Vanilla Application Window
External Reference Specifications

October 31, 1983

By

Yu-Ying Chow

1. Project Definitions

1.1. Product Abstract

A vanilla Pascal program is a text-oriented program that uses only WRITELNs to display its output, and only READLNs to get input. This type of program does not have the ability to access window manager libraries to directly interact with window, menu or the mouse. The vanilla application window (sometimes called the vanilla window or the application window) provides a vanilla Pascal program with a way to direct its output to a window and get its input from the keyboard, following the Office System user interface standards. The vanilla window gives the user of this type of program the capability to control its output window with menus and the mouse. With the vanilla window such programs can be ported to run in the Office System without any modifications in the source code as long as they have run in the Workshop.

For programs that use the QuickDraw (sometimes called vanilla QuickDraw programs) and run in the Workshop today, the vanilla window also provides a way to direct its graphic output to a window. In additions, for programs that use the Hardware unit (HWINT) to get the mouse location (MouseLocation) and the mouse events (KeybdEvent), the vanilla window also provides a way to get the mouse coordinates. However, the vanilla window will not support programs that use the Hardware unit to do other hardware control that affects the entire system. These programs have to be converted to use the ToolKit before they can run in the Office System. Other programs that need to have their own menus that are different from the vanilla window menu will have to be converted to use the ToolKit as well. There will be only one window for text and graphic output for each vanilla QuickDraw program, even if the program uses both WRITELNs and the QuickDraw.

The vanilla window emulates a terminal display. The text output that would be scrolled off the top of a typical terminal display is saved by the vanilla window. This output can be scrolled back at any time. The vanilla window provides a *Standard Terminal* that performs common terminal functions such as positioning the cursor and clearing the screen, etc. It also enables third parties to write specific terminal emulators for the vanilla window. We are providing a VT100 and a Soroc terminal compatible emulators with this product.

Each vanilla program is associated with a document which contains the output display of the program. Either all or part of the output from a vanilla program can be saved in the document. The document can be manipulated the same way as other Lisa documents. It can be "Set Aside" or "Saved & Put Away" and opened later to see the output of the previous run. In addition, each vanilla program can deal with OS files in the same way as it does in the Workshop. This separation of files and documents will be dealt with in the future Desktop Manager. In this product release, the documents will be manipulated in the Desktop, the OS files will be handled in the Workshop Shell running in the Office System.

running in the Office System, the programming environment is the same as the end user environment. The Lisabug and the NMI key capability should be available to aid debugging and interrupting a run-away program.

1.4. Related Documentations

ToolKit Documents - ToolKit team, 5/12/83

Lisa User Interface Guidelines, Part I - Alpha Draft, 4/29/83

2. Programmable Interface Specifications

The vanilla window unit provides interactions with windows for both vanilla programs and users of these programs. The programmable interface, READLN, WRITELN and screen control, etc., is described first. The user interface is described later.

Each vanilla window enables users to see through to the program output. This output is either from the program's WRITELNs or from its QuickDraw calls. These two different types of output can be presented in two different *Panels*, text panel and graphic panel, each with independent scrolling capability. In addition, there is a panel displaying the input buffer which contains the not-yet-consumed input, the input panel. The input panel will always be present. The text panel and graphic panel can be reconfigured to be separate or be combined. The following sections describe these three panels in detail.

2.1. Text Input and Input Panel

There are two sources of input to a vanilla Pascal program running in the vanilla window. One is from the user typing on the keyboard; the other is from the "Paste" command. As it will be described in more detail in the User Interface section, the "Paste" command will put the selected data in the program's input buffer. When the program does a READ, the data in the input buffer, if any, will be consumed to satisfy the READ first. And then if the input buffer is empty, the program will wait for additional input from either typing or pasting.

This input buffer, containing the not-yet-consumed input, will be displayed in the input panel.

