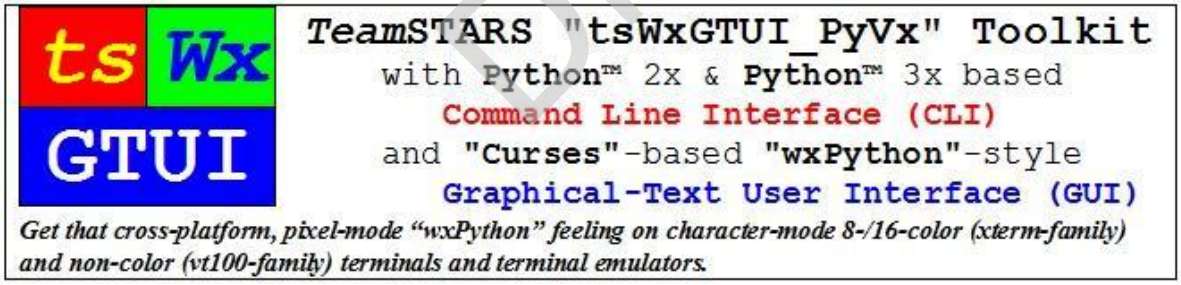
Software Gadgetry

Brochure

Vol. 1 - "tsWxGTUI" Toolkit

Rev. 0.0.0 (Pre-Alpha)

Author(s): Richard S. Gordon



Printed on 19 July, 2015

Contents

1 Features

The TeamSTARS "tsWxGTUI_PyVx" Toolkit software is engineered to facilitate your development and use of application programs that feature:

- ***Command Line Interface (CLI)*** (on page 3)
- ***Graphical-style User Interface (GUI)*** (on page 3)

1.1 Command Line Interface (CLI)

Output to the user of a chronological sequence of lines of text via a scrolling computer terminal display with input from the user via a computer terminal keyboard.

1.2 Graphical-style User Interface (GUI)

Output to the user of character strings to application-specified column and row (line) fields of a computer terminal display with input from the user via a computer terminal keyboard and pointing device (such as mouse, trackball, touchpad or touchscreen).

1.3 System Block Diagrams

1 ***Block Diagram*** (on page 5) - High level depiction of the organizational, functional and interface relationship between the *Team*STARS "tsWxGTUI_PyVx" Toolkit components and users (System Administrator, Software Engineer, System Operator and Field Service personnel):

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

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

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

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

t**sCxGlobalsPkg** --- Module to establish configuration constants and macro-type functions for the Command Line Interface mode of the "tsWxGTUI_PyVx" Toolkit. It provides a centralized mechanism for modifying/restoring those configuration constants that can be interrogated at runtime by those software components having a "need-to-know". The intent being to avoid subsequent manual searches to locate and modify or restore a constant appropriate to the current configuration. It also provides a theme-based mechanism for modifying/restoring those configuration constants as appropriate for various users and their activities.

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

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

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

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

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

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

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

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

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

t**sWxGlobals** --- Module provides a centralized mechanism for modifying/restoring those configuration constants that can be interogated at runtime by those software components having a "need-to-know". The intent being to avoid subsequent manual searches to locate and modify or restore a constant appropriate to the current configuration. It also provides a theme-based mechanism for modifying/restoring those configuration constants as appropriate for the character-mode emulation of the following pixel-mode "wxPython" features:

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

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

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

2 ***Stand Alone System Architecture*** (on page 6) - High level depiction and description of the components of and relationship between components of an isolated system operating by itself.

3 ***Stand Among System Architecture*** (on page 11) - High level depiction and description of the components of and relationship between two or more networked systems operating in collaboration with each other.

1.3.1 Block Diagram

This Block Diagram depicts the organizational, functional and interface relationship between the *Team*STARS "tsWxGTUI_PyVx" Toolkit components and users (System Administrator, Software Engineer, System Operator and Field Service personnel):

1 the external System Operator interface to "tsToolkitCLI"

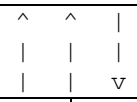
2 the internal "tsToolkitCLI" interface to "tsToolkitGUI"

3 the internal System Operator interfaces:

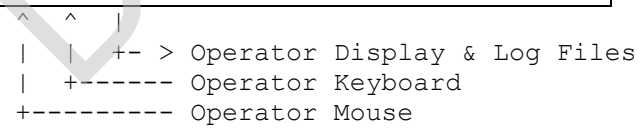
a) to "tsUtilities", "tsToolsCLI" and "tsTestsCLI" via "tsToolkitCLI"

b) to "tsToolsGUI" and "tsTestsGUI" via "tsToolkitCLI" and "tsToolkitGUI"

Graphical-style User Interface (tsToolkitGUI)	
▪ The "tsToolsGUI" is a set of graphical-style user interface application programs for tracking software development metrics.	▪ The "tsTestsGUI" is a set of graphical-style user interface application programs for regression testing and tutorial demos.
▪ The "tsLibGUI" is a library of graphical-style user interface building blocks that establishes the emulated run time environment for the high-level, pixel-mode, "wxPython" GUI Toolkit via the low-level, character-mode, "nCurses" Text User Interface Toolkit.	



Command Line Interface (tsToolkitCLI)	
▪ The "tsToolsCLI" is a set of command line interface application programs and utility scripts for: checking source code syntax and style via "plint"; generating Unix-style "man page" documentation from source code comments via "pydoc"; and installing, modifying for publication, and tracking software development metrics.	▪ The "tsTestsCLI" is a set of command line interface application programs and scripts for regression testing and tutorial demos.
▪ The "tsLibCLI" is a library of command line building blocks that establishes the POSIX-style, run time environment for pre-processing source files, launching application programs, handling events (registering events with date, time and event severity annotations) and configuring console terminal and file system input and output.	
▪ The "tsUtilities" is a library of computer system configuration, diagnostic, installation, maintenance and performnce tool components for various host hardware and software platforms.	



1.3.2 Stand Alone System Architecture

The *Team*STARS "tsWxGTUI_PyVx" Toolkit provides:

1. Python version-specific components for its Command Line Interface ("tsToolkitCLI")

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

The following description uses the component names as depicted in the ***Block Diagram*** (on page 5)

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"

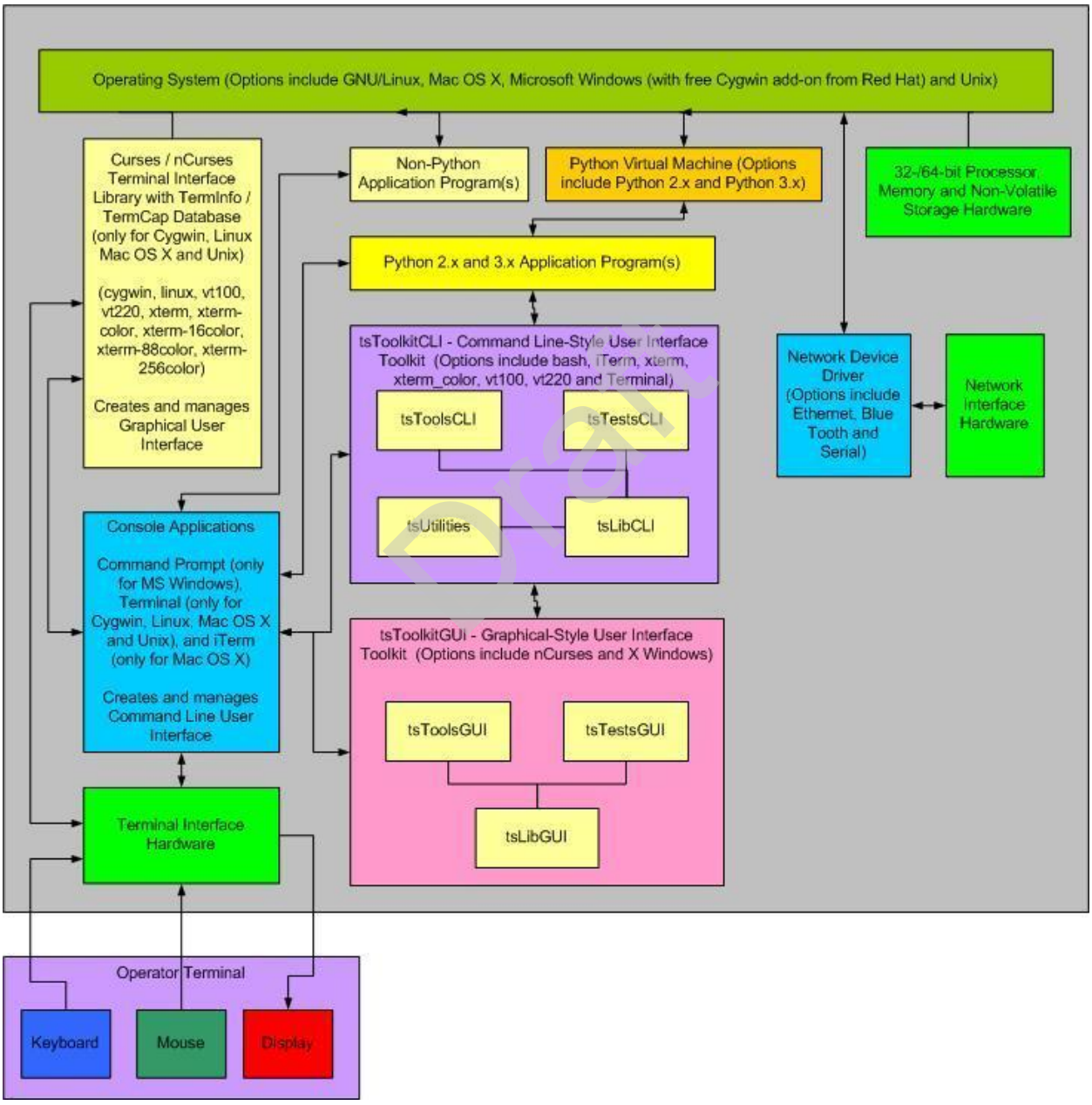
2 the internal "tsToolkitCLI" interface to "tsToolkitGUI"

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 with 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 performnce 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.

1.3.3 Stand Among System Architecture

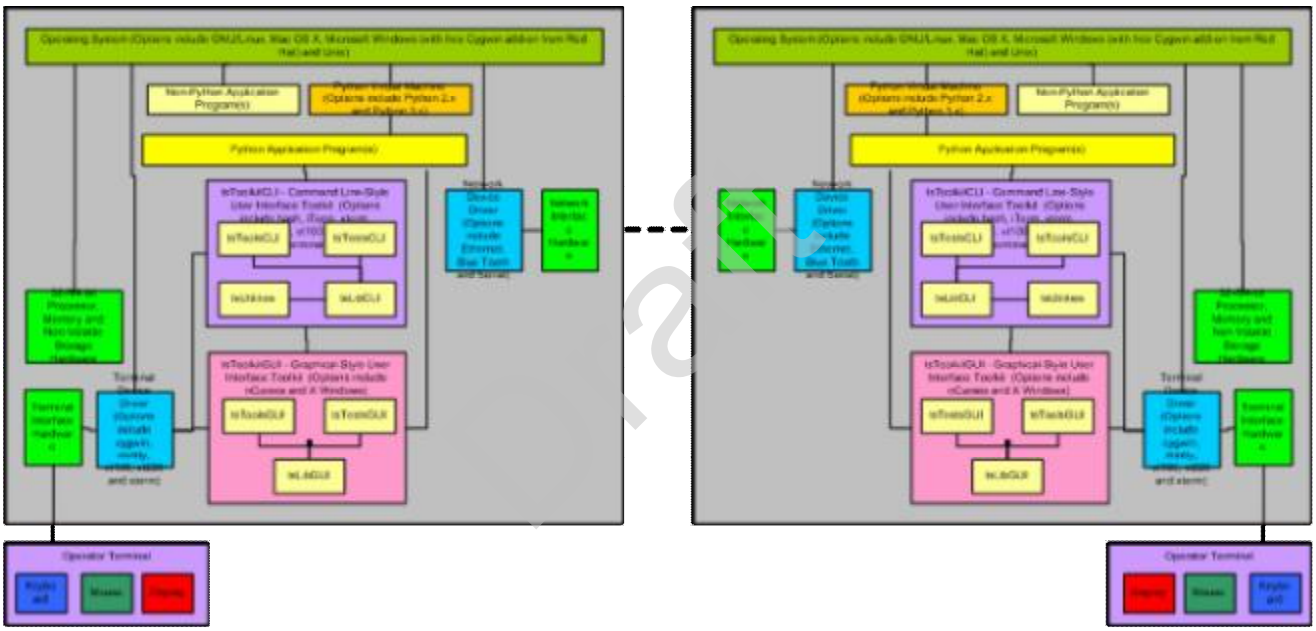
The *Team*STARS "tsWxGTUI_PyVx" Toolkit provides:

1. Python version-specific components for its Command Line Interface ("tsToolkitCLI")

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

The following description uses the component names as depicted in the ***Block Diagram*** (on page 5)

