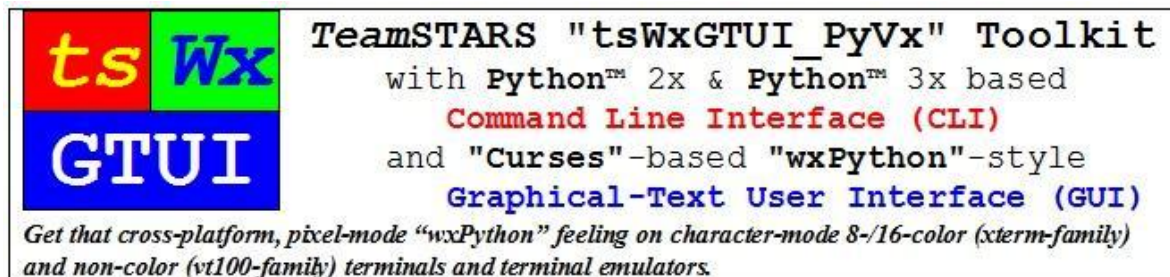


Software Requirements Specification

Vol. 7 - "tsWxGTUI_PyVx" Toolkit

Rev. 0.0.2 (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_Repository/Documents"*.

Draft

Contents

1	SCOPE (System Specification)	7
1.1	Identification (System Specification)	7
1.2	Purpose (System Specification)	10
1.3	Document Overview (System Specification)	12
2	DEFINITIONS, ABBREVIATIONS, AND ACRONYMS	25
3	REQUIREMENTS	49
3.1	Required States and Modes	49
3.1.1	Required States	49
3.1.2	Required Modes	51
3.1.3	Required Variants	57
3.2	CSCI Capability Requirements	61
3.2.1	(CSCI Capability Template)	62
3.2.2	Command Line Interface Capability	62
3.2.3	Graphical User Interface Capability	66
3.3	CSCI External Interface Requirements	93
3.3.1	Interface Identification And Diagrams	93
3.3.2	Human Interface	94
3.4	CSCI Internal Interface Requirements	96
3.5	CSCI Internal Data Requirements	96
3.6	Adaptation Requirements	97
3.7	Safety Requirements	97
3.8	Security and Privacy Requirements	98
3.9	CSCI Environment Requirements	98
3.10	Computer Resource Requirements	98
3.10.1	Computer Hardware Requirements	98
3.10.2	Computer Hardware Resource Utilization Requirements	99
3.10.3	Computer Software Requirements	99
3.10.4	Computer Communications Requirements	99
3.11	Software Quality Factors	100
3.12	Design and Implementation Constraints	100
3.12.1	Python Language Design and Implementation Constraints	101

3.13	Personnel-related Requirements.....	108
3.14	Training-related Requirements.....	109
3.15	Logistics-related Requirements.....	109
3.16	Other Requirements	109
3.17	Packaging Requirements.....	109
3.18	Precedence and Criticality of Requirements	110
4	QUALIFICATION PROVISIONS	111
5	REQUIREMENTS TRACEABILITY	113
6	NOTES	115
6.1	TO-DO-LIST.....	117
6.1.1	New Features:.....	117
6.1.2	Modifications:.....	117
6.1.3	Troubleshooting:.....	118
6.1.4	Validation/Regression Test:.....	119
6.2	Command Line Interface Library and Tools	120
6.3	Graphical Text User Interface Library and Tools (Draft).....	120
7	APPENDIXES (System Specification)	125
8	APPENDIX A - REQUIRED STATES AND MODES	127
8.1	Required States.....	127
8.1.1	Pre-Startup State	127
8.1.2	Startup State.....	128
8.1.3	Operating State	128
8.1.4	Shutdown State	128
8.2	Required Modes	129
8.2.1	Video Adapter Modes.....	129
8.2.2	User Interface Modes.....	132
9	APPENDIX B - SYSTEM CAPABILITY REQUIREMENTS	137
9.1	Hardware Capabilities	137
9.1.1	Central Processing Unit Capability.....	138
9.1.2	Random Access Memory Capability	138
9.1.3	Non-Volatile Storage Capability	138
9.1.4	Network Interface Device Capability	138
9.1.5	Keyboard Device Capability.....	138
9.1.6	Pointing Device Capability	139
9.1.7	Display Device Capability.....	139
9.1.8	Printer Device Capability	147
9.2	Software Capabilities	147
9.2.1	Host Computer Operating System Capability.....	148
9.2.2	Command Line Interface Capability.....	150
9.2.3	Graphical-Style User Interface Capability.....	151

9.2.4	Text Editing Capability.....	151
9.2.5	Word Processor Capability.....	152
9.2.6	Debugging Capability.....	152
9.2.7	Python Programming Capability.....	153
9.2.8	Host Console Terminal Application Capability.....	154
9.2.9	Optional Terminal Device Emulation Capability.....	155
9.2.10	"tsWxGTUI" Toolkit Capability.....	155
9.3	Platform Interface Capability.....	186
9.3.1	Keyboard Interface Capability.....	186
9.3.2	Mouse Interface Capability.....	186
9.3.3	Display Interface Capability.....	186
9.3.4	Event Logging Interface Capability.....	186
9.3.5	Process & Thread Launching & Terminating Capability.....	186
9.3.6	Process Scheduling Capability.....	186
9.3.7	Inter-Process Communication Capability.....	186
9.4	Application Programming Interface Capability.....	186
9.4.1	Python Built-In Module API Capability (Draft).....	186
9.4.2	High-Level GUI "wxPython"-style API Capability (Draft).....	188
9.4.3	Low-Level GUI "nCurses"-style API Capability (Draft).....	191
9.5	Operator Interface Capability.....	198
9.5.1	Command Line Interface (CLI) Capability.....	198
9.5.2	Graphical-style User Interface (GUI) Capability.....	199
10	APPENDIX C - SYSTEM EXTERNAL INTERFACE REQUIREMENTS	201
10.1	(Project-unique Identifier Of Interface Template).....	202
10.2	Interface Identification and Diagrams.....	205
10.2.1	Hardware Components.....	205
10.2.2	Software Components.....	224
10.2.3	Architecture.....	227
11	APPENDIX D - SYSTEM INTERNAL INTERFACE REQUIREMENTS	237
11.1	Toolkit Architecture.....	237
11.1.1	"object" Dependents.....	238
11.1.2	"wxObject" Dependents.....	238
11.1.3	"wxEvtHandler" Dependents.....	239
11.1.4	"wxWindow" Dependents.....	240

12	APPENDIX E - SYSTEM INTERNAL DATA REQUIREMENTS	241
13	APPENDIX F - ADAPTATION REQUIREMENTS	243
14	APPENDIX G - SAFETY REQUIREMENTS	245
15	APPENDIX H - SECURITY AND PRIVACY REQUIREMENTS	247
16	APPENDIX I - SYSTEM ENVIRONMENT REQUIREMENTS	249
17	APPENDIX J - COMPUTER RESOURCE REQUIREMENTS	251
18	APPENDIX K - SYSTEM QUALITY FACTORS	255
19	APPENDIX L - DESIGN AND CONSTRUCTION CONSTRAINTS	257
20	APPENDIX M - PERSONNEL-RELATED REQUIREMENTS	259
21	APPENDIX N - TRAINING-RELATED REQUIREMENTS	261
22	APPENDIX O - LOGISTICS-RELATED REQUIREMENTS	263
23	APPENDIX P - OTHER REQUIREMENTS	265
24	APPENDIX Q - PACKAGING REQUIREMENTS	267
25	APPENDIX R - PRECEDENCE AND CRITICALITY OF REQUIREMENTS	269
26	APPENDIXES (Software Requirements)	271

27 APPENDIX A - Nature of System and Software 271

27.1	Command Line and Graphical User Interfaces	272
27.2	System Configurations	273
27.3	Use Case(s)	274
27.3.1	System Administrator	275
27.3.2	Software Engineer	275
27.3.3	System Operator	276

28 APPENDIX B - Software Design Constraints 281

28.1	README	282
28.2	README1-Introduction.txt	288
28.3	README2-Repository.txt	306
28.4	README3-Documents.txt	315
28.5	README4-ManPages.txt	318
28.6	README5-Notebooks.txt	325
28.7	README6-SourceDistributions.txt	332
28.8	README7-DeveloperSandboxes.txt	348
28.9	README8-SitePackages.txt	364

29 TECHNICAL IMPACT ANALYSIS 371

29.1	Performance	371
29.2	Hardware Requirements	371
29.3	Software Requirements	371
29.4	Technical Risks	371
29.5	Documentation Requirements	372

30 TEST REQUIREMENTS AND RESTRICTIONS 373

30.1	Test Requirements	373
30.2	Test Restrictions	373

31 USE CASES 375

32 TRACEABILITY INFORMATION 377

33 CLASS LIST BY CATEGORY 379

33.1	Basic Windows	381
33.2	Window Layout	382
33.2.1	Related Overviews: HLINK_46	382
33.3	Managed Windows	382
33.3.1	Related Overviews: HLINK_56	383
33.4	Menus	383
33.5	Controls	384
33.6	Validators	386
33.6.1	Related Overviews: HLINK_122	386

33.7	Picker Controls.....	387
33.8	Miscellaneous Windows	388
33.9	Window Docking (wxAUI).....	389
33.9.1	Related Overviews: HLINK_155	389
33.10	Common Dialogs	389
33.10.1	Related Overviews: HLINK_160	390
33.11	HTML	391
33.12	Device Contexts	391
33.12.1	Related Overviews: HLINK_193	392
33.13	Graphics Context classes	393
33.14	Graphics Device Interface.....	394
33.15	Image and bitmap classes.....	394
33.15.1	Related Overviews: HLINK_227	395
33.16	Events.....	395
33.16.1	Related Overviews: HLINK_241	396
33.17	Application and Process Management	398
33.18	Printing Framework	398
33.18.1	Related Overviews: HLINK_295	399
33.19	Document/View Framework.....	399
33.19.1	Related Overviews: HLINK_308	400
33.20	Clipboard and Drag & Drop.....	401
33.20.1	Related Overviews: HLINK_320	401
33.21	Virtual File System	402
33.22	Threading	402
33.22.1	Related Overviews: HLINK_336	402
33.23	Runtime Type Information (RTTI).....	403
33.23.1	Related Overviews: HLINK_344	403
33.24	Debugging.....	403
33.24.1	Related Overviews: HLINK_347	404
33.25	Logging.....	404
33.25.1	Related overview: HLINK_356.....	405
33.26	Data Structures.....	406
33.27	Text Conversion.....	407
33.27.1	Related Overviews: HLINK_393	407
33.28	Containers	407
33.28.1	Related Overviews: HLINK_394	408
33.29	Smart Pointers	408
33.30	File Handling	408
33.30.1	Related overview: HLINK_406.....	409
33.31	Streams.....	409
33.31.1	Related overview: HLINK_418.....	410
33.32	XML.....	411
33.33	Archive.....	411
33.34	XML Based Resource System (XRC).....	412
33.34.1	Related overview: HLINK_455.....	412
33.35	Networking	412
33.36	Interprocess Communication	413
33.36.1	Related overview: HLINK_467	413
33.37	Help.....	413
33.38	Multimedia.....	414
33.39	OpenGL	414
33.40	Miscellaneous	415


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 7)
- *Purpose (System Specification)* (on page 10)
- *Document Overview (System Specification)* (on page 12)

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 the following Python language specific versions:</p> <ul style="list-style-type: none"> ▪ TeamSTARS "tsWxGTUI_Py1x" Toolkit (reserved for legacy Python 1x; to be created by back-port from TeamSTARS "tsWxGTUI_Py2x") ▪ TeamSTARS "tsWxGTUI_Py2x" Toolkit (for mature Python 2x; once created as the Toolkit prototype; then upgraded as Python 2x evolves) ▪ TeamSTARS "tsWxGTUI_Py3x" Toolkit (for evolving Python 3x; once created as port from TeamSTARS "tsWxGTUI_Py2x; then upgraded as Python 3x evolves) ▪ TeamSTARS "tsWxGTUI_Py4x" Toolkit (reserved for future Python 4x; to be created by port from TeamSTARS "tsWxGTUI_Py3x"; then upgraded as Python 4x evolves)
Title	TeamSTARS "tsWxGTUI_PyVx" Toolkit with Python 2x & Python 3x based Command Line Interface (CLI) and "Curses"-based "wxPython"-style, Graphical-Text User Interface (GUI)
Identification Number	N/A

Version Number	0.0.2
Release Number	0.0.2 (pre-alpha)
Build Date	09/08/2015

Draft

Usage Terms and
Conditions --- Key
Points

This is free and open source software. The *TeamSTARS* "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.

Please note the following:

- 1 Each *TeamSTARS* "tsWxGTUI_PyVx" Toolkit distribution includes a root directory whose name ("tsWxGTUI") has a suffix "_PyVx" that reflects the associated Python language syntax version:
 - a) **"tsWxGTUI_Py1x"** - *Reserved* 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 2.0.0-2.7.10**.
 - c) **"tsWxGTUI_Py3x"** - Root of third generation syntax files and subdirectories for **Python 3.0.0-3.5.0**.

There is currently no support for release candidates for **Python 3.5.0** due to its redesigned process package which is no longer compatible with earlier Python 3x releases and with the existing Python 2x releases.
 - d) **"tsWxGTUI_Py4x"** - *Reserved* for future Root of next generation syntax files and subdirectories for Python 4.x.
- 2 You can use, study, modify and redistribute the distribution only under the *Terms and Conditions* files provided in the ".tsWxGTUI_PyVx/Documents" sub-directory:
 - a) **"Copyright.txt"** - Attribution for the principal Author(s) of intellectual property created specifically for the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit.
 - b) **"Credits.txt"** - Attribution for those third-party Author(s) whose intellectual property has been adapted for use in the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit.
 - c) **"License.txt"** - Statements of rights, obligations and limitations stipulated by the Authors of intellectual property included in the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit.
 - d) **"Notices.txt"** - Announcement calling the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit recipient's attention to the applicable "Copyright.txt", "Credits.txt" and

1.2 Purpose (System Specification)

The TeamSTARS "tsWxGTUI_PyVx" Toolkit's cross-platform design facilitates the creation, enhancement, troubleshooting, maintenance, porting and support of:

- 1 Mission-critical equipment used by commercial, industrial, medical and military customers.
Such equipment can include a broad range of embedded computer systems which satisfy the Platform Hardware and Software Requirements.
- 2 Application programs used for the local and remote supervisory control and data acquisition associated with automation, communication, control, diagnostic, instrumentation and simulation.
Such application software typically includes "operator-friendly" user interfaces.

The *TeamSTARS* "tsWxGTUI_PyVx" Toolkit includes 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 for the first generation Python language 1.0.0-1.6.1 (reserved for future back-port from *TeamSTARS* "tsWxGTUI_Py2x" Toolkit)
 - b) *TeamSTARS* "tsWxGTUI_Py2x" Toolkit for the second generation Python language 2.0.0-2.7.10
 - c) *TeamSTARS* "tsWxGTUI_Py3x" Toolkit for the third generation Python language 3.0.0-3.4.3
 - d) *TeamSTARS* "tsWxGTUI_Py4x" Toolkit for the fourth generation Python language 4.0.0 (reserved for future port from *TeamSTARS* "tsWxGTUI_Py3x" Toolkit)
 - e) *TeamSTARS* "tsWxGTUI_PyVx" Toolkit reserved for distribution 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. Mission-critical equipment typically includes embedded computer systems which 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 thereof 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	<p>This document alerts potential and existing users of the availability, capabilities, limitations and sources for the latest source code release and technical support.</p> <p>Included are the following topics:</p> <p>1 ANNOUNCEMENT</p> <p>a) About</p> <p>Platform Hardware and Software Requirements</p> <p>What is the toolkit designed to do?</p> <p>What is included in the release?</p> <p>How might you use the Toolkit?</p> <p>Where to get further information?</p>
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-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> <p>1 INTRODUCTION</p> <p>a) About</p> <p>Platform Hardware and Software Requirements</p> <p>What is the toolkit designed to do?</p> <p>What is included in the release?</p> <p>How might you use the Toolkit?</p> <p>Where to get further information?</p> <p>b) System Block Diagrams</p> <p>Block Diagram</p> <p>Stand Alone System Architecture</p> <p>Stand Among System Architecture</p> <p>c) Usage Terms & Conditions</p> <p>2 SCREENSHOTS</p> <p>a) Latest XTERM & VT100 Desktops</p> <p>b) Earlier XTERM Terminal Emulator with 8-Color / 64-Color Pairs</p> <p>c) Earlier VT100 Terminal Emulator with 1-Color / 2-Color Pairs</p>
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> <p>1 SCOPE (System Specification)</p> <p>a) Identification (System Specification)</p> <p>b) Purpose (System Specification)</p> <p>c) Document Overview (System Specification)</p> <p>2 SYSTEM OVERVIEW (tsWxGTUI Introduction)</p> <p>a) Purpose of System and Software</p> <p>b) General Nature of the System and Software</p>

		<ul style="list-style-type: none"> c) History of System Development, Operation and Maintenance <p>3 OPERATOR INTERFACE</p> <ul style="list-style-type: none"> a) Command Line Interface (CLI) b) Graphical User Interface (GUI) <p>4 APPLICATION PROGRAMMING INTERFACE</p> <ul style="list-style-type: none"> a) Command Line Interface API b) Graphical User Interface API <p>5 TOOLS & UTILITIES</p> <ul style="list-style-type: none"> a) tsLinesOfCodeProjectMetrics b) tsPlatformQuery c) tStripComments d) tsStripLineNumbers e) tsTreeCopy f) tsTreeTrimLines <p>6 DEFINITIONS, ABBREVIATIONS, AND ACRONYMS</p> <p>7 REFERENCED DOCUMENTS</p> <ul style="list-style-type: none"> a) Project Documents b) Release Distribution Documents c) External Documents <p>8 NOTES</p> <ul style="list-style-type: none"> a) Operator Interface Technology <p>9 APPENDIXES</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
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> <ul style="list-style-type: none"> a) Copyright - Identifies the original author(s) of source code and documentation for building block libraries and application programs. b) License - Identifies the original author's rules for using, modifying and redistributing source code and documentation for building block libraries and application programs. c) Notices - Identifier placed on copies of the source code and documentation to inform the world of copyright ownership and the applicable license. 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).
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> <ul style="list-style-type: none"> 1 SCOPE (System Specification) 2 DEFINITIONS, ABBREVIATIONS, AND ACRONYMS 3 OVERVIEW OF REQUIRED WORK <ul style="list-style-type: none"> a) Purpose b) Requirements and Constraints c) Concept 4 PLANS FOR PERFORMING GENERAL SOFTWARE DEVELOPMENT ACTIVITIES <ul style="list-style-type: none"> a) Software Development Process 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

		<ul style="list-style-type: none"> 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 <p>6 SCHEDULES AND ACTIVITY NETWORK</p> <p>7 PROJECT ORGANIZATION AND RESOURCES</p> <ul style="list-style-type: none"> a) Project organization b) Project Resources <p>8 NOTES</p> <p>9 APPENDIXES</p> <p>10 TO-DO-LIST</p> <ul style="list-style-type: none"> a) New Features b) Modifications c) Troubleshooting d) Validation/Regression Test <p>11 SAMPLE SCREEN SHOTS</p>
5	System Specification	<p>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. of the software system).</p> <p>Included are the following topics:</p> <ul style="list-style-type: none"> 1 SCOPE (System Specification) 2 DEFINITIONS, ABBREVIATIONS, AND ACRONYMS 3 REQUIREMENTS 4 QUALIFICATION PROVISIONS a) Demonstration

		<ul style="list-style-type: none"> b) Test c) Analysis d) Inspection e) Special Qualification Methods
		5 REQUIREMENTS TRACEABILITY
		6 NOTES
		<ul 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
		18 APPENDIX K - SYSTEM QUALITY FACTORS
		19 APPENDIX L - DESIGN AND CONSTRUCTION CONSTRAINTS
		20 APPENDIX M - PERSONNEL-RELATED REQUIREMENTS
		21 APPENDIX N - TRAINING-RELATED REQUIREMENTS
		22 APPENDIX O - LOGISTICS-RELATED REQUIREMENTS

		23 APPENDIX P - OTHER REQUIREMENTS 24 APPENDIX Q - PACKAGING REQUIREMENTS 25 APPENDIX R - PRECEDENCE AND CRITICALITY OF REQUIREMENTS
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> <ol style="list-style-type: none"> 1 SCOPE (System Specification) 2 DEFINITIONS, ABBREVIATIONS, AND ACRONYMS 3 REQUIREMENTS <ol style="list-style-type: none"> a) Interface Identification and Diagrams b) (Project unique identifier of interface) c) Terminal Device Interface (TDI) d) Precedence and Criticality of Requirements 4 QUALIFICATION PROVISIONS 5 REQUIREMENTS TRACEABILITY 6 NOTES 7 APPENDIXES
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> <ol style="list-style-type: none"> 1 SCOPE (System Specification) 2 DEFINITIONS, ABBREVIATIONS, AND ACRONYMS 3 REQUIREMENTS <ol 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

		<ul style="list-style-type: none"> 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 is fixed or an enhancement is made to the product.</p> <p>Included are the following topics:</p> <ul style="list-style-type: none"> 1 SCOPE (System Specification) 2 DEFINITIONS, ABBREVIATIONS, AND ACRONYMS 3 SYSTEM REQUIREMENTS <ul style="list-style-type: none"> a) Platform Configurations (Release Notes) b) Network Configurations (Release Notes)

		<p>4 PACKAGE CONTENTS</p> <ul style="list-style-type: none"> a) Command Line Interface Library b) Graphical Text User Interface Library <p>5 FIXED BUGS & ISSUES</p> <p>6 KNOWN BUGS & ISSUES</p> <p>7 NEW FEATURES</p> <p>8 APPLICATION NOTES</p> <ul style="list-style-type: none"> a) Installation Procedure b) User Interface Design c) Developer <p>9 PERFORMANCE TUNING</p> <p>10 ACCEPTANCE TESTING</p> <p>11 COMPONENT STATUS</p> <p>12 APPENDIXES</p> <ul 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> <p>1 SCOPE (System Specification)</p> <p>2 SOFTWARE SUMMARY</p> <ul style="list-style-type: none"> a) Software Application b) Software Inventory c) Software Environment 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 <p>2 Software Development Plan</p> <p>3 System Specification</p> <ul style="list-style-type: none"> a) APPENDIX A - REQUIRED STATES AND MODES b) APPENDIX B - SYSTEM CAPABILITY REQUIREMENTS c) APPENDIX C - SYSTEM EXTERNAL INTERFACE REQUIREMENTS

		<ul style="list-style-type: none"> 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 <p>4 Interface Requirements Specification</p> <p>5 Software Requirements Specification</p> <ul style="list-style-type: none"> a) APPENDIX A - Nature of System and Software <p>6 Release Notes</p> <ul style="list-style-type: none"> a) APPENDIX A - Baseline Host Computer Platforms b) APPENDIX B - API Relationship c) APPENDIX C - Deliverables <p>7 Software Users Manual</p>
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	A list of planned development for the code.

13	Use Cases	A list of steps, typically defining interactions between a role (known in Unified Modeling Language (UML) as an "actor") and a system, to achieve a goal. The actor can be a human, an external system, or time.
----	------------------	--

Draft

Draft

2 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)." A copy of the specification is available via a download link.
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>
Use Case	<p>Excerpt From Wikipedia, the free encyclopedia</p> <p>"In software and systems engineering, a use case is a list of action or event steps, typically defining the interactions between a role (known in the Unified Modeling Language as an actor) and a system, to achieve a goal. The actor can be a human, an external system, or time. In systems engineering, use cases are used at a higher level than within software engineering, often representing missions or stakeholder goals. The detailed requirements may then be captured in the Systems Modeling Language (SysML) or as contractual statements.</p> <p>Use case analysis is an important and valuable requirement analysis technique that has been widely used in modern software engineering since their formal introduction by Ivar Jacobson in 1992. Use case driven development is a key characteristic of many process models and frameworks such as ICONIX, the Unified Process (UP), the IBM Rational Unified Process (RUP), and the Oracle Unified Method (OUM). With its inherent iterative, incremental and evolutionary nature, use case also fits well for agile development."</p>

TERM	DEFINITION
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 ▪ UnixUtils ▪ GnuWin32 ▪ GNUWin II ▪ MinGW ▪ LBW: Linux Binaries on Windows requires Interix to be installed first."
VMware Fusion	<p>From Wikipedia, the free encyclopedia</p> <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</p>

TERM	DEFINITION
	<p>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>
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</p>

TERM	DEFINITION
	Character Set feature (e.g., Multinational Character Set) and support for 8-bit downloadable character sets."
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) ▪ 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	From Wikipedia, the free encyclopedia:

TERM	DEFINITION
	<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 Vandevor, 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 appearance of spurious lines streaking across the display.

Draft

3 REQUIREMENTS

This section shall be divided into the following paragraphs to specify the CSCI requirements, that is, those characteristics of the CSCI that are conditions for its acceptance. CSCI requirements are software requirements generated to satisfy the system requirements allocated to this CSCI. Each requirement shall be assigned a project-unique identifier to support testing and traceability and shall be stated in such a way that an objective test can be defined for it. Each requirement shall be annotated with associated qualification method(s) (see section 4) and traceability to system (or subsystem, if applicable) requirements (see section 5.a) if not provided in those sections. The degree of detail to be provided shall be guided by the following rule: Include those characteristics of the CSCI that are conditions for CSCI acceptance; defer to design descriptions those characteristics that the acquirer is willing to leave up to the developer. If there are no requirements in a given paragraph, the paragraph shall so state. If a given requirement fits into more than one paragraph, it may be stated once and referenced from the other paragraphs.

3.1 Required States and Modes

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

1 *Required States* (on page 49)

2 *Required Modes* (on page 51)

3.1.1 Required States

3.1.1.1 Pre-Startup State

- 1** In the event that the platform has not been prepared for use by the System Operator, the System Administrator shall perform the following actions:

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

3.1.1.2 Startup State

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

Excerpt From Wikipedia, the free encyclopedia

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

3.1.1.3 Operating State

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

- 2 The computer shall interact with the operator to load, execute and terminate operator designated system and application software.

3.1.1.4 Shutdown State

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

3.1.2 Required Modes

3.1.2.1 Video Adapter Modes

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

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

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

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

3.1.2.1.1 Text Mode

Excerpt From Wikipedia, the free encyclopedia

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

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

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

3.1.2.1.2 Pixel Mode

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

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

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

Draft

3.1.2.2 User Interface Modes

3.1.2.2.1 Command Line Mode

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

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

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

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

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

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

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

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

Usage

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

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

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

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

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

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

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

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

3.1.2.2.2 Graphical Mode

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

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

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

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

3.1.2.2.2.1 wxPython

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

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

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

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

3.1.3 Required Variants

3.1.3.1 Release Variants

Source Code components from all Python version-specific Site-Package and Developer-Sandbox variants provide applications with the same functionality and API.

However, there are internal differences based on the Python language version and based on their role in either Toolkit building block development or in Toolkit user application development. Consequently, the components are NOT designed to be intermixed and should NOT be blindly copied from one Python version or variant to another.

Each release of the TeamSTARS "tsWxGTUI_PyVx" Toolkit includes the following application-specific variants:

1 DEVELOPER_SANDBOXES

Each sandbox variant uses a multi-layered hierarchical packaging organization and import mechanism. Its import mechanism uses programmer defined package hierarchy relationships and dynamic path generation to facilitate development activities and isolate them from interfering with previously installed products.

Components of the sandbox are organized into first-level package subdirectories ("tsLibCLI", "tsLibGUI" and "tsToolsCLI" etc.) subdirectories based on their common functional relationship. This permits packages to be imported once and to share or substitute components when appropriate. Similarly, second-level package component (building block such as "tsLoggerPkg", tool and utility module) source code is typically organized into "src" and "test" package subdirectories based on their application or quality assurance roles.

The hierarchical directory components are inter-connected via a set of "__init__.py" modules. At times, this complexity makes troubleshooting more difficult.

Each user application or Toolkit building-block module must import Toolkit modules via its explicit sandbox path (such as "from tsLibCLI import tsLogger" or "import tsLibCLI; import tsLogger").

Typically, each Python language generation has its own sandbox:

- a) The *TeamSTARS* "tsWxGTUI_Py2x" Toolkit is available for use with the second generation Python programming language , Python 2.0.0 - 2.7.10.
- b) The *TeamSTARS* "tsWxGTUI_Py3x" Toolkit is available for use with the third generation Python programming language , Python 3.0.0 - 3.4.3.

2 SITE-PACKAGES

Each site-package variant uses a single-layered packaging organization with subdirectories ("tsLibCLI", "tsLibGUI" and "tsToolsCLI" etc.) based on their common functional relationship. The components within their container package are connected via an empty "__init__.py" module.

Each user application or Toolkit building-block module must import Toolkit modules via its explicit site-package path (such as "from tsWxGTUI_Py2x.tsLibCLI import tsLogger" or "from tsWxGTUI_Py3x.tsLibCLI import tsLogger")

Typically, each Python language generation has its own site-package:

- a) The *TeamSTARS* "tsWxGTUI_Py2x" Toolkit is available for use with the second generation Python programming language , Python 2.0.0 - 2.7.9.
- b) The *TeamSTARS* "tsWxGTUI_Py3x" Toolkit is available for use with the third generation Python programming language , Python 3.0.0 - 3.4.3.

3.1.3.2 Software Life Cycle Stage

The *TeamSTARS* "tsWxGTUI_PyVx" Toolkit, like other computer software products, is evolving through a series of life cycle stages. It is currently in its pre-alpha stage at which its developers can demonstrate the look and feel of its Command Line Interface and Graphical-style User Interface (the character-mode emulation of the pixel-mode "wxPython").

Stage	Excerpts from Wikipedia, the free encyclopedia (see: <i>Software Release Life Cycle</i> (http://en.wikipedia.org/wiki/Software_release_life_cycle))
Pre-Alpha	Pre-alpha refers to all activities performed during the software project before testing. These activities can include requirements analysis, software design, software development, and unit testing. In typical open source development, there are several types of pre-alpha versions. Milestone versions include specific sets of functions and are released as soon as the functionality is complete.
Alpha	<p>The alpha phase of the release life cycle is the first phase to begin software testing (alpha is the first letter of the Greek alphabet, used as the number 1). In this phase, developers generally test the software using white-box techniques. Additional validation is then performed using black-box or gray-box techniques, by another testing team. Moving to black-box testing inside the organization is known as alpha release.[2]</p> <p>Alpha software can be unstable and could cause crashes or data loss. External availability of alpha software is uncommon in proprietary software. However, open source software, in particular, often have publicly available alpha versions, often distributed as the raw source code of the software. The alpha phase usually ends with a feature freeze, indicating that no more features will be added to the software. At this time, the software is said to be feature complete."</p>

Beta	<p>Beta, named after the second letter of the Greek alphabet, is the software development phase following alpha. It generally begins when the software is feature complete. Software in the beta phase will generally have many more bugs in it than completed software, as well as speed/performance issues and may still cause crashes or data loss. The focus of beta testing is reducing impacts to users, often incorporating usability testing. The process of delivering a beta version to the users is called beta release and this is typically the first time that the software is available outside of the organization that developed it.</p> <p>The users of a beta version are called beta testers. They are usually customers or prospective customers of the organization that develops the software, willing to test the software without charge, often receiving the final software free of charge or for a reduced price. Beta version software is often useful for demonstrations and previews within an organization and to prospective customers. Some developers refer to this stage as a preview, prototype, technical preview / technology preview (TP), or early access. Some software is kept in perpetual beta—where new features and functionality are continually added to the software without establishing a firm "final" release.</p>
Release Candidate	<p>A release candidate (RC) is a beta version with potential to be a final product, which is ready to release unless significant bugs emerge. In this stage of product stabilization, all product features have been designed, coded and tested through one or more beta cycles with no known showstopper-class bug. A release is called code complete when the development team agrees that no entirely new source code will be added to this release. There could still be source code changes to fix defects, changes to documentation and data files, and peripheral code for test cases or utilities. Beta testers, if privately selected, will often be credited for using the release candidate as though it were a finished product. Beta testing is conducted in a client's or customer's location and to test the software from a user's perspective.</p>

Release to Manufacturing	<p>The term "release to manufacturing", also known as "going gold", is a term used when a software product is ready to be delivered or provided to the customer. This build may be digitally signed, allowing the end user to verify the integrity and authenticity of the software purchase. A copy of the RTM build known as the "gold master" or GM is sent for mass duplication. RTM precedes general availability (GA), when the product is released to the public.</p> <p>It is typically used in certain retail mass-production software contexts—as opposed to a specialized software production or project in a commercial or government production and distribution—where the software is sold as part of a bundle in a related computer hardware sale and typically where the software and related hardware is ultimately to be available and sold on mass/public basis at retail stores to indicate that the software has met a defined quality level and is ready for mass retail distribution. RTM could also mean in other contexts that the software has been delivered or released to a client or customer for installation or distribution to the related hardware end user computers or machines. The term does not define the delivery mechanism or volume; it only states that the quality is sufficient for mass distribution. The deliverable from the engineering organization is frequently in the form of a golden master media used for duplication or to produce the image for the web.</p>
General Availability	<p>General availability (GA) is the marketing stage at which all necessary commercialization activities have been completed and a software product is available for purchase, depending, however, on language, region, electronic vs. media availability.[8]</p> <p>Commercialization activities could include security and compliance tests, as well as localization and world wide availability. The time between RTM and GA can be from a week to months in some cases before a generally available release can be declared because of the time needed to complete all commercialization activities required by GA. At this stage, the software has "gone live".</p>

3.2 CSCI Capability Requirements

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

3.2.1 (CSCI Capability Template)

This paragraph shall identify a required CSCI capability and shall itemize the requirements associated with the capability. If the capability can be more clearly specified by dividing it into constituent capabilities, the constituent capabilities shall be specified in subparagraphs. The requirements shall specify required behavior of the CSCI and shall include applicable parameters, such as response times, throughput times, other timing constraints, sequencing, accuracy, capacities (how much/how many), priorities, continuous operation require and allowable deviation based on operating conditions. The requirements shall include, as applicable, required behavior under unexpected, unallowed, or "out of bounds" conditions, requirements for error handling, and any provisions to be incorporated into the CSCI to provide continuity of operations in the event of emergency. Paragraph 3.3.x of this DID provides a list of topics to be considered when specifying requirements regarding inputs the CSCI must accept and outputs it must produce.

3.2.2 Command Line Interface Capability

This paragraph shall identify a required CSCI capability and shall itemize the requirements associated with the capability. If the capability can be more clearly specified by dividing it into constituent capabilities, the constituent capabilities shall be specified in subparagraphs. The requirements shall specify required behavior of the CSCI and shall include applicable parameters, such as response times, throughput times, other timing constraints, sequencing, accuracy, capacities (how much/how many), priorities, continuous operation require and allowable deviation based on operating conditions. The requirements shall include, as applicable, required behavior under unexpected, unallowed, or "out of bounds" conditions, requirements for error handling, and any provisions to be incorporated into the CSCI to provide continuity of operations in the event of emergency. Paragraph 3.3.x of this DID provides a list of topics to be considered when specifying requirements regarding inputs the CSCI must accept and outputs it must produce.

This capability 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.

- ***CLI Terminal Interface Capability*** (on page 62)
- ***"tsLibCLI" Capability*** (on page 65)
- ***"tsToolsCLI" Capability*** (on page 63)

3.2.2.1 CLI Terminal Interface Capability

The "tsWXTGUI" Toolkit software shall provide the following:

- 1 Hardware** - Terminal for input from operator (such as an electronic keyboard) and hardware for output to operator (such as an electronic display or printer).

- 2 Software** - Shell application (typically provided by the operating system) to receive, interpret and process command input from operator that cause actions which gather, process, format and output information to operator. It is a command processor that's typically run in a one dimensional text window (on the line displaying the command prompt), allowing the user to type commands which cause actions. It can also read commands from a file, called a script.

Command Line Interface mode input/output streams:

- **stdin** - POSIX-type standard input stream of one or more characters from an operator's keyboard or redirected input from a file or pipe (connection with the output stream from another running program).
- **stdout** - POSIX-type standard output stream of one or more characters to an operator's display or redirected output to a file or pipe (connection with the input stream of another running program).
- **stderr** - POSIX-type standard error output stream of one or more characters to an operator's display, system administrator's display or redirected output to a file or pipe (connection with the input stream of another running program).

Upon successfully launching the Graphical-style User Interface mode, and until terminating it, the following input/output streams become available:

- **stdscr** - Curses-type standard screen output stream of one or more characters to an operator's display. Before an application can use this stream, the application must initialize curses. This is done by calling the curses module's `initscr()` function, which will determine the terminal type, send any required setup codes to the terminal, and create various internal data structures. If successful, `initscr()` returns a window object representing the entire screen; this is usually called `stdscr`, after the name of the corresponding C variable.
- **stdscr.getch** - The `getch()` method returns an integer; if it's between 0 and 255, it represents the ASCII code of the key pressed. Values greater than 255 are special keys such as Page Up, Home, or the cursor keys. A value of -1 represents an input timeout. You can compare the value returned to constants such as `curses.KEY_PPAGE`, `curses.KEY_HOME`, or `curses.KEY_LEFT`. You can also compare the value returned to constants such as `curses.BUTTON1_PRESSED`, `curses.BUTTON1_RELEASED`, `curses.BUTTON1_CLICKED`, `curses.BUTTON1_DOUBLE_CLICKED` and `curses.BUTTON1_TRIPLE_CLICKED`.
- **stdscr.getstr** - The `getstr()` method retrieves an entire string. It isn't used very often, because its functionality is quite limited; the only editing keys available are the backspace key and the Enter key, which terminates the string. It can optionally be limited to a fixed number of characters.

3.2.2.2 "tsToolsCLI" Capability

The "tsWXTGUI" Toolkit software shall provide the following:

- 1 tsLinesOfCodeProjectMetrics** - Python application program, with a Command Line Interface (CLI), that generates reports of software project progress and the estimated cost (or contributed value) of the project when it is finally completed.
 - a) It scans an operator designated file directory tree containing the source files, in one or more programming language specific formats (such as Ada, Assembler, C/C++, Cobol, Fortran, PL/M, Python, Text, and various command line shells).

For each file, it accumulates and reports the total number of code lines, blank/comment lines, words and characters.

For each programming language format, it accumulates and reports a summary of details of the associated source files.

For the entire set of source files, it accumulates and reports a summary of details.

It uses the summary of the entire set of source files to derive, analyze, estimate and report metrics for the software development project (such as labor, cost, schedule and lines of code per day productivity).

- b) It supports one or more of the parser module(s) available in the Python version(s) supported by the application.

"argparse" (introduced with Python 2.7.0)

"optparse" (introduced with Python 2.3.0)

"getopt" (introduced with Python 1.6.0)

When used with Python 2.7 or Python 3.2, the "tsOperatorSettings.py" module (a "tsLibCLI" component of the "tsWxGTUI_PyVx" Toolkit) supports the current and legacy Python version specific parser module(s) as an experimental and educational opportunity.

However, when one seeks to back port applications to Python 2.0-2.6 or Python 3.0-3.1, the "tsOperatorSettings.py" module must be stripped of unsupported Python version specific parser modules in order to prevent a program trap which will block the application from running.

- 2 tsPlatformQuery** - Python application program, with a Command Line Interface (CLI), that captures current hardware and software information about the run time environment for the user process.

(a) Host processor hardware support includes various releases of x86, PowerPC, SPARC.

(b) Host operating system software support includes various releases of Cygwin, Linux ('Debian', 'Fedora', 'mandriva', 'redhat', 'Scientific / Centos', 'SuSE'), Mac OS X, Microsoft Windows ('XP', 'Vista', '7', '8', '8.1') and Unix ('FreeBSD / PC-BSD', 'IRIX', 'OpenIndiana', 'Solaris / SunOS').

(c) Host virtual machine software support includes various releases of Java and Python.

- 3 tsStripComments** - Python application program, with a Command Line Interface (CLI), that transforms an annotated, development version of a directory of sub-directories and Python source files into an unannotated copy.

The copy is intended to conserve storage space when installed in an embedded system. The transformation involves stripping comments and doc strings by de-tokenizing a tokenized version of each Python source file. Non-Python files are trimmed of trailing whitespace.

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

Output from fixed format FORTRAN (F77) code is NOT corrected to ensure that each statement first character begins in column 7 and that each ampersand ("&") continuation character is in column 6.

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

- 6 tsTreeTrimLines** - Python application program, with a Command Line Interface (CLI), that copies the contents of a source directory to a target directory after stripping superfuous white space (blanks) from end of each line.

3.2.2.3 "tsLibCLI" Capability

The "tsWXTGUI" Toolkit software shall provide the following:

- 1 tsApplicationPkg** - Python base class to initialize and configure the application program launched by an operator. It enables an application launched via a Command Line Interface (CLI) to initialize, configure and use the same character-mode terminal with a Graphical-style User Interface (GUI).

 - a) Input provided on the command line by an operator. The command line uses a Unix-style, free-format to promote future enhancement and on-going maintenance.
 - b) Input provided in the parameter list of the application's invocation of the class instantiation. The parameter list uses a Python-style free-format to promote future enhancement and on-going maintenance.

- 2 tsCommandLineEnvPkg** - Python class to initialize and configure the application program launched by an operator. It delivers those keyword-value pair options and positional arguments specified by the application, in its invocation parameter list. It wraps the Command Line Interface application with exception handlers to control exit codes and messages that may be used to co-ordinate other application programs..
- 3 tsCommandLineInterfacePkg** - Python class establishes methods that prompt or re-prompt the operator for input, validate that the operator has supplied the expected number of inputs and that each is of the expected type.
- 4 tsCxGlobalsPkg** - Python class to provide a theme-based centralized mechanism for modifying/restoring those configuration constants that can be interrogated at runtime by those software components having a "need-to-know". The intent being to avoid subsequent searches to locate and modify or restore a constant appropriate to the current configuration, users and their activities.
- 5 tsExceptionPkg** - Python class to define and handle error exceptions. It maps various error exceptions into an 8-bit exit code and message that can be used to coordinate the actions of a set of shell scripts.
- 6 tsLoggerPkg** - Python class to define and handle event message timestamping, formatting and output. It also supports logging of assert and check case results.

It supports the following message severity levels: NOTSET (lowest priority), DEBUG, INFO, NOTICE, WARNING, ALERT, ERROR, CRITICAL, EMERGENCY (highest priority) and PRIVATE.

It defines and handles event message processing associated with the following: wxPython/wxWidget exceptions: wxASSERT, wxASSERT_MSG, wxCHECK, wxCHECK2, wxCHECK2_MSG, wxCHECK_MSG, wxCHECK_RET, wxFAIL, wxFAIL_COND_MSG, wxFAIL_MSG and wxTRAP.

- 7 tsOperatorSettingsParserPkg** - Class to parse command line options and arguments.

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

- a) "argparse" (introduced with Python 2.7.0)
- b) "optparse" (introduced with Python 2.3.0)
- c) "getopt" (introduced with Python 1.6.0)

When used with Python 2.7 or Python 3.2, the "tsOperatorSettings.py" module (a "tsLibCLI" component of the "tsWxGTUI_PyVx" Toolkit) supports the current and legacy Python version specific parser module(s) as an experimental and educational opportunity.

However, when one seeks to back port applications to Python 2.0-2.6 or Python 3.0-3.1, the "tsOperatorSettings.py" module must be stripped of unsupported Python version specific parser modules in order to prevent a program trap which will block the application from running.

- 8 tsPlatformRunTimeEnvironmentPkg** - Class to capture current hardware and software information about the run time environment for the user process.
- 9 tsReportUtilityPkg**- Class defining methods used to format information such as time, date, data size, elapsed time and nested dictionary contents.
- 10 tsSysCommandsPkg** - Class definition and methods for issuing shell commands to the host operating system.

3.2.3 Graphical User Interface Capability

This paragraph shall identify a required CSCI capability and shall itemize the requirements associated with the capability. If the capability can be more clearly specified by dividing it into constituent capabilities, the constituent capabilities shall be specified in subparagraphs. The requirements shall specify required behavior of the CSCI and shall include applicable parameters, such as response times, throughput times, other timing constraints, sequencing, accuracy, capacities (how much/how many), priorities, continuous operation require and allowable deviabased on operating conditions. The requirements shall include, as applicable, required behavior under unexpected, unallowed, or "out of bounds" conditions, requirements for error handling, and any provisions to be incorporated into the CSCI to provide continuity of operations in the event of emer Paragraph 3.3.x of this DID provides a list of topics to be considered when specifying requirements regarding inputs the CSCI must accept and outputs it must produce.

This capability 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.

- **GUI Terminal Interface Capability** (on page 66)
- **"tsLibGUI" Capability** (on page 67)
- **"tsToolsGUI" Capability** (on page 67)

3.2.3.1 GUI Terminal Interface Capability

The "tsWXTGUI" Toolkit software shall provide the following:

- 1 Hardware** - Terminal for input from operator (such as an electronic keyboard, mouse, trackball, touchpad or touchscreen) and hardware for output to operator (such as an electronic display or printer).

- 2 Software** - Shell application (typically provided by the operating system) to receive, interpret and process command input from operator that cause actions which gather, process, format and output information to operator. It is a command processor that's typically run in a two dimensional graphical screen, allowing the user to "click" on objects (such as buttons, check boxes, radio buttons, windows and dialogs) and type commands which cause actions. It can also read commands from a file, called a script.
- **Platform Dependent API** - Software components that provide an Application Programming Interface that is uniquely customized for the platform's physical hardware and operating system software (such as Linux, Mac OS X and Windows).
 - **Terminal Independent API** - Software components of "nCurses" that provide a common Application Programming Interface regardless of a platform's physical hardware and operating system software.
 - **wxPython API** - Software components of "tsWxGraphicalTestUserInterface" that provide a subset of the "wxPython" Application Programming Interface. It includes class definition and methods to initialize, configure, create GUI objects and shutdown the platform's nCurses-based terminal interface.

3.2.3.2 "tsToolsGUI" Capability

There are currently no GUI applications.

3.2.3.3 "tsLibGUI" Capability

The "tsWXTGUI" Toolkit software shall provide the following:

- **Desktop API** (on page 68) - Describes the host operating system-style Graphical User Interface features (such as multiple top-level windows, task bar and redirected stderr and stdout displays) that the Software Engineer can include for input and output interactions with the System OperatorSystem Operator.
- **High-Level Widget API** (on page 69) - Identifies the wxPython-style Graphical User Interface features (such as frames, dialogs and buttons) that the Software Engineer can include for input and output interactions with the System Operator.
- **Event API** (see "*Event API (Draft)*" on page 73) - Describes the basic and additional wxPython-style Graphical User Interface features used for sending, receiving and handling event notifications (such as clock tick, button clicked and shift in window focus).
- **Low-Level Widget API** (on page 91) - Identifies the basic nCurses-style Graphical User Interface features that the Software Engineer can use to construct High Level Widgets that customize input and output interactions with the local or remote terminal.
- **System Preference API** (see "*System Preferences API (Draft)*" on page 92) - Describes the wxPython-style Graphical User Interface features (such as Symbolic Constants, Foreground / Background Colors, Default Styles / Themes and utility functions) that the Software Engineer can use to standardize the look and feel of input and output interactions with the System Operator.

3.2.3.3.1 Desktop API

from http://en.wikipedia.org/wiki/Multiple_document_interface:

"Graphical computer applications with a multiple document interface (MDI) are those whose windows reside under a single parent window (usually except for modal windows), as opposed to all windows being separate from each other (single document interface). Such systems often allow child windows to embed other windows inside them as well, creating complex nested hierarchies. In the usability community, there has been much debate about which interface type is preferable. Generally, SDI is seen as more useful in cases where users work with more than one application. Software companies have used both interfaces with mixed responses. For example, Microsoft changed its Office applications from SDI to MDI mode and then back to SDI, although the degree of implementation varies from one component to another.

The disadvantage of MDI usually cited is the lack of information about the currently opened windows: In order to view a list of windows open in MDI applications, the user typically has to select a specific menu ("window list" or something similar), if this option is available at all. With an SDI application, the window manager's task bar or task manager (if any) can display the currently opened windows. In recent years, applications have increasingly added "task-bars" and "tabs" to show the currently opened windows in an MDI application, which has made this criticism somewhat obsolete. Some people call this interface "tabbed document interface" (TDI). When tabs are used to manage windows, individual ones usually cannot be resized or used simultaneously."

Multiple Document Interface capabilities currently include:

- 1 Screens** - A display (shown with black or white background) of a physical terminal device or a GUI object representing an emulated terminal whose size, position and font can (usually) be changed by the user. It usually has thick borders and a title bar, and can optionally contain a menu bar, toolbar and status bar. A screen can contain one or more frames and dialogs within which may be those GUI objects that are not frames or dialogs.
- 2 Multi-Frame Environment** - A non-wxPython feature consisting of one or more Frames (shown with blue background) and Dialogs (shown with cyan background) associated with the GUI application.
- 3 Redirected Output** - A non-wxPython feature consisting of a Frame (shown with green background) containing one or more lines of text messages. The message text is produced by the application's use of print or write statements with output normally intended for the "stderr" and "stdout" devices. The GUI toolkit intercepts the output and redirects it to a reserved frame on the display screen after annotating it with a date and time stamp.
- 4 Task Bar** - A non-wxPython feature consisting of a Frame (shown with black background) that monitors and controls the application tasks. The top-right line identifies the name of the application executable. The bottom-right line contains a character-mode symbol that appears to rotate. The rotation denotes progress. The middle line(s) contain buttons for each of the application's Frame and Dialog "tasks". If a mouse button is clicked when the cursor is over a button, an associated function or method will be initiated. The function or method will send a signal to notify other GUI objects of the event. The event will raise the focus of the selected task so that its frame or dialog becomes the top-most overlays.

NOTES:

The Desktop API includes a MultiFrame Environment that establishes the following Top Level Windows:

- 1) The Application Programmer designated Top Level Frame and Dialog, at top of screen.
 - 2) The Redirected I/O Frame provides the operator with the most recent stderr, stdout and print messages, in middle of screen.
 - 3) The Task Bar Frame provides the operator with a list of currently open frame and dialog windows from which to click on and change the focus, at bottom of screen.
-

3.2.3.3.2 High-Level Widget API

NOTES:

- 1) As a character-mode GUI-style toolkit, this product does not support those "wxPython" features associated with graphical elements such as bit images, icons, proportional-spaced fonts or HTML and XML text markup. The current release supports the porting of "wxPython" 2.8.9.2 application(s) to platforms running Python 2.5.x, 2.6.x, 2.7.x, 3.1.x and 3.2.x.
 - 2) Technical and resource issues drive the development effort. Initially, development focussed on establishing the feasibility of emulating core components of the the "wxPython" and associated "wxWidgets" API. Development then iteratively seeks to establish usability-enhancing components and to then to identify and resolve any appearance, behavior and API-conformance issues.
 - 3) See the unpublished "**tsWxGTUI_PyVx**" **Vol. 7 - Design Notes** (it was created only for personal use) for the identification and an overview of those currently implemented class modules, ones currently under construction and ones for which development applicability and/or commitments are still TBD.
-

The High-Level Widget API identifies the basic wxPython-style Graphical User Interface features that the Software Engineer can include for input and output interactions with the System Operator. It includes such widgets as the following:

- 1 Button** - GUI object that may contain text or character-mode icons. If a mouse button is clicked when the cursor is over the object, an associated function or method will be initiated. The function or method will send a signal to notify other GUI objects of the event. Buttons are clickable windows that trigger an associated event (such as start, pause, resume or terminate an operation).
- 2 Carat** - GUI object that may contain a set of special character-mode symbols, usually including a solid rectangle or a blinking underline character, showing the position where the typed text will appear.
- 3 Check Box** - GUI object that may contain text or character-mode icons. If a mouse button is clicked when the cursor is over the object, it initiates an associated function or method. The function or method will toggle the check box to the next "ON", "OFF" or "TRI-STATE" value. The function or method will send a signal to notify other GUI objects of the event. Checkboxes are buttons to toggle the enabled or disabled state of an associated feature (such as start or stop logging)
- 4 Cursor** - A special character-mode symbol, usually a solid rectangle or a blinking underline character, that signifies where the next character will be displayed on the screen.

- 5 Dialog** - A GUI object with a title bar and sometimes a system menu, which can be moved around the screen. It can contain controls and other GUI objects and is often used to allow the user to make some choice or to answer a question. Dialogs are pop-up windows associated with an application task's menu bar (such as an input field for an operator's search criteria and an output field for a list of candidate WEB sites) that will be terminated upon completion.
- 6 Frame** - GUI object whose size and position can (usually) be changed by the user. It usually has thick borders and a title bar, and can optionally contain a menu bar, toolbar and status bar. A frame can contain any GUI object that is not a frame or dialog. Frames are windows associated with an application task (such as an internet WEB Browser) that can be minimized into an icon, expanded to full screen or terminated upon operator demand.
- 7 Gauge** - GUI object whose horizontal or vertical bar shows a quantity (often time). It supports two working modes: determinate and indeterminate progress. First is the usual working mode while the second can be used when the program is doing some processing but you don't know how much progress is being done. In this case, you can periodically call the Pulse function to make the progress bar switch to indeterminate mode (graphically it's usually a set of blocks which move or bounce in the bar control). Gauges are used to indicate progress of an associated operation (scale with range such as 0% to %100%).
- 8 Menu** - GUI object that features a popup (or pull down) list of items, one of which may be selected before the menu goes away (clicking elsewhere dismisses the menu). It may be used to construct either menu bars or popup menus.
- 9 Menu Bar** - GUI object that features a series of menus accessible from the top of a frame. Each operator selectable entry triggers an associated event (such as create a file).
- 10 Panel** - GUI object that features a window on which controls are placed. Panels are usually placed within a frame. Its main feature over its parent window class is code for handling child windows and TAB traversal. Panels are container windows for other Lower Level GUI Objects.
- 11 Radio Box** - GUI objects that are used to select one of number of mutually exclusive choices. A radio box is displayed as a vertical column or horizontal row of labelled radio buttons. Radio Boxes are a collection of Buttons associated with the same group (such a AM or FM band) that are interdependent such that any one can become activated after the others simultaneously become deactivated.
- 12 Radio Button** - GUI object that may contain text or character-mode icons. If a mouse button is clicked when the cursor is over the object, it initiates an associated function or method. The function or method will turn the associated radio button "ON" and turn "OFF" all other radio buttons within the associated radio box. The function or method will send a signal to notify other objects of the event. Radio Buttons are used to activate one of the associated features (such as broadcast channel selection) after the other features associated with the same Radio Box group (such a AM or FM band) have become deactivated.
- 13 ScrollBar** - GUI object that includes a horizontal or vertical ScrollBarGauge between two ScrollBarButtons. The operator uses it to re-position (pan) the contents of an associated ScrolledText window.
- 14 ScrollBarButton** - GUI object that includes one arrow symbol ("<", ">", "^" or "V") that indicates the horizontal or vertical direction of scrolling. When the operator clicks on a ScrollBarButton, the contents of the associated ScrolledText window moves.

- A single Left Click on one of the two horizontal ScrollBarButtons moves the text by one column. A rapid double Left Click on one of the two horizontal ScrollBarButtons moves the ScrolledText by one page (the horizontal column width). A single Right Click on one of the two horizontal ScrollBarButtons moves the text to the appropriate horizontally end point (either left/right most column).
- A single Left Click on one of the two vertical ScrollBarButtons moves the text by one row. A rapid double Left Click on one of the two vertical ScrollBarButtons moves the ScrolledText by one page (the vertical row height). A single Right Click on one of the two vertical ScrollBarButtons moves the text to the appropriate vertical end point (top/bottom most row).

15 ScrollBarGauge - GUI object located between two ScrollBarButtons, contains a bar graph that indicates the position and size of the displayed text relative to the non-displayed text.

- When the bar graph is empty (blank), there is no text available to be displayed.
- When it is completely filled, all of the available text is displayed.
- When it is partially filled, the starting point of the graph displays the starting point of the displayed text relative to the available text. The size of the filled graph displays the size of the displayed text relative to the available text.
- When the operator single Left Clicks on a ScrollBarGauge between its two associated ScrollBarButtons, the contents of the ScrolledText window moves in proportion to the difference between the click and the associated text end positions.

16 Scrolled - An application-independent GUI object base class for ScrolledWindow. It lays out the position and size of one ScrolledText window and one or two ScrollBars each with their associated ScrollBarGauge and pair of ScrollBarButtons. It enables an operator to select the columns and or rows of text to be fit and displayed within the available peep hole viewing area.

17 ScrolledText - GUI object that allows one or more lines of text to be appended to a retained list. In conjunction with one or two associated pairs of ScrollBarButtons, it enables an operator to select the columns and or rows of text to be fit and displayed within the available peep hole viewing area.

18 ScrolledWindow - GUI object that instantiates an application-specific instance of Scrolled to lay out the position and size of one ScrolledText window and one or two ScrollBars each with their associated ScrollBarGauge and pair of ScrollBarButtons. It enables an operator to select the columns and or rows of text to be fit and displayed within the available peep hole viewing area.

19 Splash Screen - GUI object that simulates a pixel-mode bitmap image which is displayed upon application startup. It may contain text or character-mode icons. It features a window with a thin border and text describing the application. The bitmap-like display may be created from Panel, BoxSizer, GridSizer and TextCtrl widgets. It is created and shown during application initialization, before the application's own window. It either explicitly destroys itself or disappears after a it time-out.

20 Status Bar - GUI object that feature a narrow window that can be placed along the bottom of a frame to give small amounts of status information. The object can contain one or more fields, one or more of which can be variable length according to the size of the window.

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

22 Text Ctl - GUI object that features a text control which allows text to be displayed and edited. It may be single line or multi-line. The text may be indented, line-wrapped and scrolled to fit the available display area.

23 Tool Bar (Future) - GUI object that features a series of tool entries accessible from the top of a frame. Each operator selectable entry triggers an associated event that starts the selected tool in a new Frame.

NOTES:

1) Event Handling support currently associates mouse clicks with the enclosing GUI object that is not obscured by overlaying GUI objects. The toolkit generates an event notification and dispatches it to the event handler designated by the application. It will dispatch unhandled event notifications to a default handler.

2) Keyboard Input Events are detected. The raw characters are echoed but are NOT currently forwarded to the window object having focus.

3) Mouse Input Events are detected. The x-y coordinates and button state are echoed. Single Left Clicks, Double Left Click and Right Clicks are forwarded to the window object positioned to receive and process the event. More complex GUI event detection, analysis and processing is not yet supported. Platform-specific vt100/vt220 terminal emulators do NOT conform to the xterm/xterm-color mouse click event protocol recognized by nCurses. Instead of a single mouse click event, the vt100/vt220 terminal emulators generate a string of escape sequence events. It is unknown if the "tsWxGTUI_PyVx" Toolkit can develop logic to synthesize an xterm/xterm-color mouse click event from the string of vt100/vt220 escape sequence events.

4) Automatic layout support is currently provided by the BoxSizer and GridSizer widgets or by combinations of them. More complex GUI object positioning and sizing is not yet supported.

3.2.3.3.3 Event API (Draft)

from http://docs.wxwidgets.org/trunk/overview_events.html

Introduction to Events

Like with all the other GUI frameworks, the control of flow in wxWidgets applications is event-based: the program normally performs most of its actions in response to the events generated by the user. These events can be triggered by using the input devices (such as keyboard, mouse, joystick) directly or, more commonly, by a standard control which synthesizes such input events into higher level events: for example, a wxButton can generate a click event when the user presses the left mouse button on it and then releases it without pressing Esc in the meanwhile. There are also events which don't directly correspond to the user actions, such as wxTimerEvent or wxSocketEvent.

But in all cases wxWidgets represents these events in a uniform way and allows you to handle them in the same way wherever they originate from. And while the events are normally generated by wxWidgets itself, you can also do this, which is especially useful when using custom events (see Custom Event Summary).

To be more precise, each event is described by:

Event type: this is simply a value of type wxEventType which uniquely identifies the type of the event. For example, clicking on a button, selecting an item from a list box and pressing a key on the keyboard all generate events with different event types.

Event class carried by the event: each event has some information associated with it and this data is represented by an object of a class derived from wxEvent. Events of different types can use the same event class, for example both button click and listbox selection events use wxCommandEvent class (as do all the other simple control events), but the key press event uses wxKeyEvent as the information associated with it is different.

Event source: wxEvent stores the object which generated the event and, for windows, its identifier (see Window Identifiers). As it is common to have more than one object generating events of the same type (e.g. a typical window contains several buttons, all generating the same button click event), checking the event source object or its id allows to distinguish between them.

Event Handling

There are two principal ways to handle events in wxWidgets. One of them uses event table macros and allows you to define the binding between events and their handlers only statically, i.e., during program compilation. The other one uses wxEvtHandler::Bind<>() call and can be used to bind and unbind, the handlers dynamically, i.e. during run-time depending on some conditions. It also allows the direct binding of events to:

A handler method in another object.

An ordinary function like a static method or a global function.

An arbitrary functor like boost::function<>.

The static event tables can only handle events in the object where they are defined so using `Bind<>()` is more flexible than using the event tables. On the other hand, event tables are more succinct and centralize all event handler bindings in one place. You can either choose a single approach that you find preferable or freely combine both methods in your program in different classes or even in one and the same class, although this is probably sufficiently confusing to be a bad idea.

Also notice that most of the existing wxWidgets tutorials and discussions use the event tables because they historically preceded the apparition of dynamic event handling in wxWidgets. But this absolutely doesn't mean that using the event tables is the preferred way: handling events dynamically is better in several aspects and you should strongly consider doing it if you are just starting with wxWidgets. On the other hand, you still need to know about the event tables if only because you are going to see them in many samples and examples.

So before you make the choice between static event tables and dynamically connecting the event handlers, let us discuss these two ways in more detail. In the next section we provide a short introduction to handling the events using the event tables. Please see [Dynamic Event Handling](#) for the discussion of `Bind<>()`.

Event Handling with Event Tables

To use an event table you must first decide in which class you wish to handle the events. The only requirement imposed by wxWidgets is that this class must derive from `wxEvtHandler` and so, considering that `wxWindow` derives from it, any classes representing windows can handle events. Simple events such as menu commands are usually processed at the level of a top-level window containing the menu, so let's suppose that you need to handle some events in `MyFrame` class deriving from `wxFrame`.

First define one or more event handlers. They are just simple methods of the class that take as a parameter a reference to an object of a `wxEvent`-derived class and have no return value (any return information is passed via the argument, which is why it is non-const). You also need to insert a macro

```
wxDECLARE_EVENT_TABLE()
```

somewhere in the class declaration. It doesn't matter where it appears but it's customary to put it at the end because the macro changes the access type internally so it's safest if nothing follows it. The full class declaration might look like this:

```
class MyFrame : public wxFrame
```

```
{
```

```
public:
```

```
    MyFrame(...) : wxFrame(...) { }
```

```
...
```

```
protected:
```

```
int m_whatever;
```

```
private:
```

```
// Notice that as the event handlers normally are not called from outside
```

```
// the class, they normally are private. In particular they don't need
```

```
// to be public.
```

```
void OnExit(wxCommandEvent& event);
```

```
void OnButton1(wxCommandEvent& event);
```

```
void OnSize(wxSizeEvent& event);
```

```
// it's common to call the event handlers OnSomething() but there is no
```

```
// obligation to do that; this one is an event handler too:
```

```
void DoTest(wxCommandEvent& event);
```

```
DECLARE_EVENT_TABLE()
```

```
};
```

Next the event table must be defined and, as with any definition, it must be placed in an implementation file. The event table tells wxWidgets how to map events to member functions and in our example it could look like this:

```
wxBEGIN_EVENT_TABLE(MyFrame, wxFrame)
```

```
EVT_MENU(wxID_EXIT, MyFrame::OnExit)
```

```
EVT_MENU(DO_TEST, MyFrame::DoTest)
```

```
EVT_SIZE(MyFrame::OnSize)
```

```
EVT_BUTTON(BUTTON1, MyFrame::OnButton1)
```

```
wxEND_EVENT_TABLE()
```

Notice that you must mention a method you want to use for the event handling in the event table definition; just defining it in MyFrame class is not enough.

Let us now look at the details of this definition: the first line means that we are defining the event table for MyFrame class and that its base class is wxFrame, so events not processed by MyFrame will, by default, be handled by wxFrame. The next four lines define bindings of individual events to their handlers: the first two of them map menu commands from the items with the identifiers specified as the first macro parameter to two different member functions. In the next one, EVT_SIZE means that any changes in the size of the frame will result in calling OnSize() method. Note that this macro doesn't need a window identifier, since normally you are only interested in the current window's size events.

The EVT_BUTTON macro demonstrates that the originating event does not have to come from the window class implementing the event table -- if the event source is a button within a panel within a frame, this will still work, because event tables are searched up through the hierarchy of windows for the command events. (But only command events, so you can't catch mouse move events in a child control in the parent window in the same way because wxMouseEvent doesn't derive from wxCommandEvent. See below for how you can do it.) In this case, the button's event table will be searched, then the parent panel's, then the frame's.

