XDE TUTORIALS

XEROX

610E00250
November, 1985

Xerox Corporation
Office Systems Division
2100 Geng Road
Palo Alto, California 94303

1. Basics

2. Files

3. Text

4. Mail system

5. Printing

6. Booting

7. Compile-Bind-Run

8. The debugger: CoPilot

9. Tool building

Appendices:

LIST OF FIGURES

Welcome to the Xerox Development Environment! This document contains the text of the XDE tutorials, which are designed to give you "hands-on" experience with the XDE user interface. This a self-paced instructional book; you should try each of the things discussed in the tutorials on your own machine to make sure that you are familiar with them. This hardcopy should be used primarily as a reference.

The first thing that you will need to become friends with is your mouse, which is used to direct attention to a particular area of the screen. The standard mouse has two buttons. The left button is called "Point"; the right button is "Adjust." Pressing and immediately releasing a mouse button is called "clicking" the button. Pressing down both buttons simultaneously is called "Chording" the mouse.

Mouse movements are tracked on the screen by a small black arrow called the "cursor". Try using your mouse to point at various places on the screen. You can move the mouse in any direction, or pick it up and move it when you reach the edge of the mouse pad. When you are good at using the mouse, move the cursor over to the word Display! (roughly in the middle of the MailTool window) and click Point.

Getting started

As you can see, clicking Point over the word Display! allows you to read a new message. Display! is a command associated with the MailTool, which is the name of the tool used for reading tutorial messages. The XDE user interface is largely based on visual imagery: you can usually see the commands associated with a given tool without having to ask for them or memorize them. A word followed by an exclamation point, such as Display!, Delete!, or Undelete!, indicates that the word is a command. (This is true throughout the environment; it is not unique to the MailTool.)

Clicking Point over a command invokes that command. When you invoke a command, it is best to set just the tip of the cursor over the command; placing the entire cursor on top of a command may not activate it.

Also, if you move away from a command while the button is still down, the command will not be invoked when you release the button. Try this: press and hold in Point over Display!, and then move the cursor away from the command before releasing Point. The command will not be invoked.

Now invoke Display! correctly to view the next message.

Windows

You are reading this message in what is called the "text subwindow" of your mail "window". The XDE user interface is based on the concept of the window. Each XDE "tool", or applications program, communicates with you through one or more windows. A window is just a partition of the screen in which text or graphics can be displayed. Generally, each tool owns a window, although a tool does not have to have a window and can have several windows. At the top of each window is a herald, or name stripe, that tells you the name of the tool that the window is associated with.

Windows are composed of subwindows, which are separated by horizontal black lines. The MailTool window has four subwindows. The uppermost subwindow is a message subwindow used for posting messages from the tool to the user. The second subwindow, which contains an ordered list of all the messages available to you, is called the Table of Contents. Figure 1.1 illustrates the Mail Tool.

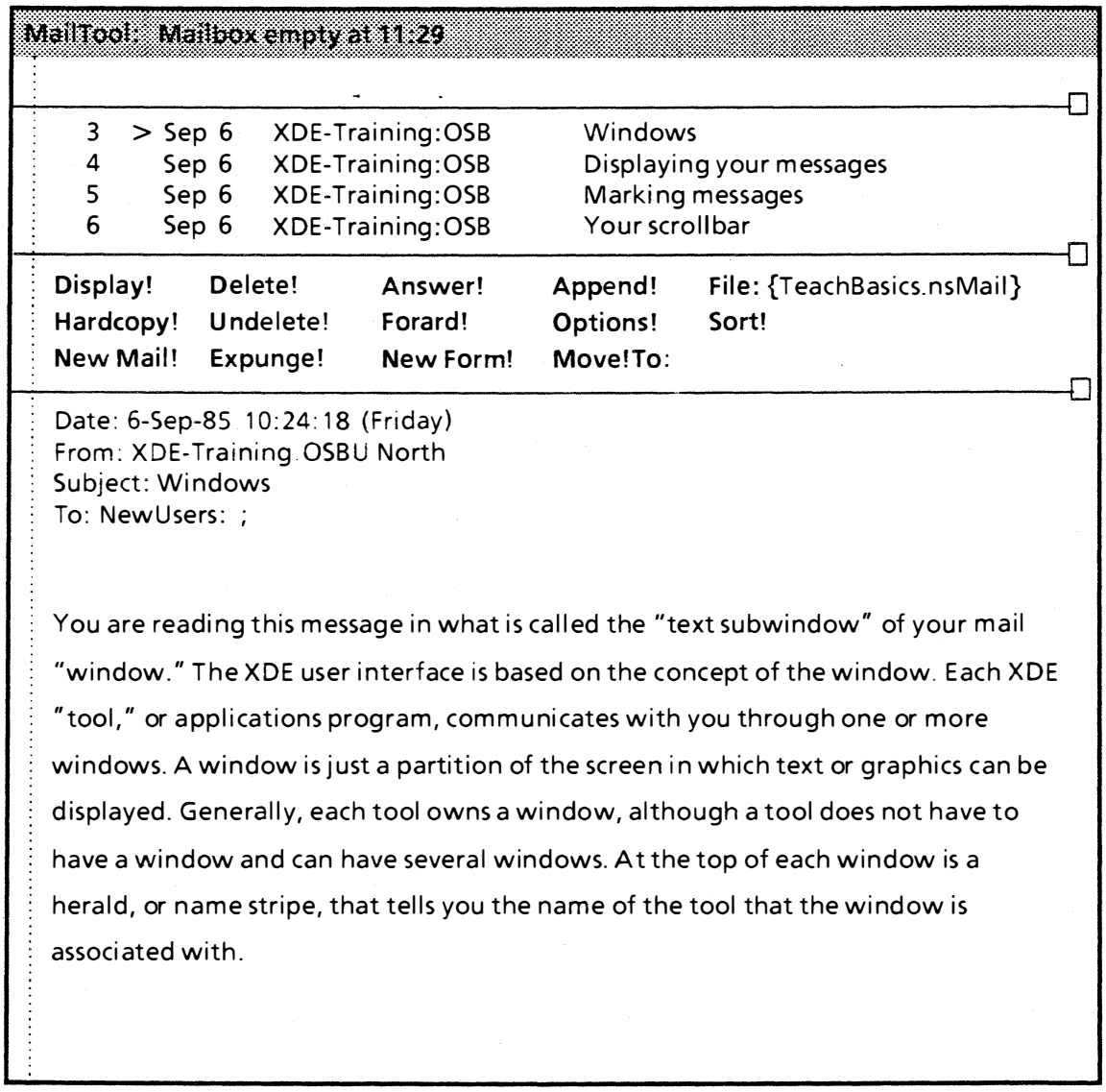


Figure 1.1: The MailTool

The third subwindow is a command subwindow. All the available commands specific to the MailTool are grouped together in this command subwindow. The items followed by ! are all commands; the items followed by : are called fields, and are used for collecting arguments to commands. You will learn more about fields later.

The fourth subwindow is a text subwindow, which is used for displaying the messages.

Displaying your messages

Display! allows you to read through a group of messages in the order in which they are listed in the Table of Contents. Thus, you do not have to explicitly specify an argument for Display!; the default argument is the next message in the mail file.

However, if you would like to read your messages in some other order (for example, if you want to read an earlier message again), you can explicitly specify any message in the file as the argument to Display!. To do this, just click Point over any character in the message title (in the Table of Contents subwindow). (Notice that the character becomes highlighted; that is, black characters become white or vice-versa.) When you have explicitly selected a message, invoking Display! will cause that message to be displayed in the bottom subwindow, regardless of whether or not you have read the messages that precede it.

Notice that there is an asterisk in front of the Table of Contents entry for each unread message When you read your mail in order, the title of the message that you are currently reading is moved to the top of your Table of Contents. There is also a small arrowhead in the Table of Contents window that points to the title of the message currently being displayed. Find the title of this message in the Table of Contents.

Try explicitly selecting the title of the next message, and then invoke Display!

Marking messages

You can mark specific messages in the Table of Contents if you like. Try selecting the blank space at the far left of a Table of Contents entry (there are three spaces to the left of the number; select the leftmost.) Once you have selected the space, type any letter, and it will appear there. Thus, for example, you can type an ! if you have a message that you are particularly interested in, or an I by messages that you thought were incomplete, or any other marking system you like. Markings can consist of only one letter. Figure 1.2 on the next page contains some messages that have been marked with an exclamation point.

To remove a marking, just select the letter and type a space.

The scrollbar

When you view text in a window, you may actually be looking at only a small portion of the available information. A window

can be thought of as an opening through which you can view a potentially infinite scroll of text; the amount of text that you see at one time is limited by the size of your window, and not by the amount of material in the text. To view text not currently visible, you can use a window feature called the scrollbar to "scroll" the text in a window up or down.

The scrollbar for a window is a narrow transparent rectangle found at the far left of the window. Move your cursor into the scrollbar for the Table of Contents subwindow of the MailTool window. (Each subwindow is scrolled separately.) When the cursor is in the correct position for scrolling, it will change into a double-headed arrow, as illustrated in Figure 1.2.

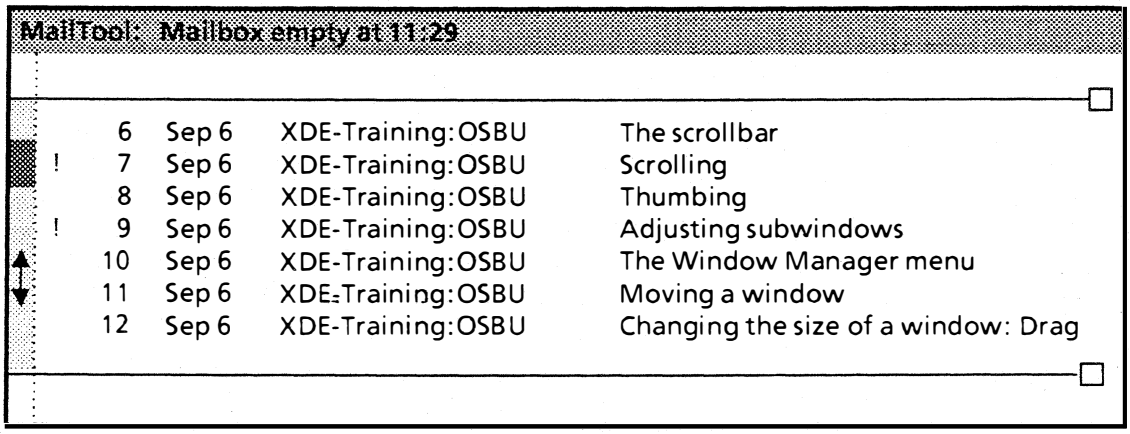


Figure 1.2: The Table of Contents with scrollbar

Try moving your cursor into the scrollbar. When the cursor is in the scrollbar, notice that the scrollbar has a dark grey region and a light grey region. These grey regions, combined, represent the entire length of the scroll behind the window; the dark grey region represents the percentage currently visible through the window. Thus, a dark small grey region means that you are viewing a small part of the file, and a large dark grey region means that you are viewing most of the file.

Scrolling

Mouse buttons direct the scrolling operation. The cursor changes when one of the buttons is pressed in the scrollbar: Point scrolls the document up (the double-headed arrow changes to point up) and Adjust scrolls it down (the double-headed arrow changes to point down). You can think of scrolling as moving the file that is behind the window so that you can see a different part of the file. The window itself remains the same; you are effectively just putting a different part of the file "in" the window.

Now try scrolling: position your cursor in the scrollbar for the Table of Contents beside message 10 and click Point. Notice that this line is now at the top of the window. Depending on how your machine is set up, you may be able to obtain continuous scrolling by holding down Point in the scrollbar region (rather than just clicking it.)

You can use Adjust to scroll back down in the file to view previous text. To practice this, put your cursor in the scrollbar somewhere in the middle of the Table of Contents and click Adjust. Message 10, or the message that was at the top of your Table of Contents before you clicked Adjust, will now be located at the position of the cursor. Practice scrolling the Table of Contents up and down.

When you are experimenting with scrolling, notice that the position of the dark grey region changes as you scroll the window up and down. This signifies that you are viewing a different part of the file; the dark grey region always shows the portion and position of the text that you are viewing with respect to the entire text.

You may have to scroll many of the later messages in this file in order to read the end of the message; if you are ever not sure whether or not there is more of a message, you should scroll up to check.

Thumbing

The scrollbar can also be used to "thumb" a file. Thumbing is analogous to opening a book by placing your thumb at the approximate position of the section you want to start reading, and pulling the book open at that point (as you might do with a dictionary).

To thumb a file, press Chord (both buttons simultaneously) while the cursor is in the scrollbar. The cursor should change to a left-pointing arrow that can be moved up or down in the scollbar. If the cursor is in the middle of the scrollbar, releasing Chord will move you to the middle of the file; if the cursor is at the top end of the scrollbar, releasing Chord will scroll you back to the beginning of the file. With practice, you will be able to reach the particular part of the file that you want to look at without having to scroll through the entire file. This is especially useful with large files.

Note: to release a Chord, always release the left button first. You don't have to release the buttons simultaneously, and you won't get the result you expect if you release the right button first.

Practice thumbing the Table of Contents. Notice that releasing Chord when the cursor is not in the scrollbar aborts the operation.

Adjusting subwindows

In addition to scrolling or thumbing the text in a file, you may also wish to change the shape, size or position of the windows on your screen. The next series of messages will discuss the available commands for manipulating your windows.

Look at the right edge of the lines that divide the window into subwindows, and note the small boxes. These boxes are used to guide the dividing lines when you move them up or down within the window. To see how they work, choose one of the boundary lines on your screen and place the point of the cursor on the box that lies on it, then press and hold down Point.

Move the cursor up and down, and note how the boundary moves with the cursor. Now move the line to the position you wish the boundary to be in and release the button. The line will be moved to that point. This feature is available for every subwindow in the Xerox Development Environment.

The window manager menu

In addition to adjusting subwindow divisions, you may also wish to make an entire window larger or smaller, or move it to a different location altogether. The Xerox Development Environment has a feature called the Window Manager menu, available for every window, which is used to control the size and position of your windows.

During the exercises on window manipulation, you should practice on an Empty Window, and not your MailTool window.

You can obtain the stack of possible "menus" for any window by placing the cursor anywhere in that window and pressing Chord. Try this now in the Empty Window. Continue to hold down Chord. You should see a stack of menus, with one menu fully displayed at the top of the stack, as illustrated in Figure 1.3. The title of the menu that is at the top of the stack should be highlighted; that is, it should appear as white letters on a black background instead of black letters on a white background. Menus are a convenient way of letting you see the available commands when you want to use them, while still conserving "real estate" on the screen the rest of the time. (Note, however, that commands in menus are not designated with !.)

