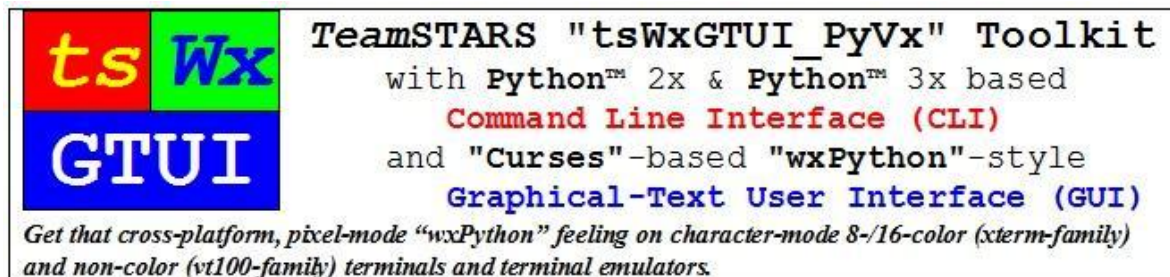


Interface Requirements Specification

Vol. 6 - "tsWxGTUI_PyVx" Toolkit

Rev. 0.0.7 (Pre-Alpha)

Author(s): Richard S. Gordon



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

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

Third-Party Component Author Copyrights & User Licenses

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

Draft

Contents

1	SCOPE (System Specification)	3
1.1	Identification (System Specification)	3
1.2	Purpose (System Specification).....	6
1.3	Document Overview (System Specification)	8
2	DEFINITIONS, ABBREVIATIONS, AND ACRONYMS	21
3	REQUIREMENTS	51
3.1	Interface Identification and Diagrams	51
3.1.1	Architecture	51
3.2	(Project Unique Identifier Of Interface Template)	62
3.3	POSIX-style Desktop Environments (DE)	65
3.4	Simplified Wrapper and Interface Generator (SWIG).....	68
3.4.1	Function	69
3.4.2	Example	69
3.4.3	Purpose	70
3.4.4	History	70
3.4.5	SWIG Tutorial	70
3.5	Terminal Device Interface (TDI).....	76
3.5.1	Display.....	76
3.5.2	Keyboard	91
3.5.3	Mouse	91
3.6	Precedence and criticality of requirements.....	92
4	QUALIFICATION PROVISIONS	93
5	REQUIREMENTS TRACEABILITY	95
6	NOTES	97
7	APPENDIXES (Interface Requirements)	99

Draft


1 SCOPE (System Specification)

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

- *Identification (System Specification)* (on page 3)
- *Purpose (System Specification)* (on page 6)
- *Document Overview (System Specification)* (on page 8)

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> <ol style="list-style-type: none"> 1 TeamSTARS "tsWxGTUI_Py1x" Toolkit (reserved for legacy Python 1x; to be created by back-port from TeamSTARS "tsWxGTUI_Py2x") 2 TeamSTARS "tsWxGTUI_Py2x" Toolkit (for mature Python 2x; created as the Toolkit prototype; then upgraded as Python 2x evolved) 3 TeamSTARS "tsWxGTUI_Py3x" Toolkit (for evolving Python 3x; created as port from TeamSTARS "tsWxGTUI_Py2x"; then upgraded as Python 3x evolves) 4 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	<i>Team</i> STARS "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	0.0.7
Release Number	0.0.7 (pre-alpha)
Build Date	08/24/2016

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 2.0.0-2.7.12**.
 - c) **"tsWxGTUI_Py3x"** - Root of third generation syntax files and subdirectories for **Python 3.0.0-3.5.2**.
 - 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 "License.txt" files.

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 2.0.0-2.7.12
 - c) *TeamSTARS* "tsWxGTUI_Py3x" Toolkit for the third generation Python language 3.0.0-3.5.2
 - 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) What is it?</p> <p>Application Design</p> <p>b) How can you get it?</p> <p>Where to find it on GitHub?</p> <p>How to download it from Github as Clone (includes all releases) or ZIP file (includes only latest release)?</p> <p>c) What does it contain?</p> <p>Documents, ManPages, Notebooks, Source Distributions and release Manifest.</p> <p>d) How to get started?</p> <p>Read "GETTING_STARTED.txt". It is the first of a number of documents for Toolkit administrators and users. It identifies which hardware and software components you will need and any preparation required for Toolkit use.</p> <p>Read "README.txt". It is the first of a number of documents for Toolkit and User application software developers, troubleshooters and maintainers.</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</p>

		<p>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 tool;kit 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-</p>
--	--	--

		Color Pairs
2	Introduction	<p>This document orients the reader to the goals and non-goals for the "tsWxGTUI_PyVx" Toolkit. Included are the following topics:</p> <ol style="list-style-type: none"> 1 SCOPE (System Specification) <ol style="list-style-type: none"> a) Identification (System Specification) b) Purpose (System Specification) c) Document Overview (System Specification) 2 SYSTEM OVERVIEW (tsWxGTUI Introduction) <ol style="list-style-type: none"> a) Purpose of System and Software b) General Nature of the System and Software c) History of System Development, Operation and Maintenance 3 OPERATOR INTERFACE <ol style="list-style-type: none"> a) Command Line Interface (CLI) b) Graphical User Interface (GUI) 4 APPLICATION PROGRAMMING INTERFACE <ol style="list-style-type: none"> a) Command Line Interface API b) Graphical User Interface API 5 TOOLS & UTILITIES <ol style="list-style-type: none"> a) tsLinesOfCodeProjectMetrics b) tsPlatformQuery c) tStripComments d) tsStripLineNumbers e) tsTreeCopy f) tsTreeTrimLines 6 DEFINITIONS, ABBREVIATIONS, AND ACRONYMS 7 REFERENCED DOCUMENTS <ol style="list-style-type: none"> a) Project Documents b) Release Distribution Documents c) External Documents 8 NOTES <ol style="list-style-type: none"> a) Operator Interface Technology 9 APPENDIXES <ol style="list-style-type: none"> a) Appendix A - Baseline Host Computer Platforms

		<ul style="list-style-type: none"> 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 Commnd 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> <p>1 SCOPE (System Specification)</p> <p>2 DEFINITIONS, ABBREVIATIONS, AND ACRONYMS</p> <p>3 OVERVIEW OF REQUIRED WORK</p> <ul style="list-style-type: none"> a) Purpose b) Requirements and Constraints

		<ul style="list-style-type: none"> 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 e) Software implementation And unit Test f) Unit integration And Testong g) CSCI Qualification Testing h) CSCI/HWCI Integration And Testing i) System Qualification Testing j) Preparing for Software Use k) Preparing for Software Transition l) Software Configuration Management m) Software Product Evaluation n) Software Quality Assurance o) Corrective Action p) Joint Technical and Management Reviews q) Other Software Development Activities
		6 SCHEDULES AND ACTIVITY NETWORK
		7 PROJECT ORGANIZATION AND RESOURCES <ul style="list-style-type: none"> a) Project organization b) Project Resources
		8 NOTES
		9 APPENDIXES
		10 TO-DO-LIST <ul style="list-style-type: none"> a) New Features b) Modifications c) Troubleshooting

		<p>d) Validation/Regression Test</p> <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> <p>1 SCOPE (System Specification)</p> <p>2 DEFINITIONS, ABBREVIATIONS, AND ACRONYMS</p> <p>3 REQUIREMENTS</p> <p>4 QUALIFICATION PROVISIONS</p> <p>a) Demonstration</p> <p>b) Test</p> <p>c) Analysis</p> <p>d) Inspection</p> <p>e) Special Qualification Methods</p> <p>5 REQUIREMENTS TRACEABILITY</p> <p>6 NOTES</p> <p>a) Partial List of Widget Toolkits</p> <p>b) Use Case(s)</p> <p>System Administrator</p> <p>Software Engineer</p> <p>System Operator</p> <p>7 APPENDIXES</p> <p>8 APPENDIX A - REQUIRED STATES AND MODES</p> <p>9 APPENDIX B - SYSTEM CAPABILITY REQUIREMENTS</p> <p>10 APPENDIX C - SYSTEM EXTERNAL INTERFACE REQUIREMENTS</p> <p>11 APPENDIX D - SYSTEM INTERNAL INTERFACE REQUIREMENTS</p> <p>12 APPENDIX E - SYSTEM INTERNAL DATA REQUIREMENTS</p> <p>13 APPENDIX F - ADAPTATION REQUIREMENTS</p> <p>14 APPENDIX G - SAFETY REQUIREMENTS</p> <p>15 APPENDIX H - SECURITY AND PRIVACY</p>

		<p>REQUIREMENTS</p> <p>16 APPENDIX I - SYSTEM ENVIRONMENT REQUIREMENTS</p> <p>17 APPENDIX J - COMPUTER RESOURCE REQUIREMENTS</p> <p>18 APPENDIX K - SYSTEM QUALITY FACTORS</p> <p>19 APPENDIX L - DESIGN AND CONSTRUCTION CONSTRAINTS</p> <p>20 APPENDIX M - PERSONNEL-RELATED REQUIREMENTS</p> <p>21 APPENDIX N - TRAINING-RELATED REQUIREMENTS</p> <p>22 APPENDIX O - LOGISTICS-RELATED REQUIREMENTS</p> <p>23 APPENDIX P - OTHER REQUIREMENTS</p> <p>24 APPENDIX Q - PACKAGING REQUIREMENTS</p> <p>25 APPENDIX R - PRECEDENCE AND CRITICALITY OF REQUIREMENTS</p>
6	Interface Requirements Specification	<p>This document specifies the requirements imposed on one or more systems, subsystems, Hardware Configuration Items (HWCI)s, Computer Software Configuration Items (CSCI)s, manual operations, or other system components to achieve one or more interfaces among these entities. An IRS can cover any number of interfaces.</p> <p>Included are the following topics:</p> <p>1 SCOPE (System Specification)</p> <p>2 DEFINITIONS, ABBREVIATIONS, AND ACRONYMS</p> <p>3 REQUIREMENTS</p> <p>a) Interface Identification and Diagrams</p> <p>b) (Project unique identifier of interface)</p> <p>c) Terminal Device Interface (TDI)</p> <p>d) Precedence and Criticality of Requirements</p> <p>4 QUALIFICATION PROVISIONS</p> <p>5 REQUIREMENTS TRACEABILITY</p> <p>6 NOTES</p> <p>7 APPENDIXES</p>
7	Software Requirements Specification	<p>This document specifies the requirements for a Computer Software Configuration Item (CSCI) and the methods to be used to ensure that each requirement has been met. Requirements pertaining to the CSCI's external interfaces may be presented in the SRS or in one or more Interface Requirements Specifications (IRSs).</p>

	<p>Included are the following topics:</p> <ul style="list-style-type: none">1 SCOPE (System Specification)2 DEFINITIONS, ABBREVIATIONS, AND ACRONYMS3 REQUIREMENTS<ul style="list-style-type: none">a) Required States and Modesb) CSCI Capability Requirementsc) CSCI External Interface Requirementsd) CSCI Internal Interface Requirementse) CSCI Internal Data Requirementsf) Adaptation Requirementsg) Safety Requirementsh) Security and Privacy Requirementsi) CSCI Environment Requirementsj) Computer Resource Requirementsk) Software Quality Factorsl) Design and Construction Constraintsm) Personnel-related Requirementsn) Training-related Requirementso) Logistics-related Requirementsp) Other Requirementsq) Packaging Requirementsr) Precedence and Criticality of Requirements4 QUALIFICATION PROVISIONS5 REQUIREMENTS TRACEABILITY6 NOTES<ul style="list-style-type: none">a) TO-DO-LISTb) Command Line Interface Libraryc) Graphical Text User Interface Library7 APPENDIXES<ul style="list-style-type: none">a) Appendix A - Nature of System and Software8 TECHNICAL IMPACT ANALYSIS9 TEST REQUIREMENTS AND RESTRICTIONS10 USE CASES11 TRACEABILITY INFORMATION
--	---

		12 CLASS LIST BY CATEGORY
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) 4 PACKAGE CONTENTS <ul style="list-style-type: none"> a) Command Line Interface Library b) Graphical Text User Interface Library 5 FIXED BUGS & ISSUES 6 KNOWN BUGS & ISSUES 7 NEW FEATURES 8 APPLICATION NOTES <ul style="list-style-type: none"> a) Installation Procedure b) User Interface Design c) Developer 9 PERFORMANCE TUNING 10 ACCEPTANCE TESTING 11 COMPONENT STATUS 12 APPENDIXES <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> <ul style="list-style-type: none"> 1 SCOPE (System Specification) 2 SOFTWARE SUMMARY

		<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)

		<ul style="list-style-type: none"> 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 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 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	<p>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.</p> <p>1 Computer User Use Case(s)</p> <ul style="list-style-type: none"> a) Role-Specific b) System-Specific c) Application-Specific <p>2 Computer Source Code Use Case(s)</p> <ul style="list-style-type: none"> a) Comparison with "wxPython" b) High, Low and Extended API Relationships

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>

TERM	DEFINITION
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. 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>

TERM	DEFINITION
Embedded Systems	<p>Excerpt From Wikipedia, the free encyclopedia</p> <p>"An embedded system is a computer system with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints. It is embedded as part of a complete device often including hardware and mechanical parts. Embedded systems control many devices in common use today. Ninety-eight percent of all microprocessors are manufactured as components of embedded systems.</p> <p>Examples of properties of typically embedded computers when compared with general-purpose counterparts are low power consumption, small size, rugged operating ranges, and low per-unit cost. This comes at the price of limited processing resources, which make them significantly more difficult to program and to interact with. However, by building intelligence mechanisms on top of the hardware, taking advantage of possible existing sensors and the existence of a network of embedded units, one can both optimally manage available resources at the unit and network levels as well as provide augmented functions, well beyond those available. For example, intelligent techniques can be designed to manage power consumption of embedded systems.</p> <p>Modern embedded systems are often based on microcontrollers (i.e. CPUs with integrated memory or peripheral interfaces), but ordinary microprocessors (using external chips for memory and peripheral interface circuits) are also common, especially in more-complex systems. In either case, the processor(s) used may be types ranging from general purpose to those specialised in certain class of computations, or even custom designed for the application at hand. A common standard class of dedicated processors is the digital signal processor (DSP).</p> <p>Since the embedded system is dedicated to specific tasks, design engineers can optimize it to reduce the size and cost of the product and increase the reliability and performance. Some embedded systems are mass-produced, benefiting from economies of scale.</p> <p>Embedded systems range from portable devices such as digital watches and MP3 players, to large stationary installations like traffic lights, factory controllers, and largely complex systems like hybrid vehicles, MRI, and avionics. Complexity varies from low, with a single microcontroller chip, to very high with multiple units, peripherals and networks mounted inside a large chassis or enclosure."</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</p>

TERM	DEFINITION
	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."
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>

TERM	DEFINITION
	Richard Stallman views GNU as a "technical means to a social end".[20]"
GNU/Linux	<p>Excerpted From https://www.gnu.org/gnu/linux-and-gnu.html :</p> <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>

TERM	DEFINITION
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 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, ntfspgms, 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</p>

TERM	DEFINITION
	<p>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 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]"</p>
Instrumentation	<p>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.</p>
Interface Requirements Specification	<p>The Interface Requirements Specification (IRS) specifies the requirements imposed on one or more systems, subsystems, Hardware Configuration Items (HWCIIs), Computer Software Configuration Items (CSCIIs), manual operations, or other system components to achieve one or more interfaces among these entities. An IRS can cover any number of interfaces.</p>
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	<p>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.</p>
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	<p>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.</p>

TERM	DEFINITION
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>
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> <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</p>

TERM	DEFINITION
	<p>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 (NS)."</p> <p>A copy of the specification is available via a download link.</p>
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	<p>From Wikipedia, the free encyclopedia</p> <p>"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."</p>
Non-Functional Requirements	<p>From Wikipedia, the free encyclopedia:</p> <p>See Functional Requirements for definition and relationship.</p>
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	<p>From Wikipedia, the free encyclopedia</p> <p>"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]"</p>
OpenSolaris	<p>From Wikipedia, the free encyclopedia</p> <p>"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.</p> <p>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]</p> <p>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</p>

TERM	DEFINITION
	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	<p>From Wikipedia, the free encyclopedia</p> <p>"Parallels Desktop for Mac by Parallels, Inc., is software providing hardware virtualization for 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>

TERM	DEFINITION
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> <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</p>

TERM	DEFINITION
	<p>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] ▪ Copyleft No <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>

TERM	DEFINITION
Root Package	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.
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.
Site-Package	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 <code>site-package.package.module</code> path identifier.
Software Engineer	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. 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.
Software Requirements Specification	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 (IRs) (DI-IPSC-81434) referenced from the SRS.