Finally, you need to implement the event handlers. As mentioned before, all event handlers take a wxEvent-derived argument whose exact class differs according to the type of event and the class of the originating window. For size events, wxSizeEvent is used. For menu commands and most control commands (such as button presses), wxCommandEvent is used. When controls get more complicated, more specific wxCommandEvent-derived event classes providing additional control-specific information can be used, such as wxTreeEvent for events from wxTreeCtrl windows.

In the simplest possible case an event handler may not use the event parameter at all. For example,

```
void MyFrame::OnExit(wxCommandEvent& WXUNUSED(event))
{
    // when the user selects "Exit" from the menu we should close
    Close(true);
}
```

In other cases you may need some information carried by the event argument, as in:

```
void MyFrame::OnSize(wxSizeEvent& event)
{
    wxSize size = event.GetSize();

    ... update the frame using the new size ...
}
```

You will find the details about the event table macros and the corresponding wxEvent-derived classes in the discussion of each control generating these events.

Dynamic Event Handling

The possibilities of handling events in this way are rather different. Let us start by looking at the syntax: the first obvious difference is that you need not use `DECLARE_EVENT_TABLE()` nor `BEGIN_EVENT_TABLE()` and the associated macros. Instead, in any place in your code, but usually in the code of the class defining the handler itself (and definitely not in the global scope as with the event tables), call its `Bind<>()` method like this:

```
MyFrame::MyFrame(...)
{
    Bind(wxEVT_COMMAND_MENU_SELECTED, &MyFrame::OnExit, this, wxID_EXIT);
}
```

Note that this pointer must be specified here.

Now let us describe the semantic differences:

Event handlers can be bound at any moment. For example, it's possible to do some initialization first and only bind the handlers if and when it succeeds. This can avoid the need to test that the object was properly initialized in the event handlers themselves. With `Bind<>()` they simply won't be called if it wasn't correctly initialized.

As a slight extension of the above, the handlers can also be unbound at any time with `Unbind<>()` (and maybe rebound later). Of course, it's also possible to emulate this behaviour with the classic static (i.e., bound via event tables) handlers by using an internal flag indicating whether the handler is currently enabled and returning from it if it isn't, but using dynamically bind handlers requires less code and is also usually more clear.

Almost last but very, very far from least is the increased flexibility which allows to bind an event to:

A method in another object.

An ordinary function like a static method or a global function.

An arbitrary functor like `boost::function<>`.

This is impossible to do with the event tables because it is not possible to specify these handlers to dispatch the event to, so it necessarily needs to be sent to the same object which generated the event. Not so with `Bind<>()` which can be used to specify these handlers which will handle the event. To give a quick example, a common question is how to receive the mouse movement events happening when the mouse is in one of the frame children in the frame itself. Doing it in a naive way doesn't work:

A `EVT_LEAVE_WINDOW(MyFrame::OnMouseLeave)` line in the frame event table has no effect as mouse move (including entering and leaving) events are not propagated up to the parent window (at least not by default).

Putting the same line in a child event table will crash during run-time because the `MyFrame` method will be called on a wrong object -- it's easy to convince oneself that the only object that can be used here is the pointer to the child, as `wxWidgets` has nothing else. But calling a frame method with the child window pointer instead of the pointer to the frame is, of course, disastrous.

However writing

```
MyFrame::MyFrame(...)
{
    m_child->Bind(wxEVT_LEAVE_WINDOW, &MyFrame::OnMouseLeave, this);
}
```

will work exactly as expected. Note that you can get the object that generated the event -- and that is not the same as the frame -- via `wxEvt::GetEventObject()` method of event argument passed to the event handler.

Really last point is the consequence of the previous one: because of increased flexibility of `Bind()`, it is also safer as it is impossible to accidentally use a method of another class. Instead of run-time crashes you will get compilation errors in this case when using `Bind()`.

Let us now look at more examples of how to use different event handlers using the two overloads of `Bind()` function: first one for the object methods and the other one for arbitrary functors (callable objects, including simple functions):

In addition to using a method of the object generating the event itself, you can use a method from a completely different object as an event handler:

```
void MyFrameHandler::OnFrameExit( wxCommandEvent & )
{
    // Do something useful.
}
```

```
MyFrameHandler myFrameHandler;
```

```
MyFrame::MyFrame()
{
    Bind( wxEVT_COMMAND_MENU_SELECTED, &MyFrameHandler::OnFrameExit,
        &myFrameHandler, wxID_EXIT );
}
```



```
}
```

Note that `MyFrameHandler` doesn't need to derive from `wxEvtHandler`. But keep in mind that then the lifetime of `myFrameHandler` must be greater than that of `MyFrame` object -- or at least it needs to be unbound before being destroyed.

To use an ordinary function or a static method as an event handler you would write something like this:

```
void HandleExit( wxCommandEvent & )
```

```
{
```

```
    // Do something useful
```

```
}
```

```
MyFrame::MyFrame()
```

```
{
```

```
    Bind( wxEVT_COMMAND_MENU_SELECTED, &HandleExit, wxID_EXIT );
```

```
}
```

And finally you can bind to an arbitrary functor and use it as an event handler:

```
struct MyFunctor
```

```
{
```

```
    void operator()( wxCommandEvent & )
```

```
{
```

```
    // Do something useful
```

```
}
```

```
};
```

```
MyFunctor myFunctor;
```

```
MyFrame::MyFrame()
```

```
{
    Bind( wxEVT_COMMAND_MENU_SELECTED, &myFunctor, wxID_EXIT );
}
```

A common example of a functor is `boost::function`:

```
using namespace boost;
```

```
void MyHandler::OnExit( wxCommandEvent & )
```

```
{
    // Do something useful
}
```

```
MyHandler myHandler;
```

```
MyFrame::MyFrame()
```

```
{
    function< void ( wxCommandEvent & ) > exitHandler( bind( &MyHandler::OnExit, &myHandler, _1 ));
```

```
    Bind( wxEVT_COMMAND_MENU_SELECTED, exitHandler, wxID_EXIT );
}
```

With the aid of `boost::bind` you can even use methods or functions which don't quite have the correct signature:

```
void MyHandler::OnExit( int exitCode, wxCommandEvent &, wxString goodByeMessage )
```

```
{
    // Do something useful
}
```

```
MyHandler myHandler;
```

```
MyFrame::MyFrame()
{
    function< void ( wxCommandEvent & ) > exitHandler(

    bind( &MyHandler::OnExit, &myHandler, EXIT_FAILURE, _1, "Bye" ));

    Bind( wxEVT_COMMAND_MENU_SELECTED, exitHandler, wxID_EXIT );
}
```

To summarize, using Bind<>() requires slightly more typing but is much more flexible than using static event tables so don't hesitate to use it when you need this extra power. On the other hand, event tables are still perfectly fine in simple situations where this extra flexibility is not needed.

How Events are Processed

The previous sections explain how to define event handlers but don't address the question of how exactly wxWidgets finds the handler to call for the given event. This section describes the algorithm used in detail. Notice that you may want to run the Event Sample while reading this section and look at its code and the output when the button which can be used to test the event handlers execution order is clicked to understand it better.

When an event is received from the windowing system, wxWidgets calls wxEvtHandler::ProcessEvent() on the first event handler object belonging to the window generating the event. The normal order of event table searching by ProcessEvent() is as follows, with the event processing stopping as soon as a handler is found (unless the handler calls wxEvent::Skip() in which case it doesn't count as having handled the event and the search continues):

Step 0) Before anything else happens, wxApp::FilterEvent() is called. If it returns anything but -1 (default), the event handling stops immediately.

Step 1) If this event handler is disabled via a call to wxEvtHandler::SetEvtHandlerEnabled() the next three steps are skipped and the event handler resumes at step (5).

Step 2) If the object is a wxWindow and has an associated validator, wxValidator gets a chance to process the event.

Step 3) The list of dynamically bound event handlers, i.e., those for which Bind<>() was called, is consulted. Notice that this is done before checking the static event table entries, so if both a dynamic and a static event handler match the same event, the static one is never going to be used unless wxEvent::Skip() is called in the dynamic one.

Step 4) The event table containing all the handlers defined using the event table macros in this class and its base classes is examined. Notice that this means that any event handler defined in a base class will be executed at this step.

Step 5) The event is passed to the next event handler, if any, in the event handler chain, i.e., the steps (1) to (4) are done for it. Usually there is no next event handler so the control passes to the next step but see Event Handlers Chain for how the next handler may be defined.

Step 6) If the object is a `wxWindow` and the event is set to propagate (by default only `wxCommandEvent`-derived events are set to propagate), then the processing restarts from the step (1) (and excluding the step (7)) for the parent window. If this object is not a window but the next handler exists, the event is passed to its parent if it is a window. This ensures that in a common case of (possibly several) non-window event handlers pushed on top of a window, the event eventually reaches the window parent.

Step 7) Finally, i.e., if the event is still not processed, the `wxApp` object itself (which derives from `wxEvtHandler`) gets a last chance to process it.

Please pay close attention to step 76 People often overlook or get confused by this powerful feature of the `wxWidgets` event processing system. The details of event propagation up the window hierarchy are described in the next section.

Also please notice that there are additional steps in the event handling for the windows-making part of `wxWidgets` document-view framework, i.e., `wxDocParentFrame`, `wxDocChildFrame` and their MDI equivalents `wxDocMDIParentFrame` and `wxDocMDIChildFrame`. The parent frame classes modify step (2) above to send the events received by them to `wxDocManager` object first. This object, in turn, sends the event to the current view and the view itself lets its associated document process the event first. The child frame classes send the event directly to the associated view which still forwards it to its document object. Notice that to avoid remembering the exact order in which the events are processed in the document-view frame, the simplest, and recommended, solution is to only handle the events at the view classes level, and not in the document or document manager classes

How Events Propagate Upwards

As mentioned above, the events of the classes deriving from `wxCommandEvent` are propagated by default to the parent window if they are not processed in this window itself. But although by default only the command events are propagated like this, other events can be propagated as well because the event handling code uses `wxEvtHandler::ShouldPropagate()` to check whether an event should be propagated. It is also possible to propagate the event only a limited number of times and not until it is processed (or a top level parent window is reached).

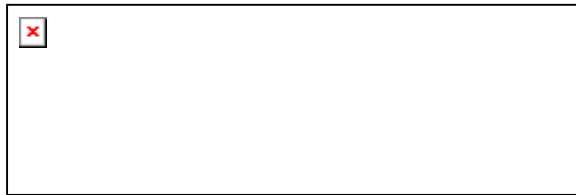
Finally, there is another additional complication (which, in fact, simplifies life of `wxWidgets` programmers significantly): when propagating the command events up to the parent window, the event propagation stops when it reaches the parent dialog, if any. This means that you don't risk getting unexpected events from the dialog controls (which might be left unprocessed by the dialog itself because it doesn't care about them) when a modal dialog is popped up. The events do propagate beyond the frames, however. The rationale for this choice is that there are only a few frames in a typical application and their parent-child relation are well understood by the programmer while it may be difficult, if not impossible, to track down all the dialogs that may be popped up in a complex program (remember that some are created automatically by `wxWidgets`). If you need to specify a different behaviour for some reason, you can use `wxWindow::SetExtraStyle(wxWS_EX_BLOCK_EVENTS)` explicitly to prevent the events from being propagated beyond the given window or unset this flag for the dialogs that have it on by default.

Typically events that deal with a window as a window (size, motion, paint, mouse, keyboard, etc.) are sent only to the window. Events that have a higher level of meaning or are generated by the window itself (button click, menu select, tree expand, etc.) are command events and are sent up to the parent to see if it is interested in the event. More precisely, as said above, all event classes not deriving from `wxCommandEvent` (see the `wxEvtHandler` inheritance map) do not propagate upward.

In some cases, it might be desired by the programmer to get a certain number of system events in a parent window, for example all key events sent to, but not used by, the native controls in a dialog. In this case, a special event handler will have to be written that will override `ProcessEvent()` in order to pass all events (or any selection of them) to the parent window.

Event Handlers Chain

The step 4 of the event propagation algorithm checks for the next handler in the event handler chain. This chain can be formed using `wxEvtHandler::SetNextHandler()`:



(referring to the image, if `A->ProcessEvent` is called and it doesn't handle the event, `B->ProcessEvent` will be called and so on...).

Additionally, in the case of `wxWindow` you can build a stack (implemented using `wxEvtHandler` double-linked list) using `wxWindow::PushEventHandler()`:



(referring to the image, if `W->ProcessEvent` is called, it immediately calls `A->ProcessEvent`; if nor A nor B handle the event, then the `wxWindow` itself is used -- i.e. the dynamically bind event handlers and static event table entries of `wxWindow` are looked as the last possibility, after all pushed event handlers were tested).

By default the chain is empty, i.e. there is no next handler.

Custom Event Summary

General approach

As each event is uniquely defined by its event type, defining a custom event starts with defining a new event type for it. This is done using `wxDEFINE_EVENT()` macro. As an event type is a variable, it can also be declared using `wxDECLARE_EVENT()` if necessary.

The next thing to do is to decide whether you need to define a custom event class for events of this type or if you can reuse an existing class, typically either `wxEvent` (which doesn't provide any extra information) or `wxCommandEvent` (which contains several extra fields and also propagates upwards by default). Both strategies are described in details below. See also the Event Sample for a complete example of code defining and working with the custom event types.

Finally, you will need to generate and post your custom events. Generation is as simple as instantiating your custom event class and initializing its internal fields. For posting events to a certain event handler there are two possibilities: using `wxEvtHandler::AddPendingEvent` or using `wxEvtHandler::QueueEvent`. Basically you will need to use the latter when doing inter-thread communication; when you use only the main thread you can also safely use the former. Last, note that there are also two simple global wrapper functions associated to the two `wxEvtHandler` mentioned functions: `wxPostEvent()` and `wxQueueEvent()`.

Using Existing Event Classes

If you just want to use a `wxCommandEvent` with a new event type, use one of the generic event table macros listed below, without having to define a new event class yourself.

Example:

```
// this is typically in a header: it just declares MY_EVENT event type
wxDECLARE_EVENT(MY_EVENT, wxCommandEvent);

// this is a definition so can't be in a header
wxDEFINE_EVENT(MY_EVENT, wxCommandEvent);

// example of code handling the event with event tables
BEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_MENU (wxID_EXIT, MyFrame::OnExit)
    ...
    EVT_COMMAND (ID_MY_WINDOW, MY_EVENT, MyFrame::OnMyEvent)
END_EVENT_TABLE()

void MyFrame::OnMyEvent(wxCommandEvent& event)
{
    // do something
```

```
wxString text = event.GetText();  
}
```

```
// example of code handling the event with Bind<>():
```

```
MyFrame::MyFrame()  
{  
    Bind(MY_EVENT, &MyFrame::OnMyEvent, this, ID_MY_WINDOW);  
}
```

```
// example of code generating the event
```

```
void MyWindow::SendEvent()  
{  
    wxCommandEvent event(MY_EVENT, GetId());  
    event.SetEventObject(this);
```

```
// Give it some contents
```

```
event.SetText("Hello");
```

```
// Do send it
```

```
ProcessWindowEvent(event);  
}
```

Defining Your Own Event Class

Under certain circumstances, you must define your own event class e.g., for sending more complex data from one place to another. Apart from defining your event class, you also need to define your own event table macro if you want to use event tables for handling events of this type.

Here is an example:

```
// define a new event class

class MyPlotEvent: public wxEvent
{
public:
    MyPlotEvent(wxEventType eventType, int winid, const wxPoint& pos)
        : wxEvent(winid, eventType),
          m_pos(pos)
    {
    }

    // accessors
    wxPoint GetPoint() const { return m_pos; }

    // implement the base class pure virtual
    virtual wxEvent *Clone() const { return new MyPlotEvent(*this); }

private:
    const wxPoint m_pos;
};

// we define a single MY_PLOT_CLICKED event type associated with the class
// above but typically you are going to have more than one event type, e.g. you
// could also have MY_PLOT_ZOOMED or MY_PLOT_PANNED &c -- in which case you
// would just add more similar lines here
wxDEFINE_EVENT(MY_PLOT_CLICKED, MyPlotEvent);
```



```
// if you want to support old compilers you need to use some ugly macros:

typedef void (wxEvtHandler::*MyPlotEventFunction)(MyPlotEvent&);

#define MyPlotEventHandler(func) wxEVENT_HANDLER_CAST(MyPlotEventFunction, func)


// if your code is only built using reasonably modern compilers, you could just

// do this instead:

#define MyPlotEventHandler(func) (&func)


// finally define a macro for creating the event table entries for the new

// event type

//

// remember that you don't need this at all if you only use Bind<>() and that

// you can replace MyPlotEventHandler(func) with just &func unless you use a

// really old compiler

#define MY_EVT_PLOT_CLICK(id, func) \

    wx__DECLARE_EVT1(MY_PLOT_CLICKED, id, MyPlotEventHandler(func))


// example of code handling the event (you will use one of these methods, not

// both, of course):

BEGIN_EVENT_TABLE(MyFrame, wxFrame)

    EVT_PLOT(ID_MY_WINDOW, MyFrame::OnPlot)

END_EVENT_TABLE()
```

```
MyFrame::MyFrame()
{
    Bind(MY_PLOT_CLICKED, &MyFrame::OnPlot, this, ID_MY_WINDOW);
}
```

```
void MyFrame::OnPlot(MyPlotEvent& event)
{
    ... do something with event.GetPoint() ...
}
```

// example of code generating the event:

```
void MyWindow::SendEvent()
{
    MyPlotEvent event(MY_PLOT_CLICKED, GetId(), wxPoint(...));
    event.SetEventObject(this);
    ProcessWindowEvent(event);
}
```

Miscellaneous Notes

Event Handlers vs Virtual Methods

It may be noted that wxWidgets' event processing system implements something close to virtual methods in normal C++ in spirit: both of these mechanisms allow you to alter the behaviour of the base class by defining the event handling functions in the derived classes.

There is however an important difference between the two mechanisms when you want to invoke the default behaviour, as implemented by the base class, from a derived class handler. With the virtual functions, you need to call the base class function directly and you can do it either in the beginning of the derived class handler function (to post-process the event) or at its end (to pre-process the event). With the event handlers, you only have the option of pre-processing the events and in order to still let the default behaviour happen you must call `wxEvtHandler::Skip()` and not call the base class event handler directly. In fact, the event handler probably doesn't even exist in the base class as the default behaviour is often implemented in platform-specific code by the underlying toolkit or OS itself. But even if it does exist at `wxWidgets` level, it should never be called directly as the event handlers are not part of `wxWidgets` API and should never be called directly.

User Generated Events vs Programmatically Generated Events

While generically `wxEvtHandler` can be generated both by user actions (e.g., `resize` of a `wxWindow`) and by calls to functions (e.g., `wxWindow::SetSize`), `wxWidgets` controls normally send `wxCommandEvent`-derived events only for the user-generated events. The only exceptions to this rule are:

`wxNotebook::AddPage`: No event-free alternatives

`wxNotebook::AdvanceSelection`: No event-free alternatives

`wxNotebook::DeletePage`: No event-free alternatives

`wxNotebook::SetSelection`: Use `wxNotebook::ChangeSelection` instead, as `wxNotebook::SetSelection` is deprecated

`wxTreeCtrl::Delete`: No event-free alternatives

`wxTreeCtrl::DeleteAllItems`: No event-free alternatives

`wxTreeCtrl::EditLabel`: No event-free alternatives

All `wxTextCtrl` methods

`wxTextCtrl::ChangeValue` can be used instead of `wxTextCtrl::SetValue` but the other functions, such as `wxTextCtrl::Replace` or `wxTextCtrl::WriteText` don't have event-free equivalents.

Pluggable Event Handlers

TODO: Probably deprecated, `Bind()` provides a better way to do this

In fact, you don't have to derive a new class from a window class if you don't want to. You can derive a new class from `wxEvtHandler` instead, defining the appropriate event table, and then call `wxWindow::SetEventHandler` (or, preferably, `wxWindow::PushEventHandler`) to make this event handler the object that responds to events. This way, you can avoid a lot of class derivation, and use instances of the same event handler class (but different objects as the same event handler object shouldn't be used more than once) to handle events from instances of different widget classes.

If you ever have to call a window's event handler manually, use the `GetEventHandler` function to retrieve the window's event handler and use that to call the member function. By default, `GetEventHandler` returns a pointer to the window itself unless an application has redirected event handling using `SetEventHandler` or `PushEventHandler`.

One use of `PushEventHandler` is to temporarily or permanently change the behaviour of the GUI. For example, you might want to invoke a dialog editor in your application that changes aspects of dialog boxes. You can grab all the input for an existing dialog box, and edit it 'in situ', before restoring its behaviour to normal. So even if the application has derived new classes to customize behaviour, your utility can indulge in a spot of body-snatching. It could be a useful technique for on-line tutorials, too, where you take a user through a series of steps and don't want them to diverge from the lesson. Here, you can examine the events coming from buttons and windows, and if acceptable, pass them through to the original event handler. Use `PushEventHandler/PopEventHandler` to form a chain of event handlers, where each handler processes a different range of events independently from the other handlers.

Window Identifiers

Window identifiers are integers, and are used to uniquely determine window identity in the event system (though you can use it for other purposes). In fact, identifiers do not need to be unique across your entire application as long they are unique within the particular context you're interested in, such as a frame and its children. You may use the `wxID_OK` identifier, for example, on any number of dialogs as long as you don't have several within the same dialog.

If you pass `wxID_ANY` to a window constructor, an identifier will be generated for you automatically by `wxWidgets`. This is useful when you don't care about the exact identifier either because you're not going to process the events from the control being created or because you process the events from all controls in one place (in which case you should specify `wxID_ANY` in the event table or `wxEvtHandler::Bind` call as well). The automatically generated identifiers are always negative and so will never conflict with the user-specified identifiers which must be always positive.

See Standard event identifiers for the list of standard identifiers available. You can use `wxID_HIGHEST` to determine the number above which it is safe to define your own identifiers. Or, you can use identifiers below `wxID_LOWEST`. Finally, you can allocate identifiers dynamically using `wxNewId()` function too. If you use `wxNewId()` consistently in your application, you can be sure that your identifiers don't conflict accidentally.

Generic Event Table Macros

<code>EVT_CUSTOM(event, id, func)</code>	Allows you to add a custom event table entry by specifying the event identifier (such as <code>wxEVT_SIZE</code>), the window identifier, and a member function to call.
<code>EVT_CUSTOM_RANGE(event, id1, id2, func)</code>	The same as <code>EVT_CUSTOM</code> , but responds to a range of window identifiers.
<code>EVT_COMMAND(id, event, func)</code>	The same as <code>EVT_CUSTOM</code> , but expects a member function with a <code>wxCommandEvent</code> argument.
<code>EVT_COMMAND_RANGE(id1, id2, event, func)</code>	The same as <code>EVT_CUSTOM_RANGE</code> , but expects a member function with a <code>wxCommandEvent</code> argument.

EVT_NOTIFY(event, id, func)	The same as EVT_CUSTOM, but expects a member function with a wxNotifyEvent argument.
EVT_NOTIFY_RANGE(event, id1, id2, func)	The same as EVT_CUSTOM_RANGE, but expects a member function with a wxNotifyEvent argument.

List of wxWidgets events

For the full list of event classes, please see the event classes group page.

3.2.3.3.4 Low-Level Widget API

The Low-Level Widget API identifies the basic Curses-/nCurses-style (Terminal Control Library) Graphical User Interface features available to the Software Engineer that supports input and output interactions with the local or remote terminal. It is used exclusively to construct, enhance and maintain the wxPython-style, High-Level Widget API emulation. It includes the following GUI objects:

- 1 keyboard** - a Curses controlled object with alphabetic, numeric, punctuation and control buttons used by an operator to enter input.
- 2 mouse** - a Curses controlled object with a position sensing roller ball and one or more buttons used by an operator to enter input that selects a GUI object on the display screen.
- 3 display screen** - a Curses controlled object with multiple columns and rows that displays information output by the application for use by an operator
- 4 window** - a Curses controlled object displayed in a designated portion of the display screen for application specific information
- 5 pad** - a Curses controlled object like a window, except that it is not restricted by the screen size, and is not necessarily associated with a particular part of the screen
- 6 panel** - a Curses controlled object like a window with the added feature of depth, so it can be stacked on top of another, and only the visible portions of each window will be displayed. Panels can be added, moved up or down in the stack, and removed

3.2.3.3.4.1 Terminal Control Library Interface

Excerpts From Wikipedia, the free encyclopedia at:

[https://en.wikipedia.org/wiki/Curses_\(programming_library\)](https://en.wikipedia.org/wiki/Curses_(programming_library))

"Overview

The curses API is described in several places. 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.

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.

In short, this means that the programmer simply creates a character matrix of how the screen should look and lets curses handle the work."

"Portability

Although the ncurses library was initially developed under Linux, OpenBSD, FreeBSD, and NetBSD it has been ported to many other ANSI/POSIX UNIX systems, mainly by Thomas Dickey. PDCurses, while not identical to ncurses, uses the same function calls and operates the same way as ncurses does except that PDCurses targets different devices, e.g., console windows for DOS, Win32, OS/2, as well as X11. Porting between the two is not difficult. For example, the roguelike game ADOM was written for Linux and ncurses, later ported to DOS and PDCurses."

- 1 Each platform shall provide a curses-style Terminal-independent Application Programming Interface (API).
 - a) Termcap-based curses on BSD UNIX
 - b) Terminfo-based curses on AT&T UNIX
 - c) Terminfo-based nCurses on Linux and on Cygwin for Microsoft Windows
- 2 Some platforms may optionally provide a PDCurses-base Terminal-independent Application Programming Interface (API).

3.2.3.3.5 System Preferences API (Draft)

The computer software product "tsWxGTUI_PyVx" Toolkit includes a configuration file, "tsWxGlobals.py" that may be customized to determine the look and feel of "wxPython" emulation.

- 1 Symbolic Constants
- 2 Foreground and Background Colors

- 3** Default Styles and Themes
- 4** Unique ID Generators
- 5** Dimensional Unit Conversion Functions
- 6** Text Font and Color Attribute Markup Constants

3.3 CSCI External Interface Requirements

This paragraph shall be divided into subparagraphs to specify the requirements, if any, for the CSCI's external interfaces. This paragraph may reference one or more Interface Requirements Specifications (IRSSs) or other documents containing these requirements.

3.3.1 Interface Identification And Diagrams

This paragraph shall identify the required external interfaces of the CSCI (that is, relationships with other entities that involve sharing, providing or exchanging data). The identification of each interface shall include a project-unique identifier and shall designate the interfacing entities (systems, configuration items, users, etc.) by name, number, version, and documentation references, as applicable. The identification shall state which entities have fixed interface characteristics (and therefore impose interface requirements on interfacing entities) and which are being developed or modified (thus having interface requirements imposed on them). One or more interface diagrams shall be provided to depict the interfaces.

See System Block Diagram

3.3.2 Human Interface

This paragraph (beginning with 3.3.2) shall identify a CSCI external interface by project-unique identifier, shall briefly identify the interfacing entities, and shall be divided into subparagraphs as needed to state the requirements imposed on the CSCI to achieve the interface. Interface characteristics of the other entities involved in the interface shall be stated as assumptions or as "When [the entity not covered] does this, the CSCI shall...", not as requirements on the other entities. This paragraph may reference other documents (such as data dictionaries, standards for communication protocols, and standards for user interfaces) in place of stating the information here. The requirements shall include the following, as applicable, presented in any order suited to the requirements, and shall note any differences in these characteristics from the point of view of the interfacing entities (such as different expectations about the size, frequency, or other characteristics of data elements):

- a. Priority that the CSCI must assign the interface
 - b. Requirements on the type of interface (such as real-time data transfer, storage-and-retrieval of data, etc.) to be implemented
 - c. Required characteristics of individual data elements that the CSCI must provide, store, send, access, receive, etc., such as:
 - 1) Names/identifiers
 - a) Project-unique identifier
 - b) Non-technical (natural-language) name
 - c) DoD standard data element name
 - d) Technical name (e.g., variable or field name in code or database)
 - e) Abbreviation or synonymous names
 - 2) Data type (alphanumeric, integer, etc.)
 - 3) Size and format (such as length and punctuation of a character string)
 - 4) Units of measurement (such as meters, dollars, nanoseconds)
 - 5) Range or enumeration of possible values (such as 0-99)
 - 6) Accuracy (how correct) and precision (number of significant digits)
 - 7) Priority, timing, frequency, volume, sequencing, and other constraints, such as whether the data element may be updated and whether business rules apply
 - 8) Security and privacy constraints
 - 9) Sources (setting/sending entities) and recipients (using/receiving entities)
-

d. Required characteristics of data element assemblies (records, messages, files, arrays, displays, reports, etc.) that the CSCI must provide, store, send, access, receive, etc., such as:

- 1) Names/identifiers
 - a) Project-unique identifier
 - b) Non-technical (natural language) name
 - c) Technical name (e.g., record or data structure name in code or database)
 - d) Abbreviations or synonymous names
- 2) Data elements in the assembly and their structure (number, order, grouping)
- 3) Medium (such as disk) and structure of data elements/assemblies on the medium
- 4) Visual and auditory characteristics of displays and other outputs (such as colors, layouts, fonts, icons and other display elements, beeps, lights)
- 5) Relationships among assemblies, such as sorting/access characteristics
- 6) Priority, timing, frequency, volume, sequencing, and other constraints, such as whether the assembly may be updated and whether business rules apply
- 7) Security and privacy constraints
- 8) Sources (setting/sending entities) and recipients (using/receiving entities)

e. Required characteristics of communication methods that the CSCI must use for the interface, such as:

- 1) Project-unique identifier(s)
 - 2) Communication links/bands/frequencies/media and their characteristics
 - 3) Message formatting
 - 4) Flow control (such as sequence numbering and buffer allocation)
 - 5) Data transfer rate, whether periodic/aperiodic, and interval between transfers
 - 6) Routing, addressing, and naming conventions
 - 7) Transmission services, including priority and grade
 - 8) Safety/security/privacy considerations, such as encryption, user authentication, compartmentalization, and auditing
-

-
- f. Required characteristics of protocols the CSCI must use for the interface, such as:
- 1) Project-unique identifier(s)
 - 2) Priority/layer of the protocol
 - 3) Packeting, including fragmentation and reassembly, routing, and addressing
 - 4) Legality checks, error control, and recovery procedures
 - 5) Synchronization, including connection establishment, maintenance, termination
 - 6) Status, identification, and any other reporting features
- g. Other required characteristics, such as physical compatibility of the interfacing entities (dimensions, tolerances, loads, plug compatibility, etc.), voltages, etc.
-

3.4 CSCI Internal Interface Requirements

This paragraph shall specify the requirements, if any, imposed on interfaces internal to the CSCI. If all internal interfaces are left to the design, this fact shall be so stated. If such requirements are to be imposed, paragraph 3.3 of this DID provides a list of topics to be considered.

3.5 CSCI Internal Data Requirements

This paragraph shall specify the requirements, if any, imposed on data internal to the CSCI. Included shall be requirements, if any, on databases and data files to be included in the CSCI. If all decisions about internal data are left to the design, this fact shall be so stated. If such requirements are to be imposed, paragraphs 3.3.x.c and 3.3.x.d of this DID provide a list of topics to be considered.

3.6 Adaptation Requirements

This paragraph shall specify the requirements, if any, concerning installation-dependent data to be provided by the CSCI (such as site-dependent latitude and longitude or site-dependent state tax codes) and operational parameters that the CSCI is required to use that may vary according to operational needs (such as parameters indicating operation-dependent targeting constants or data recording).

3.7 Safety Requirements

This paragraph shall specify the CSCI requirements, if any, concerned with preventing or minimizing unintended hazards to personnel, property, and the physical environment. Examples include safeguards the CSCI must provide to prevent inadvertent actions (such as accidentally issuing an "auto pilot off" command) and non-actions (such as failure to issue an intended "auto pilot off" command). This paragraph shall include the CSCI requirements, if any, regarding nuclear components of the system, including, as applicable, prevention of inadvertent detonation and compliance with nuclear safety rules.

3.8 Security and Privacy Requirements

This paragraph shall specify the CSCI requirements, if any, concerned with maintaining security and privacy. These requirements shall include, as applicable, the security/privacy environment in which the CSCI must operate, the type and degree of security or privacy to be provided, the security/privacy risks the CSCI must withstand, required safeguards to reduce those risks, the security/privacy policy that must be met, the security/privacy accountability the CSCI must provide, and the criteria that must be met for security/privacy certification/accreditation.

3.9 CSCI Environment Requirements

This paragraph shall specify the requirements, if any, regarding the environment in which the CSCI must operate. Examples include the computer hardware and operating system on which the CSCI must run. (Additional requirements concerning computer resources are given in the next paragraph.)

3.10 Computer Resource Requirements

This paragraph shall be divided into the following subparagraphs.

3.10.1 Computer Hardware Requirements

This paragraph shall specify the requirements, if any, regarding computer hardware that must be used by the CSCI. The requirements shall include, as applicable, number of each type of equipment, type, size, capacity, and other required characteristics of process memory, input/output devices, auxiliary storage, communications/network equipment, and other required equipment.

3.10.2 Computer Hardware Resource Utilization Requirements

This paragraph shall specify the requirements, if any, on the CSCI's computer hardware resource utilization, such as maximum allowable use of processor capacity, memory capacity, input/output device capacity, auxiliary storage device capacity, and communications/network equipment capacity. The requirements (stated, for example, as percentages of the capacity of each computer hardware resource) shall include the conditions, if any, under which the resource utilization is to be measured.

3.10.3 Computer Software Requirements

This paragraph shall specify the requirements, if any, regarding computer software that must be used by, or incorporated into, the CSCI. Examples include operating systems, database management systems, communications/ network software, utility software, input and equipment simulators, test software, and manufacturing software. The correct nomenclature, version, and document references of each such software item shall be provided.

3.10.4 Computer Communications Requirements

This paragraph shall specify the additional requirements, if any, concerning the computer communication that must be used by the CSCI. Examples include geographic locations to be linked; configuration and network topology; transmission techniques; data transfer rates; gateways; required system use times; type and volume of data to be transmitted/received; time bound for transmission/ reception/response; peak volumes of data; and diagnostic features.

3.11 Software Quality Factors

This paragraph shall specify the CSCI requirements, if any, concerned with software quality factors identified in the contract or derived from a higher level specification. Examples include quantitative requirements regarding CSCI functionality (the ability to perform all required functions), reliability (the ability to perform with correct, consistent results), maintainability (the ability to be easily corrected), availability (the ability to be accessed and operated when needed), flexibility (the ability to be easily adapted to changing requirements), portability (the ability to be easily modified for a new environment), reusability (the ability to be used in multiple applications), testability (the ability to be easily and thoroughly tested), usability (the ability to be easily learned and used), and other attributes.

3.12 Design and Implementation Constraints

This paragraph shall specify the requirements, if any, that constrain the design and implementation of the CSCI. These requirements may be specified by reference to appropriate commercial or military standards and specifications. Examples include requirements concerning:

- a. Use of a particular CSCI architecture or requirements on the architecture, such as required databases or other software units; use of standard, military, or existing components; or use of Government/acquirer-furnished property (equipment, information, or software)
 - b. Use of particular design or implementation standards; use of particular data standards; use of a particular programming language
 - c. Flexibility and expandability that must be provided to support anticipated areas of growth or changes in technology, threat, or mission
-

Since each software release shall provide the means to:

- 1 Install in developer-sandbox and/or site-package configurations; and
- 2 Use with Python 2x and/or Python 3x language generations

The design and implementation shall therefore be constrained as described in the sections of the following appendix:

- ***APPENDIX B - Software Design Constraints*** (on page 281)

3.12.1 Python Language Design and Implementation Constraints

The file created for the user to download, "tsWxGUI_PyVx-0.0.0", shall be a container of one or more Toolkit subdirectories:

1 "tsWxGTUI_Py1x" Toolkit (on page 104) (Reserved for Back-Port from Initial Baseline)

This subdirectory shall contain source code implemented in the first generation Python programming language, Python 1.0.0-Python 1.6.1.

Availability will depend on Toolkit user community interest in a Python generation that has not been actively updated since reaching its end-of-life in 2001.

2 "tsWxGTUI_Py2x" Toolkit (on page 105) (Dynamic Import; Reserved for Initial Baseline)

This subdirectory shall contain source code implemented in the second generation Python programming language, Python 2.0.0-Python 2.7.9.

This version uses a dynamic import mechanism, suitable for nested multi-level packages, in which each application and building block module:

- a) imports required library packages simply by name.

examples:

```
import tsLibCLI
```

```
import tsLibGUI
```

- b) imports required building block modules, and defines optional aliases, simply by name.

examples:

```
import tsCxGlobals
```

```
import tsExceptions as tse
```

```
import tsWx as wx
```

```
from tsReportUtilities import TsReportUtilities as tsru
```

- c) Library package "__init__.py" modules dynamically construct any associated full paths.

3 "tsWxGTUI_Py3x" Toolkit (on page 106) (Dynamic Import; Reserved for Port from Initial Baseline)

This subdirectory shall contain source code implemented in the third generation Python programming language, Python 3.0.0-Python 3.4.2.

This version uses a dynamic import mechanism, suitable for nested multi-level packages, in which each application and building block module:

- a) imports required library packages simply by name.

examples:

```
import tsLibCLI
```

```
import tsLibGUI
```

- b) imports required building block modules, and defines optional aliases, simply by name.

examples:

```
import tsCxGlobals
```

```
import tsExceptions as tse
```

```
import tsWx as wx
```

```
from tsReportUtilities import TsReportUtilities as tsrpu
```

c) Library package "__init__.py" modules dynamically construct any associated full paths.

4 "tsWxGTUI_Py4x" Toolkit (on page 107) (Reserved for Future Use)

This subdirectory shall contain source code implemented in the fourth generation Python programming language, Python 4.0.0.

Availability will depend on the availability of the first fourth generation Python release by the Python Software Foundation.

5 "tsWxGTUI_PyX1" Toolkit (Unfinished Static Import; Reserved for Back-Port from Initial Baseline)

This subdirectory shall contain source code implemented in the first generation Python programming language, Python 1.6.1.

This version uses a static import mechanism, suitable for a Python Global Module Index-style single-level of packages, in which each application and building block module:

a) imports required library packages simply by name.

examples:

```
import tsLibCLI
```

```
import tsLibGUI
```

b) imports required building block modules, and defines optional aliases, explicitly by path.

examples:

```
from tsLibCLI import tsCxGlobals
```

```
from tsLibCLI import tsExceptions as tse
```

```
from tsLibGUI import tsWx
```

```
from tsLibCLI import tsReportUtilities
```

```
tsrpu = tsReportUtilities.TsReportUtilities()
```

c) Library package "__init__.py" modules are empty.

6 "tsWxGTUI_PyX2" Toolkit (Static Import)

This subdirectory shall contain source code implemented in the second generation Python programming language, Python 2.0.0-Python 2.7.9.

This version uses a static import mechanism, suitable for a Python Global Module Index-style single-level of packages, in which each application and building block module:

a) imports required library packages simply by name.

examples:

```
import tsLibCLI
```

```
import tsLibGUI
```


- b) imports required building block modules, and defines optional aliases, explicitly by path.

examples:

```
from tsLibCLI import tsCxGlobals
from tsLibCLI import tsExceptions as tse
from tsLibGUI import tsWx
from tsLibCLI import tsReportUtilities
tsrpu = tsReportUtilities.TsReportUtilities()
```

- c) Library package "__init__.py" modules are empty.

7 "tsWxGTUI_PyX3" Toolkit (Static Import; Reserved for Port from Initial Baseline)

This version uses a static import mechanism, suitable for a Python Global Module Index-style single-level of packages, in which each application and building block module:

- a) imports required library packages simply by name.

examples:

```
import tsLibCLI
import tsLibGUI
```

- b) imports required building block modules, and defines optional aliases, explicitly by path.

examples:

```
from tsLibCLI import tsCxGlobals
from tsLibCLI import tsExceptions as tse
from tsLibGUI import tsWx
from tsLibCLI import tsReportUtilities
tsrpu = tsReportUtilities.TsReportUtilities()
```

- c) Library package "__init__.py" modules are empty.

8 Differences between the syntax, semantics and Python Global Module Index components of the aforementioned Python programming language generations shall render their subdirectories incompatible yet they shall share:

- a) The same Look & Feel for their Command Line Interface (CLI)
- b) The same Look & Feel for their Graphical-style User Interface (GUI)
- c) The same Toolkit Application Programming Interface (API).

3.12.1.1 "tsWxGTUI_Py1x" Toolkit

The TeamSTARS "tsWxGTUI_Py1x" Toolkit shall be reserved for users of the first generation Python Programming Language, Python 1.0.0-1.6.1.

The high-level Python programming language is used to implement the TeamSTARS "tsWxGTUI_Py1x" Toolkit:

- 1 The TeamSTARS "tsWxGTUI_Py1x" Toolkit's building block and tool components import and use run time library components from the Python Global Module Index and from Python user-installed site-packages.

Python is a popular, field proven, portable, cross-platform programming language.

Please see encyclopedia-based Python information at:

Python Programming Language

([http://www.http://en.wikipedia.org/wiki/Python_\(programming_language\)](http://www.http://en.wikipedia.org/wiki/Python_(programming_language)))

- 2 The TeamSTARS "tsWxGTUI_Py1x" Toolkit's Command Line Interface building block components import and use the Keyword-Value Pair and Positional argument parser ("argparse", "optparse" or "getopt") available in Python's Global Module Index.

Please see Python Global Module Index-based information at:

<http://docs.python.org/3/library/argparse.html>

<http://docs.python.org/2/library/optparse.html>

<http://docs.python.org/2/library/getopt.html>

- 3 The TeamSTARS "tsWxGTUI_Py1x" Toolkit's Text-based User Interface components import and use the Terminal handler ("curses") for character-cell displays available in Python's Global Module Index.

It uses "curses" to emulate a subset of the Application Programming Interface (API) of "wxPython", a wrapper to the popular "wxWidgets" pixel-mode, cross-platform Graphical User Interface Toolkit which is implemented in the C++ programming language.

The "wxPython" emulation retains the pixel-mode parameters of the "wxPython" API and mimics the look and feel of Microsoft "Windows XP" Displays which are similar to those of Linux "GTK+" Displays:

- a) GUI container features such as frames, dialogs and panels and buttons to close, iconize and maximize/restore the container.
- b) GUI control features such as buttons, checkboxes, radio buttons, scroll bars, scroll lists and status bars.
- c) GUI layout features such as box sizer and grid sizer.
- d) GUI operator notification features such as a scrolling log of date and time stamped event notification messages.
- e) GUI desktop features such as task bar buttons to control GUI container focus.

Please see encyclopedia-based information at:

<http://en.wikipedia.org/wiki/WxPython>

<http://en.wikipedia.org/wiki/WxWidgets>

<http://en.wikipedia.org/wiki/GTK%2B>

Please see Python Global Module Index-based information at:

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

3.12.1.2 "tsWxGTUI_Py2x" Toolkit

The TeamSTARS "tsWxGTUI_Py2x" Toolkit shall be reserved for users of the second generation Python Programming Language, Python 2.0.0-2.7.9.

The high-level Python programming language is used to implement the TeamSTARS "tsWxGTUI_Py2x" Toolkit:

- 1 The TeamSTARS "tsWxGTUI_Py2x" Toolkit's building block and tool components import and use run time library components from the Python Global Module Index and from Python user-installed site-packages.

Python is a popular, field proven, portable, cross-platform programming language.

Please see encyclopedia-based Python information at:

Python Programming Language
([http://www.http://en.wikipedia.org/wiki/Python_\(programming_language\)](http://www.http://en.wikipedia.org/wiki/Python_(programming_language)))

- 2 The TeamSTARS "tsWxGTUI_Py2x" Toolkit's Command Line Interface building block components import and use the Keyword-Value Pair and Positional argument parser ("argparse", "optparse" or "getopt") available in Python's Global Module Index.

Please see Python Global Module Index-based information at:

<http://docs.python.org/3/library/argparse.html>

<http://docs.python.org/2/library/optparse.html>

<http://docs.python.org/2/library/getopt.html>

- 3 The TeamSTARS "tsWxGTUI_Py2x" Toolkit's Text-based User Interface components import and use the Terminal handler ("curses") for character-cell displays available in Python's Global Module Index.

It uses "curses" to emulate a subset of the Application Programming Interface (API) of "wxPython", a wrapper to the popular "wxWidgets" pixel-mode, cross-platform Graphical User Interface Toolkit which is implemented in the C++ programming language.

The "wxPython" emulation retains the pixel-mode parameters of the "wxPython" API and mimics the look and feel of Microsoft "Windows XP" Displays which are similar to those of Linux "GTK+" Displays:

- a) GUI container features such as frames, dialogs and panels and buttons to close, iconize and maximize/restore the container.
- b) GUI control features such as buttons, checkboxes, radio buttons, scroll bars, scroll lists and status bars.
- c) GUI layout features such as box sizer and grid sizer.
- d) GUI operator notification features such as a scrolling log of date and time stamped event notification messages.
- e) GUI desktop features such as task bar buttons to control GUI container focus.

Please see encyclopedia-based information at:

<http://en.wikipedia.org/wiki/WxPython>

<http://en.wikipedia.org/wiki/WxWidgets>

<http://en.wikipedia.org/wiki/GTK%2B>

Please see Python Global Module Index-based information at:

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

3.12.1.3 "tsWxGTUI_Py3x" Toolkit

The TeamSTARS "tsWxGTUI_Py3x" Toolkit shall be reserved for users of the third generation Python Programming Language, Python 3.00-3.4.2.

The high-level Python programming language is used to implement the TeamSTARS "tsWxGTUI_Py3x" Toolkit:

- 1 The TeamSTARS "tsWxGTUI_Py3x" Toolkit's building block and tool components import and use run time library components from the Python Global Module Index and from Python user-installed site-packages.

Python is a popular, field proven, portable, cross-platform programming language.

Please see encyclopedia-based Python information at:

Python Programming Language

([http://www.http://en.wikipedia.org/wiki/Python_\(programming_language\)](http://www.http://en.wikipedia.org/wiki/Python_(programming_language)))

- 2 The TeamSTARS "tsWxGTUI_Py3x" Toolkit's Command Line Interface building block components import and use the Keyword-Value Pair and Positional argument parser ("argparse", "optparse" or "getopt") available in Python's Global Module Index.

Please see Python Global Module Index-based information at:

<http://docs.python.org/3/library/argparse.html>

<http://docs.python.org/2/library/optparse.html>

<http://docs.python.org/2/library/getopt.html>

- 3 The TeamSTARS "tsWxGTUI_Py3x" Toolkit's Text-based User Interface components import and use the Terminal handler ("curses") for character-cell displays available in Python's Global Module Index.

It uses "curses" to emulate a subset of the Application Programming Interface (API) of "wxPython", a wrapper to the popular "wxWidgets" pixel-mode, cross-platform Graphical User Interface Toolkit which is implemented in the C++ programming language.

The "wxPython" emulation retains the pixel-mode parameters of the "wxPython" API and mimics the look and feel of Microsoft "Windows XP" Displays which are similar to those of Linux "GTK+" Displays:

- a) GUI container features such as frames, dialogs and panels and buttons to close, iconize and maximize/restore the container.
- b) GUI control features such as buttons, checkboxes, radio buttons, scroll bars, scroll lists and status bars.
- c) GUI layout features such as box sizer and grid sizer.

- d) GUI operator notification features such as a scrolling log of date and time stamped event notification messages.
- e) GUI desktop features such as task bar buttons to control GUI container focus.

Please see encyclopedia-based information at:

<http://en.wikipedia.org/wiki/WxPython>

<http://en.wikipedia.org/wiki/WxWidgets>

<http://en.wikipedia.org/wiki/GTK%2B>

Please see Python Global Module Index-based information at:

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

3.12.1.4 "tsWxGTUI_Py4x" Toolkit

The TeamSTARS "tsWxGTUI_Py4x" Toolkit shall be reserved for users of the fourth generation Python Programming Language, Python 4.x.

The high-level Python programming language is used to implement the TeamSTARS "tsWxGTUI_Py4x" Toolkit:

- 1 The TeamSTARS "tsWxGTUI_Py4x" Toolkit's building block and tool components import and use run time library components from the Python Global Module Index and from Python user-installed site-packages.

Python is a popular, field proven, portable, cross-platform programming language.

Please see encyclopedia-based Python information at:

Python Programming Language

([http://www.http://en.wikipedia.org/wiki/Python_\(programming_language\)](http://www.http://en.wikipedia.org/wiki/Python_(programming_language)))

- 2 The TeamSTARS "tsWxGTUI_Py4x" Toolkit's Command Line Interface building block components import and use the Keyword-Value Pair and Positional argument parser ("argparse", "optparse" or "getopt") available in Python's Global Module Index.

Please see Python Global Module Index-based information at:

<http://docs.python.org/3/library/argparse.html>

<http://docs.python.org/2/library/optparse.html>

<http://docs.python.org/2/library/getopt.html>

- 3 The TeamSTARS "tsWxGTUI_Py4x" Toolkit's Text-based User Interface components import and use the Terminal handler ("curses") for character-cell displays available in Python's Global Module Index.

It uses "curses" to emulate a subset of the Application Programming Interface (API) of "wxPython", a wrapper to the popular "wxWidgets" pixel-mode, cross-platform Graphical User Interface Toolkit which is implemented in the C++ programming language.

The "wxPython" emulation retains the pixel-mode parameters of the "wxPython" API and mimics the look and feel of Microsoft "Windows XP" Displays which are similar to those of Linux "GTK+" Displays:

- a) GUI container features such as frames, dialogs and panels and buttons to close, iconize and maximize/restore the container.
- b) GUI control features such as buttons, checkboxes, radio buttons, scroll bars, scroll lists and status bars.
- c) GUI layout features such as box sizer and grid sizer.
- d) GUI operator notification features such as a scrolling log of date and time stamped event notification messages.
- e) GUI desktop features such as task bar buttons to control GUI container focus.

Please see encyclopedia-based information at:

<http://en.wikipedia.org/wiki/WxPython>

<http://en.wikipedia.org/wiki/WxWidgets>

<http://en.wikipedia.org/wiki/GTK%2B>

Please see Python Global Module Index-based information at:

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

3.12.1.5 "tsWxGTUI_PyVx" Toolkit

The TeamSTARS "tsWxGTUI_PyVx" Toolkit designation shall be reserved for distributions of multiple generations of Python Programming Language. Combinations of the following Python programming language generation designations may be included:

- 1 *"tsWxGTUI_Py1x" Toolkit* (on page 104) (Reserved for Back-Port from Initial Baseline)
- 2 *"tsWxGTUI_Py2x" Toolkit* (on page 105) (Reserved for Initial Baseline)
- 3 *"tsWxGTUI_Py3x" Toolkit* (on page 106) (Reserved for Port from Initial Baseline)
- 4 *"tsWxGTUI_Py4x" Toolkit* (on page 107) (Reserved for Future Use)

3.13 Personnel-related Requirements

This paragraph shall specify the CSCI requirements, if any, included to accommodate the number, skill levels, duty cycles, training needs, or other information about the personnel who will use or support the CSCI. Examples include requirements for number of simultaneous users and for built-in help or training features. Also included shall be the human factors engineering requirements, if any, imposed on the CSCI. These requirements shall include, as applicable, considerations for the capabilities and limitations of humans; foreseeable human errors under both normal and extreme conditions; and specific areas where the effects of human error would be particularly serious. Examples include requirements for color and duration of error messages, physical placement of critical indicators or keys, and use of auditory signals.

3.14 Training-related Requirements

This paragraph shall specify the CSCI requirements, if any, pertaining to training. Examples include training software to be included in the CSCI.

3.15 Logistics-related Requirements

This paragraph shall specify the CSCI requirements, if any, concerned with logistics considerations. These considerations may include: system maintenance, software support, system transportation modes, supply-system requirements, impact on existing facilities, and impact on existing equipment.

3.16 Other Requirements

This paragraph shall specify additional CSCI requirements, if any, not covered in the previous paragraphs.

3.17 Packaging Requirements

This section shall specify the requirements, if any, for packaging, labeling, and handling the CSCI for delivery (for example, delivery on 8 track magnetic tape labelled and packaged in a certain way). Applicable military specifications and standards may be referenced if appropriate.

3.18 Precedence and Criticality of Requirements

This paragraph shall specify, if applicable, the order of precedence, criticality, or assigned weights indicating the relative importance of the requirements in this specification. Examples include identifying those requirements deemed critical to safety, to security, or to privacy for purposes of singling them out for special treatment. If all requirements have equal weight, this paragraph shall so state.

Draft

4 QUALIFICATION PROVISIONS

This section shall define a set of qualification methods and shall specify for each requirement in Section 3 the method(s) to be used to ensure that the requirement has been met. A table may be used to present this information, or each requirement in Section 3 may be annotated with the method(s) to be used.

Qualification methods may include:

- a. Demonstration: The operation of the CSCI, or a part of the CSCI, that relies on observable functional operation not requiring the use of instrumentation, special test equipment, or subsequent analysis.
 - b. Test: The operation of the CSCI, or a part of the CSCI, using instrumentation or other special test equipment to collect data for later analysis.
 - c. Analysis: The processing of accumulated data obtained from other qualification methods. Examples are reduction, interpretation, or extrapolation of test results.
 - d. Inspection: The visual examination of CSCI code, documentation, etc.
 - e. Special qualification methods: Any special qualification methods for the CSCI, such as special tools, techniques, procedures, facilities, and acceptance limits.
-

Draft

5 REQUIREMENTS TRACEABILITY

This paragraph shall contain:

a. Traceability from each CSCI requirement in this specification to the system (or subsystem, if applicable) requirement addresses. (Alternatively, this traceability may be provided by annotating each requirement in Section 3.)

Note: Each level of system refinement may result in requirements not directly traceable to higher-level requirements. For example, a system architectural design that creates multiple CSCIs may result in requirements about how the CSCIs will interface, even though these interfaces are not covered in system requirements. Such requirements may be traced to a general requirement such as "system implementation" or to the system design decisions that resulted in their generation.

b. Traceability from each system (or subsystem, if applicable) requirement allocated to this CSCI to the CSCI requirements that address it. All system (subsystem) requirements allocated to this CSCI shall be accounted for. Those that trace to CSCI requirements contained in IRSs shall reference those IRSs.

Draft

6 NOTES

This section shall contain any general information that aids in understanding this specification (e.g., background information, glossary, rationale). This section shall include an alphabetical listing of all acronyms, abbreviations, and their meanings as used in this document and a list of any terms and definitions needed to underthis document.

Draft

Draft

6.1 TO-DO-LIST

Track and plan changes to "tsWxPkg" source files located in:

- 1 /WR/SoftwareGadgetry-PyPI
 - a) Python-2x/tsWxGTUI_Py2x
 - tsLibCLI
 - tsLibGUI
 - tsToolsCLI
 - tsToolsGUI
 - tsUtilities
 - b) Python-3x/tsWxGTUI_Py3x
 - tsLibCLI
 - tsLibGUI
 - tsToolsCLI
 - tsToolsGUI
 - tsUtilities

6.1.1 New Features:

- 1 Determine functional and interface requirements for determining and displaying the label character of the checkbox, radiobox or other button to be highlighted.
- 2 Modify accelerator related modules to save a separate table in each top-level window (i.e., earliest ancestor) so that the control remains local to each top-level window rather than shared among all. This will be needed to distinguish the window sizing buttons of each top-level-window.
- 3 For those platforms lacking mouse input capability, mark (with bold or standout highlights) each accelerator character that the operator should enter, via the keyboard, to select and control the associated checkbox, radiobox and other button.
- 4 Identify, dispatch and handle events associated with keyboard and mouse inputs. wxPython provides a flexible and user extensible dispatching mechanism that has not been emulated by the "tsWxGTUI_PyVx" Toolkit. It currently uses customized callbacks suited to scrolling of text windows and customized methods for the setting/clearing of check boxes and radio buttons.

6.1.2 Modifications:

- 1 Replace application references to curses's eight color-number based Foreground/Background colors with references to curses's eight color names.
 - Completed design, code, unit test, integration test and system test.

- 2 On eight-color cygwin, xterm and xterm-color terminals, map wxPython's sixty-eight color-names to curses's eight color-names.
 - Completed design, code, unit test, integration test and system test.
- 3 On 16-color xterm-16color terminals, map wxPython's sixty-eight and three additional xterm-16color names to curses's 16 color-names.
 - Completed design, code, unit test, integration test and system test.
- 4 On xterm-88color terminals, define color palette and color pairs to support wxPython's sixty-eight and three additional xterm-16color names.
 - Completed design, code, unit test, integration test but failed system test because host computer's standard nCurses library and Python 's curses module, were NOT built to support 256-color palette.
 - CentOS 7.0 applies 256 color-pair limit to xterm-256color emulator which is equivalent to mandating xterm-16color palette. This might be the final workaround.
- 5 On xterm-256color terminals, define color palette and color pairs to support wxPython's sixty-eight, three additional xterm-16color and an arbitrary selection of 69 additional color names.
 - Completed design, code, unit test, integration test but failed system test because host computer's standard nCurses library and Python 's curses module, weres NOT built to support 256-color palette.
 - CentOS 7.0 applies 256 color-pair limit to xterm-256color emulator which is equivalent to mandating xterm-16color palette. This might be the final workaround.
- 6 Enhance design to take advantage of curses.panels capabilities in order to support focus shifts between overlaying windows..
 - Under Construction.

6.1.3 Troubleshooting:

- 1 Debug tsWxPkg failure to report character under caret when over static text for title and status bar of a frame.
 - 05/31/2010 Exception raised by mouse click on any non-task bar window. Buttons are examples of failure triggers.
 - 04/15/2010 Code inspection and wing trace of tsWxStatusbar.py revealed no error in establishment of EarliestAncestor.
- 2 Debug txWxPkg failure to retore terminal to pre-startup state. For example, ctrl-C on Cygwin and Mac OS X Xterm should not result in display of ^C.
 - 05/31/2010 Failure persists despite incorporation of code to perform "stty sane".
 - 04/16/2010, reverted tsWxGraphicalTextUserInterface.py to place echo statements as in version 270-dated 06/28/2009 of tsGraphicalTextUserInterface.py because the ctrl-C problem did not exist in the earlier urwid-based design. This did not fix the problem.

- 3 Debug incorrect color pair displays for xterm-256color. Frame displayed with dark green instead of blue. Dialog displayed with darker green instead of cyan. Stand-alone test of `tsWxGraphicalTextUserInterface` has not verified color bars due to short display time (2-seconds) and lack of reference colors.
 - Google Search: NCurses 256 Color Support: But note that some terminals, while they can support 256 colors, are not able to change the palette. To compile NCurses with 256 color support, ... www.c-for-dummies.com/nCurses/256color/
 - If Python uses standard nCurses, it would only support 8 colors.
 - Must therefore suspend effort to resolve this issue.
 - CentOS 7.0 applies 256 color-pair limit to xterm-256color emulator which is equivalent to mandating xterm-16color palette. This might be the final workaround.
- 4 Debug compressed "ansi" display. Determine if due to Cygwin and Mac OS X ansi emulator. By contrast, Mac OS X vt100 emulator does work.
 - On Mac OS X, the "Terminal" utility creates a almost normal ansi display using "?" for vertical lines and "q" for horizontal lines. However, the "iTerm" application is unable to even create the shapes of the windows.
 - Must therefore suspend effort to resolve this issue.
- 5 Debug scrolling vt100 display. Determine if due to Cygwin vt100 emulator. By contrast, Cygwin and Mac OS X vt100 emulators in xterm window do work.
 - On Windows, the "Cygwin XWin Server" utility creates a normal display but "Cygwin Bash Shell" utility does not. Each screen refresh overflows the "Redirected Output" window beyond its top border leaving multiple copies of the top two lines visible.
 - Must therefore suspend effort to resolve this issue.

6.1.4 Validation/Regression Test:

- 1 No Validation and Regression testing of the following terminal emulations and database classes:
 - a) 8-color ansi terminal emulator with and without mapping from 68-color wxPython palette because no host platform reproduces GUI object shapes and border lines
- 2 Validation and Regression testing of the following terminal emulations and associated database classes:
 - a) Non-color vt100 and vt220 terminal emulators
 - b) 8-color cygwin mintty, xterm and xterm-color with and without mapping from standard 68-color wxPython palette
 - c) 16-color xterm-16color terminal emulator with and without mapping from enhanced 71-color wxPython-style palette
 - d) 16-color xterm-88color terminal emulator with and without mapping from enhanced 71-color wxPython-style palette
 - e) 16-color xterm-256color terminal emulator with and without mapping from enhanced 140-color wxPython-style palette

- 3 Validate establishment of class parent, child, earliest ancestor and curses panel GUI object overlay level relationships for each task.

