.
REN	Renames a file or files.
RENAME	Renames a file or files.
REPLACE	Replaces files.
RMDIR	Removes a directory.
ROBOCOPY	Advanced utility to copy files and directory trees
SET	Displays, sets, or removes Windows environment variables.
SETLOCAL	Begins localization of environment changes in a batch file.
SC	Displays or configures services (background processes).
SCHTASKS	Schedules commands and programs to run on a computer.
SHIFT	Shifts the position of replaceable parameters in batch files.
SHUTDOWN	Allows proper local or remote shutdown of machine.
SORT	Sorts input.
START	Starts a separate window to run a specified program or command.
SUBST	Associates a path with a drive letter.
SYSTEMINFO	Displays machine specific properties and configuration.
TASKLIST	Displays all currently running tasks including services.
TASKKILL	Kill or stop a running process or application.
TIME	Displays or sets the system time.
TITLE	Sets the window title for a CMD.EXE session.
TREE	Graphically displays the directory structure of a drive or path.
TYPE	Displays the contents of a text file.
VER	Displays the Windows version.
VERIFY	Tells Windows whether to verify that your files are written correctly to a disk.
VOL	Displays a disk volume label and serial number.
XCOPY	Copies files and directory trees.
WMIC	Displays WMI information inside interactive command shell.

For more information on tools see the command-line reference in the online help.

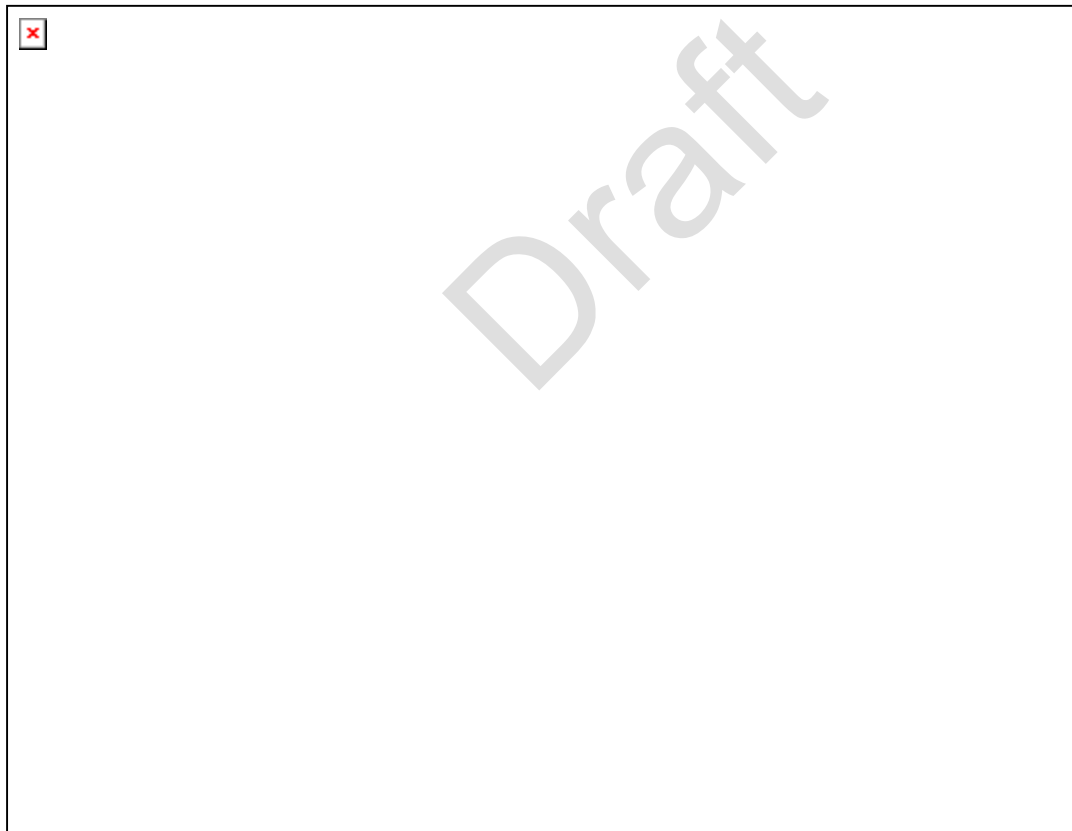
Draft

8.1.1.2 POSIX-style Bash Shell

Upon logging into a POSIX-compliant computer, a shell (such as "Bash") is the piece of software that provides an interface for users of the computer's operating system. The shell provides access to the services of the operating system. It accepts user input via the local keyboard. It presents operating system output via the local display.

The user can, when appropriate, issue a command to the local operating system requesting that it "login" the user into a remote computer. Once connected to the remote computer, the local operating system forwards the user's keyboard input to the remote operating system. It forwards the remote operating system's output to the user's local display. The process is efficient when forwarding character-mode data. There is more overhead in forwarding pixel-mode data due to the substantially larger volume of picture elements relative to text elements (for example a text character represented by an 8x12 matrix of pixels plus special effect attribute data such as color, brightness, blinking etc.) .

The following figure illustrates a Bash shell on a Windows XP system running Cygwin with its "ls" command (the POSIX equivalent of the DOS "dir /w" command).



1 Shell displays its input prompt:

- Beginning with login identity for user "rsg" on a computer named "eclecticxpm"
- Identifying current working directory path "/cygdrive/d/WR/SoftwareGadgetry-2.x"
- Ending with "\$"

- 2 Shell displays user command requesting a list of files names:
 - "ls"
- 3 Shell displays system response:
 - list of file names

Draft

8.1.2 Desktop Environments

From Wikipedia, the free encyclopedia:

"In computing, a desktop environment (DE) is an implementation of a desktop metaphor graphical user interface (GUI). The desktop environment was seen mostly on personal computers until the rise of mobile computing.[1][2] Desktop GUIs help the user to easily access and edit files, while they usually do not provide access to all of the features found in the underlying operating system. Instead, the traditional command-line interface (CLI) is still used when full control over the operating system is required.

A desktop environment typically consists of icons, windows, toolbars, folders, wallpapers and desktop widgets (see Elements of graphical user interfaces and WIMP).[3]

A GUI might also provide drag and drop functionality and other features that make the desktop metaphor more complete. A desktop environment aims to be an intuitive way for the user to interact with the computer using concepts which are similar to those used when interacting with the physical world, such as buttons and windows.

While the term desktop environment originally described a style of user interfaces following the desktop metaphor, it has also come to describe the programs that realize the metaphor itself.[4] This usage has been popularized by the Common Desktop Environment and the K Desktop Environment."

1 *Development System* (on page 309)

- a) *Microsoft Windows Desktop Environment Example* (on page 311) - Describes how a general-purpose computer used for software development can integrate both non-graphical and graphical user interfaces.

2 *Embedded System* (on page 312)

- a) *Command Line Interface Desktop* (on page 312) - Describes the purpose and appearance of the operator's non-graphical display.
- b) *Graphical User Interface Desktop* (on page 314) - Describes the purpose and appearance of the operator's graphical display.

8.1.2.1 Development System

This section describes how a general-purpose computer used for software development can integrate both non-graphical and graphical user interfaces.

- **Microsoft Windows Desktop Environment Example** (on page 311) - Depicts and scribes the capabilities of high resolution graphical user interface technology.
- **Appendix A - Baseline Host Computer Platforms** (on page 333)) - Describes the general-purpose computer system platforms used by the "tsWxGTUI_PyVx" Toolkit developers.

Make & Model	Hardware	Software	Desktop
Dell 15.6" Inspiron 7000 Laptop	<p>1999-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 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, 3.0, 3.1 and 3.2 ▪ Linux (Ubuntu 12.04) with Python 2.7, 3.0, 3.1 and 3.2 	<p>The host operating system presents its users with a graphical desktop environment which can concurrently execute multiple processes and their associated multiple threads (tasks).</p> <p>Using the full screen at its maximum SVGA resolution, the desktop can occupy:</p> <ul style="list-style-type: none"> ▪ 128 columns x 64 rows. ▪ 1024 x 768 pixels. <p>Using the full screen at its VGA resolution, the desktop can occupy:</p> <ul style="list-style-type: none"> ▪ 80 columns x 40 rows. ▪ 640 x 480 pixels.
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 internal 	<p>Host Operating System</p> <ul style="list-style-type: none"> ▪ Apple's 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 7-9 ▪ VMware Fusion 4-5 	<p>The host and each guest operating system presents its users with a graphical desktop environment which can concurrently execute multiple processes and their associated multiple threads (tasks).</p>

	<p>hard drive was used to run Apple's Mac OS X 10.7 and the hypervisor virtualization applications (Parallels Desktop 7-9 and VMware Fusion 4-5) that supported various guest operating systems.</p> <ul style="list-style-type: none"> Its 1.5 TB external hard drive was used to store and run configured versions of the Ubuntu Linux (12.04), Microsoft Windows (8.1 Professional, 8 Professional, 7 Professional with SP1 and XP Professional with SP3) and Unix (OpenSolaris 11/OpenIndiana 151) guest operating systems. 	<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 12.04) with Python 2.7 and 3.3 Windows (8.1 Professional, 8 Professional, 7 Professional with SP1, XP Professional with SP3) with POSIX-like Cygwin plug-in and Python 2.6, 2.7, 3.2 and 3.3 UNIX (OpenIndiana 151a5, a replacement for unreleased OpenSolaris 11/SunOS 5.11) with Python 2.6.4 running on Apple MacBook Pro 	<p>Using the full screen at its maximum resolution, the desktop can occupy:</p> <ul style="list-style-type: none"> 210 columns x 105 rows. 1680 x 1050 pixels.
Apple 27" iMac Desktop	<p>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.</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 9-10 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 Ubuntu Linux 	<p>Host Operating System</p> <ul style="list-style-type: none"> Apple's 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 9-10 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 (Ubuntu 12.04) with Python 2.7 and 3.3 Windows (8.1 Professional, 8 Professional, 7 Professional with SP1 and XP Professional with SP3) with POSIX-like Cygwin plug-in and Python 2.6, 2.7, 3.2 and 3.3 	<p>The host and each guest operating system presents its users with a graphical desktop environment which can concurrently execute multiple processes and their associated multiple threads (tasks).</p> <p>Using the full screen at its maximum resolution, the desktop can occupy:</p> <ul style="list-style-type: none"> 320 columns x 144 rows. 2560 x 1440 pixels.

	(12.04), Microsoft Windows (8.1 Professional, 8 Professional, 7 Professional with SP1 and XP Professional with SP3) and Unix (OpenSolaris 11/OpenIndiana 151) guest operating systems.	<ul style="list-style-type: none"> ▪ UNIX (OpenIndiana 151a5, a replacement for unreleased OpenSolaris 11/SunOS 5.11) with Python 2.6.4 running on Apple MacBook Pro 	
--	--	---	--

8.1.2.1.1 Microsoft Windows Desktop Environment Example

The Windows 7 desktop environment example:

1 GUI Icons:

Displays multiple Icons for starting an associated application when a mouse button is clicked while the mouse pointer is over the Icon.

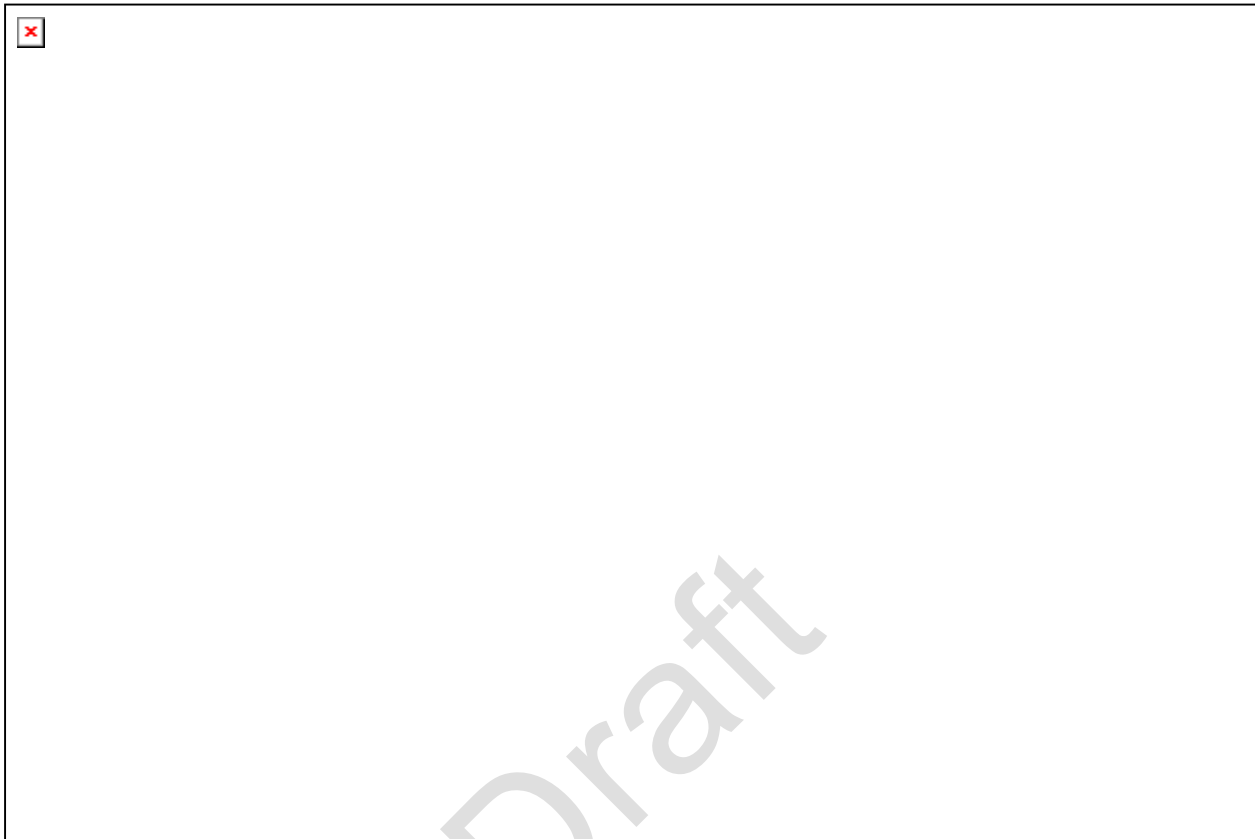
2 GUI Window Frames:

Displays GUI Window Frames for the four running applications

- Top Left - Windows/DOS (command line interface) shell displaying help for the "dir" command.
- Bottom Left - Cygwin/bash (command line interface) shell displaying help (man page) for the "ls" command.
- Top Right - Windows third-party (graphical user interface) "Total Commander" file manager displaying the directories and files in two different hard drive areas.
- Bottom Right - Windows third-party (graphical user interface) "Firefox" web browser displaying the front page for CNN news.

