

Project



TeamSTARS "tsWxGTUI_PyVx" Toolkit
with Python™ 2x & Python™ 3x based
Command Line Interface (CLI)
and "Curses"-based "wxPython"-style
Graphical-Text User Interface (GUI)

*Get that cross-platform, pixel-mode "wxPython" feeling on character-mode 8-/16-color (xterm-family)
and non-color (vt100-family) terminals and terminal emulators.*



Project

Table of Contents *(with slide show [Hyperlinks](#))*

■ **Objectives**

- **Goals (Capabilities)**
- **Non-Goals (Limitations)**

■ **Plans**

- **Technologies (Cross-Platform)**
- **Design Decisions (System Architecture, Source Code & Documentation)**
- **Release & Publication (Stages, Phases and Issues)**



Project Objectives

([Table of Contents](#))

■ Goals (Capabilities)

- [Mission Critical Capabilities](#)
- [Cross-Platform Capabilities](#)
- [Architecture Capabilities](#)
- [Documentation Audience](#)
- [Documentation Capabilities](#)
- [Engineering Notebook Capabilities](#)
- [User "How-to-Guide Capabilities](#)

■ Non-Goals (Limitations)

- [Host Virtual Machine Limitations](#)
- [GUI Virtual Machine Limitations](#)
- [Retrofit Limitations](#)



Project Objectives [\(Table of Contents\)](#)

Goals (Mission Critical Capabilities)

- Provide a foundation of building block libraries, tools and utilities that:
 - lets you work more quickly and integrate your systems more effectively
 - facilitates the rapid prototyping of application software used to monitor, control and coordinate **Mission Critical Equipment**
- The foundation components must be general purpose and re-usable in order to support the following markets:
 - **Commercial** (as in building energy management)
 - **Industrial** (as in power generation)
 - **Medical** (as in cat-scan)
 - **Military** (as in weapon control)
- Foundation components must facilitate the rapid prototyping of application software on diverse assortment of:
 - **General Purpose Desktop & Laptop Computer Systems**
 - **Application-Specific Embedded Computer Systems:**
 - **Automation** (as in robotics)
 - **Communication** (as in network traffic)
 - **Control** (as in supervisory and feedback of equipment and processes)
 - **Diagnostic** (as in hardware and software failure modes and effects)
 - **Instrumentation** (as in sensor data acquisition and analysis)
 - **Simulation** (as in flight control)



Project Objectives [\(Table of Contents\)](#)

Goals (Cross-Platform Capabilities)

- Foundation components must be compatible with and portable to a diverse assortment of popular and readily available hardware and software platforms:
 - **Open**
 - HW as in Arm & Intel microprocessor components
 - SW as in GNU's Unix-like tool components & Linus Torvalds' Linux operating system kernel components
 - **Proprietary**
 - HW as in Dell, HP, IBM & Lenovo systems
 - SW as in Microsoft's Windows & Oracle's Solaris operating system components



Project Objectives [\(Table of Contents\)](#)

Goals (Architecture Capabilities)

- Foundation architecture must be modular to support product life-cycle.
 - **Segregation of Python language generations (such as 1x, 2x and 3x)** to facilitate:
 - Future Back-Porting Python 1x from Python 2x.
 - Future Back-Porting Python 2.0.x-2.6.x from Python 2.7.x using "**Six**", a Python 2.x and 3.x compatibility library
 - Porting Python 3.x from Python 2x using Python syntactical converter "**2to3**"
 - Future Porting Python 4x from Python 3x using Python syntactical converter "**3to4**"
 - **Non-Installable Developer Sandbox** to facilitate foundation component development, maintenance and troubleshooting.
 - **Installable Site-Package** to facilitate foundation user's application software development and deployment.
- Foundation components must be customizable in order to support a diverse assortment of operators & organizations:
 - **Customizable** "txCxGlobals" file of organization-/product-specific usage terms & conditions and operator-specific CLI theme-based feature settings
 - **Customizable** "tsWxGlobals" file of organization-/product-specific GUI theme-based feature settings



Project Objectives [\(Table of Contents\)](#)

Goals (Documentation Audience)

- **Prospective & New Users** seek reason to dig deeper or look elsewhere
- **Student Users** seek rationale for unfamiliar computer hardware and software technology usage
- **Intermediate Users** seek rationale for architecture, design and implementation of unfamiliar "Python", "Curses" and "wxPython" programming techniques
- **Advance Users** seek rationale for unfamiliar project planning and engineering techniques
- **Expert Users** seek rationale for unfamiliar troubleshooting, maintenance, porting and enhancement techniques