2.2. Text Output and Text Panel

The text panel contains the WRITELN output of its program. This panel corresponds to the Pascal built-in device OUTPUT and the logical device '-CONSOLE'. The panel emulates a terminal display. It has a *screen area* at the bottom that corresponds to a terminal's screen. The default size of the

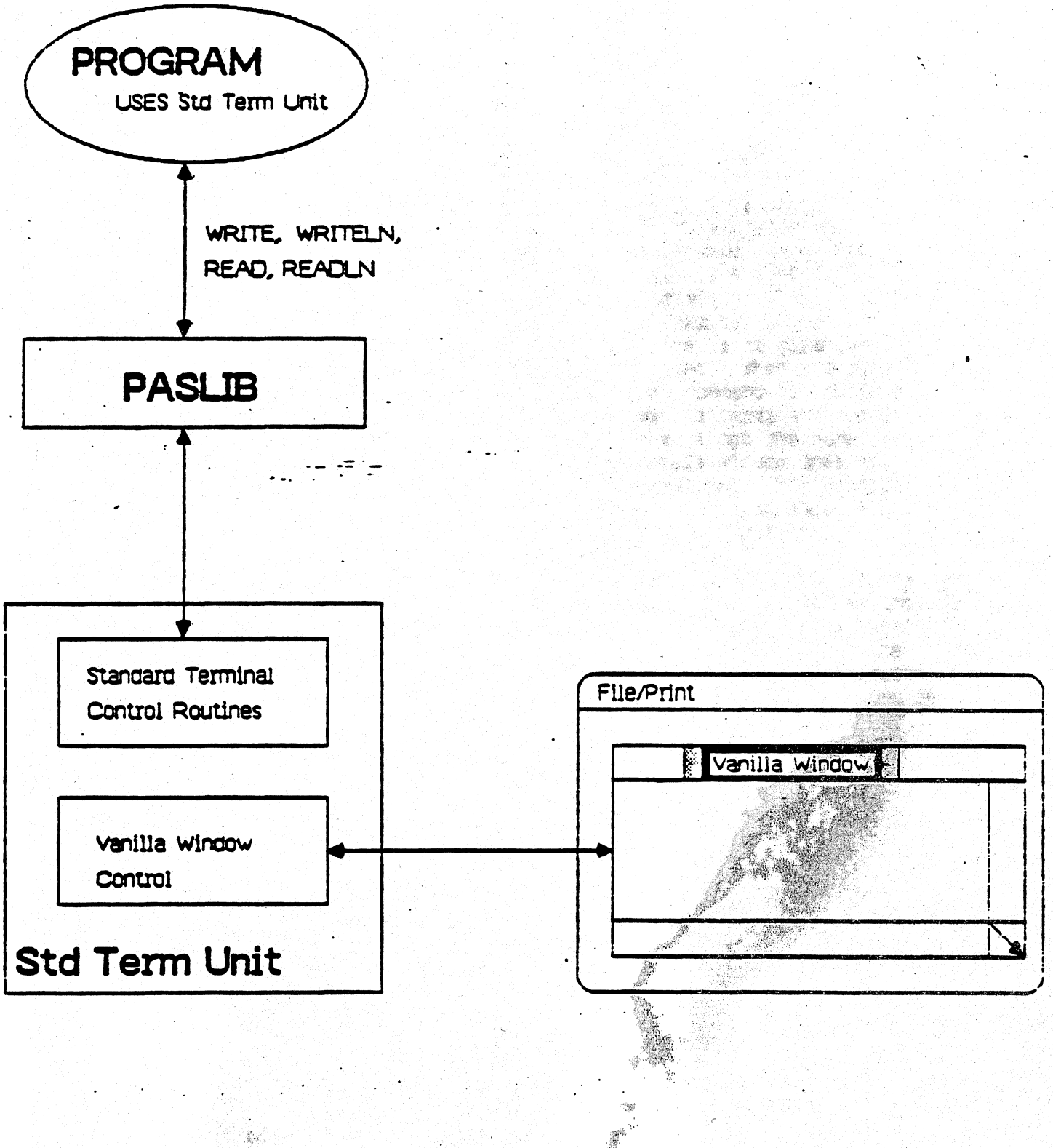

Figure 1. Data flow of WRITELNs and READLNs to a vanilla window from a program that USES the Vanilla Window/Standard Terminal Unit.

line feed). It does not interpret any input key. (Programs that use READ and READLN will have the backspace key processed by the PASLIB on input, i.e., the backspace key is never seen by programs. This is due to the Pascal run time library, not the Standard Terminal.)