Position and size can be changed by depressing the mouse button when the mouse pointer is over an edge or corner, dragging the mouse to a new position and then releasing the mouse button. This method was used to achieve the positions and sizes of the four GUI Window Frames.

- 3 GUI Task Bar - Icons for the running processes ("Firefox", "Total Commander", "DOS Shell", "Bash Shell" can be clicked on to change the operation, appearance or focus of its associated process.



8.1.2.2 Embedded System

This section describes how an embedded computer customized and used for monitoring and controlling equipment can integrate both non-graphical and graphical user interfaces.

- **Command Line Interface Desktop** (on page 312)
- **Graphical User Interface Desktop** (on page 314)

8.1.2.2.1 Command Line Interface Desktop

Early personal computers, like the 1981 model year IBM PC had the Microsoft Disk Operating System (DOS) and the choice of two displays:

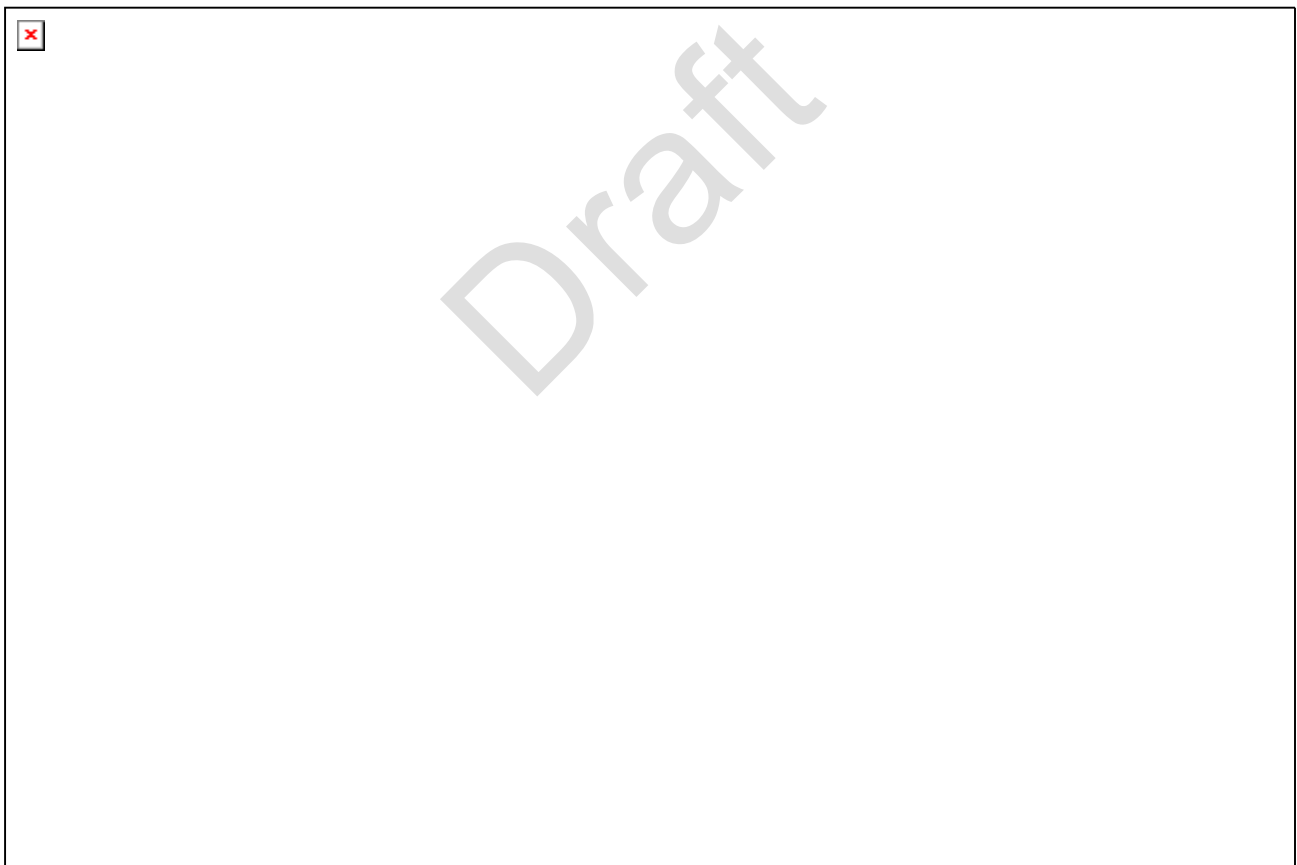
- 1 A low resolution monochrome display adapter for 720×350 pixel text. On the monochrome display DOS provided an area for text that had a width of 80 columns and a height of 25 rows (at 9 x 14 pixels per character).
- 2 A lower resolution color/graphics display adapter. The highest display resolution of any mode was 640×200, and the highest color depth supported was 4-bit (16 colors). On the color display DOS provided an area for text that had a width of 80 columns and a height of 25 (at 8 x 8 pixels per character).

Microsoft's current Windows versions still retain the 80 column x 25 row as the default initial size for the so-called DOS-box command line interface.

For computers with high resolution displays, the user can start one or more shells (A, B, C etc.). The computer screen area occupied by each shell window serves as the desktop for the associated Command Line Interface (A, B, C etc.).

For example, the following is a screen shot of a Windows 7 desktop (16" diagonal with 1440 x 900 pixel resolution) containing desktop icons for various applications, task bar icons for "AuthorIt", "Chrome", "FireFox", "Total Commander" (the 3.5" floppy disk) plus icons for the two Cygwin bash-type shells:

- 1 The smaller shell on the left is displaying output (on-line help) from the Python 2.7 Command Line Interface "test_tsOperatorSettingsParser.py" application.
- 2 The larger shell on the right is displaying output (the calculator-style keypad and various debug messages) from a Python 2.7 Graphical-style User Interface "test_tsWxGridSizer.py" application.
- 3 Either or both shells could have similarly been concurrently running Python 3.2 instead of Python 2.7 on the local computer or on a remote Linux, Mac OS X, Solaris or Windows XP computer.



8.1.2.2.2 Graphical User Interface Desktop

For computers with high resolution displays, the user can start one or more Command Line Interface shells (A, B, C etc.). The computer screen area occupied by each shell window serves as the desktop for the associated Command Line Interface (A, B, C etc.). Because each Graphical-style User Interface is built upon an associated Command Line Interface, there is an associated Graphical-style User Interface desktop which isolates the application-specific GUI objects of one Command Line Interface shell from those of other Command Line Interface shells. Each Command Line Interface shell can run concurrently with other Command Line Interface shells. By starting other Command Line Interface shells, it is possible to concurrently run:

- 1 Another instance of the same application or tool processing different input data.
- 2 An instance of another applications or tool.

Whereas all "wxWidgets" / "wxPython" GUI applications and tools share the native Graphical User Interface shell of the host computer, the "tsWxGTUI_PyVx" Toolkit isolates frames, dialogs and the Redirected Output frame within each Graphical User Interface Desktop.

8.1.2.2.2.1 Application Desktop Layout

The "tsWxGTUI_PyVx" Toolkit's "tsLibGUI" library includes the "tsWxPkg" which includes the "tsWxDisplay" class. The class manages the layout of its assigned Command Line Interface shell screen (W-Columns x H-Rows) so as to provide:

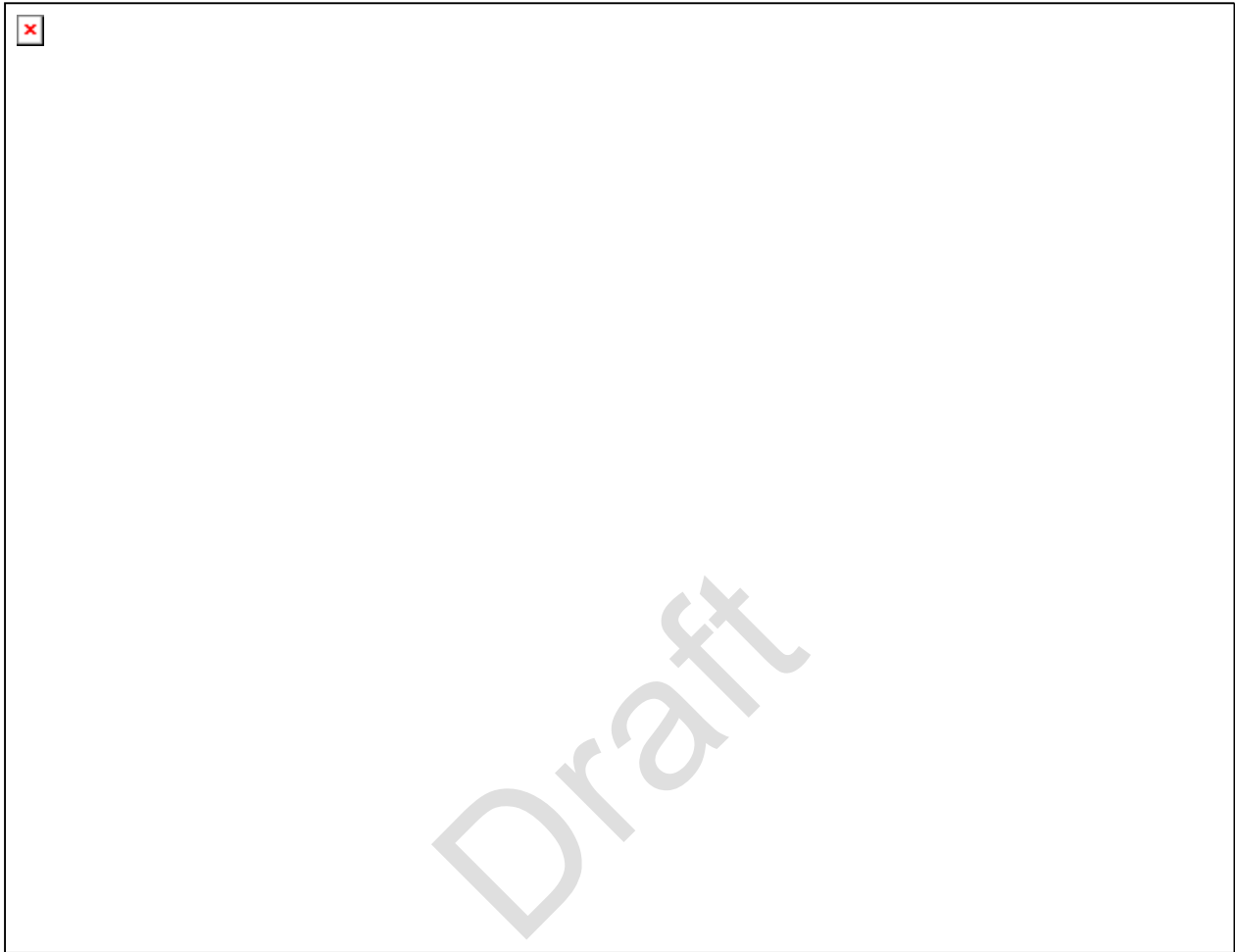
- 1 A client area for the application's Frame(s) and Dialog(s) windows.
- 2 An optional area for the application's auto-scrolling Redirected Output window.
- 3 An area for the application name, activity indicators (spinning baton, date and time) and buttons that shift focus from one Frame or Dialog to another, thereby moving partially concealed background objects to the fully visible foreground.