TERM	DEFINITION
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>
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.
Supervisory Control And Data Acquisition	<p>Excerpt From Wikipedia, the free encyclopedia</p> <p>"Supervisory control and data acquisition (SCADA) is a system for remote monitoring and control that operates with coded signals over communication channels (using typically one communication channel per remote station).</p> <p>The control system may be combined with a data acquisition system by adding the use of coded signals over communication channels to acquire information about the status of the remote equipment for display or for recording functions. It is a type of industrial control system (ICS). Industrial control systems are computer-based systems that monitor and control industrial processes that exist in the physical world. SCADA systems historically distinguish themselves from other ICS systems by being large-scale processes that can include multiple sites, and large distances. These</p>

TERM	DEFINITION
	<p>processes include industrial, infrastructure, and facility-based processes, as described below:</p> <ul style="list-style-type: none"> ▪ Industrial processes include those of manufacturing, production, power generation, fabrication, and refining, and may run in continuous, batch, repetitive, or discrete modes. ▪ Infrastructure processes may be public or private, and include water treatment and distribution, wastewater collection and treatment, oil and gas pipelines, electrical power transmission and distribution, wind farms, civil defense siren systems, and large communication systems. ▪ Facility processes occur both in public facilities and private ones, including buildings, airports, ships, and space stations. They monitor and control heating, ventilation, and air conditioning systems (HVAC), access, and energy consumption. <p>Common system components</p> <p>A SCADA system usually consists of the following subsystems:</p> <ul style="list-style-type: none"> ▪ Remote terminal units (RTUs) connect to sensors in the process and convert sensor signals to digital data. They have telemetry hardware capable of sending digital data to the supervisory system, as well as receiving digital commands from the supervisory system. RTUs often have embedded control capabilities such as ladder logic in order to accomplish boolean logic operations. ▪ Programmable logic controller (PLCs) connect to sensors in the process and convert sensor signals to digital data. PLCs have more sophisticated embedded control capabilities (typically one or more IEC 61131-3 programming languages) than RTUs. PLCs do not have telemetry hardware, although this functionality is typically installed alongside them. PLCs are sometimes used in place of RTUs as field devices because they are more economical, versatile, flexible, and configurable. ▪ A telemetry system is typically used to connect PLCs and RTUs with control centers, data warehouses, and the enterprise. Examples of wired telemetry media used in SCADA systems include leased telephone lines and WAN circuits. Examples of wireless telemetry media used in SCADA systems include satellite (VSAT), licensed and unlicensed radio, cellular and microwave. ▪ A data acquisition server is a software service which uses industrial protocols to connect software services, via telemetry, with field devices such as RTUs and PLCs. It allows clients to access data from these field devices using standard protocols. ▪ A human-machine interface or HMI is the apparatus or device which presents processed data to a human operator, and through this, the human operator monitors and interacts with the process. The HMI is a client that requests data from a data acquisition server or in most installations the HMI is the graphical user interface for the operator, collects all data from external devices, creates reports, performs alarming, sends notifications, etc. ▪ A historian is a software service which accumulates time-stamped data, boolean events, and boolean alarms in a database which can be queried or used to populate graphic trends in the HMI. The historian is a client that requests data from a data acquisition server. ▪ A supervisory (computer) system, gathering (acquiring) data on the process and sending commands (control) to the SCADA system. ▪ Communication infrastructure connecting the supervisory system to the remote terminal units. ▪ Various processes and analytical instrumentation."
System Administration Utilities	Computer programs that computer system administrators use to install, debug, maintain, or otherwise support computer hardware and software.
System	An individual who will specify, plan, supervise and/or perform the installation, configuration,

TERM	DEFINITION
Administrator	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. ▪ 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

TERM	DEFINITION
	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 qualificatuin test components for a Python-based Command Line Interface.
tsTestsGUI	The identifier for a library of qualificatuin 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	<p>The generic identifier for the following Python language generation specific <i>TeamSTARS</i> "tsWxGTUI" Toolkit releases:</p> <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.

TERM	DEFINITION
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>
UWIN	<p>From Wikipedia, the free encyclopedia</p> <p>"UWIN is a computer software package created by David Korn which allows programs written for the operating system Unix be built and run on Microsoft Windows with few, if any, changes. Some of the software development was subcontracted to Wipro, India. References, correct or not, to the software as U/Win and AT&T Unix for Windows can be found in some cases, especially from the early days of its existence.</p> <p>UWIN source and binaries are available under the Open Source Eclipse Public License 1.0 at AT&T AST/UWIN open source downloads."</p> <p>See also:</p> <ul style="list-style-type: none"> ▪ Cygwin, a similar project started at about the same time[2] ▪ Interix, the Microsoft product in this area ▪ Windows Services for Unix ▪ MKS Toolkit, a third-party proprietary product in this area ▪ DJGPP (DJ's GNU Programming Platform), a Windows port of Gnu programming tools ▪ Xming ▪ UnxUtils ▪ GnuWin32 ▪ GNUWin II ▪ MinGW ▪ LBW: Linux Binaries on Windows requires Interix to be installed first."
VMware Fusion	<p>From Wikipedia, the free encyclopedia</p>

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

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

TERM	DEFINITION
	<ul style="list-style-type: none"> Host tools, such as wxEmulator <p>Platforms supported are : x86 and ARM"</p>
wxPython	A popular Python language binding for the cross-platform, "wxWidgets" GUI Toolkit.
wxWidgets	<p>A C++ library that lets developers create applications for Windows, OS X, Linux and UNIX on 32-bit and 64-bit architectures as well as several mobile platforms including Windows Mobile, iPhone SDK and embedded GTK+.</p> <p>It has popular language bindings for Python, Perl, Ruby and many other languages.</p> <p>"wxWidgets" (formerly "wxWindows") is a widget toolkit for creating graphical user interfaces (GUIs) for cross-platform applications. wxWidgets enables a program's GUI code to compile and run on several computer platforms with minimal or no code changes. It covers systems such as Microsoft Windows, Mac OS X (Carbon and Cocoa), iOS (Cocoa Touch), GNU/Linux Linux/Unix (X11, Motif, and GTK+), OpenVMS, OS/2 and AmigaOS. A version for embedded systems is under development.</p>
wxWindows License	<p>From Wikipedia, the free encyclopedia</p> <p>'wxWidgets is distributed under a custom made wxWindows License, similar to the GNU Lesser General Public License, with an exception stating that derived works in binary form may be distributed on the user's own terms.[5] This license is a free software license approved by the FSF,[17] making wxWidgets free software. It has been approved by the Open Source Initiative (OSI).[18]"</p>
xterm	<p>From Wikipedia, the free encyclopedia:</p> <p>"Not to be confused with X terminal display hardware.</p> <p>In computing, xterm is the standard terminal emulator for the X Window System. A user can have many different invocations of xterm running at once on the same display, each of which provides independent input/output for the process running in it (normally the process is a Unix shell).</p> <p>xterm originated prior to the X Window System. It was originally written as a stand-alone terminal emulator for the VAXStation 100 (VS100) by Mark Vandevoorde, a student of Jim Gettys, in the summer of 1984, when work on X started. It rapidly became clear that it would be more useful as part of X than as a standalone program, so it was retargeted to X. As Gettys tells the story, "part of why xterm's internals are so horrifying is that it was originally intended that a single process be able to drive multiple VS100 displays." [2]</p> <p>After many years as part of the X reference implementation, around 1996 the main line of development then shifted to XFree86 (which itself forked from X11R6.3), and it is presently actively maintained by Thomas Dickey.</p> <p>Many xterm variants are also available.[3] Most terminal emulators for X started as variations on xterm."</p> <p>NOTES:</p> <p>The "tsWxGTUI_PyVx" Toolkit supports the xterm, xterm-color, xterm-16color and xterm-256color terminal emulators with the following limitations:</p> <ul style="list-style-type: none"> The "tsWxGTUI_PyVx" Toolkit supports the xterm, xterm-color and xterm-16color terminal emulators. The inability to change the curses palette made it necessary to map the 68-color wxPython palette into the default 8-color curses palette. The "tsWxGTUI_PyVx" Toolkit does NOT yet support the xterm-256color terminal emulator. The inability to recreate the 68-color wxPython palette results in inappropriate colors and the

TERM	DEFINITION
	appearance of spurious lines streaking across the display.

TERM	TEST TYPE DEFINITION
Acceptance Testing	Excerpt From: http://www.exforsys.com/tutorials/testing/testing-types.html Testing the system with the intent of confirming readiness of the product and customer acceptance.
Ad Hoc Testing	Excerpt From: http://www.exforsys.com/tutorials/testing/testing-types.html Testing without a formal test plan or outside of a test plan. With some projects this type of testing is carried out as an adjunct to formal testing. If carried out by a skilled tester, it can often find problems that are not caught in regular testing. Sometimes, if testing occurs very late in the development cycle, this will be the only kind of testing that can be performed. Sometimes ad hoc testing is referred to as exploratory testing.
Alpha Testing	Excerpt From: http://www.exforsys.com/tutorials/testing/testing-types.html Testing after code is mostly complete or contains most of the functionality and prior to users being involved. Sometimes a select group of users are involved. More often this testing will be performed in-house or by an outside testing firm in close cooperation with the software engineering department.
Automated Testing	Excerpt From: http://www.exforsys.com/tutorials/testing/testing-types.html Software testing that utilizes a variety of tools to automate the testing process and when the importance of having a person manually testing is diminished. Automated testing still requires a skilled quality assurance professional with knowledge of the automation tool and the software being tested to set up the tests.
Beta Testing	Excerpt From: http://www.exforsys.com/tutorials/testing/testing-types.html Testing after the product is code complete. Betas are often widely distributed or even distributed to the public at large in hopes that they will buy the final product when it is released.
Black Box Testing	Excerpt From: http://www.exforsys.com/tutorials/testing/testing-types.html Testing software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as a specification or requirements document..
Compatibility Testing	Excerpt From: http://www.exforsys.com/tutorials/testing/testing-types.html Testing used to determine whether other system software components such as browsers, utilities, and competing software will conflict with the software being tested.
Configuration Testing	Excerpt From: http://www.exforsys.com/tutorials/testing/testing-types.html Testing to determine how well the product works with a broad range of hardware/peripheral equipment configurations as well as on different operating systems and software.
Functional Test	Excerpt From: http://www.exforsys.com/tutorials/testing/testing-types.html Functional test can be defined as testing two or more modules together with the intent of finding defects, demonstrating that defects are not present, verifying that the module performs its intended functions as stated in the specification and establishing confidence that a program does what it is supposed to do.

TERM	TEST TYPE DEFINITION
Independent Verification and Validation (IV&V)	<p>Excerpt From: http://www.exforsys.com/tutorials/testing/testing-types.html</p> <p>The process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and doesn't fail in an unacceptable manner. The individual or group doing this work is not part of the group or organization that developed the software. A term often applied to government work or where the government regulates the products, as in medical devices.</p>
Installation Testing	<p>Excerpt From: http://www.exforsys.com/tutorials/testing/testing-types.html</p> <p>Testing with the intent of determining if the product will install on a variety of platforms and how easily it installs.</p>
Integration Testing	<p>Excerpt From: http://www.exforsys.com/tutorials/testing/testing-types.html</p> <p>Testing two or more modules or functions together with the intent of finding interface defects between the modules or functions. Testing completed at as a part of unit or functional testing, and sometimes, becomes its own standalone test phase. On a larger level, integration testing can involve a putting together of groups of modules and functions with the goal of completing and verifying that the system meets the system requirements. (see system testing)</p>
Load Testing	<p>Excerpt From: http://www.exforsys.com/tutorials/testing/testing-types.html</p> <p>Testing with the intent of determining how well the product handles competition for system resources. The competition may come in the form of network traffic, CPU utilization or memory allocation.</p>
Performance Testing	<p>Excerpt From: http://www.exforsys.com/tutorials/testing/testing-types.html</p> <p>Testing with the intent of determining how quickly a product handles a variety of events. Automated test tools geared specifically to test and fine-tune performance are used most often for this type of testing.</p>
Pilot Testing	<p>Excerpt From: http://www.exforsys.com/tutorials/testing/testing-types.html</p> <p>Testing that involves the users just before actual release to ensure that users become familiar with the release contents and ultimately accept it. Often is considered a Move-to-Production activity for ERP releases or a beta test for commercial products. Typically involves many users, is conducted over a short period of time and is tightly controlled. (see beta testing)</p>
Regression Testing	<p>Excerpt From: http://www.exforsys.com/tutorials/testing/testing-types.html</p> <p>Testing with the intent of determining if bug fixes have been successful and have not created any new problems. Also, this type of testing is done to ensure that no degradation of baseline functionality has occurred.</p>
Security Testing	<p>Excerpt From: http://www.exforsys.com/tutorials/testing/testing-types.html</p> <p>Testing of database and network software in order to keep company data and resources secure from mistaken/accidental users, hackers, and other malevolent attackers.</p>
Software Testing	<p>Excerpt From: http://www.exforsys.com/tutorials/testing/testing-types.html</p> <p>The process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and doesn't fail in an unacceptable manner. The organization and management of individuals or groups doing this work is not relevant. This term is often applied to commercial products such as internet applications. (contrast with independent verification and validation)</p>

TERM	TEST TYPE DEFINITION
Stress Testing	<p>Excerpt From: http://www.exforsys.com/tutorials/testing/testing-types.html</p> <p>Testing with the intent of determining how well a product performs when a load is placed on the system resources that nears and then exceeds capacity.</p>
System Integration Testing	<p>Testing a specific hardware/software installation. This is typically performed on a COTS (commercial off the shelf) system or any other system comprised of disparent parts where custom configurations and/or unique installations are the norm.</p>
System Test	<p>Excerpt From: http://www.exforsys.com/tutorials/testing/testing-types.html</p> <p>Several modules constitute a project. If the project is long-term project, several developers write the modules. Once all the modules are integrated, several errors may arise. The testing done at this stage is called system test.</p> <p>System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.</p> <p>Testing a specific hardware/software installation. This is typically performed on a COTS (commercial off the shelf) system or any other system comprised of disparent parts where custom configurations and/or unique installations are the norm.</p>
Unit Test	<p>Excerpt From: http://www.exforsys.com/tutorials/testing/testing-types.html</p> <p>The first test in the development process is the unit test. The source code is normally divided into modules, which in turn are divided into smaller units called units. These units have specific behavior. The test done on these units of code is called unit test. Unit test depends upon the language on which the project is developed. Unit tests ensure that each unique path of the project performs accurately to the documented specifications and contains clearly defined inputs and expected results.</p>
Unit Testing	<p>Testing of individual hardware or software units or groups of related units</p> <p>-- <i>Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY: 1990.</i></p>
Unit Testing (Unit)	<p>A "unit" (as defined in the software testing literature) is the smallest piece of code software that can be tested in isolation.</p> <ol style="list-style-type: none"> 1 "In isolation" means separate and apart from the application, and all other units. 2 The usual connotations associated with a unit are that <ol style="list-style-type: none"> a) It is a compilation unit recognized by the compiler b) It occupies a single file c) It is small (in "lines of code" sense) d) It is atomic, i.e., you normally do not think of breaking the unit into smaller pieces 3 Units, in object-oriented software, are classes, metaclasses, parameterized classes, and their corresponding instances. In programming languages that allow them, stand-alone procedures and functions are also units.