6.2 Command Line Interface Library and Tools

The Command Line Interface Library (tsLibCLI) include the following computer software components (with availability as marked):

- 1 tsLibCLI/tsApplication.py (Available)
- 2 tsLibCLI/tsCommandLineEnv.py (Available)
- 3 tsLibCLI/tsCommandLineInterface.py (Available)
- 4 tsLibCLI/tsCxGlobals.py (Available)
- 5 tsLibCLI/tsDoubleLinkedList.py (Available)
- 6 tsLibCLI/tsExceptionPkg/src/tsExceptions.py (Available)
- 7 tsLibCLI/tsLoggerPkg/src/tsLogger.py (Available)
- 8 tsLibCLI/tsOperatorSettingsParser.py (Available)
- 9 tsLibCLI/tsPlatformRunTimeEnvironment.py (Available)
- 10 tsLibCLI/tsReportUtilities.py (Available)
- 11 tsLibCLI/tsSysCommands.py (Available)

6.3 Graphical Text User Interface Library and Tools (Draft)

The Graphical Text User Interface Library (tsLibGUI) include the following computer software components (with availability as marked):

- 1 tsLibGUI/tsWxPkg/src/tsWx.py (Available)
- 2 tsLibGUI/tsWxPkg/src/tsWxAcceleratorEntry.py (Available)
- 3 tsLibGUI/tsWxPkg/src/tsWxAcceleratorTable.py (Available)
- 4 tsLibGUI/tsWxPkg/src/tsWxApp.py (Available)
- 5 tsLibGUI/tsWxPkg/src/tsWxBoxSizer.py (Available)
- 6 tsLibGUI/tsWxPkg/src/tsWxButton.py (Available)
- 7 tsLibGUI/tsWxPkg/src/tsWxCallLater.py (Future)
- 8 tsLibGUI/tsWxPkg/src/tsWxCaret.py (Available)

- 9** tsLibGUI/tsWxPkg/src/tsWxCheckBox.py (Available)
- 10** tsLibGUI/tsWxPkg/src/tsWxChoice.py (Future)
- 11** tsLibGUI/tsWxPkg/src/tsWxColor.py (Available)
- 12** tsLibGUI/tsWxPkg/src/tsWxColorDatabase.py (Available)
- 13** tsLibGUI/tsWxPkg/src/tsWxCommandLineEnv.py (Available)
- 14** tsLibGUI/tsWxPkg/src/tsWxControl.py (Available)
- 15** tsLibGUI/tsWxPkg/src/tsWxControlWithItems.py (Available)
- 16** tsLibGUI/tsWxPkg/src/tsWxCursor.py (Available)
- 17** tsLibGUI/tsWxPkg/src/tsWxDebugHandlers.py (Future)
- 18** tsLibGUI/tsWxPkg/src/tsWxDialog.py (Available)
- 19** tsLibGUI/tsWxPkg/src/tsWxDialogButton.py (Available)
- 20** tsLibGUI/tsWxPkg/src/tsWxDisplay.py (Available)
- 21** tsLibGUI/tsWxPkg/src/tsWxDoubleLinkedList.py (Available)
- 22** tsLibGUI/tsWxPkg/src/tsWxEraseEvent.py (Future)
- 23** tsLibGUI/tsWxPkg/src/tsWxEvent.py (Available)
- 24** tsLibGUI/tsWxPkg/src/tsWxEventDaemon.py (Future)
- 25** tsLibGUI/tsWxPkg/src/tsWxEventLoop.py (Available)
- 26** tsLibGUI/tsWxPkg/src/tsWxEventLoopActivator.py (Future)
- 27** tsLibGUI/tsWxPkg/src/tsWxEventQueueEntry.py (Future)
- 28** tsLibGUI/tsWxPkg/src/tsWxEventTableEntry.py (Available)
- 29** tsLibGUI/tsWxPkg/src/tsWxEvtHandler.py (Available)
- 30** tsLibGUI/tsWxPkg/src/tsWxFlexGridSizer.py (Future)
- 31** tsLibGUI/tsWxPkg/src/tsWxFocusEvent.py (Future)
- 32** tsLibGUI/tsWxPkg/src/tsWxFrame.py (Available)
- 33** tsLibGUI/tsWxPkg/src/tsWxFrameButton.py (Available)
- 34** tsLibGUI/tsWxPkg/src/tsWxGauge.py (Available)
- 35** tsLibGUI/tsWxPkg/src/tsWxGlobals.py (Available)
- 36** tsLibGUI/tsWxPkg/src/tsWxGraphicalTextUserInterface.py (Available)
- 37** tsLibGUI/tsWxPkg/src/tsWxGridBagSizer.py (Future)
- 38** tsLibGUI/tsWxPkg/src/tsWxGridSizer.py (Available)
- 39** tsLibGUI/tsWxPkg/src/tsWxItemContainer.py (Future)
- 40** tsLibGUI/tsWxPkg/src/tsWxKeyboardState.py (Available)
- 41** tsLibGUI/tsWxPkg/src/tsWxKeyEvent.py (Available)

- 42** tsLibGUI/tsWxPkg/src/tsWxLayoutAlgorithm.py (Future)
- 43** tsLibGUI/tsWxPkg/src/tsWxListBox.py (Future)
- 44** tsLibGUI/tsWxPkg/src/tsWxLog.py (Future)
- 45** tsLibGUI/tsWxPkg/src/tsWxMenu.py (Future)
- 46** tsLibGUI/tsWxPkg/src/tsWxMenuBar.py (Future)
- 47** tsLibGUI/tsWxPkg/src/tsWxMouseEvent.py (Available)
- 48** tsLibGUI/tsWxPkg/src/tsWxMouseState.py (Available)
- 49** tsLibGUI/tsWxPkg/src/tsWxMultiFrameEnv.py (Available)
- 50** tsLibGUI/tsWxPkg/src/tsWxObject.py (Available)
- 51** tsLibGUI/tsWxPkg/src/tsWxPanel.py (Available)
- 52** tsLibGUI/tsWxPkg/src/tsWxPoint.py (Available)
- 53** tsLibGUI/tsWxPkg/src/tsWxPyApp.py (Available)
- 54** tsLibGUI/tsWxPkg/src/tsWxPyEventBinder.py (Available)
- 55** tsLibGUI/tsWxPkg/src/tsWxPyOnDemandOutputWindow.py (Available)
- 56** tsLibGUI/tsWxPkg/src/tsWxPySimpleApp.py (Available)
- 57** tsLibGUI/tsWxPkg/src/tsWxPySizer.py (Available)
- 58** tsLibGUI/tsWxPkg/src/tsWxRadioBox.py (Available)
- 59** tsLibGUI/tsWxPkg/src/tsWxRadioButton.py (Available)
- 60** tsLibGUI/tsWxPkg/src/tsWxRect.py (Available)
- 61** tsLibGUI/tsWxPkg/src/tsWxScreen.py (Available)
- 62** tsLibGUI/tsWxPkg/src/tsWxScrollBar.py (Available)
- 63** tsLibGUI/tsWxPkg/src/tsWxScrollBarButton.py (Available)
- 64** tsLibGUI/tsWxPkg/src/tsWxScrollBarGauge.py (Available)
- 65** tsLibGUI/tsWxPkg/src/tsWxScrolled.py (Available)
- 66** tsLibGUI/tsWxPkg/src/tsWxScrolledText.py (Available)
- 67** tsLibGUI/tsWxPkg/src/tsWxScrolledWindow.py (Available)
- 68** tsLibGUI/tsWxPkg/src/tsWxShowEvent.py (Available)
- 69** tsLibGUI/tsWxPkg/src/tsWxSize.py (Available)
- 70** tsLibGUI/tsWxPkg/src/tsWxSizer.py (Available)
- 71** tsLibGUI/tsWxPkg/src/tsWxSizerFlags.py (Available)
- 72** tsLibGUI/tsWxPkg/src/tsWxSizerItem.py (Available)
- 73** tsLibGUI/tsWxPkg/src/tsWxSizerItemList.py (Available)
- 74** tsLibGUI/tsWxPkg/src/tsWxSizerSpacer.py (Available)
- 75** tsLibGUI/tsWxPkg/src/tsWxSlider.py (Future)

- 76** tsLibGUI/tsWxPkg/src/tsWxSplashScreen.py (Available)
- 77** tsLibGUI/tsWxPkg/src/tsWxStaticBox.py (Available)
- 78** tsLibGUI/tsWxPkg/src/tsWxStaticBoxSizer.py (Available)
- 79** tsLibGUI/tsWxPkg/src/tsWxStaticLine.py (Available)
- 80** tsLibGUI/tsWxPkg/src/tsWxStaticText.py (Available)
- 81** tsLibGUI/tsWxPkg/src/tsWxStatusBar.py (Available)
- 82** tsLibGUI/tsWxPkg/src/tsWxSystemSettings.py (Available)
- 83** tsLibGUI/tsWxPkg/src/tsWxTaskBar.py (Available)
- 84** tsLibGUI/tsWxPkg/src/tsWxTextCtrl.py (Available)
- 85** tsLibGUI/tsWxPkg/src/tsWxTimer.py (Future)
- 86** tsLibGUI/tsWxPkg/src/tsWxToggleButton.py (Future)
- 87** tsLibGUI/tsWxPkg/src/tsWxTopLevelWindow.py (Available)
- 88** tsLibGUI/tsWxPkg/src/tsWxValidator.py (Future)
- 89** tsLibGUI/tsWxPkg/src/tsWxWindow.py (Available)

Draft

7 APPENDIXES (System Specification)

Appendixes may be used to provide information separately for convenience in document main (e.g., charts, classified data). As applicable, each appendix shall be refer in the main body of the document where the data would normally have been provided. Appendixes may be bound as separate documents for ease in handling. Appendixes shall be lettered alphabetically (A, B, etc.).

Draft

Draft

8 APPENDIX A - REQUIRED STATES AND MODES

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

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

- 1 **Required States** (on page 49)
- 2 **Required Modes** (on page 51)

8.1 Required States

8.1.1 Pre-Startup State

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

- b) Establish the location and size of the local or remote terminal screen

8.1.2 Startup State

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

Excerpt From Wikipedia, the free encyclopedia

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

8.1.3 Operating State

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

8.1.4 Shutdown State

- 1 Upon the occurrence of a non-recoverable malfunction or operator shutdown command, the computer hardware shall display a diagnostic or shutdown message.
- 2 The computer shall terminate any running application software.

- 3 The computer shall terminate any running operating system software.
- 4 The computer shall halt pending the next power-on.

8.2 Required Modes

8.2.1 Video Adapter Modes

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

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

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

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

8.2.1.1 Text Mode

Excerpt From Wikipedia, the free encyclopedia

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

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

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

8.2.1.2 Pixel Mode

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

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

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

Draft

8.2.2 User Interface Modes

8.2.2.1 Command Line Mode

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

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

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

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

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

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

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

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

Usage

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

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

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

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

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

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

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

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

8.2.2.2 Graphical Mode

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

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

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

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

8.2.2.2.1 wxPython

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

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

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

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

Draft

Draft

9 APPENDIX B - SYSTEM CAPABILITY REQUIREMENTS

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

The system hardware and software (see *APPENDIX J - COMPUTER RESOURCE REQUIREMENTS* (on page 251) for configuraton usability considerations) shall provide the means for developing, documenting, enhancing, maintaining, supporting, troubleshooting and using the "tsWxGTUI_PyVx" Toolkit, and those application programs created with it, with different types of computer hardware and/or with different software packages:

- 1 *Hardware Capabilities* (on page 137)
- 2 *Software Capabilities* (on page 147)
- 3 *Platform Interface Capability* (on page 186)
- 4 *Application Programming Interface Capability* (on page 186)
- 5 *Operator Interface Capability* (on page 198)

9.1 Hardware Capabilities

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

- 1 *Central Processing Unit Capability* (on page 138)
- 2 *Random Access Memory Capability* (on page 138)
- 3 *Non-Volatile Storage Capability* (on page 138)
- 4 *Network Interface Device Capability* (on page 138)
- 5 *Keyboard Device Capability* (on page 138)
- 6 *Pointing Device Capability* (on page 139)
- 7 *Display Device Capability* (on page 139)
- 8 *Printer Device Capability* (on page 147)

9.1.1 Central Processing Unit Capability

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

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

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

9.1.2 Random Access Memory Capability

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

9.1.3 Non-Volatile Storage Capability

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

9.1.4 Network Interface Device Capability

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

9.1.5 Keyboard Device Capability

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

9.1.6 Pointing Device Capability

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

9.1.7 Display Device Capability

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

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

7: COLOR_WHITE

}

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

{

0: COLOR_BLACK,

1: COLOR_RED,

2: COLOR_GREEN,

3: COLOR_YELLOW,

4: COLOR_BLUE,

5: COLOR_MAGENTA,

6: COLOR_CYAN,

7: COLOR_WHITE

}

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

{

0: COLOR_BLACK,

1: COLOR_RED,

2: COLOR_GREEN,

3: COLOR_YELLOW,

4: COLOR_BLUE,

5: COLOR_MAGENTA,

6: COLOR_CYAN,

7: COLOR_WHITE,

8: COLOR_GRAY,

9: COLOR_MAROON,

10: COLOR_LIME_GREEN,

11: COLOR_OLIVE,

12: COLOR_NAVY,

13: COLOR_PURPLE,

14: COLOR_TEAL,

15: COLOR_SILVER

}

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

{

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

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


```
67: COLOR_VIOLET,  
68: COLOR_VIOLET_RED,  
69: COLOR_WHEAT,  
70: COLOR_YELLOW_GREEN  
}
```

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

```
{  
0: COLOR_BLACK,  
1: COLOR_RED,  
2: COLOR_GREEN,  
3: COLOR_YELLOW,  
4: COLOR_BLUE,  
5: COLOR_MAGENTA,  
6: COLOR_CYAN,  
7: COLOR_WHITE,  
8: COLOR_GRAY,  
9: COLOR_MAROON,  
10: COLOR_LIME_GREEN,  
11: COLOR_OLIVE,  
12: COLOR_NAVY,  
13: COLOR_PURPLE,  
14: COLOR_TEAL,  
15: COLOR_SILVER,  
16: COLOR_AQUAMARINE,  
17: COLOR_BLUE_VIOLET,  
18: COLOR_BROWN,  
19: COLOR_CADET_BLUE,  
20: COLOR_CORAL,  
21: COLOR_CORNFLOWER_BLUE,  
22: COLOR_DARK_GRAY,  
23: COLOR_DARK_GREEN,  
24: COLOR_DARK_OLIVE_GREEN,  
25: COLOR_DARK_ORCHID,
```

26: COLOR_DARK_SLATE_BLUE,
27: COLOR_DARK_SLATE_GRAY,
28: COLOR_DARK_TURQUOISE,
29: COLOR_DIM_GRAY,
30: COLOR_FIREBRICK,
31: COLOR_FOREST_GREEN,
32: COLOR_GOLD,
33: COLOR_GOLDENROD,
34: COLOR_GREEN_YELLOW,
35: COLOR_INDIAN_RED,
36: COLOR_KHAKI,
37: COLOR_LIGHT_BLUE,
38: COLOR_LIGHT_GRAY,
39: COLOR_LIGHT_STEEL_BLUE,
40: COLOR_MEDIUM_AQUAMARINE,
41: COLOR_MEDIUM_BLUE,
42: COLOR_MEDIUM_FOREST_GREEN,
43: COLOR_MEDIUM_GOLDENROD,
44: COLOR_MEDIUM_ORCHID,
45: COLOR_MEDIUM_SEA_GREEN,
46: COLOR_MEDIUM_SLATE_BLUE,
47: COLOR_MEDIUM_SPRING_GREEN,
48: COLOR_MEDIUM_TURQUOISE,
49: COLOR_MEDIUM_VIOLET_RED,
50: COLOR_MIDNIGHT_BLUE,
51: COLOR_ORANGE,
52: COLOR_ORANGE_RED,
53: COLOR_ORCHID,
54: COLOR_PALE_GREEN,
55: COLOR_PINK,
56: COLOR_PLUM,
57: COLOR_SALMON,
58: COLOR_SEA_GREEN,
59: COLOR_SIENNA,

60: COLOR_SKY_BLUE,
61: COLOR_SLATE_BLUE,
62: COLOR_SPRING_GREEN,
63: COLOR_STEEL_BLUE,
64: COLOR_TAN,
65: COLOR_THISTLE,
66: COLOR_TURQUOISE,
67: COLOR_VIOLET,
68: COLOR_VIOLET_RED,
69: COLOR_WHEAT,
70: COLOR_YELLOW_GREEN,
71: COLOR_ALICE_BLUE,
72: COLOR_ANTIQUE_WHITE,
73: COLOR_AZURE,
74: COLOR_BEIGE,
75: COLOR_BISQUE,
76: COLOR_BLANCHED_ALMOND,
77: COLOR_BURLYWOOD,
78: COLOR_CHARTREUSE,
79: COLOR_CHOCOLATE,
80: COLOR_CORNSILK,
81: COLOR_CRIMSON,
82: COLOR_DARK_BLUE,
83: COLOR_DARK_CYAN,
84: COLOR_DARK_GOLDENROD,
85: COLOR_DARK_KHAKI,
86: COLOR_DARK_MAGENTA,
87: COLOR_DARK_ORANGE,
88: COLOR_DARK_RED,
89: COLOR_DARK_SALMON,
90: COLOR_DARK_SEA_GREEN,
91: COLOR_DARK_VIOLET,
92: COLOR_DEEP_PINK,

93: COLOR_DEEP_SKY_BLUE,
94: COLOR_DODGER_BLUE,
95: COLOR_FLORAL_WHITE,
96: COLOR_GAINSBORO,
97: COLOR_GHOST_WHITE,
98: COLOR_HONEYDEW,
99: COLOR_HOT_PINK,
100: COLOR_INDIGO,
101: COLOR_IVORY,
102: COLOR_LAVENDER,
103: COLOR_LAVENDER_BLUSH,
104: COLOR_LAWN_GREEN,
105: COLOR_LEMON_CHIFFON,
106: COLOR_LIGHT_CORAL,
107: COLOR_LIGHT_CYAN,
108: COLOR_LIGHT_GOLDENROD_YELLOW,
109: COLOR_LIGHT_GREEN,
110: COLOR_LIGHT_PINK,
111: COLOR_LIGHT_SALMON,
112: COLOR_LIGHT_SEA_GREEN,
113: COLOR_LIGHT_SKY_BLUE,
114: COLOR_LIGHT_SLATE_GRAY,
115: COLOR_LIGHT_YELLOW,
116: COLOR_LINEN,
117: COLOR_MEDIUM_PURPLE,
118: COLOR_MINT_CREAM,
119: COLOR_MISTY_ROSE,
120: COLOR_MOCCASIN,
121: COLOR_NAVAJO_WHITE,
122: COLOR_OLD_LACE,
123: COLOR_OLIVE_DRAB,
124: COLOR_PALE_GOLDENROD,
125: COLOR_PALE_TURQUOISE,
126: COLOR_PALE_VIOLET_RED,

```

127: COLOR_PAPAYA_WHIP,
128: COLOR_PEACH_PUFF,
129: COLOR_PERU,
130: COLOR_POWDER_BLUE,
131: COLOR_ROSY_BROWN,
132: COLOR_ROYAL_BLUE,
133: COLOR_SADDLE_BROWN,
134: COLOR_SANDY_BROWN,
135: COLOR_SEA_SHELL,
136: COLOR_SLATE_GRAY,
137: COLOR_SNOW,
138: COLOR_TOMATO,
139: COLOR_WHITE_SMOKE
}

```

9.1.8 Printer Device Capability

The system hardware shall optionally provide an electro-mechanical device that outputs alpha-numeric, graphic and control data to individual sheets of paper via Laser or Ink (applied via impact or spray).

9.2 Software Capabilities

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

- 1** *Host Computer Operating System Capability* (on page 148)
- 2** *Command Line Interface Capability* (on page 150)
- 3** *Graphical-Style User Interface Capability* (on page 151)
- 4** *Text Editing Capability* (on page 151)
- 5** *Word Processor Capability* (on page 152)
- 6** *Python Programming Capability* (on page 153)
- 7** *Debugging Capability* (on page 152)
- 8** *Host Console Terminal Application Capability* (on page 154)
- 9** *Optional Terminal Device Emulation Capability* (on page 155)
- 10** *"tsWxGTUI_PyVx" Toolkit Capability* (see *"tsWxGTUI" Toolkit Capability* on page 155)
- 11** *Platform Interface Capability* (on page 186)

12 *Application Programming Interface Capability* (on page 186)

13 *Operator Interface Capability* (on page 198)

9.2.1 Host Computer Operating System Capability

The system software shall provide any of the popular 32-/64-bit multi-user, multi-process and multi-tased/multi-threaded operating system kernels with associated device drivers and run time libraries.

1 Cygwin (1.7.28)

- a) A free, Linux-style Windows 32-/64-bit add-on from Red Hat.

2 GNU/Linux

- a) CentOS (7.0)

A distribution derived from the same sources used by Red Hat, maintained by a dedicated volunteer community of developers with both 100% Red Hat-compatible versions and an upgraded version that is not always 100% upstream compatible.

- b) Debian (8)

A non-commercial distribution and one of the earliest, maintained by a volunteer developer community with a strong commitment to free software principles and democratic project management

- c) Fedora (16-21)

A community distribution sponsored by American company Red Hat.

- d) OpenSUSE (13.1)

A community distribution mainly sponsored by German company SUSE.

SUSE Linux Enterprise, derived from openSUSE, is maintained and commercially supported by SUSE.

- e) Red Hat Enterprise Linux

A derivative of Fedora, maintained and commercially supported by Red Hat

- f) Scientific (6.5-7.0)

A Linux distribution produced by Fermi National Accelerator Laboratory. It is a free and open source operating system based on Red Hat Enterprise Linux and aims to be "as close to the commercial enterprise distribution as we can get it".

This product is derived from the free and open source software made available by Red Hat, Inc., but is not produced, maintained or supported by Red Hat. Specifically, this product is built from the source code for Red Hat Enterprise Linux versions, under the terms and conditions of Red Hat Enterprise Linux's EULA and the GNU General Public License.

- g) Ubuntu (2.04 LTS & 14.04 LTS)

A popular desktop and server distribution derived from Debian, maintained by British company Canonical Ltd.

3 OS X (formerly known as Mac OS X) (10.3-10.10)

A series of Unix-based graphical interface operating systems developed and marketed by Apple Inc. It is designed to run on Mac computers.

Versions 10.5 "Leopard" running on Intel processors, 10.6 "Snow Leopard", 10.7 "Lion", 10.8 "Mountain Lion", 10.9 "Mavericks", and 10.10 "Yosemite" have obtained UNIX 03 certification.

iOS, which runs on the iPhone, iPod Touch, iPad, and the 2nd and 3rd generation Apple TV, shares the Darwin core and many frameworks with OS X.

- a) Panther (10.3)
- b) Tiger (10.4)
- c) Leopard (10.5)
- d) Snow Leopard (10.6)
- e) Lion (10.7)
- f) Mountain Lion (10.8)
- g) Mavericks (10.9)
- h) Yosemite (10.10)

4 Microsoft Windows (XP, Vista, 7, 8, 8.1)

A metafamily of graphical operating systems developed, marketed, and sold by Microsoft. It consists of several families of operating systems, each of which cater to a certain sector of the computing industry. Active Windows families include Windows NT, Windows Embedded and Windows Phone; these may encompass subfamilies, e.g. Windows Embedded Compact (Windows CE) or Windows Server. Defunct Windows families include Windows 9x and Windows Mobile.

Microsoft Windows will only require "Cygwin", the free Linux-like plug-in from Red Hat for users of the "wxPython"-style, "nCurses"-based Graphical-Text User Interface. (Home or Professional editions of Windows XP, 7, 8, 8.1)

Microsoft introduced an operating environment named Windows on November 20, 1985 as a graphical operating system shell for MS-DOS in response to the growing interest in graphical user interfaces (GUIs).

Microsoft Windows came to dominate the world's personal computer market with over 90% market share, overtaking Mac OS, which had been introduced in 1984. However, it is outsold by Android on smartphones and tablets.

Standard "Command Prompt" and "PowerShell" accessories only support Command Line Interface mode.

Optional "Cygwin" add-on provides support for the character-mode, Graphical-style User Interface via shells, such as "bash", and the POSIX-compatible Application Programming Interface to the "curses" terminal control library for Unix-like systems.

- a) XP (NT 5.1)
- b) Vista (NT 6.0)
- c) 7 (NT 6.1)
- d) 8.0 (NT 6.2)

- e) 8.1 (NT 6.3)

5 Unix

A multitasking, multiuser computer operating system that exists in many variants. The original Unix was developed at AT&T's Bell Labs research center by Ken Thompson, Dennis Ritchie, and others. From the power user's or programmer's perspective, Unix systems are characterized by a modular design that is sometimes called the "Unix philosophy," meaning the OS provides a set of simple tools that each perform a limited, well-defined function, with a unified filesystem as the main means of communication and a shell scripting and command language to combine the tools to perform complex workflows.

- a) FreeBSD (9.2, 10.0)

A free Unix-like operating system descended from Research Unix via the Berkeley Software Distribution (BSD). Although for legal reasons FreeBSD cannot use the Unix trademark, it is a direct descendant of BSD, which was historically also called "BSD Unix" or "Berkeley Unix."

- b) OpenIndiana (151a8)

A free and open-source, Unix operating system derived from OpenSolaris. Developers forked OpenSolaris after Oracle Corporation discontinued it, in order to continue development and distribution of the source code. The OpenIndiana project is stewarded by the illumos Foundation, which also stewards the illumos operating system. OpenIndiana's developers strive to make it "the defacto OpenSolaris distribution installed on production servers where security and bug fixes are required free of charge".

- c) OpenSolaris (11)

A descendant of the UNIX System V Release 4 (SVR4) code base developed by Sun and AT&T in the late 1980s. It is the only version of the System V variant of UNIX available as open source.

- d) PC-BSD, or PCBSD (10)

A Unix-like, desktop-oriented operating system built upon the most recent releases of FreeBSD. It aims to be easy to install by using a graphical installation program, and easy and ready-to-use immediately by providing KDE SC, LXDE, Xfce, and MATE as the graphical user interface.

9.2.2 Command Line Interface Capability

The system software shall provide a command-line interface (CLI), also known as command-line user interface, console user interface, and character user interface (CUI), as a means of interacting with a computer program where the user issues commands to the program in the form of successive lines of text (command lines).

See: *Console Terminal Application Capability* (see "*Host Console Terminal Application Capability*" on page 154)

9.2.3 Graphical-Style User Interface Capability

The system software shall provide a graphical user interface (GUI) as a type of user interface that allows users to interact with electronic devices through graphical icons, buttons, check boxes, gauges, sliders, scrollbars and other visual indicators, as opposed to text-based interfaces, typed command labels or text navigation. GUIs were introduced in reaction to the perceived steep learning curve of command-line interfaces (CLI), which require commands to be typed on the keyboard.

1 Linux

GNOME - A desktop environment which runs on multiple operating systems, its main focus being those based on the Linux kernel.

KDE - A desktop environment designed to run on Linux, FreeBSD, Solaris, Microsoft Windows, and OS X systems.

X11 - A windowing system for bitmap displays, common on UNIX-like operating systems.

2 Mac OS X

Apple Inc. proprietary.

X11 - A windowing system for bitmap displays, common on UNIX-like operating systems.

3 Microsoft Windows

Microsoft Corporation proprietary.

4 Unix

GNOME - A desktop environment which runs on multiple operating systems, its main focus being those based on the Linux kernel.

KDE - A desktop environment designed to run on Linux, FreeBSD, Solaris, Microsoft Windows, and OS X systems.

X11 - A windowing system for bitmap displays, common on UNIX-like operating systems.

9.2.4 Text Editing Capability

The system software shall provide a tool to create and modify files that use a simple character set, such as ASCII, to represent numbers, letters, and a small number of symbols. The only non-printing characters in the file that can be used to format the text, are newline, tab, and formfeed.

1 Linux

GNU Emacs/XEmacs

vim

2 Mac OS X

TextEdit

TextWrangler

3 Microsoft Windows

Notepad

XEmacs

4 Unix

GNU Emacs/XEmacs

vim

9.2.5 Word Processor Capability

The system software shall provide a tool to create and modify files that contain formatted text, such as enabling text to appear in boldface and italics, that use multiple fonts, and that can be structured into columns and tables. These capabilities were once associated only with desktop publishing, but are now available in the simplest word processor.

1 Linux

LibreOffice

2 Mac OS X

LibreOffice

Microsoft Office

3 Microsoft Windows

Author-it

LibreOffice

Microsoft Office

4 Unix

LibreOffice

9.2.6 Debugging Capability

The system software shall provide a tool used to debug computer programs. The tool typically displays the lines of source code. It enables the user to set break points to trace the execution path. It enables the user to display the current value of constants and variables.

1 Linux

GNU gdb

Python Idle

WingIDE Pro

2 Mac OS X

GNU gdb

Python Idle

WingIDE Pro

3 Microsoft Windows

GNU gdb

Python Idle

WingIDE Pro

4 Unix

GNU gdb

Python Idle

9.2.7 Python Programming Capability

For those applications that require it, the system software shall provide tools to execute and troubleshoot source code components written in application-specific releases of:

1 A default third-generation Python programming language.

Python 3.2.0-3.2.5

NOTE: It may not be trivial or even practical to back port and re-engineer some of the functional and interface capabilities to Python 3.0-3.1 from those available in Python 3.2.0-3.2.5.

It may be trivial to re-engineer some of the functional and interface capabilities to Python 3.3 -3.4 from those available in Python 3.2.0-3.2.5.

2 An default second-generation Python programming language.

Python 2.6.6-2.7.6

NOTE: It may not be trivial or even practical to back port and re-engineer some of the functional and interface capabilities to Python 2.0-2.6.5 from those available in 2.6.6-2.7.6.

3 An optional first-generation Python programming language.

Python 1.6.x

NOTE: It may be quite a challenge to back port and re-engineer some of the functional and interface capabilities to Python 1.6.x from those available in 2.6.6-2.7.6.

9.2.8 Host Console Terminal Application Capability

For those applications that require it, the system software shall use the system's Terminal Control Library Interface and provide application tools to emulate various system console hardware terminal features.

Each of the following application or utility programs provides its own collection of user selectable preferences for setting cursor type, font type/size, color palette, blinking cursor/text, default terminal emulator type etc.

1 GNU/Linux

GNOME Terminal

KDE Terminal

2 Mac OS X

iTerm --- Third-party utility, organizes its preferences into the General, Appearance, Profiles, Keys, Pointer and Arrangements tabs.

Terminal --- Utility similar to ones available on GNU/Linux and other Unix systems. It organizes its preferences into the General, Profiles, Window Groups and Encodings tabs.

3 Microsoft Windows *(using Cygwin, the free add-on from Red Hat, that provides a Linux-like Command Line Interface and GNU Toolkit)*

Cygwin Terminal (mintty)

4 Microsoft Windows *(Native, without using Cygwin, the free add-on from Red Hat, that provides a Linux-like Command Line Interface and GNU Toolkit)*

Command Prompt (supports a non-Linux/non-Unix Command Line Interface that does NOT support Python Curses and its character-mode Graphical-style User Interface)

Excerpt from http://en.wikipedia.org/wiki/Command_Prompt:

"Command Prompt (executable name cmd.exe) is the Microsoft-supplied command-line interpreter on OS/2, Windows CE and on Windows NT-based operating systems (including Windows 2000, XP, Vista, 7, 8, Server 2003, Server 2008, Server 2008 R2 and Server 2012). It is the analog of COMMAND.COM in MS-DOS and Windows 9x systems (where it is called "MS-DOS Prompt"), or of the Unix shells used on Unix-like systems."

Windows PowerShell (supports a non-Linux/non-Unix Command Line Interface that does NOT support Python Curses and its character-mode Graphical-style User Interface)

Excerpt from http://en.wikipedia.org/wiki/Windows_PowerShell:

"Windows PowerShell is a task automation and configuration management framework from Microsoft, consisting of a command-line shell and associated scripting language built on the .NET Framework. PowerShell provides full access to COM and WMI, enabling administrators to perform administrative tasks on both local and remote Windows systems as well as WS-Management and CIM enabling management of remote Linux systems and network devices."

5 Unix

Terminal

See *Optional Terminal Emulation Capability* (see "*Optional Terminal Device Emulation Capability*" on page 155) for additional terminal hardware emulation requirements.

9.2.9 Optional Terminal Device Emulation Capability

For those applications that require it, the system software shall enhance the *Host Console Terminal Application Capability* (on page 154) with one or more of the following optional terminal emulation features:

9.2.10 "tsWxGTUI" Toolkit Capability

This paragraph shall identify a required system capability and shall itemize the requirements associated with the capability. If the capability can be more clearly specified by dividing it into constituent capabilities, the constituent capabilities shall be specified in subparagraphs. The requirements shall specify required behavior of the system and shall include applicable parameters, such as response times, throughput times, other timing constraints, sequencing, accuracy, capacities (how much/how many), priorities, continuous operation requirements, and allowable deviations based on operating conditions. The requirements shall include, as applicable, required behavior under unexpected, unallowed, or "out of bounds" conditions, requirements for error handling, and any provisions to be incorporated into the system to provide continuity of operations in the event of emergencies. Paragraph 3.3.x of this DID provides a list of topics to be considered when specifying requirements regarding inputs the system must accept and outputs it must produce.

9.2.10.1 "tsToolkitCLI" Command Line Interface Capability

The "tsWXTGUI" Toolkit software shall provide a library of building blocks and application programs that support the development of non-graphical applications. Such applications are invoked via textual commands entered via the console keyboard. Output is viewed on the console display.

9.2.10.1.1 CLI Terminal Interface Capability

The "tsWXTGUI" Toolkit software shall provide the following:

- 1 Hardware** - Terminal for input from operator (such as an electronic keyboard) and hardware for output to operator (such as an electronic display or printer).
- 2 Software** - Shell application (typically provided by the operating system) to receive, interpret and process command input from operator that cause actions which gather, process, format and output information to operator. It is a command processor that's typically run in a one dimensional text window (on the line displaying the command prompt), allowing the user to type commands which cause actions. It can also read commands from a file, called a script.

Command Line Interface mode input/output streams:

- **stdin** - POSIX-type standard input stream of one or more characters from an operator's keyboard or redirected input from a file or pipe (connection with the output stream from another running program).
- **stdout** - POSIX-type standard output stream of one or more characters to an operator's display or redirected output to a file or pipe (connection with the input stream of another running program).
- **stderr** - POSIX-type standard error output stream of one or more characters to an operator's display, system administrator's display or redirected output to a file or pipe (connection with the input stream of another running program).

Upon successfully launching the Graphical-style User Interface mode, and until terminating it, the following input/output streams become available:

- **stdscr** - Curses-type standard screen output stream of one or more characters to an operator's display. Before an application can use this stream, the application must initialize curses. This is done by calling the curses module's `initscr()` function, which will determine the terminal type, send any required setup codes to the terminal, and create various internal data structures. If successful, `initscr()` returns a window object representing the entire screen; this is usually called `stdscr`, after the name of the corresponding C variable.
- **stdscr.getch** - The `getch()` method returns an integer; if it's between 0 and 255, it represents the ASCII code of the key pressed. Values greater than 255 are special keys such as Page Up, Home, or the cursor keys. A value of -1 represents an input timeout. You can compare the value returned to constants such as `curses.KEY_PPAGE`, `curses.KEY_HOME`, or `curses.KEY_LEFT`. You can also compare the value returned to constants such as `curses.BUTTON1_PRESSED`, `curses.BUTTON1_RELEASED`, `curses.BUTTON1_CLICKED`, `curses.BUTTON1_DOUBLE_CLICKED` and `curses.BUTTON1_TRIPLE_CLICKED`.
- **stdscr.getstr** - The `getstr()` method retrieves an entire string. It isn't used very often, because its functionality is quite limited; the only editing keys available are the backspace key and the Enter key, which terminates the string. It can optionally be limited to a fixed number of characters.

9.2.10.1.2 "tsLibCLI" Capability

The "tsWXTGUI" Toolkit software shall provide the following:

- 1 **tsApplicationPkg** - Python base class to initialize and configure the application program launched by an operator. It enables an application launched via a Command Line Interface (CLI) to initialize, configure and use the same character-mode terminal with a Graphical-style User Interface (GUI).

 - a) Input provided on the command line by an operator. The command line uses a Unix-style, free-format to promote future enhancement and on-going maintenance.
 - b) Input provided in the parameter list of the application's invocation of the class instantiation. The parameter list uses a Python-style free-format to promote future enhancement and on-going maintenance.

- 2 **tsCommandLineEnvPkg** - Python class to initialize and configure the application program launched by an operator. It delivers those keyword-value pair options and positional arguments specified by the application, in its invocation parameter list. It wraps the Command Line Interface application with exception handlers to control exit codes and messages that may be used to co-ordinate other application programs..

- 3 tsCommandLineInterfacePkg** - Python class establishes methods that prompt or re-prompt the operator for input, validate that the operator has supplied the expected number of inputs and that each is of the expected type.
- 4 tsCxGlobalsPkg** - Python class to provide a theme-based centralized mechanism for modifying/restoring those configuration constants that can be interrogated at runtime by those software components having a "need-to-know". The intent being to avoid subsequent searches to locate and modify or restore a constant appropriate to the current configuration, users and their activities.
- 5 tsExceptionPkg** - Python class to define and handle error exceptions. It maps various error exceptions into an 8-bit exit code and message that can be used to coordinate the actions of a set of shell scripts.
- 6 tsLoggerPkg** - Python class to define and handle event message timestamping, formatting and output. It also supports logging of assert and check case results.

It supports the following message severity levels: NOTSET (lowest priority), DEBUG, INFO, NOTICE, WARNING, ALERT, ERROR, CRITICAL, EMERGENCY (highest priority) and PRIVATE.

It defines and handles event message processing associated with the following: wxPython/wxWidget exceptions: wxASSERT, wxASSERT_MSG, wxCHECK, wxCHECK2, wxCHECK2_MSG, wxCHECK_MSG, wxCHECK_RET, wxFAIL, wxFAIL_COND_MSG, wxFAIL_MSG and wxTRAP.

- 7 tsOperatorSettingsParserPkg** - Class to parse command line options and arguments.

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

- a) "argparse" (introduced with Python 2.7.0)
- b) "optparse" (introduced with Python 2.3.0)
- c) "getopt" (introduced with Python 1.6.0)

When used with Python 2.7 or Python 3.2, the "tsOperatorSettings.py" module (a "tsLibCLI" component of the "tsWxGTUI_PyVx" Toolkit) supports the current and legacy Python version specific parser module(s) as an experimental and educational opportunity.

However, when one seeks to back port applications to Python 2.0-2.6 or Python 3.0-3.1, the "tsOperatorSettings.py" module must be stripped of unsupported Python version specific parser modules in order to prevent a program trap which will block the application from running.

- 8 tsPlatformRunTimeEnvironmentPkg** - Class to capture current hardware and software information about the run time environment for the user process.
- 9 tsReportUtilityPkg**- Class defining methods used to format information such as time, date, data size, elapsed time and nested dictionary contents.
- 10 tsSysCommandsPkg** - Class definition and methods for issuing shell commands to the host operating system.

9.2.10.1.3 "tsToolsCLI" Capability

The "tsWXTGUI" Toolkit software shall provide the following:

- 1 **tsLinesOfCodeProjectMetrics** - Python application program, with a Command Line Interface (CLI), that generates reports of software project progress and the estimated cost (or contributed value) of the project when it is finally completed.

- a) It scans an operator designated file directory tree containing the source files, in one or more programming language specific formats (such as Ada, Assembler, C/C++, Cobol, Fortran, PL/M, Python, Text, and various command line shells).

For each file, it accumulates and reports the total number of code lines, blank/comment lines, words and characters.

For each programming language format, it accumulates and reports a summary of details of the associated source files.

For the entire set of source files, it accumulates and reports a summary of details.

It uses the summary of the entire set of source files to derive, analyze, estimate and report metrics for the software development project (such as labor, cost, schedule and lines of code per day productivity).

- b) It supports one or more of the parser module(s) available in the Python version(s) supported by the application.

"argparse" (introduced with Python 2.7.0)

"optparse" (introduced with Python 2.3.0)

"getopt" (introduced with Python 1.6.0)

When used with Python 2.7 or Python 3.2, the "tsOperatorSettings.py" module (a "tsLibCLI" component of the "tsWxGTUI_PyVx" Toolkit) supports the current and legacy Python version specific parser module(s) as an experimental and educational opportunity.

However, when one seeks to back port applications to Python 2.0-2.6 or Python 3.0-3.1, the "tsOperatorSettings.py" module must be stripped of unsupported Python version specific parser modules in order to prevent a program trap which will block the application from running.

- 2 **tsPlatformQuery** - Python application program, with a Command Line Interface (CLI), that captures current hardware and software information about the run time environment for the user process.

(a) Host processor hardware support includes various releases of x86, PowerPC, SPARC.

(b) Host operating system software support includes various releases of Cygwin, Linux ('Debian', 'Fedora', 'mandriva', 'redhat', 'Scientific / Centos', 'SuSE'), Mac OS X, Microsoft Windows ('XP', 'Vista', '7', '8', '8.1') and Unix ('FreeBSD / PC-BSD', 'IRIX', 'OpenIndiana', 'Solaris / SunOS').

(c) Host virtual machine software support includes various releases of Java and Python.

- 3 **tsStripComments** - Python application program, with a Command Line Interface (CLI), that transforms an annotated, development version of a directory of sub-directories and Python source files into an unannotated copy.

The copy is intended to conserve storage space when installed in an embedded system. The transformation involves stripping comments and doc strings by de-tokenizing a tokenized version of each Python source file. Non-Python files are trimmed of trailing whitespace.

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

Output from fixed format FORTRAN (F77) code is NOT corrected to ensure that each statement first character begins in column 7 and that each ampersand ("&") continuation character is in column 6.

- 5 tsTreeCopy** - Python application program, with a Command Line Interface (CLI), that copies the contents of a source directory to a target directory.
- 6 tsTreeTrimLines** - Python application program, with a Command Line Interface (CLI), that copies the contents of a source directory to a target directory after stripping superfuos white space (blanks) from end of each line.

9.2.10.1.4 "tsTestsCLI" Capability

This section is under construction.

9.2.10.2 "tsToolkitGUI" Graphical-style User Interface Capability

The "tsWXTGUI" Toolkit software shall provide a library of building blocks and application programs that support the development of graphical-style applications. Such applications are invoked via textual commands entered via the console keyboard and by graphical commands entered via mouse (cursor position and button click). Output is viewed on the console display.

9.2.10.2.1 GUI Terminal Interface Capability

The "tsWXTGUI" Toolkit software shall provide the following:

- 1 Hardware** - Terminal for input from operator (such as an electronic keyboard, mouse, trackball, touchpad or touchscreen) and hardware for output to operator (such as an electronic display or printer).
- 2 Software** - Shell application (typically provided by the operating system) to receive, interpret and process command input from operator that cause actions which gather, process, format and output information to operator. It is a command processor that's typically run in a two dimensional grapical screen, allowing the user to "click" on objects (such as buttons, check boxes, radio buttons, windows and dialogs) and type commands which cause actions. It can also read commands from a file, called a script.
 - **Platform Dependent API** - Software components that provide an Application Programming Interface that is uniquely customized for the platform's physical hardware and operating system software (such as Linux, Mac OS X and Windows).
 - **Terminal Independent API** - Software components of "nCurses" that provide a common Application Programming Interface regardless of a platform's physical hardware and operating system software.

- **wxPython API** - Software components of "tsWxGraphicalTestUserInterface" that provide a subset of the "wxPython" Application Programming Interface. It includes class definition and methods to initialize, configure, create GUI objects and shutdown the platform's nCurses-based terminal interface.

9.2.10.2.2 "tsLibGUI" Capability

The "tsWXTGUI" Toolkit software shall provide the following:

- **Desktop API** (on page 68) - Describes the host operating system-style Graphical User Interface features (such as multiple top-level windows, task bar and redirected stderr and stdout displays) that the Software Engineer can include for input and output interactions with the System OperatorSystem Operator.
- **High-Level Widget API** (on page 69) - Identifies the wxPython-style Graphical User Interface features (such as frames, dialogs and buttons) that the Software Engineer can include for input and output interactions with the System Operator.
- **Event API** (see "**Event API (Draft)**" on page 73) - Describes the basic and additional wxPython-style Graphical User Interface features used for sending, receiving and handling event notifications (such as clock tick, button clicked and shift in window focus).
- **Low-Level Widget API** (on page 91) - Identifies the basic nCurses-style Graphical User Interface features that the Software Engineer can use to construct High Level Widgets that customize input and output interactions with the local or remote terminal.
- **System Preference API** (see "**System Preferences API (Draft)**" on page 92) - Describes the wxPython-style Graphical User Interface features (such as Symbolic Constants, Foreground / Background Colors, Default Styles / Themes and utility functions) that the Software Engineer can use to standardize the look and feel of input and output interactions with the System Operator.

9.2.10.2.2.1 Desktop API

from http://en.wikipedia.org/wiki/Multiple_document_interface:

"Graphical computer applications with a multiple document interface (MDI) are those whose windows reside under a single parent window (usually except for modal windows), as opposed to all windows being separate from each other (single document interface). Such systems often allow child windows to embed other windows inside them as well, creating complex nested hierarchies. In the usability community, there has been much debate about which interface type is preferable. Generally, SDI is seen as more useful in cases where users work with more than one application. Software companies have used both interfaces with mixed responses. For example, Microsoft changed its Office applications from SDI to MDI mode and then back to SDI, although the degree of implementation varies from one component to another.

The disadvantage of MDI usually cited is the lack of information about the currently opened windows: In order to view a list of windows open in MDI applications, the user typically has to select a specific menu ("window list" or something similar), if this option is available at all. With an SDI application, the window manager's task bar or task manager (if any) can display the currently opened windows. In recent years, applications have increasingly added "task-bars" and "tabs" to show the currently opened windows in an MDI application, which has made this criticism somewhat obsolete. Some people call this interface "tabbed document interface" (TDI). When tabs are used to manage windows, individual ones usually cannot be resized or used simultaneously."

Multiple Document Interface capabilities currently include:

- 1 Screens** - A display (shown with black or white background) of a physical terminal device or a GUI object representing an emulated terminal whose size, position and font can (usually) be changed by the user. It usually has thick borders and a title bar, and can optionally contain a menu bar, toolbar and status bar. A screen can contain one or more frames and dialogs within which may be those GUI objects that are not frames or dialogs.
- 2 Multi-Frame Environment** - A non-wxPython feature consisting of one or more Frames (shown with blue background) and Dialogs (shown with cyan background) associated with the GUI application.
- 3 Redirected Output** - A non-wxPython feature consisting of a Frame (shown with green background) contains one or more lines of text messages. The message text is produced by the application's use of print or write statements with output normally intended for the "stderr" and "stdout" devices. The GUI toolkit intercepts the output and redirects it to a reserved frame on the display screen after annotating it with a date and time stamp.
- 4 Task Bar** - A non-wxPython feature consisting of a Frame (shown with black background) that monitors and controls the application tasks. The top-right line identifies the name of the application executable. The bottom-right line contains a character-mode symbol that appears to rotate. The rotation denotes progress. The middle line(s) contain buttons for each of the application's Frame and Dialog "tasks". If a mouse button is clicked when the cursor is over a button, an associated function or method will be initiated. The function or method will send a signal to notify other GUI objects of the event. The event will raise the focus of the selected task so that its frame or dialog becomes the top-most overlays.

NOTES:

The Desktop API includes a MultiFrame Environment that establishes the following Top Level Windows:

- 1) The Application Programmer designated Top Level Frame and Dialog, at top of screen.
 - 2) The Redirected I/O Frame provides the operator with the most recent stderr, stdout and print messages, in middle of screen.
 - 3) The Task Bar Frame provides the operator with a list of currently open frame and dialog windows from which to click on and change the focus, at bottom of screen.
-

9.2.10.2.2.2 High-Level Widget API

NOTES:

- 1) As a character-mode GUI-style toolkit, this product does not support those "wxPython" features associated with graphical elements such as bit images, icons, proportional-spaced fonts or HTML and XML text markup. The current release supports the porting of "wxPython" 2.8.9.2 application(s) to platforms running Python 2.5.x, 2.6.x, 2.7.x, 3.1.x and 3.2.x.
 - 2) Technical and resource issues drive the development effort. Initially, development focussed on establishing the feasibility of emulating core components of the the "wxPython" and associated "wxWidgets" API. Development then iteratively seeks to establish usability-enhancing components and to then to identify and resolve any appearance, behavior and API-conformance issues.
 - 3) See the unpublished "**tsWxGTUI_PyVx**" Vol. 7 - Design Notes (it was created only for personal use) for the identification and an overview of those currently implemented class modules, ones currently under construction and ones for which development applicability and/or commitments are still TBD.
-

The High-Level Widget API identifies the basic wxPython-style Graphical User Interface features that the Software Engineer can include for input and output interactions with the System Operator. It includes such widgets as the following:

- 1 Button** - GUI object that may contain text or character-mode icons. If a mouse button is clicked when the cursor is over the object, an associated function or method will be initiated. The function or method will send a signal to notify other GUI objects of the event. Buttons are clickable windows that trigger an associated event (such as start, pause, resume or terminate an operation).
- 2 Carat** - GUI object that may contain a set of special character-mode symbols, usually including a solid rectangle or a blinking underline character, showing the position where the typed text will appear.
- 3 Check Box** - GUI object that may contain text or character-mode icons. If a mouse button is clicked when the cursor is over the object, it initiates an associated function or method. The function or method will toggle the check box to the next "ON", "OFF" or "TRI-STATE" value. The function or method will send a signal to notify other GUI objects of the event. Checkboxes are buttons to toggle the enabled or disabled state of an associated feature (such as start or stop logging)
- 4 Cursor** - A special character-mode symbol, usually a solid rectangle or a blinking underline character, that signifies where the next character will be displayed on the screen.

- 5 Dialog** - A GUI object with a title bar and sometimes a system menu, which can be moved around the screen. It can contain controls and other GUI objects and is often used to allow the user to make some choice or to answer a question. Dialogs are pop-up windows associated with an application task's menu bar (such as an input field for an operator's search criteria and an output field for a list of candidate WEB sites) that will be terminated upon completion.
- 6 Frame** - GUI object whose size and position can (usually) be changed by the user. It usually has thick borders and a title bar, and can optionally contain a menu bar, toolbar and status bar. A frame can contain any GUI object that is not a frame or dialog. Frames are windows associated with an application task (such as an internet WEB Browser) that can be minimized into an icon, expanded to full screen or terminated upon operator demand.
- 7 Gauge** - GUI object whose horizontal or vertical bar shows a quantity (often time). It supports two working modes: determinate and indeterminate progress. First is the usual working mode while the second can be used when the program is doing some processing but you don't know how much progress is being done. In this case, you can periodically call the Pulse function to make the progress bar switch to indeterminate mode (graphically it's usually a set of blocks which move or bounce in the bar control). Gauges are used to indicate progress of an associated operation (scale with range such as 0% to %100%).
- 8 Menu** - GUI object that features a popup (or pull down) list of items, one of which may be selected before the menu goes away (clicking elsewhere dismisses the menu). It may be used to construct either menu bars or popup menus.
- 9 Menu Bar** - GUI object that features a series of menus accessible from the top of a frame. Each operator selectable entry triggers an associated event (such as create a file).
- 10 Panel** - GUI object that features a window on which controls are placed. Panels are usually placed within a frame. Its main feature over its parent window class is code for handling child windows and TAB traversal. Panels are container windows for other Lower Level GUI Objects.
- 11 Radio Box** - GUI objects that are used to select one of number of mutually exclusive choices. A radio box is displayed as a vertical column or horizontal row of labelled radio buttons. Radio Boxes are a collection of Buttons associated with the same group (such a AM or FM band) that are interdependent such that any one can become activated after the others simultaneously become deactivated.
- 12 Radio Button** - GUI object that may contain text or character-mode icons. If a mouse button is clicked when the cursor is over the object, it initiates an associated function or method. The function or method will turn the associated radio button "ON" and turn "OFF" all other radio buttons within the associated radio box. The function or method will send a signal to notify other objects of the event. Radio Buttons are used to activate one of the associated features (such as broadcast channel selection) after the other features associated with the same Radio Box group (such a AM or FM band) have become deactivated.
- 13 ScrollBar** - GUI object that includes a horizontal or vertical ScrollBarGauge between two ScrollBarButtons. The operator uses it to re-position (pan) the contents of an associated ScrolledText window.
- 14 ScrollBarButton** - GUI object that includes one arrow symbol ("<", ">", "^" or "V") that indicates the horizontal or vertical direction of scrolling. When the operator clicks on a ScrollBarButton, the contents of the associated ScrolledText window moves.

- A single Left Click on one of the two horizontal ScrollBarButtons moves the text by one column. A rapid double Left Click on one of the two horizontal ScrollBarButtons moves the ScrolledText by one page (the horizontal column width). A single Right Click on one of the two horizontal ScrollBarButtons moves the text to the appropriate horizontally end point (either left/right most column).
 - A single Left Click on one of the two vertical ScrollBarButtons moves the text by one row. A rapid double Left Click on one of the two vertical ScrollBarButtons moves the ScrolledText by one page (the vertical row height). A single Right Click on one of the two vertical ScrollBarButtons moves the text to the appropriate vertical end point (top/bottom most row).
- 15 ScrollBarGauge** - GUI object located between two ScrollBarButtons, contains a bar graph that indicates the position and size of the displayed text relative to the non-displayed text.
- When the bar graph is empty (blank), there is no text available to be displayed.
 - When it is completely filled, all of the available text is displayed.
 - When it is partially filled, the starting point of the graph displays the starting point of the displayed text relative to the available text. The size of the filled graph displays the size of the displayed text relative to the available text.
 - When the operator single Left Clicks on a ScrollBarGauge between its two associated ScrollBarButtons, the contents of the ScrolledText window moves in proportion to the difference between the click and the associated text end positions.
- 16 Scrolled** - An application-independent GUI object base class for ScrolledWindow. It lays out the position and size of one ScrolledText window and one or two ScrollBars each with their associated ScrollBarGauge and pair of ScrollBarButtons. It enables an operator to select the columns and or rows of text to be fit and displayed within the available peep hole viewing area.
- 17 ScrolledText** - GUI object that allows one or more lines of text to be appended to a retained list. In conjunction with one or two associated pairs of ScrollBarButtons, it enables an operator to select the columns and or rows of text to be fit and displayed within the available peep hole viewing area.
- 18 ScrolledWindow** - GUI object that instantiates an application-specific instance of Scrolled to lay out the position and size of one ScrolledText window and one or two ScrollBars each with their associated ScrollBarGauge and pair of ScrollBarButtons. It enables an operator to select the columns and or rows of text to be fit and displayed within the available peep hole viewing area.
- 19 Splash Screen** - GUI object that simulates a pixel-mode bitmap image which is displayed upon application startup. It may contain text or character-mode icons. It features a window with a thin border and text describing the application. The bitmap-like display may be created from Panel, BoxSizer, GridSizer and TextCtrl widgets. It is created and shown during application initialization, before the application's own window. It either explicitly destroys itself or disappears after a it time-out.
- 20 Status Bar** - GUI object that feature a narrow window that can be placed along the bottom of a frame to give small amounts of status information. The object can contain one or more fields, one or more of which can be variable length according to the size of the window.
- 21 Static Text** - GUI object that features a text control that displays one or more lines of read-only text.
- 22 Text Ctl** - GUI object that features a text control which allows text to be displayed and edited. It may be single line or multi-line. The text may be indented, line-wrapped and scrolled to fit the available display area.

23 Tool Bar (Future) - GUI object that features a series of tool entries accessible from the top of a frame. Each operator selectable entry triggers an associated event that starts the selected tool in a new Frame.

NOTES:

1) Event Handling support currently associates mouse clicks with the enclosing GUI object that is not obscured by overlaying GUI objects. The toolkit generates an event notification and dispatches it to the event handler designated by the application. It will dispatch unhandled event notifications to a default handler.

2) Keyboard Input Events are detected. The raw characters are echoed but are NOT currently forwarded to the window object having focus.

3) Mouse Input Events are detected. The x-y coordinates and button state are echoed. Single Left Clicks, Double Left Click and Right Clicks are forwarded to the window object positioned to receive and process the event. More complex GUI event detection, analysis and processing is not yet supported. Platform-specific vt100/vt220 terminal emulators do NOT conform to the xterm/xterm-color mouse click event protocol recognized by nCurses. Instead of a single mouse click event, the vt100/vt220 terminal emulators generate a string of escape sequence events. It is unknown if the "tsWxGTUI_PyVx" Toolkit can develop logic to synthesize an xterm/xterm-color mouse click event from the string of vt100/vt220 escape sequence events.

4) Automatic layout support is currently provided by the BoxSizer and GridSizer widgets or by combinations of them. More complex GUI object positioning and sizing is not yet supported.

9.2.10.2.2.3 Event API (Draft)

from http://docs.wxwidgets.org/trunk/overview_events.html

Introduction to Events

Like with all the other GUI frameworks, the control of flow in wxWidgets applications is event-based: the program normally performs most of its actions in response to the events generated by the user. These events can be triggered by using the input devices (such as keyboard, mouse, joystick) directly or, more commonly, by a standard control which synthesizes such input events into higher level events: for example, a wxButton can generate a click event when the user presses the left mouse button on it and then releases it without pressing Esc in the meanwhile. There are also events which don't directly correspond to the user actions, such as wxTimerEvent or wxSocketEvent.

But in all cases wxWidgets represents these events in a uniform way and allows you to handle them in the same way wherever they originate from. And while the events are normally generated by wxWidgets itself, you can also do this, which is especially useful when using custom events (see Custom Event Summary).

To be more precise, each event is described by:

Event type: this is simply a value of type wxEventType which uniquely identifies the type of the event. For example, clicking on a button, selecting an item from a list box and pressing a key on the keyboard all generate events with different event types.

Event class carried by the event: each event has some information associated with it and this data is represented by an object of a class derived from wxEvent. Events of different types can use the same event class, for example both button click and listbox selection events use wxCommandEvent class (as do all the other simple control events), but the key press event uses wxKeyEvent as the information associated with it is different.

Event source: wxEvent stores the object which generated the event and, for windows, its identifier (see Window Identifiers). As it is common to have more than one object generating events of the same type (e.g. a typical window contains several buttons, all generating the same button click event), checking the event source object or its id allows to distinguish between them.

Event Handling

There are two principal ways to handle events in wxWidgets. One of them uses event table macros and allows you to define the binding between events and their handlers only statically, i.e., during program compilation. The other one uses wxEvtHandler::Bind<>() call and can be used to bind and unbind, the handlers dynamically, i.e. during run-time depending on some conditions. It also allows the direct binding of events to:

A handler method in another object.

An ordinary function like a static method or a global function.

An arbitrary functor like boost::function<>.

The static event tables can only handle events in the object where they are defined so using `Bind<>()` is more flexible than using the event tables. On the other hand, event tables are more succinct and centralize all event handler bindings in one place. You can either choose a single approach that you find preferable or freely combine both methods in your program in different classes or even in one and the same class, although this is probably sufficiently confusing to be a bad idea.

Also notice that most of the existing wxWidgets tutorials and discussions use the event tables because they historically preceded the apparition of dynamic event handling in wxWidgets. But this absolutely doesn't mean that using the event tables is the preferred way: handling events dynamically is better in several aspects and you should strongly consider doing it if you are just starting with wxWidgets. On the other hand, you still need to know about the event tables if only because you are going to see them in many samples and examples.

So before you make the choice between static event tables and dynamically connecting the event handlers, let us discuss these two ways in more detail. In the next section we provide a short introduction to handling the events using the event tables. Please see [Dynamic Event Handling](#) for the discussion of `Bind<>()`.

Event Handling with Event Tables

To use an event table you must first decide in which class you wish to handle the events. The only requirement imposed by wxWidgets is that this class must derive from `wxEvtHandler` and so, considering that `wxWindow` derives from it, any classes representing windows can handle events. Simple events such as menu commands are usually processed at the level of a top-level window containing the menu, so let's suppose that you need to handle some events in `MyFrame` class deriving from `wxFrame`.

First define one or more event handlers. They are just simple methods of the class that take as a parameter a reference to an object of a `wxEvent`-derived class and have no return value (any return information is passed via the argument, which is why it is `non-const`). You also need to insert a macro

```
wxDECLARE_EVENT_TABLE()
```

somewhere in the class declaration. It doesn't matter where it appears but it's customary to put it at the end because the macro changes the access type internally so it's safest if nothing follows it. The full class declaration might look like this:

```
class MyFrame : public wxFrame
{
public:
    MyFrame(...) : wxFrame(...) { }
```

```
...
```

protected:

```
int m_whatever;
```

private:

```
// Notice that as the event handlers normally are not called from outside
```

```
// the class, they normally are private. In particular they don't need
```

```
// to be public.
```

```
void OnExit(wxCommandEvent& event);
```

```
void OnButton1(wxCommandEvent& event);
```

```
void OnSize(wxSizeEvent& event);
```

```
// it's common to call the event handlers OnSomething() but there is no
```

```
// obligation to do that; this one is an event handler too:
```

```
void DoTest(wxCommandEvent& event);
```

```
DECLARE_EVENT_TABLE()
```

```
};
```

Next the event table must be defined and, as with any definition, it must be placed in an implementation file. The event table tells wxWidgets how to map events to member functions and in our example it could look like this:

```
wxBEGIN_EVENT_TABLE(MyFrame, wxFrame)
```

```
EVT_MENU(wxID_EXIT, MyFrame::OnExit)
```

```
EVT_MENU(DO_TEST, MyFrame::DoTest)
```

```
EVT_SIZE(MyFrame::OnSize)
```

```
EVT_BUTTON(BUTTON1, MyFrame::OnButton1)
```

```
wxEND_EVENT_TABLE()
```

Notice that you must mention a method you want to use for the event handling in the event table definition; just defining it in MyFrame class is not enough.

Let us now look at the details of this definition: the first line means that we are defining the event table for MyFrame class and that its base class is wxFrame, so events not processed by MyFrame will, by default, be handled by wxFrame. The next four lines define bindings of individual events to their handlers: the first two of them map menu commands from the items with the identifiers specified as the first macro parameter to two different member functions. In the next one, EVT_SIZE means that any changes in the size of the frame will result in calling OnSize() method. Note that this macro doesn't need a window identifier, since normally you are only interested in the current window's size events.

The EVT_BUTTON macro demonstrates that the originating event does not have to come from the window class implementing the event table -- if the event source is a button within a panel within a frame, this will still work, because event tables are searched up through the hierarchy of windows for the command events. (But only command events, so you can't catch mouse move events in a child control in the parent window in the same way because wxMouseEvent doesn't derive from wxCommandEvent. See below for how you can do it.) In this case, the button's event table will be searched, then the parent panel's, then the frame's.

Finally, you need to implement the event handlers. As mentioned before, all event handlers take a wxEvent-derived argument whose exact class differs according to the type of event and the class of the originating window. For size events, wxSizeEvent is used. For menu commands and most control commands (such as button presses), wxCommandEvent is used. When controls get more complicated, more specific wxCommandEvent-derived event classes providing additional control-specific information can be used, such as wxTreeEvent for events from wxTreeCtrl windows.

In the simplest possible case an event handler may not use the event parameter at all. For example,

```
void MyFrame::OnExit(wxCommandEvent& WXUNUSED(event))
{
    // when the user selects "Exit" from the menu we should close
    Close(true);
}
```

In other cases you may need some information carried by the event argument, as in:

```
void MyFrame::OnSize(wxSizeEvent& event)
{
    wxSize size = event.GetSize();
```

... update the frame using the new size ...

```
}
```

You will find the details about the event table macros and the corresponding wxEvent-derived classes in the discussion of each control generating these events.

Dynamic Event Handling

The possibilities of handling events in this way are rather different. Let us start by looking at the syntax: the first obvious difference is that you need not use `DECLARE_EVENT_TABLE()` nor `BEGIN_EVENT_TABLE()` and the associated macros. Instead, in any place in your code, but usually in the code of the class defining the handler itself (and definitely not in the global scope as with the event tables), call its `Bind<>()` method like this:

```
MyFrame::MyFrame(...)
{
    Bind(wxEVT_COMMAND_MENU_SELECTED, &MyFrame::OnExit, this, wxID_EXIT);
}
```

Note that this pointer must be specified here.

Now let us describe the semantic differences:

Event handlers can be bound at any moment. For example, it's possible to do some initialization first and only bind the handlers if and when it succeeds. This can avoid the need to test that the object was properly initialized in the event handlers themselves. With `Bind<>()` they simply won't be called if it wasn't correctly initialized.

As a slight extension of the above, the handlers can also be unbound at any time with `Unbind<>()` (and maybe rebound later). Of course, it's also possible to emulate this behaviour with the classic static (i.e., bound via event tables) handlers by using an internal flag indicating whether the handler is currently enabled and returning from it if it isn't, but using dynamically bind handlers requires less code and is also usually more clear.

Almost last but very, very far from least is the increased flexibility which allows to bind an event to:

A method in another object.

An ordinary function like a static method or a global function.

An arbitrary functor like `boost::function<>`.

This is impossible to do with the event tables because it is not possible to specify these handlers to dispatch the event to, so it necessarily needs to be sent to the same object which generated the event. Not so with `Bind<>()` which can be used to specify these handlers which will handle the event. To give a quick example, a common question is how to receive the mouse movement events happening when the mouse is in one of the frame children in the frame itself. Doing it in a naive way doesn't work:

A `EVT_LEAVE_WINDOW(MyFrame::OnMouseLeave)` line in the frame event table has no effect as mouse move (including entering and leaving) events are not propagated up to the parent window (at least not by default).

Putting the same line in a child event table will crash during run-time because the `MyFrame` method will be called on a wrong object -- it's easy to convince oneself that the only object that can be used here is the pointer to the child, as `wxWidgets` has nothing else. But calling a frame method with the child window pointer instead of the pointer to the frame is, of course, disastrous.

However writing

```
MyFrame::MyFrame(...)
{
    m_child->Bind(wxEVT_LEAVE_WINDOW, &MyFrame::OnMouseLeave, this);
}
```

will work exactly as expected. Note that you can get the object that generated the event -- and that is not the same as the frame -- via `wxEvt::GetEventObject()` method of event argument passed to the event handler.

Really last point is the consequence of the previous one: because of increased flexibility of `Bind()`, it is also safer as it is impossible to accidentally use a method of another class. Instead of run-time crashes you will get compilation errors in this case when using `Bind()`.

Let us now look at more examples of how to use different event handlers using the two overloads of `Bind()` function: first one for the object methods and the other one for arbitrary functors (callable objects, including simple functions):

In addition to using a method of the object generating the event itself, you can use a method from a completely different object as an event handler:

```
void MyFrameHandler::OnFrameExit( wxCommandEvent & )
{
    // Do something useful.
}
```

```
MyFrameHandler myFrameHandler;
```

```
MyFrame::MyFrame()
```

```
{  
  
Bind( wxEVT_COMMAND_MENU_SELECTED, &MyFrameHandler::OnFrameExit,  
  
&myFrameHandler, wxID_EXIT );  
  
}
```

Note that MyFrameHandler doesn't need to derive from wxEvtHandler. But keep in mind that then the lifetime of myFrameHandler must be greater than that of MyFrame object -- or at least it needs to be unbound before being destroyed.

To use an ordinary function or a static method as an event handler you would write something like this:

```
void HandleExit( wxCommandEvent & )  
  
{  
  
    // Do something useful  
  
}
```

```
MyFrame::MyFrame()  
  
{  
  
    Bind( wxEVT_COMMAND_MENU_SELECTED, &HandleExit, wxID_EXIT );  
  
}
```

And finally you can bind to an arbitrary functor and use it as an event handler:

```
struct MyFunctor  
  
{  
  
    void operator()( wxCommandEvent & )  
  
    {  
  
        // Do something useful  
  
    }  
  
};
```

```
MyFunctor myFunctor;
```

```
MyFrame::MyFrame()
{
    Bind( wxEVT_COMMAND_MENU_SELECTED, &myFunctor, wxID_EXIT );
}
```

A common example of a functor is `boost::function`:

```
using namespace boost;
```

```
void MyHandler::OnExit( wxCommandEvent & )
{
    // Do something useful
}
```

```
MyHandler myHandler;
```

```
MyFrame::MyFrame()
{
    function< void ( wxCommandEvent & ) > exitHandler( bind( &MyHandler::OnExit, &myHandler, _1 ) );

    Bind( wxEVT_COMMAND_MENU_SELECTED, exitHandler, wxID_EXIT );
}
```

With the aid of `boost::bind` you can even use methods or functions which don't quite have the correct signature:

```
void MyHandler::OnExit( int exitCode, wxCommandEvent &, wxString goodByeMessage )
{
    // Do something useful
}
```

```
}
```

```
MyHandler myHandler;
```

```
MyFrame::MyFrame()
```

```
{
```

```
function< void ( wxCommandEvent & ) > exitHandler(
```

```
bind( &MyHandler::OnExit, &myHandler, EXIT_FAILURE, _1, "Bye" ));
```

```
Bind( wxEVT_COMMAND_MENU_SELECTED, exitHandler, wxID_EXIT );
```

```
}
```

To summarize, using `Bind<>()` requires slightly more typing but is much more flexible than using static event tables so don't hesitate to use it when you need this extra power. On the other hand, event tables are still perfectly fine in simple situations where this extra flexibility is not needed.

How Events are Processed

The previous sections explain how to define event handlers but don't address the question of how exactly `wxWidgets` finds the handler to call for the given event. This section describes the algorithm used in detail. Notice that you may want to run the Event Sample while reading this section and look at its code and the output when the button which can be used to test the event handlers execution order is clicked to understand it better.