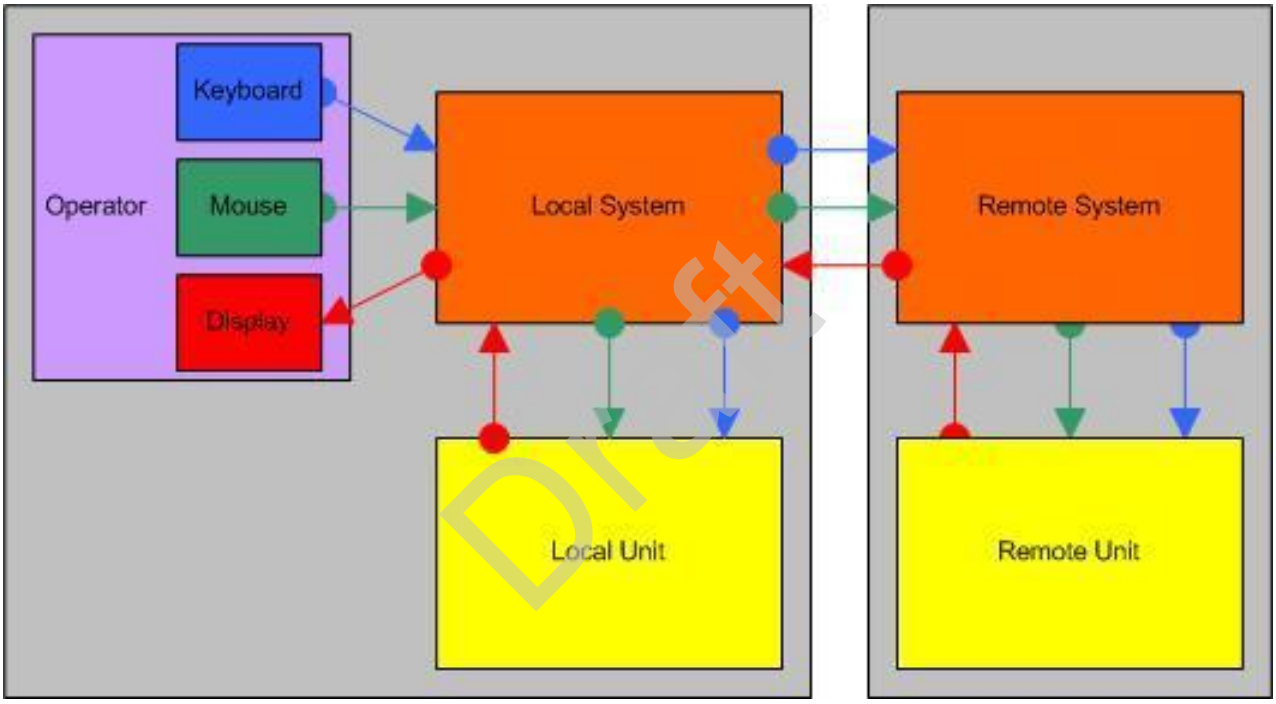
The ***Stand Alone System Architecture*** (on page 6), as previously described, may be extended to enable a single operator, working from a Local System, to interact with one or more Remote Systems.



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 13) figure depicts the desktop of a multi-user, multi-process, multi-threaded computer running the Professional Edition of Microsoft Windows 7. Among the backround of desktop icons, there are two *Team*STARS "tsWxGTUI_PyVx" Toolkit sessions. The time each session displays synchronizes within its own one second refresh interval. The local session, on the left, is actively running Python 3.x on the Windows platform. The remote session, on the right, is actively running Python 2.x on the Mac OS X Yosemite platform which also serves as the Parallels 10 Hypervisor host for diverse Guest Operating Systems including:

  Linux (CentOS7 64-bit, Fedora 21 64-bit, Scientific 6.5 32-bit and Ubuntu 12.04 32-bit)

  Windows (98 SE 16-bit, XP 32-bit, 7 32-bit, 8 32-bit and 8.1 32-bit)

  Unix (FreeBSD 9.2, PC-BSD 10, OpenIndiana 151a8 and OpenSolaris 11 32-bit)

- One or more newly opened shells may be used to login to a Remote system (via the "ssh user@Remote command). Once Remote login has been successful, the operator may run a Python or non-Python Application Program as a **Remote Unit (Right)**.

4 **Remote Unit (Right)** --- A Python or non-Python Application Program that has been launched in a Remote Command Line Interface Shell.

1.3.3.1 Multi-Session Desktop

From Wikipedia, the free encyclopedia

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

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

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

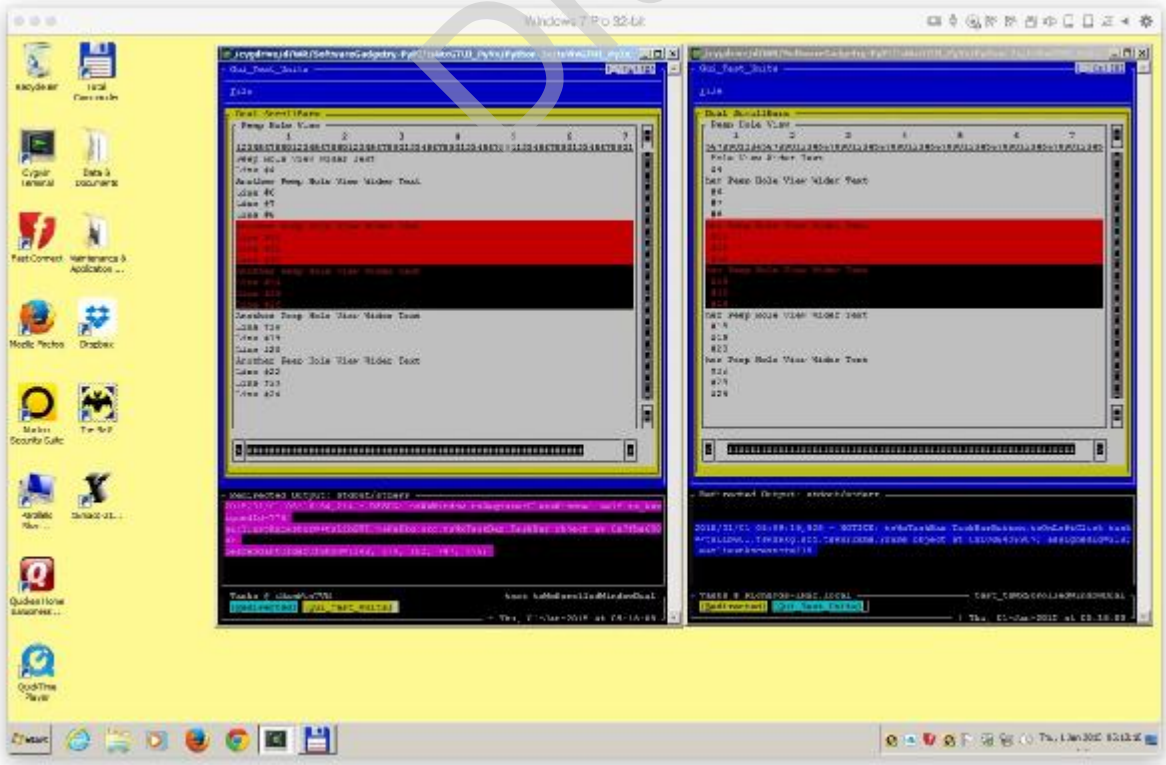
- Application layer examples:
  - HTTP sessions, which allow associating information with individual visitors
  - A telnet remote login session
- Session layer example:
  - A Session Initiation Protocol (SIP) based Internet phone call

- Transport layer example:
  - A TCP session, which is synonymous to a TCP virtual circuit, a TCP connection, or an established TCP socket.

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

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

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



The Sample Screenshot above shows a Windows 7 Desktop with:

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

2 Usage Applications

The TeamSTARS "tsWxGTUI_PyVx" Toolkit software will initially be installed in computer systems to be used for development of software and documentation.

- ***Development Platforms*** (on page 17)

It will ultimately be installed in embedded systems to be used to monitor and control commercial, industrial, manufacturing, medical or military equipment.

- ***Embedded System Platforms*** (on page 17)

2.1 Development Platforms

Such general-purpose desktop, laptop and workstation computer systems typically have upgradable or at least sufficient processing, memory, communication, input/output and file storage resources. They typically have computer terminal interface hardware suitable for a pixel-mode display that also supports character-mode.

2.2 Embedded System Platforms

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

3 Design Goals

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

The *Team*STARS "tsWxGTUI_PyVx" Toolkit software is engineered to facilitate your development and use of application programs that perform:

- ***Local Equipment Monitoring & Control*** (on page 20)
- ***Remote Equipment Monitoring & Control*** (on page 22)

3.1 Local Equipment Monitoring & Control

The *Team*STARS "tsWxGTUI_PyVx" Toolkit software is engineered to facilitate your development and use of application programs that perform Local Equipment Monitoring & Control.

Once you've logged into your local computer, you can launch one or more local shells (such as "sh" and "bash") associated with the local operating system's command line interface.

Use application name (hw_diagnostic) and the required, denoted by "<>", and optional, denoted by "[]...", items as appropriate.

1 hw_diagnostic [Switch [Switch Argument]]...

a) hw_diagnostic -h

b) hw_diagnostic --help

c) hw_diagnostic -a

d) hw_diagnostic --about

e) hw_diagnostic -v

f) hw_diagnostic --version

g) hw_diagnostic --log="/logs/rsg-sample"

2 hw_diagnostic [Verb] <Operation>

a) hw_diagnostic modify <Operation (Tune-Up Utility such as "poke")>

b) hw_diagnostic run <Operation (Setup Utility such as "load" or "unload")>

c) hw_diagnostic show <Operation (Display Utility such as "log")>

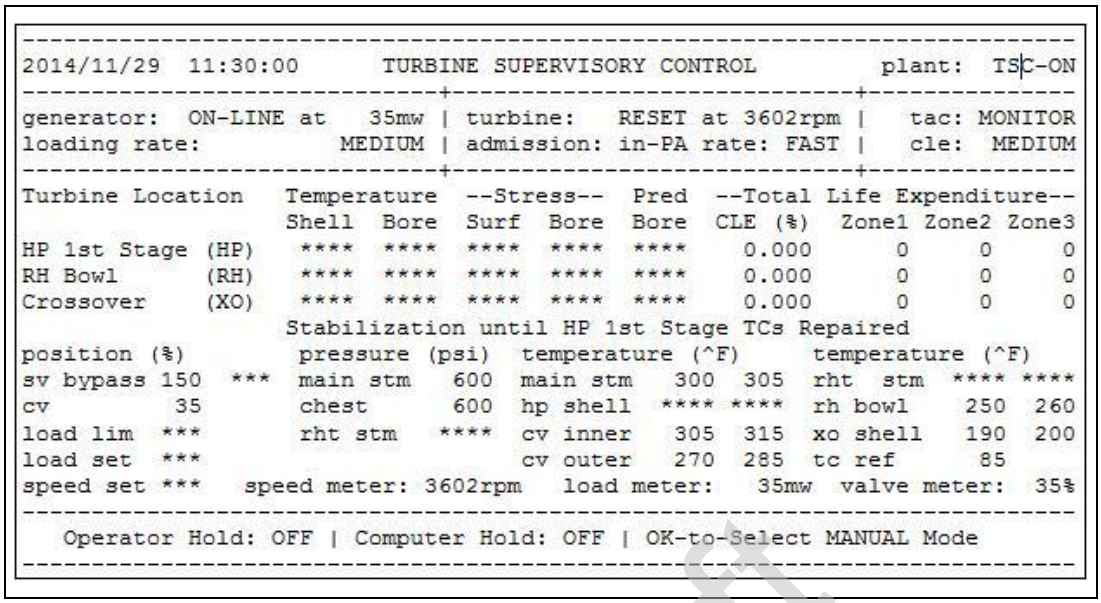
d) hw_diagnostic test <Operation (Diagnostic Test such as "ram")>

3 hw_diagnostic [Verb] <Operation> [Positional Argument]...

4 hw_diagnostic [Verb] <Operation> [Switch [Switch Argument]]... [Positional Argument]...

5 hw_diagnostic [Option [Option Argument]]... [Verb] <Operation> [Switch [Switch Argument]]... [Positional Argument]...

The local operating system's command line interface provides access to associated terminal interface and the TeamSTARS "tsWxGTUI_PyVx" Toolkit's Python and "nCurses" based character-mode user interfaces which enable you to monitor and control one or more local application programs.



3.2 Remote Equipment Monitoring & Control

The *Team*STARS "tsWxGTUI_PyVx" Toolkit software is engineered to facilitate your development and use of application programs that perform Remote Equipment Monitoring & Control.

Once you've logged into your local computer, you may then login to a remote computer using one or more secure shells ("ssh") or non-secure shells ("rsh") provided by the local operating system.

The Secure Shell ("ssh") is a cryptographic network protocol for secure data communication, remote command-line login, remote command execution, and other secure network services between two networked computers. It connects, via a secure channel over an insecure network, a server and a client running "ssh" server and "ssh" client programs, respectively.

The remote shell ("rsh") is a command line computer program that can execute shell commands as another user, and on another computer across a computer network. The remote system to which "rsh" connects runs the "rsh" daemon ("rshd").

The local and remote operating system's command line interfaces provides access to associated terminal interface and the TeamSTARS "tsWxGTUI_PyVx" Toolkit's Python and "nCurses" based character-mode user interfaces which ten communicate.