TERM	TEST TYPE DEFINITION
Unit Testing (Isolation)	<p>Unit testing (as defined in the software testing literature) requires that we test the unit in isolation. That is, we want to be able to say, to a very high degree of confidence, that any actual results obtained from the execution of test cases are purely the result of the unit under test. The introduction of other units may color our results.</p> <p>Since we can seldom, if ever, test a unit without the presence of at least some other software, unit testing involves such things as "drivers" and "stubs."</p> <p>Traditionally, a driver is a piece of software that controls (drives) the unit being tested. Drivers are usually thought of as invoking, or at least containing, the unit being tested, i.e., units (being tested) are subordinate to their respective drivers.</p> <p>Sometimes, units require the presence of other subordinate (to the unit) software. Traditionally, a stub is a piece of software that both mimics the characteristics of, and is (hopefully much) simpler than, a necessary piece of software that is immediately subordinate to the unit being tested.</p>
Unit Testing (Driver)	<p>Drivers are programs or tools that allow a tester to exercise/ examine in a controlling manner the unit of software being tested. A driver is usually expected to provide the following:</p> <ul style="list-style-type: none"> ▪ a means of defining, declaring, or otherwise creating, any variables, constants, or other items needed in the testing of the unit, and a means of monitoring the states of these items, ▪ any input and output mechanisms needed in the testing of the unit, ▪ a means of controlling the unit being tested, e.g., deciding when the unit will be invoked, and what information will be supplied upon invocation, ▪ the generation and/or handling of any necessary interrupts, and ▪ the generation and/or handling of any necessary exceptions.
Unit Testing (Stubs)	<p>Stubs are program units that are stand-ins for the other (more complex) program units that are directly referenced by the unit being tested. Stubs are usually expected to provide the following:</p> <ul style="list-style-type: none"> ▪ an interface that is identical to the interface that will be provided by the actual program unit, and ▪ the minimum acceptable behavior expected of the actual program unit. (This can be as simple as a return statement.) <p>The goal is to keep both drivers and stubs at a minimum level of complexity. If a driver or stub becomes too complex:</p> <ul style="list-style-type: none"> ▪ it will have to be formally tested itself, and ▪ the risk of the driver or stub masking, or contributing to, an error increases to an unacceptable level. <p>The simplest driver is indeed more complex than the simplest stub. However, drivers tend to reach a (manageable) maximum level of complexity and remain there, whereas it is not uncommon to see the complexity of (at least some) stubs come very close to the complexity of the units that they are supposed to be representing.</p> <p>Lastly, there are tools (e.g., test bed generators and test harnesses) that can automate the generation of drivers and stubs.</p>
User Acceptance Testing	<p>Excerpt From: http://www.exforsys.com/tutorials/testing/testing-types.html</p> <p>See Acceptance Testing.</p>

TERM	TEST TYPE DEFINITION
White Box Testing	Excerpt From: http://www.exforsys.com/tutorials/testing/testing-types.html Testing in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose.

Draft

3 REQUIREMENTS

This section shall be divided into the following paragraphs to specify the requirements imposed on one or more systems, subsystems, configuration items, manual operations, or other system components to achieve one or more interfaces among these entities. 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 interfacing entities that are conditions for their acceptance; defer to design descriptions those characteristics that the acquirer is willing to leave up to the developer. If a given requirement fits into more than one paragraph, it may be stated once and referenced from the other paragraphs. If an interfacing entity included in this specification will operate in states and/or modes having interface requirements different from other states and modes, each requirement or group of requirements for that entity 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.

3.1 Interface Identification and Diagrams

For each interface identified in 1.1, this paragraph 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.

3.1.1 Architecture

There are two architectural views of the design:

- ***Stand Alone System Architecture*** (on page 53) - Description of the components of and relationship between components of an isolated system operating by itself.
- ***Stand Among System Architecture*** (on page 57) - Description of the components of and relationship between two or more networked systems operating in collaboration with each other.

3.1.1.1 "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"

Draft

Graphical-style User Interface (tsToolkitGUI)	
<ul style="list-style-type: none"> The "tsToolsGUI" is a set of graphical-style user interface application programs for tracking software development metrics. 	<ul style="list-style-type: none"> The "tsTestsGUI" is a set of graphical-style user interface application programs for regression testing and tutorial demos.
<ul style="list-style-type: none"> The "tsLibGUI" is a library of graphical-style user interface building blocks that establishes the emulated run time environment for the high-level, pixel-mode, "wxPython" GUI Toolkit via the low-level, character-mode, "nCurses" Text User Interface Toolkit. 	

^ ^ |
 | | |
 | | v

Command Line Interface (tsToolkitCLI)	
<ul style="list-style-type: none"> The "tsToolsCLI" is a set of command line interface application programs and utility scripts for: checking source code syntax and style via "plint"; generating Unix-style "man page" documentation from source code comments via "pydoc"; and installing, modifying for publication, and tracking software development metrics. 	<ul style="list-style-type: none"> The "tsTestsCLI" is a set of command line interface application programs and scripts for regression testing and tutorial demos.
<ul style="list-style-type: none"> The "tsLibCLI" is a library of command line building blocks that establishes the POSIX-style, run time environment for pre-processing source files, launching application programs, handling events (registering events with date, time and event severity annotations) and configuring console terminal and file system input and output. 	
<ul style="list-style-type: none"> The "tsUtilities" is a library of computer system configuration, diagnostic, installation, maintenance and performance tool components for various host hardware and software platforms. 	

^ ^ |
 | | +- > Operator Display & Log Files
 | +----- Operator Keyboard
 +----- Operator Mouse

3.1.1.2 Stand Alone System Architecture

The *Team*STARS "tsWxGTUI_PyVx" Toolkit provides:

1. Python version-specific components for its Command Line Interface ("tsToolkitCLI")
2. Python version-specific components for its Graphical-style User Interface ("tsToolkitGUI").

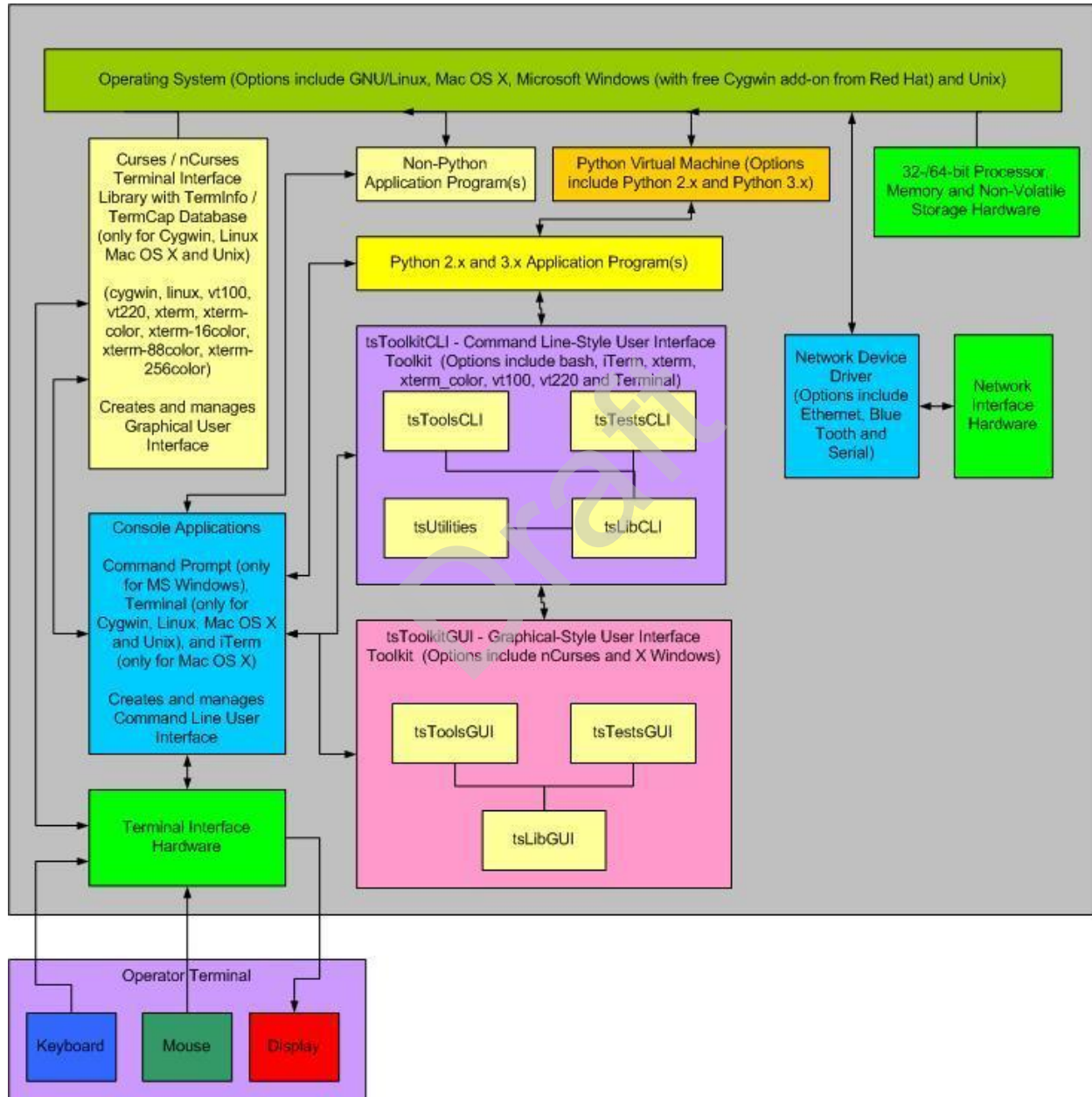
The following description uses the component names as depicted in the Block Diagram

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.

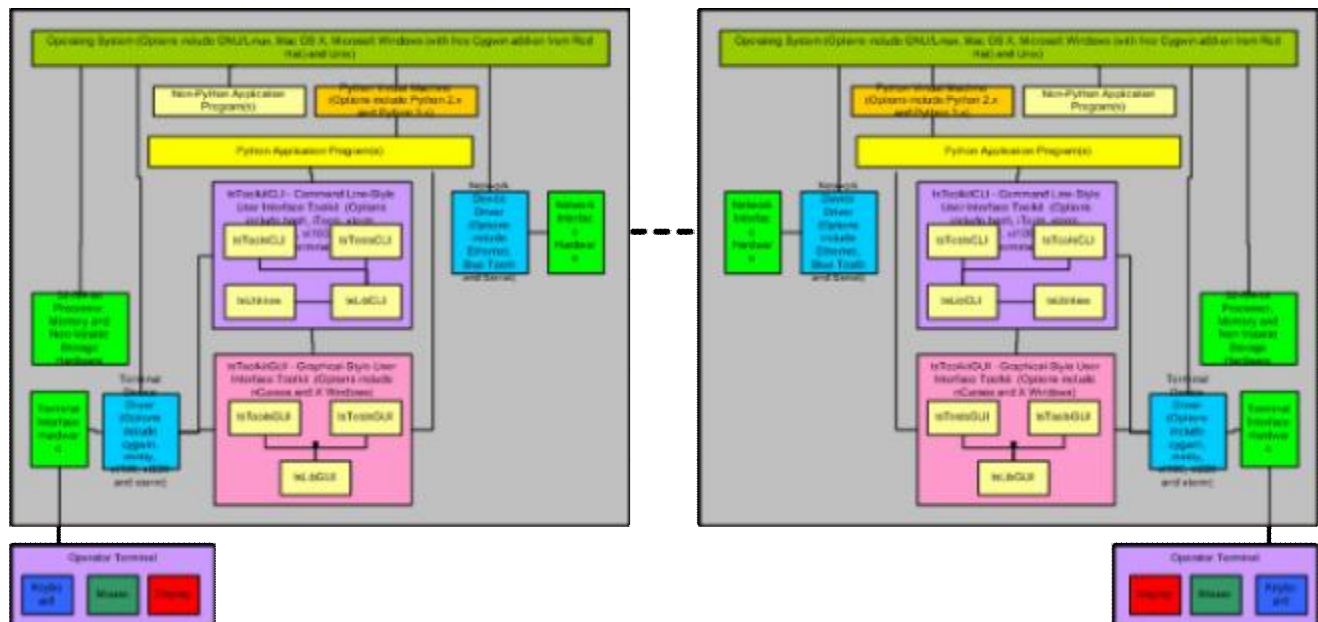
3.1.1.3 Stand Among System Architecture

The *TeamSTARS* "tsWxGTUI_PyVx" Toolkit provides:

1. Python version-specific components for its Command Line Interface ("tsToolkitCLI")
2. Python version-specific components for its Graphical-style User Interface ("tsToolkitGUI").