When an event is received from the windowing system, `wxWidgets` calls `wxEvtHandler::ProcessEvent()` on the first event handler object belonging to the window generating the event. The normal order of event table searching by `ProcessEvent()` is as follows, with the event processing stopping as soon as a handler is found (unless the handler calls `wxEvt::Skip()` in which case it doesn't count as having handled the event and the search continues):

Step 0) Before anything else happens, `wxApp::FilterEvent()` is called. If it returns anything but -1 (default), the event handling stops immediately.

Step 1) If this event handler is disabled via a call to `wxEvtHandler::SetEvtHandlerEnabled()` the next three steps are skipped and the event handler resumes at step (5).

Step 2) If the object is a `wxWindow` and has an associated validator, `wxValidator` gets a chance to process the event.

Step 3) The list of dynamically bound event handlers, i.e., those for which `Bind()` was called, is consulted. Notice that this is done before checking the static event table entries, so if both a dynamic and a static event handler match the same event, the static one is never going to be used unless `wxEvtHandler::Skip()` is called in the dynamic one.

Step 4) The event table containing all the handlers defined using the event table macros in this class and its base classes is examined. Notice that this means that any event handler defined in a base class will be executed at this step.

Step 5) The event is passed to the next event handler, if any, in the event handler chain, i.e., the steps (1) to (4) are done for it. Usually there is no next event handler so the control passes to the next step but see Event Handlers Chain for how the next handler may be defined.

Step 6) If the object is a `wxWindow` and the event is set to propagate (by default only `wxCommandEvent`-derived events are set to propagate), then the processing restarts from the step (1) (and excluding the step (7)) for the parent window. If this object is not a window but the next handler exists, the event is passed to its parent if it is a window. This ensures that in a common case of (possibly several) non-window event handlers pushed on top of a window, the event eventually reaches the window parent.

Step 7) Finally, i.e., if the event is still not processed, the `wxApp` object itself (which derives from `wxEvtHandler`) gets a last chance to process it.

Please pay close attention to step 7. People often overlook or get confused by this powerful feature of the `wxWidgets` event processing system. The details of event propagation up the window hierarchy are described in the next section.

Also please notice that there are additional steps in the event handling for the windows-making part of `wxWidgets` document-view framework, i.e., `wxDocParentFrame`, `wxDocChildFrame` and their MDI equivalents `wxDocMDIParentFrame` and `wxDocMDIChildFrame`. The parent frame classes modify step (2) above to send the events received by them to `wxDocManager` object first. This object, in turn, sends the event to the current view and the view itself lets its associated document process the event first. The child frame classes send the event directly to the associated view which still forwards it to its document object. Notice that to avoid remembering the exact order in which the events are processed in the document-view frame, the simplest, and recommended, solution is to only handle the events at the view classes level, and not in the document or document manager classes.

How Events Propagate Upwards

As mentioned above, the events of the classes deriving from `wxCommandEvent` are propagated by default to the parent window if they are not processed in this window itself. But although by default only the command events are propagated like this, other events can be propagated as well because the event handling code uses `wxEvtHandler::ShouldPropagate()` to check whether an event should be propagated. It is also possible to propagate the event only a limited number of times and not until it is processed (or a top level parent window is reached).

Finally, there is another additional complication (which, in fact, simplifies life of wxWidgets programmers significantly): when propagating the command events up to the parent window, the event propagation stops when it reaches the parent dialog, if any. This means that you don't risk getting unexpected events from the dialog controls (which might be left unprocessed by the dialog itself because it doesn't care about them) when a modal dialog is popped up. The events do propagate beyond the frames, however. The rationale for this choice is that there are only a few frames in a typical application and their parent-child relation are well understood by the programmer while it may be difficult, if not impossible, to track down all the dialogs that may be popped up in a complex program (remember that some are created automatically by wxWidgets). If you need to specify a different behaviour for some reason, you can use `wxWindow::SetExtraStyle(wxWS_EX_BLOCK_EVENTS)` explicitly to prevent the events from being propagated beyond the given window or unset this flag for the dialogs that have it on by default.

Typically events that deal with a window as a window (size, motion, paint, mouse, keyboard, etc.) are sent only to the window. Events that have a higher level of meaning or are generated by the window itself (button click, menu select, tree expand, etc.) are command events and are sent up to the parent to see if it is interested in the event. More precisely, as said above, all event classes not deriving from `wxCommandEvent` (see the `wxEvent` inheritance map) do not propagate upward.

In some cases, it might be desired by the programmer to get a certain number of system events in a parent window, for example all key events sent to, but not used by, the native controls in a dialog. In this case, a special event handler will have to be written that will override `ProcessEvent()` in order to pass all events (or any selection of them) to the parent window.

Event Handlers Chain

The step 4 of the event propagation algorithm checks for the next handler in the event handler chain. This chain can be formed using `wxEvtHandler::SetNextHandler()`:



(referring to the image, if `A->ProcessEvent` is called and it doesn't handle the event, `B->ProcessEvent` will be called and so on...).

Additionally, in the case of `wxWindow` you can build a stack (implemented using `wxEvtHandler` double-linked list) using `wxWindow::PushEventHandler()`:



(referring to the image, if `W->ProcessEvent` is called, it immediately calls `A->ProcessEvent`; if nor `A` nor `B` handle the event, then the `wxWindow` itself is used -- i.e. the dynamically bind event handlers and static event table entries of `wxWindow` are looked as the last possibility, after all pushed event handlers were tested).

By default the chain is empty, i.e. there is no next handler.

Custom Event Summary

General approach

As each event is uniquely defined by its event type, defining a custom event starts with defining a new event type for it. This is done using `wxDEFINE_EVENT()` macro. As an event type is a variable, it can also be declared using `wxDECLARE_EVENT()` if necessary.

The next thing to do is to decide whether you need to define a custom event class for events of this type or if you can reuse an existing class, typically either `wxEvent` (which doesn't provide any extra information) or `wxCommandEvent` (which contains several extra fields and also propagates upwards by default). Both strategies are described in details below. See also the Event Sample for a complete example of code defining and working with the custom event types.

Finally, you will need to generate and post your custom events. Generation is as simple as instantiating your custom event class and initializing its internal fields. For posting events to a certain event handler there are two possibilities: using `wxEvtHandler::AddPendingEvent` or using `wxEvtHandler::QueueEvent`. Basically you will need to use the latter when doing inter-thread communication; when you use only the main thread you can also safely use the former. Last, note that there are also two simple global wrapper functions associated to the two `wxEvtHandler` mentioned functions: `wxPostEvent()` and `wxQueueEvent()`.

Using Existing Event Classes

If you just want to use a `wxCommandEvent` with a new event type, use one of the generic event table macros listed below, without having to define a new event class yourself.

Example:

```
// this is typically in a header: it just declares MY_EVENT event type
```

```
wxDECLARE_EVENT(MY_EVENT, wxCommandEvent);
```

```
// this is a definition so can't be in a header
```

```
wxDEFINE_EVENT(MY_EVENT, wxCommandEvent);
```

```
// example of code handling the event with event tables
```

```
BEGIN_EVENT_TABLE(MyFrame, wxFrame)
```

```
EVT_MENU (wxID_EXIT, MyFrame::OnExit)
```

```
...
```

```
EVT_COMMAND (ID_MY_WINDOW, MY_EVENT, MyFrame::OnMyEvent)
```

```
END_EVENT_TABLE()
```

```
void MyFrame::OnMyEvent(wxCommandEvent& event)
```

```
{
```

```
    // do something
```

```
    wxString text = event.GetText();
```

```
}
```

```
// example of code handling the event with Bind<>():
```

```
MyFrame::MyFrame()
```

```
{
```

```
    Bind(MY_EVENT, &MyFrame::OnMyEvent, this, ID_MY_WINDOW);
```

```
}
```

```
// example of code generating the event
```

```
void MyWindow::SendEvent()
```

```
{
```

```
    wxCommandEvent event(MY_EVENT, GetId());
```

```
    event.SetEventObject(this);
```

```
    // Give it some contents
```

```
    event.SetText("Hello");
```

```
    // Do send it
```

```
    ProcessWindowEvent(event);
```

```
}
```

Defining Your Own Event Class

Under certain circumstances, you must define your own event class e.g., for sending more complex data from one place to another. Apart from defining your event class, you also need to define your own event table macro if you want to use event tables for handling events of this type.

Here is an example:

```
// define a new event class

class MyPlotEvent: public wxEvent
{
public:
    MyPlotEvent(wxEventType eventType, int winid, const wxPoint& pos)
        : wxEvent(winid, eventType),
          m_pos(pos)
    {
    }

    // accessors
    wxPoint GetPoint() const { return m_pos; }

    // implement the base class pure virtual
    virtual wxEvent *Clone() const { return new MyPlotEvent(*this); }

private:
    const wxPoint m_pos;
};

// we define a single MY_PLOT_CLICKED event type associated with the class
```

```
// above but typically you are going to have more than one event type, e.g. you
// could also have MY_PLOT_ZOOMED or MY_PLOT_PANNED &c -- in which case you
// would just add more similar lines here
wxDEFINE_EVENT(MY_PLOT_CLICKED, MyPlotEvent);

// if you want to support old compilers you need to use some ugly macros:
typedef void (wxEvtHandler::*MyPlotEventFunction)(MyPlotEvent&);
#define MyPlotEventHandler(func) wxEVENT_HANDLER_CAST(MyPlotEventFunction, func)

// if your code is only built using reasonably modern compilers, you could just
// do this instead:
#define MyPlotEventHandler(func) (&func)

// finally define a macro for creating the event table entries for the new
// event type
//
// remember that you don't need this at all if you only use Bind<>() and that
// you can replace MyPlotEventHandler(func) with just &func unless you use a
// really old compiler
#define MY_EVT_PLOT_CLICK(id, func) \
    wx__DECLARE_EVT1(MY_PLOT_CLICKED, id, MyPlotEventHandler(func))

// example of code handling the event (you will use one of these methods, not
// both, of course):
```

```
BEGIN_EVENT_TABLE(MyFrame, wxFrame)

EVT_PLOT(ID_MY_WINDOW, MyFrame::OnPlot)

END_EVENT_TABLE()
```

```
MyFrame::MyFrame()
{
    Bind(MY_PLOT_CLICKED, &MyFrame::OnPlot, this, ID_MY_WINDOW);
}
```

```
void MyFrame::OnPlot(MyPlotEvent& event)
{
    ... do something with event.GetPoint() ...
}
```

// example of code generating the event:

```
void MyWindow::SendEvent()
{
    MyPlotEvent event(MY_PLOT_CLICKED, GetId(), wxPoint(...));
    event.SetEventObject(this);
    ProcessWindowEvent(event);
}
```

Miscellaneous Notes

Event Handlers vs Virtual Methods

It may be noted that wxWidgets' event processing system implements something close to virtual methods in normal C++ in spirit: both of these mechanisms allow you to alter the behaviour of the base class by defining the event handling functions in the derived classes.

There is however an important difference between the two mechanisms when you want to invoke the default behaviour, as implemented by the base class, from a derived class handler. With the virtual functions, you need to call the base class function directly and you can do it either in the beginning of the derived class handler function (to post-process the event) or at its end (to pre-process the event). With the event handlers, you only have the option of pre-processing the events and in order to still let the default behaviour happen you must call `wxEvent::Skip()` and not call the base class event handler directly. In fact, the event handler probably doesn't even exist in the base class as the default behaviour is often implemented in platform-specific code by the underlying toolkit or OS itself. But even if it does exist at wxWidgets level, it should never be called directly as the event handlers are not part of wxWidgets API and should never be called directly.

User Generated Events vs Programmatically Generated Events

While generically wxEvents can be generated both by user actions (e.g., resize of a wxWindow) and by calls to functions (e.g., `wxWindow::SetSize`), wxWidgets controls normally send `wxCommandEvent`-derived events only for the user-generated events. The only exceptions to this rule are:

`wxNotebook::AddPage`: No event-free alternatives

`wxNotebook::AdvanceSelection`: No event-free alternatives

`wxNotebook::DeletePage`: No event-free alternatives

`wxNotebook::SetSelection`: Use `wxNotebook::ChangeSelection` instead, as `wxNotebook::SetSelection` is deprecated

`wxTreeCtrl::Delete`: No event-free alternatives

`wxTreeCtrl::DeleteAllItems`: No event-free alternatives

`wxTreeCtrl::EditLabel`: No event-free alternatives

All `wxTextCtrl` methods

`wxTextCtrl::ChangeValue` can be used instead of `wxTextCtrl::SetValue` but the other functions, such as `wxTextCtrl::Replace` or `wxTextCtrl::WriteText` don't have event-free equivalents.

Pluggable Event Handlers

TODO: Probably deprecated, `Bind()` provides a better way to do this

In fact, you don't have to derive a new class from a window class if you don't want to. You can derive a new class from `wxEvtHandler` instead, defining the appropriate event table, and then call `wxWindow::SetEventHandler` (or, preferably, `wxWindow::PushEventHandler`) to make this event handler the object that responds to events. This way, you can avoid a lot of class derivation, and use instances of the same event handler class (but different objects as the same event handler object shouldn't be used more than once) to handle events from instances of different widget classes.

If you ever have to call a window's event handler manually, use the `GetEventHandler` function to retrieve the window's event handler and use that to call the member function. By default, `GetEventHandler` returns a pointer to the window itself unless an application has redirected event handling using `SetEventHandler` or `PushEventHandler`.

One use of `PushEventHandler` is to temporarily or permanently change the behaviour of the GUI. For example, you might want to invoke a dialog editor in your application that changes aspects of dialog boxes. You can grab all the input for an existing dialog box, and edit it 'in situ', before restoring its behaviour to normal. So even if the application has derived new classes to customize behaviour, your utility can indulge in a spot of body-snatching. It could be a useful technique for on-line tutorials, too, where you take a user through a series of steps and don't want them to diverge from the lesson. Here, you can examine the events coming from buttons and windows, and if acceptable, pass them through to the original event handler. Use `PushEventHandler/PopEventHandler` to form a chain of event handlers, where each handler processes a different range of events independently from the other handlers.

Window Identifiers

Window identifiers are integers, and are used to uniquely determine window identity in the event system (though you can use it for other purposes). In fact, identifiers do not need to be unique across your entire application as long as they are unique within the particular context you're interested in, such as a frame and its children. You may use the `wxID_OK` identifier, for example, on any number of dialogs as long as you don't have several within the same dialog.

If you pass `wxID_ANY` to a window constructor, an identifier will be generated for you automatically by `wxWidgets`. This is useful when you don't care about the exact identifier either because you're not going to process the events from the control being created or because you process the events from all controls in one place (in which case you should specify `wxID_ANY` in the event table or `wxEvtHandler::Bind` call as well). The automatically generated identifiers are always negative and so will never conflict with the user-specified identifiers which must be always positive.

See Standard event identifiers for the list of standard identifiers available. You can use `wxID_HIGHEST` to determine the number above which it is safe to define your own identifiers. Or, you can use identifiers below `wxID_LOWEST`. Finally, you can allocate identifiers dynamically using `wxNewId()` function too. If you use `wxNewId()` consistently in your application, you can be sure that your identifiers don't conflict accidentally.

Generic Event Table Macros

<code>EVT_CUSTOM(event, id, func)</code>	Allows you to add a custom event table entry by specifying the event identifier (such as <code>wxEVT_SIZE</code>), the window identifier, and a member function to call.
--	---

EVT_CUSTOM_RANGE(event, id1, id2, func)	The same as EVT_CUSTOM, but responds to a range of window identifiers.
EVT_COMMAND(id, event, func)	The same as EVT_CUSTOM, but expects a member function with a wxCommandEvent argument.
EVT_COMMAND_RANGE(id1, id2, event, func)	The same as EVT_CUSTOM_RANGE, but expects a member function with a wxCommandEvent argument.
EVT_NOTIFY(event, id, func)	The same as EVT_CUSTOM, but expects a member function with a wxNotifyEvent argument.
EVT_NOTIFY_RANGE(event, id1, id2, func)	The same as EVT_CUSTOM_RANGE, but expects a member function with a wxNotifyEvent argument.

List of wxWidgets events

For the full list of event classes, please see the event classes group page.

9.2.10.2.2.4 Low-Level Widget API

The Low-Level Widget API identifies the basic Curses-/nCurses-style (Terminal Control Library) Graphical User Interface features available to the Software Engineer that supports input and output interactions with the local or remote terminal. It is used exclusively to construct, enhance and maintain the wxPython-style, High-Level Widget API emulation. It includes the following GUI objects:

- 1 keyboard** - a Curses controlled object with alphabetic, numeric, punctuation and control buttons used by an operator to enter input.
- 2 mouse** - a Curses controlled object with a position sensing roller ball and one or more buttons used by an operator to enter input that selects a GUI object on the display screen.
- 3 display screen** - a Curses controlled object with multiple columns and rows that displays information output by the application for use by an operator
- 4 window** - a Curses controlled object displayed in a designated portion of the display screen for application specific information
- 5 pad** - a Curses controlled object like a window, except that it is not restricted by the screen size, and is not necessarily associated with a particular part of the screen
- 6 panel** - a Curses controlled object like a window with the added feature of depth, so it can be stacked on top of another, and only the visible portions of each window will be displayed. Panels can be added, moved up or down in the stack, and removed

9.2.10.2.2.5 System Preferences API (Draft)

The computer software product "tsWxGTUI_PyVx" Toolkit includes a configuration file, "tsWxGlobals.py" that may be customized to determine the look and feel of "wxPython" emulation.

- 1 Symbolic Constants**

2 Foreground and Background Colors

3 Default Styles and Themes

4 Unique ID Generators

5 Dimensional Unit Conversion Functions

6 Text Font and Color Attribute Markup Constants

9.2.10.2.3 "tsToolsGUI" Capability

There are currently no GUI applications.

9.2.10.2.4 "tsTestsGUI" Capability

This section is under construction.

Draft

9.3 Platform Interface Capability

9.3.1 Keyboard Interface Capability

9.3.2 Mouse Interface Capability

9.3.3 Display Interface Capability

9.3.4 Event Logging Interface Capability

9.3.5 Process & Thread Launching & Terminating Capability

9.3.6 Process Scheduling Capability

9.3.7 Inter-Process Communication Capability

9.4 Application Programming Interface Capability

9.4.1 Python Built-In Module API Capability (Draft)

Reference:

Python 2.x (2.7.9) Module Index

<https://docs.python.org/2.7/py-modindex.html>

Python 3.x (3.4.2) Module Index

<https://docs.python.org/3.2/py-modindex.html>

Modules (only main-level shown):

1. argparse
2. copy
3. ctypes

4. curses
5. errno
6. fcntl
7. fnmatch
8. getopt
9. imp
10. logger
11. math
12. optprse
13. os
14. platform
15. shlex
16. shutil
17. signal
18. stat
19. struct
20. subprocess
21. syslog
22. system
23. termios
24. textwrap
25. thread
26. threading
27. time
28. traceback

9.4.2 High-Level GUI "wxPython"-style API Capability (Draft)

The pixel-mode wxPthon API represents a blending of the wxWidgets C++ class library with the Python programming language.

Version	Description
2.8.12 (Second generation)	<p>The "tsWxGTUI_PyVx" Toolkit creates a character-mode emulation of a subset of this wxPython pixel-mode version.</p> <p>http://docs.wxwidgets.org/2.8/</p> <ul style="list-style-type: none">▪ wxWidgets 2.8.12: A portable C++ and Python GUI toolkit▪ Julian Smart, Robert Roebling, Vadim Zeitlin, Robin Dunn, et al▪ March, 2011
3.0.0 (Third generation)	<p>http://docs.wxwidgets.org/3.0/</p> <ul style="list-style-type: none">▪ wxWidgets 3.0.0: A portable C++ and Python GUI toolkit▪ Julian Smart, Robert Roebling, Vadim Zeitlin, Robin Dunn, et al▪ November 11, 2013

Excerpt from /tsLibGUI/tsWxPkg/src/tsWx.py:

Identifies currently available components of the emulated wxPython API.

```

try:

    if ReadyForIntegration:

        # Release candidates suitable for integration with applications.
        # Troubleshooting will be required before uncommenting.

        # Prototypes suitable for integration with applications.

        from tsWxGlobals import *

        Debug_CLI_Configuration = Troubleshooting['Debug_CLI_Configuration'],
        Debug_CLI_Exceptions = Troubleshooting['Debug_CLI_Exceptions']
        Debug_CLI_Launch = Troubleshooting['Debug_CLI_Launch']
        Debug_CLI_Progress = Troubleshooting['Debug_CLI_Progress']
        Debug_CLI_Termination = Troubleshooting['Debug_CLI_Termination']
        Debug_GUI_Configuration = Troubleshooting['Debug_GUI_Configuration']
        Debug_GUI_Exceptions = Troubleshooting['Debug_GUI_Exceptions']
        Debug_GUI_Launch = Troubleshooting['Debug_GUI_Launch']
        Debug_GUI_Progress = Troubleshooting['Debug_GUI_Progress']
        Debug_GUI_Termination = Troubleshooting['Debug_GUI_Termination']

        from tsWxColor import Color
        from tsWxColorDatabase import ColorDatabase
        from tsWxApp import App
        from tsWxButton import Button
        from tsWxFrameButton import FrameButton
        from tsWxDialogButton import DialogButton
        from tsWxCheckBox import CheckBox
        from tsWxControl import Control
        from tsWxDialog import Dialog
        from tsWxFrame import Frame
        from tsWxNonLinkedList import NonLinkedList
        from tsWxScreen import Screen
        from tsWxSplashScreen import SplashScreen
        from tsWxGauge import Gauge
        from tsWxMenu import Menu
        from tsWxMenuBar import MenuBar
        from tsWxKeyboardState import KeyboardState
        from tsWxMouseState import MouseState
        from tsWxTextEditBox import TextEditBox
        from tsWxTextEntryDialog import TextEntryDialog
        from tsWxPasswordEntryDialog import PasswordEntryDialog
##         from tsWxDebugHandlers import DebugHandlers
        from tsWxObject import Object
        from tsWxPanel import Panel
        from tsWxPoint import Point
        from tsWxPyApp import PyApp
        from tsWxPyEventBinder import PyEventBinder
        from tsWxPyOnDemandOutputWindow import PyOnDemandOutputWindow
        from tsWxPySimpleApp import PySimpleApp
        # from tsWxCommandLineEnv import CommandLineEnv
        from tsWxPySizer import PySizer
        from tsWxRadioBox import RadioBox

```

```
from tsWxRadioButton import RadioButton
from tsWxRect import Rect
from tsWxScrollBarButton import ScrollBarButton
from tsWxScrollBarGauge import ScrollBarGauge
from tsWxScrollBar import ScrollBar
from tsWxScrolledText import ScrolledText
from tsWxScrolled import Scrolled
from tsWxScrolledWindow import ScrolledWindow
from tsWxSize import Size
from tsWxSizer import Sizer
from tsWxSizerSpacer import SizerSpacer
from tsWxSizerItemList import SizerItemList
from tsWxSizerItem import SizerItem
from tsWxStaticLine import StaticLine
from tsWxStaticText import StaticText
from tsWxGridSizer import GridSizer
from tsWxStatusBar import StatusBar
from tsWxSystemSettings import SystemSettings
from tsWxTextCtrl import TextCtrl
from tsWxTimer import Timer
from tsWxTopLevelWindow import TopLevelWindow
from tsWxWindow import Window
from tsWxEvent import *
from tsWxStaticBox import StaticBox
import tsWxKeyEvent as KeyEvent
import tsWxMouseEvent as MouseEvent
import tsWxEvtHandler as tsEvtHandler
import tsWxShowEvent as tsShowEvent
from tsWxAcceleratorEntry import AcceleratorEntry
from tsWxAcceleratorTable import AcceleratorTable
from tsWxCaret import Caret
from tsWxDisplay import Display
from tsWxEventLoop import EventLoop
from tsWxEventLoopActivator import EventLoopActivator
from tsWxFocusEvent import FocusEvent
from tsWxValidator import Validator
from tsWxEventDaemon import EventDaemon
from tsWxEventQueueEntry import EventQueueEntry
from tsWxEventTableEntry import EventTableEntry
#from tsWxEvent import Event
#from tsWxEvent import PyEventBinder
#from tsWxEvtHandler import EvtHandler
##      from tsWxMouseEvent import MouseEvent
##      from tsWxKeyEvent import KeyEvent
from tsWxBoxSizer import BoxSizer
from tsWxCursor import Cursor
from tsWxStaticBoxSizer import StaticBoxSizer

except ImportError, importCode:

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

except Exception, exceptionCode:

    print('tsWx: Exception (tsLibGUI); ' + \
```



```
'exceptionCode=%s' % str(exceptionCode))
```

9.4.3 Low-Level GUI "nCurses"-style API Capability (Draft)

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

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

Reference:

Python 2.x (2.7.6) Curses Module

<https://docs.python.org/2.7/library/curses.html#module-curses>

Python 3.x (3.2.5) Curses Module

<https://docs.python.org/3.2/library/curses.html#module-curses>

Excerpt from Python 2.7.6 Global Module Index for Curses Module:

"The curses library supplies a terminal-independent screen-painting and keyboard-handling facility for text-based terminals; such terminals include VT100s, the Linux console, and the simulated terminal provided by X11 programs such as xterm and rxvt. Display terminals support various control codes to perform common operations such as moving the cursor, scrolling the screen, and erasing areas. Different terminals use widely differing codes, and often have their own minor quirks.

In a world of X displays, one might ask "why bother"? It's true that character-cell display terminals are an obsolete technology, but there are niches in which being able to do fancy things with them are still valuable. One is on small-footprint or embedded Unixes that don't carry an X server. Another is for tools like OS installers and kernel configurators that may have to run before X is available.

The curses library hides all the details of different terminals, and provides the programmer with an abstraction of a display, containing multiple non-overlapping windows. The contents of a window can be changed in various ways— adding text, erasing it, changing its appearance—and the curses library will automagically figure out what control codes need to be sent to the terminal to produce the right output.

The curses library was originally written for BSD Unix; the later System V versions of Unix from AT&T added many enhancements and new functions. BSD curses is no longer maintained, having been replaced by ncurses, which is an open-source implementation of the AT&T interface. If you're using an open-source Unix such as Linux or FreeBSD, your system almost certainly uses ncurses. Since most current commercial Unix versions are based on System V code, all the functions described here will probably be available. The older versions of curses carried by some proprietary Unixes may not support everything, though.

No one has made a Windows port of the curses module. On a Windows platform, try the Console module written by Fredrik Lundh. The Console module provides cursor-addressable text output, plus full support for mouse and keyboard input, and is available from <http://effbot.org/zone/console-index.htm>."

Users of various Microsoft Windows distributions (8.1, 8, 7, Vista and XP) may install "Cygwin", a free, Unix-like environment and command-line interface add-on from Red Hat. "Cygwin" provides native integration of Windows-based applications, data, and other system resources with applications, software tools, and data of the Unix-like environment. The running "Cygwin" Command Line Interface includes a fully functional version of ncurses that enables Python applications, and the "tsWxGTUI_PyVx" Toolkit, to use the Python curses module.

9.4.3.1 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.



Draft

9.4.3.2 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.

9.4.3.2.1 Python Curses Module Evolution

9.4.3.2.1.1 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

9.4.3.2.1.2 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

9.4.3.2.1.3 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.

9.5 Operator Interface Capability

9.5.1 Command Line Interface (CLI) Capability

See *Command Line Interface Capability* (see "*tsToolkitCLI*" *Command Line Interface Capability*" on page 155)

9.5.2 Graphical-style User Interface (GUI) Capability

See *Graphical-style User Interface Capability* (see "*tsToolkitGUI*" *Graphical-style User Interface Capability*" on page 159)

Draft

Draft

10 APPENDIX C - SYSTEM EXTERNAL INTERFACE REQUIREMENTS

This paragraph shall be divided into subparagraphs to specify the requirements, if any, for the system's external interfaces. This paragraph may reference one or more Interface Requirements Specifications (IRs) or other documents containing these requirements.

The system hardware and software shall provide the means for interfacing the "tsWxGTUI_PyVx" Toolkit, and those application programs created with it, with different types of computer hardware and/or with different software packages:

- 1 *Hardware Components* (on page 205)
- 2 *Software Components* (on page 224)
- 3 *Architecture* (on page 227)

10.1(Project-unique Identifier Of Interface Template)

This paragraph (beginning with 3.3.2) shall identify a system external interface by project-unique identifier, shall briefly identify the interfacing entities, and shall be divided into subparagraphs as needed to state the requirements imposed on the system to achieve the interface. Interface characteristics of the other entities involved in the interface shall be stated as assumptions or as "When [the entity not covered] does this, the system shall...", not as requirements on the other entities. This paragraph may reference other documents (such as data dictionaries, standards for communication protocols, and standards for user interfaces) in place of stating the information here. The requirements shall include the following, as applicable, presented in any order suited to the requirements, and shall note any differences in these characteristics from the point of view of the interfacing entities (such as different expectations about the size, frequency, or other characteristics of data elements):

- a. Priority that the system must assign the interface
- b. Requirements on the type of interface (such as real-time data transfer, storage-and-retrieval of data, etc.) to be implemented
- c. Required characteristics of individual data elements that the system must provide, store, send, access, receive, etc., such as:
 - 1) Names/identifiers
 - a) Project-unique identifier
 - b) Non-technical (natural-language) name
 - c) DoD standard data element name
 - d) Technical name (e.g., variable or field name in code or database)
 - e) Abbreviation or synonymous names
 - 2) Data type (alphanumeric, integer, etc.)
 - 3) Size and format (such as length and punctuation of a character string)
 - 4) Units of measurement (such as meters, dollars, nanoseconds)
 - 5) Range or enumeration of possible values (such as 0-99)

- 6) Accuracy (how correct) and precision (number of significant digits)
- 7) Priority, timing, frequency, volume, sequencing, and other constraints, such as whether the data element may be updated and whether business rules apply
- 8) Security and privacy constraints
- 9) Sources (setting/sending entities) and recipients (using/receiving entities)

d. Required characteristics of data element assemblies (records, messages, files, arrays, displays, reports, etc.) that the system must provide, store, send, access, receive, etc., such as:

- 1) Names/identifiers
 - a) Project-unique identifier
 - b) Non-technical (natural language) name
 - c) Technical name (e.g., record or data structure name in code or database)
 - d) Abbreviations or synonymous names
- 2) Data elements in the assembly and their structure (number, order, grouping)
- 3) Medium (such as disk) and structure of data elements/assemblies on the medium
- 4) Visual and auditory characteristics of displays and other outputs (such as colors, layouts, fonts, icons and other display elements, beeps, lights)
- 5) Relationships among assemblies, such as sorting/access characteristics
- 6) Priority, timing, frequency, volume, sequencing, and other constraints, such as whether the assembly may be updated and whether business rules apply
- 7) Security and privacy constraints
- 8) Sources (setting/sending entities) and recipients (using/receiving entities)

e. Required characteristics of communication methods that the system must use for the interface, such as:

- 1) Project-unique identifier(s)
- 2) Communication links/bands/frequencies/media and their characteristics

- transfers
- 3) Message formatting
 - 4) Flow control (such as sequence numbering and buffer allocation)
 - 5) Data transfer rate, whether periodic/apperiodic, and interval between
 - 6) Routing, addressing, and naming conventions
 - 7) Transmission services, including priority and grade
 - 8) Safety/security/privacy considerations, such as encryption, user authentication, compartmentalization, and auditing

f. Required characteristics of protocols the system must use for the interface, such as:

- addressing
- 1) Project-unique identifier(s)
 - 2) Priority/layer of the protocol
 - 3) Packeting, including fragmentation and reassembly, routing, and
 - 4) Legality checks, error control, and recovery procedures
 - 5) Synchronization, including connection establishment, maintenance,
 - 6) Status, identification, and any other reporting features
- termination

g. Other required characteristics, such as physical compatibility of the interfacing entities (dimensions, tolerances, loads, plug compatibility, etc.), voltages, etc.

10.2 Interface Identification and Diagrams

This paragraph shall identify the required external interfaces of the system. The identification of each interface shall include a project-unique identifier and shall designate the interfacing entities (systems, configuration items, users, etc.) by name, number, version, and documentation references, as applicable. The identification shall state which entities have fixed interface characteristics (and therefore impose interface requirements on interfacing entities) and which are being developed or modified (thus having interface requirements imposed on them). One or more interface diagrams shall be provided to depict the interfaces.

10.2.1 Hardware Components

The hardware components include one or more of the following:

- **Computer Processor, Memory, Storage and Network Devices** (on page 205) - Identifies hardware components required for local and remote computer processing of commands and data from System Operator or Software Engineer.
- **Keyboard, Mouse, Trackball and Touchpad Devices** (on page 206) - Describes hardware components required for appropriate input of commands and data by System Operator or Software Engineer.
- **Display and Printer Devices** (see "**Display and Printer Output Devices**" on page 207) - Describes hardware components required for reporting of appropriate output of command responses and data to System Operator or Software Engineer.

10.2.1.1 Computer Processor, Memory, Storage and Network Devices

The hardware components include one or more of the following:

- **Central Processing Unit** - A required electronic device that loads and executes machine-readable instructions. It performs arithmetic and logic. It loads, modifies and stores machine-readable data. The processor may be any of the popular 32-/64-bit single or multi-core devices.
- **Random Access Memory** - A required electronic device that temporarily receives, stores and returns machine-readable machine instructions and data.
- **Non-Volatile Storage** - A required internal and optional external electronic device (Flash Memory) or electro-mechanical device (Hard Drive) whose non-volatile memory receives, stores and can then return source and machine-readable data after electrical power interruptions. The optional external device may be local or remote. When remote, the system must include a Network Interface Device.
- **Network Interface Device** - An optional electronic device (Modem) that inputs and outputs data over an optional wired or wireless telecommunications circuit.

10.2.1.2 Keyboard, Mouse, Trackball and Touchpad Devices

NOTE: Pointing Device input is only available with "cygwin X11 shell", "Cygwub Mintty shell", "xterm", "xterm-color" and "xterm-256color" consoles or terminal emulators. It is not available with "vt100", "vt220" and "ansi" consoles or terminal emulators.

The hardware components include one or more of the following:

- **Keyboard Device** - A required hand-operated electronic or electro-mechanical device to input alpha-numeric and control data via push buttons.
- **Pointing Device** - An optional hand-operated electronic or electro-mechanical Mouse, Trackball, Touchpad or Touchscreen device that controls the coordinates of a cursor on the computer screen as you move the positioning control (hand, finger or stylus) around.

Terminal Type	User Input Source
Cygwin Bash Shell	Keyboard
Cygwin Mintty	Keyboard & Pointing Device
Cygwin X11 Desktop	Keyboard & Pointing Device
Git Bash Shell	Keyboard
Linux Terminal	Keyboard & Pointing Device
Mac OS X iTerm / iTerm2	Keyboard & Pointing Device
Mac OS X Terminal	Keyboard
Microsoft Windows PowerShell and Command Prompt shell	Keyboard
vt100 (non-color)	Keyboard
vt220 (non-color)	Keyboard
xterm (8-color)	Keyboard & Pointing Device
xterm-color (obsolete)	Keyboard & Pointing Device
xterm-16color	Keyboard & Pointing Device
xterm-88color	Keyboard & Pointing Device
xterm-256color	Keyboard & Pointing Device

10.2.1.3 Display and Printer Output Devices

NOTES:

- 1) The *TeamSTARS* "tsWxGTUI_PyVx" Toolkit depends on the user to activate and configure the display as appropriate for the user's application.
 - 2) For "vt100" and "vt220" consoles and terminal emulators, the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit supports 'white' on 'black' output for the cygwin console and 'black' on 'white' output for the xterm console.
 - 3) For "cygwin", "linux", "xterm" and "xterm-color" consoles and terminal emulators, the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit maps the standard 68-color "wxPython" palette to the standard 8-color "curses" palette ('black', 'blue', 'cyan', 'green', 'magenta', 'red', 'white', 'yellow').
 - 4) For "linux", "xterm-16color" and "xterm-88color" and "xterm-256color" consoles and terminal emulators, the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit maps the optional 71-color "wxPython" palette to the standard 16-color "curses" palette ('black', 'blue', 'cyan', 'gray', 'green', 'lime green', 'magenta', 'maroon', 'navy', 'olive', 'purple', 'red', 'silver', 'teal', 'white', 'yellow').
 - 5) For "xterm-88color" consoles and terminal emulators, the "tsWxGTUI_PyVx" Toolkit constructs the optional 71-color "wxPython" palette.
 - 6) For "xterm-256color" consoles and terminal emulators, the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit constructs the optional 140-color "wxPython" palette.
-

The hardware components include one or more of the following:

- **Display Device** - A required electronic device that outputs alpha-numeric, graphic and control data to the computer screen.
- **Printer Device** - An optional electro-mechanical device that outputs alpha-numeric, graphic and control data to individual sheets of paper.

See the following for configuration options:

- **Character & Pixel Screen Size** (on page 208)
- **Standard 1-color Monochrome palette** (on page 209)
- **Standard 8-color "Curses" palette** (on page 209)
- **Standard 16-color "Curses" palette** (on page 212)
- **Standard 68-color "wxPython" palette** (on page 214)
- **Optional 71-color "wxPython" palette** (on page 217)
- **Optional 140-color "wxPython" palette** (on page 219)

10.2.1.3.1 Character & Pixel Screen Size

The "tsWxGTUI-Toolkit" supports monochrome vt100 and color cygwin, xterm, xterm-color and xterm-256color consoles with the following screen sizes:

Terminal	Pixel Display Terminal Graphical (Font: 8x12)	Character Display Terminal Non- Graphical
minimum	272 col x 108 row	34 col x 9 row
CGA	320 col x 200 row	40 col x 16 row
VT100	640 col x 288 row	40 col x 24 row
VGA	640 col x 480 row	80 col x 40 row
SVGA	800 col x 600 row	100 col x 50 row
XGA	1024 col x 768 row	128 col x 64 row
MAC	1152 col x 870 row	144 col x 72 row
SXGA	1280 col x 1024 row	160 col x 85 row
WXGA	1366 col x 768 row	170 col x 64 row
SXGA+	1400 col x 1050 row	175 col x 87 row
UXGA	1600 col x 1200 row	200 col x 100 row
WSXGA	1680 col x 1050 row	210 col x 87 row
UXGA	1680 col x 1200 row	210 col x 100 row
WUXGA	1920 col x 1200 row	240 col x 100 row
OXGA	2048 col x 1600 row	256 col x 133 row

NOTES:

1) The application developer may have to re-compile and rebuild a platform's "ncurses" or "curses" library with the wide character option in order to support anything other than the standard 8-color or optional 16-color "curses" palette.

2) The operator typically has control of the actual font. Smaller fonts enable the display of more information on a standard size terminal. The tabulation only depicts the display size associated with a 8x12 Font.

10.2.1.3.2 Standard 1-color Monochrome palette

From http://en.wikipedia.org/wiki/Monochrome_monitor:

A monochrome monitor is a type of computer display which was very common in the early days of computing, from the 1960s through the 1980s, before color monitors became popular. They are still widely used in applications such as computerized cash register systems. Unlike color monitors, which display text and graphics in multiple colors through the use of alternating-intensity red, green, and blue phosphors, monochrome monitors have only one color of phosphor (mono means "one", and chrome means "color"). All text and graphics are displayed in that color. Some monitors have the ability to vary the brightness of individual pixels, thereby creating the illusion of depth and color, exactly like a black-and-white television.

Monochrome monitors are commonly available in three colors: if the P1 phosphor is used, the screen is green monochrome. If the P3 phosphor is used, the screen is amber monochrome. If the P4 phosphor is used, the screen is white monochrome (known as "page white"); this is the same phosphor as used in early television sets. An amber screen was claimed to give improved ergonomics, specifically by reducing eye strain; this claim appears to have little scientific basis.[1]

For "vt100" and "vt220" consoles and terminal emulators, the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit supports 'white' (for which the host operating system may substitute 'green', 'orange' or another color) on 'black' output for the cygwin console and 'black' on 'white' output for the xterm console displays. The *TeamSTARS* "tsWxGTUI_PyVx" Toolkit maps the standard 68-color "wxPython" palette to the standard 1-color "monochrome" palette:

- 1 'black' (color Off)
- 2 'white' (color On)

10.2.1.3.3 Standard 8-color "Curses" palette

For "cygwin", "linux", "xterm" and "xterm-color" consoles and terminal emulators, the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit maps the standard 68-color "wxPython" palette to the standard 8-color "curses" palette:

- 1 'black'
- 2 'blue'
- 3 'cyan'
- 4 'green'
- 5 'magenta'
- 6 'red'
- 7 'white'
- 8 'yellow'

Color8SubstitutionMap:

- 1 COLOR_AQUAMARINE: COLOR_CYAN,
- 2 COLOR_BLACK: COLOR_BLACK,

- 3 COLOR_BLUE: COLOR_BLUE,
- 4 COLOR_BLUE_VIOLET: COLOR_BLUE,
- 5 COLOR_BROWN: COLOR_YELLOW,
- 6 COLOR_CADET_BLUE: COLOR_BLUE,
- 7 COLOR_CORAL: COLOR_RED,
- 8 COLOR_CORNFLOWER_BLUE: COLOR_BLUE,
- 9 COLOR_CYAN: COLOR_CYAN,
- 10 COLOR_DARK_GRAY: COLOR_BLACK,
- 11 COLOR_DARK_GREEN: COLOR_GREEN,
- 12 COLOR_DARK_OLIVE_GREEN: COLOR_GREEN,
- 13 COLOR_DARK_ORCHID: COLOR_MAGENTA,
- 14 COLOR_DARK_SLATE_BLUE: COLOR_BLUE,
- 15 COLOR_DARK_SLATE_GRAY: COLOR_BLACK,
- 16 COLOR_DARK_TURQUOISE: COLOR_CYAN,
- 17 COLOR_DIM_GRAY: COLOR_BLACK,
- 18 COLOR_FIREBRICK: COLOR_RED,
- 19 COLOR_FOREST_GREEN: COLOR_GREEN,
- 20 COLOR_GOLD: COLOR_YELLOW,
- 21 COLOR_GOLDENROD: COLOR_YELLOW,
- 22 COLOR_GRAY: COLOR_BLACK,
- 23 COLOR_GREEN: COLOR_GREEN,
- 24 COLOR_GREEN_YELLOW: COLOR_GREEN,
- 25 COLOR_INDIAN_RED: COLOR_RED,
- 26 COLOR_KHAKI: COLOR_YELLOW,
- 27 COLOR_LIGHT_BLUE: COLOR_BLUE,
- 28 COLOR_LIGHT_GRAY: COLOR_BLACK,
- 29 COLOR_LIGHT_STEEL_BLUE: COLOR_BLUE,
- 30 COLOR_LIME_GREEN: COLOR_GREEN,
- 31 COLOR_MAGENTA: COLOR_MAGENTA,
- 32 COLOR_MAROON: COLOR_RED,
- 33 COLOR_MEDIUM_AQUAMARINE: COLOR_CYAN,
- 34 COLOR_MEDIUM_BLUE: COLOR_BLUE,
- 35 COLOR_MEDIUM_FOREST_GREEN: COLOR_GREEN,
- 36 COLOR_MEDIUM_GOLDENROD: COLOR_YELLOW,

- 37** COLOR_MEDIUM_ORCHID: COLOR_MAGENTA,
- 38** COLOR_MEDIUM_SEA_GREEN: COLOR_GREEN,
- 39** COLOR_MEDIUM_SLATE_BLUE: COLOR_BLUE,
- 40** COLOR_MEDIUM_SPRING_GREEN: COLOR_GREEN,
- 41** COLOR_MEDIUM_TURQUOISE: COLOR_CYAN,
- 42** COLOR_MEDIUM_VIOLET_RED: COLOR_RED,
- 43** COLOR_MIDNIGHT_BLUE: COLOR_BLUE,
- 44** COLOR_NAVY: COLOR_BLUE,
- 45** COLOR_OLIVE: COLOR_GREEN,
- 46** COLOR_ORANGE: COLOR_RED,
- 47** COLOR_ORANGE_RED: COLOR_RED,
- 48** COLOR_ORCHID: COLOR_MAGENTA,
- 49** COLOR_PALE_GREEN: COLOR_GREEN,
- 50** COLOR_PINK: COLOR_RED,
- 51** COLOR_PLUM: COLOR_MAGENTA,
- 52** COLOR_PURPLE: COLOR_RED,
- 53** COLOR_RED: COLOR_RED,
- 54** COLOR_SALMON: COLOR_RED,
- 55** COLOR_SEA_GREEN: COLOR_GREEN,
- 56** COLOR_SIENNA: COLOR_RED,
- 57** COLOR_SILVER: COLOR_WHITE,
- 58** COLOR_SKY_BLUE: COLOR_CYAN,
- 59** COLOR_SLATE_BLUE: COLOR_BLUE,
- 60** COLOR_SPRING_GREEN: COLOR_GREEN,
- 61** COLOR_STEEL_BLUE: COLOR_BLUE,
- 62** COLOR_TAN: COLOR_YELLOW,
- 63** COLOR_TEAL: COLOR_CYAN,
- 64** COLOR_THISTLE: COLOR_BLACK,
- 65** COLOR_TURQUOISE: COLOR_CYAN,
- 66** COLOR_VIOLET: COLOR_MAGENTA,
- 67** COLOR_VIOLET_RED: COLOR_RED,
- 68** COLOR_WHEAT: COLOR_YELLOW,
- 69** COLOR_WHITE: COLOR_WHITE,

70 COLOR_YELLOW: COLOR_YELLOW,

71 COLOR_YELLOW_GREEN: COLOR_YELLOW

10.2.1.3.4 Standard 16-color "Curses" palette

For "linux", "xterm-16color" and (to "xterm-88color" and "xterm-256color" when the "tsWxGlobals" configuration switch "USE_256_COLOR_PAIR_LIMIT" applies) consoles and terminal emulators, the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit maps the optional 71-color "wxPython" palette to the standard 16-color "curses" palette:

- 1** 'black'
- 2** 'blue'
- 3** 'cyan'
- 4** 'gray'
- 5** 'green'
- 6** 'lime green'
- 7** 'magenta'
- 8** 'maroon'
- 9** 'navy'
- 10** 'olive'
- 11** 'purple'
- 12** 'red'
- 13** 'silver'
- 14** 'teal'
- 15** 'white'
- 16** 'yellow'

Color16SubstitutionMap:

- 1** COLOR_AQUAMARINE: COLOR_CYAN,
- 2** COLOR_BLACK: COLOR_BLACK,
- 3** COLOR_BLUE: COLOR_BLUE,
- 4** COLOR_BLUE_VIOLET: COLOR_BLUE,
- 5** COLOR_BROWN: COLOR_MAROON,
- 6** COLOR_CADET_BLUE: COLOR_BLUE,
- 7** COLOR_CORAL: COLOR_RED,
- 8** COLOR_CORNFLOWER_BLUE: COLOR_BLUE,
- 9** COLOR_CYAN: COLOR_CYAN,
- 10** COLOR_DARK_GRAY: COLOR_BLACK,

- 11** COLOR_DARK_GREEN: COLOR_GREEN,
- 12** COLOR_DARK_OLIVE_GREEN: COLOR_GREEN,
- 13** COLOR_DARK_ORCHID: COLOR_MAGENTA,
- 14** COLOR_DARK_SLATE_BLUE: COLOR_BLUE,
- 15** COLOR_DARK_SLATE_GRAY: COLOR_BLACK,
- 16** COLOR_DARK_TURQUOISE: COLOR_CYAN,
- 17** COLOR_DIM_GRAY: COLOR_GRAY,
- 18** COLOR_FIREBRICK: COLOR_RED,
- 19** COLOR_FOREST_GREEN: COLOR_GREEN,
- 20** COLOR_GOLD: COLOR_YELLOW,
- 21** COLOR_GOLDENROD: COLOR_YELLOW,
- 22** COLOR_GRAY: COLOR_GRAY,
- 23** COLOR_GREEN: COLOR_GREEN,
- 24** COLOR_GREEN_YELLOW: COLOR_GREEN,
- 25** COLOR_INDIAN_RED: COLOR_RED,
- 26** COLOR_KHAKI: COLOR_YELLOW,
- 27** COLOR_LIGHT_BLUE: COLOR_BLUE,
- 28** COLOR_LIGHT_GRAY: COLOR_BLACK,
- 29** COLOR_LIGHT_STEEL_BLUE: COLOR_BLUE,
- 30** COLOR_LIME_GREEN: COLOR_LIME_GREEN,
- 31** COLOR_MAGENTA: COLOR_MAGENTA,
- 32** COLOR_MAROON: COLOR_MAROON,
- 33** COLOR_MEDIUM_AQUAMARINE: COLOR_CYAN,
- 34** COLOR_MEDIUM_BLUE: COLOR_BLUE,
- 35** COLOR_MEDIUM_FOREST_GREEN: COLOR_GREEN,
- 36** COLOR_MEDIUM_GOLDENROD: COLOR_YELLOW,
- 37** COLOR_MEDIUM_ORCHID: COLOR_MAGENTA,
- 38** COLOR_MEDIUM_SEA_GREEN: COLOR_GREEN,
- 39** COLOR_MEDIUM_SLATE_BLUE: COLOR_BLUE,
- 40** COLOR_MEDIUM_SPRING_GREEN: COLOR_GREEN,
- 41** COLOR_MEDIUM_TURQUOISE: COLOR_CYAN,
- 42** COLOR_MEDIUM_VIOLET_RED: COLOR_RED,
- 43** COLOR_MIDNIGHT_BLUE: COLOR_BLUE,

44 COLOR_NAVY: COLOR_NAVY,
45 COLOR_OLIVE: COLOR_OLIVE,
46 COLOR_ORANGE: COLOR_RED,
47 COLOR_ORANGE_RED: COLOR_RED,
48 COLOR_ORCHID: COLOR_MAGENTA,
49 COLOR_PALE_GREEN: COLOR_GREEN,
50 COLOR_PINK: COLOR_RED,
51 COLOR_PLUM: COLOR_MAGENTA,
52 COLOR_PURPLE: COLOR_PURPLE,
53 COLOR_RED: COLOR_RED,
54 COLOR_SALMON: COLOR_RED,
55 COLOR_SEA_GREEN: COLOR_GREEN,
56 COLOR_SIENNA: COLOR_RED,
57 COLOR_SILVER: COLOR_SILVER,
58 COLOR_SKY_BLUE: COLOR_CYAN,
59 COLOR_SLATE_BLUE: COLOR_BLUE,
60 COLOR_SPRING_GREEN: COLOR_GREEN,
61 COLOR_STEEL_BLUE: COLOR_BLUE,
62 COLOR_TAN: COLOR_YELLOW,
63 COLOR_TEAL: COLOR_TEAL,
64 COLOR_THISTLE: COLOR_BLACK,
65 COLOR_TURQUOISE: COLOR_CYAN,
66 COLOR_VIOLET: COLOR_MAGENTA,
67 COLOR_VIOLET_RED: COLOR_RED,
68 COLOR_WHEAT: COLOR_YELLOW,
69 COLOR_WHITE: COLOR_WHITE,
70 COLOR_YELLOW: COLOR_YELLOW,
71 COLOR_YELLOW_GREEN: COLOR_YELLOW

10.2.1.3.5 Standard 68-color "wxPython" palette

For 8-color or 64-color pair limited "cygwin", "linux", "xterm" and "xterm-color" compatible displays, the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit supports the standard 68-color "wxPython" palette:

- 1 'aquamarine'
- 2 'black'
- 3 'blue'

- 4** 'blue violet'
- 5** 'brown'
- 6** 'cadet blue'
- 7** 'coral'
- 8** 'cornflower blue'
- 9** 'cyan'
- 10** 'dark gray'
- 11** 'dark green'
- 12** 'dark olive green'
- 13** 'dark orchid'
- 14** 'dark slate blue'
- 15** 'dark slate gray'
- 16** 'dark turquoise'
- 17** 'dim gray'
- 18** 'firebrick'
- 19** 'forest green'
- 20** 'gold'
- 21** 'goldenrod'
- 22** 'gray'
- 23** 'green'
- 24** 'green yellow'
- 25** 'indian red'
- 26** 'khaki'
- 27** 'light blue'
- 28** 'light gray'
- 29** 'light steel blue'
- 30** 'lime green'
- 31** 'magenta'
- 32** 'maroon'
- 33** 'medium aquamarine'
- 34** 'medium blue'
- 35** 'medium forest green'
- 36** 'medium goldenrod'

- 37 'medium orchid'
- 38 'medium sea green'
- 39 'medium slate blue'
- 40 'medium spring green'
- 41 'medium turquoise'
- 42 'medium violet red'
- 43 'midnight blue'
- 44 'navy'
- 45 'orange'
- 46 'orange red'
- 47 'orchid'
- 48 'pale green'
- 49 'pink'
- 50 'plum'
- 51 'purple'
- 52 'red'
- 53 'salmon'
- 54 'sea green'
- 55 'sienna'
- 56 'sky blue'
- 57 'slate blue'
- 58 'spring green'
- 59 'steel blue'
- 60 'tan'
- 61 'thistle'
- 62 'turquoise'
- 63 'violet'
- 64 'violet red'
- 65 'wheat'
- 66 'white'
- 67 'yellow'
- 68 'yellow green'

NOTES:

- 1) Python uses the "curses" or "ncurses" library with its standard 8-color palette.
 - 2) Discovered the following with Google Search for "NCurses 256 Color Support": "But note that some terminals, while they can support 256 colors, are not able to change the palette. To compile NCurses with 256 color support, ... www.c-for-dummies.com/ncurses/256color/"
-

10.2.1.3.6 Optional 71-color "wxPython" palette

For "xterm-88color" (when the "tsWxGlobals" configuration switch "USE_256_COLOR_PAIR_LIMIT" does NOT apply) consoles and terminal emulators, the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit constructs the optional 71-color "wxPython" palette:

- 1 'aquamarine'
- 2 'black'
- 3 'blue'
- 4 'blue violet'
- 5 'brown'
- 6 'cadet blue'
- 7 'coral'
- 8 'cornflower blue'
- 9 'cyan'
- 10 'dark gray'
- 11 'dark green'
- 12 'dark olive green'
- 13 'dark orchid'
- 14 'dark slate blue'
- 15 'dark slate gray'
- 16 'dark turquoise'
- 17 'dim gray'
- 18 'firebrick'
- 19 'forest green'
- 20 'gold'
- 21 'goldenrod'
- 22 'gray'
- 23 'green'
- 24 'green yellow'

- 25** 'indian red'
- 26** 'khaki'
- 27** 'light blue'
- 28** 'light gray'
- 29** 'light steel blue'
- 30** 'lime green'
- 31** 'magenta'
- 32** 'maroon'
- 33** 'medium aquamarine'
- 34** 'medium blue'
- 35** 'medium forest green'
- 36** 'medium goldenrod'
- 37** 'medium orchid'
- 38** 'medium sea green'
- 39** 'medium slate blue'
- 40** 'medium spring green'
- 41** 'medium turquoise'
- 42** 'medium violet red'
- 43** 'midnight blue'
- 44** 'navy'
- 45** 'olive'
- 46** 'orange'
- 47** 'orange red'
- 48** 'orchid'
- 49** 'pale green'
- 50** 'pink'
- 51** 'plum'
- 52** 'purple'
- 53** 'red'
- 54** 'salmon'
- 55** 'sea green'
- 56** 'sienna'
- 57** 'silver'
- 58** 'sky blue'

- 59** 'slate blue'
- 60** 'spring green'
- 61** 'steel blue'
- 62** 'tan'
- 63** 'teal'
- 64** 'thistle'
- 65** 'turquoise'
- 66** 'violet'
- 67** 'violet red'
- 68** 'wheat'
- 69** 'white'
- 70** 'yellow'
- 71** 'yellow green'

10.2.1.3.7 Optional 140-color "wxPython" palette

For "xterm-256color" (when the "tsWxGlobals" configuration switch "USE_256_COLOR_PAIR_LIMIT" does NOT apply) consoles and terminal emulators, the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit constructs the optional 140-color "wxPython" palette:

- 1** black
- 2** red
- 3** green
- 4** yellow
- 5** blue
- 6** magenta
- 7** cyan
- 8** white
- 9** gray
- 10** maroon
- 11** lime green
- 12** olive
- 13** navy
- 14** purple
- 15** teal
- 16** silver
- 17** aquamarine

- 18 blue violet
- 19 brown
- 20 cadet blue
- 21 coral
- 22 cornflower blue
- 23 dark gray
- 24 dark green
- 25 dark olive green
- 26 dark orchid
- 27 dark slate blue
- 28 dark slate gray
- 29 dark turquoise
- 30 dim gray
- 31 firebrick
- 32 forest green
- 33 gold
- 34 goldenrod
- 35 green yellow
- 36 indian red
- 37 khaki
- 38 light blue
- 39 light gray
- 40 light steel blue
- 41 medium aquamarine
- 42 medium blue
- 43 medium forest green
- 44 medium goldenrod
- 45 medium orchid
- 46 medium sea green
- 47 medium slate blue
- 48 medium spring green
- 49 medium turquoise
- 50 medium violet red
- 51 midnight blue

- 52** orange
- 53** orange red
- 54** orchid
- 55** pale green
- 56** pink
- 57** plum
- 58** salmon
- 59** sea green
- 60** sienna
- 61** sky blue
- 62** slate blue
- 63** spring green
- 64** steel blue
- 65** tan
- 66** thistle
- 67** turquoise
- 68** violet
- 69** violet red
- 70** wheat
- 71** yellow green
- 72** alice blue
- 73** antique white
- 74** azure
- 75** beige
- 76** bisque
- 77** blanched almond
- 78** burlywood
- 79** chartreuse
- 80** chocolate
- 81** cornsilk
- 82** crimson
- 83** dark blue
- 84** dark cyan

- 85** dark goldenrod
- 86** dark khaki
- 87** dark magenta
- 88** dark orange
- 89** dark red
- 90** dark salmon
- 91** dark sea green
- 92** dark violet
- 93** deep pink
- 94** deep sky blue
- 95** dodger blue
- 96** floral white
- 97** gainsboro
- 98** ghost white
- 99** honeydew
- 100** hot pink
- 101** indigo
- 102** ivory
- 103** lavender
- 104** lavender blush
- 105** lawn green
- 106** lemon chiffon
- 107** light coral
- 108** light cyan
- 109** light goldenrod yellow
- 110** light green
- 111** light pink
- 112** light salmon
- 113** light sea green
- 114** light sky blue
- 115** light slate gray
- 116** light yellow
- 117** linen
- 118** medium purple

- 119** mint cream
- 120** misty rose
- 121** moccasin
- 122** navajo white
- 123** old lace
- 124** olive drab
- 125** pale goldenrod
- 126** pale turquoise
- 127** pale violet red
- 128** papaya whip
- 129** peach puff
- 130** peru
- 131** powder blue
- 132** rosy brown
- 133** royal blue
- 134** saddle brown
- 135** sandy brown
- 136** sea shell
- 137** slate gray
- 138** snow
- 139** tomato
- 140** white smoke

10.2.2 Software Components

The software components include the following:

- Host computer operating system and its associated editing, debugging and terminal emulator applications.
- Python Virtual Machine and its associated Python source code compiler and byte code interpreter.
- The "tsWxGTUI_PyVx" Toolkit and its associated library of general purpose, re-usable utility modules.

The "tsWxGTUI_PyVx" Toolkit library emulates a subset of the Application Programming Interface (API) of the C++ language based "wxWidgets" Graphical User Interface Toolkit and its Python language based "wxPython" wrapper.

Whereas "wxWidgets" and "wxPython" support local pixel-mode terminals, the "tsWxGTUI Toolkit""tsWxGTUI_PyVx" Toolkit is designed for use by developers of Python language based applications that will run on Unix-like platforms having local or remote character-mode terminals. It enables a developer to do the following:

- 1 Port "wxPython" applications or create terminal-independent ones that can run in the local or remote console session of a computer running:
 - a) Linux (such as Debian GNU, Fedora, Gentoo, Hard Hat, Mandriva, Red Hat, SuSE, Ubuntu, Yellow Dog)
 - b) Unix (such as BSD/OS, FreeBSD, HP-UX, IBM-AIX, IRIX, Mac OS X, SCO, Solaris)
 - c) Windows (such as Windows 8 Professional, 7 Professional, Vista Professional, XP Professional, 2000 Professional) when augmented with Cygwin, the free Linux-like environment for 32-bit/64-bit Windows
- 2 Interface, via a platform's "ncurses" or "curses" programming library, with an operator selected and configured terminal emulator:
 - a) Cygwin console
 - b) VT100
 - c) Xterm
 - d) Xterm-color
 - e) Xterm-256color (*Note: Only 8-color palette is available. Availability of the 256-color palette is beyond the scope of the "tsWxGTUI_PyVx" developer. It requires that each system administrator assume the responsibility for re-compiling, building, installing and trouble shooting the wide character version of "ncurses" and its Python wrapper.*)
- 3 Display multiple rows and columns of text and a limited set of box drawing, line drawing and pseudo graphics characters in color or black and white (with normal, bold, reverse, standout and blinking attributes) in one or more windows on a two-dimensional console screen. The color palette depends on the operator selected terminal emulator:
 - a) Xterm-256color terminals support the standard 68-color "wxPython" palette.

- b) Xterm-color terminals support the standard 8-color curses palette which will be substituted for application references to the standard 68-color "wxPython" palette.
 - c) Xterm terminals support the standard 8-color curses palette which will be substituted for application references to the standard 68-color "wxPython" palette.
 - d) VT100 terminal support a single phosphor dependent color (such as green, amber or white on black).
 - e) Cygwin terminals support the standard 8-color curses palette which will be substituted for application references to the standard 68-color "wxPython" palette.
- 4** Accept and interpret operator input from:
- a) console keyboard
 - b) pointing device (mouse, trackball, touchpadn or touchscreen)

10.2.2.1 Operating System

Platform-specific, multi-user (enables a user to login locally and the sae or another user to login remotely), multi-process (enables a user to launch multiple applications and/or multiple instances of the same application), multi-threaded (enables each application to concurrently execute multiple tasks which may independently block while waiting for a triggering event notification) operating systems for 32-bit/64-bit processors:

- 1** Linux (such as Debian GNU, Fedora, Gentoo, Hard Hat, Mandriva, Red Hat, Scientific / Centos, SuSE, Ubuntu, Yellow Dog)
- 2** Mac OS X (Unix-like, Darwin-based such as 10.4/Tiger - 10.9/Mavericks)
- 3** Microsoft Windows (such as Windows 8.1 Professional, 8 Professional, 7 Professional, Vista Professional, XP Professional) when augmented with Cygwin, the free Linux-like Command Line Interface and GNU toolkit from Red Hat
- 4** Unix (such as FreeBSD/PC-BSD, HP-UX, IBM-AIX, IRIX, OpenIndiana, SCO, Solaris/SunOS)

Associated with each operating system are the following platform-specific components:

- Kernels
- Device Drivers
- Run Time Libraries
- Terminal Emulators
- Command Line Interface
- Graphical User Interface
- Network Interface

10.2.2.2 Python

Python technology makes the following contributions to the "tsWxGTUI_PyVx" Toolkit:

- 1 Enables the embedded system application programmer to design processor-independent applications that run, with little or no modification, on platforms with 32-bit and 64-bit processor architectures under "Linux", "Mac OS X", "MS Windows" (with UNIX-style "Cygwin" plug-in) and "UNIX" type operating systems.
- 2 Enables the embedded system application programmer to design terminal-independent applications that run, with little or no modification, on platforms with the pixel-mode and character-mode terminals and terminal emulators available on the local and remote computer platforms.
- 3 Python's "curses" module interfaces to the low-level, character-mode, "nCurses" GUI-toolkit, the de-facto standard for portable advanced terminal handling.
- 4 Converts positioning and sizing dimensions between the pixel-mode units used by "wxPython" applications and the character units used by "nCurses".
- 5 Converts wxPython-style color palettes and font attributes to their "nCurses" equivalents.

Cross-platform Python installations include:

1 Python 2.x

The popular, high-level, cross-platform, second generation Python 2.x programming language which is widely available.

A wiki posting by the Python Software Foundation announced that Python 3.0 was released in 2008. The final 2.x version 2.7 release came out in mid-2010, with a statement of extended support for this end-of-life release.

The 2.x branch will see no new major releases after that.

2 Python 3.x

The popular, high-level, cross-platform, third generation Python 3.x programming language which is growing in availability.

A wiki posting by the Python Software Foundation announced that Python 3.0 was released in 2008. 3.x is under active development and has already seen over five years of stable releases, including version 3.3 in 2012 and 3.4 in 2014. This means that all recent standard library improvements, for example, are only available by default in Python 3.x.

Guido van Rossum (the original creator of the Python language) decided to clean up Python 2.x properly, with less regard for backwards compatibility than is the case for new releases in the 2.x range. The most drastic improvement is the better Unicode support (with all text strings being Unicode by default) as well as saner bytes/Unicode separation.

Besides, several aspects of the core language (such as print and exec being statements, integers using floor division) have been adjusted to be easier for newcomers to learn and to be more consistent with the rest of the language, and old cruft has been removed (for example, all classes are now new-style, "range()" returns a memory efficient iterable, not a list as in 2.x).