Project Objectives [\(Table of Contents\)](#)

Goals (Documentation Capabilities)

- Documentation for system administrators & installers, developers & maintainers, operators, troubleshooters and students.
 - Establish a reference library (for Objectives, Plans, Requirements, Architecture, Design, Implementation, Debug, Test & Release)
 - Orients & Trains new contributors & users
 - Focuses or reminds contributors of goals, non-goals, plans and unresolved issues
 - Establish a convenient reference library of third-party material, found on internet, which was relevant but might eventually be subject to change or removal
 - Computer & System Engineering (Definitions, Theory, Technology & Practices)
 - External Goods & Services Resources



Project Objectives [\(Table of Contents\)](#)

Goals (Engineering Notebook Capabilities)

- Typical engineering project information with commentaries describing rationale and evolutionary changes (for System, Hardware, Software & Interfaces):
 - Concept, Dictionary, Use Cases & Development Plan
 - Requirement, Design, Test & Qualification Specifications
 - Release Notes & User's Manual
- Typical engineering project contributor information:
 - Document Authoring & Publishing Tools
 - Software Development Tools
 - Introduction and Training
- In various formats with text, tables and complex graphical images requiring an office suite (such as from "Adobe", "Microsoft", "LibreOffice" etc.) application programs typically found on a general purpose desktop computer system.



Project Objectives [\(Table of Contents\)](#)

Goals (User “How-to-Guide” Capabilities)

■ Documents

- Typical install, configure, operate and troubleshoot how-to guides with applicable terms and conditions for usage and redistribution.
- In a plain text format suitable for embedded systems with only character-mode terminals.

■ Manual Pages

- Typical on-line information about Command Line Interface & Graphical User Interface use and application programming for each building block and tool.
- In a plain text format suitable for embedded systems with only character-mode terminals.



Project Objectives [\(Table of Contents\)](#)

Non-Goals (Host Virtual Machine Limitation)

- Foundation will NOT provide “*Magical*” Host Virtual Machines which:
 - **Runs incompatible Processor & Operating System Specific Applications:**
 - You should **NOT** expect to be able to run application programs designed specifically for one processor make & model and one make & model operating system on a different processor and operating system.
 - You should **ONLY** expect to be able to run applications designed to run on a Python “virtual machine” that itself has been designed (by the Python Software Foundation) for the specific processor & operating system, tested and certified to correctly interpret and execute source code appropriate for the Python language generation (such as 1x, 2x or 3x).



Project Objectives [\(Table of Contents\)](#)

Non-Goals (GUI Virtual Machine Limitation)

- Foundation will NOT provide “*Magical*” GUI Virtual Machines which:
 - **Run incompatible GUI Applications:**
 - You should **NOT** expect to be able to run pixel-mode “wxPython”, “wxWidgets”, “Qt” or “Tcl/Tk” applications that can copy graphic images from a file to the display and dynamically construct and output an array of pixels to the display which depicts the desired icons, graphic objects and text images.
 - You should **ONLY** expect to be able to run character-mode “wxPython”-style GUI applications designed to dynamically construct and output to the display an array or sequence Curses-standard alpha-numeric, punctuation and line-drawing characters (with escape sequences to control output to a desired display screen column and row position).



Project Objectives [\(Table of Contents\)](#)

Non-Goals (Retrofit Limitations)

- Foundation will **NOT** provide "*Magical*" Python Virtual Machine cross-platforms for use with a diverse assortment of obsolete Hardware and Software platforms.
 - **Open** (HW such as 8-/16-bit Intel & Motorola microprocessors, 32-/64-bit Intel iAPX 432, i860; SW source code such as implemented in assembler, Ada, C/C++, FORTRAN and Python 1.x languages)
 - **Proprietary** (HW as in Digital Equipment Corp. & SGI systems; SW such as VAX/VMS & IRIX operating systems)
- Even if others have or could obtain such long discontinued platforms, this project author will **NOT** seek funding to obtain, reconstruct or simulate such platforms.