	X=0	1	2	3	X=W-1	
	0	1	2	3	4	
Y=0	+	-----	+			Frame & Dialog Border Top
1		Client Area for an Application's				
2		Frame(s) & Dialog(s)				
3		(ex. 35 cols x 6 rows)				
4						
5	+	-----	+			Frame & Dialog Border Bottom
H-9	+	-----	+			Stdio Frame Border Top
H-8		Stdio Output (Optional)				
H-7		(ex. 35 cols x 5 rows)				
H-6						
H-5	+	-----	+			Stdio Frame Border Bottom
H-4	+	-----	+			Task Frame Border Top
H-3		Task Bar Output				
H-2		(ex. 35 cols x 4 rows)				
Y=H-1	+	-----	+			Task Frame Border Bottom

The following screenshot illustrates the layout features:

- 1** The Command Line Interface was a "bash" shell created by "Cygwin" on a computer running Microsoft Windows XP.
- 2** The Graphical-style User Interface desktop was created by the "tsWxGTUI_PyVx" Toolkit.
- 3** The Application Program was named "test_tsWxWidgets" and was created for Python 2.7.
- 4** At the request of the Application Program, the "tsWxGTUI_PyVx" Toolkit (via its "tsWxFrame", "tsWxDialog" and "tsWxTxtCtrl" modules) created:
 - a) A client area frame named "widgets_communicate".
 - b) A client area dialog named "Dialog Non-Modal".
 - c) Logged a message with the INFO priority/severity level.
 - d) Logged a message with the NOTICE priority/severity level.
- 5** On behalf of the Application Program, the "tsWxGTUI_PyVx" Toolkit (via its "tsWxPyOnDemandOutputWindow" module) created:
 - a) A desktop frame named "Redirected Output: stdout/stderr".
 - b) A desktop frame named "Tasks" with:
 - Application name
 - Focus control buttons for two frames and one dialog
 - Periodically updated activity indicators

6 The screenshot was created on November 22, 2011 at 03:37:03 A.M.



8.1.3 Terminal Device Interface (TDI)

The terminal consists of the following devices:

- **Display** (on page 317) - Output is presented to the operator via a terminal's display. A VGA terminal has a display resolution of 640 x 480 pixels. With a font of 8 x 12 pixels, it can show 80 columns x 40 rows.
- **Keyboard** (on page 331) - Input is entered by the operator via the terminal's keyboard. A typical keyboard consists of a mechanical typewriter style matrix of push buttons. The operator may press a letter, digit, punctuation symbol or function key.
- **Mouse** (on page 331) - Point and click input is entered by the operator via the terminal's mouse, touchscreen, trackpad or trackball. The input consists of display column and row coordinates and the clicked, depressed or released state of any associated left, center and right buttons.

8.1.3.1 Display

Output is presented to the user via a terminal display device. The display consists of a matrix of column and row elements. The dimensions of VGA-type displays, for example, are 640 by 480 pixels. For the standard 8 pixel by 12 pixel courier font this is equivalent to 80 columns by 40 rows.

ANSI, VT100 and VT220 Terminal

The display of ANSI , VT100 and VT220 Terminals (*1-Color VT100 via Ubuntu Linux Terminal* (on page 319)) have a single color phosphor that could be "white", "green" or "orange". Text could be displayed in normal mode with the foreground phosphor active ("white" color ON) against the "black" background phosphor inactive ("white" color OFF is equivalent to "black" color ON). In reverse-mode, the activity of the foreground and background were reversed.

ANSI, VT100 and VT220 Terminal Emulators

The display of ANSI , VT100 and VT220 Terminal Emulators (*1-Color VT100 and VT220 with 2nd-Color Highlight via Mac OS X iTerm* (see "*1-Color VT100 with 2nd-Color Highlight via Mac OS X iTerm*" on page 323)) have tri-color phosphors ("red", "green", "blue") that could be activated in various combinations and proportions to create at least an 8-color palette ("black", "blue", "cyan", "green", "magenta", "red", "white", "yellow"). Text could be displayed in normal mode with the foreground palette active ("white" color ON) against the background palette active ("white" color OFF is equivalent to "black" color ON). In reverse-mode, the activity of the foreground and background were reversed. Highlighted text could be in a third color palette ("cyan" or "red" color ON).

XTERM Terminal Emulator

The display of an XTERM Terminal (*8-Color XTERM via Cygwin-X Console* (on page 329)) have tri-color phosphors ("red", "green", "blue") that could be activated in various combinations and proportions to create at least an 8-color palette ("black", "blue", "cyan", "green", "magenta", "red", "white", "yellow"). Text could be displayed in normal mode with the foreground palette active ("white" color ON) against the background palette active ("red" color ON). In reverse-mode, the activity of the foreground and background were reversed.

Derivatives of the xterm terminal emulator include the following:

mintty - A variant of xterm, included in Cygwin, the free GNU/Linux add-on to Microsoft Windows from Red Hat, that can emulate VT100, VT220 and each of the following:

xterm-color - An older branch of xterm that supports sixteen colors, the standard 8 colors in both normal and dim intensity representing 256 color pairs.

xterm-16color - An extension of xterm that supports sixteen colors, the standard 8 colors in both normal and dim intensity representing 256 color pairs.

xterm-88color - An extension of xterm that supports 88 colors representing 7744 color pairs.

xterm-256color - An extension of xterm that supports 181 of 256 colors encompassing only 32761 color pairs.

8.1.3.2 Usage Issues

1 Hardware

- Digital Equipment Co. vt100 terminal or compatible - keyboard, no mouse and 1-color phosphor display
- Digital Equipment Co. vt220 terminal or compatible - keyboard, no mouse and 1-color phosphor display

2 Software (Hardware Emulator)

NOTE: Reproducibility of the following requires the following test procedure: 1) open a new shell; 2) echo \$TERM; 3) TERM=vt100; 4) echo \$TERM; 5) run the python application. Subsequently changing TERM between python runs has been found to leave the emulator in a mixed mouse state in which the mouse could be erroneously enabled but unable to properly interpret its input data.

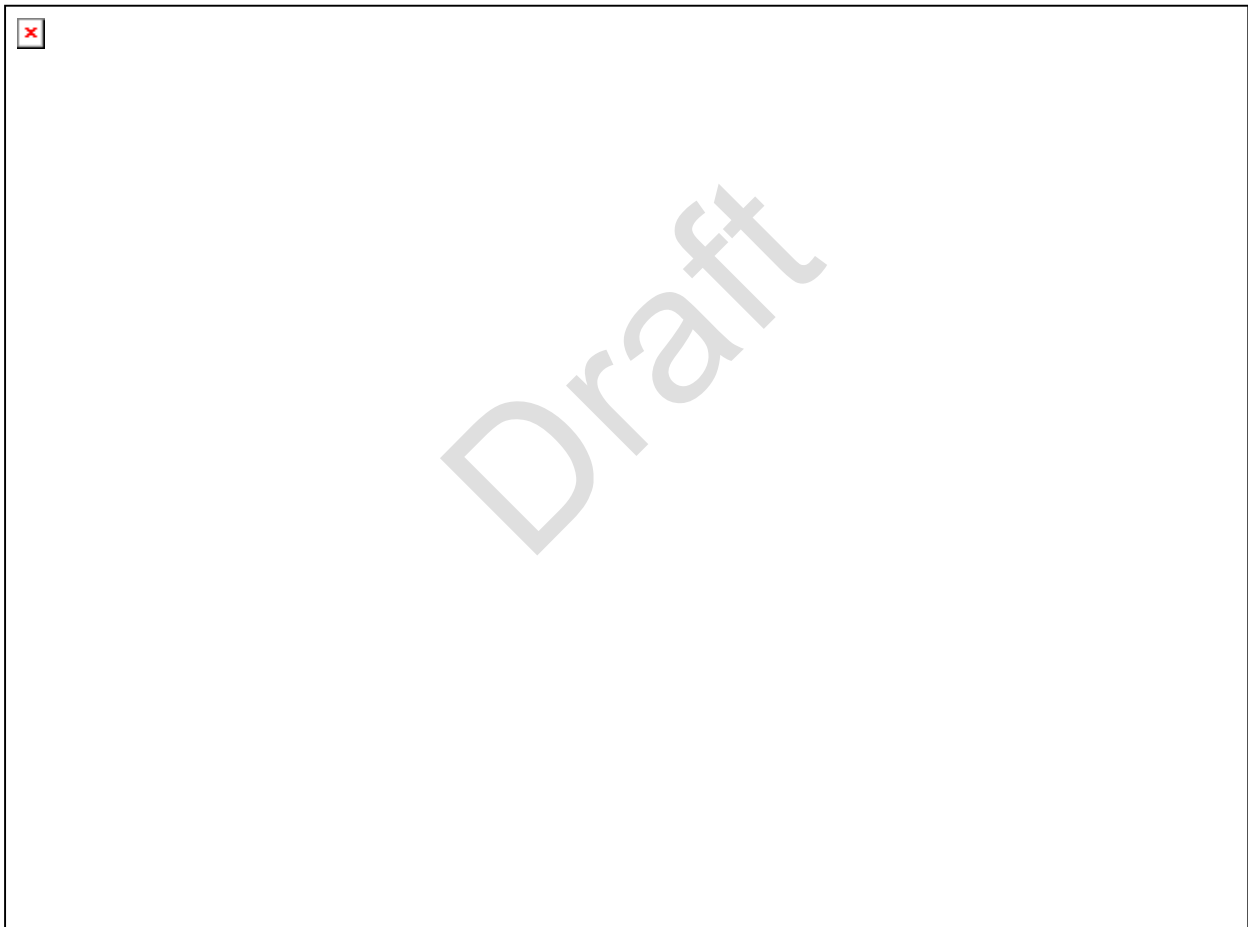
- ----- Platforms with Ubuntu Linux -----
- **Linux TERMINAL (TERM=xterm)** - keyboard, mouse/wheel mouse/trackball/trackpad with left/center/right buttons, and 8-color phosphor display
- ----- Platforms with Mac OS X -----
- **MacOS iTerm (TERM=vt100)** - keyboard, no mouse/wheel mouse/trackball/trackpad with left/center/right buttons, and 1-color phosphor display
- **MacOS iTerm (TERM=xterm)** - keyboard, mouse/wheel mouse/trackball/trackpad with left/center/right buttons, and 8-color phosphor display
- **MacOS Terminal (TERM=cygwin)** - keyboard, no mouse/wheel mouse/trackball/trackpad with left/center/right buttons, and 8-color phosphor display
- **MacOS Terminal (TERM=vt100)** - keyboard, no mouse/wheel mouse/trackball/trackpad with left/center/right buttons, and 1-color phosphor display
- **MacOS Terminal (TERM=xterm)** - keyboard, no mouse/wheel mouse/trackball/trackpad with left/center/right buttons, and 8-color phosphor display
- ----- Platforms with OpenIndiana Unix -----
- **Solaris (SunOS) Terminal (TERM=xterm)** - keyboard, mouse/wheel mouse/trackball/trackpad with left/center/right buttons, and 8-color phosphor display
- ----- Platforms with Microsoft Windows and Cygwin add-on -----
- **Cygwin console (TERM=vt100)** - keyboard, no mouse and 1-color phosphor display
- **Cygwin console (TERM=cygwin)** - keyboard, no mouse and 8-color phosphor display
- **Cygwin console (TERM=xterm)** - keyboard, mouse/wheel mouse/trackball/trackpad with left/center/right buttons, and 8-color phosphor display
- **Cygwin MinTTY (TERM=cygwin)** - keyboard, no mouse/wheel mouse/trackball/trackpad with left/center/right buttons, and 1-color phosphor display
- **Cygwin MinTTY (TERM=vt100)** - keyboard, no mouse/wheel mouse/trackball/trackpad with left/center/right buttons, and 1-color phosphor display

- **Cygwin MinTTY (TERM=xterm)** - keyboard, mouse/wheel mouse/trackball/trackpad with left/center/right buttons, and 8-color phosphor display