The following description uses the component names as depicted in the Block Diagram

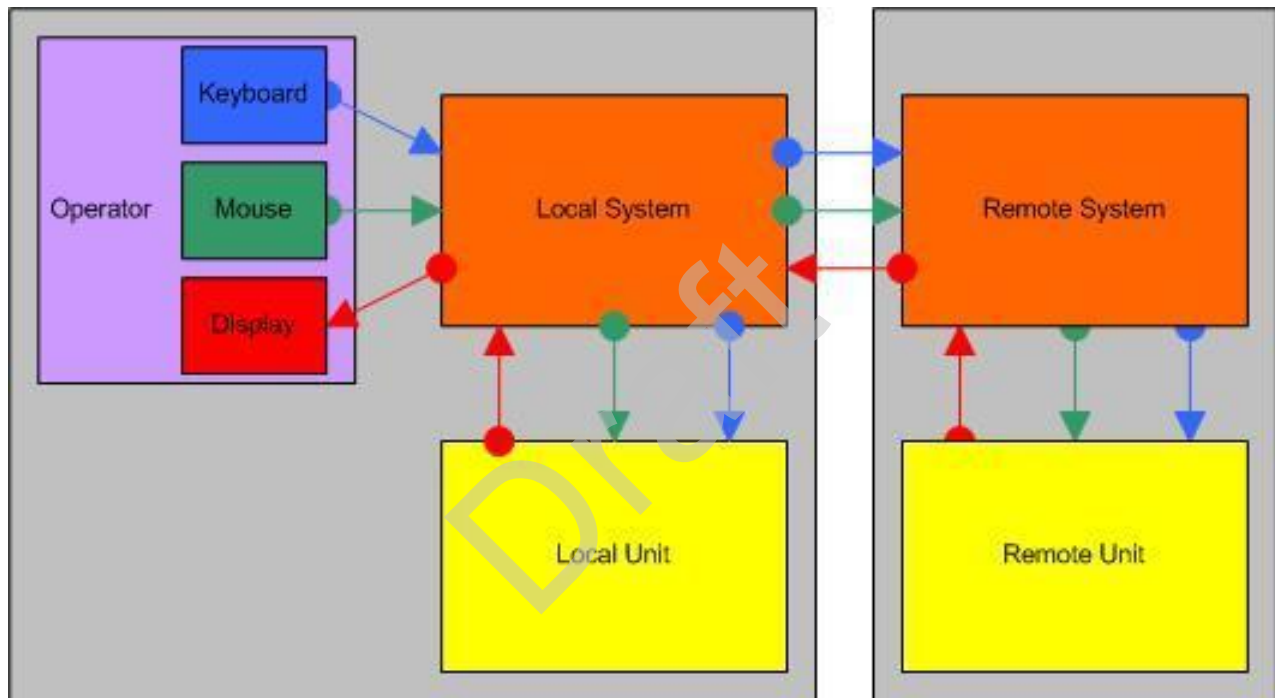
The *Stand Alone System Architecture* (on page 53), as previously described, may be extended to enable a single operator, working from a Local System, to interact with one or more Remote Systems.



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 59) 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 (CentOS 7 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.

3.1.1.3.1 Multi-Session Desktop

From Wikipedia, the free encyclopedia

"In computer science, in particular networking, a session is a semi-permanent interactive information interchange, also known as a dialogue, a conversation or a meeting, between two or more communicating devices, or between a computer and user (see Login session). A session is set up or established at a certain point in time, and then torn down at some later point. An established communication session may involve more than one message in each direction. A session is typically, but not always, stateful, meaning that at least one of the communicating parts needs to save information about the session history in order to be able to communicate, as opposed to stateless communication, where the communication consists of independent requests with responses.

An established session is the basic requirement to perform a connection-oriented communication. A session also is the basic step to transmit in connectionless communication modes. However any unidirectional transmission does not define a session.[1]

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

1 Application layer examples:

- a) HTTP sessions, which allow associating information with individual visitors
- b) A telnet remote login session

2 Session layer example:

- a) A Session Initiation Protocol (SIP) based Internet phone call

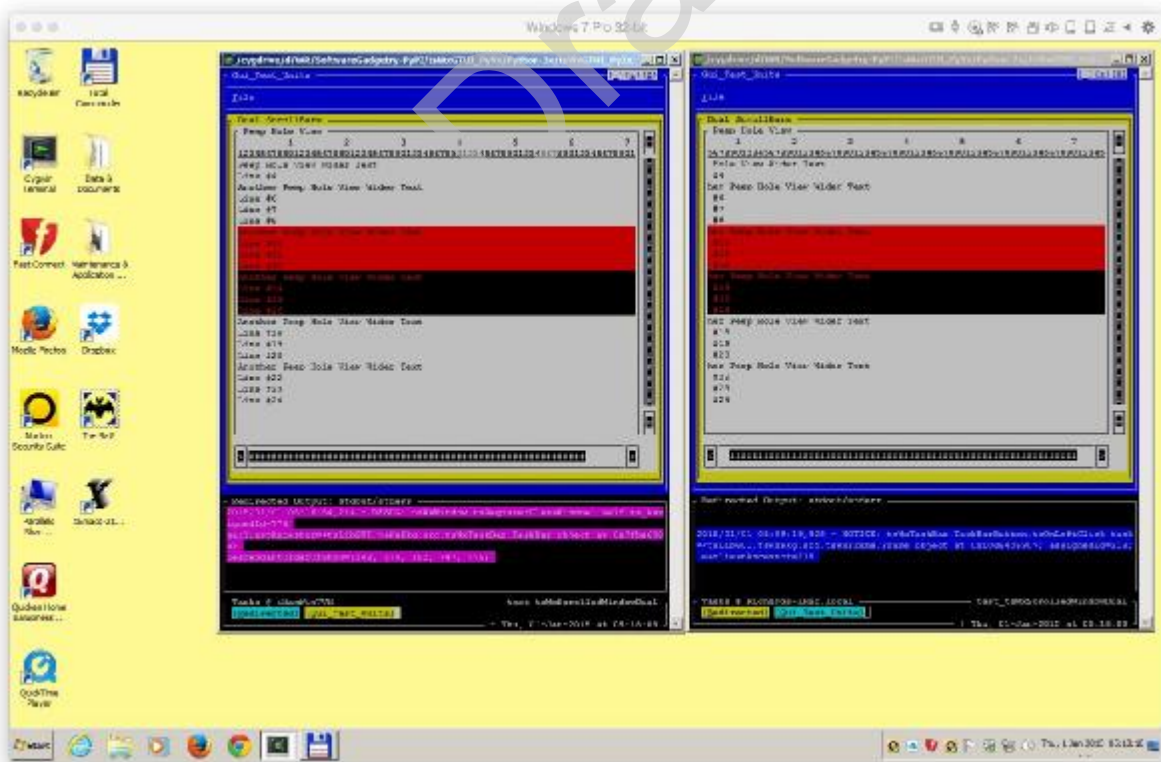
3 Transport layer example:

- a) A TCP session, which is synonymous to a TCP virtual circuit, a TCP connection, or an established TCP socket.

In the case of transport protocols that do not implement a formal session layer (e.g., UDP) or where sessions at the application layer are generally very short-lived (e.g., HTTP), sessions are maintained by a higher level program using a method defined in the data being exchanged. For example, an HTTP exchange between a browser and a remote host may include an HTTP cookie which identifies state, such as a unique session ID, information about the user's preferences or authorization level.

HTTP/1.0 was thought to only allow a single request and response during one Web/HTTP Session. However a workaround was created by David Hostettler Wain in 1996 such that it was possible to use session IDs to allow multiple phase Web Transaction Processing (TP) Systems (in ICL[disambiguation needed] nomenclature), with the first implementation being called Deity. Protocol version HTTP/1.1 further improved by completing the Common Gateway Interface (CGI) making it easier to maintain the Web Session and supporting HTTP cookies and file uploads.

Most client-server sessions are maintained by the transport layer - a single connection for a single session. However each transaction phase of a Web/HTTP session creates a separate connection. Maintaining session continuity between phases required a session ID. The session ID is embedded within the <A HREF> or <FORM> links of dynamic web pages so that it is passed back to the CGI. CGI then uses the session ID to ensure session continuity between transaction phases. One advantage of one connection-per-phase is that it works well over low bandwidth (modem) connections. Deity used a sessionID, screenID and actionID to simplify the design of multiple phase sessions."



The Sample Screenshot above shows a Windows 7 Desktop with:

- 1** A local Windows host with Python 3x session on left; and
- 2** A remote Mac OS X host with Python 2x session on right.
- 3** Each session consists of
 - a) a wxPython-style Frame named "Dual ScrollBars" containing scrollable text with optional color markup.
 - b) a wxPython-style Frame named "Redirected Output" containing date and time stamped event messages, with optional with color markup, that scroll up when new events are registered.
 - c) a Host Desktop-style Frame named "Tasks @ Host Name" and Application Name with buttons to shift focus from background to foreground. There is also a spinner (to indicate the frequency or absence of idle time) and the current date and time (to indicate when the display was last updated).

Draft

3.2 (Project Unique Identifier Of Interface Template)

This paragraph (beginning with 3.2) shall identify an 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 one or more of the interfacing entities to achieve the interface. If the interface characteristics of an entity are not covered by this IRS but need to be mentioned to specify the requirements for entities that are, those characteristics shall be stated as assumptions or as "When [the entity not covered] does this, the [entity being specified] shall...", rather than as requirements on the entities not covered by this IRS. 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 interfacing entity(ies) 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 interfacing entity(ies) 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 interfacing entity(ies) 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 interfacing entity(ies) 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 interfacing entity(ies) 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.
-

3.3 POSIX-style Desktop Environments (DE)

Excerpt From Wikipedia, the free encyclopedia

"In computing, a desktop environment (DE) is an implementation of the desktop metaphor made of a bundle of programs running on top of a computer operating system, which share a common graphical user interface (GUI), sometimes described as a graphical shell. The desktop environment was seen mostly on personal computers until the rise of mobile computing. Desktop GUIs help the user to easily access and edit files, while they usually do not provide access to all of the features found in the underlying operating system. Instead, the traditional command-line interface (CLI) is still used when full control over the operating system is required.

A desktop environment typically consists of icons, windows, toolbars, folders, wallpapers and desktop widgets (see Elements of graphical user interfaces and WIMP). A GUI might also provide drag and drop functionality and other features that make the desktop metaphor more complete. A desktop environment aims to be an intuitive way for the user to interact with the computer using concepts which are similar to those used when interacting with the physical world, such as buttons and windows.

While the term desktop environment originally described a style of user interfaces following the desktop metaphor, it has also come to describe the programs that realize the metaphor itself. This usage has been popularized by projects such as the Common Desktop Environment, K Desktop Environment, and GNOME."

"Implementation

On a system that offers a desktop environment, a window manager in conjunction with applications written using a widget toolkit are generally responsible for most of what the user sees. The window manager supports the user interactions with the environment, while the toolkit provides developers a software library for applications with a unified look and behavior.

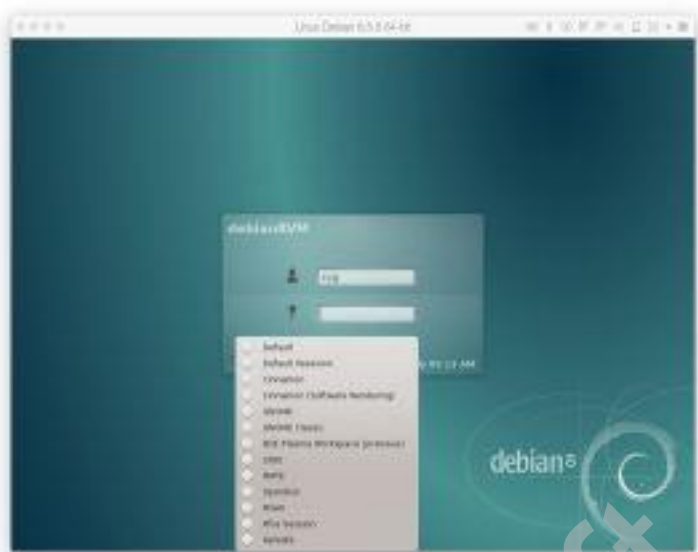
A windowing system of some sort generally interfaces directly with the underlying operating system and libraries. This provides support for graphical hardware, pointing devices, and keyboards. The window manager generally runs on top of this windowing system. While the windowing system may provide some window management functionality, this functionality is still considered to be part of the window manager, which simply happens to have been provided by the windowing system.

Applications that are created with a particular window manager in mind usually make use of a windowing toolkit, generally provided with the operating system or window manager. A windowing toolkit gives applications access to widgets that allow the user to interact graphically with the application in a consistent way."


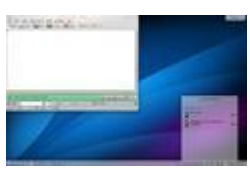



One can experience the look and feel of various Desktop Environments after using the Net Installer version of Debian Linux 8.5.0. It enables a System Administrator, or fearless user, to conveniently select and install one, several or all of the following Graphical User Interface (GUI) Desktop Environments. Once installed, a user may choose the Desktop Environment during login by:

1. Clicking on the Desktop Settings (Toothed Gear in figure below) icon within the login window

2. Selecting the desired Desktop Environment (Checkbox in figure below)
3. Entering the user name (Dialog Box after Head and Shoulder Icon) and password (Dialog Box after Key Icon)



Excerpt from *Debian Desktop Environments* (<https://wiki.debian.org/DesktopEnvironment>)

	The GNOME project provides two things: The GNOME desktop environment, an intuitive and attractive desktop for users, and the GNOME development platform, an extensive framework for building applications that integrate into the rest of the desktop.
	KDE is a powerful open source graphical desktop environment for Unix workstations. It combines ease of use, contemporary functionality, and outstanding graphical design with the technological superiority of the Unix operating system.
	Xfce is a lightweight desktop environment for various *NIX systems. Designed for productivity, it loads and executes applications quickly, while conserving system resources.
	LXDE is designed to work well with computers on the low end of the performance spectrum such as older resource-constrained machines, new generation netbooks, and other small computers.
	MATE is the continuation of GNOME 2. It provides an intuitive and attractive desktop environment using traditional metaphors for Linux and other Unix-like operating systems.
Other desktop environments available in Debian include cinnamon, lxqt, Enlightenment, fvwm-crystal, windowmaker, Sugar and possibly others.	
Other desktop environments not available in Debian include Unity, Budgie, Pantheon, ROX, Equinox/EDE, Étoilé, CDE and others.	

3.4 Simplified Wrapper and Interface Generator (SWIG)

This topic has been included because it offers the most likely means to solve limitations with the existing Python Curses module:

1. The existing Python Curses module supports only a small subset of the C-language based 32-bit "nCurses" 5.x Application Programming Interface. Manually created by a suitable skilled developer, it cannot support more than 16-colors and 256-color pairs. The 32-bit "nCurses" 5.x Application Programming Interface subset does not appear to have ever been enhanced and it would be difficult for an inexperienced developer to do so.
 2. The recently released C-language based 64-bit "nCurses" 6.x Application Programming Interface now supports more than 256-colors and 16 million-color pairs. It also has a 32-bit version counterpart for backward compatibility with older platforms with support for up to 16-colors and 256-color pairs.
 3. The SWIG technology can automatically generate a Python module, from a SWIG Interface file. It includes the host platform's C language header file. It generates a Python module that will contain a Python-style Application Programming Interface with all of the functionality of the associated C-language header file.
-

From Wikipedia, the free encyclopedia: (*SWIG* (<https://en.wikipedia.org/wiki/SWIG>))

"SWIG Original author(s) Dave Beazley

Developer(s) SWIG developers

Initial release February 1996[1]

Stable release 3.0.9 / May 29, 2016; 50 days ago

Written in C, C++

Operating system Cross-platform

License GPL

Website swig.org

The Simplified Wrapper and Interface Generator (SWIG) is an open-source software tool used to connect computer programs or libraries written in C or C++ with scripting languages such as Lua, Perl, PHP, Python, R, Ruby, Tcl, and other languages like C#, Java, JavaScript, Go, Modula-3, OCaml, Octave, Scilab and Scheme. Output can also be in the form of XML or Lisp S-expressions."