3 Associated with each Python installation are the following platform-specific components:

- a) Virtual Machine

- b) Run Time Libraries
- c) Debuggers

10.2.2.3 Application Programs

Platform-specific application programs include:

- File Managers (Midnight Commander on Linux and Unix; Finder and PathFinder on Mac OS X; and Total Commander on Microsoft Windows)
- Text Editors (XEmacs on Linux, Mac OS X, Microsoft Windows and Unix)
- Python Integrated Development Environments (Python Idle, WingIDE etc.)

10.2.3 Architecture

There are two architectural views of the design:

- ***Stand Alone System Architecture*** (on page 228) - Description of the components of and relationship between components of an isolated system operating by itself.
- ***Stand Among System Architecture*** (on page 232) - Description of the components of and relationship between two or more networked systems operating in collaboration with each other.

10.2.3.1 "tsWxGTUI_PyVx" Toolkit Block Diagram

This Block Diagram depicts the organizational, functional and interface relationship between the *TeamSTARS* "tsWxGTUI_PyVx" Toolkit components and users (System Administrator, Software Engineer, System Operator and Field Service personnel):

- 1 the external System Operator interface to "tsToolkitCLI"
- 2 the internal "tsToolkitCLI" interface to "tsToolkitGUI"
- 3 the internal System Operator interfaces:
 - a) to "tsUtilities", "tsToolsCLI" and "tsTestsCLI" via "tsToolkitCLI"
 - b) to "tsToolsGUI" and "tsTestsGUI" via "tsToolkitCLI" and "tsToolkitGUI"

Graphical-style User Interface (tsToolkitGUI)	
<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

10.2.3.2 Stand Alone System Architecture

The *Team*STARS "tsWxGTUI_PyVx" Toolkit provides:

1. Python version-specific components for its Command Line Interface ("tsToolkitCLI")
2. Python version-specific components for its Graphical-style User Interface ("tsToolkitGUI").

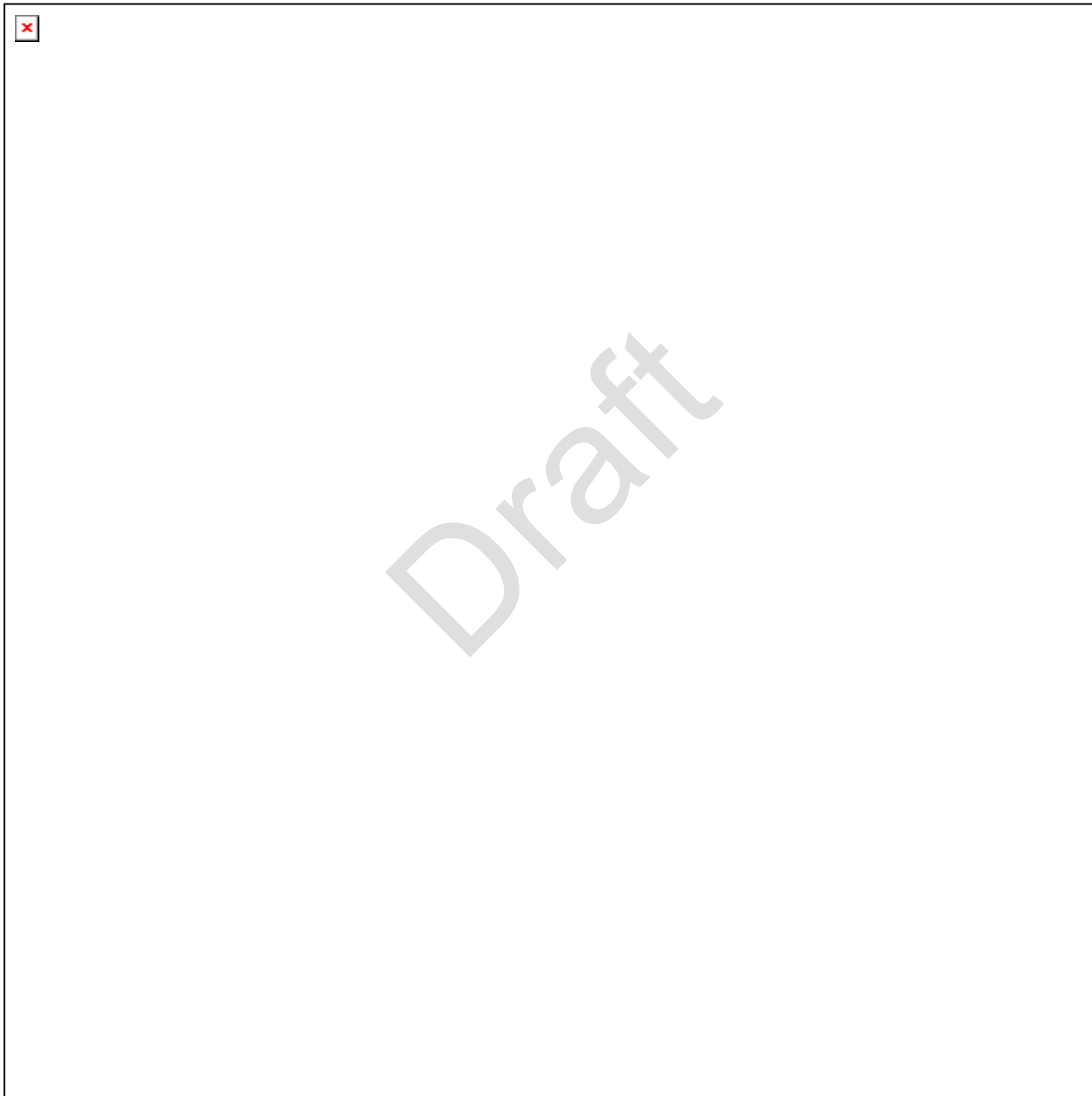
The following description uses the component names as depicted in the Block Diagram

This section depicts and describes the organization, function of and interface between various system hardware and software components and "tsWxGTUI_PyVx" Toolkit users (System Administrator, Software Engineer, System Operator and Field Service personnel):

- 1 the external System Operator interface to "tsToolkitCLI"
- 2 the internal "tsToolkitCLI" interface to "tsToolkitGUI"
- 3 the internal System Operator interfaces:

- a) to "tsLibCLI", "tsToolsCLI" . "tsTestsCLI" and "tsUtilities" via "tsToolkitCLI"
- b) to "tsLibGUI", "tsToolsGUI" and "tsTestsGUI" via "tsToolkitGUI" and "tsToolkitCLI"

This depiction represents a typical Stand Alone System configuration. In this configuration, the optional Network Hardware Interface and its associated Network Device Driver Interface should not be used, even if present, in order to avoid activities that adversely impact system performance.



- 1 Operating System** - The platform specific software (such as Linux, MacOS X, Microsoft Windows and Unix) that coordinates and manages the time-shared use of a platform's processor, memory, storage and input/output hardware resources by multiple application programs and their associated users/operators.

2 Operator Terminal - A device for human interaction that includes:

- a) A Keyboard unit for text input
- b) A Mouse unit (mouse, trackball, trackpad or touchscreen with one or more physical or logical buttons) for selecting one of many displayed GUI objects to initiate an associated action.
- c) A Display unit (1-color "ON"/"OFF" or multi-color two-dimensional screen) for output of text and graphic-style, tiled and overlaid boxes.

3 Terminal Hardware Interface - The platform specific hardware with connections to the device units of the Operator Terminal.

- a) A PS/2 Port is a type of input port developed by IBM for connecting a mouse or keyboard to a Personal Computer. It supports a mini DIN plug containing just 6 pins.
- b) An RS-232C or RS-422 port for connecting a mouse for position and button-click input.
- c) A Universal Serial Bus (USB) port is an industry standard first developed in the mid-1990s that was designed to standardize the connection of computer peripherals (including keyboards, pointing devices, digital cameras, printers, portable media players, disk drives and network adapters) to personal computers, both to communicate and to supply electric power. It has effectively replaced a variety of earlier interfaces, such as serial and parallel ports, as well as separate power chargers for portable devices.

The USB 1.0 specification was introduced in January 1996. It defined data transfer rates of 1.5 Mbit/s "Low Speed" and 12 Mbit/s "Full Speed". The first widely used version of USB was 1.1 (now called "Full-Speed"), which was released in September 1998. Its 12 Mbit/s data rate was intended for higher-speed devices such as disk drives, and its lower 1.5 Mbit/s rate for low data rate devices such as joysticks.

The USB 2.0 (now called "Hi-Speed") specification was released in April 2000. It defined a higher data transfer rate, with the resulting specification achieving 480 Mbit/s, a 40-times increase over the original USB 1.1 specification.

The USB 3.0 specification (now called "SuperSpeed") was preleased in November 2008. It defined an even higher data transfer rate (up to 5 Gbit/s) and was backwards-compatible with USB 2.0. It added a new, higher speed bus called SuperSpeed in parallel with the USB 2.0 bus.

- d) A Video Adapter is a computer circuit card that provides digital-to-analog conversion, video RAM, and a video controller so that data can be sent to a computer's display. It typically adheres to the de facto standard, Video Graphics Array (VGA). VGA describes how data - essentially red, green, blue data streams - is passed between the computer and the display. It also describes the frame refresh rates in hertz. It also specifies the number and width of horizontal lines, which essentially amounts to specifying the resolution of the pixels that are created. VGA supports four different resolution settings and two related image refresh rates. The maximum VGA resolution setting produces a display that is 640 pixels wide by 480 pixels high. For a character font that is 8 pixels wide by 12 pixels high, the longest line of text will be 80 characters wide and there can be up to 40 lines of text displayed at any moment. Higher resolutions, such as SVGA, are supported by more advanced Video Adapters. The higher resolution settings typically require use of proportionally larger displays in order to maintain the size and legibility of the displayed text.

4 Terminal Device Driver - The platform specific software for transforming data (such as single button scan codes, multi-button flags and pointer position) to and from the platform independent formats (such as upper and lower case text, display screen column and row and displayed colors, fonts and special effects) used by the Command Line Interface and Graphical User Interface software.

- 5 Command Line-Style Interface ("tsToolKitCLI")** - The platform specific keywords arguments, positional arguments and their associated values and syntax of text used to request services from the Operating System and various Application Programs.

 - a) **"tsLibCLI"** --- A library of command line building blocks that establishes the POSIX-/Unix-style, run time environment for pre-processing source files, launching application programs, handling events (registering events with date, time and event severity annotations) and configuring console terminal and file system input and output.
 - b) **"tsToolsCLI"** --- A set of command line interface application programs and utility scripts for: checking source code syntax and style via "plint"; generating Unix-style "man page" documentation from source code comments via "pydoc"; and installing, modifying for publication and tracking software development metrics.
 - c) **"tsTestsCLI"** --- A set of command line interface application programs and utility scripts for unit, integration and system level regression testing.
 - d) **"tsUtilities"** --- A library of computer system configuration, diagnostic, installation, maintenance and performance tool components for various host hardware and software platforms.
- 6 Graphical-Style User Interface ("tsToolKitGUI")** - The platform specific tiled, overlaid and click-to-select Frames, Dialogs, Pull-down Menus, Buttons, CheckBoxes, Radio Buttons, Scrollbars and associated keywords, values and syntax of text used to request services from the Operating System and various Application Programs.

 - a) **"tsLibGUI"** --- A library of graphical-style user interface building blocks that establishes the emulated run time environment for the high-level, pixel-mode, "wxPython" GUI Toolkit via the low-level, character-mode, "nCurses" Text User Interface Toolkit.
 - b) **"tsToolsGUI"** --- A set of graphical-style user interface application programs for tracking software development metrics.
 - c) **"tsTestsGUI"** --- A set of graphical-style user interface application programs and command line interface utility scripts for unit, integration and system level regression testing.
- 7 Python Application Program** - The application specific program that performs its service when executed by the Python Virtual Machine.
- 8 Non-Python Application Program** - The application specific program that performs its service when its pre-compiled, platform specific machine code is executed. Typically, these services are used to analyze, edit, view, copy, move or delete those data and log files which are of interest or no longer needed.
- 9 Python Virtual Machine** - The platform specific program that loads, interprets and executes the platform independent source code of a Python language application program.
- 10 Processor, Memory, Storage and Communication Hardware** - Platform specific resources that are required by the Operating System and Application software.
- 11 Network Hardware Interface** - The optional platform specific ethernet, blue-tooth and RS-232 serial port hardware for physical connections between the local system and one or more remote systems. It may also include such external hardware as gateways, routers, network bridges, switches, hubs, and repeaters. It may also include hybrid network devices such as multilayer switches, protocol converters, bridge routers, proxy servers, firewalls, network address translators, multiplexers, network interface controllers, wireless network interface controllers, modems, ISDN terminal adapters, line drivers, wireless access points, networking cables and other related hardware.

- 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.

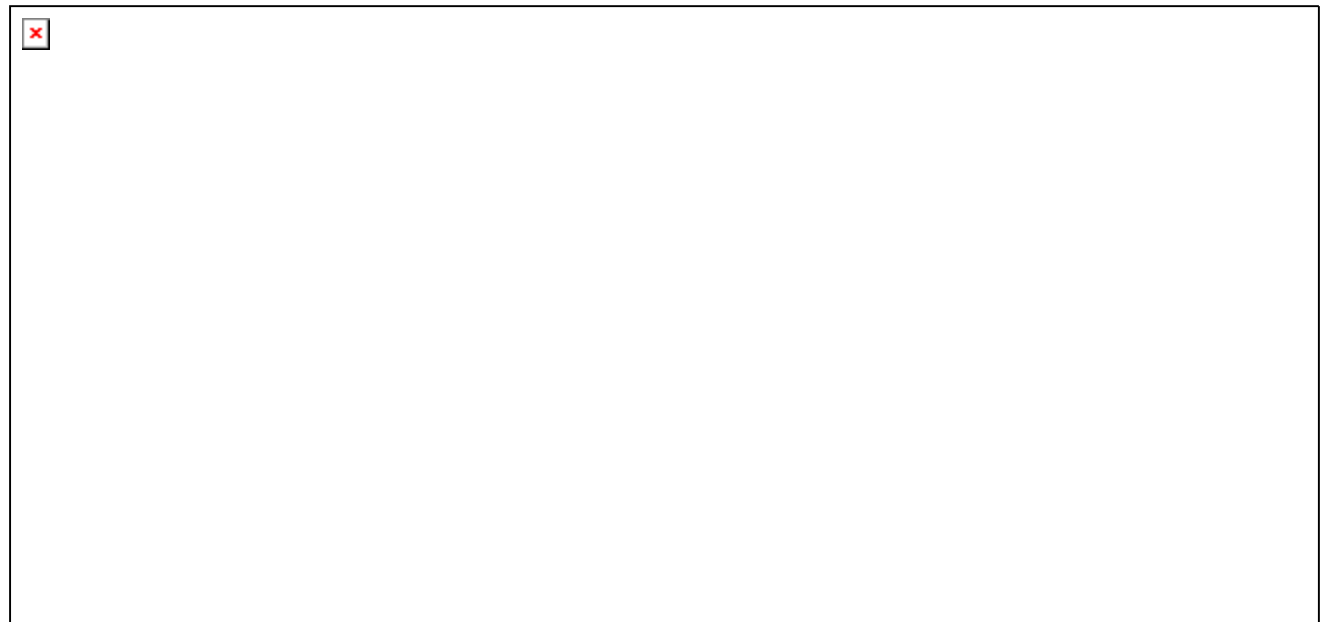
10.2.3.3 Stand Among System Architecture

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

The ***Stand Alone System Architecture*** (on page 228), 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 234) 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.**10.2.3.3.1 Multi-Session Desktop**

From Wikipedia, the free encyclopedia

"In computer science, in particular networking, a session is a semi-permanent interactive information interchange, also known as a dialogue, a conversation or a meeting, between two or more communicating devices, or between a computer and user (see Login session). A session is set up or established at a certain point in time, and then torn down at some later point. An established communication session may involve more than one message in each direction. A session is typically, but not always, stateful, meaning that at least one of the communicating parts needs to save information about the session history in order to be able to communicate, as opposed to stateless communication, where the communication consists of independent requests with responses.

An established session is the basic requirement to perform a connection-oriented communication. A session also is the basic step to transmit in connectionless communication modes. However any unidirectional transmission does not define a session.[1]

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

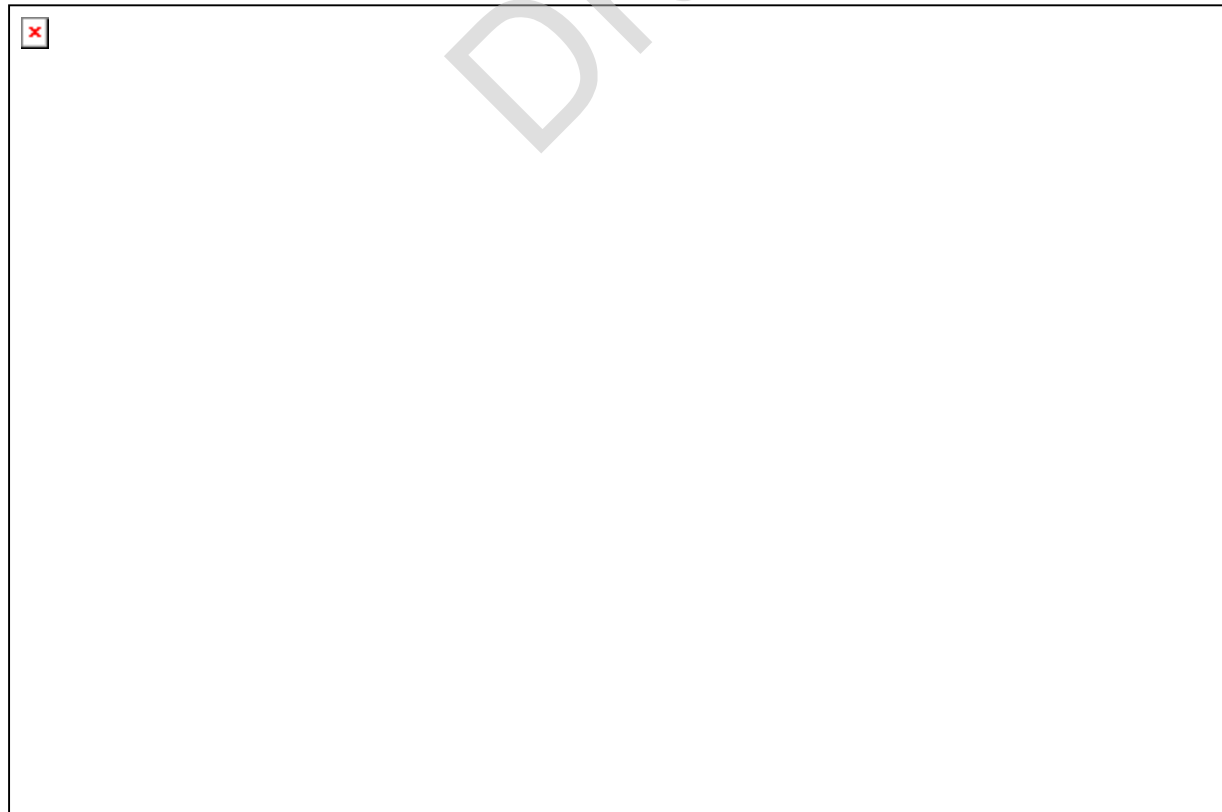
- Application layer examples:
 - HTTP sessions, which allow associating information with individual visitors
 - A telnet remote login session
- Session layer example:
 - A Session Initiation Protocol (SIP) based Internet phone call

- 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).

11 APPENDIX D - SYSTEM INTERNAL INTERFACE REQUIREMENTS

This paragraph shall specify the requirements, if any, imposed on interfaces internal to the system. If all internal interfaces are left to the design or to requirement specifications for system components, this fact shall be so stated. If such requirements are to be imposed, paragraph 3.3 of this DID provides a list of topics to be considered.

1 *Toolkit Architecture* (on page 237)

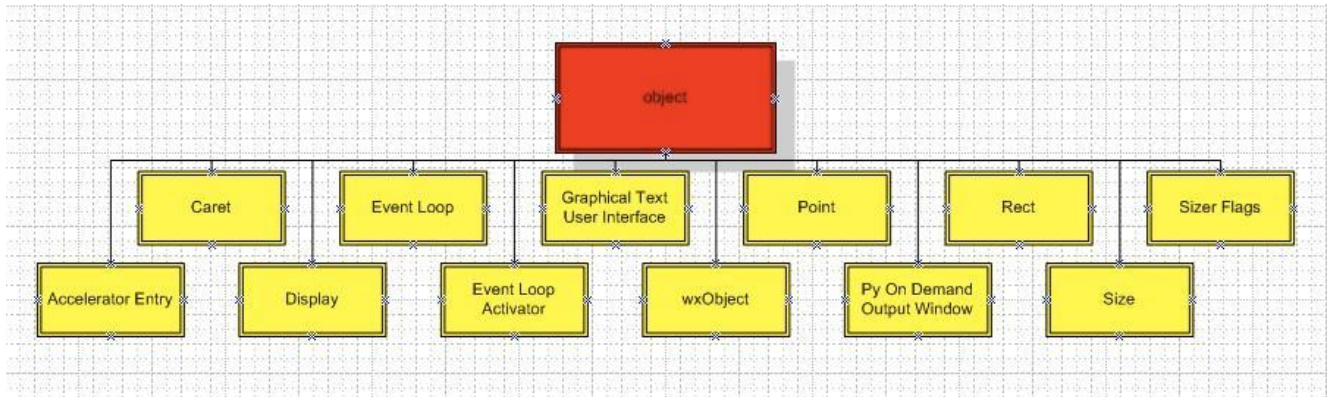
11.1 Toolkit Architecture

The following examples highlight the hierarchical dependency relationship between various tsWxGTUI objects:

- ***"object" Dependents*** (on page 238) - Python's "object" class is the parent class for the various wxPython classes.
- ***"Object" Dependents*** (see ***"wxObject" Dependents*** on page 238) - wxPython's "Object" class is a decendent of Python's "object" class and the parent class for the various wxPython classes, some of which have their own descendants.
- ***"EvtHandler" Dependents*** (see ***"wxEvtHandler" Dependents*** on page 239) - wxPython's "EvtHandler" class is a descendant of wxPython's "Object" class and the parent class for the various wxPython classes, some of which have their own descendants.
- ***"Window" Dependents*** (see ***"wxWindow" Dependents*** on page 240) - wxPython's "Window" class is a descendant of wxPython's "EvtHandler" class. It is the base class for all windows and represents any visible object on the screen.

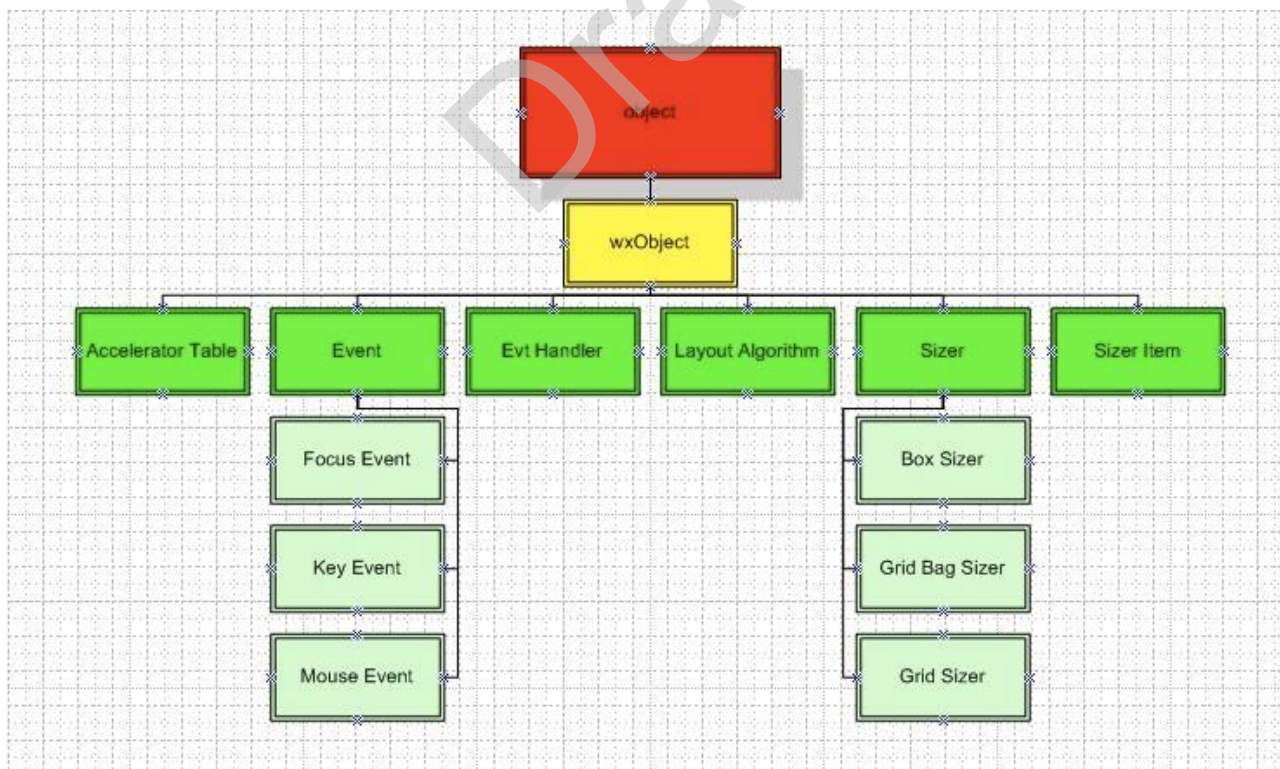
11.1.1 "object" Dependents

Python's "object" class is the parent class for the following wxPython classes.



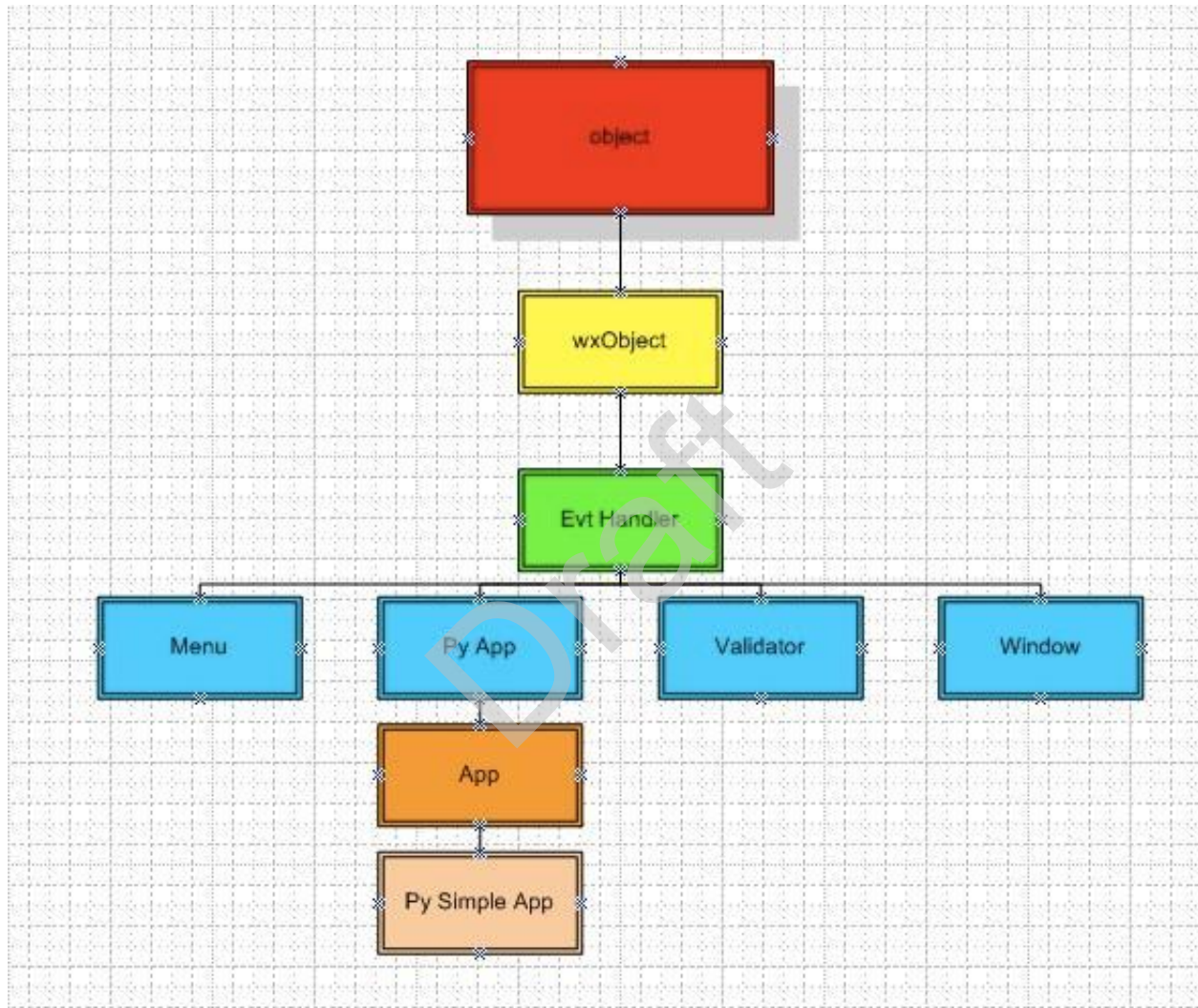
11.1.2 "wXObject" Dependents

wxPython's "Object" class is a descendant of Python's "object" class and the parent class for the following wxPython classes, some of which have their own descendants.



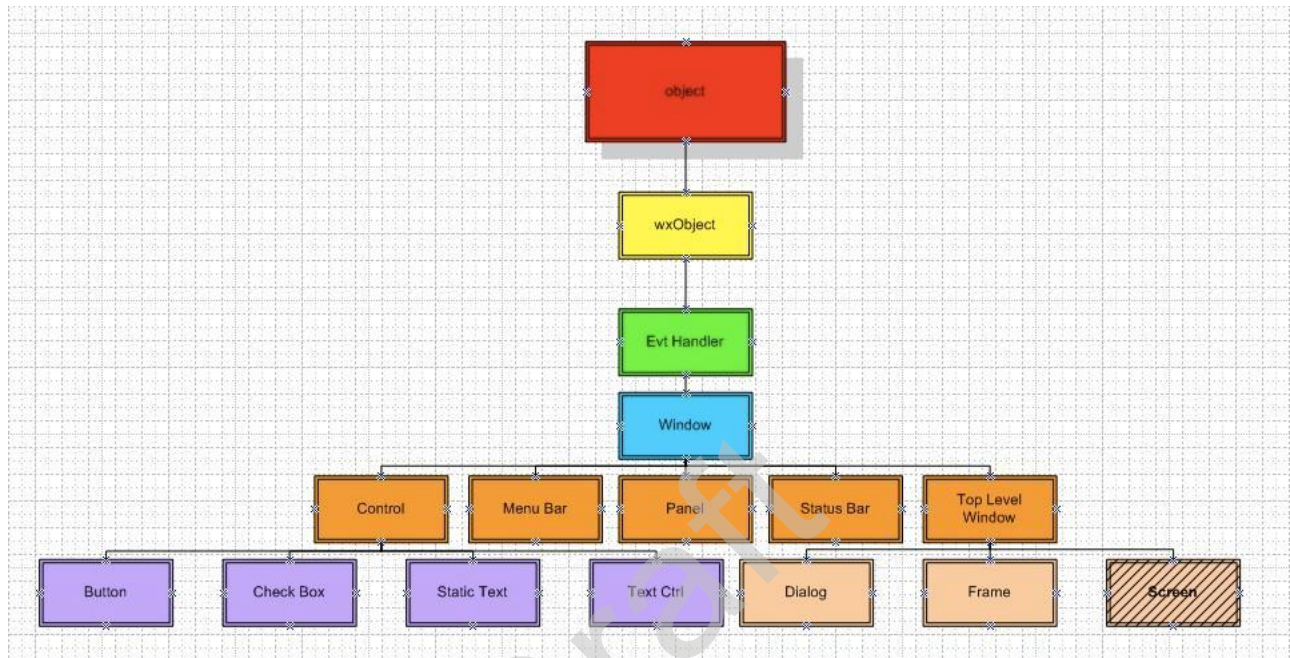
11.1.3 "wxEvtHandler" Dependents

wxPython's "EvtHandler" class is a descendant of wxPython's "Object" class and the parent class for the following wxPython classes, some of which have their own descendants.



11.1.4 "wxWindow" Dependents

wxPython's "Window" class is a descendant of wxPython's "EvtHandler" class. It is the base class for all windows and represents any visible object on the screen.



12 APPENDIX E - SYSTEM INTERNAL DATA REQUIREMENTS

This paragraph shall specify the requirements, if any, imposed on data internal to the system. Included shall be requirements, if any, on databases and data files to be included in the system. If all decisions about internal data are left to the design or to requirements specifications for system components, this fact shall be so stated. If such requirements are to be imposed, paragraphs 3.3.x.c and 3.3.x.d of this DID provide a list of topics to be considered.

Draft

Draft

13 APPENDIX F - ADAPTATION REQUIREMENTS

This paragraph shall specify the requirements, if any, concerning installation-dependent data that the system is required to provide (such as site-dependent latitude and longitude or site-dependent state tax codes) and operational parameters that the system is required to use that may vary according to operational needs (such as parameters indicating operation-dependent targeting constants or data recording).

Draft

Draft

14 APPENDIX G - SAFETY REQUIREMENTS

This paragraph shall specify the system requirements, if any, concerned with preventing or minimizing unintended hazards to personnel, property, and the physical environment. Examples include restricting the use of dangerous materials; classifying explosives for purposes of shipping, handling, and storing; abort/escape provisions from enclosures; gas detection and warning devices; grounding of electrical systems; decontamination; and explosion proofing. This paragraph shall include the system requirements, if any, for nuclear components, including, as applicable, requirements for component design, prevention of inadvertent detonation, and compliance with nuclear safety rules.

Draft

Draft

15 APPENDIX H - SECURITY AND PRIVACY REQUIREMENTS

This paragraph shall specify the system requirements, if any, concerned with maintaining security and privacy. The requirements shall include, as applicable, the security/privacy environment in which the system must operate, the type and degree of security or privacy to be provided, the security/privacy risks the system must withstand, required safeguards to reduce those risks, the security/privacy policy that must be met, the security/privacy accountability the system must provide, and the criteria that must be met for security/privacy certification/accreditation.

Draft

Draft

16 APPENDIX I - SYSTEM ENVIRONMENT REQUIREMENTS

This paragraph shall specify the requirements, if any, regarding the environment in which the system must operate. Examples for a software system are the computer hardware and operating system on which the software must run. (Additional requirements concerning computer resources are given in the next paragraph). Examples for a hardware-software system include the environmental conditions that the system must withstand during transportation, storage, and operation, such as conditions in the natural environment (wind, rain, temperature, geographic location), the induced environment (motion, shock, noise, electromagnetic radiation), and environments due to enemy action (explosions, radiation).

Draft

Draft

17 APPENDIX J - COMPUTER RESOURCE REQUIREMENTS

This paragraph shall be divided into the following subparagraphs. Depending upon the nature of the system, the computer resources covered in these subparagraphs may constitute the environment of the system (as for a software system) or components of the system (as for a hardware-software system).

To demonstrate computer resource requirements and usability considerations for various application-specific capabilities, the "tsWxGTUI_PyVx" Toolkit developers have used the following platform configurations:

Draft

Draft

Make & Model	Hardware	Software
Apple 27" iMac Desktop	<p>2013-model year, 3.5 GHz Intel Quad Core i7 processor-based desktop with 16 GB RAM, 2560x1440 pixel resolution display and sufficient performance, resources and expansion capabilities to serve as the high-level baseline development and operator platform.</p> <ul style="list-style-type: none"> Its 3 TB 7200 RPM internal hard drive had 128 GB Solid State Flash memory and was used to run Apple's Mac OS X 10.9 and the hypervisor virtualization applications (Parallels Desktop 9 and VMware Fusion 5) that supported various guest operating systems. Its 3 TB internal hard drive was also used to store and run configured versions of the Fedora Linux (20), Scientific Linux (6.4), Ubuntu Linux (12.04), Microsoft Windows (8.1/8 Pro, 7 Pro and XP Pro) and Unix (PC-BSD (9.2), OpenSolaris 11/OpenIndiana 151A8) guest operating systems. Hewlett-Packard Company P2015DN Series (265C8) LaserJet Printer connected via Ethernet Hewlett-Packard Company Officejet Pro 8500A (A910) e-All-in-One Series Printer connected via Wi-Fi or USB. 	<p>Host Operating System</p> <ul style="list-style-type: none"> Apple's Unix-like, Darwin-based Mac OS X 10.9 with Python 2.6.8, 2.7.5, 3.2.2 <p>Host Hypervisor Virtualization Applications</p> <ul style="list-style-type: none"> Parallels Desktop 9 VMware Fusion 5 <p>Concurrent or Interchangeable Guest Operating Systems (cloned from Apple 17" MacBook Pro Laptop and configured to share host drives and use 1440 x 900 display)</p> <ul style="list-style-type: none"> Linux (Fedora 20 64-bit, OpenSuSE 12.2 64-bit, Scientific (CentOS) 6.4-6.5 64-bit, Ubuntu 12.04 32-bit) with Python 2.7 and 3.2. Windows (8.1 Professional 32-bit, 8 Professional 32-bit, 7 Professional 32-bit with SP1, XP Professional 32-bit with SP3) with POSIX-like Cygwin plug-in and Python 2.6, 2.7, 3.2 and 3.3 UNIX (PC-BSD 9.2-10.0 64-bit without Parallels Tools, OpenIndiana 151A8 32-bit without Parallels Tools, a replacement for unreleased OpenSolaris 11/SunOS 5.11) with Python 2.6.4 running on Apple MacBook Pro
Apple 17" MacBook Pro Laptop	<p>2007-model year, 2.33 GHz Intel Core 2 Duo processor-based laptop with 4 GB RAM, 1920x1200 pixel resolution display and sufficient performance, resources and expansion capabilities to serve as the moderate-level baseline development and operator platform.</p> <ul style="list-style-type: none"> Its 160 GB internal hard drive was used to run Apple's Mac OS X 10.7 and the hypervisor virtualization applications (Parallels Desktop 8 and VMware Fusion 5) that supported various guest operating systems. Its 1.5 TB external hard drive was used to store and run configured versions of the Ubuntu Linux (12.04), Microsoft Windows (8 Pro, 7 Pro and XP Pro) and Unix (OpenSolaris 11/OpenIndiana 151) guest operating systems. Hewlett-Packard Company P2015DN Series (265C8) LaserJet Printer connected 	<p>Host Operating System</p> <ul style="list-style-type: none"> Apple's Unix-like, Darwin-based Mac OS X 10.7 with Python 2.6.8, 2.7.5, 3.2.2 <p>Host Hypervisor Virtualization Applications</p> <ul style="list-style-type: none"> Parallels Desktop 8 VMware Fusion 5 <p>Interchangeable Guest Operating Systems (configured to share host drives and use 1440 x 900 display)</p> <ul style="list-style-type: none"> Linux (Fedora 20 64-bit, OpenSuSE 12.2 64-bit, Scientific (CentOS) 6.4-6.5 64-bit, Ubuntu 12.04 32-bit) with Python 2.7 and 3.2. Windows (8.1 Professional 32-bit, 8 Professional 32-bit, 7 Professional 32-bit with SP1, XP Professional 32-bit with SP3) with POSIX-like Cygwin plug-in and Python 2.6, 2.7, 3.2 and 3.3 UNIX (PC-BSD 9.2-10.0 64-bit without

	<p>via Ethernet</p> <ul style="list-style-type: none"> ▪ Hewlett-Packard Company Officejet Pro 8500A (A910) e-All-in-One Series Printer connected via Wi-Fi or USB. 	<p>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</p>
<p>Dell 15.6" Inspiron 7000 Laptop</p>	<p>1998-model year, 366 MHz Intel Pentium II-based laptop with 384 MB RAM, 640x480/1024x768 pixel resolution display and expansion capabilities (marginal resources and performance) sufficient enough to serve as the low-level baseline development and operator platform.</p> <ul style="list-style-type: none"> ▪ Its interchangeable 32 GB (4200 RPM) ATA hard drives were used to run Windows XP Pro or to run Ubuntu 12.04 LTS Linux operating system kernel with GNU toolchain. ▪ The platform's limited memory and available PCMCIA network adapters were compatible with: (1) Windows XP Pro via Xircom RBEM56G-100, a RealPort CardBus Ethernet 10/100+Modem 56 or 3CPM 3CRWE62092A Wireless LAN PC Card; (2) Ubuntu 12.04 LTS Linux via LINKSYS WPC11 Instant Wireless Network PC Card. The platform was incompatible with later versions of Windows and with other Linux distributions. later versions of Windows or with other Linux distributions. ▪ Hewlett-Packard Company Photosmart C3180 All-in-One Printer, Scanner, Copier connected via USB. 	<p>Interchangeable Host Operating System</p> <ul style="list-style-type: none"> ▪ Windows XP Professional with SP3) with POSIX-like Cygwin plug-in and Python 2.6, 2.7, 3.2 and 3.3 ▪ Linux (Ubuntu 12.04) with Python 2.7 and 3.3

18 APPENDIX K - SYSTEM QUALITY FACTORS

This paragraph shall specify the requirements, if any, pertaining to system quality factors. Examples include quantitative requirements concerning system functionality (the ability to perform all required functions), reliability (the ability to perform with correct, consistent results -- such as mean time between failure for equipment), maintainability (the ability to be easily serviced, repaired, or corrected), availability (the ability to be accessed and operated when needed), flexibility (the ability to be easily adapted to changing requirements), portability of software (the ability to be easily modified for a new environment), reusability (the ability to be used in multiple applications), testability (the ability to be easily and thoroughly tested), usability (the ability to be easily learned and used), and other attributes.

Draft

Draft

19 APPENDIX L - DESIGN AND CONSTRUCTION CONSTRAINTS

This paragraph shall specify the requirements, if any, that constrain the design and construction of the system. For hardware-software systems, this paragraph shall include the physical requirements imposed on the system. These requirements may be specified by reference to appropriate commercial or military standards and specifications. Examples include requirements concerning:

- a. Use of a particular system architecture or requirements on the architecture, such as required subsystems; use of standard, military, or existing components; or use of Government/acquirer-furnished property (equipment, information, or software)
- b. Use of particular design or construction standards; use of particular data standards; use of a particular programming language; workmanship requirements and production techniques
- c. Physical characteristics of the system (such as weight limits, dimensional limits, color, protective coatings); interchangeability of parts; ability to be transported from one location to another; ability to be carried or set up by one, or a given number of, persons
- d. Materials that can and cannot be used; requirements on the handling of toxic materials; limits on the electromagnetic radiation that the system is permitted to generate
- e. Use of nameplates, part marking, serial and lot number marking, and other identifying markings
- f. Flexibility and expandability that must be provided to support anticipated areas of growth or changes in technology, threat, or mission

Draft

20 APPENDIX M - PERSONNEL-RELATED REQUIREMENTS

This paragraph shall specify the system requirements, if any, included to accommodate the number, skill levels, duty cycles, training needs, or other information about the personnel who will use or support the system. Examples include requirements for the number of work stations to be provided and for built-in help and training features. Also included shall be the human factors engineering requirements, if any, imposed on the system. These requirements shall include, as applicable, considerations for the capabilities and limitations of humans, foreseeable human errors under both normal and extreme conditions, and specific areas where the effects of human error would be particularly serious. Examples include requirements for adjustable-height work stations, color and duration of error messages, physical placement of critical indicators or buttons, and use of auditory signals.

Draft

Draft

21 APPENDIX N - TRAINING-RELATED REQUIREMENTS

This paragraph shall specify the system requirements, if any, pertaining to training. Examples include training devices and training materials to be included in the system.

Draft

Draft

22 APPENDIX O - LOGISTICS-RELATED REQUIREMENTS

This paragraph shall specify the system requirements, if any, concerned with logistics considerations. These considerations may include: system maintenance, software support, system transportation modes, supply-system requirements, impact on existing facilities, and impact on existing equipment.

Draft

Draft

23 APPENDIX P - OTHER REQUIREMENTS

This paragraph shall specify additional system requirements, if any, not covered in the previous paragraphs. Examples include requirements for system documentation, such as specifications, drawings, technical manuals, test plans and procedures, and installation instruction data, if not covered in other contractual documents.

Draft

Draft

24 APPENDIX Q - PACKAGING REQUIREMENTS

This section shall specify the requirements, if any, for packaging, labeling, and handling the system and its components for delivery. Applicable military specifications and standards may be referenced if appropriate.

Draft

Draft

25 APPENDIX R - PRECEDENCE AND CRITICALITY OF REQUIREMENTS

This paragraph shall specify, if applicable, the order of precedence, criticality, or assigned weights indicating the relative importance of the requirements in this specification. Examples include identifying those requirements deemed critical to safety, to security, or to privacy for purposes of singling them out for special treatment. If all requirements have equal weight, this paragraph shall so state.

Draft

Draft

26 APPENDIXES (Software Requirements)

27 APPENDIX A - Nature of System and Software

The nature of the system and software depends on the observer's functional role:

- **System Administrator** (on page 275) - In this role, the perspective and activities are that of an individual who will specify, plan, supervise and/or perform the installation, configuration, maintenance, repair and support of the computer hardware, operating system, application software and network to be used by Software Engineers and System Operators.
- **Software Engineer** (on page 275) - In this role, the perspective and activities are that of an individual who will specify, plan, supervise and/or perform the design, development, coding, testing, debugging, maintenance and support of application programs, with Command Line Interface or Graphical User Interface, to be used by System Operators.
- **System Operator** (on page 276) - In this role, the perspective and activities are that of an individual who will use various application programs, with associated Command Line Interface or Graphical User Interface, to interactively supervise communication, control, data base, diagnostic, instrumentation or simulation activities.

Regardless of functional role, the general nature of user interfaces and systems are summarized in the following sections:

- **Command Line and Graphical User Interfaces** (on page 272) - Describes the two types of interfaces a human operator typically uses to interact with a computer system.
- **Hardware Components** - Describes typical computer processor, memory, storage, networking, input and output hardware.
- **Software Components** - Describes typical operating system, Python Virtual Machine and "tsWxGTUI_PyVx" Toolkit software

Typical hardware and software configurations are identified in the following section:

- **System Configurations** (on page 273) - Identifies various assemblies of computer products that would be suitable for use by Software Engineers or System Operators.

Typical "tsWxGTUI_PyVx" Toolkit scenarios for the System Administrator, Software Engineer and System Operator are summarized in the following sections:

- Application Programming Interface - The following description is taken from Wikipedia, the free encyclopedia, at http://en.wikipedia.org/wiki/Application_programming_interface:
 - An application programming interface (API) is a particular set of rules ('code') and specifications that software programs can follow to communicate with each other. It serves as an interface between different software programs and facilitates their interaction, similar to the way the user interface facilitates interaction between humans and computers. API can be created for applications, libraries, operating systems, etc., as a way of defining their "vocabularies" and resources request conventions (e.g. function-calling conventions).
 - It may include specifications for routines, data structures, object classes, and protocols used to communicate between the consumer program and the implementer program of the API.
- *Use Case(s)* (on page 274) - The following description is taken from Wikipedia, the free encyclopedia, at http://en.wikipedia.org/wiki/Use_case:
 - A use case in software engineering and systems engineering, is a description of steps or actions between a user (or "actor") and a software system which leads the user towards something useful.[1] The user or actor might be a person or something more abstract, such as an external software system or manual process.
 - Use cases are a software modeling technique that helps developers determine which features to implement and how to gracefully resolve errors.[2]
 - Within systems engineering, use cases are used at a higher level than within software engineering, often representing missions or stakeholder goals. The detailed requirements may then be captured in SysML requirement diagrams or similar mechanisms.

27.1 Command Line and Graphical User Interfaces

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

- **Command Line Interface:** Humans began interacting with computers via a mechanical terminal device, such as a teletype, and the computer's Command Line Interface (CLI). The computer would print out a single line of text to prompt the user for input. It would wait for the user to type in a command. In responding to the command, it would print out the appropriate data followed by a prompt for the next command. For additional details see: *Text Mode* (on page 52).
- **Graphical User Interface:** Over the years, human interactions have become more complex. Modern computer now concurrently wait for multiple user inputs while updating multiple output areas on an electronic terminal device, such as a laptop computer. Today, the Graphical User Interface (GUI) is more convenient to use than the Command Line Interface. It more fully displays the information and actions available to a user. Considerable time and effort goes into the design and implementation of a GUI. Technological advances have created toolkits that provide standard class, method and data components that enable the developer to design and implement only the application specific components. For additional details see: *Graphical Mode* (on page 56).

27.2 System Configurations

The following assemblies of computer hardware and software products would be suitable for use by:

- Software Engineers, who develop application programs with the "tsWxGTUI_PyVx" toolkit
- System Operators, who use the application programs

The "tsWxGTUI_PyVx" Toolkit and the application programs produced by it are designed for use on commonly available Laptop, Desktop or Workstation type platforms. It is only necessary that the platform be appropriate for the application and include the following:

- 1 keyboard input device
- 2 pointing input device (mouse, touchpad, touchscreen or trackball)
- 3 display output device (with at least 640 x 480 pixel size)
- 4 Unix-style "nCurses" program library
- 5 Python virtual machine with its Python programming language interpreter
- 6 Computer processor, memory, disk storage and other peripheral devices

Operating System Software	Add-On Software	Central Processing Unit Hardware
Apple Mac OS X (10.4.x-10.7.x etc.)	<ul style="list-style-type: none"> ▪ Python 2.6.x ▪ Python 2.7.x ▪ Python 3.1.x ▪ Python 3.2.x ▪ Parallels Desktop 5 / 6 / 7 for Mac (runs various Linux and Windows Virtual Machines on Mac OS X using shared computer processor, memory, peripheral, and network resources) 	<ul style="list-style-type: none"> ▪ Intel & AMD / PC Compatibles (32-bit & 64-bit) ▪ Freescale PowerPC Compatibles (32-bit & 64-bit)
Linux (Fedora, Mandriva, Red Hat, Susse, Ubuntu etc.)	<ul style="list-style-type: none"> ▪ Python 2.6.x ▪ Python 2.7.x ▪ Python 3.1.x ▪ Python 3.2.x 	<ul style="list-style-type: none"> ▪ Intel & AMD / PC Compatibles (32-bit & 64-bit) ▪ Freescale PowerPC Compatibles (32-bit & 64-bit)
Microsoft Windows (such as 2000, XP, Vista, 7 etc.)	<ul style="list-style-type: none"> ▪ Cygwin 1.7.x, Linux-style operating environment ▪ Python 2.6.x ▪ Python 2.7.x ▪ Python 3.1.x ▪ Python 3.2.x 	<ul style="list-style-type: none"> ▪ Intel & AMD / PC Compatibles (32-bit & 64-bit)

Oracle/Sun Solaris (such as Solaris 9, 10, 11 etc.)	<ul style="list-style-type: none">▪ Python 2.6.x▪ Python 2.7.x▪ Python 3.1.x▪ Python 3.2.x	<ul style="list-style-type: none">▪ Intel & AMD / PC Compatibles (32-bit & 64-bit)▪ Oracle/Sun SPARC or compatibles
Unix (such as FreeBSD, HP-UX, IBM-AIX)	<ul style="list-style-type: none">▪ Python 2.6.x▪ Python 2.7.x▪ Python 3.1.x▪ Python 3.2.x	<ul style="list-style-type: none">▪ Intel & AMD / PC Compatibles (32-bit & 64-bit)▪ Freescale PowerPC Compatibles (32-bit & 64-bit)

27.3 Use Case(s)

From Wikipedia, the free encyclopedia

"In software and systems engineering, a use case is a list of steps, typically defining interactions between a role (known in Unified Modeling Language (UML) as an "actor") and a system, to achieve a goal. The actor can be a human, an external system, or time.

In systems engineering, use cases are used at a higher level than within software engineering, often representing missions or stakeholder goals. The detailed requirements may then be captured in Systems Modeling Language (SysML) or as contractual statements.

As an important requirement technique, use cases have been widely used in modern software engineering over the last two decades. Use case driven development is a key characteristic of process models and frameworks like Unified Process (UP), Rational Unified Process (RUP), Oracle Unified Method (OUM), etc. With its iterative and evolutionary nature, use case is also a good fit for agile development."

The use case(s) describe the expected activities associated with various functional roles:

- 1 Operating System Administration**
- 2 Python Language Administration**
- 3 "tsWxGTUI_PyVx" Toolkit Development**
- 4 Python Application Development**
- 5 Python Application Usage**
- 6 System Troubleshooting & Maintenance**

27.3.1 System Administrator

Based on: http://en.wikipedia.org/wiki/System_administrator

A System Administrator is typically responsible for supervising and/or performing the following:

- Installation, configuration, maintenance, repair and support of all system hardware, operating system and application software and network components.
- Scripting or light programming.
- Project management for systems-related projects.
- Supervising or training computer operators.
- Consultant for computer problems beyond the knowledge of technical support staff.
- To perform his or her job well, a System Administrator must demonstrate a blend of technical skills and responsibility.

The subject matter of System Administration includes computer systems and the ways people use them in an organization. This entails a knowledge of operating systems and applications, as well as hardware and software troubleshooting, but also knowledge of the purposes for which people in the organization use the computers. Perhaps the most important skill for a System Administrator is problem solving -- frequently under various sorts of constraints and stress. The System Administrator is on call when a computer system goes down or malfunctions, and must be able to quickly and correctly diagnose what is wrong and how best to fix it. System Administrators are not Software Engineers. It is not usually within their duties to design or write new application software. However, System Administrators must understand the behavior of software in order to deploy it and to troubleshoot problems, and generally know several programming languages used for scripting or automation of routine tasks.

27.3.2 Software Engineer

Derived from: <http://www.bls.gov/oco/ocos303.htm>

Computer Software Engineers design and develop software for use by System Operators. They perform the following:

- Begin by analyzing users' needs and then design, test, and develop software to meet those needs.
- Apply the theories and principles of computer science and mathematical analysis to create, test, and evaluate the systems, application software and command line or graphical user interface that make computers work.
- During this process they create flowcharts, diagrams, and other documentation, and may also create the detailed sets of instructions, called algorithms, that actually tell the computer what to do.
- They may also be responsible for converting these instructions into a computer language, a process called programming or coding, but this usually is the responsibility of computer programmers.

27.3.3 System Operator

Derived from <http://whatis.techtarget.com/definition/system-operator-sysop>

A System Operator is the person who runs the day-to-day operation of the computer system. The term suggests a person who is available when the system is.

The System Operator is ultimately the end-user of application programs, with associated command line or graphical user interfaces, that interactively supervise various communication, control, data base, diagnostic, instrumentation or simulation activities.

At earlier stages in the computer system's installation, configuration and software development process, the System Operator may temporarily be a **System Administrator** (on page 275) or **Software Engineer** (on page 275).

At later stages in the computer system's maintenance, the System Operator may temporarily be one of the **Field Service Personnel**.

The activities are somewhat role-specific and system specific. The following use cases illustrate the similarities and differences:

- *Apple Mac OS X Use Case* (see "*Apple Mac OS X Unix-like, Darwin-based Use Case*" on page 276)
- *Cygwin Use Case* (on page 278)
- Microsoft Windows Use Case
- *Ubuntu Linux Use Case* (on page 279)
- *FreeBSD/PC-BSD Unix Use Case* (on page 277)

27.3.3.1 Apple Mac OS X Unix-like, Darwin-based Use Case

Operation of a typical computer system advances through various stages:

- 1 Manual Power-ON of Host Computer Hardware (Booting)
- 2 Automatic reset of Host Computer Hardware
- 3 Automatic startup of Host Computer Operating System Software
- 4 Automatic startup of Host Computer Pixel-mode Graphical User Interface Software (**login**)
- 5 Optional installation of new or updated Host Computer Operating System Libraries and Software
- 6 Optional installation of new or updated "tsWxGTUI_PyVx" Toolkit Application Libraries and Software
- 7 Manual startup of Host Computer Command Line Interface Software (**Applications -> iTerm** or **Applications -> Utilities -> terminal**)
- 8 Manual startup of "tsWxGTUI_PyVx" Toolkit Application Software (**python3.2 test_tsWxWidgets.py**)
 - a) Automatic startup of Character-mode Graphical User Interface Software

- b) Automatic startup of Character-mode Top Level GUI objects
- c) Automatic startup of Character-mode Lower Level GUI objects
- d) Automatic shutdown of Character-mode Lower Level GUI objects
- e) Automatic shutdown of Character-mode Top Level GUI objects
- f) Automatic shutdown of Character-mode Graphical User Interface Software (**ctrl-c**)
- 9** Manual shutdown of Host Computer Command Line Interface Software (**exit**)
- 10** Manual shutdown of Host Computer Pixel-mode Graphical User Interface Software (**logout**)
- 11** Manual shutdown of Host Computer Operating System Software (**shutdown**)
- 12** Manual Power-OFF of Host Computer Hardware (Shutdown)

27.3.3.2 FreeBSD/PC-BSD Unix Use Case

Operation of a typical computer system advances through various stages:

- 1** Manual Power-ON of Host Computer Hardware (Booting)
- 2** Automatic reset of Host Computer Hardware
- 3** Automatic startup of Host Computer Operating System Software
- 4** Automatic startup of Host Computer Pixel-mode Graphical User Interface Software (**login**)
- 5** Optional installation of new or updated Host Computer Operating System Libraries and Software
- 6** Optional installation of new or updated "tsWxGTUI_PyVx" Toolkit Application Libraries and Software
- 7** Manual startup of Host Computer Command Line Interface Software (**Kickoff Application Launcher -> terminal**)
- 8** Manual startup of "tsWxGTUI_PyVx" Toolkit Application Software (**python test_tsWxWidgets.py**)
 - a) Automatic startup of Character-mode Graphical User Interface Software
 - b) Automatic startup of Character-mode Top Level GUI objects
 - c) Automatic startup of Character-mode Lower Level GUI objects
 - d) Automatic shutdown of Character-mode Lower Level GUI objects
 - e) Automatic shutdown of Character-mode Top Level GUI objects
 - f) Automatic shutdown of Character-mode Graphical User Interface Software (**ctrl-c**)
- 9** Manual shutdown of Host Computer Command Line Interface Software (**exit**)
- 10** Manual shutdown of Host Computer Operating System Software (**Kickoff Application Launcher -> Leave-> Shutdown**)
- 11** Manual Power-OFF of Host Computer Hardware (Shutdown)

27.3.3.3 Cygwin Use Case

NOTES:

1) Cygwin is a free Linux-like plug-in environment, from Red Hat, for 32-bit and 64-bit versions Microsoft Windows. It includes a Command Line Interface (e.g. bash shell), GNU compiler and utility toolchain, Python programming tools and the curses Terminal Control library.

2) Without the Cygwin plug-in, a Microsoft Windows system will not be able to operate the "tsWxGTUI_PyVx" Toolkit in its Graphical User Interface (GUI) mode.

3) Use this procedure only AFTER the installation of "Cygwin". Until then, use the procedure in Microsoft Windows Use Case.

Operation of a typical computer system advances through various stages:

- 1 Manual Power-ON of Host Computer Hardware (Booting)
- 2 Automatic reset of Host Computer Hardware
- 3 Automatic startup of Host Computer Operating System Software
- 4 Automatic startup of Host Computer Pixel-mode Graphical User Interface Software (**login**)
- 5 Optional installation of new or updated Host Computer Operating System Libraries and Software
- 6 Optional installation of new or updated "tsWxGTUI_PyVx" Toolkit Application Libraries and Software
- 7 Manual startup of Host Computer Command Line Interface Software (**Start -> Programs -> Cygwin -> Cygwin Bash Shell**)
- 8 Manual startup of "tsWxGTUI_PyVx" Toolkit Application Software (**python test_tsWxWidgets.py**)
 - a) Automatic startup of Character-mode Graphical User Interface Software
 - b) Automatic startup of Character-mode Top Level GUI objects
 - c) Automatic startup of Character-mode Lower Level GUI objects
 - d) Automatic shutdown of Character-mode Lower Level GUI objects
 - e) Automatic shutdown of Character-mode Top Level GUI objects
 - f) Automatic shutdown of Character-mode Graphical User Interface Software (**ctrl-c**)
- 9 Manual shutdown of Host Computer Command Line Interface Software (**exit**)
- 10 Manual shutdown of Host Computer Pixel-mode Graphical User Interface Software (**logout**)
- 11 Manual shutdown of Host Computer Operating System Software (**shutdown**)
- 12 Manual Power-OFF of Host Computer Hardware (Shutdown)

27.3.3.4 Ubuntu Linux Use Case

Operation of a typical computer system advances through various stages:

- 1 Manual Power-ON of Host Computer Hardware (Booting)
- 2 Automatic reset of Host Computer Hardware
- 3 Automatic startup of Host Computer Operating System Software
- 4 Automatic startup of Host Computer Pixel-mode Graphical User Interface Software (**login**)
- 5 Optional installation of new or updated Host Computer Operating System Libraries and Software
- 6 Optional installation of new or updated "tsWxGTUI_PyVx" Toolkit Application Libraries and Software
- 7 Manual startup of Host Computer Command Line Interface Software (**Applications -> Accessories -> terminal**)
- 8 Manual startup of "tsWxGTUI_PyVx" Toolkit Application Software (**python3.2 test_tsWxWidgets.py**)
 - a) Automatic startup of Character-mode Graphical User Interface Software
 - b) Automatic startup of Character-mode Top Level GUI objects
 - c) Automatic startup of Character-mode Lower Level GUI objects
 - d) Automatic shutdown of Character-mode Lower Level GUI objects
 - e) Automatic shutdown of Character-mode Top Level GUI objects
 - f) Automatic shutdown of Character-mode Graphical User Interface Software (**ctrl-c**)
- 9 Manual shutdown of Host Computer Command Line Interface Software (**exit**)
- 10 Manual shutdown of Host Computer Pixel-mode Graphical User Interface Software (**System -> Logout**)
- 11 Manual shutdown of Host Computer Operating System Software (**System -> Shutdown**)
- 12 Manual Power-OFF of Host Computer Hardware (Shutdown)

Draft

28 APPENDIX B - Software Design Constraints

Since each software release shall provide the means to:

- 1** Install in developer-sandbox and/or site-package configurations; and
- 2** Use with Python 2x and/or Python 3x language generations

The design and implementation shall therefore be constrained as described in the following sections.

Draft

28.1 README

```
#-----
#"Time-stamp: <06/05/2015  6:31:31 AM rsg>
#-----

===== File: README.txt =====

+-----+-----+ TeamSTARS "tsWxGTUI_PyVx" Toolkit
| ts | Wx |      with Python 2x & Python 3x based
+-----+-----+      Command Line Interface (CLI)
| G T U I |      and "Curses"-based "wxPython"-style,
+-----+-----+      Graphical-Text User Interface (GUI)

Get that cross-platform, pixel-mode "wxPython" feeling
on character-mode 8-/16-color (xterm-family) & non-color
(vt100-family) terminals and terminal emulators.

You can find this and other files in the following sub-
directories:

<Your Working Repository>
(e.g. "Technical_Preview")
|
|   Working repository containing directories and
|   files to be packaged into downloadable "tarball"
|   and/or "zip" files via the setup shell scripts
|   at the bottom of this diagram.
|
+-- ["Documents"] (Original)
|   |
|   |   This directory contains a collection of files
|   |   which provide the Toolkit recipient with an
|   |   understanding of the purpose, goals & capabil-
|   |   ities, non-goals & limitations, terms & condi-
|   |   tions and procedures for installing, operating,
|   |   modifying and redistributing the Toolkit.
|   |
|   +-- "README.txt"
|   +-- "README1-Introduction.txt"
|   +-- "README2-Repository.txt"
|   +-- "README3-Documents.txt"
|   +-- "README4-ManPages.txt"
|   +-- "README5-Notebooks.txt"
|   +-- "README6-SourceDistributions.txt"
|   +-- "GETTING_STARTED.txt"
|
+-- ["ManPages"] (Original)
|   |
|   |   Deliverable Toolkit manual pages are a
|   |   form of online software documentation
|   |   usually found on a Unix or Unix-like
|   |   operating system.
```

```

|
|
|   Topics covered include computer programs
|   (including library and system calls),
|   formal standards and conventions, and even
|   abstract concepts.
|
|   Unlike their Unix or Unix-like counterparts,
|   a Toolkit user may NOT invoke a man page by
|   issuing the "man command". Instead, a user
|   must display a man page by issuing the
|   "less <man document file>" command.
|
|
|--- ["tsManPagesLibCLI"]
|--- ["tsManPagesLibGUI"]
|--- ["tsManPagesTestsLibCLI"]
|--- ["tsManPagesTestsLibGUI"]
|--- ["tsManPagesToolsCLI"]
|--- ["tsManPagesToolsGUI"]      (Future)
|--- ["tsManPagesToolsLibCLI"]
|--- ["tsManPagesToolsLibGUI"]  (Future)
|--- ["tsManPagesUtilitiesCLI"] (Future)
|
|--- "README4-ManPages.txt"
|
+--- ["Notebooks"] (Pre-dates Documents)
|
|   Contains a collection of commentaries that
|   express opinions or offerings of explanations
|   about events or situations that might
|   be useful to Toolkit installers, developers,
|   operators, troubleshooters and distributors.
|   The documents may be in Application-specific
|   formats (such as Adobe PDF, JPEG Bit-mapped
|   image, LibreOffice, Microsoft Office, plain
|   text).
|
|
|--- ["DeveloperNotebook"]
|
|   Contains a collection of:
|       API-References-Pixel-Mode-wxPython
|       and Developer-ReadMe-Files
|
|--- ["EngineeringNotebook"]
|
|   Contains a Toolkit Developer oriented collection of:
|       Project (purpose,
|               goals,
|               non-goals,
|               features,
|               capabilities,
|               limitations),
|       Plan (software life-cycle),
|       Requirements (purpose,
|                   goals,
|                   non-goals,

```



```

|
|
|   +-- ["Site-Packages"]
|       |
|       |   Site-packages is the location where third-
|       |   party packages are installed (i.e., those
|       |   not part of the core Python distribution).
|       |   NOTE: That with Linux, Mac OS X and Unix
|       |   operating systems one must have root priv-
|       |   ileages to write to that location.
|       |
|       +-- ["tsWxGTUI_PyVx"] (Site-Package)
|           |
|           +-- ["Documents"] (Copy)
|           |
|           +-- ["ManPages"] (Copy)
|           |
|           +-- ["Python-2x"] (Site-Package)
|               |
|               +-- ["tsWxGTUI_Py2x"]
|               |
|           +-- ["Python-3x"] (Site-Package,
|               |   Ported from Python-2x)
|               +-- ["tsWxGTUI_Py3x"]
|
+-- "MANIFEST.in"
|
|   Deliverable File inclusion criteria list.
|
+-- "MANIFEST_template.in"
|
|   Deliverable Generic file inclusion criteria list
|   template for any Python version-specific TeamSTARS
|   "tsWxGTUI_PyVx" Toolkit.
|
+-- "MANIFEST_TREE.html"
|
|   Non-Deliverable Diagram (Multi-Level Org Chart)
|   depicting the hierarchical relationship between files
|   in the release, in Hypertext Markup Language format.
|
|   Diagram created via Command "./MANIFEST_TREE.sh".
|
+-- "MANIFEST_TREE.sh"
|
|   Deliverable POSIX-style Command Line Interface shell
|   script to generate diagrams depicting the hierarchical
|   relationship between files in the release
|   ("MANIFEST_TREE.html" and "MANIFEST_TREE.txt").
|
+-- "MANIFEST_TREE.txt"
|
|   Non-Deliverable Diagram (Multi-Level Org Chart)
|   depicting the hierarchical relationship between
|   files in the release, in Plain Text format.