The standard Lisa applications provide "Apple-Period" key combination to terminate long operations. Most of the Workshop programs have provided this feature as well. But such feature has to be explicitly built in the application programs. Programs from other machines do not have such features unless they are modified. To be as much as consistent as possible with the standard Office System applications, the vanilla window will provide an option to detect the "Apple-Period" key and suspend the program. This will give users immediate reponse when they run those programs that do not detect this key. (Third parties, when porting programs from other computer systems, will be recommended to put in this feature, but there is no garantee and no way to detect that such convention is followed.) When a program is suspended, the user can select the "Resume" command to continue the program execution, or the "Save & Put Away" to terminate the program. Note that suspending program execution here is not the same as terminating long operations, since the vanilla window does not have any access to the application pograms' operations. This program suspension can be disabled for programs that have already responded to this key. See section 2.4 for the procedure definition.

The Standard Terminal has a set of Setup features with which the user can reconfigure the Standard Terminal. This set of features includes: adjusting the maximum number of columns in the Standard Terminal to either 80 or 132; selecting whether to wraparound or not at the maximum column position of the screen; setting the tab positions. The users can invoke Setup features from the "Setup" menu described below. Terminal emulators can override the standard configurable setup features.

The following procedures describe the standard screen control functions that are callable from programs and terminal emulators. In addition to these functions, the state of the Standard Terminal is also available to terminal emulators, such as the current cursor position, etc. Third party software can override some of these functions to emulate specific behavior in their terminals. These procedures are not to be overridden except where indicated.

TWrite (string), StrWrite (string), CtrKeyWrite (char)

These procedures display the string passed in the window. The string may not contain any escape or control sequence, except a single control character that belongs to the standard control key set. TWrite is the one that will be called by the WRITE and WRITELN from the PASLIB. It then calls the StrWrite or CtrKeyWrite depending on the string. StrWrite assumes that the string passed has no control keys, and the string will be displayed starting from the cursor position. The wraparound feature will be checked and performed accordingly. StrWrite is a standard function not to be overridden by other terminal emulators. CtrKeyWrite handles the standard control keys. Writing a line feed character when the

less than 0, the cursor position will be used. If the parameter clearall is true, all the tab positions will be cleared.

GetLine (y, string, delete), PutLine (y, string, insert)

The output lines in the screen area and the buffer area can be rearranged from terminal emulator units. These are particularly useful when terminal emulators need to set up a scrolling region within the screen area. The lines that will be scrolled off the top of the scrolling region can be deleted (using GetLine with delete being true) and inserted (using PutLine with insert being true) at the bottom of the buffer area if the lines are to be saved. The lines in the buffer area are referenced using the home position as the base, i.e., the line number is -1 for the last line of the buffer area, etc. GetLine and PutLine can also be used to rearrange characters within a line.

RedrawScreen, RedrawLine

At any time when the screen area or the cursor line is changed by using the GetLine and PutLine, the screen image can be redrawn by these routines. RedrawScreen will draw the entire screen, and RedrawLine will draw the cursor line.

ChangeAttributes (newattr), ChangeMaxColumns (maxcolumns)

The Standard Terminal provides several character styles and fonts (fixed pitch only). ChangeAttributes can be used to change either the character style or the character font. The new attributes will apply to characters written from the current cursor position until the next call to ChangeAttributes. ChangeMaxColumns is used to change the maximum number of columns from 80 to 132 or back.

StopOutputKey (ch), StartOutputKey (ch) : Boolean

The output display can be temporarily stopped by a control key entered by a user, and can be restarted again by another control key entered by the user. The Standard Terminal will provide this capability for standard control keys : Control-S and Control Q, i.e. Apple-S and Apple-Q. StopOutputKey and StartOutputKey return true when the input character is Control-S and Control-Q respectively. The display control in the Standard Terminal unit will make calls to these predicate routines and stop or start the output display accordingly. Terminal emulators can use other control keys by overriding these functions.