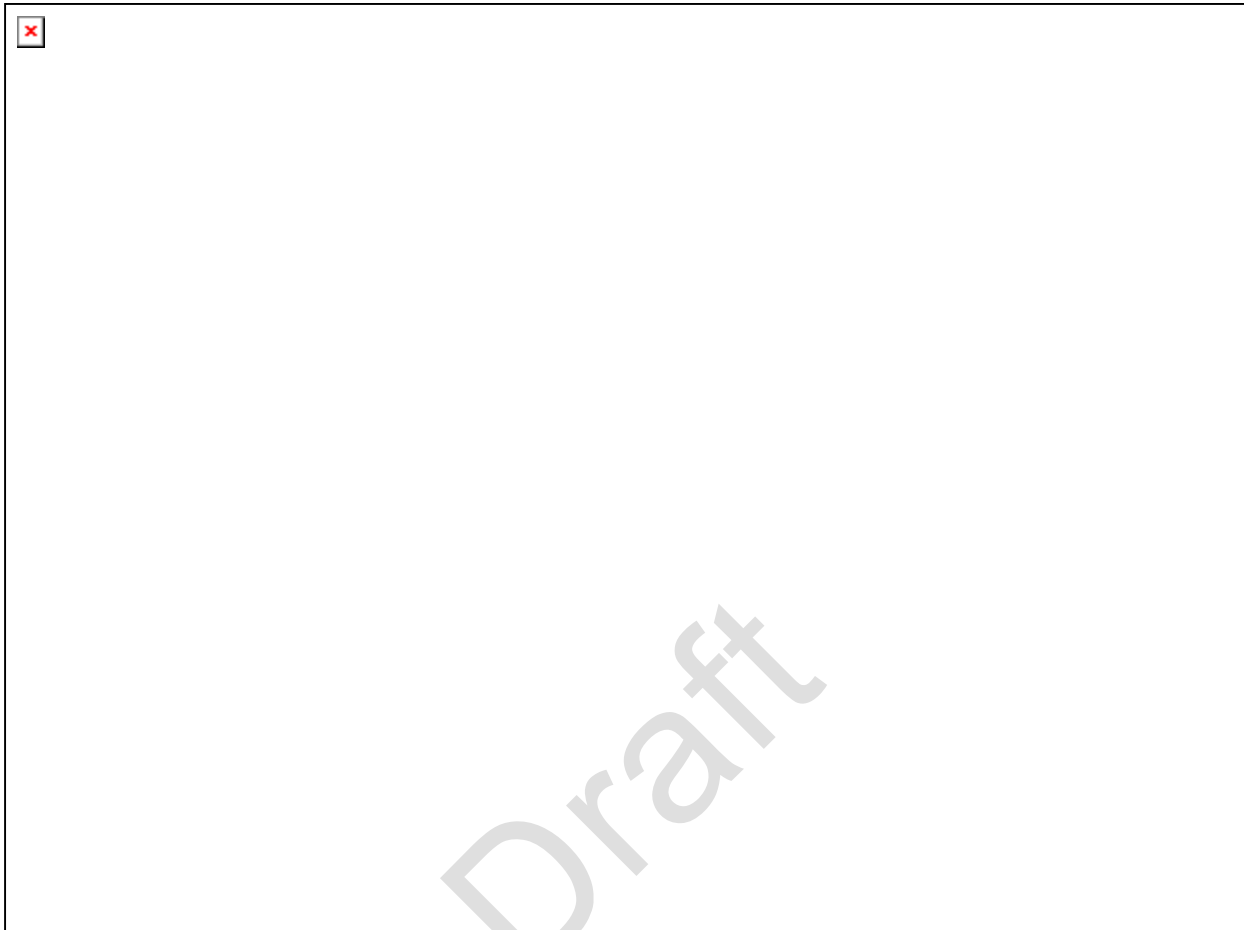
8.1.3.2.1 Linux (Ubuntu) Platform

- *1-Color VT100 via Ubuntu Linux Terminal* (on page 319)
- *8-Color XTERM via Ubuntu Linux Terminal* (on page 320)

8.1.3.2.1.1 1-Color VT100 via Ubuntu Linux Terminal



8.1.3.2.1.2 8-Color XTERM via Ubuntu Linux Terminal

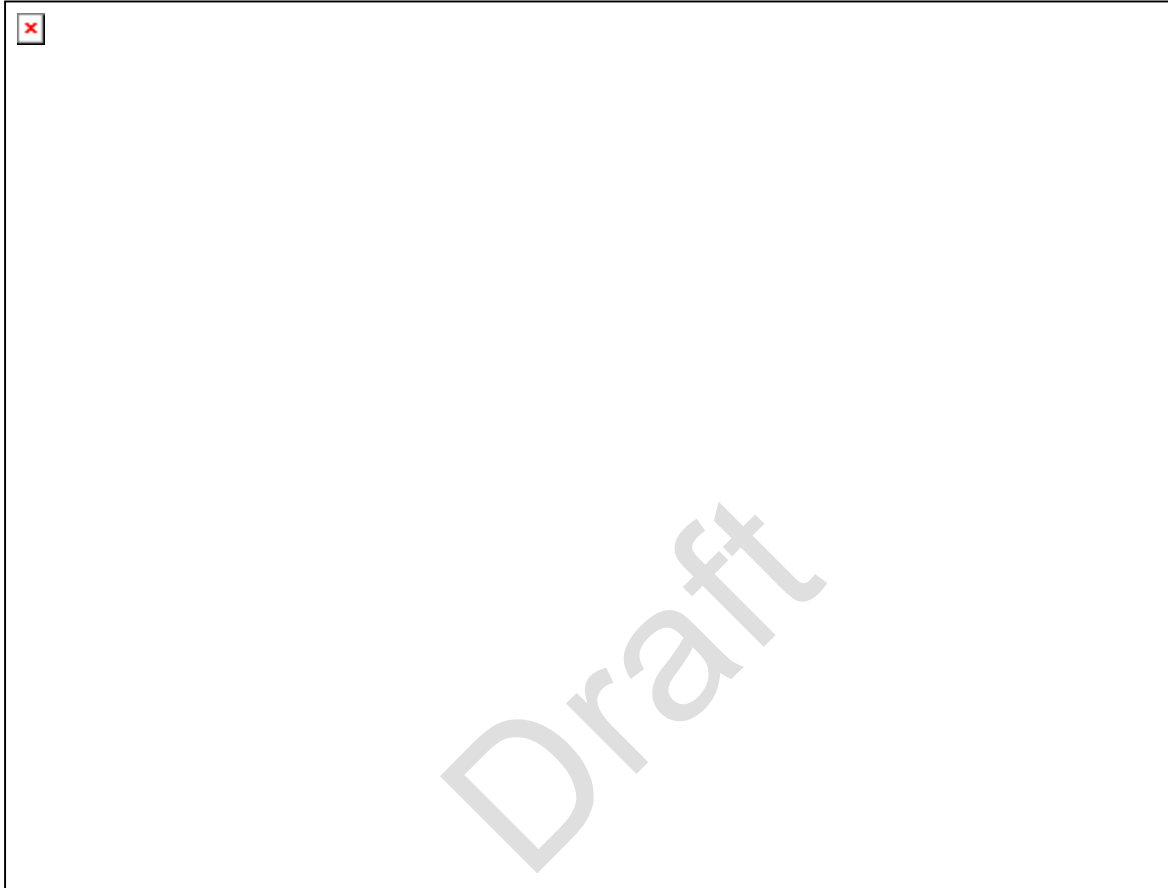


8.1.3.2.2 Mac OS X Platform

- *8-Color ANSI via Mac OS X Terminal* (on page 321)
- *1-Color VT100 via Mac OS X Terminal* (on page 322)
- *1-Color VT100 with 2nd-Color Highlight via Mac OS X iTerm* (on page 323)
- *8-Color XTERM via Mac OS X iTerm* (on page 324)
- *8-Color XTERM via Mac OS X Terminal* (on page 325)

8.1.3.2.2.1 8-Color ANSI via Mac OS X Terminal

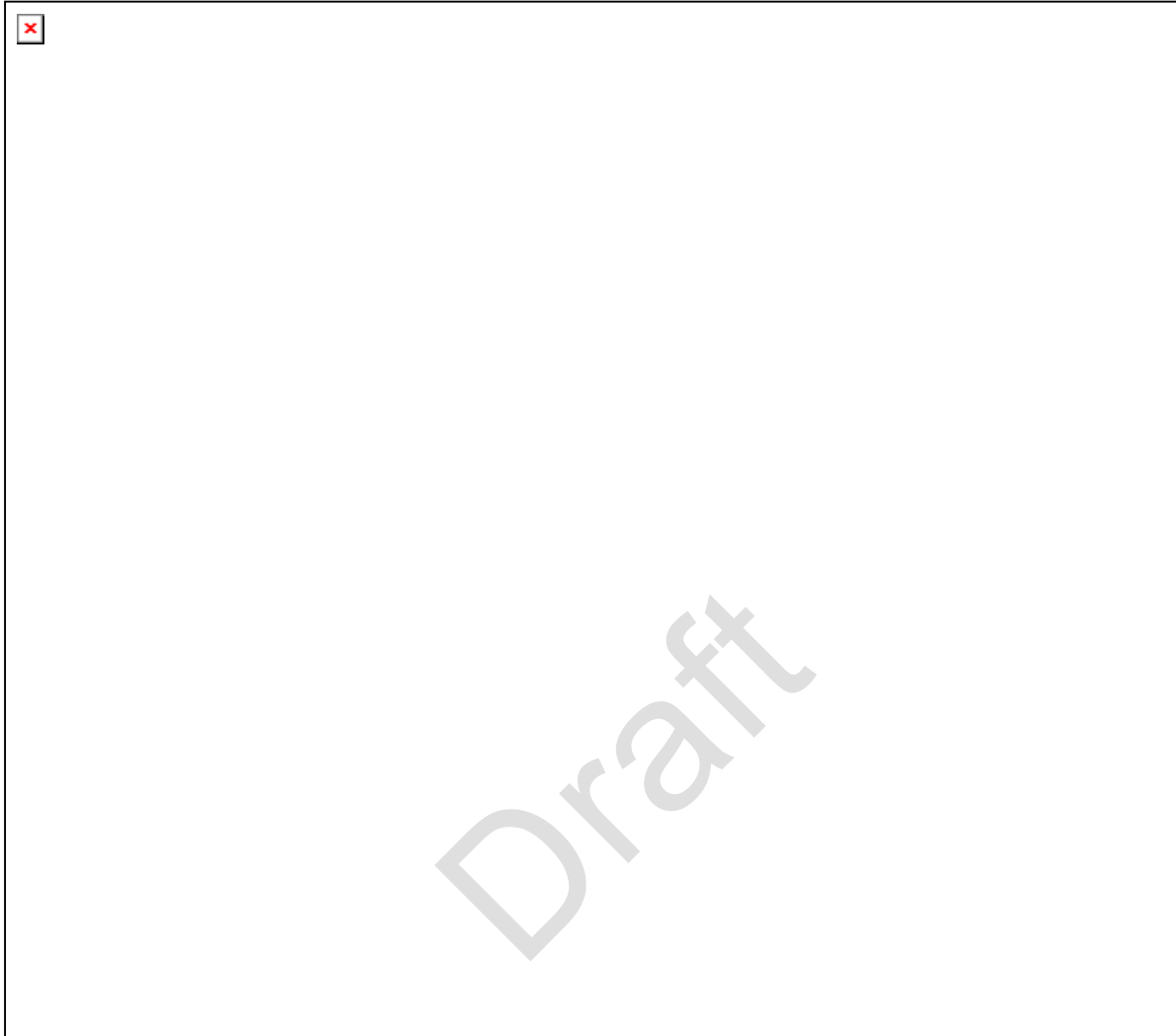
The Mac OS X Terminal does not provide a usable ANSI Terminal Emulation. It fails to properly render border generating characters.