4 Design Non-Goals

This enables you to monitor and control one or more remote and local application programs, from the convenience of your local computer terminal, with greater speed and efficiency than possible with the larger communication traffic associated with pixel-mode.

What the TeamSTARS"tsWxGTUI_PyVx" Toolkit is NOT ?

The *Team*STARS "tsWxGTUI_PyVx" Toolkit has the following limitations because it has NOT been designed to provide the following:

1. ***Multi-Operating System Run Time Environment*** (on page 23)
2. ***Pixel-Mode Graphical User Interface*** (on page 23)

4.1 Multi-Operating System Run Time Environment

The *Team*STARS "tsWxGTUI_PyVx" Toolkit has the following limitations because it has NOT been designed to provide a magical cross-platform way to run native Linux, Mac OS X, Microsoft Windows and Unix applications on each other's development and embedded system platforms.

4.2 Pixel-Mode Graphical User Interface

The *Team*STARS "tsWxGTUI_PyVx" Toolkit has the following limitations because it has NOT been designed to provide a magical cross-platform way to run native pixel-mode "wxPython" / "wxWidgets" applications on platforms with character-mode terminals or terminal emulators (such as vt100, vt220, cygwin mintty, xterm, xterm-color, xterm-16color, xterm-88color and xterm-256color).

In fact, when "porting" pixel-mode "wxPython" source code, you must replace pixel graphic elements such as:

1. Icons and curved lines with suitable character-cell elements from the "Curses" / "nCurses" line draw character set (such as horizontal and vertical lines, line intersection and shape corners).
2. Proportional sized fonts and text attributes with suitable character-cell elements from the "Curses" / "nCurses" character set (such as fixed sized font with blinking, normal, bold, dim and reverse video attributes).

5 Usage Terms & Conditions

1. The *Team*STARS "tsWxGTUI_PyVx" Toolkit and its third-party components are copyrighted works that are licensed and distributed as free and open source software, 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 HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

2 You may use, modify and redistribute individual copyrighted works only under the terms and conditions of the license designated by the *Team*STARS "tsWxGTUI_PyVx" Toolkit and its third-party component work's copyright holder(s).

6 SCREENSHOTS

From Wikipedia, the free encyclopedia

"A screen dump, screen capture (or screen-cap), screenshot (or screen shot), screengrab (or screen grab), or print screen[1] is an image taken by the computer user to record the visible items displayed on the monitor, television, or another visual output device. Usually, this is a digital image using the (host) operating system or software running on the computer, but it can also be a capture made by a camera or a device intercepting the video output of the display (such as a DVR). That latent image converted and saved to an image file such as to JPEG or PNG format is also called a screenshot.

Screenshots can be used to demonstrate a program, a particular problem a user might be having, or generally when display output needs to be shown to others or archived. For example, after being emailed a screenshot, a Web page author might be surprised to see how his page looks on a different Web browser and can take corrective action. Likewise with differing email software programs, (particularly such as in a cell phone, tablet, etc.,) a sender might have no idea how his email looks to others until he sees a screenshot from another computer and can (hopefully) tweak his settings appropriately."

- ***XTERM with 8-Color / 64-Color Pairs*** (on page 26)
- ***VT-100 with 1-Color / 2-Color Pairs*** (on page 37)

6.1 XTERM with 8-Color / 64-Color Pairs

From Wikipedia, the free encyclopedia:

"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).

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."

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.

Xterm variants are also available.

Most terminal emulators for X started as variations on xterm."

Typically, xterms support keyboard and mouse input and a display whose character cells consist of an array of RED-GREEN-BLUE phosphors per pixel that are independently programmable in intensity so as to produce any of 8-colors (with their associated 64-color pairs) per character:

1 BLACK

2 RED

3 GREEN

4 YELLOW

5 BLUE

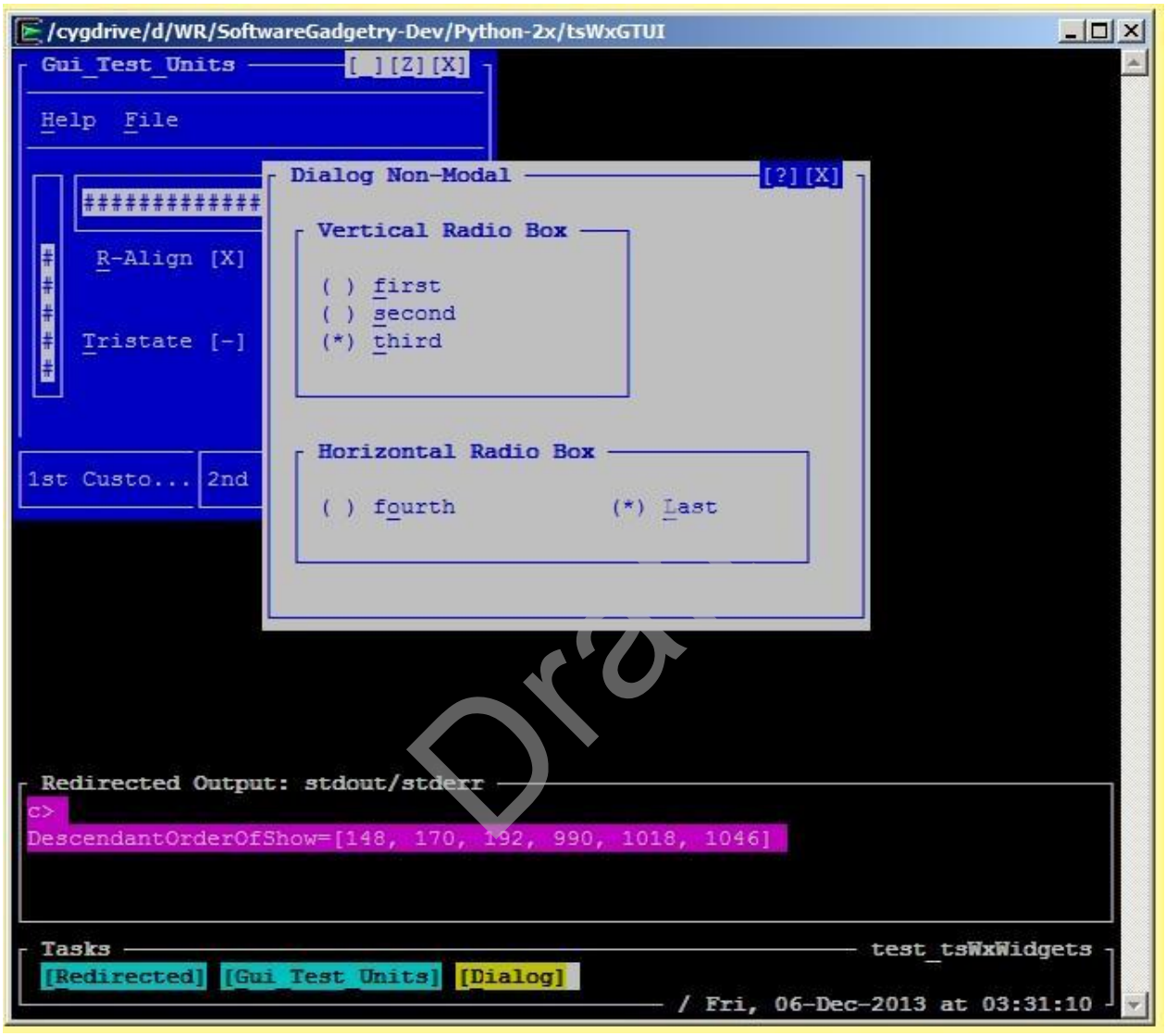
6 MAGENTA

7 CYAN

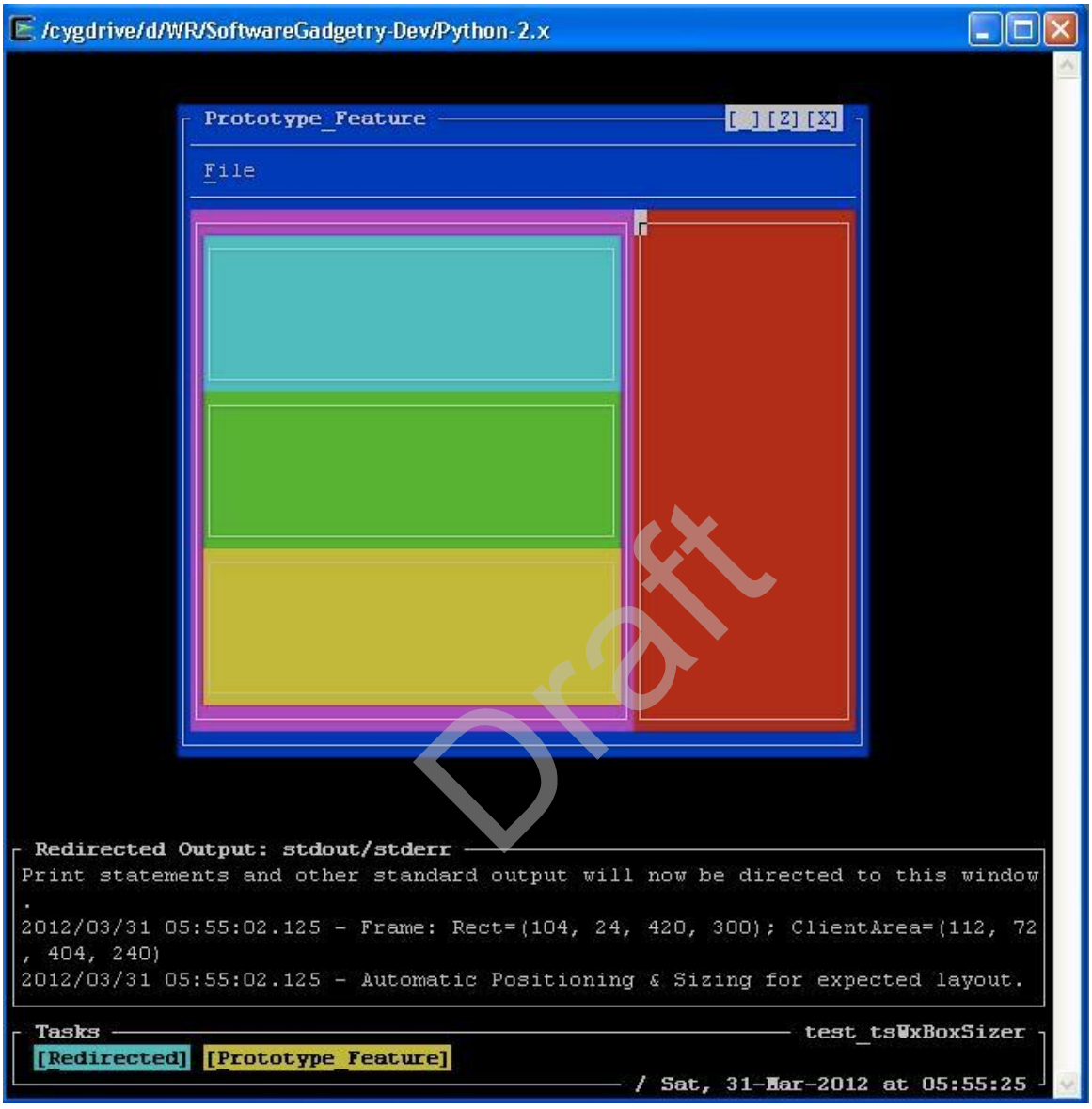
8 WHITE

NOTE: The following screen shots demonstrate various widgets on a terminal that reports having colors. It is the same application that runs on terminals that report NOT having colors. The application remains unaware of the presence or absence of colors.