2.2.1.2. VT100 ANSI Terminal Emulator

The VT100 ANSI terminal is popular in the industry. We will provide a VT100 ANSI compatible terminal emulator to help port a large number of vanilla Pascal programs to the Office System. This will also demonstrate an example for other terminal emulators.

This emulator will exist in a separate unit. It interprets all the VT100 and VT52 escape sequences, except the ones related to the host

inside the graphic panel. If the mouse is not inside the panel, a (-1, -1) point is returned. This procedure emulates the MouseLocation in the HWINT unit. The MouseDown function returns true when the mouse button is pressed, the corresponding mouse coordinate is also returned. Application programs that need to track the mouse can call this function repeatedly until the mouse button is pressed. Programs that are calling MouseLocation and KeybdEvent now will have to be modified to call MouseCoordinate and MouseDown respectively. Since the mouse is not used for making selection in the graphic panel (the menu command "Select All Graph" must be used to make selection), there will be no conflict in using the mouse.

For programs that need to use the mouse more than the capability provided above will have to be converted to use the ToolKit before they can run in the Office System. For programs to run in the vanilla window, they can not access the low level event queue to get the mouse and the diskette button events. They can not change the system clock. They can not display the hourglass busy cursor since it is used by the system to show that the system is busy. The kind of hardware access allowed will be that which does not affect the entire system or other applications, e.g. the speaker, the microsecond and millisecond timer, etc.

The graphic support will be in a unit separate from the text support (the Standard Terminal) unit. A vanilla QuickDraw program has to USE both of these units to run in the vanilla window. Figure 3 shows the data flow from a program to its window.

2.4. Configure the Panels

The vanilla window will provide a procedure for programs to rearrange the panels and their orientations, and to change various feature mentioned above.

VAWConfigure (HavePanel, TGOrientation, HaveBuffer, PassApplePeriod)

The parameter HavePanel can be either to have both text panel and graphic panel or to have one panel for both text and graphic output. Programs, if have text panel and graphic panel, can specify to have the text panel above, below, to the right of, or to the left of the graphic panel via the parameter TGOrientation. Screen oriented programs can specify that no buffer area is desired by setting HaveBuffer to false. Programs that need to get the "Apple-Period" key can specify PassApplePeriod to be true. And the "Apple-Period" can be detected by calling PAbortFlag as defined in the Workshop manual.

Programs should make this call before generating any output. If this call is not made, the vanilla window will use its defaults. The default for the text output only programs will be one panel for text output, having buffer area, and no "Apple-Period" passed to programs. For graphic programs, the default is to have one panel only for both text and graphic output, no buffer area, and no "Apple-Period" passed to programs.

Figure 4 shows a few window lay out of different panel orientations.