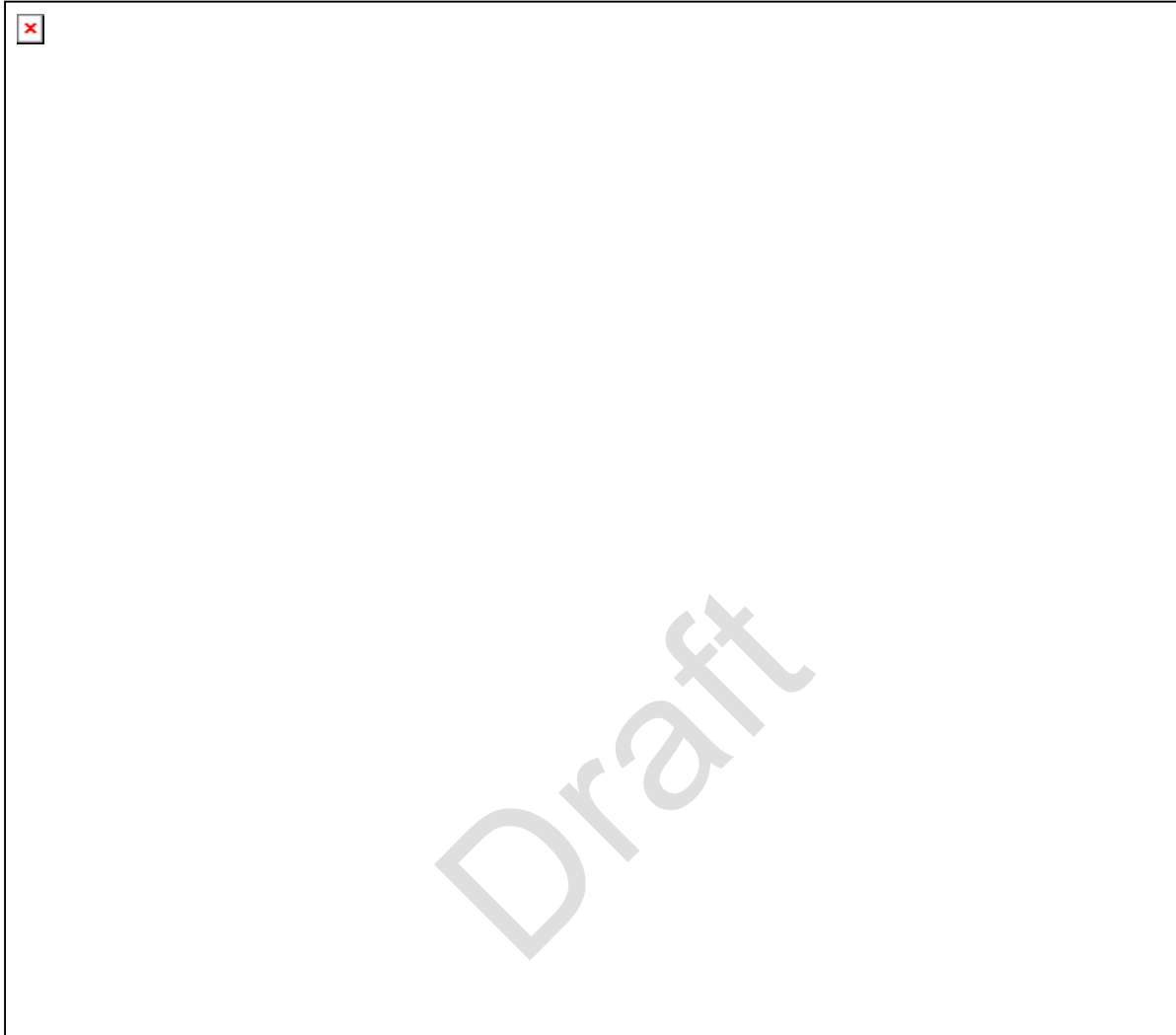
8.1.3.2.2.2 1-Color VT100 via Mac OS X Terminal



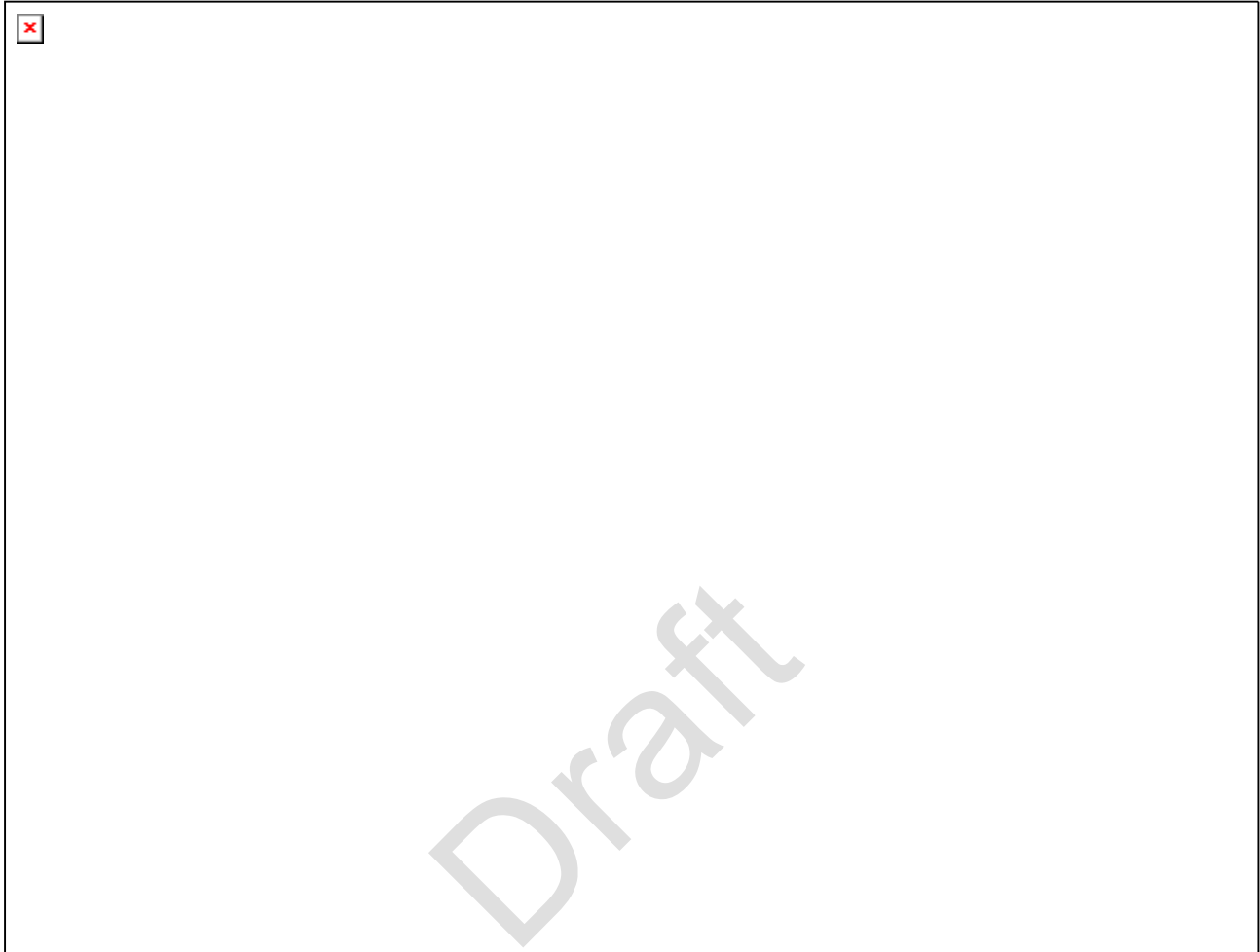
8.1.3.2.2.3 1-Color VT100 with 2nd-Color Highlight via Mac OS X iTerm



8.1.3.2.2.4 8-Color XTERM via Mac OS X iTerm



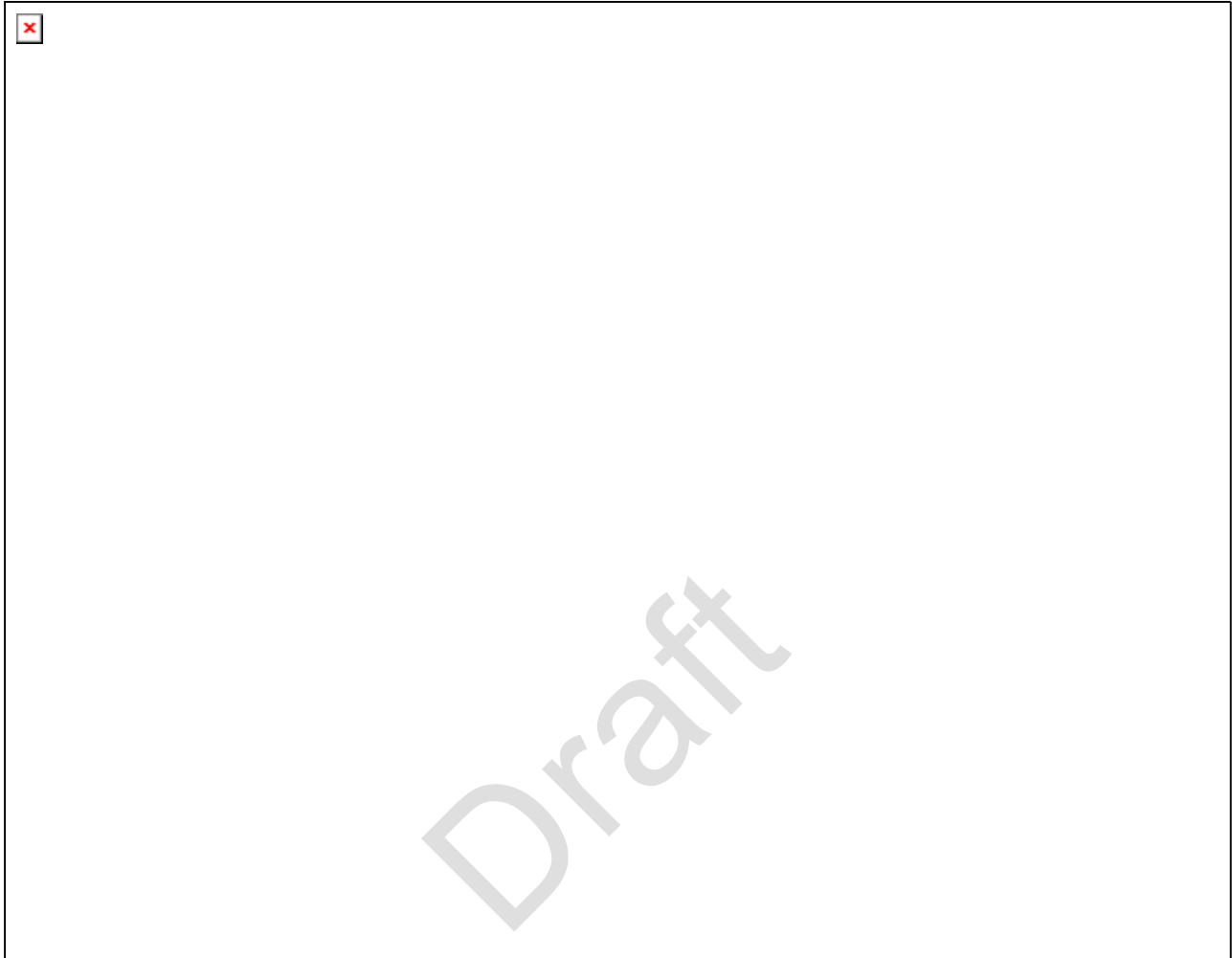
8.1.3.2.5 8-Color XTERM via Mac OS X Terminal



8.1.3.2.3 Microsoft Windows (with free Linux-like Cygwin add-on from Red Hat) Platform

- *8-Color Terminal via Cygwin Console* (on page 326)
- *1-Color VT100 via Cygwin-X Console* (on page 327)
- *1-Color VT100 with 2nd-color Highlight via Cygwin Console* (on page 328)
- *8-Color XTERM via Cygwin-X Console* (see "*Keyboard*" on page 331)