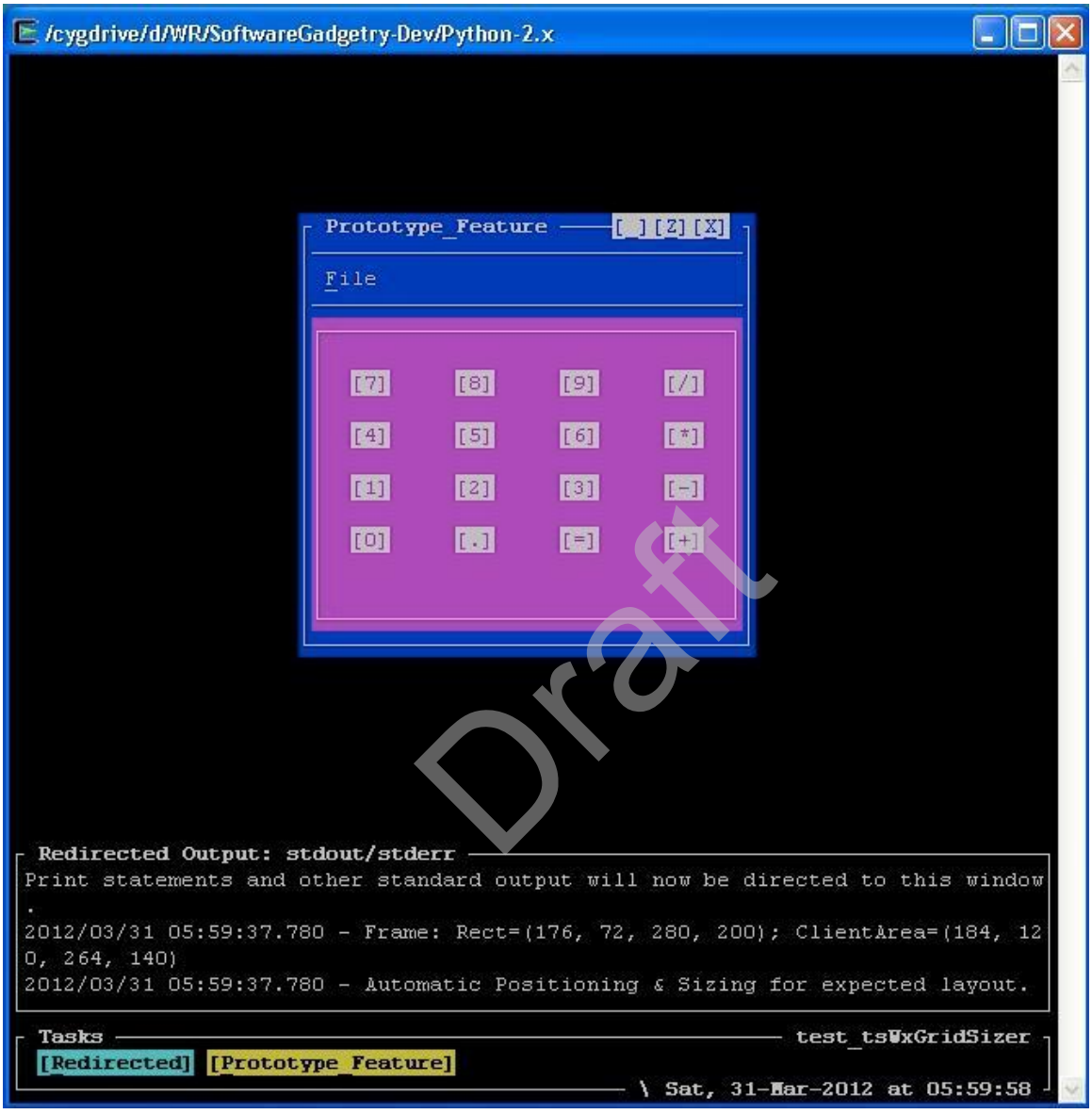
6.1.1 test_tsWxWidgets using xterm (via Cygwin mintty)



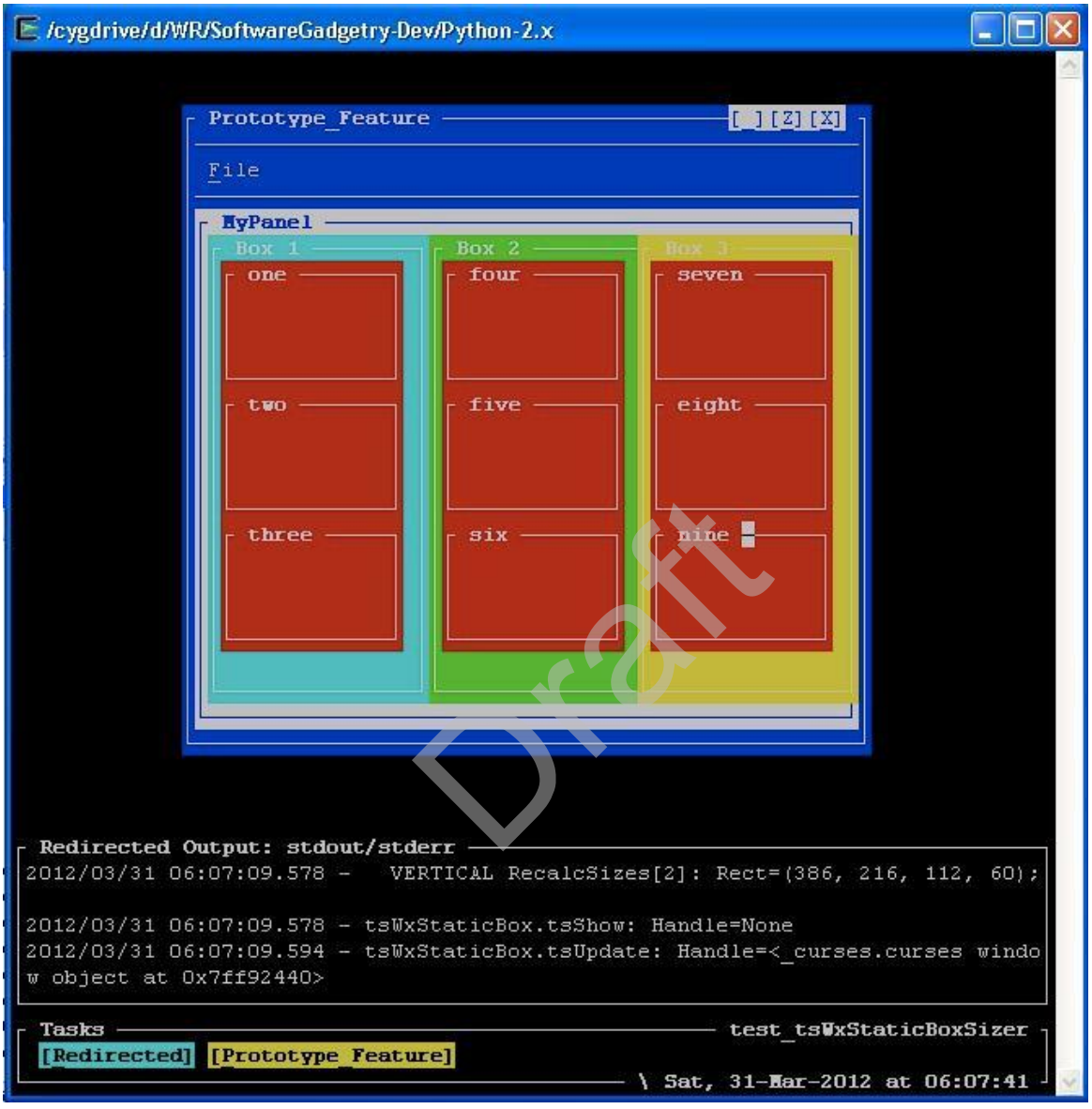
6.1.2 test_tsWxBoxSizer using xterm (via Cygwin mintty)



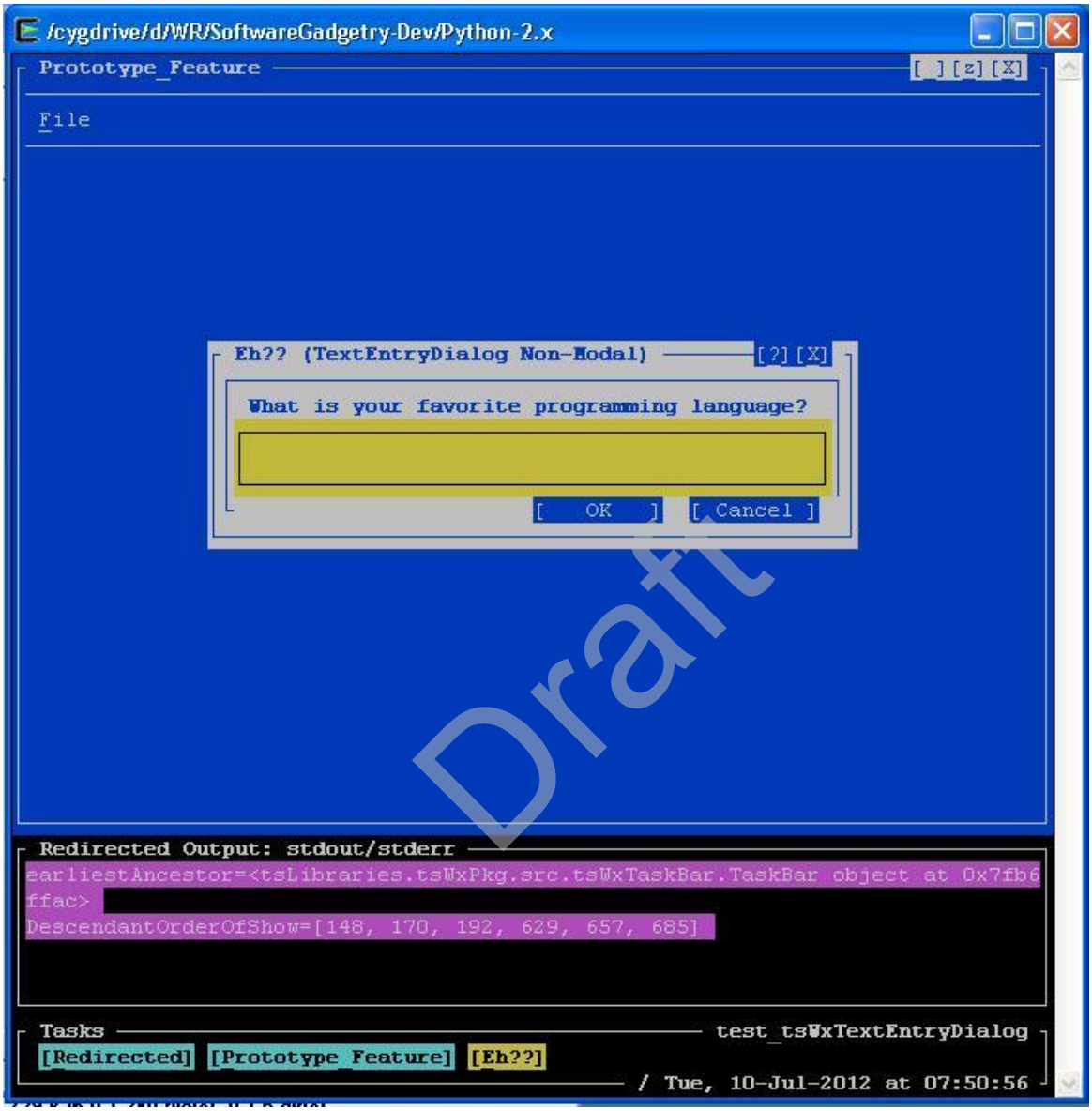
6.1.3 test_tsWxGridSizer using xterm (via Cygwin mintty)



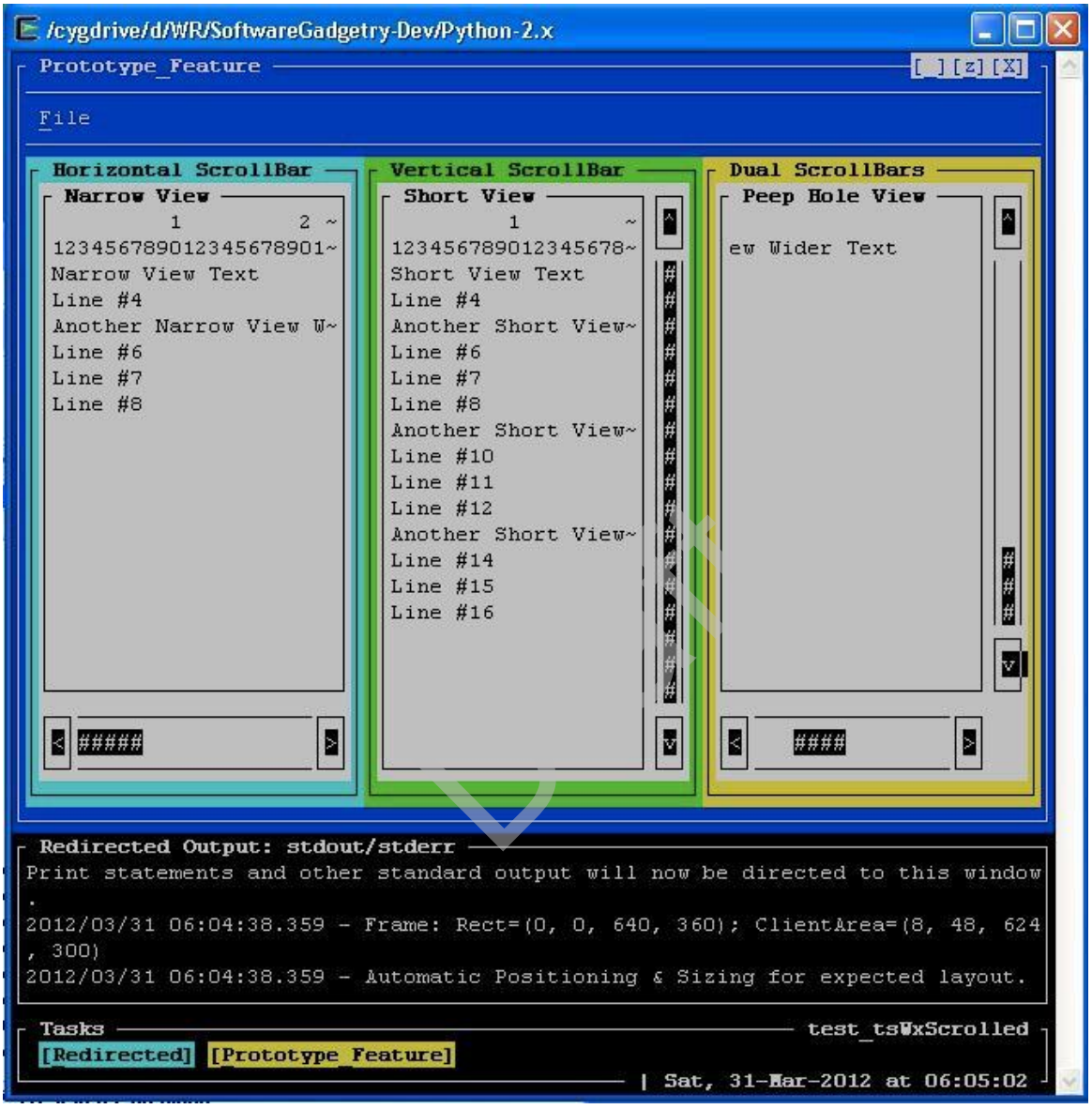
6.1.4 test_tsWxStaticBoxSizer using xterm (via Cygwin mintty)