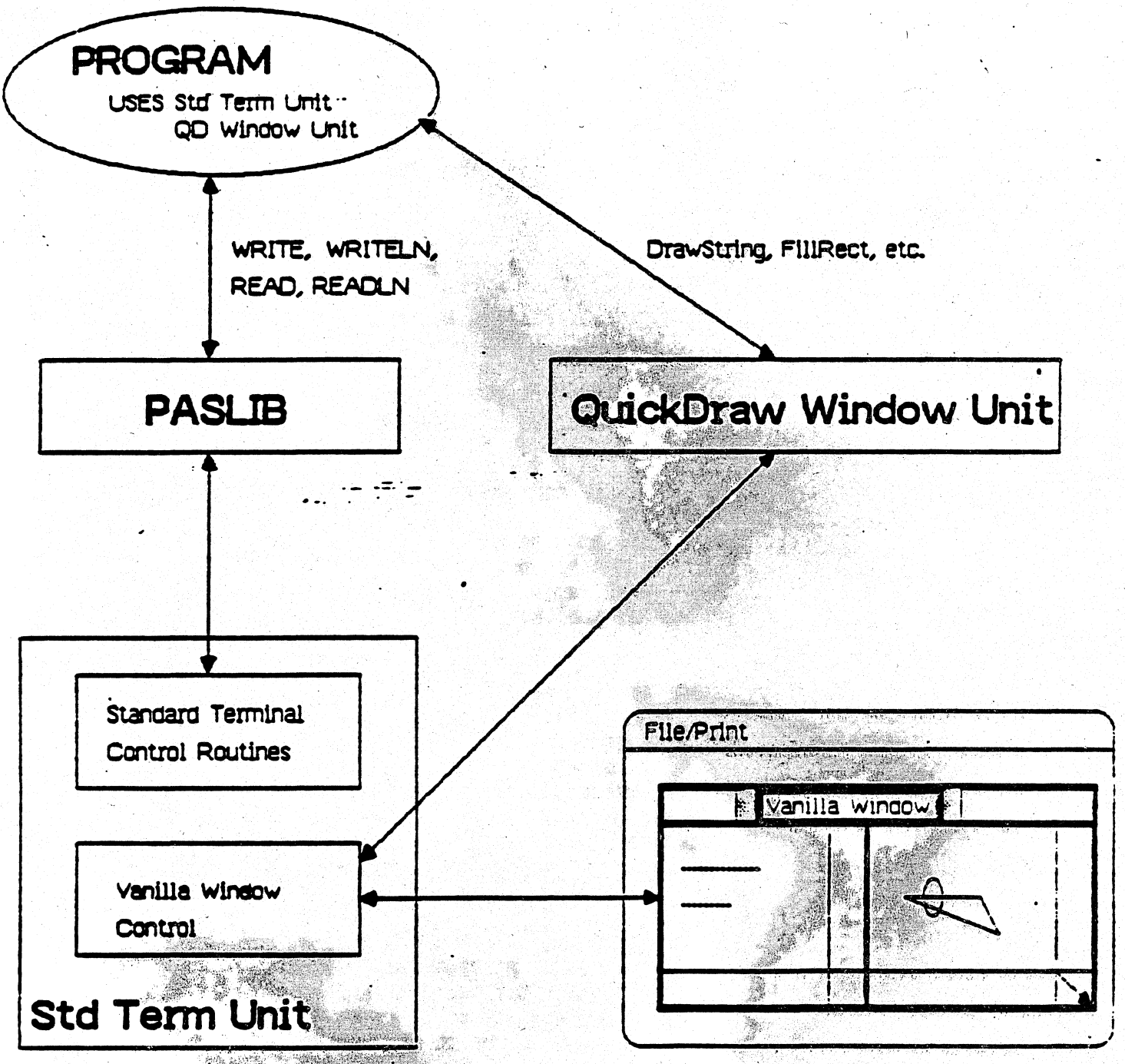

Figure 3. Data flow of WRITELNs, READLNs and QuickDraw output to a vanilla window from a program that uses the Vanilla Window/Standard Terminal Unit and the QuickDraw Window Unit.

3. User Interface Specifications

The vanilla window provides the Lisa standard user interface for vanilla programs. The window manipulation and mouse usage follow the standard user interface. The vanilla window also provides a menu bar for the user to interact with the vanilla programs.

There is an important difference between the vanilla programs and Lisa applications such that we need to provide some extra commands to allow users to fully control program execution and to make the vanilla window user interface be more consistant with other Lisa applications. A vanilla program, unlike a Lisa style program which always loops on getting events and processing events, runs from the beginning to the end. When the program reaches the end, it has completed executing, and is not reponding to any key inputs. We do not want the window to close before the user has a chance to look at the program's output. At this stage, we need to make the vanilla program alive but idle, and waiting for the "Set Aside" or the "Save & Put Away" commands. And we need to provide a way to re-execute a vanilla program from the beginning.

If the "Set Aside" command is selected, the vanilla window will be shrunk to its document icon and left on the Desktop. If the document is reopened, the vanilla program will still be at this "end" stage; the program is not restarted. The user can examine the output of the previous run of the program by going through this process.