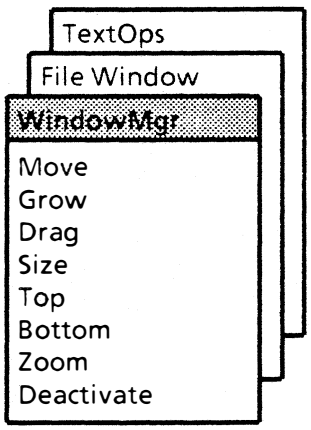


Figure 1.3: Menus

If the Window Manager menu ("Window Mgr") is not already at the top of the stack of menus, you will have to bring it to the top. To do this, point the cursor at "Window Mgr" (it will highlight), continue to hold down Adjust while you release Point and then click Point again. The Window Manager menu should now be fully displayed at the top of the stack. Bringing a menu to the top of the stack may be a bit difficult at first; you should practice until you are good at it.

Moving a window

The Move command is used to move a window about the display screen without changing its size.

Bring the Window Manager menu to the top of the stack, continue to hold down Chord, and select "Move" from the menu by moving the cursor on top of it. ("Move" will be highlighted when the cursor is in the correct position.) Release Chord (both buttons at once), and the cursor will change into the shape of a corner with an "M" in it.

Practice moving the cursor in and out of the various boundaries of the Empty Window, and watch the corner change shape to represent different corners of the window. Using Move is like picking up a piece of paper by one corner. Thus, the corner represented by the cursor is basically the argument to the Move command; it specifies the corner by which you are "picking up" the window. Figure 1.4 illustrates moving a window by the lower lefthand corner.

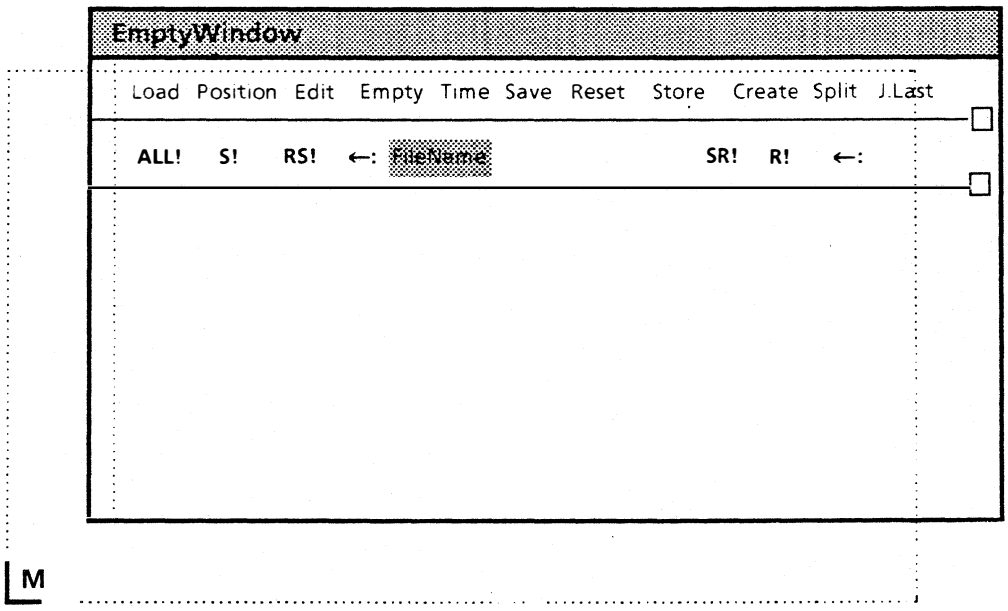


Figure 1.4: Moving a window

When you are ready to move the window, move the cursor to the desired position, and click Point. The corner represented by the cursor will be moved to that location.

Move allows you to move a window about the display area, but doesn't let you change its size or shape. Try moving the Empty Window around, using different corners as your anchor. Experiment until you are comfortable with this command.

Changing the size of a window: Drag

The Drag and Grow commands allow you to change the size of a window on the screen. Bring up the Window Manager menu by chording just as you did to invoke the Move command, but

this time select "Drag" instead. The cursor will change to look like an arrow that points to a line. The line represents a border of the window. The Drag command moves (drags) one border of the window either outward (to make the window bigger) or inward (to make the window smaller).

Move the cursor in and out of the window and notice how the cursor changes to represent the different borders of the window. When you click Point, the specified border of the window will stretch or shrink to the position of the cursor, and the rest of the window will remain the same. Practice moving around the borders of the Empty Window with the Drag command. Notice that you can use Drag to shrink the window as well as to enlarge it.

Changing the size of a window: Grow

Drag allows you to adjust the position of one window border at a time. The Grow command, on the other hand, allows you to adjust length and height simultaneously. Select Grow in the Window Manager menu. The cursor should look like a corner with a "G" in it. Move this corner in and out of the window and watch it change to the shape of the corner closest to where you exit the window, as it did with the Move command. Position the cursor and click Point.

Grow allows you to pull a corner of the window in any direction, enlarging or shrinking the window along its width and height. Experiment with this command in the Empty Window.

Top and bottom

In the XDE, you can overlap and stack windows, just as you can stack pieces of paper on a desk. If one window either partially or completely covers another window, you may wish to change the order of the stack (much as you would shuffle the stack of papers). The "Top" and "Bottom" commands in the Window Manager menu are used to control the position of a window in the stack: Top places a window on top of all others; Bottom places it beneath them.

To practice these commands, you must first have some overlapping windows. Move your windows around until you have a group of windows in a stack. (The Herald window, which is the wide rectangle at the top of your screen can be moved just like any other window. You can use it in your stack of windows if you like.)

Now try the Top and Bottom commands. Invoking these commands occasionally causes one or more windows to be completely obscured, and it is not uncommon to forget a window that is invisible. You may therefore wish to manipulate your windows so that a tiny portion of each window is showing. You can invoke the menu for any window as long as the visible portion of the window is at least the size of the cursor.

Size and zoom

The Size command on the Window Manager menu reduces a window to a "tiny" rectangle of fixed size. Tiny windows can appear anywhere on the screen. (You will learn how to control the position of a tiny window in a later message.)

Try invoking Size on the Empty Window, and notice that the name of the window remains visible. (Depending on how your machine is set up, the tiny window will probably appear somewhere near the top of your screen; you may have to use Top or Bottom to find the tiny window.)

When a window is tiny, you can call up a menu in the same way as when it is normal size; however, Move, Grow, and Drag do not work for tiny windows. Since the size of a tiny window is fixed, there is no way to use Drag or Grow; you will learn how to move a tiny window in a later lesson.

Invoking Size on a tiny window puts it back to its original size and position, and places it on top of any other window at that location. Invoke Size again on the Empty Window.

As its name implies, the Zoom command causes a window to increase in size dramatically, so that it takes up all the available room on the screen. Like Size, Zoom can be reversed by invoking it a second time. Invoke Zoom twice on the Empty Window and watch it zoom up and then back down. A "zoomed" window is just like any other; you can use the Window Manager commands to put it under other windows, or to change its size if you like.

Deactivating a window

The last command on the Window Manager menu is "Deactivate," which removes a window from the display and makes it inactive. Windows can be in one of two basic states: active or inactive. Active windows are those that are open on your screen and that you are currently able to read. An active window may be tiny; tiny windows are essentially active windows that have been temporarily moved out of the way. The contents of a window are not altered when it is reduced to the tiny state.

Deactivation, on the other hand, causes a window to lose any contents that you have typed into it. Deactivating a window destroys the tool window, but the tool itself is still available for future use.

Thus, deactivating a window only affects the information that you have typed into that window; it does not affect information that is stored in files. For example, if you deactivate an Empty Window, you will lose any text that you have typed into that window. However, if you deactivate the MailTool window, you will not lose any text; the table of contents and the associated messages are stored in files and are thus not affected by deactivation.

Deactivate the Empty Window. When a window is deactivated, the name of the associated tool is added to the Inactive menu. To bring up this menu, press Chord in the grey bit area (any

area not covered by a window) and select the Inactive menu from the list of available menus. This menu contains the names of all the tools that have been deactivated since the last time that you booted, as illustrated in Figure 1.5. Select Empty Window from the list of inactive tools, and the Empty Window will once again be active on your screen.

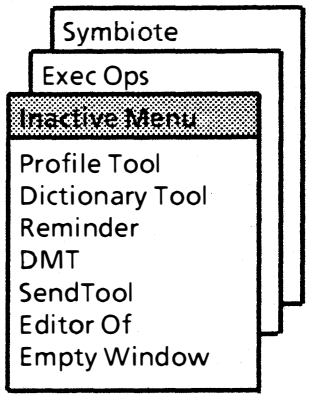


Figure 1.5: Inactive menu

Accelerators for Move, Grow, and Drag

Until now, you have been using the commands on the Window Manager menu to manipulate your windows. The next few messages discuss faster ways to invoke these commands. In the XDE, such shortcuts are called "accelerators."

Try moving the cursor to the herald of the Empty Window (the white-on-black "label" at the top of the window). As the cursor enters the herald, it changes to a bulls-eye shape, and sections of the herald video-invert. Move the cursor from the left side of the herald to the right side, and notice that the herald is divided into three sections

The left and right sections, which are equivalent, offer quick ways to invoke the Move, Grow, Drag, Top, and Bottom commands. Position the cursor in an outer section of the herald of the Empty Window. Press and hold down Adjust, and the cursor will assume the "M in a corner" shape When you release Adjust, the window will move to where the cursor is. In other words, to Move a window, you can either:

(i) Get the Window Manager menu and select Move,
Position the cursor and click Point;

OR

(ii) Move the cursor into an outside section of the herald
Press and hold down Adjust
Position the cursor and release.

You can execute Grow and Drag in a similar accelerated manner. Move the cursor back into an outside section of the

herald, and hold down Adjust. Continue to hold it down and click Point; the cursor will change into the "G" for the Grow command. Still holding down Adjust, click Point again, and the cursor will change into the shape of an arrow pointing at a border, ready for a Drag command. Click Point a few more times to cycle through these three commands, then practice using these accelerators to adjust your windows.

(Note: if you select Move, Grow, or Drag from the Window Manager menu, you can hold down Adjust and click Point to cycle through to reach any of the others. Thus, if you accidentally select the wrong command from the menu, you can easily enough reach another command.)

Accelerators for Top and Bottom

The last message discussed how to quickly invoke the Move, Grow, and Drag commands. Now you will learn how to invoke the Top and Bottom commands.

Move the cursor back into the left or right section of the herald of the Empty Window. If a window is already on top of all other windows on the screen, clicking Point in an outer section of its herald will invoke the Bottom command. If the window is underneath any other window, clicking Point will invoke the Top command. Each time that you click Point, you will invoke the inverse of whichever command you invoked last time. Try making a stack of windows and trying these accelerators.

Accelerators for Size and Zoom

Position the cursor in the center section of the Empty Window's herald. Click Adjust. You have just invoked the Size command. Clicking Point will invoke the Zoom command. Experiment a little with the accelerated Zoom and Size commands.

Invoke the Size command to shrink the Empty Window down to a tiny rectangle and try out the accelerated commands on the tiny window. Most of the commands function as they do on a full-sized window, with the exception of Move, Grow, and Drag. The Grow and Drag commands cannot be used to change the shape of a tiny window; the size of such a small window is fixed. The Move command works a little differently than it does on a full-sized window. Try it and see how it is different.

In a tiny window, the Move command can only be invoked with the accelerator (clicking Adjust in the right or left herald), and not with the Window Manager menu. Furthermore, no "M in a corner" appears when the command is invoked; instead, the cursor is tracked by the tiny window itself. Practice using the Move command to move the tiny Empty Window around.

Notice that the Herald Window, which is the long banner at the top of your screen, does not have a window herald when it is in the active state. You can still use the accelerated commands on this window, but you will have to move it from its position at the top of the screen. That is, the top edge of the window serves as its window herald, but you can't access that edge when the window is at the very top of your screen.

Each time you make a window tiny, it will return to the position from which it was sized. Thus, you can arrange the tiny windows on your screen any way that you like; the structure will not be lost unless you reboot. (In a later tutorial, we discuss how to specify the initial set up of your windows.)

Summary of accelerators

Here is a summary of the window manager accelerators available through the herald of a window:

The left and right sections of the herald are equivalent. Clicking Point in either of these two sections invokes Top and Bottom; holding down Adjust and clicking Point makes Move, Grow, and Drag available.

The center section of the herald is used for Size and Zoom; Point invokes Zoom and Adjust Sizes the window. If you have trouble remembering these accelerators, you might want to write them down until you become more familiar with them.

The accelerators are illustrated in Figure 1.6.

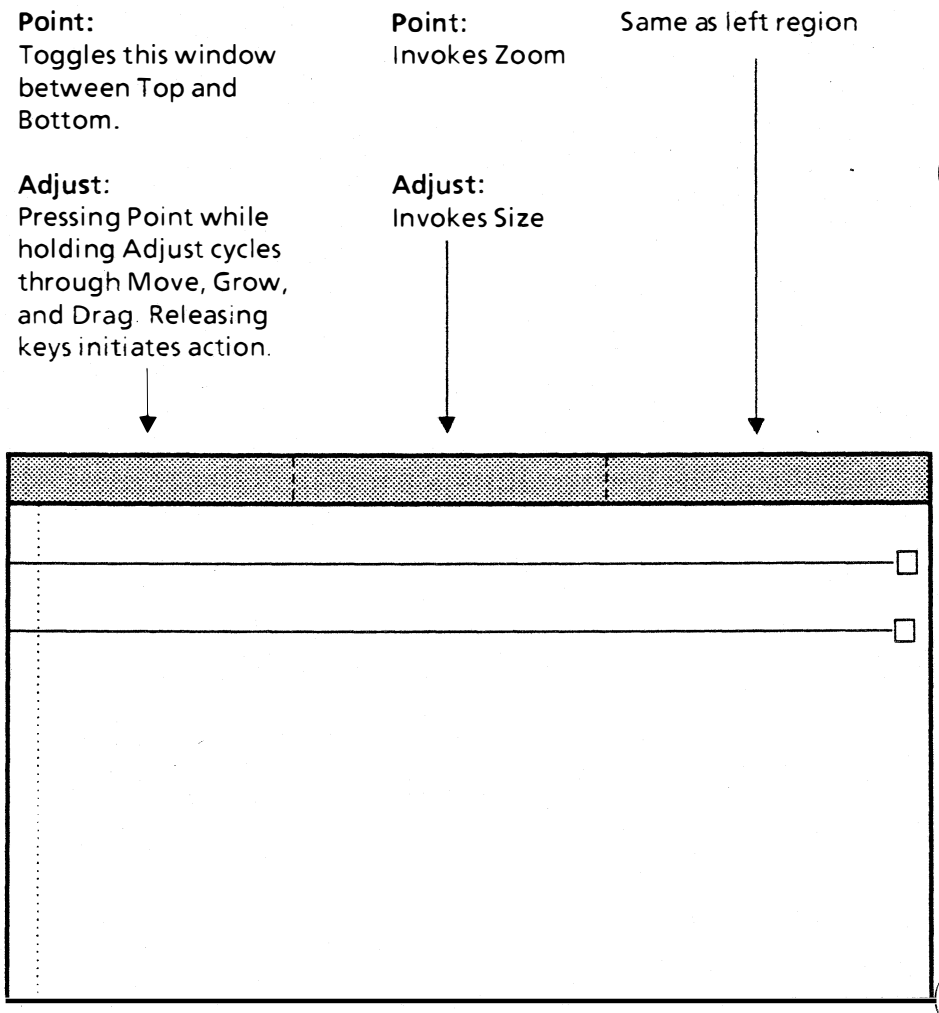


Figure 1.6: Window Manager accelerators

Entering text

The remaining information in this tutorial deals with the position of windows on your screen; how to enter text in a window, how to edit existing text, and how to control the font in which characters on your screen are displayed.