3.4.1 Function

From Wikipedia, the free encyclopedia: (*SWIG* (<https://en.wikipedia.org/wiki/SWIG>))

"The aim is to allow the calling of native functions (that were written in C or C++) by other programming languages, passing complex data types to those functions, keeping memory from being inappropriately freed, inheriting object classes across languages, etc. The programmer writes an interface file containing a list of C/C++ functions to be made visible to an interpreter. SWIG will compile the interface file and generate code in regular C/C++ and the target programming language. SWIG will generate conversion code for functions with simple arguments; conversion code for complex types of arguments must be written by the programmer. The SWIG tool creates source code that provides the glue between C/C++ and the target language. Depending on the language, this glue comes in two forms:

- a shared library that an extant interpreter can link to as some form of extension module, or
- a shared library that can be linked to other programs compiled in the target language (for example, using Java Native Interface (JNI) in Java).

SWIG is not used for calling interpreted functions by native code; this must be done by the programmer manually."

3.4.2 Example

From Wikipedia, the free encyclopedia: (*SWIG* (<https://en.wikipedia.org/wiki/SWIG>))

"SWIG wraps simple C declarations by creating an interface that closely matches the way in which the declarations would be used in a C program. For example, consider the following interface file:

```
%module example
%inline %{
extern double sin(double x);
extern int strcmp(const char *, const char *);
extern int Foo;
%}
#define STATUS 50
#define VERSION "1.1"
```

In this file, there are two functions `sin()` and `strcmp()`, a global variable `Foo`, and two constants `STATUS` and `VERSION`. When SWIG creates an extension module, these declarations are accessible as scripting language functions, variables, and constants respectively. In Python:

```
>>> example.sin(3)
0.141120008
>>> example strcmp('Dave','Mike')
-1
```

```
>>> print example.cvar.Foo
42
>>> print example.STATUS
50
>>> print example.VERSION
1.1"
```

3.4.3 Purpose

From Wikipedia, the free encyclopedia: (*SWIG* (<https://en.wikipedia.org/wiki/SWIG>))

""There are two main reasons to embed a scripting engine in an existing C/C++ program:

The program can then be customized far faster, via a scripting language instead of C/C++. The scripting engine may even be exposed to the end user, so that they can automate common tasks by writing scripts.

Even if the final product is not to contain the scripting engine, it may nevertheless be very useful for writing test scripts.

There are several reasons to create dynamic libraries that can be loaded into extant interpreters, including:

Provide access to a C/C++ library which has no equivalent in the scripting language.

Write the whole program in the scripting language first, and after profiling, rewrite performance critical code in C or C++."

3.4.4 History

From Wikipedia, the free encyclopedia: (*SWIG* (<https://en.wikipedia.org/wiki/SWIG>))

"SWIG is written in C and C++ and has been publicly available since February 1996. The initial author and main developer was Dave Beazley who developed SWIG while working as a graduate student at Los Alamos National Laboratory and the University of Utah and while on the faculty at the University of Chicago. Development is currently supported by an active group of volunteers led by William Fulton. SWIG has been released under a GNU General Public License."

3.4.5 SWIG Tutorial

From: *SIG Tutorial* (<http://www.swig.org/tutorial.html>)

"TUTORIAL

So you want to get going in a hurry? To illustrate the use of SWIG, suppose you have some C functions you want added to Tcl, Perl, Python, Java and C#. Specifically, let's say you have them in a file 'example.c'

/* File : example.c */

```
#include <time.h>
double My_variable = 3.0;

int fact(int n) {
    if (n <= 1) return 1;
    else return n*fact(n-1);
}

int my_mod(int x, int y) {
    return (x%y);
}

char *get_time()
{
    time_t ltime;
    time(&ltime);
    return ctime(&ltime);
}
```

Interface file

Now, in order to add these files to your favorite language, you need to write an "interface file" which is the input to SWIG. An interface file for these C functions might look like this :

```
/* example.i */
%module example
%{
    /* Put header files here or function declarations like below */
    extern double My_variable;
    extern int fact(int n);
    extern int my_mod(int x, int y);
    extern char *get_time();
}%

extern double My_variable;
extern int fact(int n);
extern int my_mod(int x, int y);
extern char *get_time();
```

Building a Tcl module

At the UNIX prompt, type the following (shown for Linux, see the SWIG Wiki Shared Libraries page for help with other operating systems):

```
unix % swig -tcl example.i
unix % gcc -fpic -c example.c example_wrap.c \
-I/usr/local/include
unix % gcc -shared example.o example_wrap.o -o example.so
unix % tclsh
% load ./example.so example
% puts $My_variable
3.0
% fact 5
```

```
120
% my_mod 7 3
1
% get_time
Sun Feb 11 23:01:07 1996

%
```

The `swig` command produces a file `example_wrap.c` that should be compiled and linked with the rest of the program. In this case, we have built a dynamically loadable extension that can be loaded into the Tcl interpreter using the `'load'` command.

Building a Python module

Turning C code into a Python module is also easy. Simply do the following (shown for Irix, see the SWIG Wiki Shared Libraries page for help with other operating systems):

```
unix % swig -python example.i
unix % gcc -c example.c example_wrap.c \
-I/usr/local/include/python2.1
unix % ld -shared example.o example_wrap.o -o _example.so
```

We can now use the Python module as follows :

```
>>> import example
>>> example.fact(5)
120
>>> example.my_mod(7,3)
1
>>> example.get_time()
'Sun Feb 11 23:01:07 1996'
>>>
```

Building a Perl module

You can also build a Perl5 module as follows (shown for Linux, see the SWIG Wiki Shared Libraries page for help with other operating systems):

```
unix % swig -perl5 example.i
unix % gcc -c `perl -MConfig -e 'print join(" ", @Config{qw(ccflags optimize
cccdlflags)}), \
"-I$Config{archlib}/CORE")'` example.c example_wrap.c
unix % gcc `perl -MConfig -e 'print $Config{lddlflags}'` example.o
example_wrap.o -o example.so
unix % perl
use example;
print $example::My_variable,"\n";
print example::fact(5),"\n";
print example::get_time(),"\n";
<ctrl-d>
3.0
120
Sun Feb 11 23:01:07 1996
unix %
```

Building a Java module

SWIG will also generate JNI code for accessing C/C++ code from Java. Here is an example building a Java module (shown for Cygwin, see the SWIG Wiki Shared Libraries page for help with other operating systems):

```
$ swig -java example.i
$ gcc -c example.c example_wrap.c -I/c/jdk1.3.1/include -
I/c/jdk1.3.1/include/win32
$ gcc -shared example.o example_wrap.o -mno-cygwin -Wl,--add-stdcall-alias -o
example.dll
$ cat main.java
public class main {
public static void main(String argv[]) {
System.loadLibrary("example");
System.out.println(example.getMy_variable());
System.out.println(example.fact(5));
System.out.println(example.get_time());
}
}
$ javac main.java
$ java main
3.0
120
Mon Mar 4 18:20:31 2002
$
```

Building a C# module

SWIG will also generate code for accessing C/C++ code from C# using PInvoke. Here is an example building a C# module (shown for Linux, see the SWIG Wiki Shared Libraries page for help with other operating systems). It uses the open source DotGNU Portable.NET C# compiler which runs on most Unix systems, but works equally well using other C# compilers:

```
$ swig -csharp example.i
$ gcc -c -fpic example.c example_wrap.c
$ gcc -shared example.o example_wrap.o -o libexample.so
$ csc -o runme *.cs
$ cat runme.cs
using System;
public class runme {
static void Main() {
Console.WriteLine(example.My_variable);
Console.WriteLine(example.fact(5));
Console.WriteLine(example.get_time());
}
}
$ ilrun runme
3
120
Tue May 13 10:45:45 2003
$
```

SWIG for the truly lazy

As it turns out, it is not always necessary to write a special interface file. If you have a header file, you can often just include it directly in the SWIG interface. For example:

```
%module example
%{
/* Includes the header in the wrapper code */
#include "header.h"
%}

/* Parse the header file to generate wrappers */
#include "header.h"
```

Alternatively, some people might just include SWIG directives in a header file with conditional compilation. For example:

```
#ifdef SWIG
%module example
%{
#include "header.h"
%}
#endif

extern int fact(int n);
...
```

Running SWIG under Microsoft Windows

SWIG also works perfectly well under all known 32 bit versions of Windows including 95/98/NT/2000/XP. SWIG is typically invoked from the command prompt and can be used with NMAKE. Modules are typically compiled in the form of a DLL that can be dynamically loaded into Tcl, Python, or whatever language you are using. With a little work, SWIG can also be used as a custom build option within MS Developer Studio.

That's it (well, more or less)

That's about everything you need to know to get started. Here's the short checklist :

- Make sure you specify a module name.
- Use ANSI C/C++ syntax
- Figure out how to compile a shared library module / dynamic link library (may require reading a few man pages for your compiler).
- Relax.

Surely there's more to it...

The above example is intentionally simple, but the general idea extends to more complicated C/C++ programming tasks. In fact, it is important to know that SWIG is a fairly complete C++ compiler with support for nearly every language feature. This includes preprocessing, pointers, classes, inheritance, and even C++ templates. SWIG can also be used to package structures and classes into proxy classes in the target language---exposing the underlying functionality in a very natural manner.

To illustrate, suppose you wanted to wrap the following C++ data structure:

```
// pair.h. A pair like the STL
namespace std {
  template<class T1, class T2> struct pair {
    T1 first;
    T2 second;
    pair() : first(T1()), second(T2()) { };
    pair(const T1 &f, const T2 &s) : first(f), second(s) { }
  };
}
```

To wrap with SWIG, you might specify the following interface:

```
// pair.i - SWIG interface
%module pair
%{
#include "pair.h"
%}

// Ignore the default constructor
%ignore std::pair::pair();

// Parse the original header file
#include "pair.h"

// Instantiate some templates

%template(pairii) std::pair<int,int>;
%template(pairdi) std::pair<double,int>;
```

Now, compiling (Python):

```
$ swig -python -c++ pair.i
$ c++ -c pair_wrap.c -I/usr/local/include/python2.1
$ c++ -shared pair_wrap.o -o _pair.so
$ python
Python 2.1 (#3, Aug 20 2001, 15:41:42)
[GCC 2.95.2 19991024 (release)] on sunos5
Type "copyright", "credits" or "license" for more information.
>>> import pair
>>> a = pair.pairii(3,4)
>>> a.first
3
>>> a.second
4
>>> a.second = 16
>>> a.second
16
```

```
>>> b = pair.pairdi(3.5,8)
>>> b.first
3.5
>>> b.second
8
```

Feedback and questions concerning this site should be posted to the swig-devel mailing list.

Last modified : Sun May 29 15:22:36 2016"

3.5 Terminal Device Interface (TDI)

The terminal consists of the following devices:

- **Display** (on page 76) - Output is presented to the operator via a terminal's display. A VGA terminal has a display resolution of 640 x 480 pixels. With a font of 8 x 12 pixels, it can show 80 columns x 40 rows.
- **Keyboard** (on page 91) - Input is entered by the operator via the terminal's keyboard. A typical keyboard consists of a mechanical typewriter style matrix of push buttons. The operator may press a letter, digit, punctuation symbol or function key.
- **Mouse** (on page 91) - Point and click input is entered by the operator via the terminal's mouse, touchscreen, trackpad or trackball. The input consists of display column and row coordinates and the clicked, depressed or released state of any associated left, center and right buttons.

3.5.1 Display

This is a required electronic device that outputs alpha-numeric, graphic (or graphic-style) and control data to the computer screen.

3.5.1.1 Display Technology

The computer must fill the display with a matrix of column and row elements. The dimensions of VGA-type displays, for example, are 640 by 480 pixels. For the standard 8 pixel by 12 pixel courier font this is equivalent to 80 columns by 40 rows.

ANSI, VT100 and VT220 Terminal

The display of ANSI , VT100 and VT220 Terminals (*1-Color VT100 via Ubuntu Linux Terminal* (on page 81)) have a single color phosphor that could be "white", "green" or "orange". Text could be displayed in normal mode with the foreground phosphor active ("white" color ON) against the "black" background phosphor inactive ("white" color OFF is equivalent to "black" color ON). In reverse-mode, the activity of the foreground and background were reversed.

ANSI, VT100 and VT220 Terminal Emulators

The display of ANSI , VT100 and VT220 Terminal Emulators (*1-Color VT100 and VT220 with 2nd-Color Highlight via Mac OS X iTerm* (see "*1-Color VT100 with 2nd-Color Highlight via Mac OS X iTerm*" on page 85)) have tri-color phosphors ("red", "green", "blue") that could be activated in various combinations and proportions to create at least an 8-color palette ("black", "blue", "cyan", "green", "magenta", "red", "white", "yellow"). Text could be displayed in normal mode with the foreground palette active ("white" color ON) against the background palette active ("white" color OFF is equivalent to "black" color ON). In reverse-mode, the activity of the foreground and background were reversed. Highlighted text could be in a third color palette ("cyan" or "red" color ON).

XTERM Terminal Emulator

The display of an XTERM Terminal (*8-Color XTERM via Cygwin-X Console* (on page 91)) have tri-color phosphors ("red", "green", "blue") that could be activated in various combinations and proportions to create at least an 8-color palette ("black", "blue", "cyan", "green", "magenta", "red", "white", "yellow"). Text could be displayed in normal mode with the foreground palette active ("white" color ON) against the background palette active ("red" color ON). In reverse-mode, the activity of the foreground and background were reversed.

Derivatives of the xterm terminal emulator include the following:

mintty - A variant of xterm, included in Cygwin, the free GNU/Linux add-on to Microsoft Windows from Red Hat, that can emulate VT100, VT220 and each of the following:

xterm-color - An older branch of xterm that supports sixteen colors, the standard 8 colors in both normal and dim intensity representing 256 color pairs.

xterm-16color - An extension of xterm that supports sixteen colors, the standard 8 colors in both normal and dim intensity representing 256 color pairs.

xterm-88color - An extension of xterm that supports 88 colors representing 7744 color pairs.

xterm-256color - An extension of xterm that supports 181 of 256 colors encompassing only 32761 color pairs.

3.5.1.2 Usage Issues

1 Hardware

- Digital Equipment Co. vt100 terminal or compatible - keyboard, no mouse and 1-color phosphor display
- Digital Equipment Co. vt220 terminal or compatible - keyboard, no mouse and 1-color phosphor display

2 Software (Hardware Emulator)

NOTE: Reproducibility of the following requires the following test procedure: 1) open a new shell; 2) echo \$TERM; 3) TERM=vt100; 4) echo \$TERM; 5) run the python application. Subsequently changing TERM between python runs has been found to leave the emulator in a mixed mouse state in which the mouse could be erroneously enabled but unable to properly interpret its input data.