8.1.3.2.3.1 8-Color Terminal via Cygwin Console

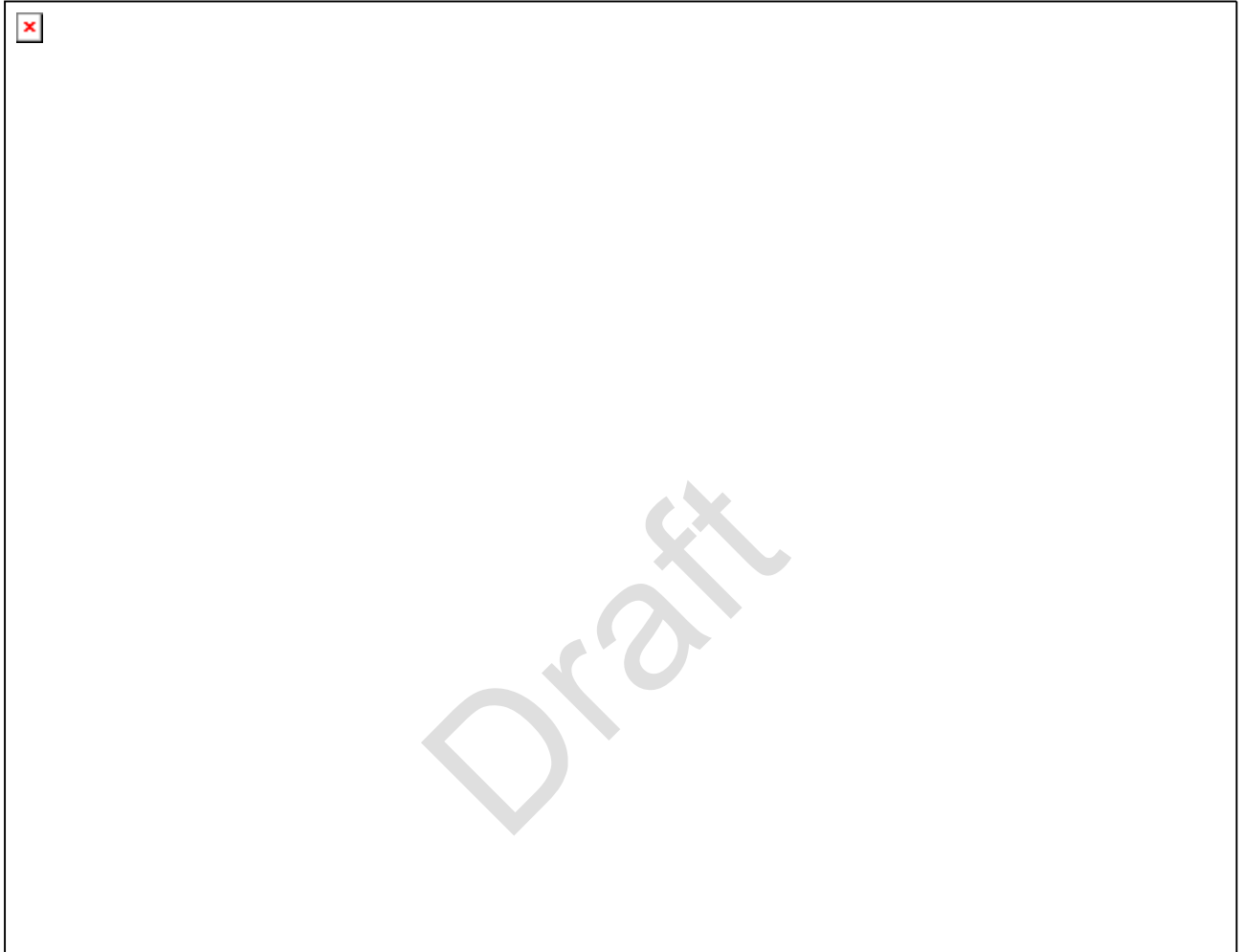


8.1.3.2.3.2 1-Color VT100 via Cygwin-X Console

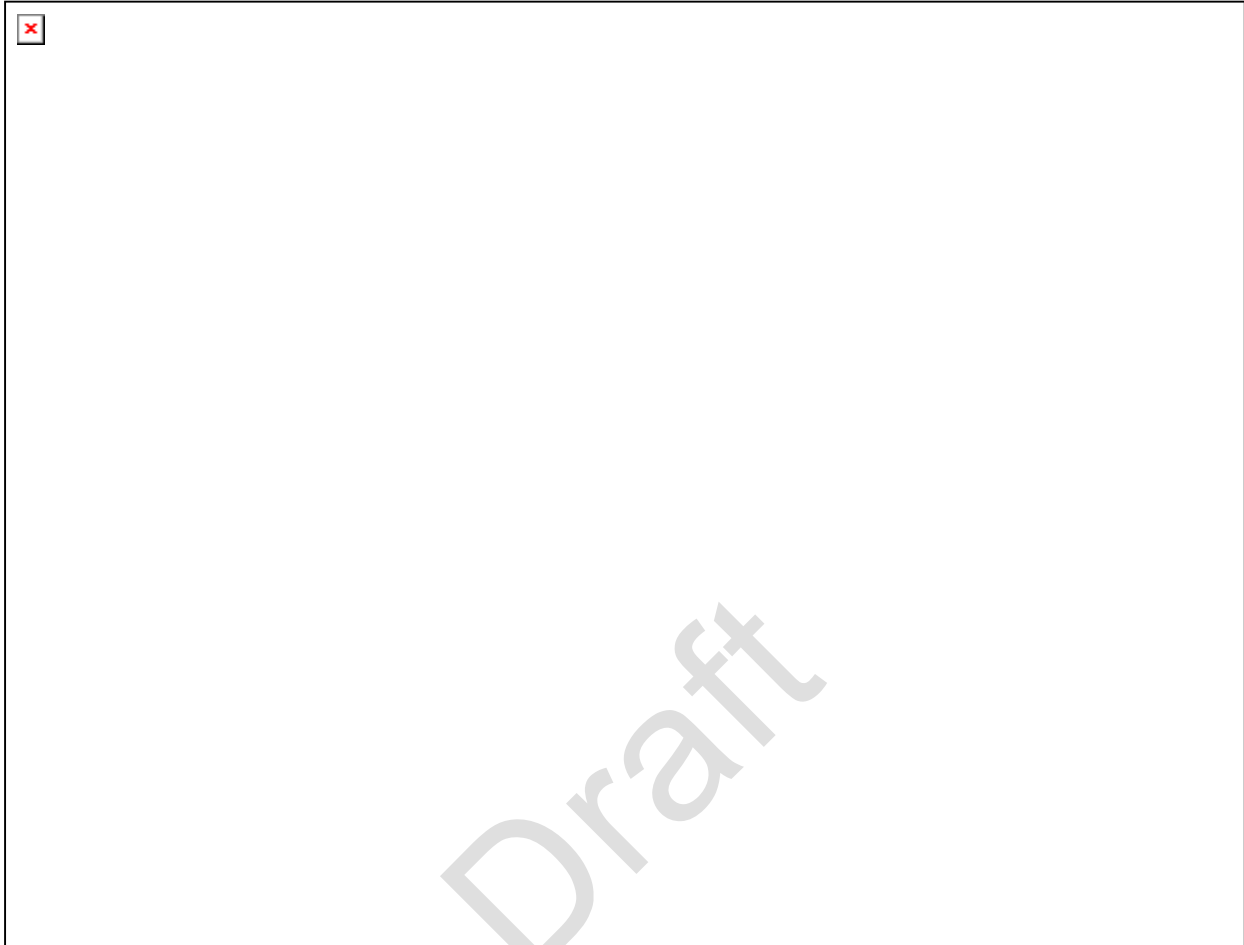


8.1.3.2.3.3 1-Color VT100 with 2nd-color Highlight via Cygwin Console

The Cygwin Console does not provide a usable VT100 Terminal Emulation. It repeatedly scrolls when rendering border generating characters.



8.1.3.2.3.4 8-Color XTERM via Cygwin-X Console



8.1.3.2.4 Unix (OpenIndiana) Platform

8-Color XTERM via OpenIndiana Terminal (on page 331)

From Wikipedia, the free encyclopedia:

"OpenIndiana is a Unix-like computer operating system released as free and open source software. It forked from OpenSolaris after the discontinuation of that project by Oracle[1] and aims to continue development and distribution of the OpenSolaris codebase.[2] The project operates under the umbrella of the Illumos Foundation.[2] The stated aim of the project is "[...] to become the defacto OpenSolaris distribution installed on production servers where security and bug fixes are required free of charge".[3]"

"Project Indiana was originally conceived by Sun Microsystems, to construct a binary distribution around the OpenSolaris source code base.[4]"

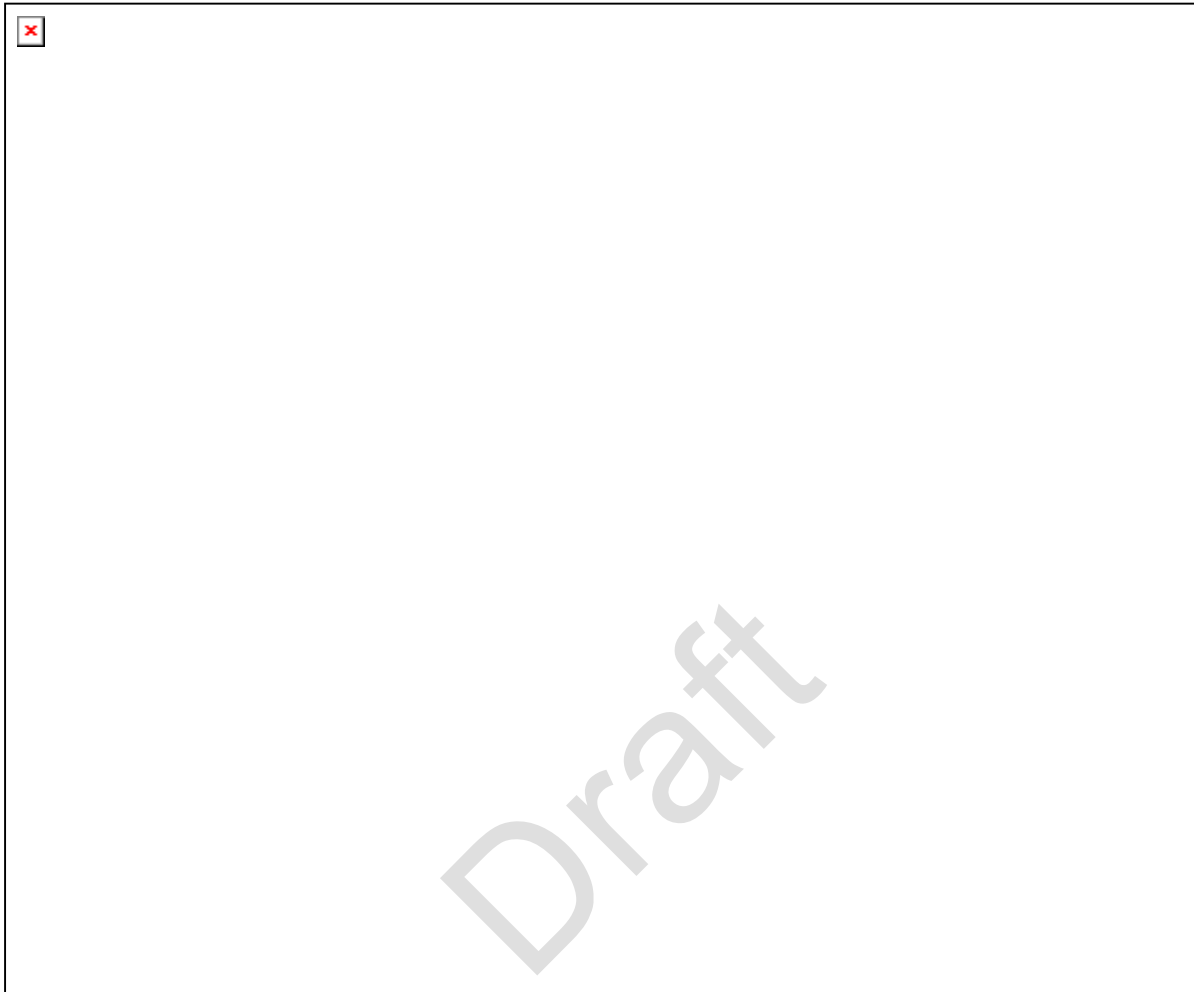
OpenIndiana was conceived after negotiations of a takeover of Sun Microsystems by Oracle were proceeding, in order to ensure continued availability and further development of an OpenSolaris-based OS, as it is widely used. Uncertainty among the OpenSolaris development community led some developers to form tentative plans for a fork of the existing codebase.

These plans came to fruition following the announcement of discontinuation of support for the OpenSolaris project by Oracle.[5][6]"

8.1.3.3 Application Notes

- 1 Parallels Desktop for Mac** (releases 7, 8, 9 and 10) lacked sufficient support for creating a usable OpenSolaris virtual Machine. Unable to find tools and guidance for establishing network connections needed to download tools and updates for the following:
 - a) Sun Microsystems released OpenSolaris 9 (SunOS 5.9) for Sparc and x86 platforms.
 - b) Oracle released OpenSolaris 10 (SunOS 5.10) for Sparc and x86 platforms.
- 2 VMware Fusion** (release 5) provided sufficient support for creating a usable OpenSolaris virtual Machine.
 - a) Oracle released OpenSolaris 10 (SunOS 5.10) for Sparc and x86 platforms. Oracle provided no updates.
 - b) Illumos Foundation released OpenIndiana 151a5 (SunOS 5.11) for Sparc and x86 platforms. Illumos currently provides a limited set of updates. For example, Firefox remains at release 3.6 rather than at its latest release 15. One can at least download those development tools (compilers and utilities) not in the original installation.

8.1.3.3.1.1 8-Color XTERM via OpenIndiana Terminal



8.1.3.4 Keyboard

Input is submitted by the operator via a terminal keyboard device. The keyboard consists of a matrix of push buttons. The operator may press a letter, digit, punctuation symbol or function key. The user may also simultaneously press and hold a combination of the shift, control or alt keys.

8.1.3.5 Mouse

Input of GUI object selections is submitted by the operator via a terminal mouse device. A touchscreen, trackpad or trackball device may be substituted for a mouse device.

The input consists of the cursor position data needed to identify the selected GUI-object, the button identity (left, middle, right) and the type of selection (pressed and held, released, single click, double click and triple click).

9 APPENDIXES

Draft

10 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 53) - Linux (Fedora 20 / OpenSUSE 13.1 / Scientific (CentOS) 6.5 / Ubuntu 12.04 & 14.04), Mac OS X (10.7 / 10.8 / 10.9), Microsoft Windows (8.1/8 Professional / 7 Professional / XP Professional each with Cygwin 1.7.x) and Unix (PC-BSD 10 & OpenIndiana 151a8 / Solaris 11 / SunOS 5.11).
 - 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) 1999-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 58) - 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) 1999-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 342) - Link to YouTube Video by Mark Kohler.