Project Plans [\(Table of Contents\)](#)

- [Adopt Cross-Platform Technology Plan](#)
- [Adopt Modular Software Architecture Plan](#)
- [Adopt Python 2x Source Code Plan](#)
- [Adopt Python 3x Source Code Plan](#)
- [Adopt Development, Debug and Test Environment Plan](#)
- [Adopt Python Virtual Machine \(VM\) Environment Plan](#)
- [Adopt Python Virtual Machine \(VM\) Plan](#)
- [Adopt Engineering Notebook Plan](#)
- [Adopt Operator, System Administrator & Field Service Plan](#)
- [Adopt Document Focus Plan](#)
- [Adopt Release & Publication Plan](#)



Project Plans [\(Table of Contents\)](#)

Adopt Cross-Platform Technology Plan

- Apply Hardware & Software Technology that lets you work more quickly and integrate your systems more effectively:
 - Popular, readily available and/or within the project budget
 - Suitable for rapid prototyping
 - Field proven with a long term track record of support
 - Software & Documentation must be free to study, modify, use and redistribute



Project Plans [\(Table of Contents\)](#)

Adopt Modular Software Architecture Plan

- Provide source code for cross-platform software development:
 - **Libraries** of Application and Troubleshooting Building Block components.
 - **Tools** for Facilitating and Tracking developer productivity.
 - **Utilities** for Monitoring and Changing hardware and software configuration.
 - **Tests** (Unit, Integration, System, Regression and Acceptance) for design verification and quality assurance.
 - **Examples** for Algorithms, Coding Style, Programmer Productivity Metrics and System Performance.
- Provide source code and associated install tools for:
 - **Site-Packages** (tailored for each Python 2x and Python 3x generation language) that installs and thereby connects third-party packages with one or more System Administrator designated previously installed Python 2.x.y or 3.x.y distribution.
 - **Developer-Sandboxes** (tailored for each Python 2x and Python 3x generation language) that will isolate untested code changes and outright experimentation from the production ("Site-Package") environment or repository.



Project Plans [\(Table of Contents\)](#)

Adopt Cross-Platform Software Environment Plan

- **POSIX**, a Unix-like operating system complying with the Portable Operating System Interface:
 - **Apple Mac OS X** (Darwin-/BSD-based Unix)
 - **GNU/Linux** (combination of Unix-like GNU tools with Linux kernel)
 - **Microsoft Windows** (requires **Cygwin**, the GNU/Linux-like toolkit and command-line interface plug-in from Red Hat)
 - **Unix** (derived directly or indirectly from the original AT&T UNIX)
- **Python**, an interpreted, object-oriented programming language and cross-platform virtual machine:
 - **Python 3x** (**actively evolving & maintained** 3rd generation language)
 - **Python 2x** (**mature & actively maintained** 2nd generation language)
 - **Python 1x** (**obsolete & unmaintained** 1st generation language)
- **wxPython**, a cross-platform GUI Application Programming Interface



Project Plans [\(Table of Contents\)](#)

Adopt Python 2x Source Code Plan

- Develop software in mature & actively maintained Python 2x (2nd generation language)
 - Create non-installable Python 2x **Developer Sandbox** to facilitate troubleshooting
 - “__init__.py” defines nested package structure and dependency relationships
 - Modules import from other modules & packages within “try-except” logic to report import errors
 - Create installable Python 2x **Site-Package** to facilitate to use of Toolkit building blocks in same manner as library components registered in Python Global Module Index.
 - Copy Python 2x **Developer Sandbox**
 - Replace “__init__.py” modules with empty ones
 - Replace “try-except” import logic with explicit references to site-package identifier



Project Plans [\(Table of Contents\)](#)

Adopt Python 3x Source Code Plan

- Develop software in actively evolving & maintained Python 3x (3rd generation language)
 - Create non-installable Python 3x **Developer Sandbox** to facilitate troubleshooting
 - Copy non-installable Python 2x **Developer Sandbox**
 - Convert syntax of Python 2x to Python 3x (3rd generation language) using Python "**2to3**" utility
 - Debug to identify and resolve remaining issues
 - Create installable Python 3x **Site-Package** to facilitate to use of Toolkit building blocks in same manner as library components registered in Python Global Module Index.
 - Copy Python 3x **Developer Sandbox**
 - Replace "`__init__.py`" modules with empty ones
 - Replace "`try-except`" import logic with explicit references to site-package identifier



Project Plans [\(Table of Contents\)](#)

Adopt Development, Debug and Test Environment Plan

- Representative & Readily Available
 - Processors (32-/64-bit data register width)
 - Single Core --- A component containing a single processor performing all of the work.
 - Multi-Core or Multi-Processor --- One or more components containing multiple processors each performing their delegated portion of the work.
 - Processor-specific "Host" and optional "Guest" operating systems
 - Host OS --- Primary operating system connected to other computers or terminals to which it provides data or computing services via a hard-wired connection or switched telecommunication network.
 - Guest OS --- Secondary operating system that is either part of a partitioned system or part of a virtual machine (VM) setup.
- Develop, Debug and Test on available Sample Platforms