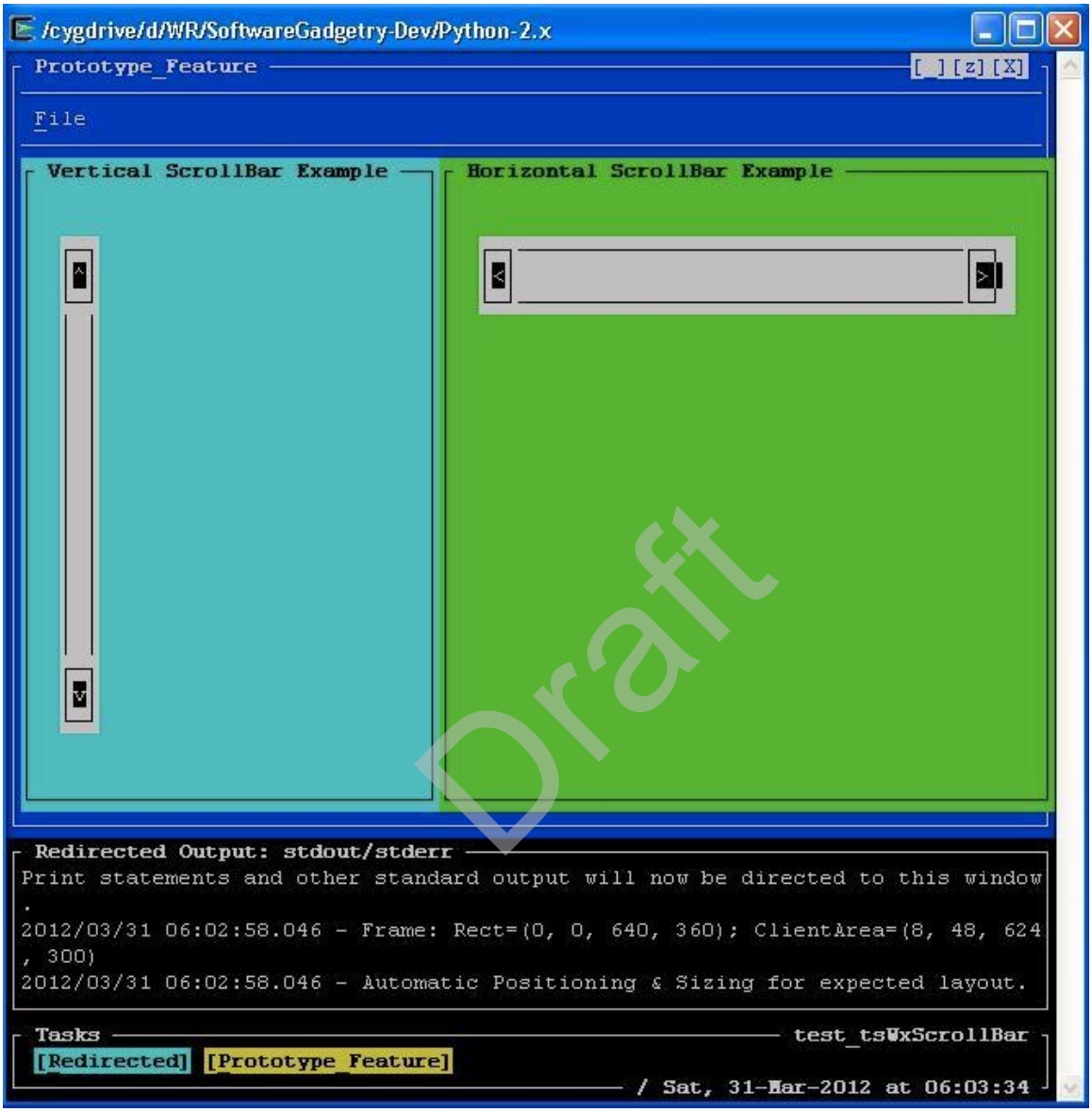
6.1.5 test_tsWxTextEntryDialog using xterm (via Cygwin mintty)



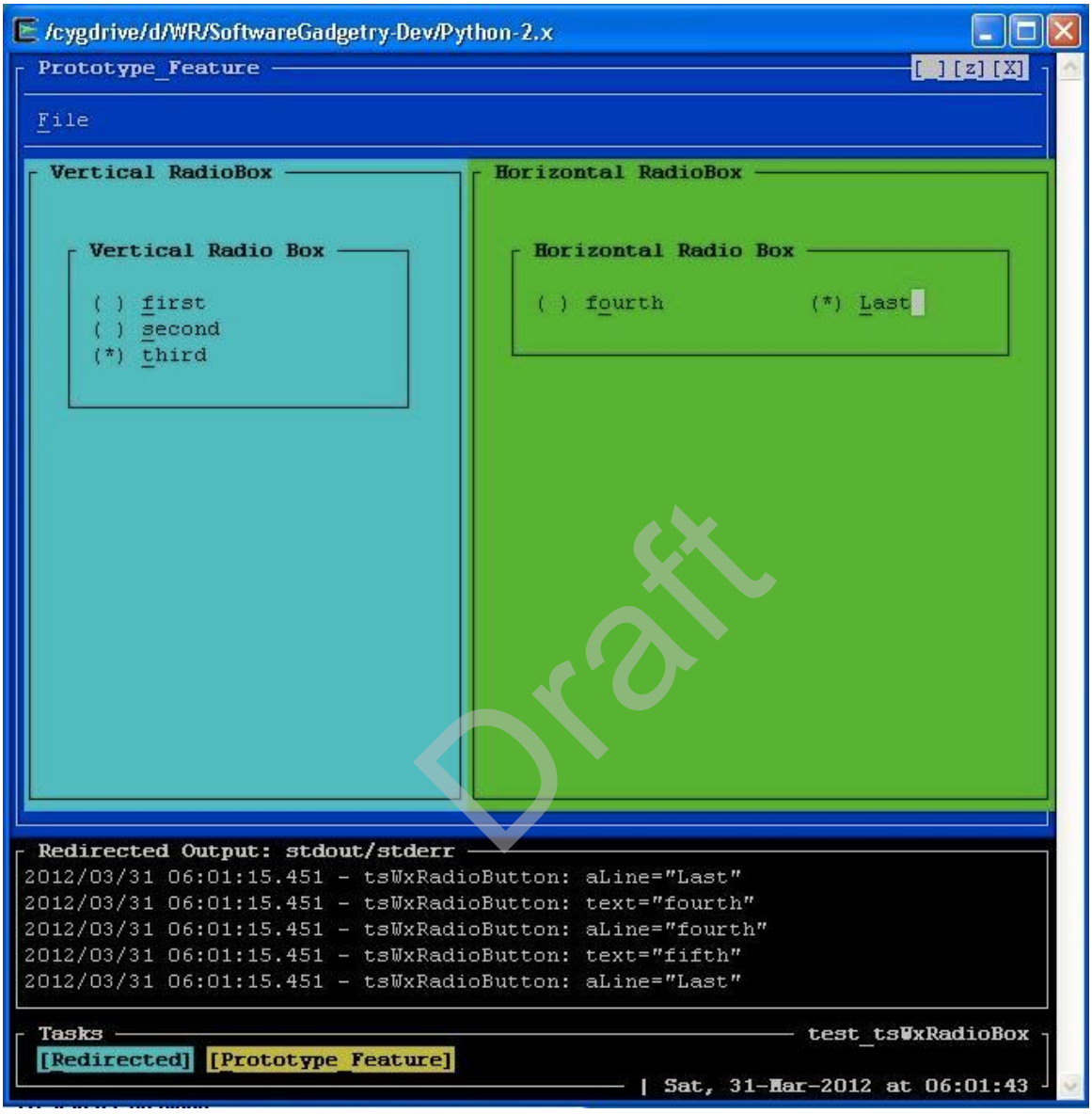
6.1.6 test_tsWxScrolled using xterm (via Cygwin mintty)



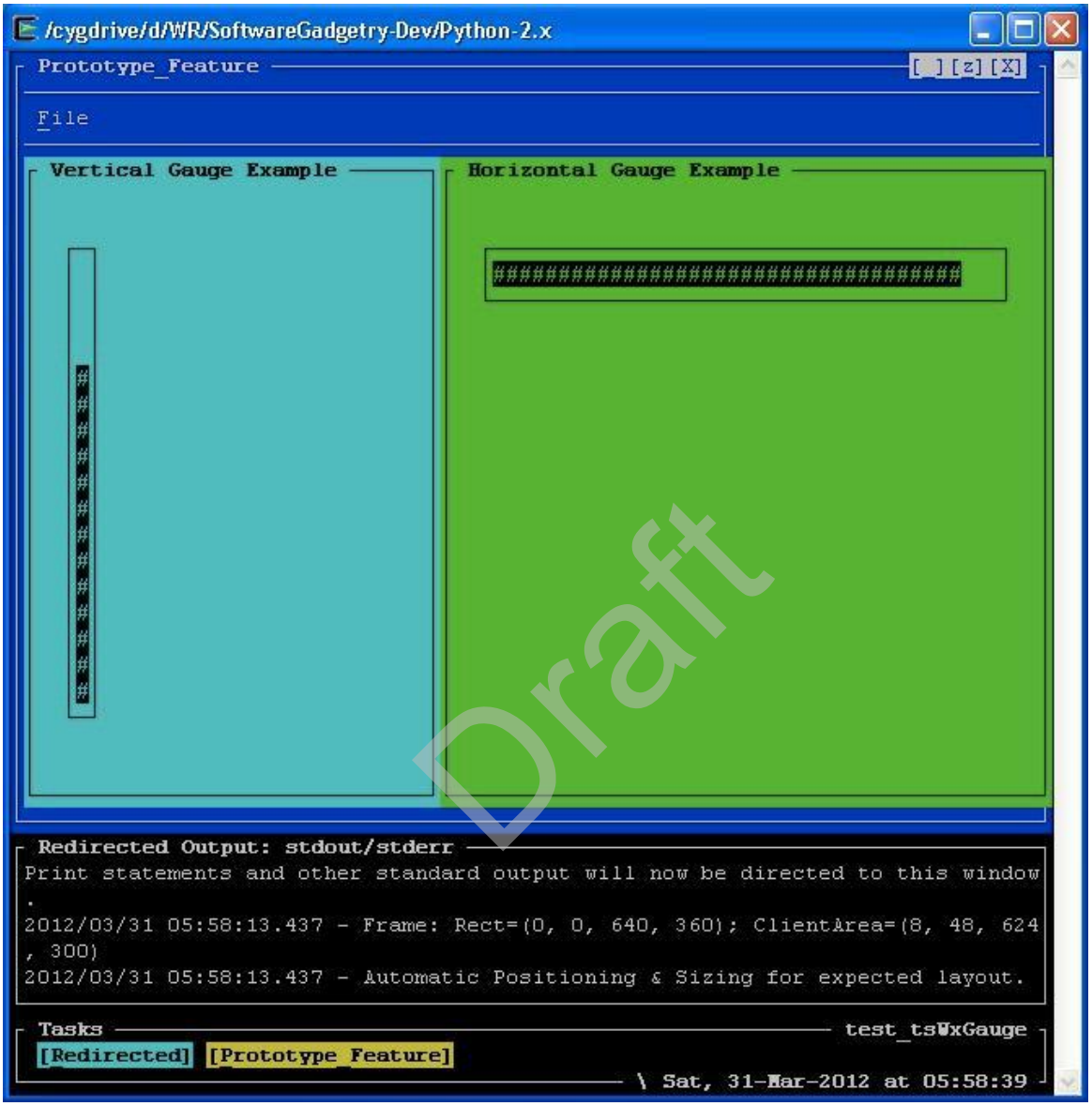
6.1.7 test_tsWxScrollBar using xterm (via Cygwin mintty)