- ----- Platforms with Ubuntu Linux -----
- **Linux TERMINAL (TERM=xterm)** - keyboard, mouse/wheel mouse/trackball/trackpad with left/center/right buttons, and 8-color phosphor display
- ----- Platforms with Mac OS X -----
- **MacOS iTerm (TERM=vt100)** - keyboard, no mouse/wheel mouse/trackball/trackpad with left/center/right buttons, and 1-color phosphor display
- **MacOS iTerm (TERM=xterm)** - keyboard, mouse/wheel mouse/trackball/trackpad with left/center/right buttons, and 8-color phosphor display
- **MacOS Terminal (TERM=cygwin)** - keyboard, no mouse/wheel mouse/trackball/trackpad with left/center/right buttons, and 8-color phosphor display
- **MacOS Terminal (TERM=vt100)** - keyboard, no mouse/wheel mouse/trackball/trackpad with left/center/right buttons, and 1-color phosphor display
- **MacOS Terminal (TERM=xterm)** - keyboard, no mouse/wheel mouse/trackball/trackpad with left/center/right buttons, and 8-color phosphor display
- ----- Platforms with OpenIndiana Unix -----
- **Solaris (SunOS) Terminal (TERM=xterm)** - keyboard, mouse/wheel mouse/trackball/trackpad with left/center/right buttons, and 8-color phosphor display
- ----- Platforms with Microsoft Windows and Cygwin add-on -----
- **Cygwin console (TERM=vt100)** - keyboard, no mouse and 1-color phosphor display
- **Cygwin console (TERM=cygwin)** - keyboard, no mouse and 8-color phosphor display
- **Cygwin console (TERM=xterm)** - keyboard, mouse/wheel mouse/trackball/trackpad with left/center/right buttons, and 8-color phosphor display
- **Cygwin MinTTY (TERM=cygwin)** - keyboard, no mouse/wheel mouse/trackball/trackpad with left/center/right buttons, and 1-color phosphor display
- **Cygwin MinTTY (TERM=vt100)** - keyboard, no mouse/wheel mouse/trackball/trackpad with left/center/right buttons, and 1-color phosphor display

- **Cygwin MinTTY (TERM=xterm)** - keyboard, mouse/wheel mouse/trackball/trackpad with left/center/right buttons, and 8-color phosphor display

3.5.1.3 Display Options

1 Basic Computer Display for Embedded System Operator

Applications may be as simple as a single frame window. A frame requires: 1) a title line with optional window control buttons; and 2) additional optional lines for a menu bar, tool bar, status bar, buttons, check boxes, radio boxes & buttons, gauges and scrolled text. The display size for this is 35 col x 16 row (280 x 200 pixels).

More complex applications involve multiple side-by-side and overlapping frames which may optionally be arranged in a desktop consisting of: 1) a collection of application frame and dialog windows; 2) a scrolling operator event notification window ; and 3) a taskbar whose buttons enable the operator to raise hidden frames and dialogues from the invisible background to the visible foreground. For the sample splash screen, the display size can range from 60 col x 25 row (480 x 300 pixels) to over 80 col x 50 row (640 x 600 pixels).

A graphic and/or character-mode single font text display (whose size could be somewhere between that of a 3" smart phone and a 12" tablet) that supports the application (or its splash screen) and industry standard terminal emulators:

- a) xterm (8-color, 64-color pairs)
- b) xterm-16color (16-color, 256-color-pairs)
- c) vt100 (1-color ON/OFF, 2-color-pair NORMAL/REVERSE)
- d) vt220 (1-color ON/OFF, 2-color-pair NORMAL/REVERSE)

2 Basic Computer Display for Software Engineering Workstation

A 12" pixel-mode multi-font graphics display that supports at least 80 col x 40 row (640 x 480 pixels) and 16,777,216 colors and industry standard terminal emulators.

- a) xterm (8-color, 64-color pairs)
- b) xterm-16color (16-color, 256-color-pairs)
- c) vt100 (1-color ON/OFF, 2-color-pair NORMAL/REVERSE)
- d) vt220 (1-color ON/OFF, 2-color-pair NORMAL/REVERSE)

3 Optional Computer Display for Software Engineering Workstation

Any one of the following may be substituted for the **Basic Computer Terminal Display** based on your need to simultaneously view multiple software engineering documents and activities.

Derived From Wikipedia, the free encyclopedia at "https://en.wikipedia.org/wiki/Display_resolution"

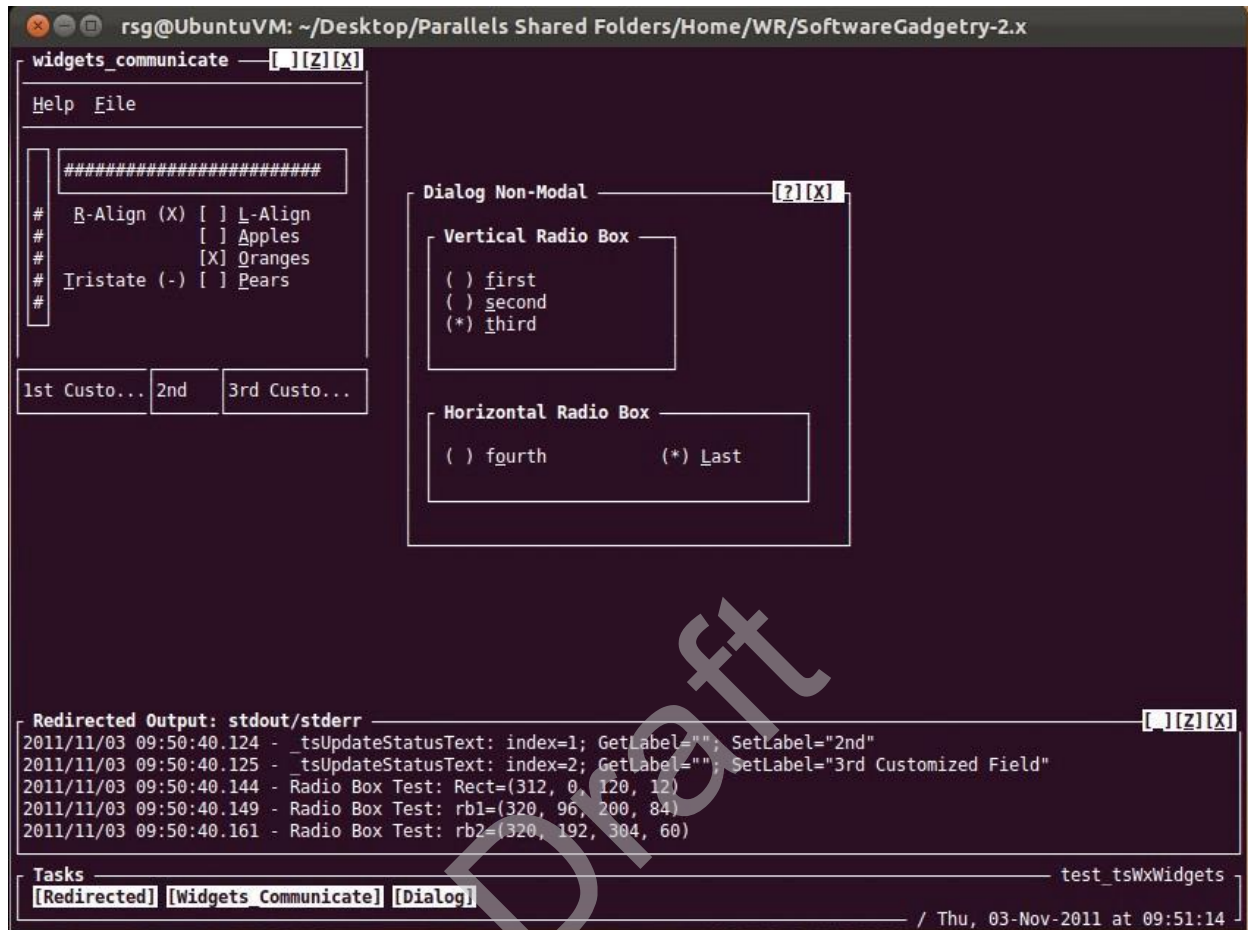
Acronym (usage)	Aspect ratio	Width (pixels)	Height (pixels)
VGA (12" CRT)	4:3	640	480
SVGA (14" CRT)	4:3	800	600
XGA (15" Laptop)	4:3	1024	768

XGA+	4:3	1152	864
WXGA	16:9	1280	720
WXGA	5:3	1280	768
WXGA	16:10	1280	800
SXGA– (UVGA)	4:3	1280	960
SXGA	5:4	1280	1024
HD	~16:9	1360	768
HD	~16:9	1366	768
SXGA+	4:3	1400	1050
WXGA+ (17" Laptop)	16:10	1440	900
HD+	16:9	1600	900
UXGA	4:3	1600	1200
WSXGA+	16:10	1680	1050
FHD	16:9	1920	1080
WUXGA	16:10	1920	1200
QWXGA	16:9	2048	1152
WQHD (27" Desktop)	16:9	2560	1440
WQXGA	16:10	2560	1600

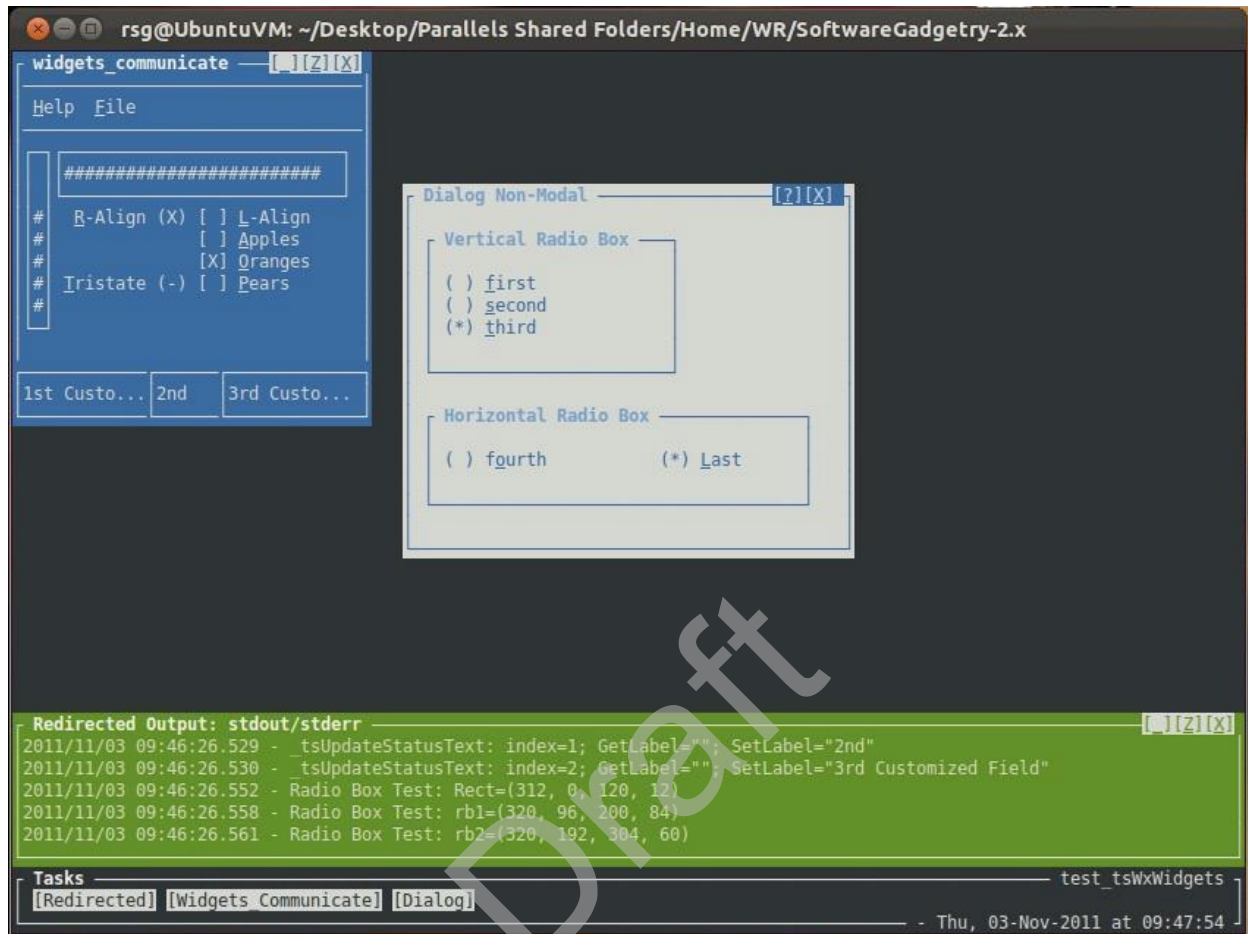
3.5.1.4 Linux (Ubuntu) Platform

- *1-Color VT100 via Ubuntu Linux Terminal* (on page 81)
- *8-Color XTERM via Ubuntu Linux Terminal* (on page 82)

3.5.1.4.1 1-Color VT100 via Ubuntu Linux Terminal



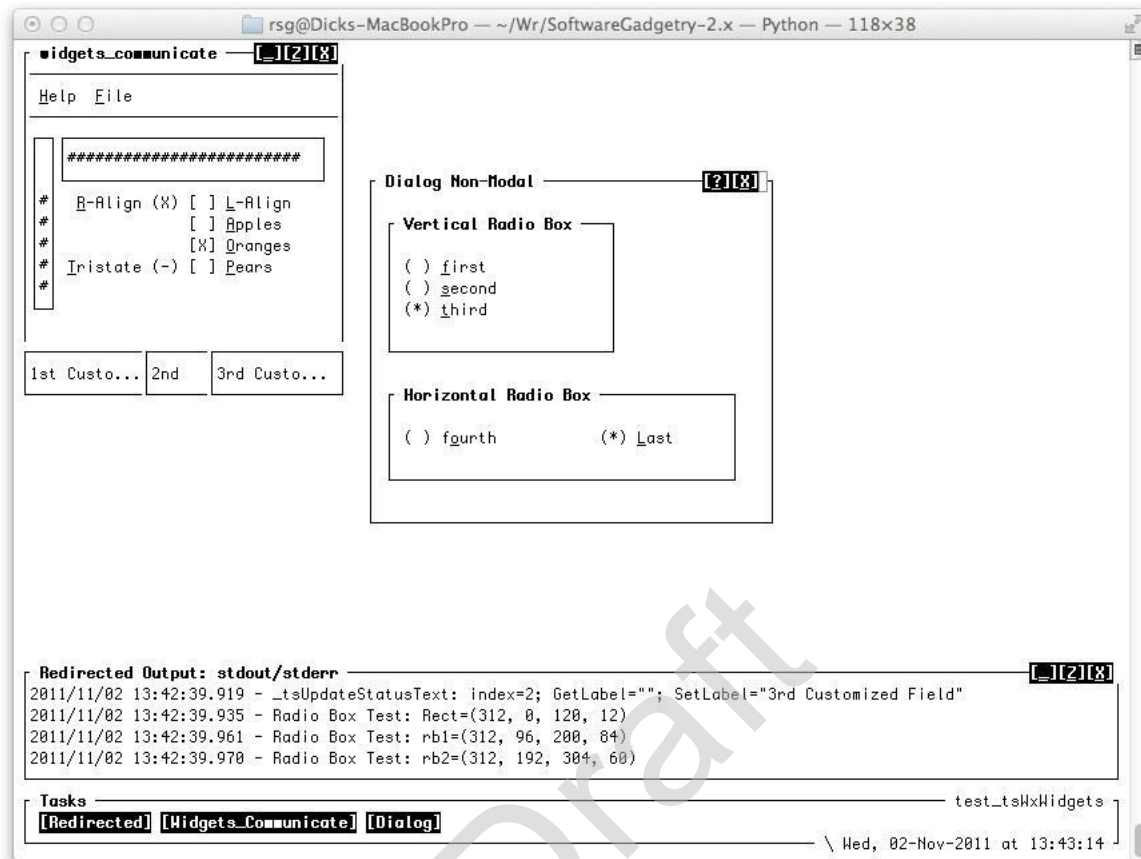
3.5.1.4.2 8-Color XTERM via Ubuntu Linux Terminal