```

```
|
|   Diagram created via Command "./MANIFEST_TREE.sh".
|
+-- "setup_Technical_Preview_tar_file.sh"
|
|   Deliverable POSIX-style Command Line Interface shell
|   script to generate downloadable "tarball" file.
|
+-- "setup_Technical_Preview_zip_file.sh"
|
|   Deliverable POSIX-style Command Line Interface shell
|   script to generate downloadable "zip" file.
|
+-- "README.txt"

===== WELCOME =====
```

The "tsWxGTUI" Toolkit provides a collection of building-block components, tools, utilities, and tests for creating, enhancing, troubleshooting, maintaining and supporting application programs that are suitable for embedded systems.

1. Application Programs

Automation, communication, control, diagnostic, instrumentation and simulation application programs typically require an "operator-friendly" Command Line Interface (CLI) or a Graphical-style User Interface (GUI) that can be controlled locally or remotely.

2. Embedded Systems

Mission-critical systems for commercial, industrial, medical and military applications are typically customized and optimized for a specific use. Unlike their general-purpose desktop, laptop and workstation counterparts, embedded systems typically have limited, application-specific processing, memory, communication, input/output and file storage resources. Some may have character-mode hardware only suitable for their operating system's command line console.

3. What should you do to get started?

Browse through the following information to get an overview of the distribution and its contents:

- 3.1 "./tsWxGTUI_PyVx/Documents/README1-Introduction.txt"
- 3.2 "./tsWxGTUI_PyVx/Documents/README2-Repository.txt"
- 3.3 "./tsWxGTUI_PyVx/Documents/README3-Documents.txt"
- 3.4 "./tsWxGTUI_PyVx/Documents/README4-ManPages.txt"
- 3.5 "./tsWxGTUI_PyVx/Documents/README5-Notebooks.txt"
- 3.6 "./tsWxGTUI_PyVx/Documents/README6-SourceDistribution"
- 3.7 "./tsWxGTUI_PyVx/Documents/GETTING_STARTED.txt"
- 3.8 "./tsWxGTUI_PyVx/Documents/DEMO.txt"

4. Experience the features, look and feel of the toolkit by running through the scenarios presented in DEMO.txt file and browsing through the Python source code for the associated application programs and building blocks.

--

Richard S. Gordon
SoftwareGadgetry@comcast.net

===== End-Of-File =====

Draft

28.2 README1-Introduction.txt

```
#-----
#"Time-stamp: <06/05/2015  5:09:51 AM rsg>
#-----

===== File: README1-Introduction.txt =====

+-----+-----+ TeamSTARS "tsWxGTUI_PyVx" Toolkit
| ts | Wx |      with Python 2x & Python 3x based
+-----+-----+      Command Line Interface (CLI)
| G T U I |      and "Curses"-based "wxPython"-style,
+-----+-----+      Graphical-Text User Interface (GUI)

Get that cross-platform, pixel-mode "wxPython" feeling
on character-mode 8-/16-color (xterm-family) & non-color
(vt100-family) terminals and terminal emulators.

You can find this and other plain-text files in the
Toolkit subdirectory named:

    <Your Working Repository>
    (e.g. "Technical_Preview")
    |
    |   Working repository containing directories and
    |   files to be packaged into downloadable "tarball"
    |   and/or "zip" files via the setup shell scripts
    |   at the bottom of this diagram.
    |
    +-- ["Documents"]

===== TABLE OF CONTENTS =====

1. What is it?

    1.1 In a Nutshell
    1.2 The Gory Details

2. How is it implemented?

    2.1 Python Programming Language
    2.2 Python-based Command Line Interface (CLI)
    2.3 "wxPython"-style Graphical-Text User Interface (GUI)

3. What are the System Requirements?

    3.1 Hardware
    3.2 Software

4. The Latest Versions

    4.1 "tsWxGTUI_Py2x-0.0.0" for Python 2.4.1 - 2.7.9
    4.2 "tsWxGTUI_Py3x-0.0.0" for Python 3.0.0 - 3.4.3
```

5. Deliverables

- 5.1 Documentation
- 5.2 Source Code

6. Installation

7. Licensing

8. Contacts

9. Acknowledgments

===== WHAT IS IT? =====

1. What is it?

The "tsWxGTUI" Toolkit provides a collection of building-block components, tools, utilities, and tests.

The Toolkit facilitates the effort of software developers creating enhanceing, troubleshooting, maintaining and supporting application programs with character-mode Command Line and Graphical-style User Interfaces that are suitable for the local and remote monitoring and control of computer systems that are embedded in mission-critical equipment.

1.1 In a Nutshell

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

1.1.1 User-friendly Interfaces:

a. Command Line Interface (CLI)

Output to the user of a chronological sequence of lines of text via a scrolling computer terminal display with input from the user via a computer terminal keyboard.

b. Graphical-style User Interface (GUI)

Output to the user of character strings to application-specified column and row (line) fields of a computer terminal display with input from the user via a computer terminal keyboard and pointing device (such as mouse, trackball, touchpad or touchscreen).

1.1.2 General-purpose, portability, maintainability, re-usability, scalability, deployability:

a. Toolkit Applications

Software development and installation toolkit for automation, communication, control, diagnostic, instrumentation and simulation applications.

b. Usage Applications

Computerized mainframe, workstation, desktop, laptop, tablet and embedded systems with 32-bit/64-bit processors from various manufacturers and popular operating systems including GNU/Linux, Mac OS X, Microsoft Windows and Unix.

1.2 The Gory Details

Software development systems typically have sufficient and upgradable resources including 32-bit/64-bit processors, random access memory, non-volatile memory (such as electro-mechanical hard drives and electronic flash memory), network interface devices and operator control consoles with keyboard, mouse and large high-cost pixel-mode/graphics-mode displays that can also operate in character-mode/text-mode.

Embedded systems typically are optimized for specific commercial, industrial, medical and military applications. The more capable systems have 32-bit/64-bit processors, random access memory, non-volatile memory, network interface devices and operator control consoles with keyboard, mouse and small low-cost character-mode/text-mode displays.

The Toolkit provides utilities, tools and a library of building blocks for you to create application programs that can raise the productivity and reduce the applied time of software developers and maintainers by its use of the high-level Python programming language.

It can also raise the productivity of system administrators, software developers, equipment operators and field service personnel by providing a suitable user-friendly interface:

1.2.1 A Command Line Interface (CLI)

Please see encyclopedia-based information at:

[http://en.wikipedia.org/wiki/
Command-line_interface](http://en.wikipedia.org/wiki/Command-line_interface)

1.2.2 A Text-based User Interface (TUI)

The Text-based User Interface (TUI) emulates a subset of the Application Programming Interface (API) for wxPython, a popular cross-platform Graphical User Interface (GUI). The baseline API for the TUI is that of wxPython 2.8.9.2. Once development has been completed, the API can be updated to track the latest wxPython version.

Please see encyclopedia-based information at:

http://en.wikipedia.org/wiki/Text-based_user_interface

http://en.wikipedia.org/wiki/Graphical_user_interface

It can raise overall productivity through its ability to monitor remote computer systems and associated equipment from the convenience of a local centralized computer system.

The network communication speed of character data will be faster than that of pixel data because:

1. There would be more pixels than character cells involved in programmatic change. For example, on a 307,200 (640x480) pixel display, there are only 3,200 (80 cols x 40 rows) character cells when using a 96 (8x12) pixel font.
2. There would only be 1 byte involved in the following programmatic character cell changes:
 - a) Which of the 256 text characters, and pre-assigned 96 (8x12) pixel combinations, to apply to individual character cells.
 - b) Which of the 256 (16x16) color pairs, and pre-assigned 256x256x256 red-green-blue foreground color intensity levels, and 256x256x256 red-green-blue background color intensity levels to apply to individual strings of one or more character cells.
 - c) Which of the 8 text character attribute combinations (alternate character set, blink mode, bold mode, dim mode, normal attribute, reverse background and foreground colors, standout mode and underline mode) to apply to individual strings of one or more character cells.

3. Alternative Solutions

- a) The Java programming language user community has access to a small footprint console interface.

See "<http://www.codeproject.com/Articles/328417/Java-Console-apps-made-easy>"

In the absence of any statement about character-mode, it is presumed that the Java console operates only in pixel-mode.

- b) The wxWidgets 3.0 C++ programming language user community has access to a set of console interface classes that were not in wxWidgets 2.8 or 2.9.

See "http://docs.wxwidgets.org/trunk/classwx_app_console.html"

In the absence of any statement about character-mode, it is presumed that the set of "wxWidgets 3.0" console interface classes operate only in standard pixel-mode.

- c) "Urwid"

Excerpt from "<http://urwid.org/manual/overview.html>"

"Urwid is a console user interface library for Python. Urwid offers an alternative to using Python's curses module directly and handles many of the difficult and tedious tasks for you.

../_images/introduction.png

Each Urwid component is loosely coupled and designed to be extended by the user.

Display modules are responsible for accepting user input and converting escape sequences to lists of keystrokes and mouse events. They also draw the screen contents and convert attributes used in the canvases rendered to the actual colors that appear on screen.

The included widgets are simple building blocks and examples that try not to impose a particular style of interface. It may be helpful to think of Urwid as a console widget construction set rather than a finished UI library like GTK or Qt. The Widget base class describes the widget interface and widget layout describes how widgets are nested and arranged on the screen.

Text is the bulk of what will be displayed in

any console user interface. Urwid supports a number of text encodings and Urwid comes with a configurable text layout that handles the most of the common alignment and wrapping modes. If you need more flexibility you can also write your own text layout classes.

Urwid supports a range of common display attributes, including 256-color foreground and background settings, bold, underline and standout settings for displaying text. Not all of these are supported by all terminals, so Urwid helps you write applications that support different color modes depending on what the user's terminal supports and what they choose to enable."

<additional info not reproduced>

d) "npyscreen"

Excerpts from "<https://code.google.com/p/npyscreen/>"

"Npyscreen is a python widget library and application framework for programming terminal or console applications. It is built on top of ncurses, which is part of the standard library."

<additional info not reproduced>

"Strengths

This framework should be powerful enough to create everything from quick, simple programs to complex, multi-screen applications. It is designed to make doing the simple tasks very quick and to take much of the pain out of writing larger applications.

There is a very wide variety of default widgets - everything from simple text fields to more complex tree and grid views.

The framework is easy to extend. That said, if you have a requirement for a widget that is not currently included you can try emailing me and I'll see whether I have time to help - no promises!"

===== HOW IS IT IMPLEMENTED? =====

2. How is it implemented?

2.1 Python Programming Language

From <https://docs.python.org/3/faq/general.html>

From <https://docs.python.org/2/faq/general.html>

"What is Python:

Python is an interpreted, interactive, object-oriented programming language. It incorporates modules, exceptions, dynamic typing, very high level dynamic data types, and classes. Python combines remarkable power with very clear syntax. It has interfaces to many system calls and libraries, as well as to various window systems, and is extensible in C or C++. It is also usable as an extension language for applications that need a programmable interface. Finally, Python is portable: it runs on many Unix variants, on the Mac, and on PCs ..."

Python 3x and 2x for 32-bit and 64-bit processors:

"... on Windows 2000 and later."

Python 2x for 16-bit and 32-bit processors:

"... under MS-DOS, Windows, Windows NT, and OS/2."

Python 1x for 16-bit and 32-bit processors

System requirements are no longer available.

From <http://ftp.python.org/download/releases/1.6.1/>

"Python 1.6 was the last of the versions developed at [the Corporation for National Research Initiatives] CNRI and the only version issued by CNRI with an open source license. Following the release of Python 1.6, and after Guido van Rossum left CNRI to work with commercial software developers, it became clear that the ability to use Python with software available under the GNU General Public License (GPL) was very desirable. CNRI and the Free Software Foundation (FSF) interacted to develop enabling wording changes to the Python license. Python 1.6.1 is essentially the same as Python 1.6, with a few minor bug fixes, and with a GPL-compatible license."

2.1.1 The high-level Python programming language is used to implement the TeamSTARS "tsWxGTUI_PyVx" Toolkit.

The TeamSTARS "tsWxGTUI_PyVx" Toolkit's building block and tool components import and use run time library components from the Python Global Module Index and from Python user-installed site-packages.

Python is a popular, field proven, portable, cross-platform programming language.

Please see encyclopedia-based Python information at:

[http://en.wikipedia.org/wiki/Python_\(programming_language\)](http://en.wikipedia.org/wiki/Python_(programming_language))

2.2 Python-based Command Line Interface (CLI)

2.2.1 The TeamSTARS "tsWxGTUI_PyVx" Toolkit's Command Line Interface building block components import and use the Keyword-Value Pair and Positional argument parsers:

Please see Python Global Module Index-based information at:

<http://docs.python.org/3/library/argparse.html>

<http://docs.python.org/2/library/optparse.html>

<http://docs.python.org/2/library/getopt.html>

Availability:

a) "argparse"

Introduced in Python 2.7 and Python 3.2.

b) "optparse"

Introduced in Python 2.3 (deprecated in Python 2.7) and Python 3.0 (deprecated in Python 3.2).

c) "getopt"

Available in Python 1.6, 2.0 and Python 3.0.

2.3 "wxPython"-style Graphical-Text User Interface (GUI)

1.3.1 The TeamSTARS "tsWxGTUI_PyVx" Toolkit's Text-based User Interface components import and use the Terminal handler ("curses") for character-cell displays available in Python's Global Module Index.

It uses "curses" to emulate a subset of the Application Programming Interface (API) of "wxPython", a wrapper to the popular "wxWidgets" pixel-mode, cross-platform Graphical User Interface Toolkit which is implemented in the C++ programming language.

The "wxPython" emulation retains the pixel-mode parameters of the "wxPython" API and mimics the look and feel of Microsoft "Windows XP" Displays which are similar to those of Linux "GTK+" Displays:

- a) GUI container features such as frames, dialogs and panels and buttons to close, iconize and maximize/restore the container.
- b) GUI control features such as buttons, checkboxes, radio buttons, scroll bars, scroll lists and status bars.
- c) GUI layout features such as box sizer and grid sizer.
- d) GUI operator notification features such as a scrolling log of date and time stamped event notification messages.
- e) GUI desktop features such as task bar buttons to control GUI container focus.

Please see encyclopedia-based information at:

<http://en.wikipedia.org/wiki/WxPython>

<http://en.wikipedia.org/wiki/WxWidgets>

<http://en.wikipedia.org/wiki/GTK%2B>

Please see Python Global Module Index-based information at:

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

<http://docs.python.org/3/library/curses.html>

===== WHAT ARE THE SYSTEM REQUIREMENTS? =====

3. What are the System Requirements?

The design of the TeamSTARS "tsWxGTUI_PyVx" Toolkit supports a wide range of possible system configurations.

Cross-platform Python virtual machine technology is often available for Intel and non-Intel 32-bit and 64-bit processor based systems running proprietary and non-proprietary operating systems.

For example, Toolkit development and testing has involved the following system configurations:

3.1 Hardware

The TeamSTARS "tsWxGTUI_PyVx" Toolkit development and testing platforms have involved three classes of equipment represented by the following:

a. Minimal Usability and Performance

1998-model year, 366 MHz Intel Pentium II-based Dell Inspiron 7000 laptop with 384 MB RAM, 640x480/1024x768 pixel resolution display and dual PCMCIA Card expansion capabilities for network and peripheral device interfaces (with marginal resources and performance) sufficient enough to serve as the low-end, single user baseline development and operator platform.

Its interchangeable 32 GB (4200 RPM) ATA hard drives were used to run either:

- * Microsoft Windows Desktop with Cygwin, the free GNU/Linux-like Plug-in from Red Hat (XP Pro); or
- * Ubuntu GNU/Linux Desktop (12.04)

The platform's limited memory and available PCMCIA network adapters were incompatible with later versions of Windows or with other Linux distributions. (Windows XP recognized Xircom Ethernet and 3Com Wireless adapters; Ubuntu Linux 12.04 recognized only Linksys Wireless adapter.

b. Moderate Usability and Performance

2007-model year, 2.33 GHz Intel Core 2 Duo processor-based Apple 17" MacBook Pro laptop with 4 GB RAM, 1920x1200 pixel resolution display and sufficient performance, resources and expansion capabilities to serve as the mid-range, single user baseline development and operator platform.

Its 160 GB (5400 RPM) SATA 1.5 Gb/s internal hard drive was used to run Apple's Mac OS X 10.7 and the hypervisor virtualization applications (Parallels Desktop 8 and VMware Fusion 5) that supported various guest operating systems.

Its 1.5 TB (7200 RPM) SATA 3 Gb/s external hard drive was used to store and concurrently run an assortment of up to two configured versions selected (for normal use rather than for stress-testing) from the following guest operating systems:

- * Fedora GNU/Linux Desktop (17)
- * Ubuntu GNU/Linux Desktop (12.04)
- * Microsoft Windows Desktop with Cygwin, the free GNU/Linux-like Plug-in from Red Hat (XP Pro, 7 Pro, 8 Pro and 8.1 Pro)
- * OpenIndiana (OpenSolaris 11-based) Unix Desktop (151a6)

NOTES:

- 1) A Seagate ST31500341AS 1.5TB SATA 7200 RPM was used as the external hard drive. The wear and tear from using a consumer product (over 8-10 hours a day, 7 days each week for 7 years) for the Guest Operating System swapfile and data storage ultimately wore it out (unrecoverable disk head crash).
- 2) Subsequent research indicated that a more appropriate hard drive would have been be an Hitachi Ultrastar 7K3000 HUA723020ALA641 2TB 7200RPM 64MB Cache SATA 6.0Gb/s 3.5" Enterprise Hard Drive -OEM.

c. High Usability and Performance

2013-model year, 3.5 GHz Intel Quad Core i7 processor-based Apple 27" iMac desktop with 16 GB RAM, 2560x1440 pixel resolution display and sufficient performance, resources and expansion capabilities to serve as the high-end, multi-user baseline development and operator platform.

Its 3 TB (7200 RPM) SATA 6 Gb/s internal hard drive had 128 GB Solid State Flash memory and was used to run Apple's Mac OS X 10.9-10.10 and the hypervisor virtualization applications (Parallels Desktop 9-10 and VMware Fusion 5 and 7) that supported various guest operating systems.

Its 3 TB (7200 RPM) SATA 6 Gb/s internal hard drive was normally used to store and concurrently run an assortment of up to four configured versions selected (for normal use rather than for stress-testing) from the following guest operating systems:

- * CentOS GNU/Linux Desktop (7.0)
- * Debian GNU/Linux Desktop (8) and Server (Turnkey 13.0) Configured with Apache, Bugzilla and MySQL.
- * Fedora GNU/Linux Desktop (20-22)

- * OpenSuSE GNU/Linux Desktop (13.1)
- * Scientific GNU/Linux Desktop (6.5 & 7.0)
- * Ubuntu GNU/Linux Desktop (12.04 & 14.04)
- * Microsoft Windows Desktop with Cygwin, the free GNU/Linux-like Plug-in from Red Hat (XP Pro, 7 Pro, 8 Pro, 8.1 Pro and 10 Technical Preview)
- * PC-BSD (FreeBSD-based) Unix Desktop (9.2 & 10.0)
- * OpenIndiana (OpenSolaris 11-based) Unix Desktop (151a8)

3.2 Software

The TeamSTARS "tsWxGTUI_PyVx" Toolkit development and testing platforms have involved an assortment of single and multi-user, multi-process and multi-threaded POSIX-compatible operating system releases.

The following operating system information consists of annotated excerpts from Wikipedia, the free encyclopedia, in order to preserve a snapshot of relevant content that might otherwise be subject to change and lose relevance:

a. GNU/Linux

GNU is a Unix-like computer operating system developed by the GNU Project, ultimately aiming to be a "complete Unix-compatible software system" composed wholly of free software.

Linux is a Unix-like and mostly POSIX-compliant computer operating system assembled under the model of free and open-source software development and distribution. The defining component of Linux is the Linux kernel, an operating system kernel first released on 5 October 1991 by Linus Torvalds. The Free Software Foundation uses the name GNU/Linux to describe the operating system, which has led to some controversy.

Many computer users run a modified version of the GNU system every day, without realizing it. Through a peculiar turn of events, the version of GNU which is widely used today is often called "Linux", and many of its users are not aware that it is basically the GNU system, developed by the GNU Project.

Please see encyclopedia-based information at:

<http://en.wikipedia.org/wiki/GNU>

<http://en.wikipedia.org/wiki/Linux>
http://en.wikipedia.org/wiki/Linux_kernel

- * CentOS, a distribution derived from the same sources used by Red Hat, maintained by a dedicated volunteer community of developers with both 100% Red Hat-compatible versions and an upgraded version that is not always 100% upstream compatible. (7.0)
- * Debian, a non-commercial distribution and one of the earliest, maintained by a volunteer developer community with a strong commitment to free software principles and democratic project management
- * Fedora, a community distribution sponsored by American company Red Hat. (17-22)
- * OpenSUSE, a community distribution mainly sponsored by German company SuSE. (13.1)
- * SuSE Linux Enterprise, derived from openSUSE, is maintained and commercially supported by SuSE.
- * Red Hat Enterprise Linux, a derivative of Fedora, maintained and commercially supported by Red Hat
- * Scientific, a Linux distribution produced by Fermi National Accelerator Laboratory. It is a free and open source operating system based on Red Hat Enterprise Linux and aims to be "as close to the commercial enterprise distribution as we can get it.". (6.5 and 7.0)

This product is derived from the free and open source software made available by Red Hat, Inc., but is not produced, maintained or supported by Red Hat. Specifically, this product is built from the source code for Red Hat Enterprise Linux versions, under the terms and conditions of Red Hat Enterprise Linux's EULA and the GNU General Public License.

- * Ubuntu, a popular desktop and server distribution derived from Debian, maintained by British company Canonical Ltd. (12.04 LTS and 14.04 LTS)
- b. Microsoft Windows, a metafamily of graphical operating systems developed, marketed, and sold by Microsoft. It consists of several families of operating systems, each of which cater to a certain sector of the computing industry. Active Windows families include Windows NT, Windows Embedded and Windows Phone; these may encompass subfamilies, e.g. Windows Embedded Compact (Windows CE) or Windows Server. Defunct Windows families include Windows 9x and

Windows Mobile.

Microsoft Windows will only require "Cygwin", the free Linux-like plug-in from Red Hat for users of the "wxPython"-style, "Curses"-based Graphical-Text User Interface. (Home or Professional editions of Windows XP, 7, 8, 8.1, 10 Technical Preview)

Microsoft introduced an operating environment named Windows on November 20, 1985 as a graphical operating system shell for MS-DOS in response to the growing interest in graphical user interfaces (GUIs). Microsoft Windows came to dominate the world's personal computer market with over 90% market share, overtaking Mac OS, which had been introduced in 1984. However, it is outsold by Android on smartphones and tablets.

- c. OS X (formerly known as Mac OS X), a series of Unix-based graphical interface operating systems developed and marketed by Apple Inc. It is designed to run on Mac computers. (10.4 "Tiger"-10.10 "Yosemite")

NOTES: From http://en.wikipedia.org/wiki/OS_X

"The first releases of Mac OS X from 1999 to 2006 can run only on the PowerPC based Macs of the period. After Apple announced it would shift to using Intel x86 CPUs from 2006 onwards, Tiger and Leopard were released in versions for Intel and PowerPC processors. Snow Leopard is the first version released only for Intel Macs. Since the release of Mac OS X 10.7 "Lion", OS X has dropped support for 32-bit Intel processors as well. It now runs exclusively on 64-bit Intel CPUs."

Mac OS X Version	PowerPC Platform (NOT tested with Toolkit)
-----	-----
10.0: "Cheetah"	(32-bit PowerPC)
10.1: "Puma"	(32-bit PowerPC)
10.2: "Jaguar"	(32-bit PowerPC)
10.3: "Panther"	(32-bit PowerPC)
10.4: "Tiger"	(32-bit PowerPC and Intel)
10.5: "Leopard"	(32-bit PowerPC and Intel)

Mac OS X Version	Intel Platforms (tested with Toolkit)
-----	-----
10.4: "Tiger"	(32-bit PowerPC and Intel)
10.5: "Leopard"	(32-bit PowerPC and Intel)
10.6: "Snow Leopard"	(32-bit Intel)
10.7: "Lion"	(32-bit Intel)
10.8: "Mountain Lion"	(64-bit Intel)

10.9: "Mavericks" (64-bit Intel)
10.10: "Yosemite" (64-bit Intel)

Versions 10.5 "Leopard" running on Intel processors, 10.6 "Snow Leopard", 10.7 "Lion", 10.8 "Mountain Lion", 10.9 "Mavericks", and 10.10 "Yosemite" have obtained UNIX 03 certification.

iOS, which runs on the iPhone, iPod Touch, iPad, and the 2nd and 3rd generation Apple TV, shares the Darwin core and many frameworks with OS X.

- d. Unix, a multitasking, multiuser computer operating system that exists in many variants. The original Unix was developed at AT&T's Bell Labs research center by Ken Thompson, Dennis Ritchie, and others. From the power user's or programmer's perspective, Unix systems are characterized by a modular design that is sometimes called the "Unix philosophy," meaning the OS provides a set of simple tools that each perform a limited, well-defined function, with a unified filesystem as the main means of communication and a shell scripting and command language to combine the tools to perform complex workflows.
- * FreeBSD, a free Unix-like operating system descended from Research Unix via the Berkeley Software Distribution (BSD). Although for legal reasons FreeBSD cannot use the Unix trademark, it is a direct descendant of BSD, which was historically also called "BSD Unix" or "Berkeley Unix." (10.0)
 - * OpenIndiana, a free and open-source, Unix operating system derived from OpenSolaris. Developers forked OpenSolaris after Oracle Corporation discontinued it, in order to continue development and distribution of the source code. The OpenIndiana project is stewarded by the illumos Foundation, which also stewards the illumos operating system. OpenIndiana's developers strive to make it "the defacto OpenSolaris distribution installed on production servers where security and bug fixes are required free of charge". (151a8)
 - * OpenSolaris, a descendant of the UNIX System V Release 4 (SVR4) code base developed by Sun and AT&T in the late 1980s. It is the only version of the System V variant of UNIX available as open source.
 - * PC-BSD, or PCBSD, a Unix-like, desktop-oriented operating system built upon the most recent releases of FreeBSD. It aims to be easy to install by using a graphical installation program, and

easy and ready-to-use immediately by providing KDE SC, LXDE, Xfce, and MATE as the graphical user interface. (10.0)

===== THE LATEST VERSIONS =====

4. The Latest Versions

The latest TeamSTARS "tsWxGTUI_PyVx" Toolkit version is a pre-alpha stage, pre-production release identified as:

4.1 "tsWxGTUI_Py2x-0.0.0" for Python 2.4.1 - 2.7.9

4.2 "tsWxGTUI_Py3x-0.0.0" for Python 3.0.0 - 3.4.3

===== DELIVERABLES =====

5. Deliverables

Deliverables for the TeamSTARS "tsWxGTUI_PyVx" Toolkit include the following:

5.1 Documentation

5.1.1 Documentation in Plain Text Format

The TeamSTARS "tsWxGTUI_PyVx" Toolkit documentation is included, in plain text format, in the directory named:

./tsWxGTUI_PyVx/Documents

5.1.2 Documentation in HTML Format

Since the TeamSTARS "tsWxGTUI_PyVx" Toolkit emulates a character-mode compatible subset of the wxPython and wxWidgets pixel-mode GUI Application Programming Interface (API), the documentation includes archive copies of pixel-mode API in its Hypertext Markup Language format.

The archive copies are provided because the original On-Line versions are no longer available on the wxWidgets and wxPython web sites.

5.2 Source Code

Excerpt From Wikipedia, the free encyclopedia:

"Python is an open source programming language that was made to both look good and be easy to read. It was created by a programmer named Guido van Rossum in 1991. The language is named after the television show Monty Python's Flying Circus and many examples and tutorials include jokes from the show.

Python is an interpreted language. An interpreted language allows the programmer to give the source code to the computer and the computer runs the code right away. This means if the programmer needs to change the code they can quickly see the results. This makes Python a good programming language for beginners and for making programs rapidly because you do not have to compile the code to make it run, and compiling takes a lot of time. But because the computer has to figure out what the code does every time the code runs, Python is a very slow language. Sometimes, it can be 200 times slower than C [programming language].

Python is also a high-level programming language. A high-level language has advanced features which let the programmer to tell the computer what to do without having to worry about how the computer is going to do that as much as low-level programming languages. This makes writing programs easier and faster. Some of the rules of how you write code in Python are taken from C, and Python can run some C code."

Since Python is not a conventional compiled language, its language syntax does not include compiler directives to conditionally compile language version specific features. Consequently there must be separate source code directories and files for Python 2x and Python 3x.

While many language features are common to Python 2x and Python 3x:

- a) obsolescent ones may be deprecated, available for a limited time (like the print statements and old-style classes introduced in Python 1x that were retained only in Python 2x) but not recommended;
- b) obsolete ones ultimately disappear (like the print statements and old-style classes syntax no longer in Python 3.x); and
- c) enhanced ones may be introduced (like the print function syntax introduced in Python 2x and the new except statement syntax introduced in Python 3.x).

That being said, cross-platform regression testing on various Linux, Mac OS X, Microsoft Windows (with and without the free, Linux-like Cygwin plug-in) and Unix has established the set of Python 2x and Python 3x versions which fully support the Toolkit's Command Line Interface and Graphical User Interface.

Release of the source code enables Toolkit users to customize the source code so as to support older or newer Python versions and platforms.

===== INSTALLATION =====

6. Installation

Please see the file named:

./Documents/INSTALL.txt

===== LICENSING =====

7. Licensing

Please see the file named:

./Documents/LICENSE.txt

===== CONTACTS =====

8. Contacts

Technical Support Requests should be directed via email sent to:

SoftwareGadgetry@comcast.net

===== ACKNOWLEDGMENTS =====

9. Acknowledgments

Please see files with the following names:

./Documents/AUTHORS.txt

./Documents/COPYRIGHT.txt

./Documents/CREDITS.txt

./Documents/THANKS.txt

===== End-Of-File =====

28.3 README2-Repository.txt

```
#-----
#"Time-stamp: <06/03/2015  6:46:09 AM rsg>
#-----

===== File: README2-Repository.txt =====

+-----+-----+ TeamSTARS "tsWxGTUI_PyVx" Toolkit
| ts | Wx |      with Python 2x & Python 3x based
+-----+-----+      Command Line Interface (CLI)
| G T U I |      and "Curses"-based "wxPython"-style,
+-----+-----+      Graphical-Text User Interface (GUI)

Get that cross-platform, pixel-mode "wxPython" feeling
on character-mode 8-/16-color (xterm-family) & non-color
(vt100-family) terminals and terminal emulators.

You can find this and other files in the following sub-
directories:

<Your Working Repository>
(e.g. "Technical_Preview")
|
|   Working repository containing directories and
|   files to be packaged into downloadable "tarball"
|   and/or "zip" files via the setup shell scripts
|   at the bottom of this diagram.
|
+-- ["Documents"]
|
|   |   This directory contains a collection of files
|   |   which provide the Toolkit recipient with an
|   |   understanding of the purpose, goals & capabil-
|   |   ities, non-goals & limitations, terms & condi-
|   |   tions and procedures for installing, operating,
|   |   modifying and redistributing the Toolkit.
|   |
|   +-- "README.txt"
|   +-- "README1-Introduction.txt"
|   +-- "README2-Repository.txt"
|   +-- "README3-Documents.txt"
|   +-- "README4-ManPages.txt"
|   +-- "README5-Notebooks.txt"
|   +-- "README6-SourceDistributions.txt"
|
+-- ["ManPages"]
|
|   |   Deliverable Toolkit manual pages are a
|   |   form of online software documentation
|   |   usually found on a Unix or Unix-like
|   |   operating system.
|   |
```

```

|   |   Topics covered include computer programs
|   |   (including library and system calls),
|   |   formal standards and conventions, and even
|   |   abstract concepts.
|   |
|   |   Unlike their Unix or Unix-like counterparts,
|   |   a Toolkit user may NOT invoke a man page by
|   |   issuing the "man command". Instead, a user
|   |   must display a man page by issuing the
|   |   "less <man document file>" command.
|   |
|   +-- ["tsManPagesLibCLI"]
|   +-- ["tsManPagesLibGUI"]
|   +-- ["tsManPagesTestsLibCLI"]
|   +-- ["tsManPagesTestsLibGUI"]
|   +-- ["tsManPagesToolsCLI"]
|   +-- ["tsManPagesToolsGUI"]           (Future)
|   +-- ["tsManPagesToolsLibCLI"]
|   +-- ["tsManPagesToolsLibGUI"]       (Future)
|   +-- ["tsManPagesUtilitiesCLI"]      (Future)
|   |
|   +-- "README4-ManPages.txt"
|
+-- ["Notebooks"]                       (Pre-dates Documents)
|
|   |   Contains a collection of commentaries that
|   |   express opinions or offerings of explanations
|   |   about events or situations that might
|   |   be useful to Toolkit installers, developers,
|   |   operators, troubleshooters and distributors.
|   |   The documents may be in Application-specific
|   |   formats (such as Adobe PDF, JPEG Bit-mapped
|   |   image, LibreOffice, Microsoft Office, plain
|   |   text).
|   |
|   +-- ["DeveloperNotebook"]
|   |
|   |   |   Contains a collection of:
|   |   |   API-References-Pixel-Mode-wxPython
|   |   |   and Developer-ReadMe-Files
|   |
|   +-- ["EngineeringNotebook"]
|   |
|   |   |   Contains a Toolkit Developer oriented collection of:
|   |   |   Project (purpose,
|   |   |   |   goals,
|   |   |   |   non-goals,
|   |   |   |   features,
|   |   |   |   capabilities,
|   |   |   |   limitations),
|   |   |   Plan (software life-cycle),
|   |   |   Requirements (purpose,
|   |   |   |   goals,
|   |   |   |   non-goals,
|   |   |   |   features,

```



```

|         |         |         | usually found on a Unix or Unix-like oper-
|         |         |         | ating system.
|         |         |         |
|         |         |         | Topics covered include computer programs
|         |         |         | (including library and system calls),
|         |         |         | formal standards and conventions, and even
|         |         |         | abstract concepts.
|         |         |         |
|         |         |         | A user may NOT invoke a man page by issu-
|         |         |         | ing the man command. Instead, a user may
|         |         |         | display a man page by issuing the
|         |         |         | less <man document file> command.
|         |         |         |
|         |         |         | +--- ["tsManPagesLibCLI"]
|         |         |         | +--- ["tsManPagesLibGUI"]
|         |         |         | +--- ["tsManPagesTestsLibCLI"]
|         |         |         | +--- ["tsManPagesTestsLibGUI"]
|         |         |         | +--- ["tsManPagesToolsCLI"]
|         |         |         | +--- ["tsManPagesToolsGUI"] (Future)
|         |         |         | +--- ["tsManPagesToolsLibCLI"]
|         |         |         | +--- ["tsManPagesToolsLibGUI"] (Future)
|         |         |         | +--- ["tsManPagesUtilitiesCLI"] (Future)
|         |         |         |
|         |         |         | +--- ["Python-2x"]
|         |         |         | |
|         |         |         | | +--- ["tsWxGTUI_Py2x"]
|         |         |         | |
|         |         |         | +--- ["Python-3x"] (Ported from Python-2x)
|         |         |         | |
|         |         |         | | +--- ["tsWxGTUI_Py3x"]
|         |         |         |
| +--- ["Site-Packages"]
| |
| | Site-packages is the location where third-
| | party packages are installed (i.e., those
| | not part of the core Python distribution).
| | NOTE: That with Linux, Mac OS X and Unix
| | operating systems one must have root priv-
| | ilidges to write to that location.
| |
| +--- ["tsWxGTUI_PyVx"]
| |
| | +--- ["Documents"]
| |
| | This directory contains a collection of files
| | which provide the Toolkit recipient with an
| | understanding of the purpose, goals & capabil-
| | ities, non-goals & limitations, terms & condi-
| | tions and procedures for installing, operating,
| | modifying and redistributing the Toolkit.
| |
| +--- ["ManPages"]
| |
| | Deliverable Toolkit manual pages are a
| | form of online software documentation

```

```

|                                     | usually found on a Unix or Unix-like oper-
|                                     | ating system.
|                                     |
|                                     | Topics covered include computer programs
|                                     | (including library and system calls),
|                                     | formal standards and conventions, and even
|                                     | abstract concepts.
|                                     |
|                                     | A user may NOT invoke a man page by issu-
|                                     | ing the man command. Instead, a user may
|                                     | display a man page by issuing the
|                                     | less <man document file> command.
|                                     |
|                                     |
|                                     | +-- ["tsManPagesLibCLI"]
|                                     | +-- ["tsManPagesLibGUI"]
|                                     | +-- ["tsManPagesTestsLibCLI"]
|                                     | +-- ["tsManPagesTestsLibGUI"]
|                                     | +-- ["tsManPagesToolsCLI"]
|                                     | +-- ["tsManPagesToolsGUI"]          (Future)
|                                     | +-- ["tsManPagesToolsLibCLI"]
|                                     | +-- ["tsManPagesToolsLibGUI"]    (Future)
|                                     | +-- ["tsManPagesUtilitiesCLI"] (Future)
|                                     |
|                                     | +-- ["Python-2x"]
|                                     | |
|                                     | | +-- ["tsWxGTUI_Py2x"]
|                                     | |
|                                     | +-- ["Python-3x"] (Ported from Python-2x)
|                                     | |
|                                     | | +-- ["tsWxGTUI_Py3x"]
|                                     | |
+-- "MANIFEST.in"
|
|     Deliverable File inclusion criteria list.
|
+-- "MANIFEST_template.in"
|
|     Deliverable Generic file inclusion criteria list
|     template for any Python version-specific TeamSTARS
|     "tsWxGTUI_PyVx" Toolkit.
|
+-- "MANIFEST_TREE.html"
|
|     Non-Deliverable Diagram (Multi-Level Org Chart)
|     depicting the hierarchical relationship between files
|     in the release, in Hypertext Markup Language format.
|
|     Diagram created via Command "./MANIFEST_TREE.sh".
|
+-- "MANIFEST_TREE.sh"
|
|     Deliverable POSIX-style Command Line Interface shell
|     script to generate diagrams depicting the hierarchical
|     relationship between files in the release
|     ("MANIFEST_TREE.html" and "MANIFEST_TREE.txt").
|

```



```

+-- "MANIFEST_TREE.txt"
|
|   Non-Deliverable Diagram (Multi-Level Org Chart)
|   depicting the hierarchical relationship between
|   files in the release, in Plain Text format.
|
|   Diagram created via Command "./MANIFEST_TREE.sh".
|
+-- "setup_Technical_Preview_tar_file.sh"
|
|   Deliverable POSIX-style Command Line Interface shell
|   script to generate downloadable "tarball" file.
|
+-- "setup_Technical_Preview_zip_file.sh"
|
|   Deliverable POSIX-style Command Line Interface shell
|   script to generate downloadable "zip" file.
|
+-- "README.txt"

```

===== TABLE OF CONTENTS =====

1. Repository

1.1 Documents

1.2 ManPages

1.3 Notebooks-Site-Packages

1.3.1 Preview-Project

1.3.2 Preview-Site-Packages

1.4 SourceDistributions-Site-Packages

1.4.1 tsWxGTUI_Py2x

1.4.2 tsWxGTUI_Py3x

1.5 Manifest

===== REPOSITORY =====

1. Repository

Excerpt From Wikipedia, the free encyclopedia:

"A software repository is a storage location from which software packages may be retrieved and installed on a computer."

This is the repository for the TeamSTARS "tsWxGTUI_PyVx" Toolkit. It is the collection of documentation and computer program source code files that is being distributed by its author in order to publish and share the intellectual property with others.

It contains the following subdirectories and files:

1.1 Documents

Contains introductory and other training information for installers, developers, operators, troubleshooters and distributors.

Toolkit software documentation is written in plain text that accompanies computer software. It either explains how it operates or how to use it, and may mean different things to people in different roles. Types of software documentation include:

- a) Requirements - Statements that identify attributes, capabilities, characteristics, or qualities of a system. This is the foundation for what shall be or has been implemented.
- b) Architecture/Design - Overview of software. Includes relations to an environment and construction principles to be used in design of software components.
- c) Technical - Documentation of code, algorithms, interfaces, and APIs.
- d) End user - Manuals for the end-user, system administrators and support staff.

1.2 ManPages

Contains a collection of man pages. A man page (short for manual page) is a form of online software documentation usually found on a Unix or Unix-like operating system. Topics covered include computer programs (including library and system calls), formal standards and conventions, and even abstract concepts.

Categories of man pages include:

- a) tsManPagesLibCLI
- b) tsManPagesLibGUI
- c) tsManPagesTestsLibCLI
- d) tsManPagesTestsLibGUI
- e) tsManPagesTestsToolsLibCLI
- f) tsManPagesTestsToolsLibGUI (Future)
- g) tsManPagesToolsCLI
- h) tsManPagesToolsGUI (Future)
- i) tsManPagesUtilities (Future)

1.3 Notebooks

Contains a collection of commentaries that express opinions or offerings of explanations about events

or situations that might be useful to Toolkit installers, developers, operators, troubleshooters and distributors. The documents may be in Application-specific formats (Adobe PDF, JPEG Bit-mapped image, Microsoft Office, Plain text etc.).

The collection includes:

- a) DeveloperDocuments (API-References-Pixel-Mode-wxPython and Developer-ReadMe-Files); and
- b) EngineeringDocuments (Product Marketing Documentation, Project Documentation and Technical Documentation).

1.3.1 Preview-Project

1.3.2 Preview-Site-Packages

1.4 SourceDistributions-Site-Packages

Contains a collection of computer program source code files that the Toolkit recipient will need to re-create the Toolkit.

The collection includes:

1.4.1 tsWxGTUI_Py2x

The source code files appropriate for use with Python 2.4-2.7.

1.4.2 tsWxGTUI_Py3x

The source code files appropriate for use with Python 3.0-3.4. These files are generated by converting their Python 2x counterparts with the 2to3 translation utility followed by debugging of unresolved syntax and type conversion issues.

A site-package is the location where third-party packages will be installed (i.e., those not part of the core Python distribution that includes os, sys, platform, curses, logging etc.).

NOTES:

- a) With Linux, Mac OS X and Unix operating systems one must have "root" or administrator privileges to write to the site-package location.
- b) If the user will not have permission to directly access the Repository but has a need

to know specific contents, the system administrator should copy the appropriate contents of the Documents, ManPages and Notenook directories into each Site-Package.

The copies were not included in the distribution in order to:

- (1) avoid increasing the release development and qualification efforts; and
 - (2) minimize the size of the downloadable "tar" and "zip" files.
- c) Users of third-party site-packages must explicitly import via its path from top-level package through lower-level packages to module:

```
site-package.package.module
```

- d) Examples for Python 2.4-2.7 site-packages:

```
from tsWxGTUI_Py2x.tsLibCLI import tsCxGlobals
from tsWxGTUI_Py2x.tsLibCLI import tsPlatformRunTimeEnvironment
from tsWxGTUI_Py2x.tsLibCLI import tsExceptions as tse
from tsWxGTUI_Py2x.tsLibCLI import tsLogger

from tsWxGTUI_Py2x.tsLibGUI import tsWx as wx
```

- d) Examples for Python 3.0-3.4 site-packages:

```
from tsWxGTUI_Py3x.tsLibCLI import tsCxGlobals
from tsWxGTUI_Py3x.tsLibCLI import tsPlatformRunTimeEnvironment
from tsWxGTUI_Py3x.tsLibCLI import tsExceptions as tse
from tsWxGTUI_Py3x.tsLibCLI import tsLogger

from tsWxGTUI_Py3x.tsLibGUI import tsWx as wx
-----
```

1.5 Manifest

For a listing of the repository contents (complete with last modified date, time, size and access permissions), see:

```
"./MANIFEST_TREE.html"
"./MANIFEST_TREE.txt"
```

For additional commentary on the repository contents see:

```
"./Documents/README-Manifest.txt"
```

===== End-Of-File =====

28.4 README3-Documents.txt

```
#-----
#"Time-stamp: <06/03/2015  7:05:59 AM rsg>"
#-----

===== File: README3-Documents.txt =====

+-----+-----+ TeamSTARS "tsWxGTUI_PyVx" Toolkit
| ts | Wx |      with Python 2x & Python 3x based
+-----+-----+      Command Line Interface (CLI)
| G T U I |      and "Curses"-based "wxPython"-style,
+-----+-----+      Graphical-Text User Interface (GUI)

Get that cross-platform, pixel-mode "wxPython" feeling
on character-mode 8-/16-color (xterm-family) & non-color
(vt100-family) terminals and terminal emulators.

You can find this and other files in the following
Toolkit subdirectories:

<Your Working Repository>
(e.g. "Technical_Preview")
|
|   Working repository containing directories and
|   files to be packaged into downloadable "tarball"
|   and/or "zip" files via the setup shell scripts
|   at the bottom of this diagram.
|
+-- ["Documents"]
|
|   |   This directory contains a collection of files
|   |   which provide the Toolkit recipient with an
|   }   understanding of the purpose, goals & capabil-
|   |   ities, non-goals & limitations, terms & condi-
|   |   tions and procedures for installing, operating,
|   |   modifying and redistributing the Toolkit.
|   |
|   +-- "AUTHORS.txt"
|   +-- "BUGS.txt"
|   +-- "CHANGE_LOG.txt"
|   +-- "CONFIGURE.txt"
|   +-- "COPYING.txt"
|   +-- "COPYRIGHT.txt"
|   +-- "CREDITS.txt"
|   +-- "DEMO.txt"
|   +-- "FAQ.txt"
|   +-- "GETTING_STARTED.txt"
|   +-- "INSTALL.txt"
|   +-- "LICENSE.txt"
|   +-- "NEWS.txt"
|   +-- "NOTICES.txt"
```

```
|      +-- "OPERATE.txt"
|      +-- "README.txt"
|      +-- "README1-Introduction.txt"
|      +-- "README2-Repository.txt"
|      +-- "README3-Documents.txt"
|      +-- "README4-ManPages.txt"
|      +-- "README5-Notebooks.txt"
|      +-- "README6-SourceDistributions.txt"
|      +-- "THANKS.txt"
|      +-- "TO-DO.txt"
|      +-- "TROUBLESHOOT.txt"
|
+-- ["ManPages"]
|
+-- ["Notebooks"]
|
+-- ["SourceDistributions"]
|
+-- "README.txt"

===== ["DOCUMENTS"] =====

This directory contains a collection of files which provide the Toolkit recipient with an understanding of the purpose, goals & capabilities, non-goals & limitation, terms & conditions and procedures and for installing, operating, modifying and redistributing the Toolkit.

"GETTING_STARTED.txt" --- Introduces new recipients to
                           the system requirements and
                           third-party resources available to new Toolkit users.

"README.txt" --- Introduces new recipients to the
                  purpose, goals, non-goals, design
                  and features of the computer
                  software product. Supplements
                  include the following:

                  "README1-Introduction.txt"
                  "README2-Repository.txt"
                  "README3-Documents.txt"
                  "README4-ManPages.txt"
                  "README5-Notebooks.txt"
                  "README6-SourceDistributions.txt"

"AUTHORS.txt" --- List of the principal "tsWxGTUI"
                  Toolkit author(s) and authors
                  credited for work covered by a prior
                  copyright and license.

"BUGS.txt" --- List of Known Problems / Issues.

"CHANGE_LOG.txt" --- List of Additions, Modification and
                    Deletions.
```

"CONFIGURE.txt" --- Instructions for applying factory and site-specific configurations.

"COPYING.txt" --- Instructions for copying all or a portion of the distribution.

"FAQ.txt" --- Answers to Frequently Asked Questions.

"INSTALL.txt" --- Describes steps to download, extract install and configure the "tsWxGUI" Toolkit.

"LICENSE.txt" --- General and special arrangements, provisions, rules, specifications and standards that form an integral part of the agreement or contract between the creator and recipient of Copy-righted and Licensed Work.

"MANIFEST.txt" --- Tally List for deliverable items.

"NEWS.txt" --- Announcements of new releases.

"NOTICES.txt" --- Details the copyright(s) and license(s).

"OPERATE.txt" --- Describes steps to use the "tsWxGUI" Toolkit.

"THANKS.txt" --- Acknowledgments to those otherwise unsung 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.

"TROUBLESHOOT.txt" --- Provides a list of available reference resources and a guide for planning, developing and troubleshooting a cross-platform system of hundreds of files each containing a few, tens or hundred of class, data and method definitions. Its complexity becomes apparent in the recent software Lines-Of-Code metrics.

===== End-Of-File =====

28.5 README4-ManPages.txt

```
#-----
#"Time-stamp: <06/03/2015  7:40:26 AM rsg>"
#-----

===== File: README4-ManPages.txt =====

+-----+-----+ TeamSTARS "tsWxGTUI_PyVx" Toolkit
| ts | Wx |      with Python 2x & Python 3x based
+-----+-----+      Command Line Interface (CLI)
| G T U I |      and "Curses"-based "wxPython"-style,
+-----+-----+      Graphical-Text User Interface (GUI)

Get that cross-platform, pixel-mode "wxPython" feeling
on character-mode 8-/16-color (xterm-family) & non-color
(vt100-family) terminals and terminal emulators.

You can find this and other files in the following
Toolkit subdirectories:

<Your Working Repository>
(e.g. "Technical_Preview")
|
+-- ["Documents"]
|
+-- ["ManPages"]
|
|   Deliverable Toolkit manual pages are a
|   form of online software documentation
|   usually found on a Unix or Unix-like oper-
|   ating system.
|
|   Topics covered include computer programs
|   (including library and system calls),
|   formal standards and conventions, and even
|   abstract concepts.
|
|   A user may NOT invoke a man page by issu-
|   ing the man command. Instead, a user may
|   display a man page by issuing the
|   less <man document file> command.
|
+-- "README4-ManPages.txt"
|
+-- ["tsManPagesLibCLI"]
|
|   +-- "runPydoc_tsManPagesCLI.sh"
|   +-- "runPylint_tsManPagesCLI.sh"
|   +-- "tsApplication.man"
|   +-- "tsCommandLineEnv.man"
|   +-- "tsCommandLineInterface.man"
|   +-- "tsCxGlobals.man"
```



```

|         +-- "tsDoubleLinkedList.man"
|         +-- "tsExceptions.man"
|         +-- "tsGistGetTerminalSize.man"
|         +-- "tsLogger.man"
|         +-- "tsOperatorSettingsParser.man"
|         +-- "tsPlatformRunTimeEnvironment.man"
|         +-- "tsReportUtilities.man"
|         +-- "tsSysCommands.man"
|
| +-- ["tsManPagesLibGUI"]
|     |
|     +-- "runPydoc_tsManPagesGUI.sh"
|     +-- "runPylint_tsManPagesGUI.sh"
|     +-- "tsWx.man"
|     +-- "tsWxAcceleratorEntry.man"
|     +-- "tsWxAcceleratorTable.man"
|     +-- "tsWxApp.man"
|     +-- "tsWxBoxSizer.man"
|     +-- "tsWxButton.man"
|     +-- "tsWxCallLater.man"
|     +-- "tsWxCaret.man"
|     +-- "tsWxCheckBox.man"
|     +-- "tsWxChoice.man"
|     +-- "tsWxColor.man"
|     +-- "tsWxColorDatabase.man"
|     +-- "tsWxControl.man"
|     +-- "tsWxControlWithItems.man"
|     +-- "tsWxCursor.man"
|     +-- "tsWxDebugHandlers.man"
|     +-- "tsWxDialog.man"
|     +-- "tsWxDialogButton.man"
|     +-- "tsWxDisplay.man"
|     +-- "tsWxDoubleLinkedList.man"
|     +-- "tsWxEraseEvent.man"
|     +-- "tsWxEvent.man"
|     +-- "tsWxEventDaemon.man"
|     +-- "tsWxEventLoop.man"
|     +-- "tsWxEventLoopActivator.man"
|     +-- "tsWxEventQueueEntry.man"
|     +-- "tsWxEventTableEntry.man"
|     +-- "tsWxEvtHandler.man"
|     +-- "tsWxFlexGridSizer.man"
|     +-- "tsWxFocusEvent.man"
|     +-- "tsWxFrame.man"
|     +-- "tsWxFrameButton.man"
|     +-- "tsWxGauge.man"
|     +-- "tsWxGlobals.man"
|     +-- "tsWxGraphicalTextUserInterface.man"
|     +-- "tsWxGridBagSizer.man"
|     +-- "tsWxGridSizer.man"
|     +-- "tsWxItemContainer.man"
|     +-- "tsWxKeyboardState.man"
|     +-- "tsWxKeyEvent.man"
|     +-- "tsWxListBox.man"
|     +-- "tsWxLog.man"

```



```

|         +-- "test_tsPlatformRunTimeEnvironment.man"
|         +-- "test_tsSysCommand.man"
|
|     +-- ["tsManPagesTestsLibGUI"]
|     |
|     |     +-- "buildManPagesTestsGUI.sh"
|     |     +-- "test_tsWxBoxSizer.man"
|     |     +-- "test_tsWxCheckBox.man"
|     |     +-- "test_tsWxDisplay.man"
|     |     +-- "test_tsWxDoubleLinkedList.man"
|     |     +-- "test_tsWxGlobals.man"
|     |     +-- "test_tsWxGraphicalTextUserInterface.man"
|     |     +-- "test_tsWxGridSizer.man"
|     |     +-- "test_tsWxMultiFrameEnv.man"
|     |     +-- "test_tsWxRSM.man"
|     |     +-- "test_tsWxScrolledWindow.man"
|     |     +-- "test_tsWxScrolledWindowDual.man"
|     |     +-- "test_tsWxSplashScreen.man"
|     |     +-- "test_tsWxWidgets.man"
|     |
|     +-- ["tsManPagesToolsCLI"]
|     |
|     |     +-- "buildManPagesToolsCLI.sh"
|     |     +-- "runPydoc_tsManPagesToolsCLI.sh"
|     |     +-- "runPylint_tsManPagesToolsCLI.sh"
|     |     +-- "tsLinesOfCodeProjectMetrics.man"
|     |     +-- "tsPlatformQuery.man"
|     |     +-- "tsStripComments.man"
|     |     +-- "tsStripLineNumbers.man"
|     |     +-- "tsTreeCopy.man"
|     |     +-- "tsTreeTrimLines.man"
|     |
|     +-- ["tsManPagesToolsGUI"] (Future)
|     |
|     |     +-- To-Be-Determined
|     |
|     +-- ["tsManPagesToolsLibCLI"]
|     |
|     |     +-- To-Be-Determined
|     |
|     +-- ["tsManPagesToolsLibGUI"] (Future)
|     |
|     |     +-- To-Be-Determined
|     |
|     +-- ["tsManPagesUtilitiesCLI"] (Future)
|     |
|     |     +-- To-Be-Determined
|
+-- ["Notebooks"]
+-- ["SourceDistributions"]
+-- "README.txt"

```

===== TABLE OF CONTENTS =====

1. ["ManPages"]

2. How to create and install a manpage (Future)

===== ["ManPages"] =====

1. ["ManPages"]

The following defines the purpose and use of a set of on-line reference documents. This Toolkit provides utility scripts that:

- a) create rudimentary ManPages from source code;
- b) do NOT yet merge Toolkit ManPages with ManPages installed by the host computer operating system or with the installation of other third-pary add-ons.

Excerpt From Wikipedia, the free encyclopedia:

"A man page (short for manual page) is a form of online software documentation usually found on a Unix or Unix-like operating system. Topics covered include computer programs (including library and system calls), formal standards and conventions, and even abstract concepts. A user may invoke a man page by issuing the man command.

By default, man typically uses a terminal pager program such as more or less to display its output.

Usage

To read a manual page for a Unix command, type:

```
man <command_name>
```

Pages are traditionally referred to using the notation "name(section)": for example, ftp(1). The same page name may appear in more than one section of the manual, such as when the names of system calls, user commands, or macro packages coincide. Examples are man(1) and man(7), or exit(2) and exit(3).

The syntax for accessing the non-default manual section varies between different man implementations. On Solaris, for example, the syntax for reading printf(3C) is:

```
man -s 3c printf
```

On Linux and BSD derivatives the same invocation would be:

```
man 3 printf
```

which searches for printf in section 3 of the man pages."

===== "How to create and install a manpage" =====

2. How to create and install a manpage (Future)

The following may be useful for a future Toolkit enhancement.

Excerpts from Google Search:

"HowTo: Linux / UNIX Create a Manpage - nixCraft
www.cyberciti.biz/faq/linux-unix-creating-a-manpage/
 May 6, 2010 - How do I create a man page for my shell
 or python script under Linux / UNIX ...
 install -g 0 -o 0 -m 0644 nuseradd.1 /usr/local/man/man8/ gzip ...

How to create a manpage? - Ask Ubuntu
askubuntu.com/questions/42923/how-to-create-a-manpage
 Ask Ubuntu
 May 15, 2011 - With the help of Gmanedit · Install
 gmanedit you are able to create manpages with a
 graphical GUI. Gtk+ Manpages Editor is an editor for man ...

Linux Man Page Howto - Who is Jens Schweikhardt?
www.schweikhardt.net/man_page_howto.html
 The next decision is the directory in which it will
 finally be installed (say, when the user runs ` make
 install ' for your package.) On Linux, all man pages
 are below ...

Linux Howtos: System -> Creating Your Own MAN Page
www.linuxhowtos.org/system/creatingman.htm
 We will be using groff macros to create our manual page.
 These macros always ... Normally you put the version
 number of your program here. [center header]

How can I add man page entries for my own power tools?
unix.stackexchange.com/.../how-can-i-add-man-page-ent...
 Stack Exchange
 Feb 4, 2011 - I have no idea about how I can make my
 home-grown specialist scripts ... and you can create
 a man page from the POD file with the pod2man ... You
 can then optionally (b|g)zip it and put it in the
 appropriate man directory.

How to add entry in Linux man page database - Stack ...
stackoverflow.com/.../how-to-add-entry-in-linux-man-page-database
 Dec 25, 2012 - I have a manual page for mongoose web
 server named as mongoose.1 as a result of doing make
 and make install command to install ...

How should a formatted man page look?
www.tldp.org/HOWTO/Man-Page/q3.html
 Linux Documentation Project

Here comes the man page for the (hypothetical)
foo program. ... However, if you install using
'make prefix=/opt/gnu' the references in the man
page change to ..."

===== End-Of-File =====

Draft

28.6 README5-Notebooks.txt

```
#-----
#"Time-stamp: <06/03/2015  6:50:16 AM rsg>"
#-----

===== File: README5-Notebooks.txt =====

+-----+-----+ TeamSTARS "tsWxGTUI_PyVx" Toolkit
| ts | Wx |      with Python 2x & Python 3x based
+-----+-----+      Command Line Interface (CLI)
| G T U I |      and "Curses"-based "wxPython"-style,
+-----+-----+      Graphical-Text User Interface (GUI)

Get that cross-platform, pixel-mode "wxPython" feeling
on character-mode 8-/16-color (xterm-family) & non-color
(vt100-family) terminals and terminal emulators.

<Your Working Repository>
(e.g. "Technical_Preview")
|
|   Working repository containing directories and
|   files to be packaged into downloadable "tarball"
|   and/or "zip" files via the setup shell scripts
|   at the bottom of this diagram.
|
+-- ["Documents"]
|
|   |   This directory contains a collection of files
|   |   which provide the Toolkit recipient with an
|   |   understanding of the purpose, goals & capabil-
|   |   ities, non-goals & limitations, terms & condi-
|   |   tions and procedures for installing, operating,
|   |   modifying and redistributing the Toolkit.
|   |
|   +-- "README.txt"
|   +-- "README1-Introduction.txt"
|   +-- "README2-Repository.txt"
|   +-- "README3-Documents.txt"
|   +-- "README4-ManPages.txt"
|   +-- "README5-Notebooks.txt"
|   +-- "README6-SourceDistributions.txt"
|
+-- ["ManPages"]
|
|   |   Deliverable Toolkit manual pages are a
|   |   form of online software documentation
|   |   usually found on a Unix or Unix-like
|   |   operating system.
|   |
|   |   Topics covered include computer programs
|   |   (including library and system calls),
```

```
|      | formal standards and conventions, and even
|      | abstract concepts.
|
|      | Unlike their Unix or Unix-like counterparts,
|      | a Toolkit user may NOT invoke a man page by
|      | issuing the "man command". Instead, a user
|      | must display a man page by issuing the
|      | "less <man document file>" command.
|
|      |
|      | +--- ["tsManPagesLibCLI"]
|      | +--- ["tsManPagesLibGUI"]
|      | +--- ["tsManPagesTestsLibCLI"]
|      | +--- ["tsManPagesTestsLibGUI"]
|      | +--- ["tsManPagesToolsCLI"]
|      | +--- ["tsManPagesToolsGUI"]      (Future)
|      | +--- ["tsManPagesToolsLibCLI"]
|      | +--- ["tsManPagesToolsLibGUI"]  (Future)
|      | +--- ["tsManPagesUtilitiesCLI"] (Future)
|      |
|      | +--- "README4-ManPages.txt"
|
+--- ["Notebooks"]      (Pre-dates Documents)
|
|      | Contains a collection of commentaries that
|      | express opinions or offerings of explana-
|      | tions about events or situations that might
|      | be useful to Toolkit installers, developers,
|      | operators, troubleshooters and distributors.
|      | The documents may be in Application-specific
|      | formats (such as Adobe PDF, JPEG Bit-mapped
|      | image, LibreOffice, Microsoft Office, plain
|      | text).
|
|      | +--- ["DeveloperNotebook"]
|      |
|      |      | Contains a collection of:
|      |      | API-References-Pixel-Mode-wxPython
|      |      | and Developer-ReadMe-Files
|      |
|      | +--- ["EngineeringNotebook"]
|      |
|      |      | Contains a Toolkit Developer oriented collection of:
|      |      | Project (purpose,
|      |      |      | goals,
|      |      |      | non-goals,
|      |      |      | features,
|      |      |      | capabilities,
|      |      |      | limitations),
|      |      | Plan (software life-cycle),
|      |      | Requirements (purpose,
|      |      |      | goals,
|      |      |      | non-goals,
|      |      |      | features,
|      |      |      | capabilities,
|      |      |      | limitations,
|      |      |      | file system configuration,
```



```

|         hardware & software interface,
|         software,
|         system,
|         user configuration options),
|     Design (API emulation strategy, architecture),
|     Implementation (developer-sandbox, site-package),
|     Test (unit, integration, system, acceptance),
|     Marketing (announcement, brochure),
|     Release (introduction,
|             release notes,
|             software user's manual,
|             terms & conditions,
|             dictionary),
|     Third-party Resources
|
|+-- ["ProjectNotebook"]
|
|     Contains a Toolkit User oriented collection of:
|     ["EngineeringNotebook"] abstracts
|
|+-- "README5-Notebooks.txt"
|
|+-- ["SourceDistributions"]
|
|     Contains a collection of computer program
|     source code files that the Toolkit recip-
|     ient will need to install, operate, modify
|     and re-distribute the Toolkit.
|
|+-- "README6-SourceDistributions.txt"
|
|+-- ["Developer-Sandboxes"] (Pre-dates Site-Packages)
|
|     A sandbox is a testing environment that iso-
|     lates untested code changes and outright
|     experimentation from the production environ-
|     ment or repository.
|
|+-- ["tsWxGTUI_PyVx"]
|
|     +-- ["Documents"]
|
|         This directory contains a collection of files
|         which provide the Toolkit recipient with an
|         understanding of the purpose, goals & capabil-
|         ities, non-goals & limitations, terms & condi-
|         tions and procedures for installing, operating,
|         modifying and redistributing the Toolkit.
|
|+-- ["ManPages"]
|
|     Deliverable Toolkit manual pages are a
|     form of online software documentation
|     usually found on a Unix or Unix-like oper-
|     ating system.