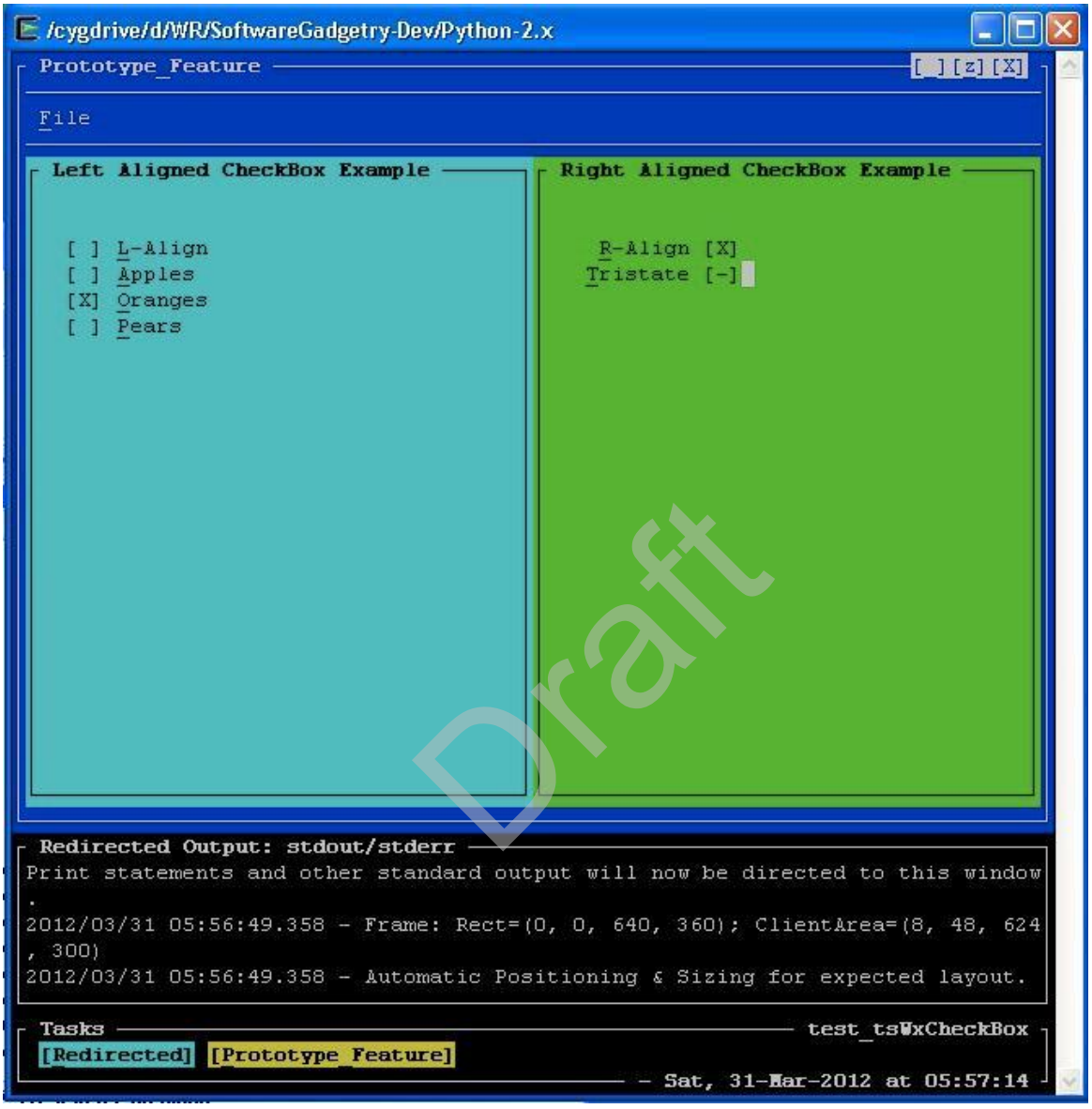
6.1.8 test_tsWxRadioBox using xterm (via Cygwin mintty)



6.1.9 test_tsWxGauge using xterm (via Cygwin mintty)



6.1.10 test_tsWxCheckBox using xterm (via Cygwin mintty)



6.2 VT-100 with 1-Color / 2-Color Pairs

From Wikipedia, the free encyclopedia:

"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.

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.

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. In addition, the VT100 provided backwards compatibility for VT52 users, with support for the VT52 control sequences.

In 1983, the VT100 was replaced by the more-powerful VT200 series terminals such as the VT220.

In August 1995 the terminal business of Digital was sold to Boundless Technologies."

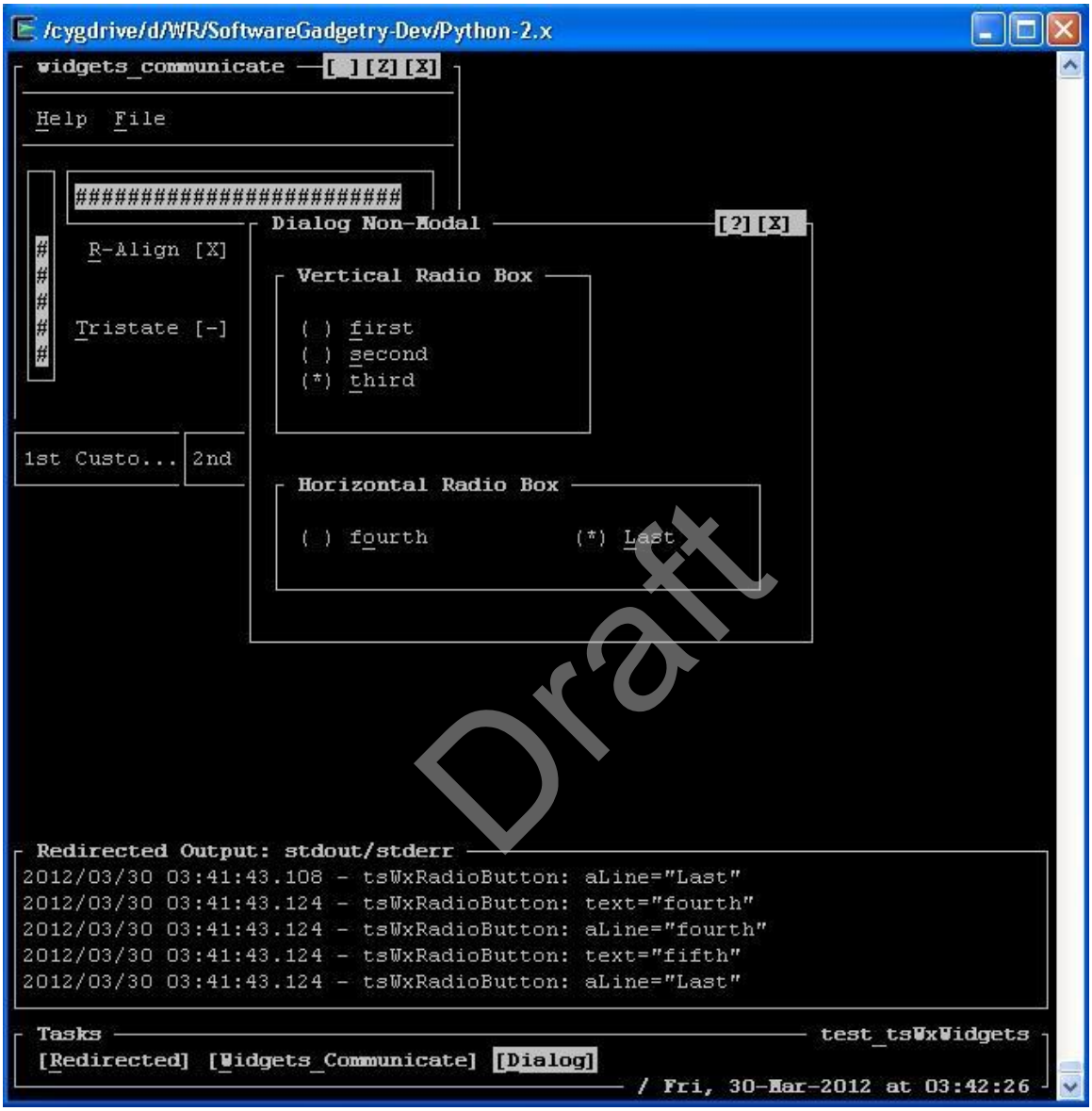
Typically, vt100/vt220 terminals support keyboard (without mouse) input and a display whose character cells consist of a single WHITE, GREEN or ORANGE color phosphor per pixel that are independently programmable in intensity so as to produce any of 2-colors (with their associated 2-color pairs) per character:

1 BLACK (phosphor pixel "OFF")

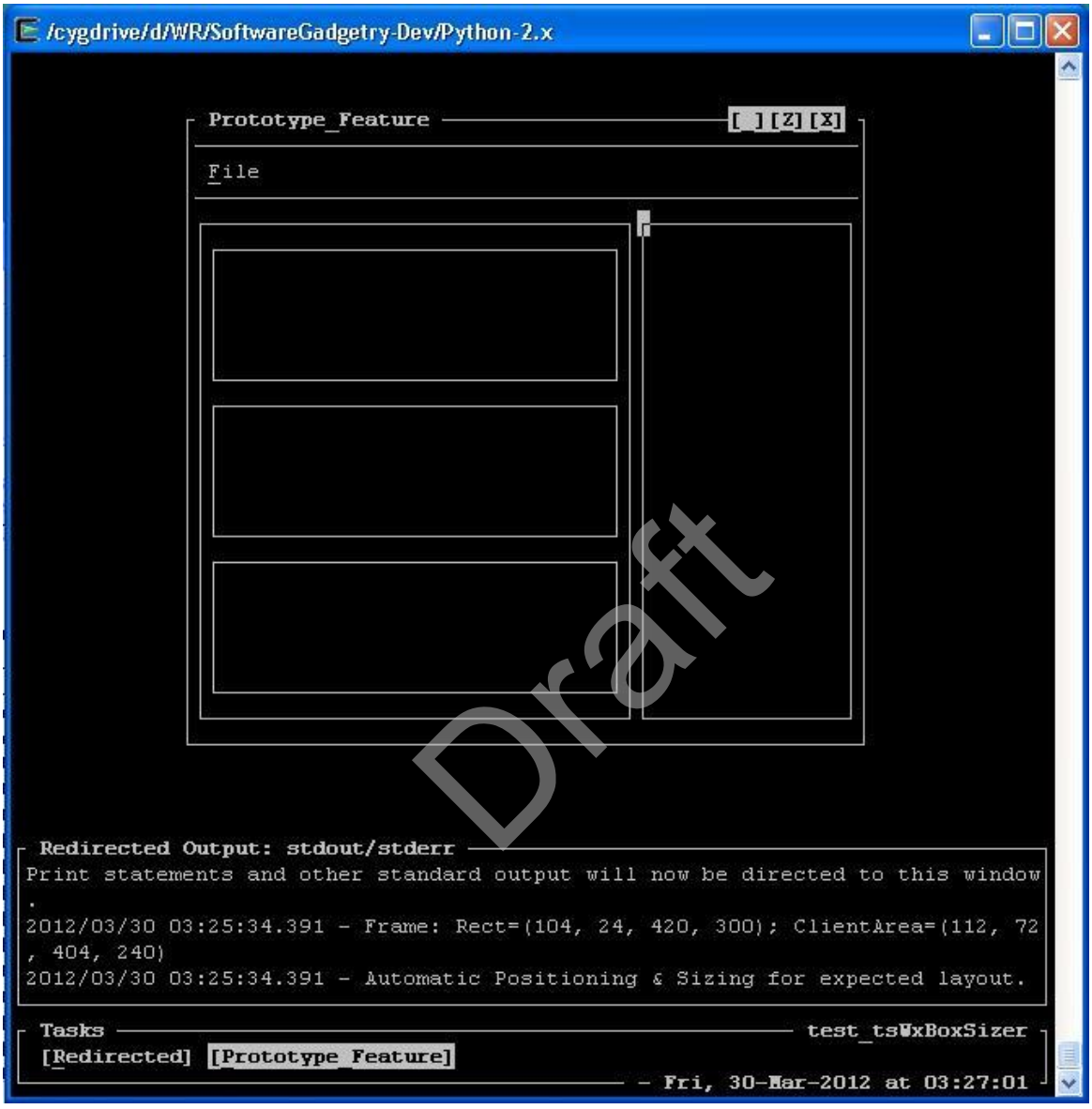
2 WHITE, GREEN or ORANGE (phosphor pixel "ON")

NOTE: The following screen shots demonstrate various widgets on a terminal that reports NOT having colors. It is the same application that runs on terminals that report having colors. The application remains unaware of the presence or absence of colors.