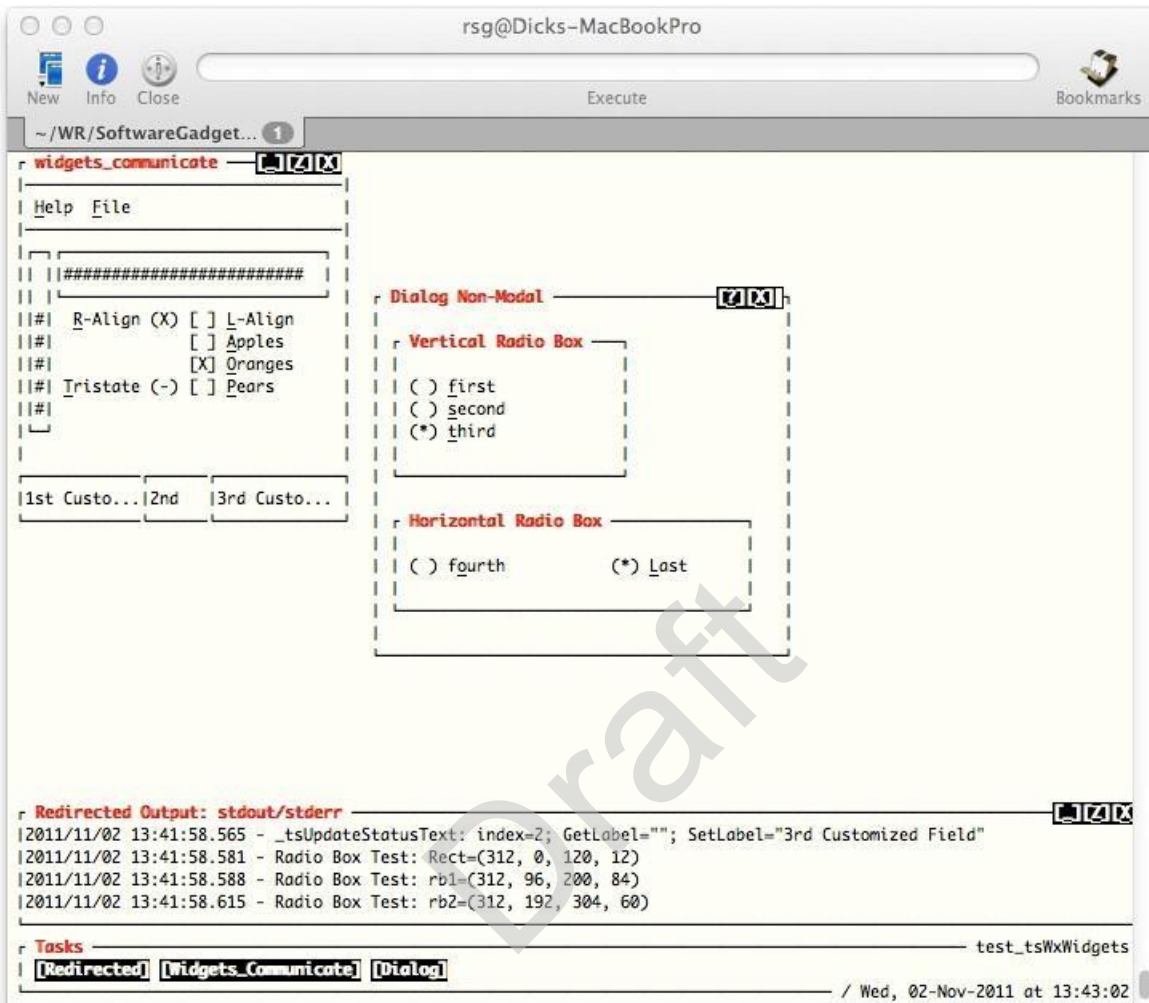
3.5.1.5 Mac OS X Platform

- *8-Color ANSI via Mac OS X Terminal* (on page 83)
- *1-Color VT100 via Mac OS X Terminal* (on page 84)
- *1-Color VT100 with 2nd-Color Highlight via Mac OS X iTerm* (on page 85)
- *8-Color XTERM via Mac OS X iTerm* (on page 86)
- *8-Color XTERM via Mac OS X Terminal* (on page 87)

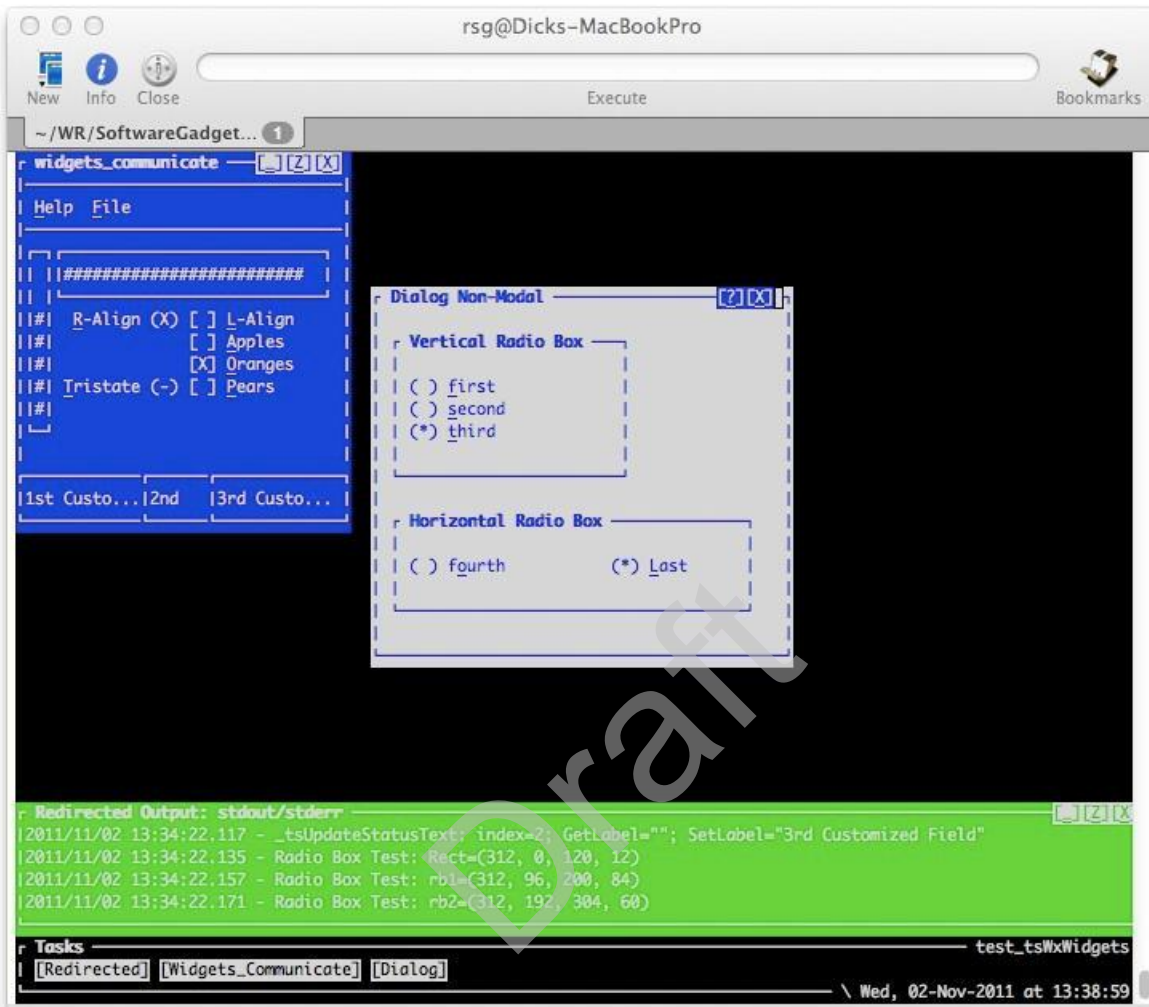
3.5.1.5.2 1-Color VT100 via Mac OS X Terminal



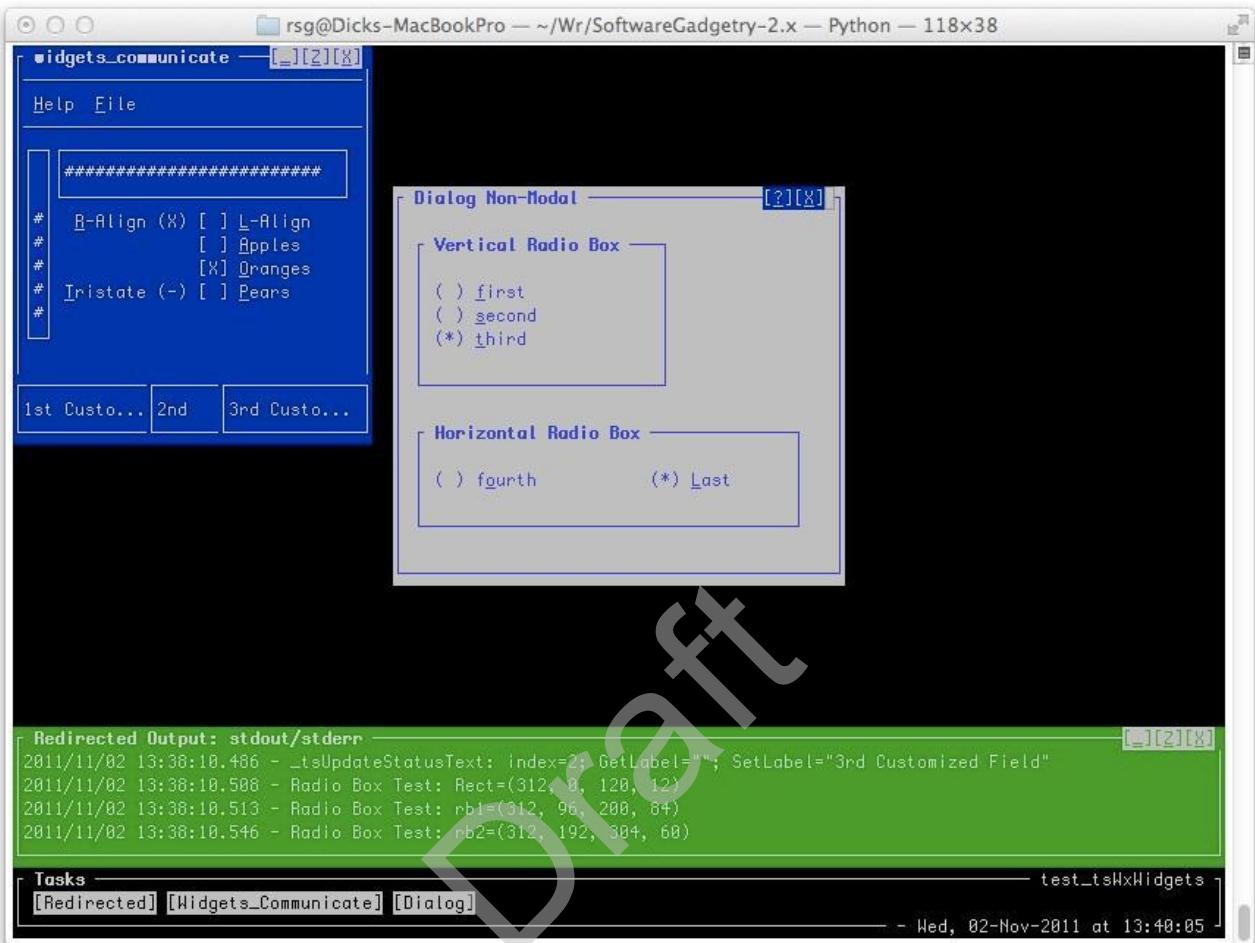
3.5.1.5.3 1-Color VT100 with 2nd-Color Highlight via Mac OS X iTerm



3.5.1.5.4 8-Color XTERM via Mac OS X iTerm



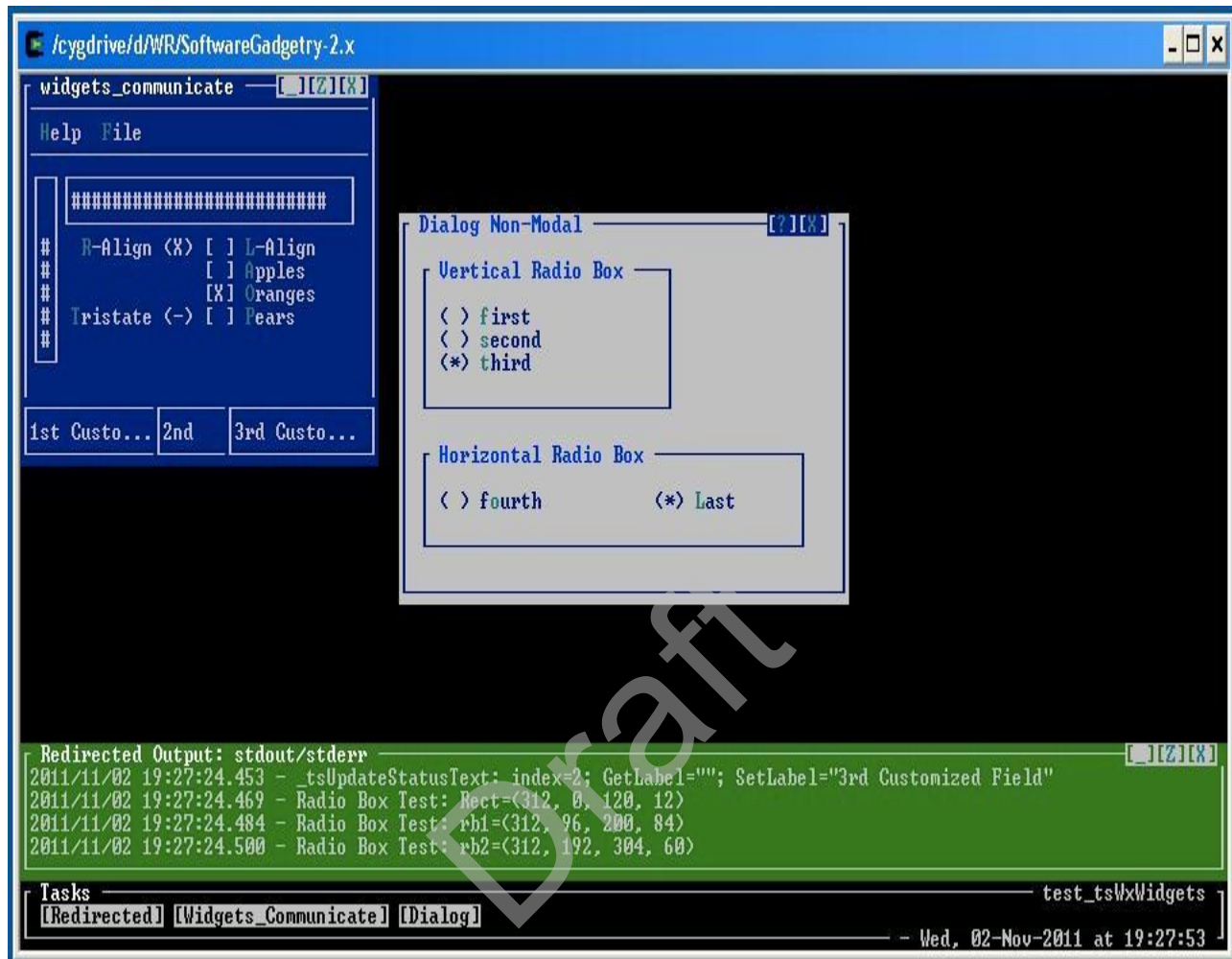
3.5.1.5.5 8-Color XTERM via Mac OS X Terminal