```



```

|           | Topics covered include computer programs
|           | (including library and system calls),
|           | formal standards and conventions, and even
|           | abstract concepts.
|           |
|           | A user may NOT invoke a man page by issu-
|           | ing the man command. Instead, a user may
|           | display a man page by issuing the
|           | less <man document file> command.
|           |
|           |
|           | +-- ["tsManPagesLibCLI"]
|           | +-- ["tsManPagesLibGUI"]
|           | +-- ["tsManPagesTestsLibCLI"]
|           | +-- ["tsManPagesTestsLibGUI"]
|           | +-- ["tsManPagesToolsCLI"]
|           | +-- ["tsManPagesToolsGUI"] (Future)
|           | +-- ["tsManPagesToolsLibCLI"]
|           | +-- ["tsManPagesToolsLibGUI"] (Future)
|           | +-- ["tsManPagesUtilitiesCLI"] (Future)
|           |
|           | +-- ["Python-2x"]
|           | |
|           | | +-- ["tsWxGTUI_Py2x"]
|           | |
|           | +-- ["Python-3x"] (Ported from Python-2x)
|           | |
|           | | +-- ["tsWxGTUI_Py3x"]
|           |
| +-- "MANIFEST.in"
|
|     Deliverable File inclusion criteria list.
|
| +-- "MANIFEST_template.in"
|
|     Deliverable Generic file inclusion criteria list
|     template for any Python version-specific TeamSTARS
|     "tsWxGTUI_PyVx" Toolkit.
|
| +-- "MANIFEST_TREE.html"
|
|     Non-Deliverable Diagram (Multi-Level Org Chart)
|     depicting the hierarchical relationship between files
|     in the release, in Hypertext Markup Language format.
|
|     Diagram created via Command "./MANIFEST_TREE.sh".
|
| +-- "MANIFEST_TREE.sh"
|
|     Deliverable POSIX-style Command Line Interface shell
|     script to generate diagrams depicting the hierarchical
|     relationship between files in the release
|     ("MANIFEST_TREE.html" and "MANIFEST_TREE.txt").
|
| +-- "MANIFEST_TREE.txt"
|

```

```
|      Non-Deliverable Diagram (Multi-Level Org Chart)
|      depicting the hierarchical relationship between
|      files in the release, in Plain Text format.
|
|      Diagram created via Command "./MANIFEST_TREE.sh".
|
+-- "setup_Technical_Preview_tar_file.sh"
|
|      Deliverable POSIX-style Command Line Interface shell
|      script to generate downloadable "tarball" file.
|
+-- "setup_Technical_Preview_zip_file.sh"
|
|      Deliverable POSIX-style Command Line Interface shell
|      script to generate downloadable "zip" file.
|
+-- "README.txt"
```

===== TABLE OF CONTENTS =====

- 1. ["DeveloperNotebook"]
 - 1.1 ["API-References-Pixel-Mode-wxPython"] (Future)
 - 1.2 ["DeveloperReadMeFiles"] (Future)
- 2. ["EngineeringNotebook"]
 - 2.1 Product Marketing Documentation (Future)
 - 2.2 Project Documentation
 - 2.3 Technical Documentation
- 3. ["ProjectNotebook"]
 - 3.1 ["API-Preview"]
 - 3.2 ["API-References-Pixel-Mode-wxPython"] (Future)
 - 3.3 ["DeveloperReadMeFiles"] (Future)

===== ["DeveloperNotebook"] =====

- 1. ["DeveloperNotebook"]

This directory contains a collection of files which provide the Toolkit Building Block, Tool and Application programmer with an understanding of the design and usage of the Toolkit's CLI and GUI Application Programming Interface.

- 1.1 ["API-References-Pixel-Mode-wxPython"]
- 1.2 ["DeveloperReadMeFiles"]

===== ["EngineeringNotebook"] =====

- 2. ["EngineeringNotebook"]

This directory contains a collection of files, in application-specific formats, which provide the Toolkit architect, product manager, project engineer, system engineer, software engineer, test engineer and troubleshooter with:

2.1 Product Marketing Documentation

- a) announcement notice(s)
- b) brochure(s)
- c) introduction

2.2 Project Documentation

- a) goal and capability objectives
- b) non-goal and limitation constraints
- c) project plans

2.3 Technical Documentation

- a) architectural plans
- b) system specifications
- c) Interface specifications
- d) software specifications
- e) design specifications
- f) test specifications
- g) software user manual

===== ["ProjectNotebook"] =====

3. ["ProjectNotebook"]

===== End-Of-File =====

28.7 README6- SourceDistributions.txt

```
#-----
#"Time-stamp: <06/05/2015  4:25:31 AM rsg>"
#-----

===== File: README6-SourceDistributions.txt =====

+-----+-----+ TeamSTARS "tsWxGTUI_PyVx" Toolkit
| ts | Wx |       with Python 2x & Python 3x based
+-----+-----+       Command Line Interface (CLI)
| G T U I |       and "Curses"-based "wxPython"-style,
+-----+-----+       Graphical-Text User Interface (GUI)

Get that cross-platform, pixel-mode "wxPython" feeling
on character-mode 8-/16-color (xterm-family) & non-color
(vt100-family) terminals and terminal emulators.

You can find this and other files in the following sub-
directories:

<Your Working Repository>
(e.g. "Technical_Preview")
|
|   Working repository containing directories and
|   files to be packaged into downloadable "tarball"
|   and/or "zip" files via the setup shell scripts
|   at the bottom of this diagram.
|
+-- ["Documents"]
|
|   This directory contains a collection of files
|   which provide the Toolkit recipient with an
|   understanding of the purpose, goals & capabil-
|   ities, non-goals & limitations, terms & condi-
|   tions and procedures for installing, operating,
|   modifying and redistributing the Toolkit.
|
+-- "README3-Documents.txt"
|
+-- ["ManPages"]
|
|   Deliverable Toolkit manual pages are a
|   form of online software documentation
|   usually found on a Unix or Unix-like oper-
|   ating system.
|
+-- "README4-ManPages.txt"
|
+-- ["Notebooks"]
```

```

|
| Contains a collection of commentaries that
| express opinions or offerings of explana-
| tions about events or situations that might
| be useful to Toolkit installers, developers,
| operators, troubleshooters and distributors.
| The documents may be in Application-specific
| formats (Adobe PDF, JPEG Bit-mapped image,
| Microsoft Office, Plain text etc.).
|
| +-- "README5-Notebooks.txt"
|
+-- ["SourceDistributions"]
|
| Contains a collection of computer program
| source code files that the Toolkit recip-
| ient will need to install, operate, modify
| and re-distribute the Toolkit.
|
| +-- "README6-SourceDistributions.txt"
|
+-- ["Developer-Sandboxes"] (Pre-dates Site-Packages)
|
| A sandbox is a testing environment that iso-
| lates untested code changes and outright experi-
| mentation from the production environment or
| repository.
|
| +-- ["tsWxGTUI_PyVx"]
|
| Contains one or more Python language gener-
| ation-specific releases each sharing the same
| programmer (API) and user (CLI & GUI) inter-
| faces, documents and manual pages.
|
| +-- ["Documents"] (copy)
|
| +-- ["ManPages"] (copy)
|
+-- ["Python-2x"]
|
| Second generation Python programming
| language.
|
| +-- ["tsWxGTUI_Py2x"]
|
| +-- ["tsDemoArchive"]
|
| +-- ["src"]
|
| +-- "TermsAndConditions.txt"
|
| +-- ["tsLibCLI"]
|
| +-- ["tsApplicationPkg"]

```

[illegible]


```

    |         +-- ["src"]
    |         +-- ["test"]
    |     +-- ["tsLibGUI"]
    |     |   +-- ["tsWxPkg"]
    |     |   |   +-- ["src"]
    |     |   |   +-- ["test"]
    |     +-- ["tsToolsCLI"]
    |     |   +--
["tsLinesOfCodeProjectMetricsPkg"]
    |         |       |
    |         |       +-- ["src"]
    |         |       +-- ["test"]
    |         +-- ["tsPlatformQueryPkg"]
    |         |   +-- ["src"]
    |         |   +-- ["test"]
    |         +-- ["tsStripCommentsPkg"]
    |         |   +-- ["src"]
    |         |   +-- ["test"]
    |         +-- ["tsStripLineNumbersPkg"]
    |         |   +-- ["src"]
    |         |   +-- ["test"]
    |         +-- ["tsTreeCopyPkg"]
    |         |   +-- ["src"]
    |         |   +-- ["test"]
    |         +-- ["tsTreeTrimLinesPkg"]
    |         |   +-- ["src"]
    |         |   +-- ["test"]
    |     +-- ["tsToolsGUI"]
    |     +-- ["tsUtilities"]
+-- ["Python-3x"] (Ported from Python-2x)
|   Third generation Python programming
|   language.
|   +-- ["tsWxGTUI_Py3x"]

```

```
|      +--- ["Site-Packages"]
|      |
|      |   A site-packages is the location where third-
|      |   party packages are installed (i.e., those
|      |   not part of the core Python distribution).
|      |   NOTE: That with Linux, Mac OS X and Unix
|      |   operating systems one must have root priv-
|      |   ileages to write to that location.
|      |
|      +--- ["tsWxGTUI_PyVx"]
|          |
|          +--- ["Documents"] (copy)
|          |
|          +--- ["ManPages"] (copy)
|          |
|          +--- ["Python-2x"]
|              |
|              |   Second generation Python programming
|              |   language.
|              |
|              +--- ["tsWxGTUI_Py2x"]
|                  |
|                  +--- ["tsDemoArchive"]
|                      |
|                      +--- ["tsTestsLibCLI"]
|                      +--- ["tsTestsLibGUI"]
|                      +--- ["tsTestsToolsCLI"]
|                      +--- ["tsTestsToolsGUI"]
|                      +--- ["tsTestsToolsLibCLI"]
|                      +--- ["tsTestsToolsLibGUI"]
|                  +--- ["tsLibCLI"]
|                  |
|                  +--- ["tsLibGUI"]
|                  |
|                  +--- ["tsToolsCLI"]
|                  |
|                  +--- ["tsToolsGUI"]
|                  |
|                  +--- ["tsUtilities"]
|
|      +--- ["Python-3x"] (Ported from Python-2x)
|          |
|          |   Third generation Python programming
|          |   language.
|          |
|          +--- ["tsWxGTUI_Py3x"]
|
+--- "README.txt"
```

===== TABLE OF CONTENTS =====

1. Source Code
2. Developer-Sandbox

- 2.1 tsLibCLI (equivalent to 3.1.1 tsTestsLibCLI)
- 2.2 tsLibGUI (equivalent to 3.1.2 tsTestsLibGUI)
- 2.3 tsToolsCLI (equivalent to 3.1.3 tsTestsToolsCLI
and 3.1.5 tsTestsToolsLibCLI)
- 2.4 tsToolsGUI (equivalent to 3.1.4 tsTestsToolsGUI)
- 2.5 tsUtilities (equivalent to 3.8 tsUtilities)

3. Site-Package

3.1 tsDemoArchive

- 3.1.1 tsTestsLibCLI
- 3.1.2 tsTestsLibGUI
- 3.1.3 tsTestsToolsCLI
- 3.1.4 tsTestsToolsGUI
- 3.1.5 tsTestsToolsLibCLI
- 3.1.6 tsTestsToolsLibGUI

- 3.2 tsLibCLI
- 3.3 tsLibGUI
- 3.4 tsToolsCLI
- 3.5 tsToolsGUI
- 3.6 tsToolsLibCLI
- 3.7 tsToolsLibGUI
- 3.8 tsUtilities

===== SourceDistribution =====

1. Source Code

Excerpt From Wikipedia, the free encyclopedia:

"In computing, source code is any collection of computer instructions (possibly with comments) written using some human-readable computer language, usually as text. The source code of a program is specially designed to facilitate the work of computer programmers, who specify the actions to be performed by a computer mostly by writing source code. The source code is often transformed by a compiler program into low-level machine code understood by the computer. The machine code might then be stored for execution at a later time. Alternatively, an interpreter can be used to analyze and perform the outcomes of the source code program directly on the fly.

Most computer applications are distributed in a form that includes executable files, but not their source code. If the source code were included, it would be useful to a user, programmer, or system administrator, who may wish to modify the program or to understand how it works.

Aside from its machine-readable forms, source code also appears in books and other media; often in the form of small code snippets, but occasionally complete code bases; a well-known case is the source code of PGP."

===== DEVELOPER-SANDBOX =====

2. Developer-Sandbox

Excerpt From Wikipedia, the free encyclopedia:

"A sandbox is a testing environment that isolates untested code changes and outright experimentation from the production environment or repository, in the context of software development including Web development and revision control. Sandboxing protects "live" servers and their data, vetted source code distributions, and other collections of code, data and/or content, proprietary or public, from changes that could be damaging (regardless of the intent of the author of those changes) to a mission-critical system or which could simply be difficult to revert. Sandboxes replicate at least the minimal functionality needed to accurately test the programs or other code under development (e.g. usage of the same environment variables as, or access to an identical database to that used by, the stable prior implementation intended to be modified; there are many other possibilities, as the specific functionality needs vary widely with the nature of the code and the application[s] for which it is intended.)

The concept of the sandbox (sometimes also called a working directory, a test server or development server) is typically built into revision control software such as CVS and Subversion (SVN), in which developers "check out" a copy of the source code tree, or a branch thereof, to examine and work on. Only after the developer has (hopefully) fully tested the code changes in their own sandbox should the changes be checked back into and merged with the repository and thereby made available to other developers or end users of the software.[1]

By further analogy, the term "sandbox" can also be applied in computing and networking to other temporary or indefinite isolation areas, such as security sandboxes and search engine sandboxes (both of which have highly specific meanings), that prevent incoming data from affecting a "live" system (or aspects thereof) unless/ until defined requirements or criteria have been met."

Unlike the contents of the installable site-package, this sandbox uses a multi-level tree of subdirectories and associated files whose topology is defined by a set of package "__init__.py" files which collaborate in performing dynamic path generation and importing of modules and subpackages. Applications import individual packages and individual modules simply by name (if module name is unique) or by package.module name (if module name is not unique).

Source Code Manifest Checklist:

```

2.1 tsLibCLI      (equivalent to 3.1.1 tsTestsLibCLI)
2.2 tsLibGUI      (equivalent to 3.1.2 tsTestsLibGUI)
2.3 tsToolsCLI    (equivalent to 3.1.3 tsTestsToolsCLI
                    and 3.1.5 tsTestsToolsLibCLI)
2.4 tsToolsGUI    (equivalent to 3.1.4 tsTestsToolsGUI)
2.5 tsUtilities   (equivalent to 3.8 tsUtilities)

```

```
===== SITE-PACKAGE =====
```

3. Site-Package

Site-packages is the location where third-party packages are installed (i.e., those not part of the core Python distribution). NOTE: That with Linux, Mac OS X and Unix operating systems one must have root privileges to write to that location.

Unlike the contents of the Developer-Sandbox, the third-party site-package and its users must explicitly import via the site-package.package.module path identifier.

```

<Your Working Repository>
(e.g. "Technical_Preview")
|
|   Working repository containing directories and
|   files to be packaged into downloadable "tarball"
|   and/or "zip" files via the setup shell scripts
|   at the bottom of this diagram.
|
+-- ["Documents"]
|   |
|   |   This directory contains a collection of files
|   |   which provide the Toolkit recipient with an
|   |   understanding of the purpose, goals & capabilities,
|   |   non-goals & limitations, terms & conditions and
|   |   procedures for installing, operating, modifying and
|   |   redistributing the Toolkit.
|   |
|   +-- "README3-Documents.txt"
|
+-- ["ManPages"]
|   |
|   |   Deliverable Toolkit manual pages are a
|   |   form of online software documentation usually found
|   |   on a Unix or Unix-like operating system.
|   |
|   +-- "README4-ManPages.txt"
|
+-- ["Notebooks"]
|   |
|   |   Contains a collection of commentaries that express
|   |   opinions or offerings of explanations about events or
|   |   situations that might

```

```

|         | be useful to Toolkit installers, developers,
|         | operators, troubleshooters and distributors.
|         | The documents may be in Application-specific
|         | formats (Adobe PDF, JPEG Bit-mapped image,
|         | Microsoft Office, Plain text etc.).
|
|         +--- "README5-Notebooks.txt"
|
+--- ["SourceDistributions"]
|
|         | Contains a collection of computer program
|         | source code files that the Toolkit recip-
|         | ient will need to install, operate, modify
|         | and re-distribute the Toolkit.
|
|         +--- "README6-SourceDistributions.txt"
|
+--- ["Developer-Sandboxes"] (Pre-dates Site-Packages)
|
|         | A sandbox is a testing environment that iso-
|         | lates untested code changes and outright
|         | experimentation from the production environ-
|         | ment or repository.
|
|         +--- ["tsWxGTUI_PyVx"]
|         |
|         |         +--- ["Documents"] (copy)
|         |         |
|         |         +--- ["ManPages"] (copy)
|         |         |
|         |         +--- ["Python-2x"]
|         |         |
|         |         |         +--- ["tsWxGTUI_Py2x"]
|         |         |
|         |         +--- ["Python-3x"] (Ported from Python-2x)
|         |         |
|         |         +--- ["tsWxGTUI_Py3x"]
|
+--- ["Site-Packages"]
|
|         | Site-packages is the location where third-
|         | party packages are installed (i.e., those
|         | not part of the core Python distribution).
|         | NOTE: That with Linux, Mac OS X and Unix
|         | operating systems one must have root priv-
|         | ileages to write to that location.
|
|         +--- ["tsWxGTUI_PyVx"]
|         |
|         |         +--- ["Documents"] (copy)
|         |         |
|         |         +--- ["ManPages"] (copy)
|         |         |
|         |         +--- ["Python-2x"]
|         |         |
|         |         |         +--- ["tsWxGTUI_Py2x"]

```

```
+-- "README.txt"
```

3.1 tsDemoArchive

A collection of Python application programs for test (unit, integration, system, regression and acceptance) and demonstration of features, capabilities and limitations of the Command Line Interface (CLI) or Graphical-style User Interface (GUI) mode of operation.

3.1.1.1 tsTestsLibCLI

- 3.1.1.1.1 test_TermsAndConditions.py
- 3.1.1.1.2 test_tsApplication.py
- 3.1.1.1.3 test_tsCommandLineEnv.py
- 3.1.1.1.4 test_tsCommandLineInterface.py
- 3.1.1.1.5 test_tsCxGlobals.py
- 3.1.1.1.6 test_tsDoubleLinkedList.py
- 3.1.1.1.7 test_tsException.py
- 3.1.1.1.8 test_tsLogger.py
- 3.1.1.1.9 test_tsOperatorSettingsParser.py
- 3.1.1.1.10 test_tsPlatformRunTimeEnvironment.py
- 3.1.1.1.11 test_tsReportUtility.py
- 3.1.1.1.12 test_tsSysCommands.py

3.1.1.2 tsTestsLibGUI

- 3.1.2.1 test_cursesOverlappingPanels.py
- 3.1.2.2 test_cursesPanels.py
- 3.1.2.3 test_tsWxBoxSizer.py
- 3.1.2.4 test_tsWxCheckBox.py
- 3.1.2.5 test_tsWxColorPalette.py
- 3.1.2.6 test_tsWxDisplay.py
- 3.1.2.7 test_tsWxDoubleLinkedList.py
- 3.1.2.8 test_tsWxGlobals.py
- 3.1.2.9 test_tsWxGraphicalTextUserInterface.py
- 3.1.2.10 test_tsWxGridSizer.py
- 3.1.2.11 test_tsWxMultiFrameEnv.py
- 3.1.2.12 test_tsWxPasswordEntryDialog.py
- 3.1.2.13 test_tsWxRSM.py
- 3.1.2.14 test_tsWxScrolledWindow.py
- 3.1.2.15 test_tsWxScrolledWindowDual.py
- 3.1.2.16 test_tsWxSplashScreen.py
- 3.1.2.17 test_tsWxSystemSettings.py
- 3.1.2.17 test_tsWxTextEntryDialog.py
- 3.1.2.18 test_tsWxWidgets.py

3.1.1.3 tsTestsToolsCLI

- 3.1.3.1 ["basicFileTypes"]
- 3.1.3.2 ["regressionTestFileTypes"]
- 3.1.3.3 ["sampleHelp"]
- 3.1.3.4 "File_Extensions.txt"
- 3.1.3.5 tsLinesOfCodeProjectMetrics.py
- 3.1.3.6 tsPlatformQuery.py
- 3.1.3.7 tsReVersion.py
- 3.1.3.8 tsStripComments.py
- 3.1.3.9 tsStripLineNumbers.py
- 3.1.3.10 tsTreeCopy.py
- 3.1.3.11 tsTreeTrimLines.py

3.1.1.4 tsTestsToolsGUI

<None>

3.1.1.5 tsTestsToolsLibCLI


```

3.1.5.1 ["basicFileTypes"]
3.1.5.2 ["regressionTestFileTypes"]
3.1.3.3 "File_Extensions.txt"
3.1.5.4 test_tsLinesOfCode.py
3.1.5.5 test_tsStripComments.py
3.1.5.6 test_tsStripSettingsParser.py

```

3.1.6 tsTestsToolsLibGUI

<None>

3.2 tsLibCLI

A collection of building-block components for Python application programs used during the Command Line Interface mode of operation.

```

3.2.1 tsApplication.py
3.2.2 tsCommandLineEnv.py
3.2.3 tsCommandLineInterface.py
3.2.4 tsCxGlobals.py
3.2.5 tsDoubleLinkedList.py
3.2.6 tsException.py
3.2.7 tsGistGetTerminalSize.py
3.2.8 tsLogger.py
3.2.9 tsOperatorSettingsParser.py
3.2.10 tsPlatformRunTimeEnvironment.py
3.2.11 tsReportUtility.py
3.2.12 tsSysCommands.py

```

3.3 tsLibGUI

A collection of building-block components for Python application programs used during the Graphical-style User Interface mode of operation.

3.3.1 wxLibPython

A collection of building-blocks for the wxPython API Graphical-style User Interface mode of operation.

NOTE: "wxWidgets", used by software developers working in the C++ programming language, or an alternate programming language-specific wrapper (such as "wxPython") provides a pixel-mode cross-platform Application Programming Interface (API) which depends on the host platform's native pixel-mode GUI services (such as GNU/Linux, Mac OS X, Microsoft Windows or Unix).

```

3.3.1.1 tsWx.py
3.3.1.2 tsWxAcceleratorEntry.py
3.3.1.3 tsWxAcceleratorTable.py
3.3.1.4 tsWxApp.py

```

3.3.1.5 tsWxBoxSizer.py
3.3.1.6 tsWxButton.py
3.3.1.7 tsWxCallLater.py
3.3.1.8 tsWxCaret.py
3.3.1.9 tsWxCheckBox.py
3.3.1.10 tsWxChoice.py
3.3.1.11 tsWxColor.py
3.3.1.12 tsWxColorDatabase.py
3.3.1.13 tsWxControl.py
3.3.1.14 tsWxControlWithItems.py
3.3.1.15 tsWxCursor.py
3.3.1.16 tsWxDebugHandlers.py
3.3.1.17 tsWxDialog.py
3.3.1.18 tsWxDialogButton.py
3.3.1.19 tsWxDisplay.py
3.3.1.20 tsWxEraseEvent.py
3.3.1.21 tsWxEvent.py
3.3.1.22 tsWxEventDaemon.py
3.3.1.23 tsWxEventLoop.py
3.3.1.24 tsWxEventLoopActivator.py
3.3.1.25 tsWxEvtHandler.py
3.3.1.26 tsWxFlexGridSizer.py
3.3.1.27 tsWxFocusEvent.py
3.3.1.28 tsWxFrame.py
3.3.1.29 tsWxFrameButton.py
3.3.1.30 tsWxGauge.py
3.3.1.31 tsWxGridBagSizer.py
3.3.1.32 tsWxGridSizer.py
3.3.1.33 tsWxItemContainer.py
3.3.1.34 tsWxKeyEvent.py
3.3.1.35 tsWxListBox.py
3.3.1.36 tsWxLog.py
3.3.1.37 tsWxMenu.py
3.3.1.38 tsWxMenuBar.py
3.3.1.39 tsWxMouseEvent.py
3.3.1.40 tsWxMouseState.py
3.3.1.41 tsWxObject.py
3.3.1.42 tsWxPanel.py
3.3.1.43 tsWxPasswordEntryDialog.py
3.3.1.44 tsWxPoint.py
3.3.1.45 tsWxPyApp.py
3.3.1.46 tsWxPyEventBinder.py
3.3.1.47 tsWxPyOnDemandOutputWindow.py
3.3.1.48 tsWxPySimpleApp.py
3.3.1.49 tsWxPySizer.py
3.3.1.50 tsWxRadioBox.py
3.3.1.51 tsWxRadioButton.py
3.3.1.52 tsWxRect.py
3.3.1.53 tsWxScreen.py
3.3.1.54 tsWxScrollBar.py
3.3.1.55 tsWxScrolled.py
3.3.1.56 tsWxScrolledText.py
3.3.1.57 tsWxScrolledWindow.py
3.3.1.58 tsWxShowEvent.py
3.3.1.59 tsWxSize.py
3.3.1.60 tsWxSizer.py

- 3.3.1.61 tsWxSizerFlags.py
- 3.3.1.62 tsWxSizerItem.py
- 3.3.1.63 tsWxSizerItemList.py
- 3.3.1.64 tsWxSlider.py
- 3.3.1.65 tsWxSplashScreen.py
- 3.3.1.66 tsWxStaticBox.py
- 3.3.1.67 tsWxStaticBoxSizer.py
- 3.3.1.68 tsWxStaticLine.py
- 3.3.1.69 tsWxStaticText.py
- 3.3.1.70 tsWxStatusBar.py
- 3.3.1.71 tsWxSystemSettings.py
- 3.3.1.72 tsWxTextCtrl.py
- 3.3.1.73 tsWxTextEditBox.py
- 3.3.1.74 tsWxTextEntry.py
- 3.3.1.75 tsWxTextEntryDialog.py
- 3.3.1.76 tsWxTimer.py
- 3.3.1.77 tsWxToggleButton.py
- 3.3.1.78 tsWxTopLevelWindow.py
- 3.3.1.79 tsWxValidator.py
- 3.3.1.80 tsWxWindow.py

3.3.2 wxLibCurses

A collection of building-blocks for the Curses API Graphical-style User Interface mode of operation.

NOTE: These building blocks provide a cross-platform character-mode emulation of the host platform's native pixel-mode GUI services (such as GNU/Linux, Mac OS X, Microsoft Windows or Unix):

- a) converting between pixel- and character-mode dimensions;
- b) detects and/or sets up the color palette and foreground/background color pair combinations;
- c) handles input from terminal keyboard and mouse;
- d) formats and outputs control (position, size, font, color) and data (borders, lines, labels and messages) to the terminal display.
- e) determines which pixel-mode GUI object triggered the mouse click; and which one should receive the event notification.
- f) manages the terminal display layout to provide an application desktop area for frames and dialogs; an operator notification area for date and time stamped event messages; and a task bar area.

- 3.3.2.1 tsWxCursesKeyCodesDataBase.py
- 3.3.2.2 tsWxCursesMouseButtonCodesDataBase.py
- 3.3.2.3 tsWxCursesServices.py

- 3.3.2.4 tsWxDoubleLinkedList.py
- 3.3.2.5 tsWxEventQueueEntry.py
- 3.3.2.6 tsWxEventTableEntry.py
- 3.3.2.7 tsWxGlobals.py
- 3.3.2.8 tsWxGraphicalTextUserInterface.py
- 3.3.2.9 tsWxKeyboardState.py
- 3.3.2.10 tsWxMultiFrameEnv.py
- 3.3.2.11 tsWxNonLinkedList.py
- 3.3.2.12 tsWxPythonColor16DataBase.py
- 3.3.2.13 tsWxPythonColor16SubstitutionMap.py
- 3.3.2.14 tsWxPythonColor256DataBase.py
- 3.3.2.15 tsWxPythonColor88DataBase.py
- 3.3.2.16 tsWxPythonColor8DataBase.py
- 3.3.2.17 tsWxPythonColor8SubstitutionMap.py
- 3.3.2.18 tsWxPythonColorDataBaseRGB.py
- 3.3.2.19 tsWxPythonColorNames.py
- 3.3.2.20 tsWxPythonColorRGBNames.py
- 3.3.2.21 tsWxPythonColorRGBValues.py
- 3.3.2.22 tsWxPythonMonochromeDataBase.py
- 3.3.2.23 tsWxPythonPrivateLogger.py
- 3.3.2.24 tsWxScrollBarButton.py
- 3.3.2.25 tsWxScrollBarGauge.py
- 3.3.2.26 tsWxSizerSpacer.py
- 3.3.2.27 tsWxTaskBar.py
- 3.3.2.28 tsWxWindowCurses.py

3.4 tsToolsCLI

A collection of Python application programs for the Command Line Interface mode of operation.

- 3.4.1 tsChange_tsWxGTUI_PyVx.py
- 3.4.2 tsChange_wxPython_API_Version.py
- 3.4.3 tsLinesOfCodeProjectMetrics.py
- 3.4.4 tsPlatformQuery.py
- 3.4.5 tsStripComments.py
- 3.4.6 tsStripLineNumbers.py
- 3.4.7 tsTreeCopy.py
- 3.4.8 tsTreeTrimLines.py

3.5 tsToolsGUI

A collection of Python application programs for the Graphical-style User Interface mode of operation.

<none>

3.6 tsToolsLibCLI

A collection of building-block components for Python application program tools used during the Command Line Interface mode of operation.

3.7 tsToolsLibGUI

A collection of building-block components for Python

application program tools used during the Graphical-style User Interface mode of operation.

<none>

3.8 tsUtilities

A collection of operating system shell scripts and Python scripts used for hardware and software administration during the Command Line Interface mode of operation.

===== End-Of-File =====

Draft

28.8 README7- DeveloperSandboxes.txt

```
#-----
#"Time-stamp: <06/05/2015  4:25:31 AM rsg>"
#-----
```

```
===== File: README6-SourceDistributions.txt =====
```

```
+-----+-----+ TeamSTARS "tsWxGTUI_PyVx" Toolkit
| ts | Wx |      with Python 2x & Python 3x based
+-----+-----+      Command Line Interface (CLI)
| G T U I |      and "Curses"-based "wxPython"-style,
+-----+-----+      Graphical-Text User Interface (GUI)
```

Get that cross-platform, pixel-mode "wxPython" feeling on character-mode 8-/16-color (xterm-family) & non-color (vt100-family) terminals and terminal emulators.

You can find this and other files in the following sub-directories:

```
<Your Working Repository>
(e.g. "Technical_Preview")
|
|   Working repository containing directories and
|   files to be packaged into downloadable "tarball"
|   and/or "zip" files via the setup shell scripts
|   at the bottom of this diagram.
|
+-- ["Documents"]
|
|   This directory contains a collection of files
|   which provide the Toolkit recipient with an
|   understanding of the purpose, goals & capabilities,
|   non-goals & limitations, terms & conditions and
|   procedures for installing, operating, modifying and
|   redistributing the Toolkit.
|
+-- "README3-Documents.txt"
|
+-- ["ManPages"]
|
|   Deliverable Toolkit manual pages are a
|   form of online software documentation
|   usually found on a Unix or Unix-like operating
|   system.
|
+-- "README4-ManPages.txt"
|
+-- ["Notebooks"]
```

```

|
| Contains a collection of commentaries that
| express opinions or offerings of explana-
| tions about events or situations that might
| be useful to Toolkit installers, developers,
| operators, troubleshooters and distributors.
| The documents may be in Application-specific
| formats (Adobe PDF, JPEG Bit-mapped image,
| Microsoft Office, Plain text etc.).
|
+-- "README5-Notebooks.txt"
|
+-- ["SourceDistributions"]
|
| Contains a collection of computer program
| source code files that the Toolkit recip-
| ient will need to install, operate, modify
| and re-distribute the Toolkit.
|
+-- "README6-SourceDistributions.txt"
|
+-- ["Developer-Sandboxes"] (Pre-dates Site-Packages)
|
| A sandbox is a testing environment that iso-
| lates untested code changes and outright experi-
| mentation from the production environment or
| repository.
|
+-- ["tsWxGTUI_PyVx"]
|
| Contains one or more Python language gener-
| ation-specific releases each sharing the same
| programmer (API) and user (CLI & GUI) inter-
| faces, documents and manual pages.
|
+-- ["Documents"] (copy)
|
+-- ["ManPages"] (copy)
|
+-- ["Python-2x"]
|
| Second generation Python programming
| language.
|
+-- ["tsWxGTUI_Py2x"]
|
| +-- ["tsDemoArchive"]
|
| | +-- ["src"]
|
| | +-- "TermsAndConditions.txt"
|
| +-- ["tsLibCLI"]
|
| +-- ["tsApplicationPkg"]

```



```
["tsLinesOfCodeProjectMetricsPkg"]
```

```
|      +--- ["Site-Packages"]
|      |
|      |   A site-packages is the location where third-
|      |   party packages are installed (i.e., those
|      |   not part of the core Python distribution).
|      |   NOTE: That with Linux, Mac OS X and Unix
|      |   operating systems one must have root priv-
|      |   ileages to write to that location.
|      |
|      +--- ["tsWxGTUI_PyVx"]
|          |
|          +--- ["Documents"] (copy)
|          |
|          +--- ["ManPages"] (copy)
|          |
|          +--- ["Python-2x"]
|              |
|              |   Second generation Python programming
|              |   language.
|              |
|              +--- ["tsWxGTUI_Py2x"]
|                  |
|                  +--- ["tsDemoArchive"]
|                      |
|                      +--- ["tsTestsLibCLI"]
|                      +--- ["tsTestsLibGUI"]
|                      +--- ["tsTestsToolsCLI"]
|                      +--- ["tsTestsToolsGUI"]
|                      +--- ["tsTestsToolsLibCLI"]
|                      +--- ["tsTestsToolsLibGUI"]
|                  +--- ["tsLibCLI"]
|                  |
|                  +--- ["tsLibGUI"]
|                  |
|                  +--- ["tsToolsCLI"]
|                  |
|                  +--- ["tsToolsGUI"]
|                  |
|                  +--- ["tsUtilities"]
|
|      +--- ["Python-3x"] (Ported from Python-2x)
|          |
|          |   Third generation Python programming
|          |   language.
|          |
|          +--- ["tsWxGTUI_Py3x"]
|
+--- "README.txt"
```

===== TABLE OF CONTENTS =====

1. Source Code
2. Developer-Sandbox

- 2.1 tsLibCLI (equivalent to 3.1.1 tsTestsLibCLI)
- 2.2 tsLibGUI (equivalent to 3.1.2 tsTestsLibGUI)
- 2.3 tsToolsCLI (equivalent to 3.1.3 tsTestsToolsCLI
and 3.1.5 tsTestsToolsLibCLI)
- 2.4 tsToolsGUI (equivalent to 3.1.4 tsTestsToolsGUI)
- 2.5 tsUtilities (equivalent to 3.8 tsUtilities)

3. Site-Package

3.1 tsDemoArchive

- 3.1.1 tsTestsLibCLI
- 3.1.2 tsTestsLibGUI
- 3.1.3 tsTestsToolsCLI
- 3.1.4 tsTestsToolsGUI
- 3.1.5 tsTestsToolsLibCLI
- 3.1.6 tsTestsToolsLibGUI

- 3.2 tsLibCLI
- 3.3 tsLibGUI
- 3.4 tsToolsCLI
- 3.5 tsToolsGUI
- 3.6 tsToolsLibCLI
- 3.7 tsToolsLibGUI
- 3.8 tsUtilities

===== SourceDistribution =====

1. Source Code

Excerpt From Wikipedia, the free encyclopedia:

"In computing, source code is any collection of computer instructions (possibly with comments) written using some human-readable computer language, usually as text. The source code of a program is specially designed to facilitate the work of computer programmers, who specify the actions to be performed by a computer mostly by writing source code. The source code is often transformed by a compiler program into low-level machine code understood by the computer. The machine code might then be stored for execution at a later time. Alternatively, an interpreter can be used to analyze and perform the outcomes of the source code program directly on the fly.

Most computer applications are distributed in a form that includes executable files, but not their source code. If the source code were included, it would be useful to a user, programmer, or system administrator, who may wish to modify the program or to understand how it works.

Aside from its machine-readable forms, source code also appears in books and other media; often in the form of small code snippets, but occasionally complete code bases; a well-known case is the source code of PGP."

===== DEVELOPER-SANDBOX =====

2. Developer-Sandbox

Excerpt From Wikipedia, the free encyclopedia:

"A sandbox is a testing environment that isolates untested code changes and outright experimentation from the production environment or repository, in the context of software development including Web development and revision control. Sandboxing protects "live" servers and their data, vetted source code distributions, and other collections of code, data and/or content, proprietary or public, from changes that could be damaging (regardless of the intent of the author of those changes) to a mission-critical system or which could simply be difficult to revert. Sandboxes replicate at least the minimal functionality needed to accurately test the programs or other code under development (e.g. usage of the same environment variables as, or access to an identical database to that used by, the stable prior implementation intended to be modified; there are many other possibilities, as the specific functionality needs vary widely with the nature of the code and the application[s] for which it is intended.)

The concept of the sandbox (sometimes also called a working directory, a test server or development server) is typically built into revision control software such as CVS and Subversion (SVN), in which developers "check out" a copy of the source code tree, or a branch thereof, to examine and work on. Only after the developer has (hopefully) fully tested the code changes in their own sandbox should the changes be checked back into and merged with the repository and thereby made available to other developers or end users of the software.[1]

By further analogy, the term "sandbox" can also be applied in computing and networking to other temporary or indefinite isolation areas, such as security sandboxes and search engine sandboxes (both of which have highly specific meanings), that prevent incoming data from affecting a "live" system (or aspects thereof) unless/ until defined requirements or criteria have been met."

Unlike the contents of the installable site-package, this sandbox uses a multi-level tree of subdirecories and associated files whose topology is defined by a set of package "__init__.py" files which collaborate in performing dynamic path generation and importing of modules and subpackages. Applications import individual packages and individual modules simply by name (if module name is unique) or by package.module name (if module name is not unique).

Source Code Manifest Checklist:

```

2.1 tsLibCLI      (equivalent to 3.1.1 tsTestsLibCLI)
2.2 tsLibGUI      (equivalent to 3.1.2 tsTestsLibGUI)
2.3 tsToolsCLI    (equivalent to 3.1.3 tsTestsToolsCLI
                    and 3.1.5 tsTestsToolsLibCLI)
2.4 tsToolsGUI    (equivalent to 3.1.4 tsTestsToolsGUI)
2.5 tsUtilities   (equivalent to 3.8 tsUtilities)

```

```
===== SITE-PACKAGE =====
```

3. Site-Package

Site-packages is the location where third-party packages are installed (i.e., those not part of the core Python distribution). NOTE: That with Linux, Mac OS X and Unix operating systems one must have root privileges to write to that location.

Unlike the contents of the Developer-Sandbox, the third-party site-package and its users must explicitly import via the `site-package.package.module` path identifier.

```

<Your Working Repository>
(e.g. "Technical_Preview")
|
|   Working repository containing directories and
|   files to be packaged into downloadable "tarball"
|   and/or "zip" files via the setup shell scripts
|   at the bottom of this diagram.
|
+-- ["Documents"]
|   |
|   |   This directory contains a collection of files
|   |   which provide the Toolkit recipient with an
|   |   understanding of the purpose, goals & capabilities,
|   |   non-goals & limitations, terms & conditions and
|   |   procedures for installing, operating, modifying and
|   |   redistributing the Toolkit.
|   |
|   +-- "README3-Documents.txt"
|
+-- ["ManPages"]
|   |
|   |   Deliverable Toolkit manual pages are a
|   |   form of online software documentation
|   |   usually found on a Unix or Unix-like operating
|   |   system.
|   |
|   +-- "README4-ManPages.txt"
|
+-- ["Notebooks"]
|   |
|   |   Contains a collection of commentaries that
|   |   express opinions or offerings of explanations
|   |   about events or situations that might

```

```

|         | be useful to Toolkit installers, developers,
|         | operators, troubleshooters and distributors.
|         | The documents may be in Application-specific
|         | formats (Adobe PDF, JPEG Bit-mapped image,
|         | Microsoft Office, Plain text etc.).
|
|         +--- "README5-Notebooks.txt"
|
+--- ["SourceDistributions"]
|
|         | Contains a collection of computer program
|         | source code files that the Toolkit recip-
|         | ient will need to install, operate, modify
|         | and re-distribute the Toolkit.
|
|         +--- "README6-SourceDistributions.txt"
|
+--- ["Developer-Sandboxes"] (Pre-dates Site-Packages)
|
|         | A sandbox is a testing environment that iso-
|         | lates untested code changes and outright
|         | experimentation from the production environ-
|         | ment or repository.
|
|         +--- ["tsWxGTUI_PyVx"]
|         |
|         |         +--- ["Documents"] (copy)
|         |         |
|         |         +--- ["ManPages"] (copy)
|         |         |
|         |         +--- ["Python-2x"]
|         |         |
|         |         |         +--- ["tsWxGTUI_Py2x"]
|         |         |
|         |         +--- ["Python-3x"] (Ported from Python-2x)
|         |         |
|         |         +--- ["tsWxGTUI_Py3x"]
|
+--- ["Site-Packages"]
|
|         | Site-packages is the location where third-
|         | party packages are installed (i.e., those
|         | not part of the core Python distribution).
|         | NOTE: That with Linux, Mac OS X and Unix
|         | operating systems one must have root priv-
|         | ileages to write to that location.
|
|         +--- ["tsWxGTUI_PyVx"]
|         |
|         |         +--- ["Documents"] (copy)
|         |         |
|         |         +--- ["ManPages"] (copy)
|         |         |
|         |         +--- ["Python-2x"]
|         |         |
|         |         |         +--- ["tsWxGTUI_Py2x"]

```

```
+-- "README.txt"
```

3.1 tsDemoArchive

A collection of Python application programs for test (unit, integration, system, regression and acceptance) and demonstration of features, capabilities and limitations of the Command Line Interface (CLI) or Graphical-style User Interface (GUI) mode of operation.

3.1.1.1 tsTestsLibCLI

- 3.1.1.1.1 test_TermsAndConditions.py
- 3.1.1.1.2 test_tsApplication.py
- 3.1.1.1.3 test_tsCommandLineEnv.py
- 3.1.1.1.4 test_tsCommandLineInterface.py
- 3.1.1.1.5 test_tsCxGlobals.py
- 3.1.1.1.6 test_tsDoubleLinkedList.py
- 3.1.1.1.7 test_tsException.py
- 3.1.1.1.8 test_tsLogger.py
- 3.1.1.1.9 test_tsOperatorSettingsParser.py
- 3.1.1.1.10 test_tsPlatformRunTimeEnvironment.py
- 3.1.1.1.11 test_tsReportUtility.py
- 3.1.1.1.12 test_tsSysCommands.py

3.1.1.2 tsTestsLibGUI

- 3.1.2.1 test_cursesOverlappingPanels.py
- 3.1.2.2 test_cursesPanels.py
- 3.1.2.3 test_tsWxBoxSizer.py
- 3.1.2.4 test_tsWxCheckBox.py
- 3.1.2.5 test_tsWxColorPalette.py
- 3.1.2.6 test_tsWxDisplay.py
- 3.1.2.7 test_tsWxDoubleLinkedList.py
- 3.1.2.8 test_tsWxGlobals.py
- 3.1.2.9 test_tsWxGraphicalTextUserInterface.py
- 3.1.2.10 test_tsWxGridSizer.py
- 3.1.2.11 test_tsWxMultiFrameEnv.py
- 3.1.2.12 test_tsWxPasswordEntryDialog.py
- 3.1.2.13 test_tsWxRSM.py
- 3.1.2.14 test_tsWxScrolledWindow.py
- 3.1.2.15 test_tsWxScrolledWindowDual.py
- 3.1.2.16 test_tsWxSplashScreen.py
- 3.1.2.17 test_tsWxSystemSettings.py
- 3.1.2.17 test_tsWxTextEntryDialog.py
- 3.1.2.18 test_tsWxWidgets.py

3.1.1.3 tsTestsToolsCLI

- 3.1.3.1 ["basicFileTypes"]
- 3.1.3.2 ["regressionTestFileTypes"]
- 3.1.3.3 ["sampleHelp"]
- 3.1.3.4 "File_Extensions.txt"
- 3.1.3.5 tsLinesOfCodeProjectMetrics.py
- 3.1.3.6 tsPlatformQuery.py
- 3.1.3.7 tsReVersion.py
- 3.1.3.8 tsStripComments.py
- 3.1.3.9 tsStripLineNumbers.py
- 3.1.3.10 tsTreeCopy.py
- 3.1.3.11 tsTreeTrimLines.py

3.1.1.4 tsTestsToolsGUI

<None>

3.1.1.5 tsTestsToolsLibCLI


```

3.1.5.1 ["basicFileTypes"]
3.1.5.2 ["regressionTestFileTypes"]
3.1.3.3 "File_Extensions.txt"
3.1.5.4 test_tsLinesOfCode.py
3.1.5.5 test_tsStripComments.py
3.1.5.6 test_tsStripSettingsParser.py

```

3.1.6 tsTestsToolsLibGUI

<None>

3.2 tsLibCLI

A collection of building-block components for Python application programs used during the Command Line Interface mode of operation.

```

3.2.1 tsApplication.py
3.2.2 tsCommandLineEnv.py
3.2.3 tsCommandLineInterface.py
3.2.4 tsCxGlobals.py
3.2.5 tsDoubleLinkedList.py
3.2.6 tsException.py
3.2.7 tsGistGetTerminalSize.py
3.2.8 tsLogger.py
3.2.9 tsOperatorSettingsParser.py
3.2.10 tsPlatformRunTimeEnvironment.py
3.2.11 tsReportUtility.py
3.2.12 tsSysCommands.py

```

3.3 tsLibGUI

A collection of building-block components for Python application programs used during the Graphical-style User Interface mode of operation.

3.3.1 wxLibPython

A collection of building-blocks for the wxPython API Graphical-style User Interface mode of operation.

NOTE: "wxWidgets", used by software developers working in the C++ programming language, or an alternate programming language-specific wrapper (such as "wxPython") provides a pixel-mode cross-platform Application Programming Interface (API) which depends on the host platform's native pixel-mode GUI services (such as GNU/Linux, Mac OS X, Microsoft Windows or Unix).

```

3.3.1.1 tsWx.py
3.3.1.2 tsWxAcceleratorEntry.py
3.3.1.3 tsWxAcceleratorTable.py
3.3.1.4 tsWxApp.py

```

3.3.1.5 tsWxBoxSizer.py
3.3.1.6 tsWxButton.py
3.3.1.7 tsWxCallLater.py
3.3.1.8 tsWxCaret.py
3.3.1.9 tsWxCheckBox.py
3.3.1.10 tsWxChoice.py
3.3.1.11 tsWxColor.py
3.3.1.12 tsWxColorDatabase.py
3.3.1.13 tsWxControl.py
3.3.1.14 tsWxControlWithItems.py
3.3.1.15 tsWxCursor.py
3.3.1.16 tsWxDebugHandlers.py
3.3.1.17 tsWxDialog.py
3.3.1.18 tsWxDialogButton.py
3.3.1.19 tsWxDisplay.py
3.3.1.20 tsWxEraseEvent.py
3.3.1.21 tsWxEvent.py
3.3.1.22 tsWxEventDaemon.py
3.3.1.23 tsWxEventLoop.py
3.3.1.24 tsWxEventLoopActivator.py
3.3.1.25 tsWxEvtHandler.py
3.3.1.26 tsWxFlexGridSizer.py
3.3.1.27 tsWxFocusEvent.py
3.3.1.28 tsWxFrame.py
3.3.1.29 tsWxFrameButton.py
3.3.1.30 tsWxGauge.py
3.3.1.31 tsWxGridBagSizer.py
3.3.1.32 tsWxGridSizer.py
3.3.1.33 tsWxItemContainer.py
3.3.1.34 tsWxKeyEvent.py
3.3.1.35 tsWxListBox.py
3.3.1.36 tsWxLog.py
3.3.1.37 tsWxMenu.py
3.3.1.38 tsWxMenuBar.py
3.3.1.39 tsWxMouseEvent.py
3.3.1.40 tsWxMouseState.py
3.3.1.41 tsWxObject.py
3.3.1.42 tsWxPanel.py
3.3.1.43 tsWxPasswordEntryDialog.py
3.3.1.44 tsWxPoint.py
3.3.1.45 tsWxPyApp.py
3.3.1.46 tsWxPyEventBinder.py
3.3.1.47 tsWxPyOnDemandOutputWindow.py
3.3.1.48 tsWxPySimpleApp.py
3.3.1.49 tsWxPySizer.py
3.3.1.50 tsWxRadioBox.py
3.3.1.51 tsWxRadioButton.py
3.3.1.52 tsWxRect.py
3.3.1.53 tsWxScreen.py
3.3.1.54 tsWxScrollBar.py
3.3.1.55 tsWxScrolled.py
3.3.1.56 tsWxScrolledText.py
3.3.1.57 tsWxScrolledWindow.py
3.3.1.58 tsWxShowEvent.py
3.3.1.59 tsWxSize.py
3.3.1.60 tsWxSizer.py

- 3.3.1.61 tsWxSizerFlags.py
- 3.3.1.62 tsWxSizerItem.py
- 3.3.1.63 tsWxSizerItemList.py
- 3.3.1.64 tsWxSlider.py
- 3.3.1.65 tsWxSplashScreen.py
- 3.3.1.66 tsWxStaticBox.py
- 3.3.1.67 tsWxStaticBoxSizer.py
- 3.3.1.68 tsWxStaticLine.py
- 3.3.1.69 tsWxStaticText.py
- 3.3.1.70 tsWxStatusBar.py
- 3.3.1.71 tsWxSystemSettings.py
- 3.3.1.72 tsWxTextCtrl.py
- 3.3.1.73 tsWxTextEditBox.py
- 3.3.1.74 tsWxTextEntry.py
- 3.3.1.75 tsWxTextEntryDialog.py
- 3.3.1.76 tsWxTimer.py
- 3.3.1.77 tsWxToggleButton.py
- 3.3.1.78 tsWxTopLevelWindow.py
- 3.3.1.79 tsWxValidator.py
- 3.3.1.80 tsWxWindow.py

3.3.2 wxLibCurses

A collection of building-blocks for the Curses API Graphical-style User Interface mode of operation.

NOTE: These building blocks provide a cross-platform character-mode emulation of the host platform's native pixel-mode GUI services (such as GNU/Linux, Mac OS X, Microsoft Windows or Unix):

- a) converting between pixel- and character-mode dimensions;
- b) detects and/or sets up the color palette and foreground/background color pair combinations;
- c) handles input from terminal keyboard and mouse;
- d) formats and outputs control (position, size, font, color) and data (borders, lines, labels and messages) to the terminal display.
- e) determines which pixel-mode GUI object triggered the mouse click; and which one should receive the event notification.
- f) manages the terminal display layout to provide an application desktop area for frames and dialogs; an operator notification area for date and time stamped event messages; and a task bar area.

- 3.3.2.1 tsWxCursesKeyCodesDataBase.py
- 3.3.2.2 tsWxCursesMouseButtonCodesDataBase.py
- 3.3.2.3 tsWxCursesServices.py

- 3.3.2.4 tsWxDoubleLinkedList.py
- 3.3.2.5 tsWxEventQueueEntry.py
- 3.3.2.6 tsWxEventTableEntry.py
- 3.3.2.7 tsWxGlobals.py
- 3.3.2.8 tsWxGraphicalTextUserInterface.py
- 3.3.2.9 tsWxKeyboardState.py
- 3.3.2.10 tsWxMultiFrameEnv.py
- 3.3.2.11 tsWxNonLinkedList.py
- 3.3.2.12 tsWxPythonColor16DataBase.py
- 3.3.2.13 tsWxPythonColor16SubstitutionMap.py
- 3.3.2.14 tsWxPythonColor256DataBase.py
- 3.3.2.15 tsWxPythonColor88DataBase.py
- 3.3.2.16 tsWxPythonColor8DataBase.py
- 3.3.2.17 tsWxPythonColor8SubstitutionMap.py
- 3.3.2.18 tsWxPythonColorDataBaseRGB.py
- 3.3.2.19 tsWxPythonColorNames.py
- 3.3.2.20 tsWxPythonColorRGBNames.py
- 3.3.2.21 tsWxPythonColorRGBValues.py
- 3.3.2.22 tsWxPythonMonochromeDataBase.py
- 3.3.2.23 tsWxPythonPrivateLogger.py
- 3.3.2.24 tsWxScrollBarButton.py
- 3.3.2.25 tsWxScrollBarGauge.py
- 3.3.2.26 tsWxSizerSpacer.py
- 3.3.2.27 tsWxTaskBar.py
- 3.3.2.28 tsWxWindowCurses.py

3.4 tsToolsCLI

A collection of Python application programs for the Command Line Interface mode of operation.

- 3.4.1 tsChange_tsWxGTUI_PyVx.py
- 3.4.2 tsChange_wxPython_API_Version.py
- 3.4.3 tsLinesOfCodeProjectMetrics.py
- 3.4.4 tsPlatformQuery.py
- 3.4.5 tsStripComments.py
- 3.4.6 tsStripLineNumbers.py
- 3.4.7 tsTreeCopy.py
- 3.4.8 tsTreeTrimLines.py

3.5 tsToolsGUI

A collection of Python application programs for the Graphical-style User Interface mode of operation.

<none>

3.6 tsToolsLibCLI

A collection of building-block components for Python application program tools used during the Command Line Interface mode of operation.

3.7 tsToolsLibGUI

A collection of building-block components for Python

application program tools used during the Graphical-style User Interface mode of operation.

<none>

3.8 tsUtilities

A collection of operating system shell scripts and Python scripts used for hardware and software administration during the Command Line Interface mode of operation.

===== End-Of-File =====

Draft

28.9 README8-SitePackages.txt

```
#-----
#"Time-stamp: <06/04/2015  9:24:01 PM rsg>"
#-----

===== Title Page for File: README8-Site-Packages.txt =====

+-----+-----+ TeamSTARS "tsWxGTUI_PyVx" Toolkit
| ts | Wx |      with Python 2x & Python 3x based
+-----+-----+      Command Line Interface (CLI)
| G T U I |      and "Curses"-based "wxPython"-style,
+-----+-----+      Graphical-Text User Interface (GUI)

Get that cross-platform, pixel-mode "wxPython" feeling
on character-mode 8-/16-color (xterm-family) & non-color
(vt100-family) terminals and terminal emulators.

You can find this and other files in the following sub-
directories:

<Your Working Repository>
(e.g. "Technical_Preview")
|
|   Working repository containing directories and
|   files to be packaged into downloadable "tarball"
|   and/or "zip" files via the setup shell scripts
|   at the bottom of this diagram.
|
+-- ["Documents"]
|
|   |   This directory contains a collection of files
|   |   which provide the Toolkit recipient with an
|   |   understanding of the purpose, goals & capabil-
|   |   ities, non-goals & limitations, terms & condi-
|   |   tions and procedures for installing, operating,
|   |   modifying and redistributing the Toolkit.
|   |
|   |   Deliverable Toolkit manual pages are a
|   |   form of online software documentation
|   |   usually found on a Unix or Unix-like oper-
|   |   ating system.
|   |
+-- "README3-Documents.txt"
|
+-- ["ManPages"]
|
|   |   Deliverable Toolkit manual pages are a
|   |   form of online software documentation
|   |   usually found on a Unix or Unix-like oper-
|   |   ating system.
|   |
+-- "README4-ManPages.txt"
```

```

|
+-- ["Notebooks"]
|
|   Contains a collection of commentaries that
|   express opinions or offerings of explana-
|   tions about events or situations that might
|   be useful to Toolkit installers, developers,
|   operators, troubleshooters and distributors.
|   The documents may be in Application-specific
|   formats (Adobe PDF, JPEG Bit-mapped image,
|   Microsoft Office, Plain text etc.).
|
+-- "README5-Notebooks.txt"
|
+-- ["SourceDistributions"]
|
|   Contains a collection of computer program
|   source code files that the Toolkit recip-
|   ient will need to install, operate, modify
|   and re-distribute the Toolkit.
|
+-- "README6-SourceDistributions.txt"
|
+-- ["Developer-Sandboxes"] (Pre-dates Site-Packages)
|
|   A sandbox is a testing environment that iso-
|   lates untested code changes and outright experi-
|   mentation from the production environment or
|   repository.
|
+-- ["tsWxGTUI_PyVx"]
|
|   Contains one or more Python language gener-
|   ation-specific releases each sharing the same
|   programmer (API) and user (CLI & GUI) inter-
|   faces, documents and manual pages.
|
+-- ["Documents"]
|
+-- ["ManPages"]
|
+-- ["Python-2x"]
|
|   Second generation Python programming
|   language.
|
+-- ["tsWxGTUI_Py2x"]
|
|   +-- ["tsDemoArchive"]
|   |
|   |   +-- ["src"]
|   |   |
|   |   +-- "TermsAndConditions.txt"
|   |
+-- ["tsLibCLI"]

```


Third generation Python programming language.

```

|                                     +--- ["tsWxGTUI_Py3x"]
|                                     |
|                                     +--- ["Site-Packages"]
|                                     |
|                                     |   A site-packages is the location where third-
|                                     |   party packages are installed (i.e., those
|                                     |   not part of the core Python distribution).
|                                     |   NOTE: That with Linux, Mac OS X and Unix
|                                     |   operating systems one must have root priv-
|                                     |   ileages to write to that location.
|                                     |
|                                     +--- ["tsWxGTUI_PyVx"]
|                                     |
|                                     |   +--- ["Documents"]
|                                     |   |
|                                     |   +--- ["ManPages"]
|                                     |   |
|                                     |   +--- ["Python-2x"]
|                                     |   |
|                                     |   |   Second generation Python programming
|                                     |   |   language.
|                                     |   |
|                                     |   |   +--- ["tsWxGTUI_Py2x"]
|                                     |   |   |
|                                     |   |   |   +--- ["tsDemoArchive"]
|                                     |   |   |   |
|                                     |   |   |   |   +--- ["tsTestsLibCLI"]
|                                     |   |   |   |   +--- ["tsTestsLibGUI"]
|                                     |   |   |   |   +--- ["tsTestsToolsCLI"]
|                                     |   |   |   |   +--- ["tsTestsToolsGUI"]
|                                     |   |   |   |   +--- ["tsTestsToolsLibCLI"]
|                                     |   |   |   |   +--- ["tsTestsToolsLibGUI"]
|                                     |   |   |   |
|                                     |   |   |   +--- ["tsLibCLI"]
|                                     |   |   |   |
|                                     |   |   |   +--- ["tsLibGUI"]
|                                     |   |   |   |
|                                     |   |   |   +--- ["tsToolsCLI"]
|                                     |   |   |   |
|                                     |   |   |   +--- ["tsToolsGUI"]
|                                     |   |   |   |
|                                     |   |   |   +--- ["tsUtilities"]
|                                     |   |   |
|                                     |   |   +--- ["Python-3x"] (Ported from Python-2x)
|                                     |   |   |
|                                     |   |   |   Third generation Python programming
|                                     |   |   |   language.
|                                     |   |   |
|                                     |   |   +--- ["tsWxGTUI_Py3x"]
|                                     |
|                                     +--- "README.txt"

```

===== TABLE OF CONTENTS =====

1. Source Code

2. Developer-Sandbox

3. Site-Package

===== SourceDistribution =====

1. Source Code

Excerpt From Wikipedia, the free encyclopedia:

"In computing, source code is any collection of computer instructions (possibly with comments) written using some human-readable computer language, usually as text. The source code of a program is specially designed to facilitate the work of computer programmers, who specify the actions to be performed by a computer mostly by writing source code. The source code is often transformed by a compiler program into low-level machine code understood by the computer. The machine code might then be stored for execution at a later time. Alternatively, an interpreter can be used to analyze and perform the outcomes of the source code program directly on the fly.

Most computer applications are distributed in a form that includes executable files, but not their source code. If the source code were included, it would be useful to a user, programmer, or system administrator, who may wish to modify the program or to understand how it works.

Aside from its machine-readable forms, source code also appears in books and other media; often in the form of small code snippets, but occasionally complete code bases; a well-known case is the source code of PGP."

===== DEVELOPER-SANDBOX =====

2. Developer-Sandbox

Excerpt From Wikipedia, the free encyclopedia:

"A sandbox is a testing environment that isolates untested code changes and outright experimentation from the production environment or repository, in the context of software development including Web development and revision control. Sandboxing protects "live" servers and their data, vetted source code distributions, and other collections of code, data and/or content, proprietary or public, from changes that could be damaging (regardless of the intent of the author of those changes) to a mission-critical system or which could simply be difficult to revert. Sandboxes replicate at least the minimal functionality needed to accurately test the programs or other code under development (e.g. usage of the same environment variables as, or access to an identical database to that used by, the stable prior implementa-

tion intended to be modified; there are many other possibilities, as the specific functionality needs vary widely with the nature of the code and the application[s] for which it is intended.)

The concept of the sandbox (sometimes also called a working directory, a test server or development server) is typically built into revision control software such as CVS and Subversion (SVN), in which developers "check out" a copy of the source code tree, or a branch thereof, to examine and work on. Only after the developer has (hopefully) fully tested the code changes in their own sandbox should the changes be checked back into and merged with the repository and thereby made available to other developers or end users of the software.[1]

By further analogy, the term "sandbox" can also be applied in computing and networking to other temporary or indefinite isolation areas, such as security sandboxes and search engine sandboxes (both of which have highly specific meanings), that prevent incoming data from affecting a "live" system (or aspects thereof) unless/until defined requirements or criteria have been met."

Unlike the contents of the installable site-package, this sandbox uses a multi-level tree of subdirectories and associated files whose topology is defined by a set of package "__init__.py" files which collaborate in performing dynamic path generation and importing of modules and subpackages. Applications import individual packages and individual modules simply by name (if module name is unique) or by package.module name (if module name is not unique).

===== SITE-PACKAGE =====

3. Site-Package

Site-packages is the location where third-party packages are installed (i.e., those not part of the core Python distribution). NOTE: That with Linux, Mac OS X and Unix operating systems one must have root privileges to write to that location.

Unlike the contents of the Developer-Sandbox, the third-party site-package and its users must explicitly import via the site-package.package.module path identifier.

===== End-Of-File =====

29 TECHNICAL IMPACT ANALYSIS

29.1 Performance

29.2 Hardware Requirements

Development and testing will require the following hardware:

- TBD

29.3 Software Requirements

Development and testing will require the following software:

- TBD

29.4 Technical Risks

The technical risks include the following:

- Availability of applicable Hardware Requirements for software development.
- Availability of applicable Software Requirements for software development.
- Diversion of Software Engineers to unscheduled, higher priority activities

29.5 Documentation Requirements

The Software Engineering team, will generate the following documentation:

- System Administrator's Guide - Describes the preparation configuration, installation, diagnostics and troubleshooting of the Graphical Text User Interface Toolkit.
- Program's Reference Manual - Describes the design, building and usage of the Application Programming Interface.
- Operator's Guide - Describes the preparation, startup, operation and shutdown of GUI Applications.

Draft

30 TEST REQUIREMENTS AND RESTRICTIONS

30.1 Test Requirements

The test requirements include the following:

REQUIREMENT	DESCRIPTION
Unit Test	Testing will be performed to exercise each API and each Test Unit (diagnostic test).
Negative Unit Test	Testing will be required to verify that each API returns proper error messages.
System Test	Testing will perform tests on representative Use Cases. Testing will be evaluate suitability for the appropriate engineering, manufacturing and customer users.

30.2 Test Restrictions

The test restrictions include the following:

RESTRICTION	DESCRIPTION
Stand-Alone Mode with Command Line Interface	Testing will be performed via the shell commands available on the system console.
Stand-Among Mode with Command Line Interface	Testing will be performed via the shell commands available on the system console.
Stand-Among Mode with Graphical User Interface	Testing will be performed via the menu selections available on the system console.

Draft

31 USE CASES

Draft

Draft

32 TRACEABILITY INFORMATION

This section is under development.