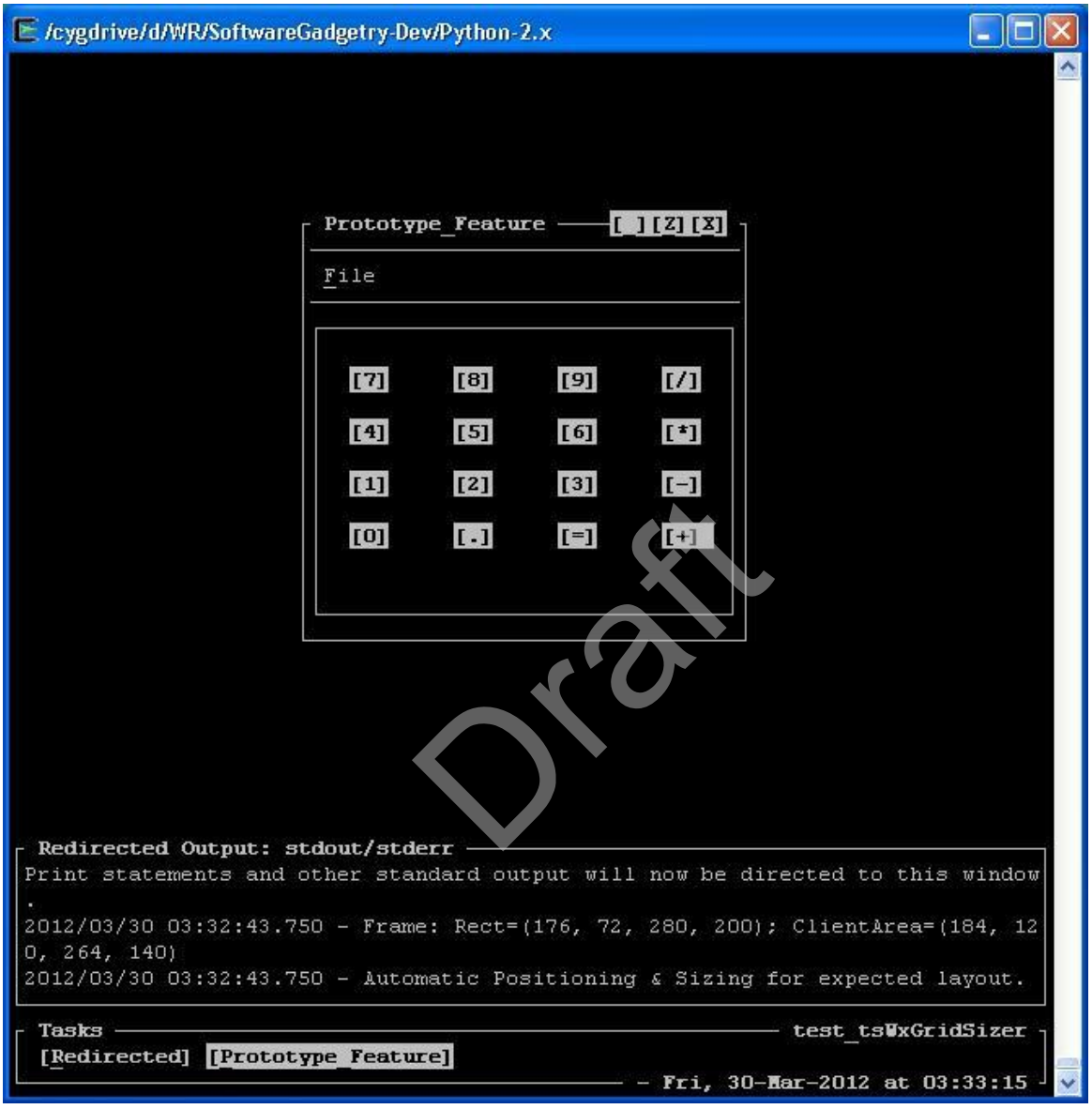
6.2.1 test_tsWxWidgets using vt100 (via Cygwin mintty)



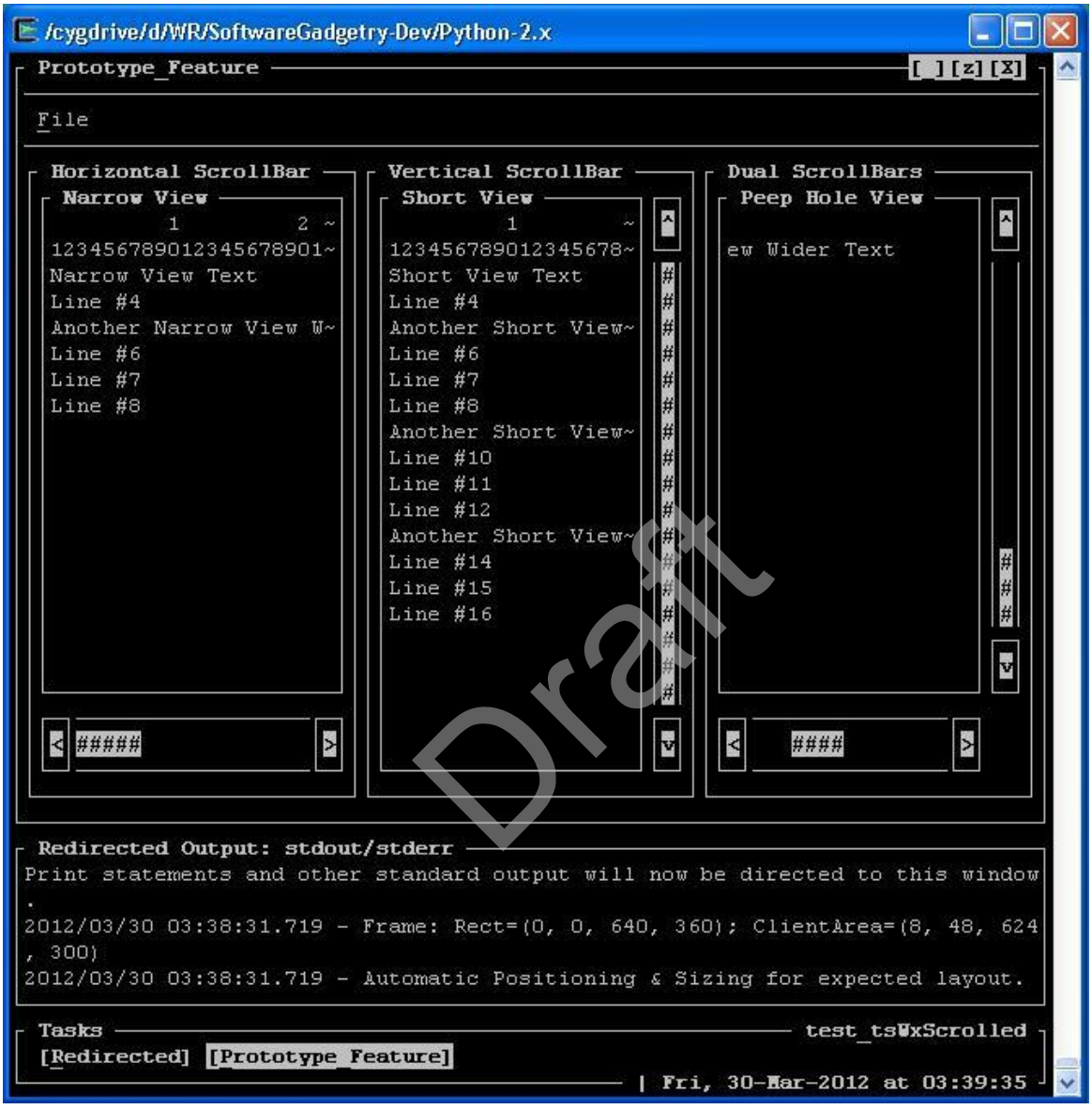
6.2.2 test_tsWxBoxSizer using vt100 (via Cygwin mintty)



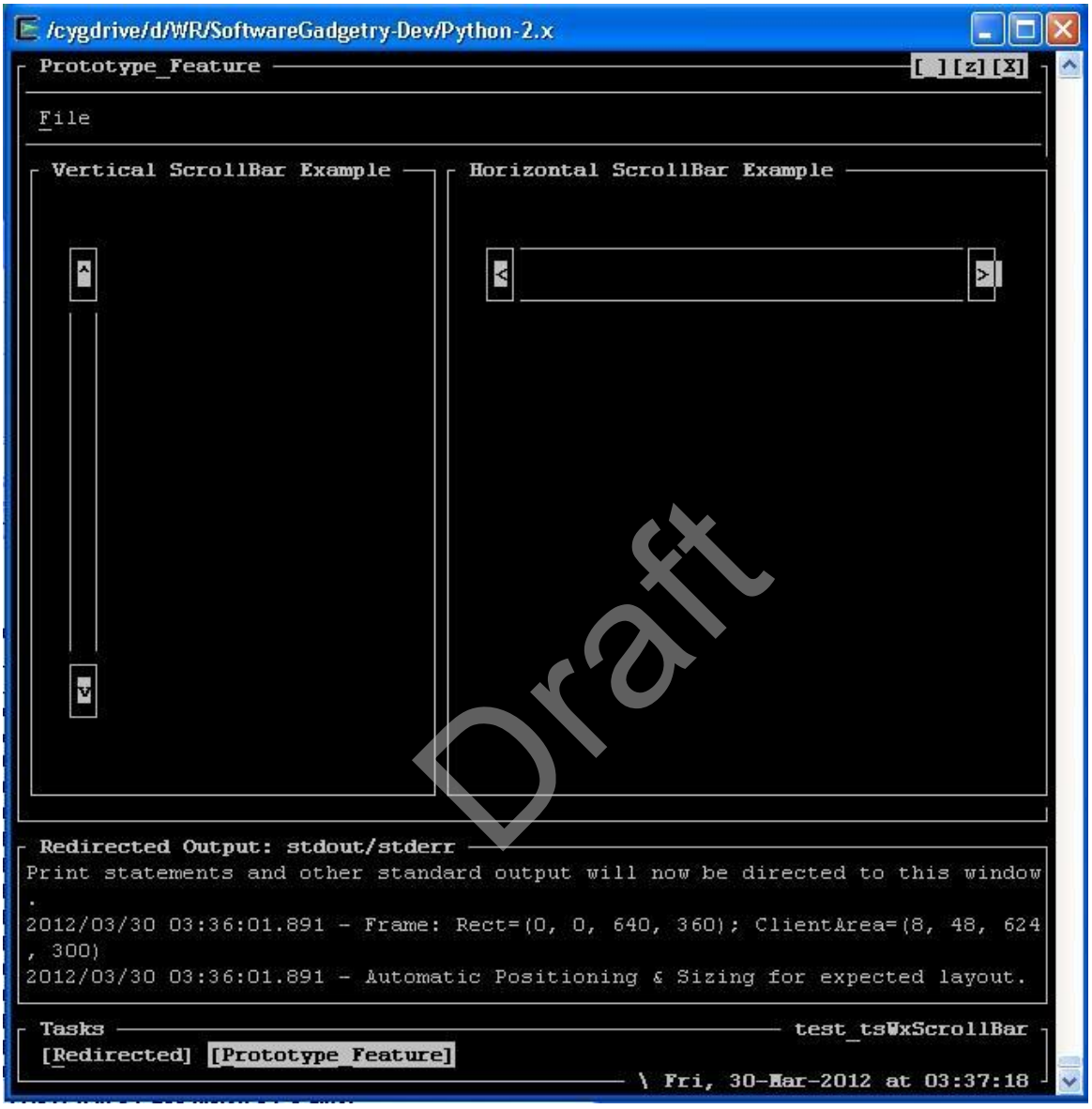
6.2.3 test_tsWxGridSizer using vt100 (via Cygwin mintty)-



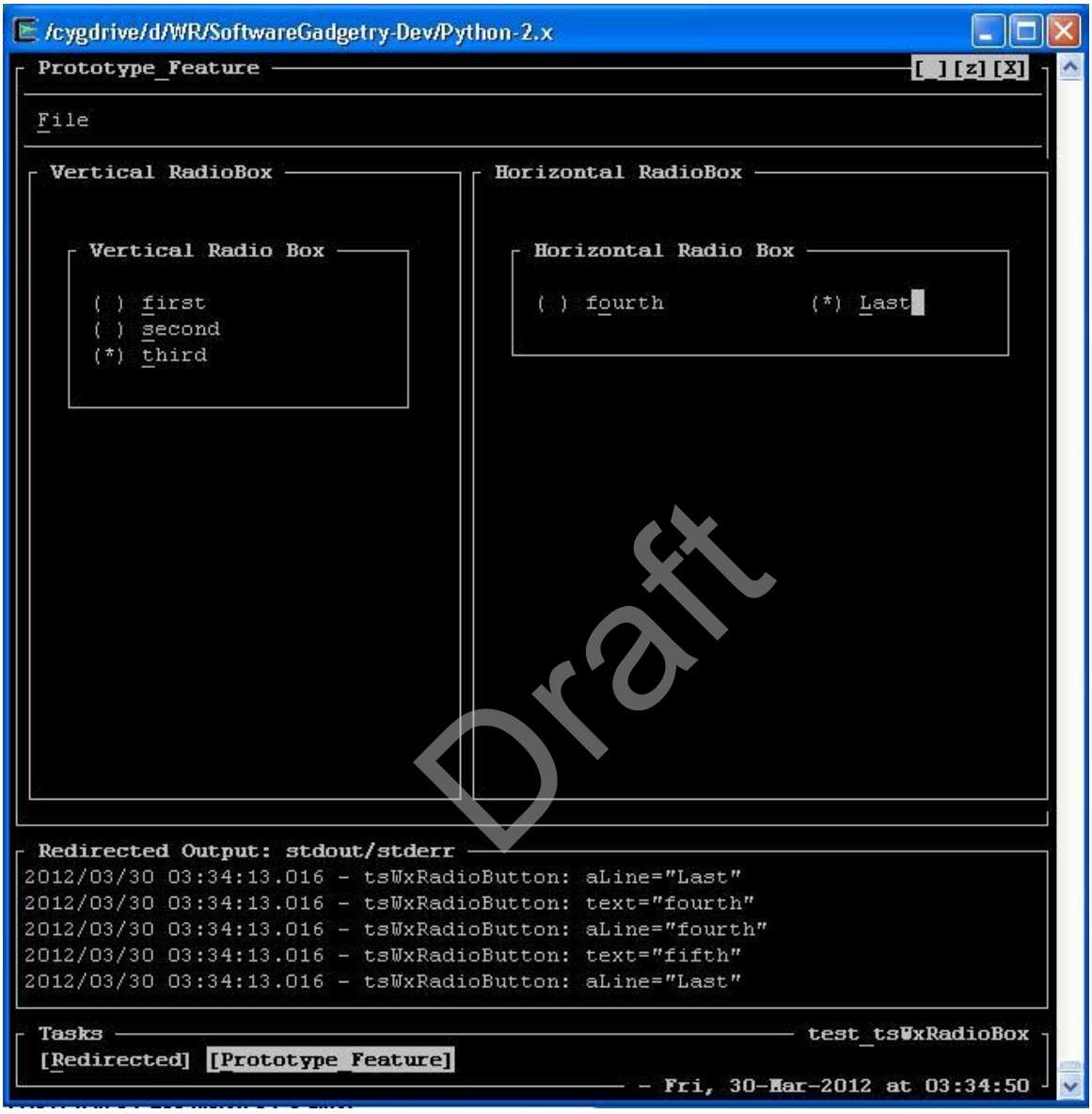
6.2.4 test_tsWxScrolled using vt100 (via Cygwin mintty)