If the user wants to re-run the program, he can tear off the document stationary pad for the program to start from the beginning with a blank document. Or the user can put the "Set Aside" document icon back to a disk, and then reopen it to see the previous output. After the window is fully opened, the user can select the "Restart" command to start the program from the beginning with the output of the previous run still in the window. This allows the user to collect several execution results in one document. If the user starts with a blank document, the program will start execution automatically.

If the user selects the "Save & Put Away" command when the program is at the "end" stage, the vanilla window will be shrunk back to its icon. The output is saved in the document. After putting away the document, it can be reopened again to see the output of the previous run. If the user wants to run the program again, the "Restart" command has to be selected. Or, the user can just browse the document and put it away or set it aside again.

At any time when the program has not reached the end statement, the user can select the "Set Aside" or the "Save & Put Away" command, and the window will be closed to its document icon. If "Set Aside" is selected, the program continues execution until a read or the "end" statement. When the document is reopened, there may be more output than when it was set aside. If "Save & Put Away" is selected, the program will be terminated. When the document is reopened, the output of the previous run will be displayed, and the program will start from the beginning if the "Restart" command is selected.

The "Flush Input" command is used to delete all the input in the input buffer before the program reads the input.

The "Select All Text" will select the entire text panel.

The "Select All Graph" will select the entire graphic panel. This command will appear when the graphic panel exists.

3.1.3. Terminal Specifics - Setup

This menu provides the terminal specific commands.

The "Setup" command allows the Standard Terminal or specific terminal emulator to be reconfigured. When this menu is selected, a dialog box will show up with some check box for a user to select the appropriate features. The Standard Terminal will provide the following setup features:

Characters per-line - 80 or 132,

Wraparound - yes or no,

Tab spaces : 8 or other number.

Specific terminal emulator can override these to show its special or more setup features.

3.1.4. Execution - Restart, Resume

The "Restart" command will restart program execution after the user has reopened a document that has been put away.

The "Resume" command will continue program execution after the user has typed "Apple-Period" key combination to suspend program execution.

4. Additional support needed for the vanilla window

4.1. PASLIB

The PASLIB supports several logical devices. In addition to direct OUTPUT to the vanilla window, the '-CONSOLE' logical device and the '-KEYBOARD', the non-echoing input logical device, should also be directed to the window when a vanilla program uses the vanilla window unit. If a program is not using the vanilla window unit, all its WRITELNs are sent to the Alternate Screen. In any case, programs can open '-ALTCONSOLE' (RESET of this name) to send debugging WRITELNs to the Alternate Screen even if the programs are using the vanilla window unit. Programs can also open '-RS232A-x' (or RS232B:) to send WRITELNs to an external terminal. The KEYPRESS function needs to have additional support to allow one of these console destinations to be

The Pascal REWRITE procedure requires creation of a temporary file. With the vanilla window, there may be more than one program running at the same time. The current make and kill mechanism for temporary files causes the second REWRITE call to fail if the first REWRITE is in between the make and kill calls. We need an indivisible OS call to make a temporary file so simultaneous REWRITEs will not fail.

6.3. ToolKit and Stable and Extensible Mechanism need to be ready.

The vanilla window will use the ToolKit for its implementation. The ToolKit has to be ready before we can release this product. The vanilla window will make more programs depend on the ToolKit and the lower-level libraries. We need the Stable and Extensible Mechanism in place when we release this product to third party users. We need to be able to release enhanced versions of the ToolKit or the vanilla window in the future without causing programs to c

6.4. The new OS needs to be ready.

With the vanilla window, a vanilla program running in the Office System will take a much longer time to come up than in the Workshop today, since the program has to go through the initialization of each of the supporting libraries routines. There will be much more programs running in the Office System than today. The new OS has to be ready before we can release this product.

6.5. Schedule

The first internal release - September, 83 - Done
with functional interface in place but not the user interface (menus),
with text panel only
with VT100 emulator and Soroc Emulator

With this release, we can start porting the Workshop tools.

The second internal release - November, 83
with user interface in place, text panel only

The third internal release - December, 83
with graphic panel

The Alpha Release - January, 84

The Beta Release - February, 84

Product Release - March, 84