Click Point anywhere in your Empty Window, and notice that a binking caret appears. This is your type-in point.

Whenever you type, the text will appear at the type-in point. In the Xerox Development Environment, only one window can have an active type-in point at any given time. Thus, when you want to enter text in a window, you need to first click a mouse button over that window to activate its type-in point.

Set your type-in point in the Empty Window and start typing. Type several words, and then hit the backspace key (a backward arrow, above the carriage return). The last letter in your text will be deleted. SHIFT and backspace together will delete the last word that you typed Try it. These methods of editing are helpful if you realize that you have made a typing mistake while you are still typing. Now continue typing to the end of the line.

When you reach the end of the line, do not enter a carriage return. Continue typing, and notice that the system automatically breaks your text at the edge of a window and sends the overflow to the next line. Type several lines of text. Now change the shape of your window using the Grow or Drag command; the text will be reformatted to conform to the new shape of the window.

Selecting text

Pick any letter in the text you have just typed and point the cursor at it. (Remember to use just the tip of the cursor.) Click Point once, and the character will video-invert; you have "selected" that letter. Now choose another letter and click Point twice in rapid succession to select an entire word. Three clicks will select the whole paragraph, four clicks the complete textual entity, and five clicks will return you to a single character. Try cycling through this sequence.

Another method of selecting text is extension with Adjust. Select the first character of this sentence with Point, then select the last character with Adjust; you will have selected the entire sentence. With that sentence selected, move your cursor to the first sentence in this paragraph and click Adjust over a letter in the word "text". The selected text will be extended backward to the new position. Thus, you can extend a selection either forward or backward using the extension technique.

Extension of selected text operates in the same units as the original selection: if you select a character with Point, the extension will be by characters; if you initially select a word, the extension will be by words, and so on. Practice selecting text until you are comfortable with both methods; the next few lessons will show you how you can manipulate selections of text within and across windows.

Adding text

You can easily add text to any position in an existing editable document. For example, to add text to your Empty window, position the cursor in front of the character where you would like to insert your additions. Click Point to set your type-in point. Now start typing; the text will be inserted at the type-in point. Experiment with adding to the text in your Empty window, and notice that you can insert text at any position in a file. (Note that you always have to set a type-in point before you can enter text in a file; this is how you specify the location in the file where you want to put the new text.)

To set a type-in point at the beginning or end of a word, you may find it easiest to select the entire word with two clicks rather than trying to select the space that separates two words. Select a word by clicking twice at any letter in the first half of the word, and your type-in point will be at the beginning of that word. Select it by clicking near the end of the word, and your insertion point will be at the end of the word. Try positioning your type-in point using this method.

Note, however, that there are some windows that won't allow you to insert text into them. These windows are "read-only" windows, because you can read the information in them, but you can't change it. For example, make a selection in your Table of Contents subwindow and try to type in some new text The screen will blink at you; the system blinks the screen whenever you try to do something illegal.

Moving and copying text

The next few messages discuss copying or moving text from one position to another using the special keys DELETE, COPY, STUFF, PASTE, and MOVE. These keys are all located in the cluster on the left side of your keyboard.

If you find that your keyboard does not have some of the keys described in this tutorial, refer to the one page keyboard summary document. This shows the mapping between the keycap names and functions; you may have to refer to this document to find out which key or combination of keys performs a certain function. This document is commonly kept beneath the mouse pad. Find your copy of it, and familiarize yourself with it. If you do not have one, you can refer to the last page of the User Environment chapter of the XDE User's Guide

Set a type-in point anywhere in your Empty window and type in the sentences "Copying text is very easy. Moving text is also easy." Now suppose that you would like to move the word "very" to the second sentence. To do this, you first set a type-in point just before the "e" that starts the last word. (Click twice at the letter "e" to select the word and position the type-in in front of the word.) This is the location to which you are going to move the text. Now hold down the MOVE key and select the word "very". Release the MOVE key.

Your sentences probably now read "Copying text is easy Moving text is also veryeasy." When you move or copy text, you need to be aware of whether or not you are moving the spaces that separate words as well as the letters themselves. Selecting

a word with multiple clicking does not also select the spaces that precede and follow it. However, you can use the extension technique to include the surrounding spaces.

Now try copying text from one place to another. This command works in the same way as the MOVE command, except that it preserves the original text in addition to copying it to a new location. To copy text, first set a type-in point where you would like the new text to appear. Then hold down COPY, select the text to be copied, and release COPY. Try COPYing some text, and try to make sure that the spacing comes out right.

Stuffing text

You can also use the STUFF key to copy text from one window into another or within a single window. To use STUFF, first make sure that you have a type-in point set in your Empty window. Now select some text from this message, move your cursor back into the Empty window and click Adjust. Clicking Adjust over a window resets the type-in point to the place where it was last set in that window, but does not change the current selection. Had you tried to use Point instead of Adjust to re-establish the type-in point in the Empty window, the current selection would no longer be the sentence above; it would be a character in your empty window. Now press STUFF and the text will be copied to the new location.

Note that if you are trying to use STUFF to copy text within a single window, you will have to use PROPS-Point instead of Adjust. (If you try to use Adjust, you will just extend the selection.) Try stuffing text within a window: select some text in your Empty window, set another type-in point in the same window by pressing the PROPS key and clicking Point, and then press the STUFF key.

Experiment with MOVE, COPY and STUFF until you are comfortable with them

Deleting text

To delete text from a window, select the text and press the DELETE key. Again, you will need to pay attention to whether you also select the spaces and the punctuation. For example, when deleting a sentence, you will want to also delete the spaces either preceding it or following it to avoid having extra spaces separating the remaining sentences.

Text that is deleted from the screen is not immediately destroyed. Instead, it is held in storage (called the "trash bin") until other text is deleted. The advantage of this is that you always have a chance to change your mind about the last section of text that you have deleted. The trash bin can hold a large amount of text, but it is reset each time that you press the DELETE key. For example, if you have DELETEd an entire file, the contents of the file will all be stored in the trash bin. However, if you then DELETE an extra space, your existing trash bin (the contents of the file) will be removed and replaced with only a space character. Text deleted with BS or BW is not inserted in the trash bin.

If you decide that you would like to re-insert the text that is in the trash bin, press PASTE and the text will be inserted at your type-in point. (This is sometimes called "cutting and pasting".)

DELETE a piece of text, type a few words if you wish, then press and release the PASTE key. You will have moved the previously deleted text to a new place. (Note that deleting text also sets the type-in point, so you can just insert new text immediately after deleting old text.) Set a type-in point in another place and press PASTE again. The text will be pasted at that point as well.

You can PASTE the text in the trash bin as many times as you like, but that the bin only holds one segment of deleted text at a time, and any previously deleted text is really gone.

Local files

When you enter text into an Empty Window, such text is not automatically stored in a file. To avoid losing the text when you deactivate the window or restart your system, you need to store it in a local file. To store text in a file, first type or copy the contents into an Empty Window. The subject of the text could be your comments on this tutorial, the names of the people that you have met today, or anything else that you like.

When you are finished entering text, select the name that you would like the file to have (if it is not in the text, type it in the blank space following RS! in the lower subwindow. To do this, first select the colon (←:) with Point to set a type-in point.) A file name is limited to 100 characters, and cannot contain any spaces, or question marks. Plus sign, minus sign, period, and dollar sign are the only special characters that are acceptable for a file name. (The screen will flash if you attempt to name a file with an illegal name.) Your Empty Window should look something like the one in Figure 1 7

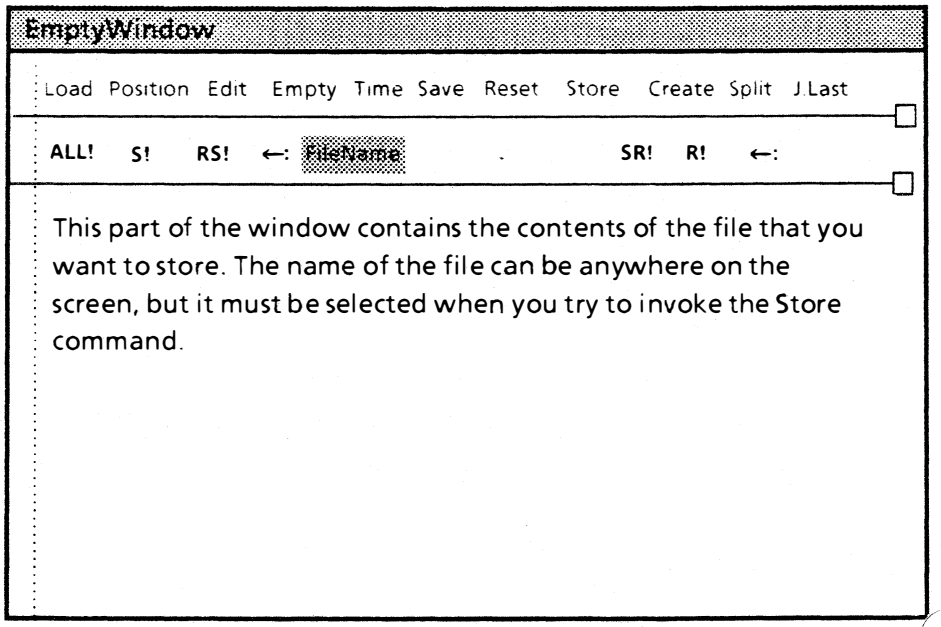


Figure 1.7: Storing a file

When you have SELECTED the name that you wish your file to have, you need to invoke the "Store" command. To do this, Chord anywhere in the Empty window, bring the File Window menu to the top of the stack, and select Store. Just typing the name on the screen is not enough; you must have the name selected (highlighted) when you invoke the Store command. If you don't have a selection, there is no way for Store to know what you want the name of the file to be.

When you have invoked Store, a message will appear in the herald that gives the name of the file, followed by a parenthetical expression telling you whether it is an old file name or a new file name. If it is an old file name, confirming the command will cause the current file to be written on the old file, and you will lose the old file When you store a file, you will always be told whether it is a new or an old file name, so that you will not inadvertently rewrite an existing file.

The cursor now looks like a mouse. This image is asking you for confirmation of your command. To confirm the command, click Point. (To abort, click Adjust.) When you have confirmed the command, the new name of the file will appear in the herald of the window, and the file will be stored on your local disk.

(Note that you will not be able to set a type-in point when you have stored a file in a window. A later message will discuss how to edit an existing file.)

EM Symbiote

Look at the top of your Empty window, below the window herald As shown in Figure 1.8, there are two subwindows at the top of it; the uppermost of these is called the Editable Menu (EM) Symbiote.

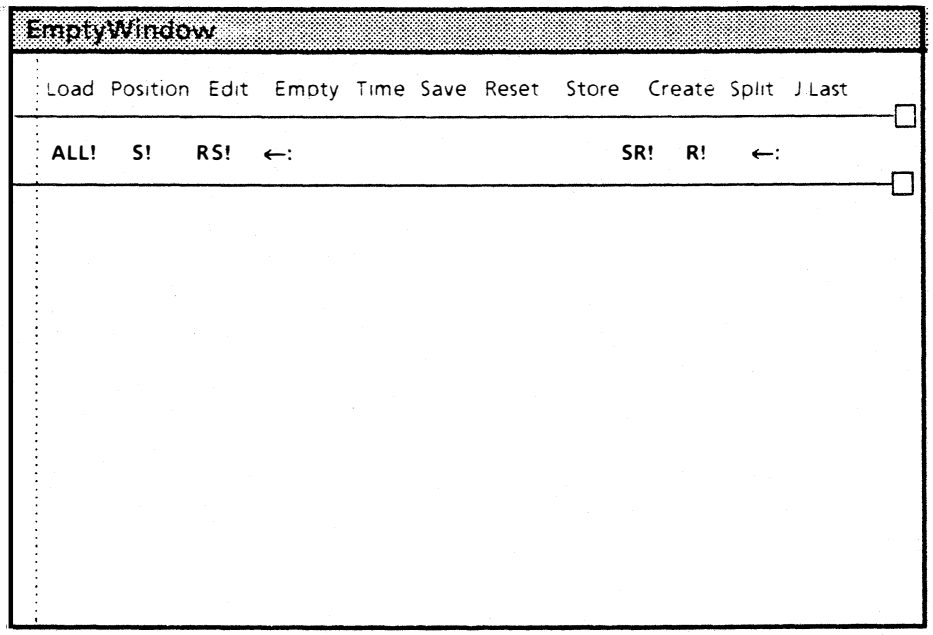


Figure 1.8: Empty window with EM Symbiote

The EM Symbiote can be edited to contain any collection of commands. The word "symbiote" signifies that the subwindow

functions independently of the window itself; you can attach or detach a symbiote without changing the properties of the window itself. Symbiotes are added to the window to make some of the more frequently used commands easily accessible without having to use pop-up menus.

For example, your symbiote should contain the word Store. This is the same Store command that you just invoked from the File Window menu. Thus, the accelerated way to store a file is to select the name of the file and click Point over the word Store in the symbiote menu.

You can edit the list of available commands. However, when editing the list of commands, you will not be able to use Point to select the text. The reason for this restriction is simple; clicking Point at a word in this symbiote will invoke the command.

To set a type-in point in the EM, you will have to use PROPS-Point, rather than just Point. For example, to delete a command, hold down the PROPS key and click Point following the command to be deleted. Now backspace over it. Enter a new command, if you like, or retype the old one. For now, don't worry about it if you don't recognize all of the commands listed in your EM symbiote; you will learn about most of them later. The purpose of this lesson is simply to make you aware of the EM symbiote as an alternative to your pop-up File Window menu.

You will also learn later how to permanently set the list of commands that appear in your EM symbiote.

Loading a file

Deactivating the window in which a file was stored does not destroy the contents of a file, although it does remove the text from the File window. Deactivate your File window (the one with a file loaded in), and then select EmptyWindow from the Inactive menu to bring it back on your screen. It will once again be an Empty window.

To load your file into the Empty window so that you can again view its contents, type the name of the file, select it, and invoke "Load" from the File Window menu or the EM symbiote. When you have loaded a file, the name in the herald will change from "Empty Window" to the name of the file that you have loaded in.

Now invoke the Create command, either from the EM symbiote or the File Window menu, to get another Empty window on your screen. You will be asked to confirm the command with Point. (The location of the cursor when you click Point will be the location of the new Empty window.)