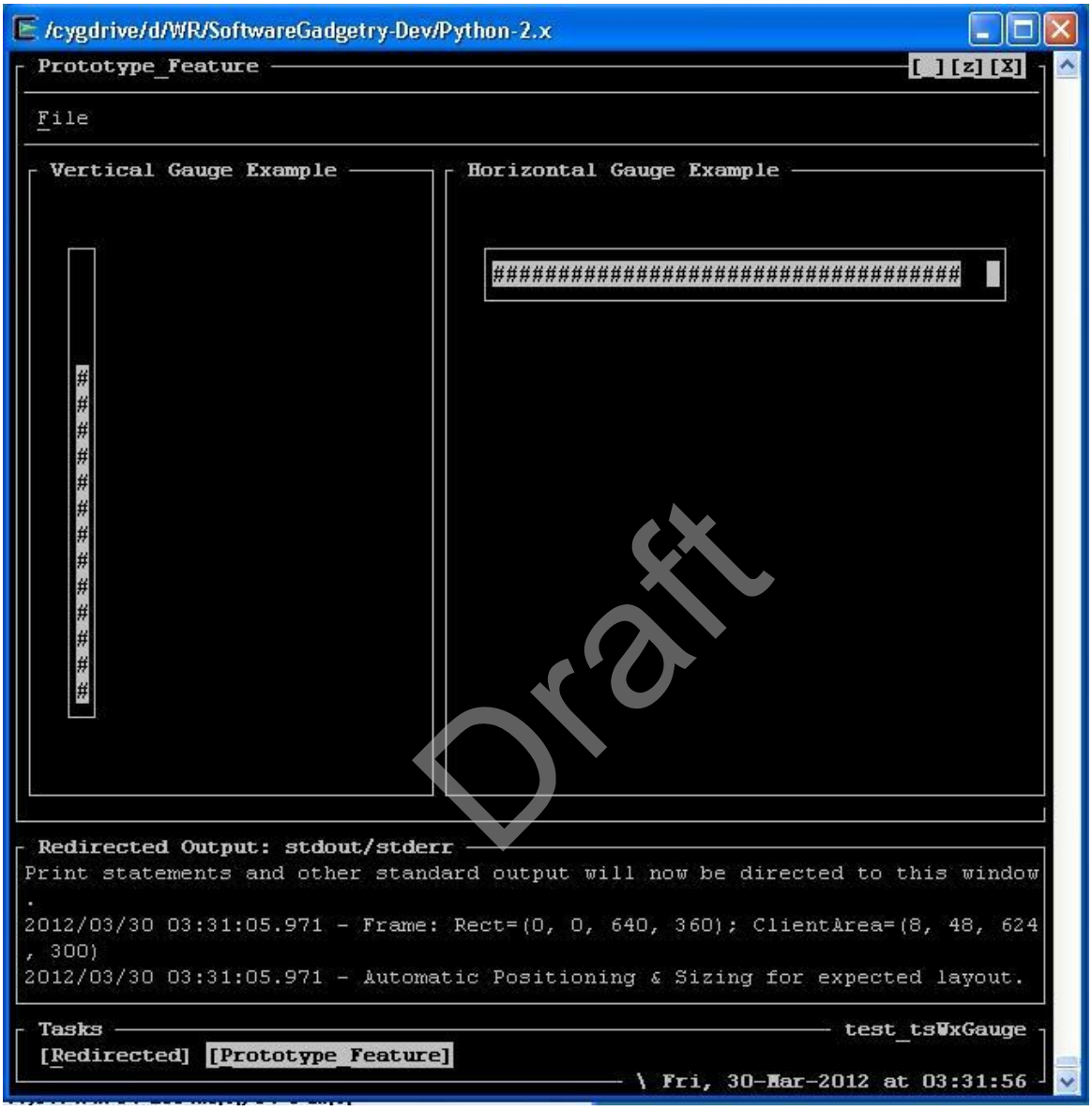
6.2.5 test_tsWxScrollBar using vt100 (via Cygwin mintty)



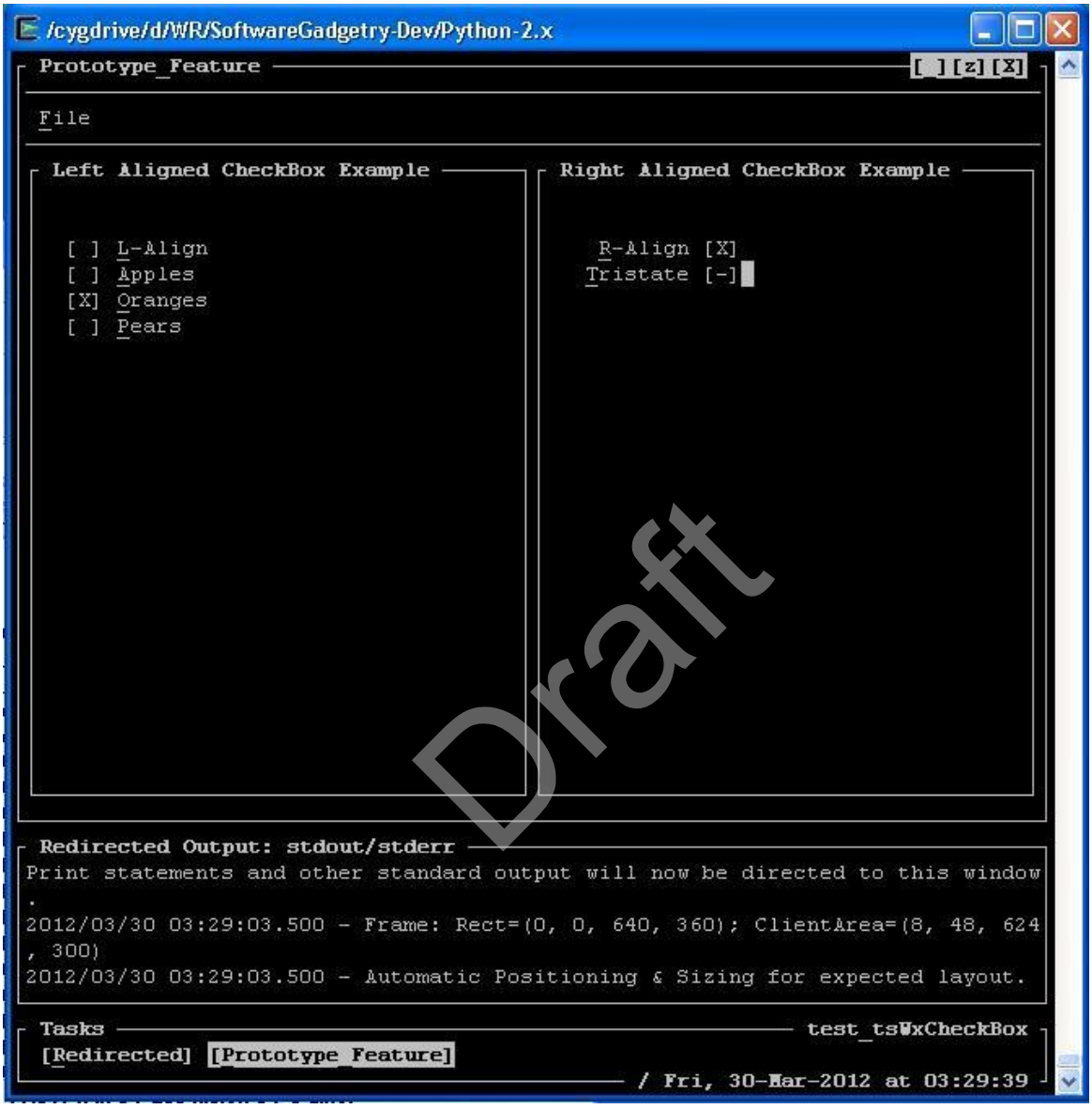
6.2.6 test_tsWxRadioBox using vt100 (via Cygwin mintty)



6.2.7 test_tsWxGauge using vt100 (via Cygwin mintty)



6.2.8 test_tsWxCheckBox using vt100 (via Cygwin mintty)



6.3 Multi-Session Desktop

From Wikipedia, the free encyclopedia

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

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

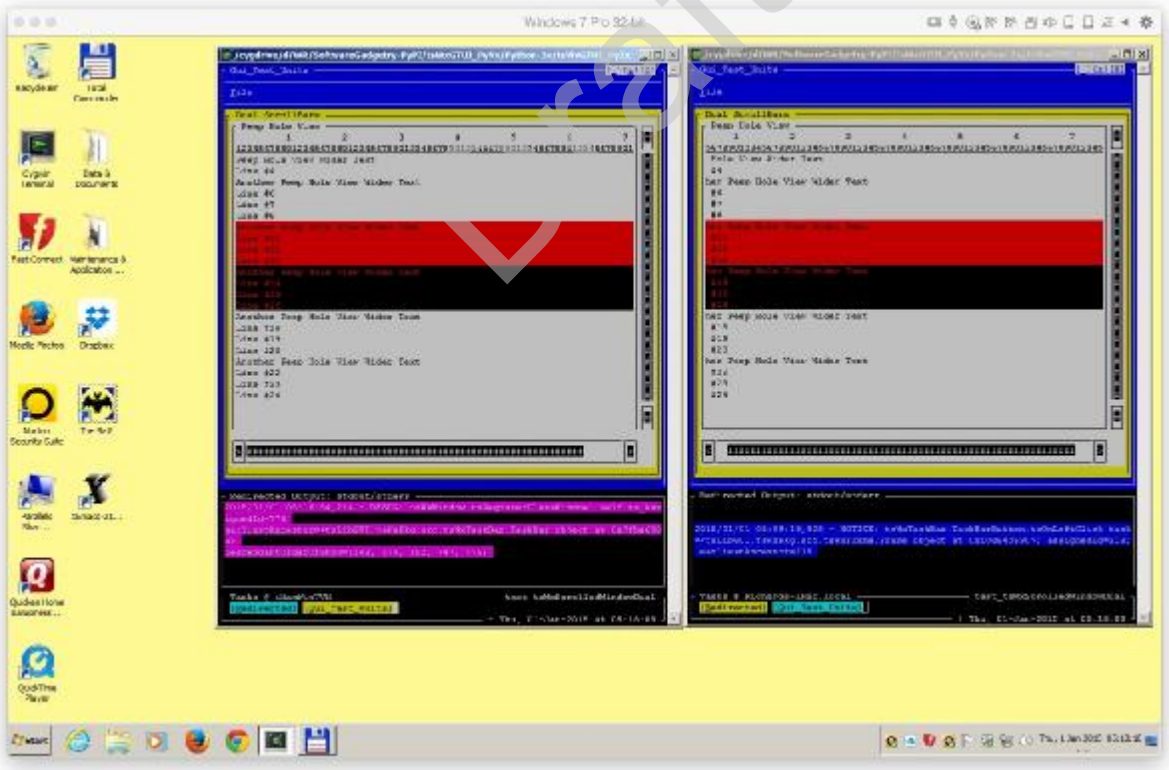
Communication sessions may be implemented as part of protocols and services at the application layer, at the session layer or at the transport layer in the OSI model.

- Application layer examples:
  - HTTP sessions, which allow associating information with individual visitors
  - A telnet remote login session
- Session layer example:
  - A Session Initiation Protocol (SIP) based Internet phone call
- Transport layer example:
  - A TCP session, which is synonymous to a TCP virtual circuit, a TCP connection, or an established TCP socket.

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

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

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



The Sample Screenshot above shows a Windows 7 Desktop with:

1 A local Windows host with Python 3x session on left; and

2 A remote Mac OS X host with Python 2x session on right.

3 Each session consists of

a) a wxPython-style Frame named "Dual ScrollBars" containing scrollable text with optional color markup.

b) a wxPython-style Frame named "Redirected Output" containing date and time stamped event messages, with optional with color markup, that scroll up when new events are registered.

c) a Host Desktop-style Frame named "Tasks @ Host Name" and Application Name with buttons to shift focus from background to foreground. There is also a spinner (to indicate the frequency or absence of idle time) and the current date and time (to indicate when the display was last updated).