10.1 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 10.9-10.10 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 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 (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 9-10 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 7.0 64-bit, Fedora 21 64-bit, OpenSUSE 12.2 & 13.1 64-bit, Scientific 6.4 & 6.5 64-bit, Ubuntu 12.04 & 14.04 32-bit with Python 2.7 and 3.2. Microsoft Windows (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 installation 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 support mouse even with xterm.) 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"	2007-model year, 2.33 GHz Intel Core 2 Duo	Host Operating System

MacBook Pro Laptop	<p>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. 	<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*

10.2 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>

10.3 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
 0:04 who wants to build an embedded system let's use paid time to build in
 medicine
 0:09 system
 0:09 okay in the right place the story so far
 0:14 hardware is cheap specifically the price difference between a bit
 0:19 and 32-bit processors a small and dwindling I miss them more and more
 0:24 places we can use Python
 0:25 that board has a 700 megahertz processor
 0:29 GPA you internet USB 256 megs of RAM
 0:33 as perfect for running Python his car Raspberry Pi
 0:37 i cost \$35 dollars my premise is the programmers develop faster
 0:42 with Linux in Python been a real time embedded operating system and see
 0:46 the tools and documentation a much better
 0:49 even when the documentation is the source just considering Python capable
 0:55 systems hardware resources can vary widely and when to use three examples
 0:59 for the stock
 1:00 electronic kiosks routers
 1:04 and thermostats with one problem
 1:08 Linux distributions are designed for desktops
 1:12 not embedded systems the solution is
 1:16 is the piece as you can from a Linux distribution and use Python
 1:19 to build the rest the first step is to choose the right distribution for
 your
 1:25 product
 1:25 kiosks usually have PC class hardware
 1:29 ice cream next 36 processor a touch screen a pointing device
 1:34 and tens of gigabytes of storage but there are some important differences
 1:39 a desktop is one user and lots about locations
 1:42 a kiosk is one application and lots of users
 1:45 KS have no human administrator our configuration and upgrades must be
 1:50 automatic
 1:51 but was great support for PC hardware
 1:54 and 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 pot 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 Linex 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

17:15 structuring code for portability are not mine I store them
17:19 if you wanna steal them to read these papers is clinics where you can
17:26 and use Python for the rest rate better embedded systems faster
17:29 i'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

21:00that would be another interesting approach to try didn't know they support
 21:04a lot of
 21:05I'm different processors because they
 21:09they 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"

10.3.1 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

11 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.

11.1 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.

11.2 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 ▪ Display (68-color pixel mode) 	<ul style="list-style-type: none"> ▪ Keyboard ▪ Mouse, trackball, touchpad or touchscreen ▪ Keyboard Shortcut Key and/or Optional Mouse (Future) ▪ 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 (Debian-/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 (Debian-/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	<ul style="list-style-type: none"> ▪ "wxWidgets" internally uses host 	<ul style="list-style-type: none"> ▪ "tsWxGTUI_PyVx" internally uses Python

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
Interface Software	<p>Operating System specific API for its Terminal Device Interface.</p> <ul style="list-style-type: none"> ▪ 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. 	<p>"Curses" module specific API for its Terminal Device Interface.</p> <ul style="list-style-type: none"> ▪ 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. ▪ 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.

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"> ▪ 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.
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

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
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 emulator software. ▪ 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.

11.3 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 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.) 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)

12 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. <p>Access (data base)</p> <p>Excel (spreadsheet)</p> <p>PowerPoint (presentation)</p> <p>Word (word processor)</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
	<ul style="list-style-type: none"> ▪ Microsoft Visio --- A software application (http://www.microsoftstore.com/store/msusa/en_US/list/Visio/categoryID.62687700) intended to be used by knowledge workers. Visio (diagrams) ▪ Fineprint Software (http://fineprint.com) 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.) pdfFactory Pro (An application, for Microsoft Windows, that provides Adobe PDF compatible output.) <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

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. 10 --- SDIST Appendixes ▪ Vol. 11 --- SDIST Dictionary ▪ 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</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>format, in the subdirectory named ".tsDocCLI". The subdirectory is organized, by topic, to contain the following:</p> <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

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>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.</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"

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_5_tsWxGTUI_HighLevel_WidgetApi_application.py" ▪ "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"

13 APPENDIX D - ACCOMPLISHMENTS

(Draft)

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

- *Capabilities* (on page 385)
- *Limitations* (on page 398)
- *Comparison of wxPython with tsWxGTUI* (see "*Comparison of wxPython with tsWxGTUI (Development Plan)*" on page 399)

13.1 Capabilities

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

13.1.1 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.

13.1.2 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 tsCxBlobalsPkg

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.

13.1.3 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.

13.1.4 tsTestsCLI Capabilities**1 tsTestsCLI Functional Capabilities**

TBD

13.1.5 tsUtilities Capabilities**1 tsUtilities Functional Capabilities**

TBD

13.1.6 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.

MenuBar 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% Offset
0% |                                         | Empty 0% max buffer
+-----+-----+-----+-----+

+-----+-----+-----+-----+ 0% Offset
0% |#####|                                         | First 25% max buffer
+-----+-----+-----+-----+

+-----+-----+-----+-----+ 25% Offset
25% |          #####|                                         | Second 25% max buffer
+-----+-----+-----+-----+

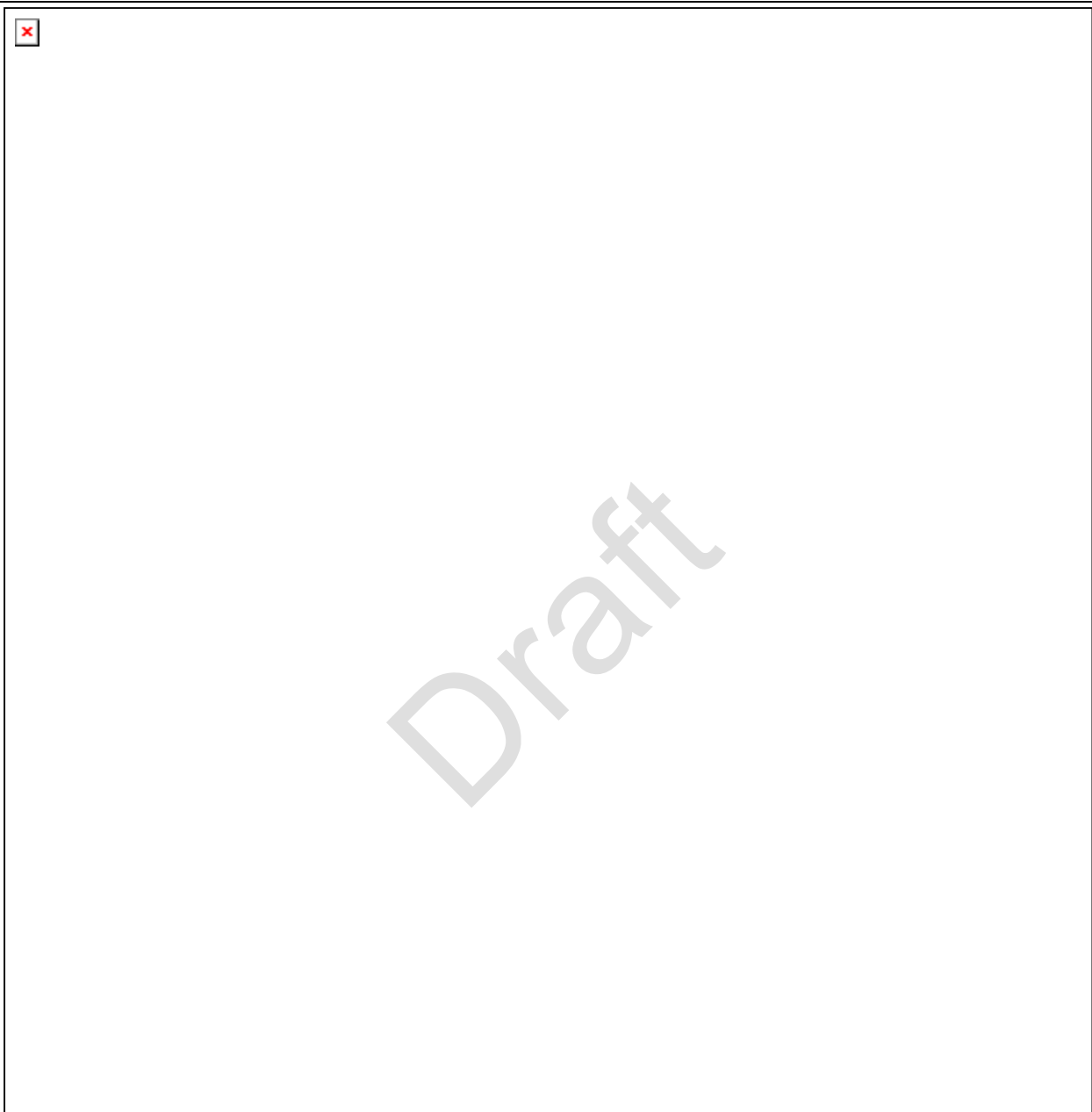
```

50%	<pre> +-----+-----+-----+-----+ ##### +-----+-----+-----+-----+ </pre>	50% Offset Third 25% max buffer
75%	<pre> +-----+-----+-----+-----+ ##### +-----+-----+-----+-----+ </pre>	75% Offset Fourth 25% max buffer
0%	<pre> +-----+-----+-----+-----+ ##### ##### ##### +-----+-----+-----+-----+ </pre>	0% Offset First 50% max buffer
25%	<pre> +-----+-----+-----+-----+ ##### ##### ##### +-----+-----+-----+-----+ </pre>	25% Offset Middle 50% max buffer
50%	<pre> +-----+-----+-----+-----+ ##### ##### ##### +-----+-----+-----+-----+ </pre>	50% Offset Last 50% max buffer
0%	<pre> +-----+-----+-----+-----+ ##### ##### ##### +-----+-----+-----+-----+ </pre>	0% Offset First 75% max buffer
25%	<pre> +-----+-----+-----+-----+ ##### ##### ##### +-----+-----+-----+-----+ </pre>	25% Offset Last 75% max buffer
100%	<pre> +-----+-----+-----+-----+ ##### ##### ##### +-----+-----+-----+-----+ </pre>	0% Offset All 100% 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).

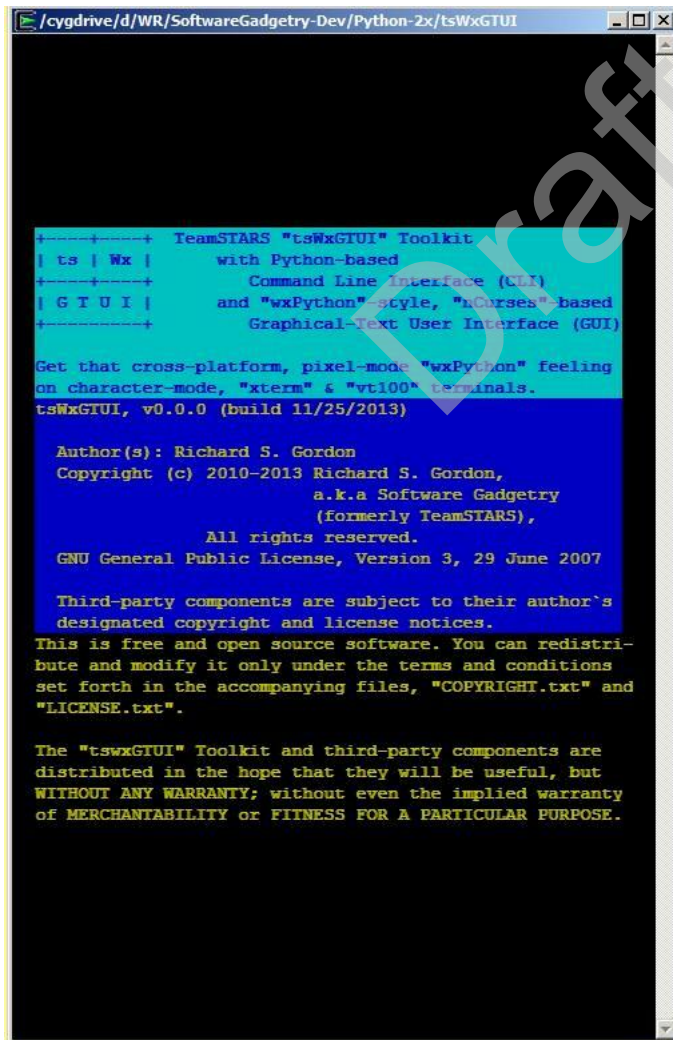
ScrolledWindows - Composite of various ScrollBarButtons, ScrollBarGauge(s) and ScrolledText window(s) as depicted in the following figure.



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.



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.

13.1.7 **tsToolsGUI Capabilities**

1 **tsToolsGUI Functional Capabilities**

TBD

13.1.8 **tsTestsGUI Capabilities**

1 **tsTestsGUI Functional Capabilities**

TBD

13.2 Limitations

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

13.2.1 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.

13.2.2 tsLibGUI Functional Limitations

- 1** See: *Capabilities* (on page 385)
- 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.

13.3 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 	<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.

13.4 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

14 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 406)
- 2 *Command Line Interface (CLI)* (on page 422)
 - a) *Shell Definition & History* (on page 426)
 - b) *Text-based User Interface (TUI)* (on page 429)
 - c) *POSIX Terminal Device Interface (TDI)* (see "*POSIX Terminal Device Interface (TDI)*" on page 433)
- 3 *Graphical User Interface (GUI)* (on page 435)
 - a) *Graphical Device Interface (GDI)* (on page 435)
 - b) *Widget Toolkit* (on page 436)
- 4 *Python Programming Language* (on page 445)

14.1 VGA-compatible Text Mode

This section provides a perspective for understanding the historical capabilities and limitations of the terminal (or terminal emulator) display typical of a Command Line Interface.

The computer platforms available in 2013, just as the Intel MDS (Microcomputer Development System) available in 1983 provide a default display for text with 80 columns and 25 rows. Platform-specific terminal emulators provide different keyboard and mouse input features but the same xterm, xterm-color, xterm-16color, xterm-256color, vt100 and vt220 display features.

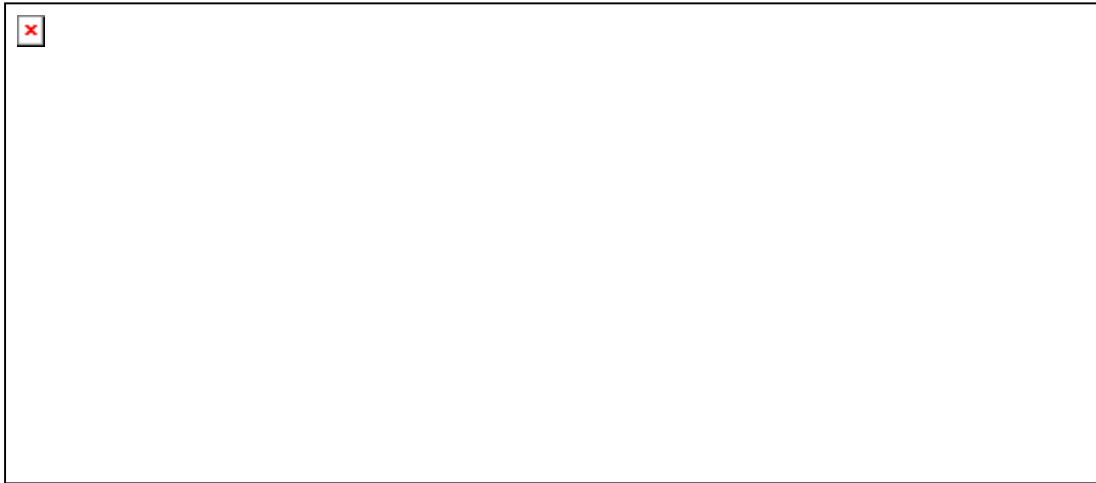
It has been the 'tsWxGTUI' Toolkit author's experience that this has been the case regardless of the host computer's hardware (IBM Cell Broadband Engine, Intel Core i7/Core 2 Duo/Pentium, Motorola PowerPC/68K and Oracle/Sun Sparc) or operating system software (Linux, Mac OS X, Microsoft Windows and Unix).

Thomas E. Dickey is a free software developer of several widely used programs and libraries related to text terminals and terminal emulators. He is the current developer or maintainer of: **xterm**, the X11 terminal emulator; **terminfo**, used by almost all full-screen text mode programs on a Unix system; and GNU **ncurses**, also used by most programs that display "full-screen" text. He published a FAQ on xterm-8bit, xterm-16color, xterm-256color at <http://invisible-island.net/xterm/xterm.faq> (<http://invisible-island.net/xterm/xterm.faq>).

From Wikipedia, the free encyclopedia:

"The implementation of computer monitor text mode on VGA-compatible hardware is quite complex. Its use on PC-compatible computers was widespread in 1980s–1990s (particularly under DOS systems), but persists today for some applications even on modern desktop computers. Main features of VGA text mode are colored (arbitrary 16 color palette) characters and their background, blinking, various shapes of the cursor (block/underline/hidden static/blinking), loadable fonts (with various glyph sizes). Linux console usually (but not necessarily) uses hardware VGA-compatible text modes, and Win32 console environment has an ability to switch its application to text mode for some text window sizes.

Distinctive features of VGA text



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]

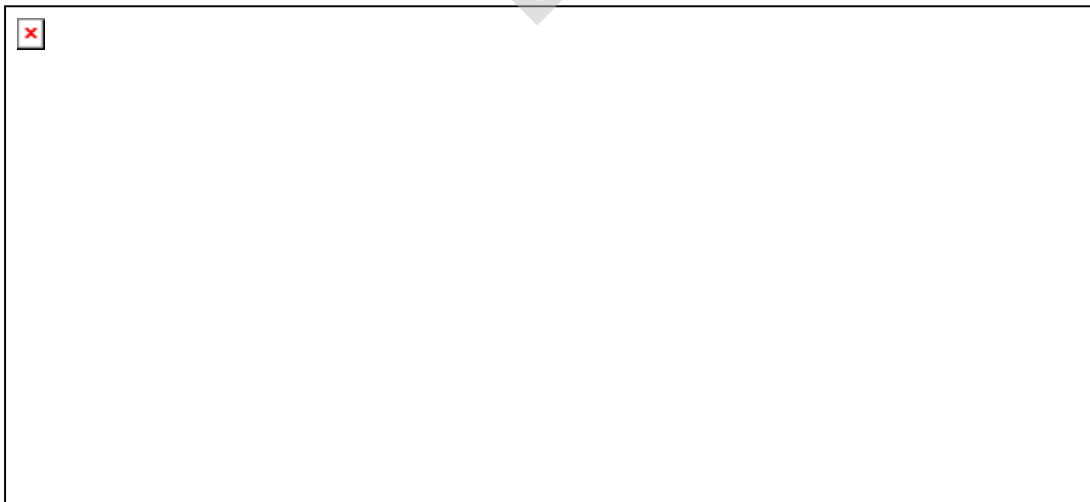
Screen fonts used in EGA and VGA are monospace raster fonts containing 256 glyphs with 8 dots glyph width and a some glyph height fixed for each font, less or equal to 32. Each row of a glyph is coded in a 8 bit byte, with high bits at the left and low at the right. Along with several hardware-dependent 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.



CodePage-737



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

14.2 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 Msvcr, 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

14.3 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

14.3.1 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."

14.3.1.1 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."

14.3.1.2 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.

14.3.2 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]"

14.3.3 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."

14.4 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 'gooney')[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)."

14.4.1 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."

14.4.2 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 436)

14.4.2.1 Partial List of Widget Toolkits

Excerpted From Wikipedia, the free encyclopedia.

- *Low Level* (on page 437)
- *High Level* (on page 437)

14.4.2.1.1 Low Level

Excerpt From Wikipedia, the free encyclopedia:

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

14.4.2.1.2 High Level

Excerpt From Wikipedia, the free encyclopedia:

- *On Mac OS X* (on page 437)
- *On Windows* (on page 437)
- *On Unix under X11* (on page 438)

14.4.2.1.2.1 On Mac OS X

Excerpt From Wikipedia, the free encyclopedia:

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

14.4.2.1.2.2 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

14.4.2.1.2.3 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

14.4.2.2 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 occurances 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 433) (such as "curses")."

14.4.2.2.1 "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

14.4.2.2.1.1 "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

14.4.2.2.1.1.1 "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."

14.5 Python Programming Language

- 1 *Python Technology Overview* (on page 446)
- 2 *CLI Shell Mode Examples* (on page 451)
- 3 *TUI Low-Level Curses Mode Examples* (on page 473)
- 4 *GUI High-Level wxPython Mode Examples* (on page 485)

14.5.1 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/>

14.5.2 CLI Shell Mode Examples

14.5.2.1 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>

14.5.2.2 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>

14.5.2.3 Python (CLI-mode) Structured Script Demo

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

14.5.2.4 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	<code>print()</code>
Report events that occur during normal operation of a program (e.g. for status monitoring or fault investigation)	<code>logging.info()</code> (or <code>logging.debug()</code> 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."

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

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

    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()