You can now use this window to try the accelerated way of loading a file. (You can load the same file into two different windows if you like) Type the filename as the first line in the window, and then press the DOIT key. (The DOIT key is labelled MARGINS on your keyboard)

Now invoke Destroy, either from the EM symbiote or from the File Window menu, to get rid of one of your File windows.

Editing a file

To make changes in a file, you must invoke the Edit command. Try it. Notice that the herald of the window changes to indicate that you are editing the file. You can now edit the file in any way that you like; however, if you attempt to edit a file without invoking the Edit command, the screen will flash and your keystrokes will be ignored.

When you have finished editing a file, you will have to invoke Save to save your changes. Changes that are made to a file while it is being edited are not actually made to the file until you have invoked this command; if you deactivate a window that is being edited you will lose all of your edits.

Be careful to distinguish between the Save command and the Store command. Save is used to save the contents of a File window under the name that appears in the herald of that window; it is used when you are changing an existing file and wish to save the new version.

Invoking Store in a File window asks the window to take any selected text as its argument and store the contents of the File window under the selected name; this command can be used to initially name the contents of an Empty window or to change the name of an existing file.

DMT

You should now be familiar with the basics of window and text manipulation in the Xerox Development Environment. If you had difficulty with any of the things covered in this tutorial, you should go back and practice them until you are really comfortable.

The last thing that you should know is how to leave your screen when you leave your office for a while. When you are ready to go home for the night, or when you know that you will be away from your machine for a long period of time, you should "cover" your screen with the DMT tool. (The reason for this name is purely historical; the letters do not relate to its current function.) DMT allows you to turn your screen black when you are not working so that the phosphor on your screen will not wear out.

You can return from DMT at any time by pressing the STOP key.

DMT may be listed in your Inactive menu. If you select it from this menu, your screen will go entirely black, except for a small square that flashes the day and time. Another way to invoke DMT is run the program from the Executive by moving your cursor to the Executive window and type DMT, followed by a carriage return.

To deactivate DMT, you can press the STOP key or invoke the Deactivate command in the Window Manager menu. (You can manipulate the DMT pattern just as you can any other window.) Your screen will reappear just as it was before you

invoked DMT. If you are through for the day, you can invoke DMT now and go home. If you are not through, you should try DMT anyway, just so that you are familiar with it.

There are many alternatives to DMT available, for those who want a more interesting pattern to display on their screen. Most of these alternatives are classed as unsupported utilties, which means that they are not part of the standard released system. These tutorials in general discuss only the released tools. However, once you are familiar with the basics of the system, you should take a look at the unsupported utilities that are available.

As examples of such programs, you have BrushDMT.bcd, Poly.bcd, KineticFractal.bcd and SpaceOut.bcd on your machine. All of these programs are DMT alternatives; try them out if you like.

Reaching the next tutorial

When you are ready to go on to another tutorial, Chord over the word File: in the command subwindow of the MailTool. A menu will appear with a list of all possible tutorial or mail files. When you select the next tutorial, called TeachFiles.nsmail, from this list, this mail file will be put away, and TeachFiles.nsmail will appear in your MailTool window, ready for you to learn about the file systems of the Xerox Development Environment.

This tutorial discusses how to store, list, delete, retrieve, and organize files, both on your local disk and on remote file servers.

A remote file server is a large capacity file storage device that is shared among many users. Thus, you will use remote file servers as a place to store files that others will need access to, and as a place from which to obtain copies of files created by others. Because remote servers have larger capacity and better reliability than your local disk, you should also back up important local files on your remote server.

The early lessons in this module discuss local files; the later lessons discuss remote files. You should create a practice local file to use for this tutorial, if you don't already have one around.

Naming conventions

By convention, a file name has two parts, the main name and the extension, which are separated by a period. For example, the file name "Introduction.doc" has main name "Introduction" and extension "doc". A file name can contain more than one period.

Generally, the main name identifies the file and the extension identifies the type of the file. Although you are free to use any extension that you like when naming files, there are several common extensions that are used throughout the Xerox Development Environment. Standard extensions include:

.bcd	--Mesa object file
.cm	--command file
.doc	--documentation file
.ip	--interpress format file
.mesa	--Mesa language source file
.nsmail	--mail file
.txt	--text file

The XDE is generally insensitive to case in file names. Thus, ALPHA.mesa, alPHa.mESa, and alphA.mesa all refer to the same file. However, capitalization is often used as visual punctuation, especially when a file name consists of more than one word, as in TripReport or MasterList. Since most tools display file names the way they are defined, regardless of how you refer to them, you need only be careful with capitalization when naming a file for the first time.

The Executive window

There are two ways to list the files that are stored on your local disk: with the File Tool and with the Executive. This lesson covers the basics of using the Executive. Find your Executive window; if it is not active, look through your tiny windows and on the inactive list. The Executive is illustrated in Figure 2.1.

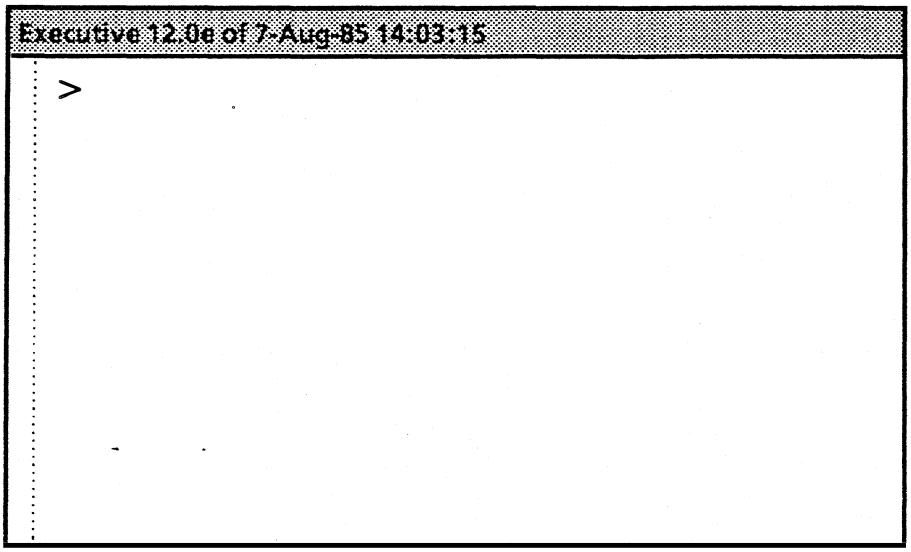


Figure 2.1: The Executive window

Bring up your Executive window. ">" is the symbol which the Executive uses to signify that it is ready to accept your input; this symbol (called a command prompt) should appear in the window.

The Executive has a standard teletype interface, which means that many of the editing functions commonly available in the Xerox Development Environment do not work in this window For example, set a type-in point and type something after the command prompt (>), but do not type a carriage return. Now select the word and attempt to delete it. XXX will appear to indicate that the command has been aborted, but the letters will not be erased from the screen.

In the Executive window, the type-in point is always at the end of the existing text in the window Thus, you cannot use any editing technique that requires text selection. However, you can use backspace (BS) and backword (BW) to change the end of a line, and you can copy text into the Executive window from other places on the screen. Try using BS and BW, but try not to enter a carriage return: you might do something unexpected.

In the Executive window, pressing DELETE at any time prior to pressing RETURN will cancel the entire command line and return a blank command line.

Using the Executive to list files

Use Delete to clear out any text you may have entered into the Executive, and type a question mark. This will list the available commands and files, as shown in Figure 2.2. The list will probably be a long one: when it has finished, scroll up (if necessary) so that you can see the beginning of the list.

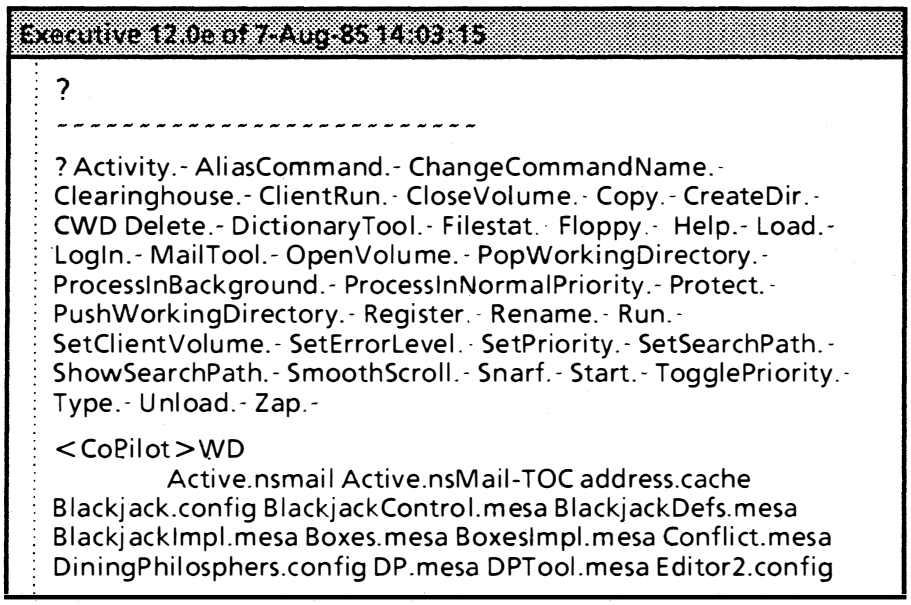


Figure 2.2: The Executive with a list of files

The beginning of the list is a group of words followed by ".~". The "dot tilde" extension is a convention that is used to designate a command name. You don't have to type the extension when you invoke the command; the purpose of the extension is to enable you to distinguish command names from file names. For now, you don't need to know the meanings of all of the commands; you just need to realize that they are commands. Many of them will be discussed later in this tutorial.

The rest of the list is file names. Each group of files is headed by a name of the format <CoPilot> or <CoPilot>tools> or the like. These headings represent local directories that are on the search path. You may only have one group, or you may have several. The next few lessons discuss what a search path is, and how to create and manipulate local directories.

Local directories

<CoPilot> is the name of a logical volume, which you can think of as a partition of your hard disk. Thus, the list of files that you see when you list the files "on your local disk" is in fact the list of files stored on the logical volume CoPilot. There may be as many as ten logical volumes on a single physical volume, and the files stored on each are entirely separate. Thus, CoPilot is the name of the root directory for all of the files stored on the CoPilot logical volume.

The fully-qualified name of a file describes the path from the machine on which the file is located down to the file itself. The

general form for a file name is [MachineName]<directory>subdirectories>name.extension.

For example, the simple file name user.cm could be more completely named as <CoPilot>user.cm (the file named user.cm on the root directory CoPilot. (The fully-qualified name of this file would include the name of your machine; however, you do not need to include the name of your machine when you are referring to a local file, just as you do not need to include the area code when making a local phone call.)

You can also sub-divide your local disk into a group of local directories, which are basically just file groupings. For example, you could create a local directory called "mail", one called "programs", and one called "text". You can create as many local directories as you like. You can also subdivide directories: you might want to have the local directory "programs" further divided into individual project names, for example.

The complete name of a file thus includes any local directories when applicable. For example, the file <CoPilot>mail>schedule.nsmail would be the complete name of a file stored on the local directory called "mail" on the CoPilot volume.

The search path

Since the complete name of a file gives the path to finding that file, you might think that you need to give the complete path name of a file each time you refer to it. Fortunately, however, this is not the case. For example, when referring to a local file name you never have to include the machine name: there is no question as to the machine being referred to; it is defaulted to your machine. (We discuss how to specify a machine name other than your own later in this tutorial.)

You can also specify a "search path," which is just a list of local directories, and the order in which they are to be searched. Creating and manipulating search paths is done with a special tool called the Search Path tool, illustrated in Figure 2.3.

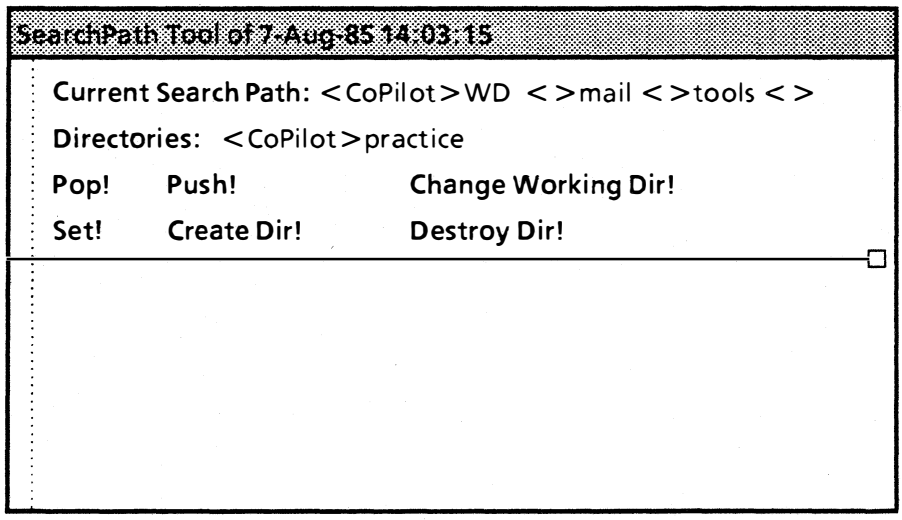


Figure 2.3: Search Path Tool window

The first line in the Search Path tool shows the current search path. When you give the simple name of a file, the system will start looking for it in the first subdirectory in the list, and will continue the search until it finds it or until the search path is exhausted. **Note:** < > is just shorthand for >CoPilot.

The search path does not necessarily contain all of the local directories in existence. If a directory is not listed on the search path, you effectively cannot see the files contained in that directory unless you refer to them by their fully-qualified name. Thus, if the directory >yes is on the search path, and the directory >no is not; then you can refer to the file >yes>fred as just fred, but you would have to refer to the file >no>sam as >no>sam in order to be able to see it.

Any new files that you create will automatically be stored in the directory at the head of the search path unless you specifically designate another directory.

To see all the directories on your local disk, Chord anywhere in the Search Path window and bring the menu labelled All Directories to the top of the stack, as illustrated in Figure 2.4. This lists all directories on the disk, regardless of whether or not they are on the search path.

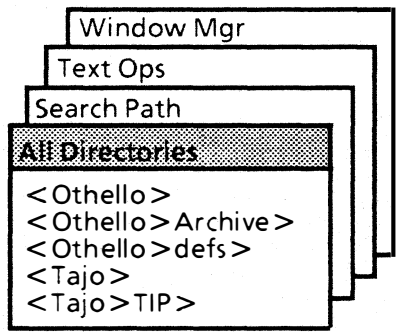


Figure 2.4: All Directories menu

Adding new directories to the search path

You can create new local directories with the Search Path tool. Type in "<Copilot>practice" in the field labelled Directories: (Select the colon following the word, and then type in Practice, or copy it in from this message.) Now invoke the Create Dir! command. You should see feedback in the bottom subwindow of the Search Path tool window telling you that the Practice directory has been created. Now Chord again, and notice that the name of your new directory has been added to the All Directories menu.