Project Plans [\(Table of Contents\)](#)

Adopt Python Virtual Machine (VM) Environment Plan

- A Cross-Platform Environment is created by VMs which are typically:
 - **Implemented** in a platform-independent programming language such as "C/C++"
 - **Compiled** into executable platform-specific VM building block library functions
 - **Executed** upon an operator's shell command (such as "python **DEMO.py** -help")
- VMs typically execute the Python application, upon its launch, via the following process:
 - **Compile** any un-compiled Python application & imported source code into a processor-independent byte-code containing tokens for each standard Python statement or subprogram operation
 - **Interpret** the VM tokens
 - **Execute** VM token-associated executable platform-specific VM building block library functions



Project Plans [\(Table of Contents\)](#)

Adopt Python Virtual Machine (VM) Plan

- Project Author and Release Adopters
 - Default use of popular Python (2.x.y and 3.x.y) Virtual Machines developed:
 - by the Python Software Foundation (PSF)
 - for popular and readily available Intel processors (x86 & x64) and processor-specific operating systems.
 - Optional use of equivalent Python (2.x.y and 3.x.y) Virtual Machines (or the source code to build them) developed:
 - by the Python Software Foundation
 - for other, less popular, processor types and processor-specific operating systems.



Project Plans [\(Table of Contents\)](#)

Adopt Engineering Notebook Plan

- **Engineering Notebooks (master in repository on development system)**
 - Collection of commentaries that express opinions or offerings of explanations about events or situations that might be useful to installers, developers, operators, troubleshooters and distributors of the toolkit framework.
 - Formats include text, tables and complex graphical images requiring an office suite application programs (such as from "Adobe", "Microsoft", "LibreOffice" etc.) typically found on a general purpose workstation, desktop or laptop computer systems.
- **Project (master in repository on development system; copy in site-package on embedded system)**
 - Excerpts from the Engineering Project Notebook that is in plain text format and suitable for embedded systems with only character-mode terminals.



Project Plans [\(Table of Contents\)](#)

Adopt Operator, System Administrator & Field Service Plan

- **Documents (master in repository on development systems with copies in developer sandboxes; copies also in site-packages on embedded systems)**
 - Typical install, configure, operate and troubleshoot how-to guides with applicable terms and conditions for usage and redistribution. In a plain text format suitable for embedded systems with only character-mode terminals.
- **Manual Pages (master in repository on development systems with copies in developer sandboxes; copies also in site-packages on embedded systems)**
 - Typical on-line information about command line use and application programming for each building block and tool. Each is generated, by a Python source code processing tool, in a plain text format suitable for embedded systems with only character-mode terminals.



Project Plans [\(Table of Contents\)](#)

Adopt Document Focus Plan

- **System Administrators and Field Service** will need to know or learn:
 - Hardware and Software Requirements for System and Applications
 - Available Hardware and Software Product & Support Resources
 - How to Install, Configure, Operate and Troubleshoot the Hardware and Software for System and Applications
- **Operators** will need to know or learn:
 - Available System and Application Hardware and Software features and how to use them
- **Developers** will need to know or learn:
 - Hardware and Software Architecture
 - Hardware and Software Interfaces
 - Software Design and Qualification Requirements
 - How to Install, Configure, Operate and Troubleshoot the Hardware and Software for local and remote Systems and Applications



Project [\(Table of Contents\)](#)

Adopt Release & Publication Plan

- **Pre-Alpha**

- Pre-testing phase begins during the software and documentation development.
- Milestone versions include specific sets of functions and are released as soon as the functionality is complete.

- **Alpha**

- Testing phase begins when the software and documentation still may not contain all of the features that are planned for the final version.
- The software can be unstable and could cause crashes or data loss.

- **Beta**

- Testing phase begins when the software and documentation is feature complete but likely to contain a number of known or unknown bugs.
- It will generally have many more bugs in it than completed software, as well as speed/performance issues and may still cause crashes or data loss.

- **Release Candidate**

- Testing phase begins when the software and documentation has potential to be a final product, which is ready to release unless significant bugs emerge.
- In this stage of product stabilization, all product features have been designed, coded and tested through one or more beta cycles with no known showstopper-class bug.
- A release is called code complete when the development team agrees that no entirely new source code and documentation will be added to this release.
- There could still be source code changes to fix defects, changes to documentation and data files, and peripheral code for test cases or utilities.

- **Release to World Wide Web**

- The means of software and documentation delivery, at the final release or at any previous testing stage, that utilizes the Internet for distribution.
- No physical media are produced in this type of release mechanism by the manufacturer.