```

14.5.2.6 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 = '\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\tkw=%s' % (str(args), str(kw))
## msg = '\n\t%s\n\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%s;\n\t\t%s' % (fmt1, fmt2, fmt3)
## print(msg)
## myLogger.debug(msg)


## fmt1 = '\n\t%s.mainTest (command line argv): ' % label
## fmt2 = '\n\t\t args (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%s: %s' % (str(key), str(value))
##     else:
##         text += ', \n\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 = '\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\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\t%s.mainTest (command line argv): ' % label
        fmt2 = '\n\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%s: %s' % (str(key), str(value))
    else:
        text += ', \n\t\t\t%s: %s' % (str(key), str(value))
text += '}'
fmt3 = '\n\t\t\toptions (sorted)= %s' % text
msg = fmt1 + fmt2 + fmt3
print(msg)
myLogger.debug(msg)

return (args, options)

#-----

def EntryPoint(*args, **kw):
    '''
    Application Programming Interface, It gathers and displays
    configuration parameters as keyword-value pair options and
    positional arguments.
    '''
    myLogger = Logger.TsLogger(threshold=Logger.DEBUG,
                               name='EntryPoint')

    fmt1 = '\n\n  EntryPoint:'
    fmt2 = '\n\n      myLogger=%s' % myLogger
    fmt3 = '\n\n      appLogger=%s' % myLogger.appLogger
    fmt4 = '\n\n      theLogName=%s' % myLogger.theLogName
    fmt5 = '\n\n      theLogPath=%s' % myLogger.theLogPath
    fmt6 = '\n\n      theLogThreshold=%s\n\n' % myLogger.theLogThreshold
    msg = fmt1 + fmt2 + fmt3 + fmt4 + fmt5 + fmt6
    print(msg)
    myLogger.debug(msg)

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

    if True:
        (args, options) = mainTest(*args, **kw)
    else:
        mainTest(*args, **kw)
        args = myApp.args
        options = myApp.options

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

```

```
#-----  
  
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()
```

14.5.3 TUI Low-Level Curses Mode Examples

14.5.3.1 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\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()

```

14.5.3.2 Python (TUI-mode) "Curses" Multi-Panel Control Demo

EXAMPLE	SOURCE CODE
Command Line Interface launches Python 2.7 "Curses" (TUI-mode) Application Program	\$ python2.7 test_nCursesDemo.py
	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__)
```


14.5.4 GUI High-Level wxPython Mode Examples

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

EXAMPLE	SOURCE CODE
Command Line Interface launches Python 2.7 "wxPython" (GUI-mode) Application Program	<pre>\$ python2.7 the_wxPython_Main_Program.py</pre>
	<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>

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

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

```

#!/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\twxWidgets team.' + \
                 '\n\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\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():

```



```
fmt3 = 'options (unsorted)=%s' % str(options)
msg = '\n\t%s\n\t\t%s;\n\t\t%s' % (fmt1, fmt2, fmt3)
print(msg)
myLogger.debug(msg)


fmt1 = '\n\t%s.mainTest (command line argv): ' % label
fmt2 = '\n\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%s: %s' % (str(key), str(value))
    else:
        text += ', \n\t\t\t%s: %s' % (str(key), str(value))
text += '}'
fmt3 = '\n\t\ttoptions (sorted)= %s' % text
msg = fmt1 + fmt2 + fmt3
print(msg)
myLogger.debug(msg)


return (args, options)


#-----

def EntryPoint(*args, **kw):
    '''
    Application Programming Interface, It gathers and displays
    configuration parameters as keyword-value pair options and
    positional arguments.
    '''
    myLogger = Logger.TsLogger(threshold=Logger.DEBUG,
                               name='EntryPoint')

    fmt1 = '\n\n  EntryPoint:'
    fmt2 = '\n\n      myLogger=%s' % myLogger
    fmt3 = '\n\n      appLogger=%s' % myLogger.appLogger
    fmt4 = '\n\n      theLogName=%s' % myLogger.theLogName
    fmt5 = '\n\n      theLogPath=%s' % myLogger.theLogPath
    fmt6 = '\n\n      theLogThreshold=%s\n\n' % myLogger.theLogThreshold
    msg = fmt1 + fmt2 + fmt3 + fmt4 + fmt5 + fmt6
    print(msg)
    myLogger.debug(msg)


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


if True:
    (args, options) = mainTest(*args, **kw)
else:
    mainTest(*args, **kw)
```

```
        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()
```

15 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 499)
- *System Maintenance* (on page 506)
- *System Operation* (on page 509)

15.1 System Development

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

- *Rationale* (on page 500)
- *Requirements* (on page 500)
- *Goals* (on page 501)
- *Non-Goals* (on page 502)
- *Assumptions* (on page 502)
- *Deliverables* (on page 504)
- *Stakeholders* (on page 505)
- *Prerequisites* (on page 506)
- *Dependents* (on page 506)

15.1.1 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.

15.1.2 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.

15.1.3 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.

15.1.4 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.

15.1.5 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)

15.1.6 Deliverables

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

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

15.1.7 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

15.1.8 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 touch screen 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.

15.1.9 Dependents

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

- Documentation Requirements.

15.2 System Maintenance

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

- *Developer Platform Maintenance* (on page 507)
- *Operator Platform Maintenance* (on page 508)

15.2.1 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

15.2.2 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) applications 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) 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	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

15.3 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