To add an existing subdirectory to the search path, just select the name of the directory from the All Directories menu. Add the directory Practice to the search path.

If you had entered just "practice" in the form field, rather than "<Copilot>practice", the new directory would have been created as a subdirectory of the directory currently on the top of the search path, rather than as a subdirectory of the root directory itself. The easiest way to make sense out of this is to

try it: enter "practice2" into the Directory: field and invoke CreateDir!. Bring up the All Directories menu, and notice that your new directory is <CoPilot>Practice>Practice2" rather than "CoPilot>Practice2."

You can also add a directory to the search path by typing its name in the Directory: field and invoking Push!

Removing directories from the search path

You can remove directories from the search path in a similar manner. Just bring up the Search Path menu, and select the name of the directory that you wish to remove from the search path. Remove the Practice directory, and use the Destroy Dir! command to destroy the directory. As a protective measure, the system will not allow you to destroy a directory that contains any files, or a directory that is on the search path. Thus, if you have created a <CoPilot>Practice>Practice2 directory, you will have to delete the Practice2 directory before you can delete the Practice directory.

If you want to completely change the search path, you can type in your desired search path in the Directories: field and then invoke Set!

You should consult the Search Path tool chapter of your XDE User's Guide if you would like more information on the operation of the Search Path tool.

Most of the Search Path tool commands are also available from the Executive window. Check your list of Executive commands, and you should see CreateDir, PopWorkingDirectory, PushWorkingDirectory, SetSearchPath, and ShowSearchPath. You can manipulate your search path from either the Search Path Tool or the Executive or both, depending on what is most convenient for you.

For more information on the Executive commands, consult the Executive chapter of your XDE User's Guide.

Renaming files

The Executive Rename command is used to rename local files. Rename can be used to rename the simple name of a file, as in renaming a file called Conference.temp to Conference.txt. Rename can also be used to rename the path name associated with a file; that is, to change the local directory that it is found on.

As an example of this, assume that you have a local directory named "draft", and there is a file on that local directory named Status.txt. When you have finished your status report, you decide to move it from the directory "draft" to the main directory CoPilot. To do this, you would enter the following line in your Executive window:

Rename <CoPilot>Status.txt ← <CoPilot>Draft>Status.txt

As you can see, the syntax of rename requires the destination file name, followed by a ←, followed by the source file name The ← must be surrounded by spaces. (The ← character is not

on your keyboard; you get one by typing the key labelled with a left quotation mark and a left apostrophe. If you can't find it, you will have to check your keyboard mapping chart.)

You can also abbreviate the Rename command if you like; ren will work just as well. Capitalization is unimportant: coPilot is as good as CoPilot or copilot. However, the new file name will be capitalized exactly as you type it; in fact, you can use Rename to change the capitalization of a filename.

Experiment with Rename a bit. If you have trouble, you can always refer to the Executive chapter of your XDE User's Guide.

Deleting files

The Executive Delete command is used to delete files from the local disk. Delete really does delete a file: once you have used delete, there is not usually any way to recover your file.

You are going to use scratch files to experiment with the Delete command. When you edit a source file, the edits are not immediately made to the actual file. Instead, the system creates a temporary copy of the original file, and the edits are made to that file. When you invoke Save, the edited version becomes the "real" file, and the unedited version is saved as a backup. These backup versions are labelled with a single $ following the filename. The $$ versions are also backup files; they contain all characters entered into the file during the actual editing process. Thus, a $ file is a copy of the complete, unedited file, and a $$ file is a log of the changes made during the last editing session.

To find out if you have any existing scratch files, type *$? in the Executive window. The * character is a wildcard, used to match any or all characters. Thus, this string asks the Executive to list all files that end in the character $ If you don't have a scratch file, generate one by editing a source file. (You should have a practice file around by now.)

Now try deleting the scratch file. Since this is practice, it doesn't matter whether you use the $ version or the $$ version. Generally speaking, however, the $$,versions are less useful to keep around than $ versions. You should also note that scratch files are never deleted automatically, and that you should occasionally clean up your disk by deleting scratch files that you don't need.

The file tool

This tutorial has covered most of the file manipulation commands available from the Executive window. The rest of the lessons in this tutorial will use the File Tool. Therefore, you should put your Executive tool away for a while (either Deactivate it or Size it), and bring up your File Tool window, as illustrated in Figure 2.5

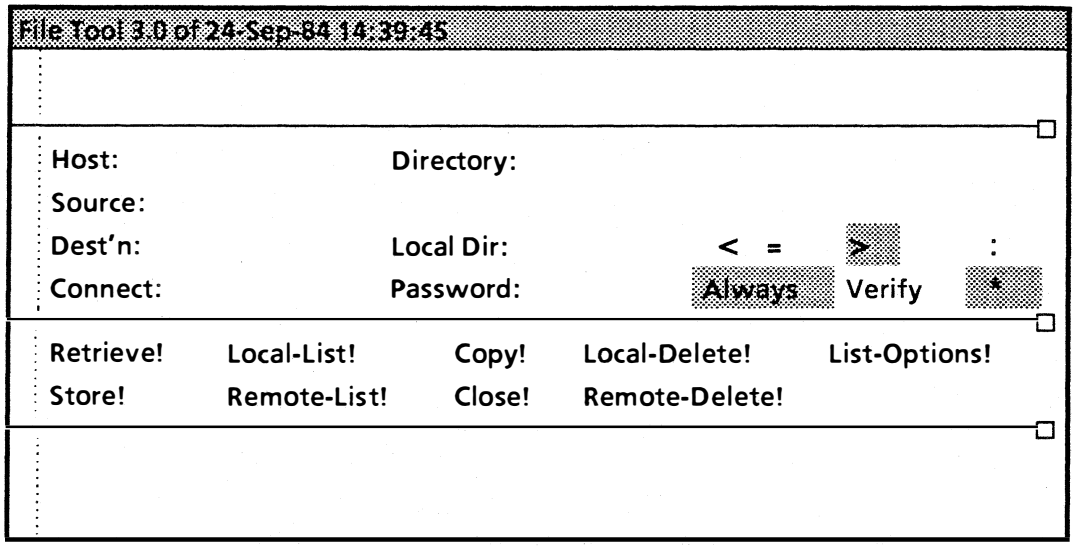


Figure 2.5: File Tool window

The File Tool has four subwindows. The uppermost subwindow is a message subwindow; the second subwindow is a form subwindow; the third is a command subwindow, and the fourth is a log subwindow. Take a look at the command subwindow.

At the far right of the command subwindow there is a command called List-Options! Invoke this command; a small options window should appear on top of the command subwindow. This window contains various options that you can ask for when listing information about a file. These are boolean options: that is, they are either on or off. Highlighted items are on; these items represent the information that is currently provided by a list command. To change the setting of a boolean, you need only click over it with Point. Try turning some booleans on and off. When you have set the options that you like, invoke the Apply! command in the Options subwindow. This will put your choices into effect and remove the options window.

The Local-List command lists the specified information about files on the local disk. This command operates on the files listed in the Source: field of the form subwindow. (Throughout the XDE, items followed by a colon indicate that you are to enter a value following the colon. The word preceding the colon is usually a hint as to what information is required.)

Thus, for example, if you want to see all of the files on your local disk, set a type-in point and enter an asterisk (the wild card) in the Source field. (* matches zero or more characters; i.e., *abc matches abc and xabc and yabc.) You can request information about one file, many files, or all files. You can use the wild card anywhere in the file name; for example, if you would like to list all files that have the extension .nsmail, you could enter *.nsmail in the Source field and then invoke Local-List!

The LocalDir: field specifies which local directory is to be searched. If this field is blank, Local-List will list all appropriate

files on the search path. If there is a directory in this field, the search will be restricted to the files on that local directory.

Experiment with Local-List! and List-Options!

Naming conventions: Domains and Organizations

The rest of the messages in this tutorial discuss manipulating files that are stored on remote file servers. To access such files, you first need to understand the conventions by which they are named, and you need to be able to identify yourself to show that you have access to the specified file.

In the XDE, all objects (machines and users) are named according to a hierarchical naming system. The world of objects is divided into organizations, and the organizations are further subdivided into domains. These divisions are logical rather than physical; an organization is typically a corporation (e.g. Xerox), and its domains reflect administrative, physical, or functional divisions within that corporation. Names are of the form <simple name>:<domain>:<organization>.

The simple name of a user is just his or her legal name, such as Mark K. Hahn, or Franklin Lee Yien. Within a particular domain, a user name must be unique: thus, there can be only one Mark K. Hahn in the domain OSBU North, but there can be another in the domain OSBU South.

When referring to a user name, you need only give as much of the name as is required to uniquely identify it. Thus, if you are within the domain OSBU North, you can refer to a printer as just "Pegasus", but if you are outside of that domain you must refer to it as "Pegasus:OSBU North", and if you are outside of the organization Xerox, you must refer to it as "Pegasus:OSBU North:Xerox".

For simplicity, user names can also be aliased. Thus, Hahn might be an acceptable alias for Mark K. Hahn. Aliases are generally created at the same time as the user name; you must use the registered alias or aliases for a name. Aliases must be unique; your system administrator is responsible for ensuring that user names and aliases are unique.

The Clearinghouse command

Each user has an account on a remote file server, which is a separate machine with a large file storage capacity. You should check with someone to find out the name of your server.

To be able to communicate with remote file servers, you must first be logged in. You do not have to be logged in to manipulate files on your local disk, since all files stored there belong to you, but you do need to log in to identify yourself to the remote server.

The first step is to use the Executive's Clearinghouse command to set the domain and organization.

In the Executive, type Clearinghouse, followed by a carriage return. You will be prompted for the name of a domain. Fill in your domain, followed by another carriage return. Next, you

will be prompted for an organization. Fill in the appropriate organization and then another carriage return.

Note: You must include the carriage returns. You cannot type "Clearinghouse osbu north xerox" on one line.

Logging in

To login, type Login (or just log) to the Executive, and follow with a carriage return. The system will prompt you for your user name (your last name; this is usually an acceptable alias.) If you have already logged in once, your name will automatically appear in the User: field. If your name is already there, just confirm it with a carriage return. If it is not there, type it in, and follow with either a space or a carriage return. If you make a mistake while tying your user name, you can use the backspace key to fix the mistake. Figure 2.6 illustrates logging in.

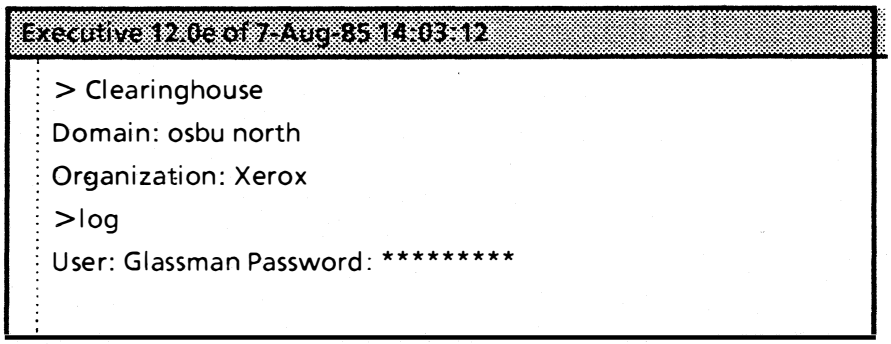


Figure 2.6: Logging in

You will now be prompted for a password. Enter your password; case does not matter. If you make a mistake, you can backspace over it, or hit the DELETE key to start over. (You will learn how to change your password later.)

Type a carriage return after you have finished typing in your password. Within a few minutes, the Herald window will update itself to show whether or not you are correctly logged in. If you think you have made a mistake, just login again right away; the new login will automatically override the old one

Since each machine only has a single user, there is no need for a logout command. However, if you would like to destroy your login so that no one else can use your account to read or send mail or to retrieve files, you can do so by logging in without a password or with an incorrect one. (Note: this will not prevent you (or anyone else sitting at your machine) from looking at the files on your local disk, including your mail files. Therefore, if you are concerned about security, you will have to learn to store your files on the remote server and not on the local disk.)

Storing files on a remote server

Once you are logged in, you can do remote filing operations using the File Tool To store a file on your remote directory, you have to specify where that directory is. Therefore, you should set a type-in point in the Host: field and type in the name of your remote host (file server). This field tells the File Tool the

name of the machine on which you want to store your file. (If you don't know the name of this server, you will have to ask.)

Now press the NEXT key. Your type-in point should now be in the field following the Directory: field. The NEXT key is an accelerator for moving among fields in order.

Fill in the Directory: field with your directory, usually your last name. (If you don't know the name of your directory, ask!)

You can also divide the remote directory into subdirectories, just as you can on the local disk. For example, if you are working on a program and would like to have all the pieces grouped together, you might want to create a subdirectory that is the name of the program. The name of the directory and its subdirectories must be separated by the angle bracket sign. For example, your Directory: entry might be "Miller>current". You may use as many levels of subdirectories as you like.

(**Note**: / and > are equivalent for separating directories.)

The Source field

When you have filled in the name of the directory (and optional subdirectories), use NEXT to advance to the Source field. "Source" is the local name of the file or files that you want to store; type the name of the file in the Source field. If you wish to operate on more than one file, separate the file names with spaces (or use the * wildcard character.) You only need to include a local directory if the file is not on the search path. For example, Figure 2.7 illustrates how to fill in the fields to store the file MesaUnit.df to the subdirectory Mesa Unit in the directory Hal Training on the file server Hal.

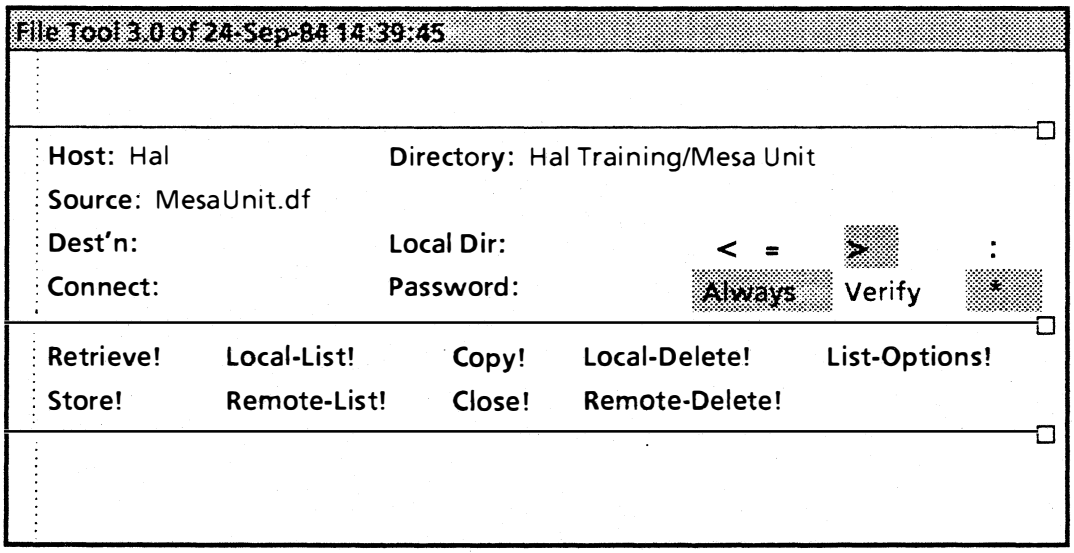


Figure 2.7: Storing a file

Now invoke Store! in the command subwindow, and your file will be stored on the remote file server under the specified directory. While the File Tool is actually performing an operation, the command subwindow is cleared except for a pair of black boxes, which "dance" to show the rate of data

transfer. When the operation finishes, the command subwindow will reappear.