Draft

Draft

33 CLASS LIST BY CATEGORY

This page contains a summarized listing of wxWidgets/wxPython classes, please see the *Full Class List by Category* (http://docs.wxwidgets.org/trunk/group__group__class.html) page for a full listing.

Draft

<ul style="list-style-type: none"> ▪ Basic Windows (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ Window Layout (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ Managed Windows (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ Menus (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ Controls (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ Validators (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ Picker Controls (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ Window Docking (wxAUI) (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ Common Dialogs 	<ul style="list-style-type: none"> ▪ Device Contexts (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ Graphics Device Interface (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ Graphics Context classes (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ Image and bitmap classes (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ Events (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ Application and Process Management (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ Printing Framework (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ Document/View Framework (http://docs.wxwidgets.org/trunk/page_class_cat.html) 	<ul style="list-style-type: none"> ▪ Runtime Type Information (RTTI) (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ Debugging (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ Logging (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ Data Structures (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ Text Conversion (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ Containers (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ Smart Pointers (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ File Handling (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ Streams (http://docs.wxwidgets.org/trunk/page_class_cat.html) 	<ul style="list-style-type: none"> ▪ Networking (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ Archive (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ Interprocess Communication (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ Help (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ Multimedia (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ OpenGL (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ Miscellaneous (http://docs.wxwidgets.org/trunk/page_class_cat.html)
--	---	--	---

<p>(http://docs.wxwidgets.org/trunk/page_class_cat.html)</p> <ul style="list-style-type: none"> ▪ HTML (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ Miscellaneous Windows 	<p><i>nk/page_class_cat.html</i>)</p> <ul style="list-style-type: none"> ▪ Clipboard and Drag & Drop (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ Virtual File System (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ Threading (http://docs.wxwidgets.org/trunk/page_class_cat.html) 	<p><i>nk/page_class_cat.html</i>)</p> <ul style="list-style-type: none"> ▪ XML (http://docs.wxwidgets.org/trunk/page_class_cat.html) ▪ XML Based Resource System (XRC) (http://docs.wxwidgets.org/trunk/page_class_cat.html) 	
--	--	--	--

33.1 Basic Windows

The following are the most important window classes:

- ***wxWindow*** (http://docs.wxwidgets.org/trunk/classwx_window.html): base class for all windows and controls
- ***wxControl*** (http://docs.wxwidgets.org/trunk/classwx_control.html): base class (mostly) for native controls/widgets
- ***wxPanel*** (http://docs.wxwidgets.org/trunk/classwx_panel.html): window which can smartly manage child windows
- ***wxScrolledWindow*** (http://docs.wxwidgets.org/trunk/group_group_class_miscwnd.html): Window with automatically managed scrollbars (see ***wxScrolled***)
- ***wxTopLevelWindow*** (http://docs.wxwidgets.org/trunk/classwx_top_level_window.html): Any top level window, dialog or frame

33.2 Window Layout

There are two different systems for laying out windows (and dialogs in particular). One is based upon so-called sizers and it requires less typing, thinking and calculating and will in almost all cases produce dialogs looking equally well on all platforms, the other is based on so-called constraints and is deprecated, though still available.

33.2.1 Related Overviews: HLINK_46

These are the classes relevant to sizer-based layout:

- ***wxSizer*** (http://docs.wxwidgets.org/trunk/classwx_sizer.html): Abstract base class
- ***wxBoxSizer*** (http://docs.wxwidgets.org/trunk/classwx_box_sizer.html): A sizer for laying out windows in a row or column
- ***wxGridSizer*** (http://docs.wxwidgets.org/trunk/classwx_grid_sizer.html): A sizer for laying out windows in a grid with all fields having the same size
- ***wxFlexGridSizer*** (http://docs.wxwidgets.org/trunk/classwx_flex_grid_sizer.html): A sizer for laying out windows in a flexible grid
- ***wxGridBagSizer*** (http://docs.wxwidgets.org/trunk/classwx_grid_bag_sizer.html): Another grid sizer that lets you specify the cell an item is in, and items can span rows and/or columns.
- ***wxStaticBoxSizer*** (http://docs.wxwidgets.org/trunk/classwx_static_box_sizer.html): Same as ***wxBoxSizer*** (http://docs.wxwidgets.org/trunk/classwx_box_sizer.html), but with a surrounding static box
- ***wxWrapSizer*** (http://docs.wxwidgets.org/trunk/classwx_wrap_sizer.html): A sizer which wraps its child controls as size permits

Other layout classes:

- ***wxLayoutAlgorithm*** (http://docs.wxwidgets.org/trunk/classwx_layout_algorithm.html): An alternative window layout facility

33.3 Managed Windows

There are several types of window that are directly controlled by the window manager (such as MS Windows, or the Motif Window Manager). Frames and dialogs are similar in wxWidgets, but only dialogs may be modal.

33.3.1 Related Overviews: HLINK_56

- ***wxDialog*** (http://docs.wxwidgets.org/trunk/classwx_dialog.html): Dialog box
- ***wxFrame*** (http://docs.wxwidgets.org/trunk/classwx_frame.html): Normal frame
- ***wxMDIChildFrame*** (http://docs.wxwidgets.org/trunk/classwx_m_d_i_child_frame.html): MDI child frame
- ***wxMDIParentFrame*** (http://docs.wxwidgets.org/trunk/classwx_m_d_i_parent_frame.html): MDI parent frame
- ***wxMiniFrame*** (http://docs.wxwidgets.org/trunk/classwx_mini_frame.html): A frame with a small title bar
- ***wxPopupWindow*** (http://docs.wxwidgets.org/trunk/classwx_popup_window.html): A toplevel window without decorations, e.g. for a combobox pop-up
- ***wxPropertySheetDialog*** (http://docs.wxwidgets.org/trunk/classwx_property_sheet_dialog.html): Property sheet dialog
- ***wxSplashScreen*** (http://docs.wxwidgets.org/trunk/classwx_splash_screen.html): Splash screen class
- ***wxTipWindow*** (http://docs.wxwidgets.org/trunk/classwx_tip_window.html): Shows text in a small window
- ***wxWizard*** (http://docs.wxwidgets.org/trunk/classwx_wizard.html): A wizard dialog

33.4 Menus

- ***wxMenu*** (http://docs.wxwidgets.org/trunk/classwx_menu.html): Displays a series of menu items for selection
- ***wxMenuBar***: Contains a series of menus for use with a frame
- ***wxMenuItem*** (http://docs.wxwidgets.org/trunk/classwx_menu_item.html): Represents a single menu item

33.5 Controls

Typically, these are small windows which provide interaction with the user. Controls that are not static can have *wxValidator* (http://docs.wxwidgets.org/trunk/classwx_validator.html) associated with them.

- *wxAnimationCtrl* (http://docs.wxwidgets.org/trunk/classwx_animation_ctrl.html): A control to display an animation
- *wxControl* (http://docs.wxwidgets.org/trunk/classwx_control.html): The base class for controls
- *wxBitmapButton* (http://docs.wxwidgets.org/trunk/classwx_bitmap_button.html): Push button control, displaying a bitmap
- *wxBitmapComboBox* (http://docs.wxwidgets.org/trunk/classwx_bitmap_combo_box.html): A combobox with bitmaps next to text items
- *wxBitmapToggleButton* (http://docs.wxwidgets.org/trunk/classwx_bitmap_toggle_button.html): A toggle button with bitmaps.
- *wxButton* (http://docs.wxwidgets.org/trunk/classwx_button.html): Push button control, displaying text
- *wxCalendarCtrl* (http://docs.wxwidgets.org/trunk/classwx_calendar_ctrl.html): Control showing an entire calendar month
- *wxCheckBox* (http://docs.wxwidgets.org/trunk/classwx_check_box.html): Checkbox control
- *wxCheckListBox* (http://docs.wxwidgets.org/trunk/classwx_check_list_box.html): A listbox with a checkbox to the left of each item
- *wxChoice* (http://docs.wxwidgets.org/trunk/classwx_choice.html): Choice control (a combobox without the editable area)
- *wxCollapsiblePane* (http://docs.wxwidgets.org/trunk/classwx_collapsible_pane.html): A panel which can be shown/hidden by the user
- *wxComboBox* (http://docs.wxwidgets.org/trunk/classwx_combo_box.html): A choice with an editable area
- *wxComboCtrl* (http://docs.wxwidgets.org/trunk/classwx_combo_ctrl.html): A combobox with application defined popup
- *wxDataViewCtrl* (http://docs.wxwidgets.org/trunk/classwx_data_view_ctrl.html): A control to display tabular or tree like data
- *wxDataViewTreeCtrl* (http://docs.wxwidgets.org/trunk/classwx_data_view_tree_ctrl.html): A specialized *wxDataViewCtrl* (http://docs.wxwidgets.org/trunk/classwx_data_view_ctrl.html) with a *wxTreeCtrl*-like API
- *wxDataViewListCtrl* (http://docs.wxwidgets.org/trunk/classwx_data_view_list_ctrl.html): A specialized *wxDataViewCtrl* (http://docs.wxwidgets.org/trunk/classwx_data_view_ctrl.html) for displaying and editing simple tables.
- *wxEditableListBox* (http://docs.wxwidgets.org/trunk/classwx_editable_list_box.html): A listbox with editable items.
- *wxFileCtrl* (http://docs.wxwidgets.org/trunk/classwx_file_ctrl.html): A control for selecting a file. Useful for custom file dialogs.

- ***wxGauge*** (http://docs.wxwidgets.org/trunk/classwx_gauge.html): A control to represent a varying quantity, such as time remaining
- ***wxGenericDirCtrl*** (http://docs.wxwidgets.org/trunk/classwx_generic_dir_ctrl.html): A control for displaying a directory tree
- ***wxGrid*** (http://docs.wxwidgets.org/trunk/classwx_grid.html): A control to display spread-sheet like data in tabular form
- ***wxHeaderCtrl*** (http://docs.wxwidgets.org/trunk/classwx_header_ctrl.html): a small control to display the top header of tabular data
- ***wxHtmlListBox*** (http://docs.wxwidgets.org/trunk/classwx_html_list_box.html): An abstract class for creating listboxes showing HTML content
- ***wxHyperlinkCtrl*** (http://docs.wxwidgets.org/trunk/classwx_hyperlink_ctrl.html): A static text which opens an URL when clicked
- ***wxListBox*** (http://docs.wxwidgets.org/trunk/classwx_list_box.html): A list of strings for single or multiple selection
- ***wxListCtrl*** (http://docs.wxwidgets.org/trunk/classwx_list_ctrl.html): A control for displaying lists of strings and/or icons, plus a multicolumn report view
- ***wxListView*** (http://docs.wxwidgets.org/trunk/classwx_list_view.html): A simpler interface (façade) for ***wxListCtrl*** (http://docs.wxwidgets.org/trunk/classwx_list_ctrl.html) in report mode
- ***wxNotebook*** (http://docs.wxwidgets.org/trunk/classwx_notebook.html): A notebook class
- ***wxOwnerDrawnComboBox*** (http://docs.wxwidgets.org/trunk/classwx_owner_drawn_combo_box.html): A combobox with owner-drawn list items
- ***wxPropertyGrid*** (http://docs.wxwidgets.org/trunk/classwx_property_grid.html): A complex control to display hierachical, editable information
- ***wxRadioBox*** (http://docs.wxwidgets.org/trunk/classwx_radio_box.html): A group of radio buttons
- ***wxRadioButton*** (http://docs.wxwidgets.org/trunk/classwx_radio_button.html): A round button to be used with others in a mutually exclusive way
- ***wxRearrangeCtrl*** (http://docs.wxwidgets.org/trunk/classwx_rearrange_ctrl.html): A control allowing the user to rearrange a list of items.
- ***wxRichTextCtrl*** (http://docs.wxwidgets.org/trunk/classwx_rich_text_ctrl.html): Generic rich text editing control
- ***wxSimpleHtmlListBox*** (http://docs.wxwidgets.org/trunk/classwx_simple_html_list_box.html): A listbox showing HTML content
- ***wxStaticBox*** (http://docs.wxwidgets.org/trunk/classwx_static_box.html): A static, or group box for visually grouping related controls
- ***wxScrollBar*** (http://docs.wxwidgets.org/trunk/classwx_scroll_bar.html): Scrollbar control
- ***wxSearchCtrl*** (http://docs.wxwidgets.org/trunk/classwx_search_ctrl.html): A text input control used to initiate a search
- ***wxSpinButton*** (http://docs.wxwidgets.org/trunk/classwx_spin_button.html): A spin or 'up-down' control
- ***wxSpinCtrl*** (http://docs.wxwidgets.org/trunk/classwx_spin_ctrl.html): A spin control - i.e. spin button and text control displaying an integer

- ***wxSpinCtrlDouble*** (http://docs.wxwidgets.org/trunk/classwx_spin_ctrl_double.html): A spin control - i.e. spin button and text control displaying a real number
- ***wxStaticText*** (http://docs.wxwidgets.org/trunk/classwx_static_text.html): One or more lines of non-editable text
- ***wxTextCtrl*** (http://docs.wxwidgets.org/trunk/classwx_text_ctrl.html): Single or multiline text editing control
- ***wxToggleButton*** (http://docs.wxwidgets.org/trunk/classwx_toggle_button.html): A button which stays pressed when clicked by user.
- ***wxTreeCtrl*** (http://docs.wxwidgets.org/trunk/classwx_tree_ctrl.html): Tree (hierarchy) control
- ***wxStaticBitmap*** (http://docs.wxwidgets.org/trunk/classwx_static_bitmap.html): A control to display a bitmap
- ***wxStyledTextCtrl*** (http://docs.wxwidgets.org/trunk/classwx_styled_text_ctrl.html): A wxWidgets implementation of the Scintilla source code editing component for plain text editing.
- ***wxSlider*** (http://docs.wxwidgets.org/trunk/classwx_slider.html): A slider that can be dragged by the user
- ***wxVListBox*** (http://docs.wxwidgets.org/trunk/classwx_v_list_box.html): A listbox supporting variable height rows

33.6 Validators

These are the window validators, used for filtering and validating user input.

33.6.1 Related Overviews: HLINK_122

- ***wxValidator*** (http://docs.wxwidgets.org/trunk/classwx_validator.html): Base validator class
- ***wxTextValidator*** (http://docs.wxwidgets.org/trunk/classwx_text_validator.html): Text control validator class
- ***wxGenericValidator*** (http://docs.wxwidgets.org/trunk/classwx_generic_validator.html): Generic control validator class

33.7 Picker Controls

A picker control is a control whose appearance and behaviour is highly platform-dependent.

- ***wxColourPickerCtrl*** (http://docs.wxwidgets.org/trunk/classwx_colour_picker_ctrl.html): A control which allows the user to choose a colour
- ***wxDirPickerCtrl*** (http://docs.wxwidgets.org/trunk/classwx_dir_picker_ctrl.html): A control which allows the user to choose a directory
- ***wxFilePickerCtrl*** (http://docs.wxwidgets.org/trunk/classwx_file_picker_ctrl.html): A control which allows the user to choose a file
- ***wxFontPickerCtrl*** (http://docs.wxwidgets.org/trunk/classwx_font_picker_ctrl.html): A control which allows the user to choose a font
- ***wxDDatePickerCtrl*** (http://docs.wxwidgets.org/trunk/classwx_date_picker_ctrl.html): Small date picker control

33.8 Miscellaneous Windows

The following are a variety of classes that are derived from `wxWindow` (http://docs.wxwidgets.org/trunk/classwx_window.html).

- **`wxCollapsiblePane`** (http://docs.wxwidgets.org/trunk/classwx_collapsible_pane.html): A panel which can be shown/hidden by the user
- **`wxPanel`** (http://docs.wxwidgets.org/trunk/classwx_panel.html): A window whose colour changes according to current user settings
- **`wxScrolledWindow`** (http://docs.wxwidgets.org/trunk/group_group_class_miscwnd.html): Window with automatically managed scrollbars (see **`wxScrolled`** (http://docs.wxwidgets.org/trunk/classwx_scrolled.html))
- **`wxHScrolledWindow`** (http://docs.wxwidgets.org/trunk/classwx_h_scrolled_window.html): As **`wxScrolledWindow`** (http://docs.wxwidgets.org/trunk/group_group_class_miscwnd.html) but supports columns of variable widths
- **`wxVScrolledWindow`** (http://docs.wxwidgets.org/trunk/classwx_v_scrolled_window.html): As **`wxScrolledWindow`** (http://docs.wxwidgets.org/trunk/group_group_class_miscwnd.html) but supports rows of variable heights
- **`wxHVScrolledWindow`** (http://docs.wxwidgets.org/trunk/classwx_h_v_scrolled_window.html): As **`wxScrolledWindow`** (http://docs.wxwidgets.org/trunk/group_group_class_miscwnd.html) but supports scroll units of variable sizes.
- **`wxGrid`** (http://docs.wxwidgets.org/trunk/classwx_grid.html): A grid (table) window
- **`wxInfoBar`** (http://docs.wxwidgets.org/trunk/classwx_info_bar.html): An information bar usually shown on top of the main window.
- **`wxSplitterWindow`** (http://docs.wxwidgets.org/trunk/classwx_splitter_window.html): Window which can be split vertically or horizontally
- **`wxStatusBar`** (http://docs.wxwidgets.org/trunk/classwx_status_bar.html): Implements the status bar on a frame
- **`wxToolBar`** (http://docs.wxwidgets.org/trunk/classwx_tool_bar.html): Toolbar class
- **`wxNotebook`** (http://docs.wxwidgets.org/trunk/classwx_notebook.html): Notebook class
- **`wxListbook`** (http://docs.wxwidgets.org/trunk/classwx_listbook.html): Similar to notebook but using list control
- **`wxChoicebook`** (http://docs.wxwidgets.org/trunk/classwx_choicebook.html): Similar to notebook but using choice control
- **`wxTreebook`** (http://docs.wxwidgets.org/trunk/classwx_treebook.html): Similar to notebook but using tree control
- **`wxSashWindow`** (http://docs.wxwidgets.org/trunk/classwx_sash_window.html): Window with four optional sashes that can be dragged
- **`wxSashLayoutWindow`** (http://docs.wxwidgets.org/trunk/classwx_sash_layout_window.html): Window that can be involved in an IDE-like layout arrangement

- ***wxWizardPage*** (http://docs.wxwidgets.org/trunk/classwx_wizard_page.html): A base class for the page in wizard dialog.
- ***wxWizardPageSimple*** (http://docs.wxwidgets.org/trunk/classwx_wizard_page_simple.html): A page in wizard dialog.

33.9 Window Docking (wxAUI)

wxAUI is a set classes for writing a customizable application interface with built-in docking, floatable panes and a flexible MDI-like interface.

33.9.1 Related Overviews: HLINK_155

- ***wxAuiManager*** (http://docs.wxwidgets.org/trunk/classwx_aui_manager.html): The central class for managing the interface
- ***wxAuiNotebook*** (http://docs.wxwidgets.org/trunk/classwx_aui_notebook.html): A replacement notebook class with extra features
- ***wxAuiPaneInfo*** (http://docs.wxwidgets.org/trunk/classwx_aui_pane_info.html): Describes a single pane
- ***wxAuiDockArt*** (http://docs.wxwidgets.org/trunk/classwx_aui_dock_art.html): Art and metrics provider for customizing the docking user interface
- ***wxAuiTabArt***: Art and metrics provider for customizing the notebook user interface

33.10 Common Dialogs

Common dialogs are ready-made dialog classes which are frequently used in an application.

33.10.1 Related Overviews: HLINK_160

- ***wxDialog*** (http://docs.wxwidgets.org/trunk/classwx_dialog.html): Base class for common dialogs
- ***wxColourDialog*** (http://docs.wxwidgets.org/trunk/classwx_colour_dialog.html): Colour chooser dialog
- ***wxDirDialog*** (http://docs.wxwidgets.org/trunk/classwx_dir_dialog.html): Directory selector dialog
- ***wxFileDialog*** (http://docs.wxwidgets.org/trunk/classwx_file_dialog.html): File selector dialog
- ***wxFindReplaceDialog*** (http://docs.wxwidgets.org/trunk/classwx_find_replace_dialog.html): Text search/replace dialog
- ***wxFontDialog*** (http://docs.wxwidgets.org/trunk/classwx_font_dialog.html): Font chooser dialog
- ***wxMessageDialog*** (http://docs.wxwidgets.org/trunk/classwx_message_dialog.html): Simple message box dialog
- ***wxMultiChoiceDialog*** (http://docs.wxwidgets.org/trunk/classwx_multi_choice_dialog.html): Dialog to get one or more selections from a list
- ***wxPageSetupDialog*** (http://docs.wxwidgets.org/trunk/classwx_page_setup_dialog.html): Standard page setup dialog
- ***wxPasswordEntryDialog*** (http://docs.wxwidgets.org/trunk/classwx_password_entry_dialog.html): Dialog to get a password from the user
- ***wxPrintDialog*** (http://docs.wxwidgets.org/trunk/classwx_print_dialog.html): Standard print dialog
- ***wxProgressDialog*** (http://docs.wxwidgets.org/trunk/classwx_progress_dialog.html): Progress indication dialog
- ***wxRearrangeDialog*** (http://docs.wxwidgets.org/trunk/classwx_rearrange_dialog.html): Dialog allowing the user to rearrange a list of items.
- ***wxRichTextFormattingDialog*** (http://docs.wxwidgets.org/trunk/classwx_rich_text_formatting_dialog.html): A dialog for formatting the content of a ***wxRichTextCtrl*** (http://docs.wxwidgets.org/trunk/classwx_rich_text_ctrl.html)
- ***wxRichMessageDialog*** (http://docs.wxwidgets.org/trunk/classwx_rich_message_dialog.html): Nicer message box dialog
- ***wxSingleChoiceDialog*** (http://docs.wxwidgets.org/trunk/classwx_single_choice_dialog.html): Dialog to get a single selection from a list and return the string
- ***wxSymbolPickerDialog*** (http://docs.wxwidgets.org/trunk/classwx_symbol_picker_dialog.html): Symbol selector dialog
- ***wxTextEntryDialog*** (http://docs.wxwidgets.org/trunk/classwx_text_entry_dialog.html): Dialog to get a single line of text from the user
- ***wxWizard*** (http://docs.wxwidgets.org/trunk/classwx_wizard.html): A wizard dialog.

33.11 HTML

wxWidgets provides a set of classes to display text in HTML format. These classes include a help system based on the HTML widget.

- ***wxHtmlHelpController*** (http://docs.wxwidgets.org/trunk/classwx_html_help_controller.html): HTML help controller class
- ***wxHtmlWindow*** (http://docs.wxwidgets.org/trunk/classwx_html_window.html): HTML window class
- ***wxHtmlEasyPrinting*** (http://docs.wxwidgets.org/trunk/classwx_html_easy_printing.html): Simple class for printing HTML
- ***wxHtmlPrintout*** (http://docs.wxwidgets.org/trunk/classwx_html_printout.html): Generic HTML ***wxPrintout*** (http://docs.wxwidgets.org/trunk/classwx_printout.html) class
- ***wxHtmlParser*** (http://docs.wxwidgets.org/trunk/classwx_html_parser.html): Generic HTML parser class
- ***wxHtmlTagHandler*** (http://docs.wxwidgets.org/trunk/classwx_html_tag_handler.html): HTML tag handler, pluginable into ***wxHtmlParser*** (http://docs.wxwidgets.org/trunk/classwx_html_parser.html)
- ***wxHtmlWinParser*** (http://docs.wxwidgets.org/trunk/classwx_html_win_parser.html): HTML parser class for ***wxHtmlWindow*** (http://docs.wxwidgets.org/trunk/classwx_html_window.html)
- ***wxHtmlWinTagHandler*** (http://docs.wxwidgets.org/trunk/classwx_html_win_tag_handler.html): HTML tag handler, pluginable into ***wxHtmlWinParser*** (http://docs.wxwidgets.org/trunk/classwx_html_win_parser.html)

33.12 Device Contexts

Device contexts are surfaces that may be drawn on, and provide an abstraction that allows parameterisation of your drawing code by passing different device contexts.

33.12.1 Related Overviews: HLINK_193

- ***wxAutoBufferedPaintDC*** (http://docs.wxwidgets.org/trunk/classwx_auto_buffered_paint_d_c.html): A helper device context for double buffered drawing inside OnPaint().
- ***wxBufferedDC*** (http://docs.wxwidgets.org/trunk/classwx_buffered_d_c.html): A helper device context for double buffered drawing.
- ***wxBufferedPaintDC*** (http://docs.wxwidgets.org/trunk/classwx_buffered_paint_d_c.html): A helper device context for double buffered drawing inside OnPaint().
- ***wxClientDC*** (http://docs.wxwidgets.org/trunk/classwx_client_d_c.html): A device context to access the client area outside OnPaint() events
- ***wxPaintDC*** (http://docs.wxwidgets.org/trunk/classwx_paint_d_c.html): A device context to access the client area inside OnPaint() events
- ***wxWindowDC*** (http://docs.wxwidgets.org/trunk/classwx_window_d_c.html): A device context to access the non-client area
- ***wxScreenDC*** (http://docs.wxwidgets.org/trunk/classwx_screen_d_c.html): A device context to access the entire screen
- ***wxDC*** (http://docs.wxwidgets.org/trunk/classwx_d_c.html): The device context base class
- ***wxMemoryDC*** (http://docs.wxwidgets.org/trunk/classwx_memory_d_c.html): A device context for drawing into bitmaps
- ***wxMetafileDC*** (http://docs.wxwidgets.org/trunk/classwx_metafile_d_c.html): A device context for drawing into metafiles
- ***wxMirrorDC*** (http://docs.wxwidgets.org/trunk/classwx_mirror_d_c.html): A proxy device context allowing for simple mirroring.
- ***wxPostScriptDC*** (http://docs.wxwidgets.org/trunk/classwx_post_script_d_c.html): A device context for drawing into PostScript files
- ***wxPrinterDC*** (http://docs.wxwidgets.org/trunk/classwx_printer_d_c.html): A device context for drawing to printers

33.13 Graphics Context classes

These classes are related to drawing using a new vector based drawing API and are based on the modern drawing backend GDI+, CoreGraphics and Cairo.

- **wxGraphicsRenderer**: Represents a drawing engine.
- **wxGraphicsContext** (http://docs.wxwidgets.org/trunk/classwx_graphics_context.html): Represents a graphics context currently being drawn on.
- **wxGraphicsBrush** (http://docs.wxwidgets.org/trunk/classwx_graphics_brush.html): Brush for drawing into a **wxGraphicsContext** (http://docs.wxwidgets.org/trunk/classwx_graphics_context.html)
- **wxGraphicsPen** (http://docs.wxwidgets.org/trunk/classwx_graphics_pen.html): Pen for drawing into a **wxGraphicsContext** (http://docs.wxwidgets.org/trunk/classwx_graphics_context.html)
- **wxGraphicsFont** (http://docs.wxwidgets.org/trunk/classwx_graphics_font.html): Font for drawing text on a **wxGraphicsContext** (http://docs.wxwidgets.org/trunk/classwx_graphics_context.html)
- **wxGraphicsMatrix** (http://docs.wxwidgets.org/trunk/classwx_graphics_matrix.html): Represents an affine matrix for drawing transformation
- **wxGraphicsPath** (http://docs.wxwidgets.org/trunk/classwx_graphics_path.html): Represents a path for drawing

33.14 Graphics Device Interface

These classes are related to drawing on device contexts and windows.

- ***wxColour*** (http://docs.wxwidgets.org/trunk/classwx_colour.html): Represents the red, blue and green elements of a colour
- ***wxDCClipper*** (http://docs.wxwidgets.org/trunk/classwx_d_c_clipper.html): Wraps the operations of setting and destroying the clipping region
- ***wxBrush*** (http://docs.wxwidgets.org/trunk/classwx_brush.html): Used for filling areas on a device context
- ***wxBrushList*** (http://docs.wxwidgets.org/trunk/classwx_brush_list.html): The list of previously-created brushes
- ***wxFont*** (http://docs.wxwidgets.org/trunk/classwx_font.html): Represents fonts
- ***wxFontList*** (http://docs.wxwidgets.org/trunk/classwx_font_list.html): The list of previously-created fonts
- ***wxPen*** (http://docs.wxwidgets.org/trunk/classwx_pen.html): Used for drawing lines on a device context
- ***wxPenList*** (http://docs.wxwidgets.org/trunk/classwx_pen_list.html): The list of previously-created pens
- ***wxPalette*** (http://docs.wxwidgets.org/trunk/classwx_palette.html): Represents a table of indices into RGB values
- ***wxRegion*** (http://docs.wxwidgets.org/trunk/classwx_region.html): Represents a simple or complex region on a window or device context
- ***wxRendererNative*** (http://docs.wxwidgets.org/trunk/classwx_renderer_native.html): Abstracts high-level drawing primitives

33.15 Image and bitmap classes

These classes represent images and bitmap in various formats and ways to access and create them.

33.15.1 Related Overviews: HLINK_227

- *wxAnimation* (http://docs.wxwidgets.org/trunk/classwx_animation.html): Represents an animation
- *wxBitmap* (http://docs.wxwidgets.org/trunk/classwx_bitmap.html): Represents a platform dependent bitmap
- *wxBitmapHandler* (http://docs.wxwidgets.org/trunk/classwx_bitmap_handler.html): Class for loading a saving a *wxBitmap* (http://docs.wxwidgets.org/trunk/classwx_bitmap.html) in a specific format
- *wxCursor* (http://docs.wxwidgets.org/trunk/classwx_cursor.html): A small, transparent bitmap representing the cursor
- *wxIcon* (http://docs.wxwidgets.org/trunk/classwx_icon.html): A small, transparent bitmap for assigning to frames and drawing on device contexts
- *wxImage* (http://docs.wxwidgets.org/trunk/classwx_image.html): A platform-independent image class
- *wxImageHandler* (http://docs.wxwidgets.org/trunk/classwx_image_handler.html): Class for loading a saving a *wxImage* (http://docs.wxwidgets.org/trunk/classwx_image.html) in a specific format
- *wxImageList* (http://docs.wxwidgets.org/trunk/classwx_image_list.html): A list of images, used with some controls
- *wxMask* (http://docs.wxwidgets.org/trunk/classwx_mask.html): Represents a mask to be used with a bitmap for transparent drawing
- *wxMemoryDC* (http://docs.wxwidgets.org/trunk/classwx_memory_d_c.html): A device context for drawing into bitmaps
- *wxPixelData* (http://docs.wxwidgets.org/trunk/classwx_pixel_data.html): Class template for direct access to *wxBitmap*'s and *wxImage*'s internal data

33.16 Events

An event object contains information about a specific event. Event handlers (usually member functions) have a single, event argument.

33.16.1 Related Overviews: HLINK_241

- ***wxActivateEvent*** (http://docs.wxwidgets.org/trunk/classwx_activate_event.html): A window or application activation event
- ***wxCalendarEvent*** (http://docs.wxwidgets.org/trunk/classwx_calendar_event.html): Used with ***wxCalendarCtrl*** (http://docs.wxwidgets.org/trunk/classwx_calendar_ctrl.html)
- ***wxCalculateLayoutEvent*** (http://docs.wxwidgets.org/trunk/classwx_calculate_layout_event.html): Used to calculate window layout
- ***wxChildFocusEvent*** (http://docs.wxwidgets.org/trunk/classwx_child_focus_event.html): A child window focus event
- ***wxClipboardTextEvent*** (http://docs.wxwidgets.org/trunk/classwx_clipboard_text_event.html): A clipboard copy/cut/paste treebook event event
- ***wxCloseEvent*** (http://docs.wxwidgets.org/trunk/classwx_close_event.html): A close window or end session event
- ***wxCommandEvent*** (http://docs.wxwidgets.org/trunk/classwx_command_event.html): An event from a variety of standard controls
- ***wxContextMenuEvent*** (http://docs.wxwidgets.org/trunk/classwx_context_menu_event.html): An event generated when the user issues a context menu command
- ***wxDateEvent*** (http://docs.wxwidgets.org/trunk/classwx_date_event.html): Used with ***wxDatePickerCtrl*** (http://docs.wxwidgets.org/trunk/classwx_date_picker_ctrl.html)
- ***wxDialUpEvent*** (http://docs.wxwidgets.org/trunk/classwx_dial_up_event.html): Event send by ***wxDialUpManager*** (http://docs.wxwidgets.org/trunk/classwx_dial_up_manager.html)
- ***wxDropFilesEvent*** (http://docs.wxwidgets.org/trunk/classwx_drop_files_event.html): A drop files event
- ***wxEraseEvent*** (http://docs.wxwidgets.org/trunk/classwx_erase_event.html): An erase background event
- ***wxEvent*** (http://docs.wxwidgets.org/trunk/classwx_event.html): The event base class
- ***wxFindDialogEvent***: Event sent by ***wxFindReplaceDialog*** (http://docs.wxwidgets.org/trunk/classwx_find_replace_dialog.html)
- ***wxFocusEvent*** (http://docs.wxwidgets.org/trunk/classwx_focus_event.html): A window focus event
- ***wxKeyEvent*** (http://docs.wxwidgets.org/trunk/classwx_key_event.html): A keypress event
- ***wxIconizeEvent*** (http://docs.wxwidgets.org/trunk/classwx_iconize_event.html): An iconize/restore event
- ***wxIdleEvent*** (http://docs.wxwidgets.org/trunk/classwx_idle_event.html): An idle event
- ***wxInitDialogEvent*** (http://docs.wxwidgets.org/trunk/classwx_init_dialog_event.html): A dialog initialisation event
- ***wxJoystickEvent*** (http://docs.wxwidgets.org/trunk/classwx_joystick_event.html): A joystick event
- ***wxKeyboardState*** (http://docs.wxwidgets.org/trunk/classwx_keyboard_state.html): State of the keyboard modifiers.
- ***wxListEvent*** (http://docs.wxwidgets.org/trunk/classwx_list_event.html): A list control event

- ***wxMaximizeEvent*** (http://docs.wxwidgets.org/trunk/classwx_maximize_event.html): A maximize event
- ***wxMenuEvent*** (http://docs.wxwidgets.org/trunk/classwx_menu_event.html): A menu event
- ***wxMouseCaptureChangedEvent*** (http://docs.wxwidgets.org/trunk/classwx_mouse_capture_changed_event.html): A mouse capture changed event
- ***wxMouseCaptureLostEvent*** (http://docs.wxwidgets.org/trunk/classwx_mouse_capture_lost_event.html): A mouse capture lost event
- ***wxMouseEvent*** (http://docs.wxwidgets.org/trunk/classwx_mouse_event.html): A mouse event
- ***wxMouseState*** (http://docs.wxwidgets.org/trunk/classwx_mouse_state.html): State of the mouse
- ***wxMoveEvent*** (http://docs.wxwidgets.org/trunk/classwx_move_event.html): A move event
- ***wxNavigationKeyEvent*** (http://docs.wxwidgets.org/trunk/classwx_navigation_key_event.html): An event set by navigation keys such as tab
- ***wxNotebookEvent***: A notebook control event
- ***wxNotifyEvent*** (http://docs.wxwidgets.org/trunk/classwx_notify_event.html): A notification event, which can be vetoed
- ***wxPaintEvent*** (http://docs.wxwidgets.org/trunk/classwx_paint_event.html): A paint event
- ***wxProcessEvent***: A process ending event
- ***wxQueryLayoutInfoEvent*** (http://docs.wxwidgets.org/trunk/classwx_query_layout_info_event.html): Used to query layout information
- ***wxRichTextEvent***: A rich text editing event
- ***wxScrollEvent*** (http://docs.wxwidgets.org/trunk/classwx_scroll_event.html): A scroll event from sliders, stand-alone scrollbars and spin buttons
- ***wxScrollWinEvent*** (http://docs.wxwidgets.org/trunk/classwx_scroll_win_event.html): A scroll event from scrolled windows
- ***wxSizeEvent*** (http://docs.wxwidgets.org/trunk/classwx_size_event.html): A size event
- ***wxSocketEvent***: A socket event
- ***wxSpinEvent*** (http://docs.wxwidgets.org/trunk/classwx_spin_event.html): An event from ***wxSpinButton*** (http://docs.wxwidgets.org/trunk/classwx_spin_button.html)
- ***wxSplitterEvent*** (http://docs.wxwidgets.org/trunk/classwx_splitter_event.html): An event from ***wxSplitterWindow*** (http://docs.wxwidgets.org/trunk/classwx_splitter_window.html)
- ***wxSysColourChangedEvent*** (http://docs.wxwidgets.org/trunk/classwx_sys_colour_changed_event.html): A system colour change event
- ***wxTimerEvent*** (http://docs.wxwidgets.org/trunk/classwx_timer_event.html): A timer expiration event
- ***wxTreebookEvent***: A treebook control event
- ***wxTreeEvent*** (http://docs.wxwidgets.org/trunk/classwx_tree_event.html): A tree control event

- *wxUpdateUIEvent* (http://docs.wxwidgets.org/trunk/classwx_update_u_i_event.html): A user interface update event
- *wxWindowCreateEvent*: A window creation event
- *wxWindowDestroyEvent* (http://docs.wxwidgets.org/trunk/classwx_window_destroy_event.html): A window destruction event
- *wxWizardEvent* (http://docs.wxwidgets.org/trunk/classwx_wizard_event.html): A wizard event

33.17 Application and Process Management

- *wxApp* (http://docs.wxwidgets.org/trunk/classwx_app.html): Application class
- *wxCmdLineParser* (http://docs.wxwidgets.org/trunk/classwx_cmd_line_parser.html): Command line parser class
- *wxDynamicLibrary* (http://docs.wxwidgets.org/trunk/classwx_dynamic_library.html): Class to work with shared libraries.
- *wxProcess* (http://docs.wxwidgets.org/trunk/classwx_process.html): Process class

33.18 Printing Framework

A printing and previewing framework is implemented to make it relatively straightforward to provide document printing facilities.

33.18.1 Related Overviews: HLINK_295

- ***wxPreviewFrame*** (http://docs.wxwidgets.org/trunk/classwx_preview_frame.html): Frame for displaying a print preview
- ***wxPreviewCanvas*** (http://docs.wxwidgets.org/trunk/classwx_preview_canvas.html): Canvas for displaying a print preview
- ***wxPreviewControlBar*** (http://docs.wxwidgets.org/trunk/classwx_preview_control_bar.html): Standard control bar for a print preview
- ***wxPrintDialog*** (http://docs.wxwidgets.org/trunk/classwx_print_dialog.html): Standard print dialog
- ***wxPageSetupDialog*** (http://docs.wxwidgets.org/trunk/classwx_page_setup_dialog.html): Standard page setup dialog
- ***wxPrinter*** (http://docs.wxwidgets.org/trunk/classwx_printer.html): Class representing the printer
- ***wxPrinterDC*** (http://docs.wxwidgets.org/trunk/classwx_printer_d_c.html): Printer device context
- ***wxPrintout*** (http://docs.wxwidgets.org/trunk/classwx_printout.html): Class representing a particular printout
- ***wxPrintPreview*** (http://docs.wxwidgets.org/trunk/classwx_print_preview.html): Class representing a print preview
- ***wxPrintData*** (http://docs.wxwidgets.org/trunk/classwx_print_data.html): Represents information about the document being printed
- ***wxPrintDialogData*** (http://docs.wxwidgets.org/trunk/classwx_print_dialog_data.html): Represents information about the print dialog
- ***wxPageSetupDialogData*** (http://docs.wxwidgets.org/trunk/classwx_page_setup_dialog_data.html): Represents information about the page setup dialog

33.19 Document/View Framework

wxWidgets supports a document/view framework which provides housekeeping for a document-centric application.

33.19.1 Related Overviews: HLINK_308

- ***wxCommand*** (http://docs.wxwidgets.org/trunk/classwx_command.html): Base class for undo/redo actions
- ***wxCommandProcessor*** (http://docs.wxwidgets.org/trunk/classwx_command_processor.html): Maintains the undo/redo stack
- ***wxDocument*** (http://docs.wxwidgets.org/trunk/classwx_document.html): Represents a document
- ***wxView*** (http://docs.wxwidgets.org/trunk/classwx_view.html): Represents a view
- ***wxDocTemplate*** (http://docs.wxwidgets.org/trunk/classwx_doc_template.html): Manages the relationship between a document class and a view class
- ***wxDocManager*** (http://docs.wxwidgets.org/trunk/classwx_doc_manager.html): Manages the documents and views in an application
- ***wxDocChildFrame*** (http://docs.wxwidgets.org/trunk/classwx_doc_child_frame.html): A child frame for showing a document view
- ***wxDocParentFrame*** (http://docs.wxwidgets.org/trunk/classwx_doc_parent_frame.html): A parent frame to contain views
- ***wxDocMDIChildFrame*** (http://docs.wxwidgets.org/trunk/classwx_doc_m_d_i_child_frame.html): An MDI child frame for showing a document view
- ***wxDocMDIParentFrame*** (http://docs.wxwidgets.org/trunk/classwx_doc_m_d_i_parent_frame.html): An MDI parent frame to contain views
- ***wxFileHistory*** (http://docs.wxwidgets.org/trunk/classwx_file_history.html): Maintains a list of the most recently visited files

33.20 Clipboard and Drag & Drop

33.20.1 Related Overviews: HLINK_320

- *wxDataObject* (http://docs.wxwidgets.org/trunk/classwx_data_object.html): Data object class
- *wxDataFormat* (http://docs.wxwidgets.org/trunk/classwx_data_format.html): Represents a data format
- *wxTextDataObject* (http://docs.wxwidgets.org/trunk/classwx_text_data_object.html): Text data object class
- *wxFileDataObject* (http://docs.wxwidgets.org/trunk/classwx_file_data_object.html): File data object class
- *wxBitmapDataObject* (http://docs.wxwidgets.org/trunk/classwx_bitmap_data_object.html): Bitmap data object class
- *wxURLDataObject* (http://docs.wxwidgets.org/trunk/classwx_u_r_l_data_object.html): URL data object class
- *wxCustomDataObject* (http://docs.wxwidgets.org/trunk/classwx_custom_data_object.html): Custom data object class
- *wxClipboard* (http://docs.wxwidgets.org/trunk/classwx_clipboard.html): Clipboard class
- *wxDropTarget* (http://docs.wxwidgets.org/trunk/classwx_drop_target.html): Drop target class
- *wxFileDropTarget* (http://docs.wxwidgets.org/trunk/classwx_file_drop_target.html): File drop target class
- *wxTextDropTarget* (http://docs.wxwidgets.org/trunk/classwx_text_drop_target.html): Text drop target class
- *wxDropSource* (http://docs.wxwidgets.org/trunk/classwx_drop_source.html): Drop source class

33.21 Virtual File System

wxWidgets provides a set of classes that implement an extensible virtual file system, used internally by the HTML classes.

- ***wxFSFile*** (http://docs.wxwidgets.org/trunk/classwx_f_s_file.html): Represents a file in the virtual file system
- ***wxFileSystem*** (http://docs.wxwidgets.org/trunk/classwx_file_system.html): Main interface for the virtual file system
- ***wxFileSystemHandler*** (http://docs.wxwidgets.org/trunk/classwx_file_system_handler.html): Class used to announce file system type

33.22 Threading

wxWidgets provides a set of classes to make use of the native thread capabilities of the various platforms.

33.22.1 Related Overviews: HLINK_336

- ***wxThread***: Thread class
- ***wxThreadHelper*** (http://docs.wxwidgets.org/trunk/classwx_thread_helper.html): Manages background threads easily
- ***wxMutex*** (http://docs.wxwidgets.org/trunk/classwx_mutex.html): Mutex class
- ***wxMutexLocker*** (http://docs.wxwidgets.org/trunk/classwx_mutex_locker.html): Mutex locker utility class
- ***wxCriticalSection*** (http://docs.wxwidgets.org/trunk/classwx_critical_section.html): Critical section class
- ***wxCriticalSectionLocker*** (http://docs.wxwidgets.org/trunk/classwx_critical_section_locker.html): Critical section locker utility class
- ***wxCondition*** (http://docs.wxwidgets.org/trunk/classwx_condition.html): Condition class
- ***wxSemaphore*** (http://docs.wxwidgets.org/trunk/classwx_semaphore.html): Semaphore class

33.23 Runtime Type Information (RTTI)

wxWidgets supports runtime manipulation of class information, and dynamic creation of objects given class names.

33.23.1 Related Overviews: HLINK_344

See also:

RTTI Functions and Macros (http://docs.wxwidgets.org/trunk/group__group__funcmacro__rtti.html)

- **wxClassInfo**: Holds runtime class information
- **wxObject** (http://docs.wxwidgets.org/trunk/classwx_object.html): Root class for classes with runtime information

33.24 Debugging

wxWidgets supports some aspects of debugging an application through classes, functions and macros.

33.24.1 Related Overviews: HLINK_347

See also:

Debugging Functions and Macros

(http://docs.wxwidgets.org/trunk/group_group_funcmacro_debug.html)

- ***wxDebugContext*** (http://docs.wxwidgets.org/trunk/classwx_debug_context.html): Provides memory-checking facilities
- ***wxDebugReport*** (http://docs.wxwidgets.org/trunk/classwx_debug_report.html): Base class for creating debug reports in case of a program crash.
- ***wxDebugReportCompress*** (http://docs.wxwidgets.org/trunk/classwx_debug_report_compress.html): Class for creating compressed debug reports.
- ***wxDebugReportUpload*** (http://docs.wxwidgets.org/trunk/classwx_debug_report_upload.html): Class for uploading compressed debug reports via HTTP.
- ***wxDebugReportPreview*** (http://docs.wxwidgets.org/trunk/classwx_debug_report_preview.html): Abstract base class for previewing the contents of a debug report.
- ***wxDebugReportPreviewStd***
(http://docs.wxwidgets.org/trunk/classwx_debug_report_preview_std.html): Standard implementation of ***wxDebugReportPreview*** (http://docs.wxwidgets.org/trunk/classwx_debug_report_preview.html).

33.25 Logging

wxWidgets provides several classes and functions for message logging.

33.25.1 Related overview: HLINK_356

See also:

Logging Functions and Macros

- ***wxLog*** (http://docs.wxwidgets.org/trunk/classwx_log.html): The base log class
- ***wxLogStderr*** (http://docs.wxwidgets.org/trunk/classwx_log_stderr.html): Log messages to a C STDIO stream
- ***wxLogStream*** (http://docs.wxwidgets.org/trunk/classwx_log_stream.html): Log messages to a C++ iostream
- ***wxLogTextCtrl*** (http://docs.wxwidgets.org/trunk/classwx_log_text_ctrl.html): Log messages to a ***wxTextCtrl*** (http://docs.wxwidgets.org/trunk/classwx_text_ctrl.html)
- ***wxLogWindow*** (http://docs.wxwidgets.org/trunk/classwx_log_window.html): Log messages to a log frame
- ***wxLogGui*** (http://docs.wxwidgets.org/trunk/classwx_log_gui.html): Default log target for GUI programs
- ***wxLogNull*** (http://docs.wxwidgets.org/trunk/classwx_log_null.html): Temporarily suppress message logging
- ***wxLogChain*** (http://docs.wxwidgets.org/trunk/classwx_log_chain.html): Allows to chain two log targets
- ***wxLogInterposer*** (http://docs.wxwidgets.org/trunk/classwx_log_interposer.html): Allows to filter the log messages
- ***wxLogInterposerTemp*** (http://docs.wxwidgets.org/trunk/classwx_log_interposer_temp.html): Allows to filter the log messages
- ***wxStreamToTextRedirector*** (http://docs.wxwidgets.org/trunk/classwx_stream_to_text_redirector.html): Allows to redirect output sent to cout to a ***wxTextCtrl*** (http://docs.wxwidgets.org/trunk/classwx_text_ctrl.html)

33.26 Data Structures

These are the data structure classes supported by wxWidgets.

- ***wxAny*** (http://docs.wxwidgets.org/trunk/classwx_any.html): A class for storing arbitrary types that may change at run-time
- ***wxCmdLineParser*** (http://docs.wxwidgets.org/trunk/classwx_cmd_line_parser.html): Command line parser class
- ***wxDateSpan*** (http://docs.wxwidgets.org/trunk/classwx_date_span.html): A logical time interval.
- ***wxDateTime***: A class for date/time manipulations
- ***wxLongLong*** (http://docs.wxwidgets.org/trunk/classwx_long_long.html): A portable 64 bit integer type
- ***wxObject*** (http://docs.wxwidgets.org/trunk/classwx_object.html): The root class for most wxWidgets classes
- ***wxPathList*** (http://docs.wxwidgets.org/trunk/classwx_path_list.html): A class to help search multiple paths
- ***wxPoint*** (http://docs.wxwidgets.org/trunk/classwx_point.html): Representation of a point
- ***wxRect*** (http://docs.wxwidgets.org/trunk/classwx_rect.html): A class representing a rectangle
- ***wxRegEx*** (http://docs.wxwidgets.org/trunk/classwx_reg_ex.html): Regular expression support
- ***wxRegion*** (http://docs.wxwidgets.org/trunk/classwx_region.html): A class representing a region
- ***wxString*** (http://docs.wxwidgets.org/trunk/classwx_string.html): A string class
- ***wxStringTokenizer*** (http://docs.wxwidgets.org/trunk/classwx_string_tokenizer.html): A class for interpreting a string as a list of tokens or words
- ***wxRealPoint*** (http://docs.wxwidgets.org/trunk/classwx_real_point.html): Representation of a point using floating point numbers
- ***wxSize*** (http://docs.wxwidgets.org/trunk/classwx_size.html): Representation of a size
- ***wxTimeSpan*** (http://docs.wxwidgets.org/trunk/classwx_time_span.html): A time interval.
- ***wxURI*** (http://docs.wxwidgets.org/trunk/classwx_u_r_i.html): Represents a Uniform Resource Identifier
- ***wxVariant*** (http://docs.wxwidgets.org/trunk/classwx_variant.html): A class for storing arbitrary types that may change at run-time

33.27 Text Conversion

These classes define objects for performing conversions between different multibyte and Unicode encodings and wide character strings.

- ***wxMBConv*** (http://docs.wxwidgets.org/trunk/classwx_m_b_conv.html): Base class for all convertors, defines the API implemented by all the other convertor classes.
- ***wxMBConvUTF7*** (http://docs.wxwidgets.org/trunk/classwx_m_b_conv_u_t_f7.html): Convertor for UTF-7
- ***wxMBConvUTF8*** (http://docs.wxwidgets.org/trunk/classwx_m_b_conv_u_t_f8.html): Convertor for UTF-8
- ***wxMBConvUTF16*** (http://docs.wxwidgets.org/trunk/classwx_m_b_conv_u_t_f16.html): Convertor for UTF-16
- ***wxMBConvUTF32*** (http://docs.wxwidgets.org/trunk/classwx_m_b_conv_u_t_f32.html): Convertor for UTF-32
- ***wxCSCConv*** (http://docs.wxwidgets.org/trunk/classwx_c_s_conv.html): Convertor for any system-supported encoding which can be specified by name.

33.27.1 Related Overviews: HLINK_393

33.28 Containers

These are classes, templates and class macros are used by wxWidgets. Most of these classes provide a subset or almost complete STL API.

33.28.1 Related Overviews: HLINK_394

- ***wxArray<T>*** (http://docs.wxwidgets.org/trunk/classwx_array_3_01_t_01_4.html): A type-safe dynamic array implementation (macro based)
- ***wxArrayString*** (http://docs.wxwidgets.org/trunk/classwx_array_string.html): An efficient container for storing ***wxString*** (http://docs.wxwidgets.org/trunk/classwx_string.html) objects
- ***wxHashMap<T>***: A type-safe hash map implementation (macro based)
- ***wxHashSet<T>***: A type-safe hash set implementation (macro based)
- ***wxHashTable*** (http://docs.wxwidgets.org/trunk/classwx_hash_table.html): A simple hash table implementation (deprecated, use ***wxHashMap*** (http://docs.wxwidgets.org/trunk/classwx_hash_map.html))
- ***wxList<T>*** (http://docs.wxwidgets.org/trunk/classwx_list_3_01_t_01_4.html): A type-safe linked list implementation (macro based)
- ***wxVector<T>*** (http://docs.wxwidgets.org/trunk/classwx_vector_3_01_t_01_4.html): Template base vector implementation identical to `std::vector`

33.29 Smart Pointers

wxWidgets provides a few smart pointer class templates.

- ***wxObjectDataPtr<T>*** (http://docs.wxwidgets.org/trunk/classwx_object_data_ptr_3_01_t_01_4.html): A shared pointer (using intrusive reference counting)
- ***wxScopedPtr<T>*** (http://docs.wxwidgets.org/trunk/classwx_scoped_ptr_3_01_t_01_4.html): A scoped pointer
- ***wxSharedPtr<T>*** (http://docs.wxwidgets.org/trunk/classwx_shared_ptr_3_01_t_01_4.html): A shared pointer (using non-intrusive reference counting)
- ***wxWeakRef<T>*** (http://docs.wxwidgets.org/trunk/classwx_weak_ref_3_01_t_01_4.html): A weak reference

33.30 File Handling

wxWidgets has several small classes to work with disk files and directories.

33.30.1 Related overview: HLINK_406

- ***wxFileName*** (http://docs.wxwidgets.org/trunk/classwx_file_name.html): Operations with the file name and attributes
- ***wxDir*** (http://docs.wxwidgets.org/trunk/classwx_dir.html): Class for enumerating files/subdirectories.
- ***wxDirTraverser*** (http://docs.wxwidgets.org/trunk/classwx_dir_traverser.html): Class used together with ***wxDir*** (http://docs.wxwidgets.org/trunk/classwx_dir.html) for recursively enumerating the files/subdirectories
- ***wxFile*** (http://docs.wxwidgets.org/trunk/classwx_file.html): Low-level file input/output class.
- ***wxFile*** (http://docs.wxwidgets.org/trunk/classwx_f_file.html): Another low-level file input/output class.
- ***wxTempFile*** (http://docs.wxwidgets.org/trunk/classwx_temp_file.html): Class to safely replace an existing file
- ***wxTextFile*** (http://docs.wxwidgets.org/trunk/classwx_text_file.html): Class for working with text files as with arrays of lines
- ***wxStandardPaths*** (http://docs.wxwidgets.org/trunk/classwx_standard_paths.html): Paths for standard directories
- ***wxPathList*** (http://docs.wxwidgets.org/trunk/classwx_path_list.html): A class to help search multiple paths
- ***wxFileSystemWatcher*** (http://docs.wxwidgets.org/trunk/classwx_file_system_watcher.html): Class providing notifications of file system changes

33.31 Streams

wxWidgets has its own set of stream classes as an alternative to the standard stream libraries and to provide enhanced functionality.

33.31.1 Related overview: HLINK_418

- ***wxStreamBase*** (http://docs.wxwidgets.org/trunk/classwx_stream_base.html): Stream base class
- ***wxStreamBuffer*** (http://docs.wxwidgets.org/trunk/classwx_stream_buffer.html): Stream buffer class
- ***wxInputStream*** (http://docs.wxwidgets.org/trunk/classwx_input_stream.html): Input stream class
- ***wxOutputStream*** (http://docs.wxwidgets.org/trunk/classwx_output_stream.html): Output stream class
- ***wxCountingOutputStream*** (http://docs.wxwidgets.org/trunk/classwx_counting_output_stream.html): Stream class for querying what size a stream would have.
- ***wxFilterInputStream*** (http://docs.wxwidgets.org/trunk/classwx_filter_input_stream.html): Filtered input stream class
- ***wxFilterOutputStream*** (http://docs.wxwidgets.org/trunk/classwx_filter_output_stream.html): Filtered output stream class
- ***wxBufferedInputStream*** (http://docs.wxwidgets.org/trunk/classwx_buffered_input_stream.html): Buffered input stream class
- ***wxBufferedOutputStream*** (http://docs.wxwidgets.org/trunk/classwx_buffered_output_stream.html): Buffered output stream class
- ***wxMemoryInputStream*** (http://docs.wxwidgets.org/trunk/classwx_memory_input_stream.html): Memory input stream class
- ***wxMemoryOutputStream*** (http://docs.wxwidgets.org/trunk/classwx_memory_output_stream.html): Memory output stream class
- ***wxDataInputStream*** (http://docs.wxwidgets.org/trunk/classwx_data_input_stream.html): Platform-independent binary data input stream class
- ***wxDataOutputStream*** (http://docs.wxwidgets.org/trunk/classwx_data_output_stream.html): Platform-independent binary data output stream class
- ***wxTextInputStream*** (http://docs.wxwidgets.org/trunk/classwx_text_input_stream.html): Platform-independent text data input stream class
- ***wxTextOutputStream*** (http://docs.wxwidgets.org/trunk/classwx_text_output_stream.html): Platform-independent text data output stream class
- ***wxFileInputStream*** (http://docs.wxwidgets.org/trunk/classwx_file_input_stream.html): File input stream class
- ***wxFileOutputStream*** (http://docs.wxwidgets.org/trunk/classwx_file_output_stream.html): File output stream class
- ***wxFFileInputStream*** (http://docs.wxwidgets.org/trunk/classwx_f_file_input_stream.html): Another file input stream class
- ***wxFFileOutputStream*** (http://docs.wxwidgets.org/trunk/classwx_f_file_output_stream.html): Another file output stream class
- ***wxTempFileOutputStream*** (http://docs.wxwidgets.org/trunk/classwx_temp_file_output_stream.html): Stream to safely replace an existing file

- ***wxStringInputStream*** (http://docs.wxwidgets.org/trunk/classwx_string_input_stream.html): String input stream class
- ***wxStringOutputStream*** (http://docs.wxwidgets.org/trunk/classwx_string_output_stream.html): String output stream class
- ***wxZlibInputStream*** (http://docs.wxwidgets.org/trunk/classwx_zlib_input_stream.html): Zlib and gzip (compression) input stream class
- ***wxZlibOutputStream*** (http://docs.wxwidgets.org/trunk/classwx_zlib_output_stream.html): Zlib and gzip (compression) output stream class
- ***wxZipInputStream*** (http://docs.wxwidgets.org/trunk/classwx_zip_input_stream.html): Input stream for reading from ZIP archives
- ***wxZipOutputStream*** (http://docs.wxwidgets.org/trunk/classwx_zip_output_stream.html): Output stream for writing from ZIP archives
- ***wxTarInputStream*** (http://docs.wxwidgets.org/trunk/classwx_tar_input_stream.html): Input stream for reading from tar archives
- ***wxTarOutputStream*** (http://docs.wxwidgets.org/trunk/classwx_tar_output_stream.html): Output stream for writing from tar archives
- ***wxSocketInputStream*** (http://docs.wxwidgets.org/trunk/classwx_socket_input_stream.html): Socket input stream class
- ***wxSocketOutputStream*** (http://docs.wxwidgets.org/trunk/classwx_socket_output_stream.html): Socket output stream class

33.32 XML

- ***wxXmlDocument*** (http://docs.wxwidgets.org/trunk/classwx_xml_document.html): A class to parse XML files
- ***wxXmlNode*** (http://docs.wxwidgets.org/trunk/classwx_xml_node.html): A class which represents XML nodes
- ***wxXmlAttribute*** (http://docs.wxwidgets.org/trunk/classwx_xml_attribute.html): A class which represent an XML attribute

33.33 Archive

- ***wxArchiveInputStream*** (http://docs.wxwidgets.org/trunk/classwx_archive_input_stream.html)
- ***wxArchiveOutputStream*** (http://docs.wxwidgets.org/trunk/classwx_archive_output_stream.html)
- ***wxArchiveEntry*** (http://docs.wxwidgets.org/trunk/classwx_archive_entry.html)

33.34 XML Based Resource System (XRC)

Resources allow your application to create controls and other user interface elements from specifications stored in an XML format.

33.34.1 Related overview: HLINK_455

- *wxXmlResource* (http://docs.wxwidgets.org/trunk/classwx_xml_resource.html): The main class for working with resources
- *wxXmlResourceHandler* (http://docs.wxwidgets.org/trunk/classwx_xml_resource_handler.html): The base class for XML resource handlers

33.35 Networking

wxWidgets provides its own classes for socket based networking.

- *wxDialUpManager* (http://docs.wxwidgets.org/trunk/classwx_dial_up_manager.html): Provides functions to check the status of network connection and to establish one
- *wxIPv4address* (http://docs.wxwidgets.org/trunk/classwx_ip_v4address.html): Represents an Internet address
- *wxIPaddress* (http://docs.wxwidgets.org/trunk/classwx_ip_address.html): Represents an Internet address
- *wxSocketBase* (http://docs.wxwidgets.org/trunk/classwx_socket_base.html): Represents a socket base object
- *wxSocketClient* (http://docs.wxwidgets.org/trunk/classwx_socket_client.html): Represents a socket client
- *wxSocketServer* (http://docs.wxwidgets.org/trunk/classwx_socket_server.html): Represents a socket server
- *wxSocketEvent* (http://docs.wxwidgets.org/trunk/classwx_socket_event.html): A socket event
- *wxFTP*: FTP protocol class
- *wxHTTP* (http://docs.wxwidgets.org/trunk/classwx_http.html): HTTP protocol class
- *wxURL* (http://docs.wxwidgets.org/trunk/classwx_url.html): Represents a Universal Resource Locator

33.36 Interprocess Communication

wxWidgets provides simple interprocess communications facilities based on Windows DDE, but available on most platforms using TCP.

33.36.1 Related overview: HLINK_467

- ***wxCClient*** (http://docs.wxwidgets.org/trunk/classwx_client.html), ***wxDDEClient*** (http://docs.wxwidgets.org/trunk/classwx_d_d_e_client.html): Represents a client
- ***wxConnection***, ***wxDDEConnection*** (http://docs.wxwidgets.org/trunk/classwx_d_d_e_connection.html): Represents the connection between a client and a server
- ***wxServer*** (http://docs.wxwidgets.org/trunk/classwx_server.html), ***wxDDEServer*** (http://docs.wxwidgets.org/trunk/classwx_d_d_e_server.html): Represents a server

33.37 Help

- ***wxHelpController*** (http://docs.wxwidgets.org/trunk/classwx_help_controller.html): Family of classes for controlling help windows
- ***wxHtmlHelpController*** (http://docs.wxwidgets.org/trunk/classwx_html_help_controller.html): HTML help controller class
- ***wxContextHelp*** (http://docs.wxwidgets.org/trunk/classwx_context_help.html): Class to put application into context-sensitive help mode
- ***wxContextHelpButton*** (http://docs.wxwidgets.org/trunk/classwx_context_help_button.html): Button class for putting application into context-sensitive help mode
- ***wxHelpProvider***: Abstract class for context-sensitive help provision
- ***wxSimpleHelpProvider*** (http://docs.wxwidgets.org/trunk/classwx_simple_help_provider.html): Class for simple context-sensitive help provision
- ***wxHelpControllerHelpProvider*** (http://docs.wxwidgets.org/trunk/classwx_help_controller_help_provider.html): Class for context-sensitive help provision via a help controller
- ***wxToolTip*** (http://docs.wxwidgets.org/trunk/classwx_tool_tip.html): Class implementing tooltips

33.38 Multimedia

- *wxMediaCtrl* (http://docs.wxwidgets.org/trunk/classwx_media_ctrl.html): Display multimedia contents.

33.39 OpenGL

- *wxGLCanvas* (http://docs.wxwidgets.org/trunk/classwx_g_l_canvas.html): Canvas that you can render OpenGL calls to.
- *wxGLContext* (http://docs.wxwidgets.org/trunk/classwx_g_l_context.html): Class to ease sharing of OpenGL data resources.

Draft

33.40 Miscellaneous

- ***wxCaret*** (http://docs.wxwidgets.org/trunk/classwx_caret.html): A caret (cursor) object
- ***wxConfigBase*** (http://docs.wxwidgets.org/trunk/classwx_config_base.html): Classes for reading/writing the configuration settings
- ***wxTimer*** (http://docs.wxwidgets.org/trunk/classwx_timer.html): Timer class
- ***wxStopWatch*** (http://docs.wxwidgets.org/trunk/classwx_stop_watch.html): Stop watch class
- ***wxMimeTypeManager*** (http://docs.wxwidgets.org/trunk/classwx_mime_types_manager.html): MIME-types manager class
- ***wxSystemSettings*** (http://docs.wxwidgets.org/trunk/classwx_system_settings.html): System settings class for obtaining various global parameters
- ***wxSystemOptions*** (http://docs.wxwidgets.org/trunk/classwx_system_options.html): System options class for run-time configuration
- ***wxAcceleratorTable*** (http://docs.wxwidgets.org/trunk/classwx_accelerator_table.html): Accelerator table
- ***wxAutomationObject*** (http://docs.wxwidgets.org/trunk/classwx_automation_object.html): OLE automation class
- ***wxFontMapper*** (http://docs.wxwidgets.org/trunk/classwx_font_mapper.html): Font mapping, finding suitable font for given encoding
- ***wxEncodingConverter*** (http://docs.wxwidgets.org/trunk/classwx_encoding_converter.html): Encoding conversions
- ***wxCalendarDateAttr*** (http://docs.wxwidgets.org/trunk/classwx_calendar_date_attr.html): Used with ***wxCalendarCtrl*** (http://docs.wxwidgets.org/trunk/classwx_calendar_ctrl.html)
- ***wxQuantize*** (http://docs.wxwidgets.org/trunk/classwx_quantize.html): Class to perform quantization, or colour reduction
- ***wxSingleInstanceChecker*** (http://docs.wxwidgets.org/trunk/classwx_single_instance_checker.html): Check that only single program instance is running