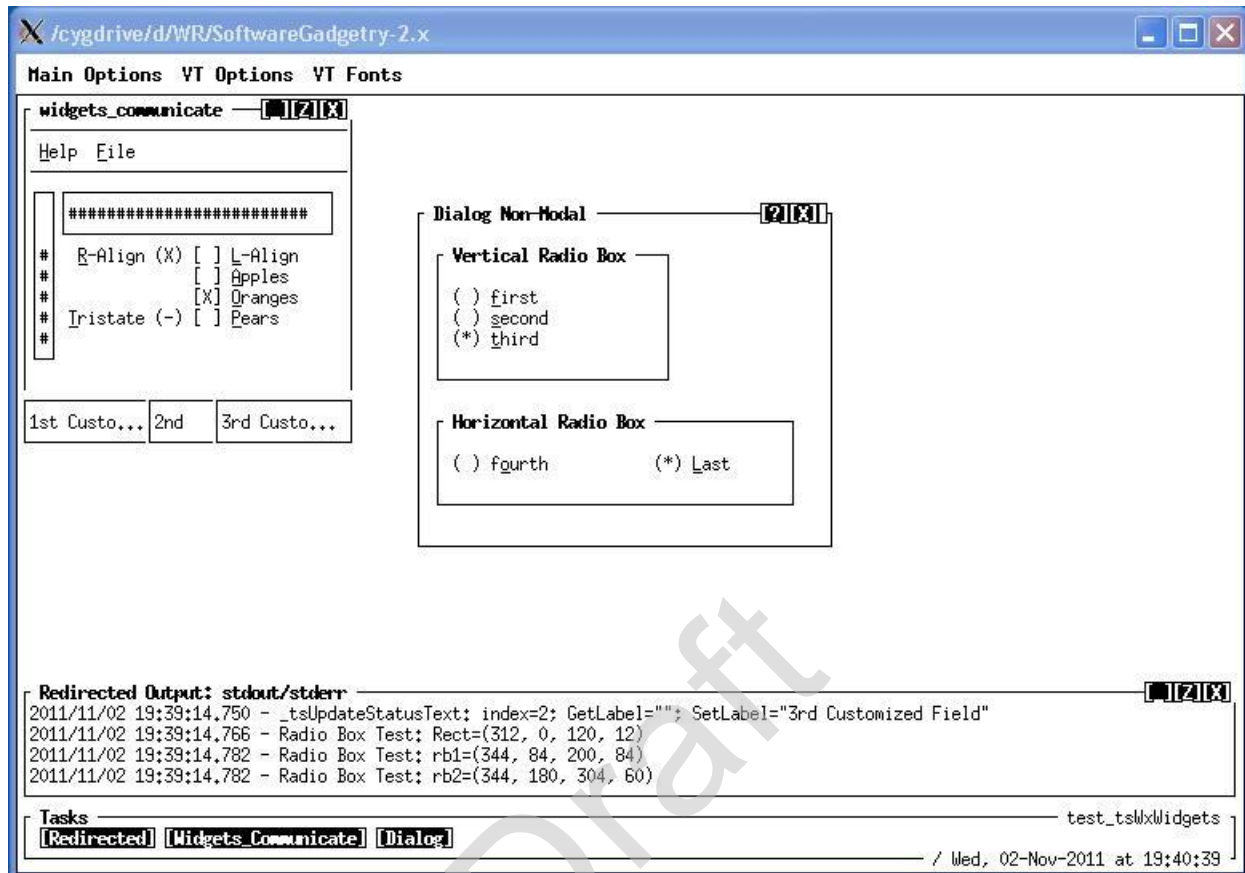
3.5.1.6 Microsoft Windows (with free Linux-like Cygwin add-on from Red Hat) Platform

- *8-Color Terminal via Cygwin Console* (on page 88)
- *1-Color VT100 via Cygwin-X Console* (on page 89)
- *1-Color VT100 with 2nd-color Highlight via Cygwin Console* (on page 90)
- *8-Color XTERM via Cygwin-X Console* (see "*Keyboard*" on page 91)

3.5.1.6.1 8-Color Terminal via Cygwin Console

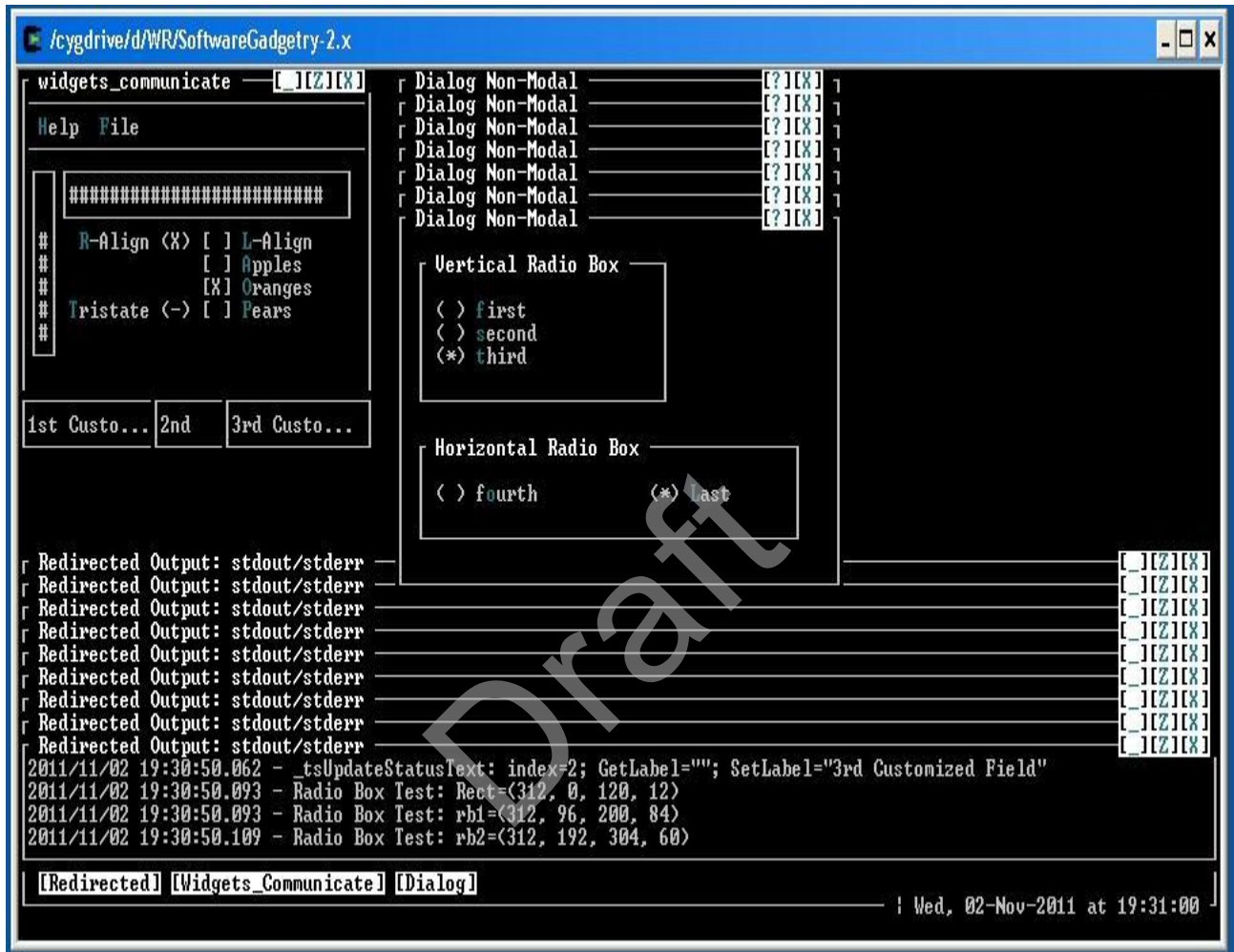


3.5.1.6.2 1-Color VT100 via Cygwin-X Console

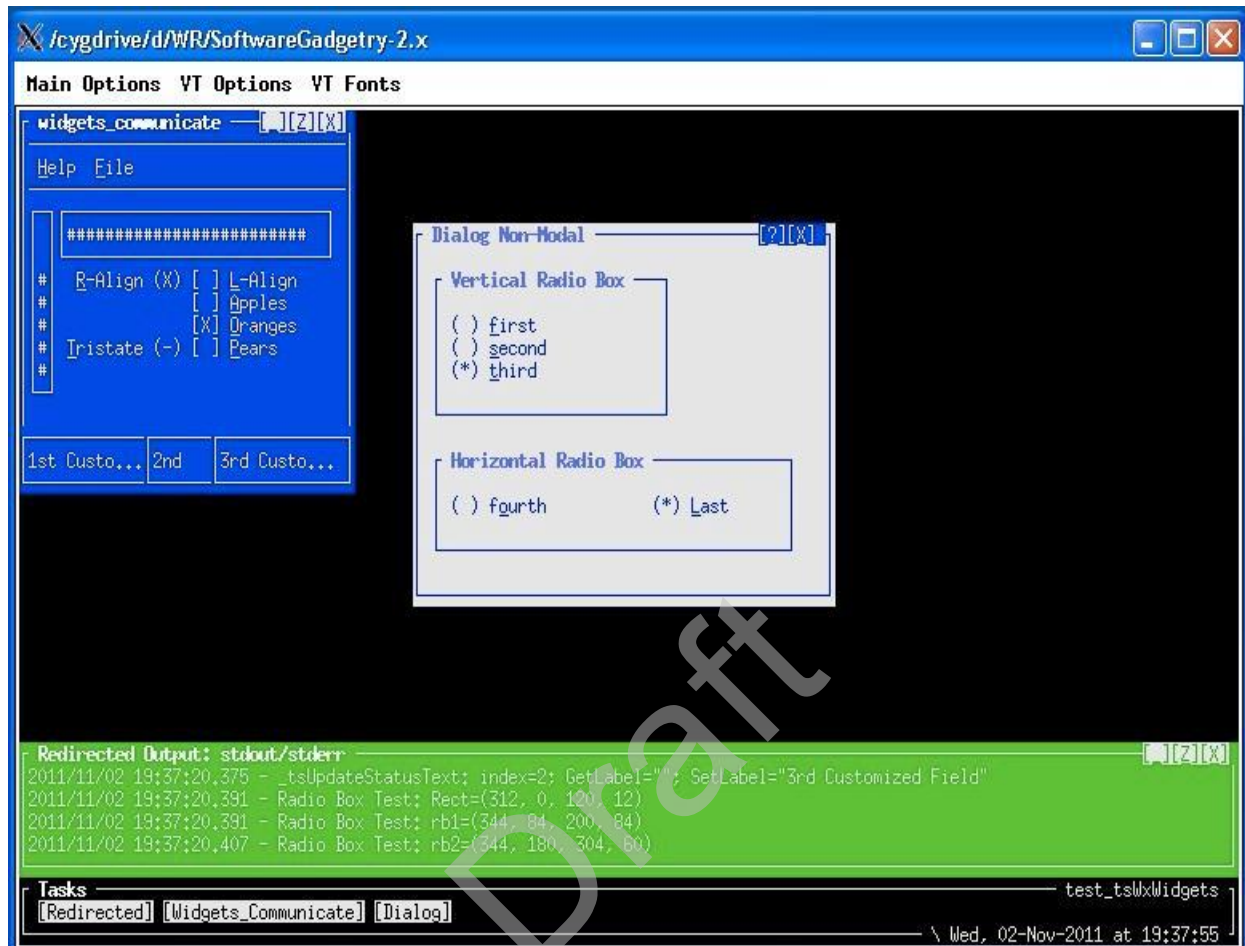


3.5.1.6.3 1-Color VT100 with 2nd-color Highlight via Cygwin Console

The Cygwin Console does not provide a usable VT100 Terminal Emulation. It repeatedly scrolls when rendering border generating characters.



3.5.1.6.4 8-Color XTERM via Cygwin-X Console



3.5.2 Keyboard

Input is submitted by the operator via a terminal keyboard device, The keyboard consists of a matrix of push buttons. The operator may press a letter, digit, punctuation symbol or function key. The use may also simultaneously press and hold a combination of the shift, control or alt keys.

3.5.3 Mouse

Input of GUI object selections is submitted by the operator via a terminal mouse device. A touchscreen, trackpad or trackball device may be substituted for a mouse device.

The input consists of the cursor position data needed to identify the selected GUI-object, the button identity (left, middle, right) and the type of selection (pressed and held, released, single click, double click and triple click).

3.6 Precedence and criticality of requirements

This paragraph shall be numbered as the last paragraph in Section 3 and 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 qualification 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 interfacing entities that relies on observable functional operation not requiring the use of instrumentation, special test equipment, or subsequent analysis.
 - b. Test: The operation of interfacing entities using instrumentation or 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 interfacing entities, documentation, etc.
 - e. Special qualification methods: Any special qualification methods for the interfacing entities, such as special tools, techniques, procedures, facilities, and acceptance limits.
-

Draft

5 REQUIREMENTS TRACEABILITY

For system-level interfacing entities, this paragraph does not apply. For each subsystem- or lower-level interfacing entity covered by this IRS, this paragraph shall contain:

- a. Traceability from each requirement imposed on the entity in this specification to the system (or subsystem, if applicable) requirements it 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 that has been allocated to the interfacing entity and that affects an interface covered in this specification to the requirements in this specification that address it.
-

Draft

6 NOTES

This section shall contain any general information that aids in understanding this document (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 understand this document.

Draft

Draft

7 APPENDIXES (Interface Requirements)

Appendixes may be used to provide information published separately for convenience in document maintenance (e.g., charts, classified data). As applicable, each appendix shall be referenced 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