You can store different versions of the file under the same name; they will automatically be given sequential version numbers. The earliest version of a file is filename!1. When you retrieve a file from a remote server, you get the most recent version unless you specifically designate an earlier version.

The Verify option

Using the File Tool to retrieve, list and delete remote files is very much like using it to store a file. You may not need to practice each of these commands, but feel free to experiment.

When you are experimenting with the Delete commands, you can use the Verify boolean of the File Tool to protect yourself from deleting something that you didn't really mean to delete. Find the word Verify near the right edge of the File Tool command subwindow. This word represents a boolean option: when it is highlighted, the option is on; when it is not highlighted, the option is off. To turn on Verify, simply click over the word (the default value is usually off).

When Verify is turned on, invoking a Delete command will not actually delete the file. Instead, the command subwindow will change, and offer you the choices of Confirm! Deny! and Stop!. Invoking Confirm! proceeds with the deletion; invoking Deny! denies it. Confirm! and Deny! appear on a file-by-file basis: if you specify the deletion of a large list of files, you will be asked to confirm or deny each one.

Stop! aborts the entire command, rather than just denying the deletion of a single file. For example, you would want to use Stop! if you acccidentally did a Local-Delete of * instead of a Local List of *. You don't want to confirm or deny each file individually; you just want to stop the command altogether.

Whether or not you use Verify is entirely up to you.

Deleting files

Local-Delete and Remote-Delete are used to delete files from your local workstation or from a remote file server, respectively. Local-Delete can only be used to delete files from your own local workstation; hence, you need only fill in the Source field when using this command. Using this command is equivalent to using the Executive to delete files. You can also specify a local directory if you like; if you don't specify one, the File Tool will look for the file via the current search path.

Local-Delete allows you to delete files either one at a time or by using expansion characters (such as *) to encompass a group of files. With Remote-Delete, however, you can only delete one version of a file at a time; you cannot delete all versions of a file simultaneously. For example, try storing your practice file several more times so that you have three or four versions of it stored on your directory. (You will have to make changes in your file before you can store it. Thus, store a copy, make a change, store another copy, and so on.)

Fill in the fields with the correct information, and invoke Remote-Delete!. Notice that only the EARLIEST version of the file has been deleted. In general, you can only use Remote-Delete! on files from your own directory.

Listing and retrieving remote files

Retrieve! allows you to copy a file from a remote directory onto your local disk. To use this command, just fill in the Host, Directory and Source fields, and then invoke Retrieve! You may retrieve a file from any remote directory to which you have access; you are not restricted to your own directory, or even to your own host. You can use the LocalDir field to specify the local directory that you want the file to be stored in. Note: you will be asked for confirmation on file retrieval when you have the Verify option turned on.

The Dest'n field allows you to rename a file. Thus, if you are retrieving a file called ThisFileHasANameThatIsTooLong, and you want to call it just LongFileName on your own machine, you would put ThisFileHasANameThatIsTooLong in the Source: field, and LongFileName in the Dest'n: field.

Leaving the Dest'n field blank will cause the file to be stored under its original name; that is, the same name as in the Source field.

The Dest'n field is also used with the Copy! command, which copies a local file into another local file. When using Copy!, the name of the file that you are copying from goes in the Source field; the name of the file that you wish to copy into goes in the Dest'n field, as illustrated in Figure 2.8.

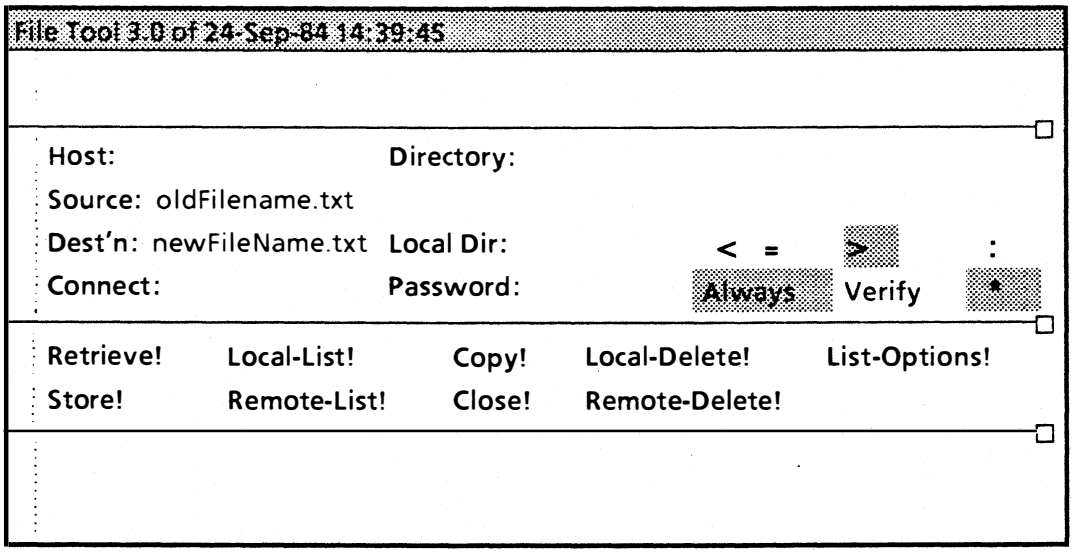


Figure 2.8: Copying a file

Close!

When you are using your file tool to communicate with a remote file server, there is an open connection between your machine and the file server. If you Size or Deactivate the File

Tool window, the connection will automatically be closed. However, if you like to leave your File Tool window active on your screen, you should use the Close! command to close your connection (and free any resources needed to maintain it) while you are not actually using the tool. This is good policy; get in the habit of using Close!.

At the far right of the second subwindow of the File Tool is a collection of mysterious-looking symbols that have been ignored in this tutorial. These symbols are boolean options controlling fine points of the operation of the File Tool. If you are interested, they are fully documented in the File Tool chapter of your XDE User's Guide.

The File Transfer Program

The File Transfer Program (FTP) is a predecessor of the File Tool, and thus the two share many functions. You will want to use the File Tool for most of your filing operations; however, FTP is more useful when you need to operate on a large number of files, such as when you execute a transfer of an entire group of files from one machine to another. The FTP program runs from the Executive window, and FTP commands can thus be included in an Executive command file.

An Executive command file is just a list of Executive commands that are executed in order without requiring any interaction from you. Thus, for example, if you want to back up a group of files onto a remote file server every night before you leave work, you can write an Executive command file that contains the commands to do so. You can then run that command file without having to interact with the File Tool directly.

If you are interested in writing command files that use the FTP program, you should read the FTP chapter of your XDE User's Guide.

FSWindowTool

FSWindowTool is a tool that allows you to access either local files or files on remote file servers. This is a complex tool that provides a great deal of functionality: when you want to do a filing operation and you can't figure out how to do it--try this tool.

To use this tool, you need to run the file NSFilingConfig.bcd, and then FSWindowTool.bcd from the Executive window. Figure 2 9 illustrates this window when it first appears. You have a choice of three options: local, mesa, and remote. Reasonably enough, choosing local allows you to operate on local logical volumes, remote allows you to operate on files stored on a remote file server, and Mesa allows you to operate on files in the "Mesa world." The Mesa world is essentially your CoPilot volume; the Mesa development environment is the old name for the environment itself. In general, you will probably use this tool to operate on files that are stored on your ViewPoint volume or on a remote file server.

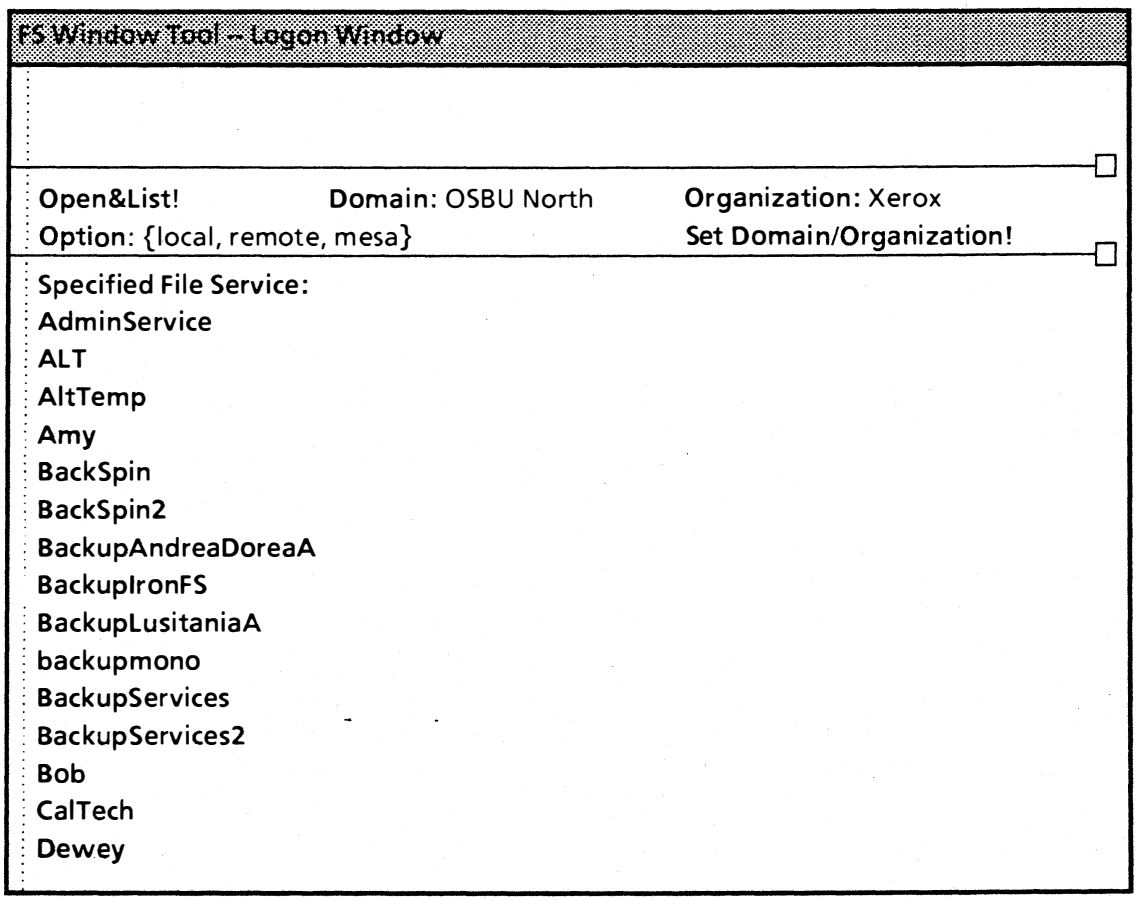


Figure 2.9: FSWindowTool

For now, you don't really need to understand what logical volumes are; you only need to become familiar enough with the FSWindowTool that you can use it when you have to. The following lessons will provide a brief overview of how to use this tool. You should experiment with it as you go along.

Changing directory protection

As an example of using this tool, assume that you want to change the access rights on your directory on your remote file server. To do this, first select the remote option on the logon window. (To do this operation, you will have to be logged in; if you aren't currently logged in, bring up your Executive window and do so You can also use the SetDomain/Organization command to specify the domain and organization of the file server that you are interested in. If the file server that you are interested in is not in your domain, you will have to use this command.)

Selecting the remote option will provide a list of remote file servers in the specified domain and organization. Find the name of your server, select it, and invoke the Open&List command. This will open a window that lists the file drawers on that particular file server. (The list always contains 16 items; if you can't find the name of your file drawer, invoke the Next Page! command)

When you have found your file drawer, invoke Open&List! again. This will open another window, this time containing a list of the files and or subdirectories on your file drawer.

To change the protection on a specific folder or file, select the name of that file from the list, and invoke the Attributes! command. This will open another window that contains the attributes of the specified file. One of the attributes in this list is the Access List:, which specifies the current access rights for the file.

To change this list, just fill in the name of a person or group in the Access Entry: Name: field, and select the desired attributes (Read, Write, Add, Remove, or Owner) from the form subwindow. This items are booleans; if an access is selected, invoking Change Access List! will give the specified person that access; if the access is not highlighted, invoking Change Access List! will remove that access for the specified person.

Experiment

As you can see, this is a very complicated tool, and we haven't discussed all of its capabilities. Experiment with it now as long as you like, and make sure that you read the documentation on this tool. It is a very useful tool. You can use the Close! command in the form subwindow to close each of the windows. When you are through experimenting with this tool, you should now be familiar with most basic filing operations If you are unsure about any of the information presented here, you should go back and review the material until you are fully comfortable with it. When you are ready for more, chord over the File: field in the MailTool command subwindow and select TeachText.nsmail.

3. TEXT

This module covers some of the conveniences available for creating and editing text in the Xerox Development Environment.

Bring up an Empty window on your screen, and load a file into it to use for the following exercises. Most of the commands discussed in this module are best illustrated when there is a reasonably large amount of text in the window being used. (If you don't have a practice file--make one!)

FontMonster

There is a tool called FontMonster that allows you to control the font of the characters in a given subwindow. This tool is not part of the release, however, so it is not necessarily on you machine. To find out if FontMonster is running on your machine, Chord in the grey bit area. If you are running FontMonster, you should see a FontMonster menu. If you do not see such a menu, check with someone to find out how to get FontMonster running on your machine.

Bring the menu labelled [FontMonster] to the top of the menu stack. This menu contains a list of all the font files that are stored on your local disk and available for you to use. To change the font of the characters in this subwindow, select the name of the new font from the [FontMonster] menu; the cursor will turn into a "face". Now click Point over this subwindow, and the font will change.

Warning: Do not use FontMonster on the Table of Contents for the MailTool window. This subwindow is not a standard subwindow, and the characters will not display properly.

You can change the font of most subwindows in the environment. Later in this tutorial we discuss how to change the standard system font, the menu font, and the menu of available fonts.

TextOps commands: Split

All file windows have a menu called the Text Ops menu, which provides commands for text manipulation. To get the Text Ops menu, chord in any file window. The next few sections discuss the commands in the Text Ops menu. Many of these commands are also in the EM Symbiote; you can invoke them either from this symbiote or from the Text Ops menu.

The Split command is used to divide a region into two subwindows, which can be scrolled separately. This is useful if you want to be able to view two sections of a file

simultaneously, or if you would like to maintain your place in a file while looking back at another part of the same file.

Invoke Split in your source window, and notice that a mouse icon appears to ask you for confirmation. Position the cursor at the point where you would like to place your dividing line, and click Point to confirm the Split command. (**Note**: the subwindows must be each be at least three lines high.) Figure 3.1 illustrates a split file window with a Mesa source file in it.

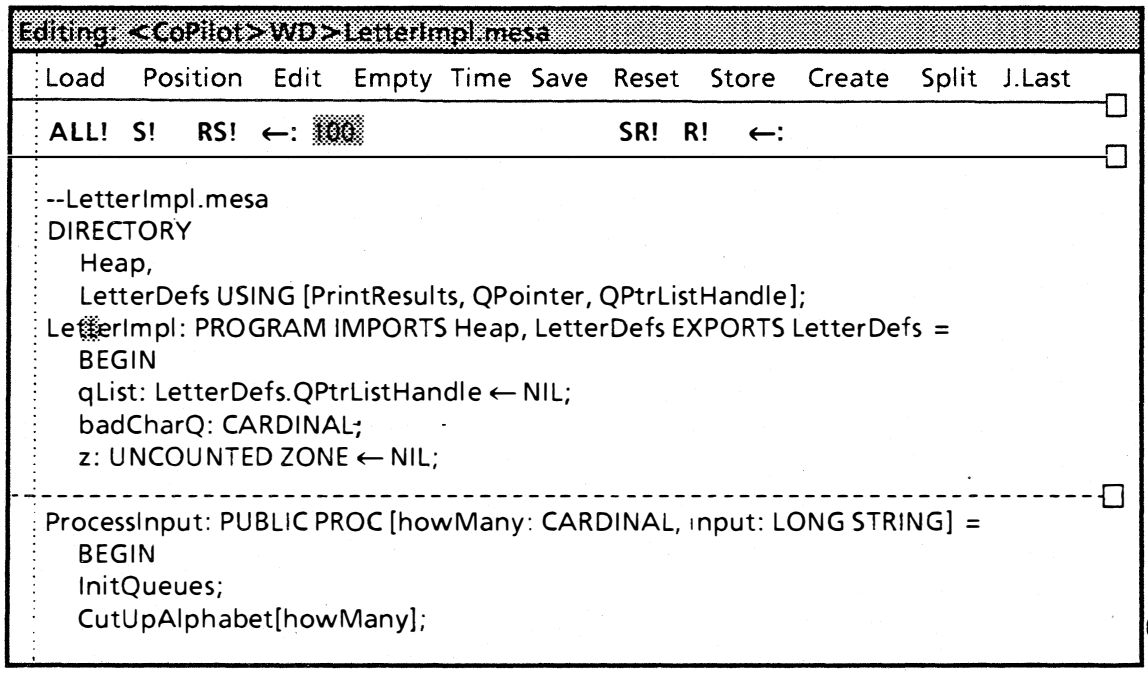


Figure 3.1: A split window

The dotted dividing line can be moved in the same manner as any dividing line; moving it off the top or bottom of a window will remove it. You can split a window multiple times

Position, J.First, J.Last, J.Select, and Wrap

The Position command is used to specify a particular character by its numerical position in the text. For example, suppose that you run a program (such as the compiler) that tells you that there is something wrong with the 100'th character in your file. To scroll this character to the top of the window, first make sure that the number 100 is typed somewhere on the screen (in digits). If the number does not already appear in the text, type it in one of the fields in the Find symbiote Select it. Invoke Position, and the 100th character in the text will be selected and scrolled to the middle of your subwindow. (This is illustrated in Figure 3.1 above.)

Note: Depending on how your machine is set up, the selected character may be at the top, the middle, or the bottom of the subwindow. We discuss how to determine where the character will appear later in this tutorial.

J.First positions the first line of text in a file at the top of a subwindow; J.Insert positions the type-in point at the top;

J.Select positions the first character of any selected text at the top; J.Last positions the last line of text at the top of the window. Experiment with each of these commands; you will find that they are often more convenient to use than your scrollbar. (You might find "Jump to" a convenient mnemonic for J.)

The Wrap command controls whether text that is typed without any carriage returns will continue onto the next line or disappear off the screen. Unless you turn Wrap off, a line that has not been terminated by a carriage return will automatically be continued onto the next line. Invoke Wrap again (to turn it off), and type a line of text. Notice what happens when you reach the edge of your screen and keep typing. Now turn Wrap back on by invoking it again. You will probably not want to turn Wrap off very often.

The J. series commands are also available on your keyboard. Refer to your keyboard chart or to the last page in Chapter Four of your XDE User's Guide to find out which combinations of keys will perform these functions.

The Find commands

Suppose that you are working on a file in which you have been discussing masa, and someone points out to you that the word should be spelled "mesa". You don't want to have to skim the entire file in order to find each time that you misspelled the word, so you decide to ask the system to do it for you.

One way of doing this is to select one instance of the misspelled word and then press the FIND key on the left side of your keyboard. This will search for the next occurrence of the selected text. When another instance is found, it will video-invert. You can then edit it, and press FIND again in order to find the next instance of the mistake (You can also use the Find command from the Text Ops menu the Find command and the FIND key are equivalent.)

One difficulty with this method is that the search for the pattern starts at the instance that you have selected. If you happen to select the first occurrence of something in the file, this is fine; otherwise, you will not "find" the instances that occur early in the file. For example, try selecting the word file in this sentence. Now invoke the Find command using the FIND key. It will find the next instance of the word file that occurs in this tutorial, but will not return to find any instances of it that occurred before this paragraph.

You can, however, hold the SHIFT key down while pressing FIND; this will cause the search to proceed backward from the selected text. Try this. Notice that the FIND key always needs a currently selected argument; in other words, you have to have at least one instance of the desired sequence available in order to use this command.

The Edit symbiote

A difficulty with using the FIND key to alter misspellings is that you need to make each of the edits individually. For more

advanced editing capability, you can use a second symbiote, called the Edit Symbiote.

The next several lessons discuss the Edit Symbiote. You should take some time to experiment with the various features as they are presented.Don't worry about remembering all the details, though; you can always look them up later.

The Edit symbiote is found beneath the EM Symbiote; it contains the commands All!, S!, RS!, SR!, and R!. There are also two fields in this subwindow, indicated by "←:". The arrow points to the command or commands with which the field is associated. The field on the left is the "find" field; the field on the right is the "replace" field.

Note that if you set a type-in point in an Edit symbiote and press the DOIT key, the symbiote will increase to two lines high instead of just one.

The commands function as follows:

All! changes every instance of the string in the find field to the string in the replace field.

S! (Search) searches for the string in the find field.

RS! (Replace/Search) does an R! followed by an S! .

SR! (Search/Replace) does an S! followed by an R!

R! (Replace) replaces the current selection with the text in the replace field.

Search expressions

There are several special conventions that you can use in the find field; you are not restricted to just using literals. This message defines the expressions that you can use in this field.

\# matches any character.

% when used as the first element in a pattern, matches the beginning of a line.

[...] allows you to define a character class. A character class consists of zero or more of the following items:

- a literal
- a range of literals (a-g)
- a negated character class
- an escaped character (\c). The two escaped characters that you are most likely to use are \n (carriage return) and \t (tab).

[~...] specifies a negated character class

* "short closure"; finds the shortest possible match.

** "long closure"; finds the longest possible match.

For example, to find a word whose first character is a consonant with a curve in it, whose second letter is a vowel, and whose third letter is between b and r, you could use the following character class:

[bcdgjpqrs][aeiuo][b-r]

To find a word that starts with an upper case letter:

[A-Z][a-z]**

Replace expressions

The Replace expressions are a concatenation of the following:

s	literal
@&	complete match, matches the text found
@n@	partial match, where n = 1,2,3... . n corresponds to the nth element of the regular expression

For example, to delete leading zeroes from numbers:

Find: [~0-9][0]**[0-9]

Replace: @1@@3@

Result: 000000B = > 0B, 00343B = > 343B

Editor property sheet

There is a property sheet associated with the edit symbiote that allows you to control the details of the operation of these commands. This sheet can be found in the window labelled Editor. Its active state is illustrated in Figure 3.2.

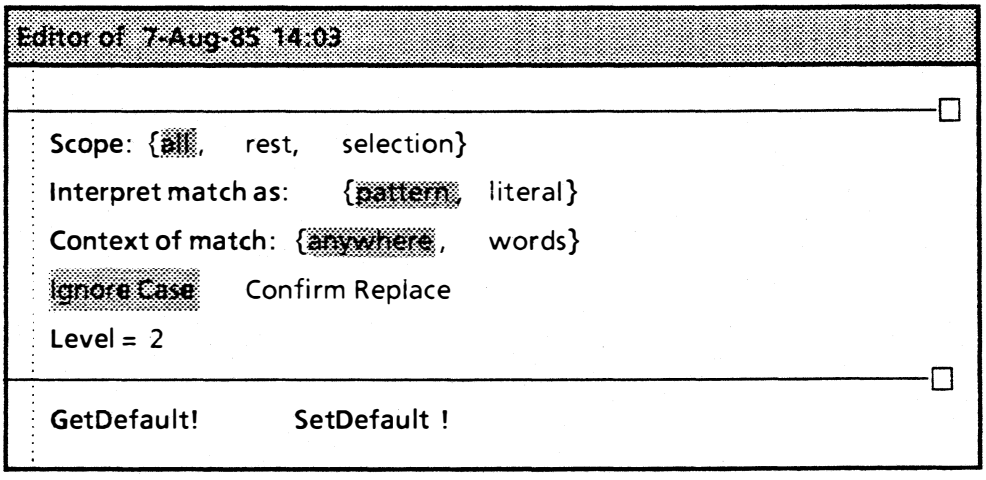


Figure 3.2: The Editor property sheet

Scope: controls the scope of the All! function. {all} uses the entire file as the scope for All!. {rest} causes the command to start either at the top of the page, or from the insertion point if it is visible. {selection} restricts the replacement to the current selection.

Interpret match as:
determines whether the string in the find field will be interpreted as a pattern or a literal; that is, determines whether characters such as * are interpreted as just characters or as wildcards.

Context of match:
determines whether the desired string can occur in the middle of a word or not. "anywhere" means that the pattern can match within a larger word; "words" will only match patterns that are surrounded by non alphanumeric characters.

IgnoreCase
When this boolean is on (highlighted), case will be ignored when searching for literals.

Confirm Replace
When this boolean is selected, you will be asked to confirm each replacement individually.

Level is discussed in the next message.

The property sheet also has a command subwindow with the commands:

GetDefault! Sets the properties back to a default state.

SetDefault! Lets you specify the default state.

The EditOps menu

When an edit symbiote is attached to a window, an Edit Ops menu is attached to that window as well. This menu contains the same commands as the symbiote, and the additional commands Nest UnNest Match and Count. These commands, which operate only on text subwindows, function as follows:

Nest shifts the lines containing the current selection "level" characters to the right, where level is specified in the editor property sheet.

UnNest shifts the lines containing the current selection level characters to the left.

Match identifies matching parentheses, angle, square, and curly brackets. When one of these character types is selected, Match will extend the selection to the matching character. If a character that is not one of these types is selected, the selection will extend in both directions until it contains a match. Successive uses of Match will match larger scopes.

Count counts the number of occurrences of a pattern. The scope and target are specified as in the All! command.

The result is given in the message subwindow of the editor property sheet.

The Edit dictionary

The Edit dictionary allows you to create your own set of abbreviations for use in any text window, so that you can simplify the typing of words and phrases that you use often. To get the window for this tool on your screen, press the SHIFT key and the DEF'N key (on the right side of your keyboard) simultaneously. The Dictionary Tool is illustrated in Figure 3.3.

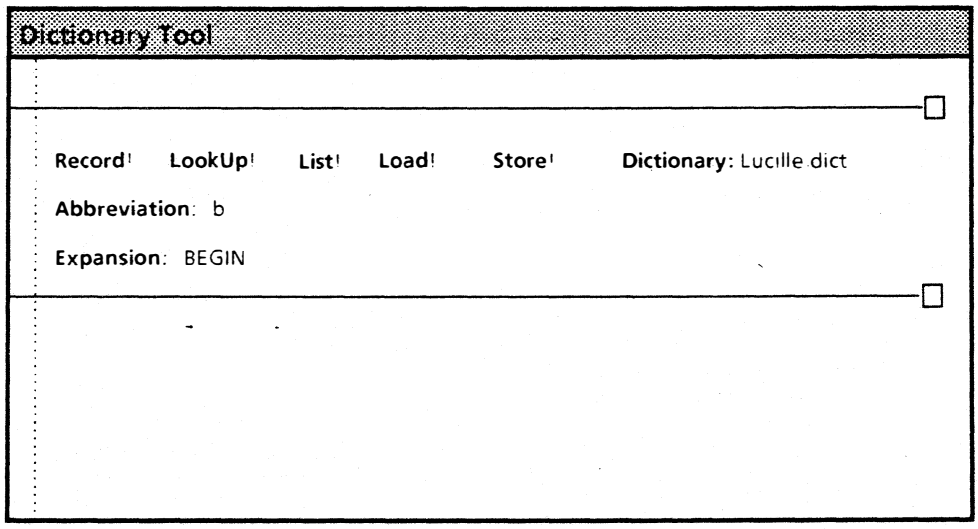


Figure 3.3: The Dictionary Tool

The Dictionary tool maintains a dictionary of abbreviation-expansion pairs in a format that permits fast look-up. There is no limit to the number of pairs that you can have in your dictionary. However, when you boot your volume, the pairs stored in your Dictionary will be lost. To protect against this, you can store the dictionary in a file with the extension ".dict". This file will not be destroyed when you have to boot, and you can just load it back into the dictionary tool whenever you like.

Find the Dictionary: field at the far right side of the tool. This field is used to specify the name of the "dictionary" into which you are going to enter your abbreviations. You can fill in this field with any name that you like, provided that it ends with the suffix .dict. You can leave "Default.dict" if you like; this dictionary is currently empty.

You can also maintain several separate dictionaries. For example, you might want one dictionary that contains words that you frequently use when programming, and one dictionary that contains words that you frequently use when creating text files.

Defining an abbreviation

To define an abbreviation, you first need to fill in the Expansion: field with the word or phrase that you want to abbreviate For example, suppose that you type "BEGIN" a lot,

and get tired of using the SHIFT key all the time. To circumvent this problem, you can define "b" to stand for "BEGIN".

To do this, fill in "BEGIN" in the expansion field. Now fill in the abbreviation that you would like to stand for the expanded term in the abbreviation field; in this case, "b".

Now invoke Record!. The abbreviation is now recorded in the dictionary. To store it permanently in the .dict file , invoke the Store! command. Now bring up an Empty Window, and type "b", followed by the EXPAND key (in the cluster at the right of your keyboard). You can use this key to expand abbreviations while working in any window that allows you to type in text.

Note: the abbreviation must be separated from any previous text by a space. Thus, b will expand to BEGIN, but ab will not expand.

In addition to defining an abbreviation for a particular word, you can also define an entire phrase. For example, you might wish to define "xde" to stand for "Xerox Development Environment". Abbreviations cannot contain spaces, however.

Automatic abbreviations

One difficulty with defining abbreviations in the Dictionary tool is that you add abbreviations at different times, and may make the mistake of defining an abbreviation that you have already defined. This enters the new abbreviation-expansion pair in the dictionary, but destroys the old pair. If you would like to avoid this problem, you can let the dictionary tool avoid duplicate definitions for you.

If you enter a word in both the Abbreviation: and the Expansion: field (for example, if you enter "begin" as the abbreviation for "BEGIN"), the dictionary tool will automatically pick the shortest possible abbreviation for that word. For example, it will define "b" as "BEGIN" as long as BEGIN is the only entry beginning with that letter. If you then make another entry that defines "base" as "base", the dictionary will require you to type "be" to identify "BEGIN" and "ba" to identify "base".

Allowing the dictionary to choose the abbreviation algorithm for you has the advantage that the abbreviation system is consistent and that you can always obtain the correct abbreviation for something; however, you can always choose your own abbreviations if you would like to do it differently. In either case, you can always use the List! command to view the pairs currently in the dictionary. This will help you avoid overwriting an existing abbreviation.

To let the Dictionary tool choose the abbreviations, fill in the entire word or phrase in both fields (Abbreviation: and Expansion:); to choose the abbreviation yourself, enter the full phrase in the Expansion: field, and your chosen abbreviation in the Abbreviation: field.

Other Dictionary tool commands

In addition to the Record! command, there are several other commands available for the Dictionary tool.

Lookup! is used to reference the expansion associated with a particular abbreviation. Try using this command now to look up the abbreviation that you just defined.

List! lists the pairs currently stored in the dictionary database.

Load! loads the specified .dict file into the Dictionary tool database. You will need to load your .dict files each time that you re-initialize (reboot) your volume.

Try defining some words and phrases in your dictionary, changing the name of the dictionary for which you are defining terms, and experimenting with the commands available for the Dictionary tool. For more information on the Dictionary Tool, refer to the Edit dictionary chapter of your XDE User's Guide. When you are comfortable with this tool, you are ready to move on to the next message.

Your user.cm file

Your user.cm is a special file that allows you to personally tailor your workstation environment. For example, this file allows you to specify which tools you want loaded when you first boot, where you want the tiny window for a particular tool to appear, and where you would like active windows to appear. You should have a copy of a user.cm file on your local disk. Load this file into a source window.

The user.cm file consists of a group of separate entries, each headed by a title in brackets, such as [System]. A user.cm file can contain a section for every tool in the environment, as well as sections that apply to an entire logical volume or group of logical volumes. Your user.cm file can be as complex or as simple as you like: you do not even have to have one at all. Remember, though, that the trade-off for a complex user.cm file is that it will take you longer to initialize your system when you boot.

A user.cm file can be organized in any way that you choose; there is no required order for the entries. Some users choose to have the individual sections sorted alphabetically; others group by association, and still others have no system at all. The organization of your user.cm file is entirely up to you.

Whatever the organization of the user.cm file existing on your disk, you should be able to find each of the sections discussed in this tutorial. If you can't find one, create it (just edit it into the file).

The [System] entry

The [System] section specifies parameters that apply globally to your entire system. It is thus the most general section of your user.cm, and is initially processed when a volume is booted.

(Any changes you make to the [System] section will not be applied until you reboot.) A sample [System] entry might be:

```
[System]
Domain: OSBU North
FileWindow: [x: 0, y: 100, w: 512, h: 321] [x: 300, y: 778]
    CurrentTasks.txt/t
Font: Gacha12.strike
MenuFont: TimesRoman8.strike
Organization: Xerox
Screen: White
SetPositionBalanceBeam: middle
User: Glassman
```

In this case, the individual entries within the section are sorted alphabetically, but the order of entries is up to you.

Domain:, **Organization:**, and **User:** provide information about the user of the system. You should update these entries in your user.cm. (You will have to ask someone what domain and organization you belong to.)

FileWindow: specifies that upon initialization, a file window will be created at the given location. A window is positioned with four parameters: the x and y coordinates of its upper left corner, and its width and height. The numbers are given in pixels: the screen measures 1024 pixels wide by 808 pixels high. The position [x:0, y:0] is at the upper left of your screen. Any or all of these four numbers can be defaulted; the default values are [x:0, y:0, w:512, h:400]. To default all values, specify [].

The second set of numbers in the FileWindow entry, in this case [x: 300, y: 778], determines the tiny position for the window Finally, you can give a name after the tiny position to specify the name of a file that you would like to have loaded into the file window. This is optional; you can have an Empty window if you like. The switch following the file name can be i(inactive), a(active), or t(tiny); a switch is just a way of "fine-tuning" a command. You don't have to have a switch at all.

You may have as many FileWindow: entries in your [System] section as you like.

Font: and **Menufont:** determine the font in which characters on your screen will appear. To have your screen displayed in a particular font, you must have a corresponding file on your local disk that contains the information on how to display that font. You will have to ask someone where the font files are stored, and which fonts are available to you. Note that the font file corresponding to the font that you request for your system font must be stored on your root directory (not on a subdirectory). The reason for this is that the [System] section is processed before the search path can be set, and the only directory in which files can be found is the root directory.

SetPositionBalanceBeam: applies to the Position command; it specifies where the selected character will appear. The possible values here are middle, top, and bottom. If you specify middle, the character will be scrolled to the middle of the screen; top

will position the character at the top of the screen, and bottom will position the character at the bottom of the screen.

Screen: specifies the background color for your screen. There are two choices: black and white.

[CoPilot: System]

You can also have a user.cm entry for each individual logical volume; for example, [Tajo:System] or [CoPilot:System]. Any entries in the volume-specific system section will override the more general ones provided in the [System] section, if there is a conflict. The two most common entries in the [CoPilot:System] section are illustrated below.

```
SearchPath: <CoPilot>tools <CoPilot>

InitialCommand: Run. MailTool Print Compiler Calendar
```

The SearchPath entry sets the initial search path.

Commands listed in the InitialCommand line will be run during initialization. An InitialCommand line consists of Run. followed by a list of the programs that you wish to run. Remember that a long list of tools will take a correspondingly long time to load.

[FileWindow]

In the FileWindow section, you list the commands that you would like to appear in your EM symbiote, and determine the order in which they are to appear. For example,

```
[FileWindow]
 Menu: Create Destroy Break Save Store Position Reset Split
 SetUp: Always Menu Edit
```

Menu: is the list of commands for the EM symbiote, in the order in which they will be listed in the menu. **SetUp:** determines when the EM Symbiote should appear. Always indicates that you would like symbiotes attached to all file windows. Menu and Edit indicate that you would like both symbiotes. If you wanted only an EM symbiote, and not an Edit symbiote, you could use Always Menu.

Entries that apply to all tools

The sections discussed so far have been global sections; that is, they apply to more than one tool and are processed when the volume is booted. You can also have a section for every tool in the environment. Tool-specific sections are processed when the tool is loaded.

This section lists entries that can appear in the section for any tool. These entries apply to the tool's window, rather than to the actual functionality of the tool.

A **WindowBox:** entry specifies the location that the tool window should occupy when active. A WindowBox: entry requires four parameters: x, y, w, and h. These parameters are the same as the ones for the FileWindow: entry in the System section, discussed in the message titled [System].

A **TinyPlace:** entry specifies the x and y coordinates of a tiny window, in pixels.

An **InitialState:** entry can have one of three values: Active, Inactive, or Tiny.

Entries for specific tools

Almost every tool in the Xerox Development Environment provides possible user.cm entries. The user.cm entries for each tool are documented at the end of the associated chapter in the XDE User's Guide. You should eventually read these sections and customize your user.cm accordingly.

For example, search through your user.cm for the [FontMonster] section. This section lists the fonts that are to appear in the FontMonster menu. To change the items on this menu, you need to retrieve the appropriate font files from the remote fonts directory, and change the entries in this list accordingly. You also need to make sure that the directory where the font files are stored is on the search path when FontMonster is run. (The .strike extension is conventional for fonts; all font files have this extension.)

FontMonster will create a new menu each time that it is run. You will need to reboot if you want to get rid of the old menus.

What now?

You have finished the core set of tutorials. There are three more optional tutorials available, called TeachBooting.nsmail, TeachPrinting.nsmail, and TeachMailSystem.nsmail. You may do these tutorials in any order

When you are through with the optional tutorials, you should read TeachCompile-Bind-Run.nsmail, TeachDebugger.nsmail, and TeachToolBuilding.nsmail, in that order. (Note: these final three tutorials are only for those of you who will be programming in the XDE.)

This module concentrates on the electronic mail system used in the Xerox Development Environment. It discusses how to read your mail and how to send mail to others, how to print a mail message, and how to fix a damaged mail file.

The Mail Send Tool

The mail system has two basic functions: sending mail and retrieving mail that others have sent to you. You send mail with a tool called the Send tool, which is illustrated in Figure 4.1. To get this window on your screen, invoke New Form! in the command subwindow of the MailTool window. If your Mail Send window obscures any of the text in the MailTool window when it opens, adjust your windows until it no longer does so.

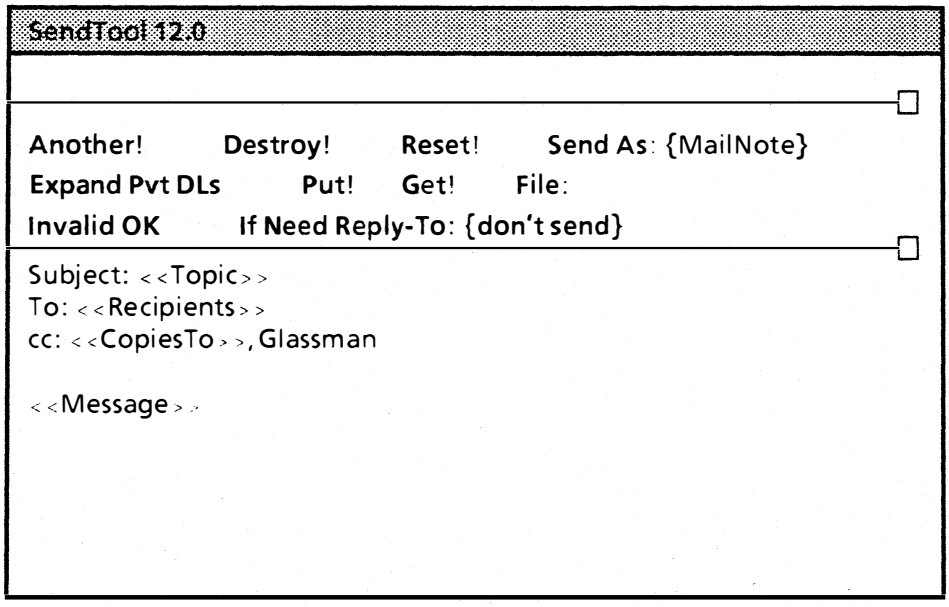


Figure 4.1: The Send Tool

The Mail Send window has three subwindows: a message subwindow, a form subwindow, and a text subwindow. The text subwindow also contains some fields, such as Subject.

To send a message, you must first specify a subject and at least one recipient for it. To simplify the job of addressing a message, the Send Tool text subwindow has fields for subject and recipients. To enter a subject for your message, click Point over the word Subject:, and then press the NEXT key. The word Topic will disappear, and a type-in point will appear in its place, ready for you to enter the subject of your message.

Type in a subject title for your message. Remember that the topic of the message will appear in the Table of Contents of the

recipient's mail window. Therefore, the topic should accurately express the content of your message so that interested people will take the time to read the message, but others can delete it without reading it. For example, if your message contains ideas for improving this tutorial, the topic might be "Suggestions: improving the mail tutorial," not "Tutorial" or "Suggestions".

When you are finished entering your subject, press the NEXT key again. As you have seen, this key allows you to move between fields without using your mouse. You can use NEXT with any tool that has fields.

You should now be in the To: field, with your type-in point set. The word Recipients should disappear just as Topic did; the words enclosed in brackets are simply reminders of the kind of information that should go in the particular field.

The To: field

The To: field should contain the user names of the recipients of your message. A user name has three parts, separated by colons, as in James R. Herz:OSBU North:Xerox. The last part is an "organization", the middle part is a "domain", and the first part is the name of someone in that domain and registry.

Domains and organizations are just a device for grouping names. Domains correspond roughly to organizations within Xerox, such as OSBU (Office Systems Business Unit) North or OBSU South. Most Xerox employees are in the organization Xerox. A fully-qualified name includes all three parts, but you don't always need to use a fully-qualified name. When there is no ambiguity, you can omit both the domain and registry.

For example, you can omit the domain and organization for recipients who are in your registry, just as you may omit an area code when telephoning someone in your neighborhood. Thus, someone in OSBU North could send a message with the following acceptable message header:

```
Subject: Demonstration of recipient naming
To: Person1, Person2, Herz
cc: FarAwayPerson1:OSBU South
```

MailTool assumes that names without registries are in the sender's domain and organization, which in this case is OSBU North:Xerox. However, since the person receiving a copy of the message has an explicit domain (OSBU South), MailTool knows to send the copy of the message there. (Note: Commas must be included between the names of multiple recipients.)

Fill in the To: field with the name of a friend, or your own name. Most users have aliases for their fully-qualified name. The most common alias is just a person's last name, but in some cases includes a first initial or several initials. If you are unsure of someone's user name, check before you send him mail. Because there is occasional ambiguity with user names, you may sometimes receive mail which was not intended for you. If this happens, you should forward the message message back to the person who sent you the message, and explain that the message did not reach its intended destination.

You can capitalize the recipient names any way that you like: Jones:OSBU North, jones:osbu north, and jOneS.OSbu nOrTh are all equivalent.

The other header fields

The next field in the text subwindow is the cc: field, in which you can enter names of people to whom you wish to send copies of your message. Your own name is automatically included in this field, so that you will receive a copy of each message that you send. You may fill in this field with any acceptable recipient, just as in the To field, or you may delete the field altogether if you do not wish to send any copies of your message, even to yourself. The cc: field is optional; the Subject: and T:o fields are mandatory.

When you have a message in your mail file, the name of the sender appears in the Table of Contents. For example, the messages in this file are all from XDE-Training:OSBU North. When you send a message, therefore, your name will appear in the Table of Contents for the recipient. There are times, however, when you want a name other than your own user name to appear in this field. For example, if you are acting for your group, you might want to have the message "from" your group, rather than from you personally.

To do this, you can add a From: field to the message, directly below the Subject: field. In this field, you can specify the name that you would like to have appear in the Table of Contents. (When you use the From: field, MailTool will automatically generate a Sender: field that contains your own user name. Thus, when you use the From: field, the recipients of your message will be able to tell that you actually sent the message; the only effect of using a From field is to change the name that appears in the Table of Contents.) If you received the message in Figure 4.2, for example, XDE Consultants would appear in the From: field of the message, not the name of the user who actually sent the message

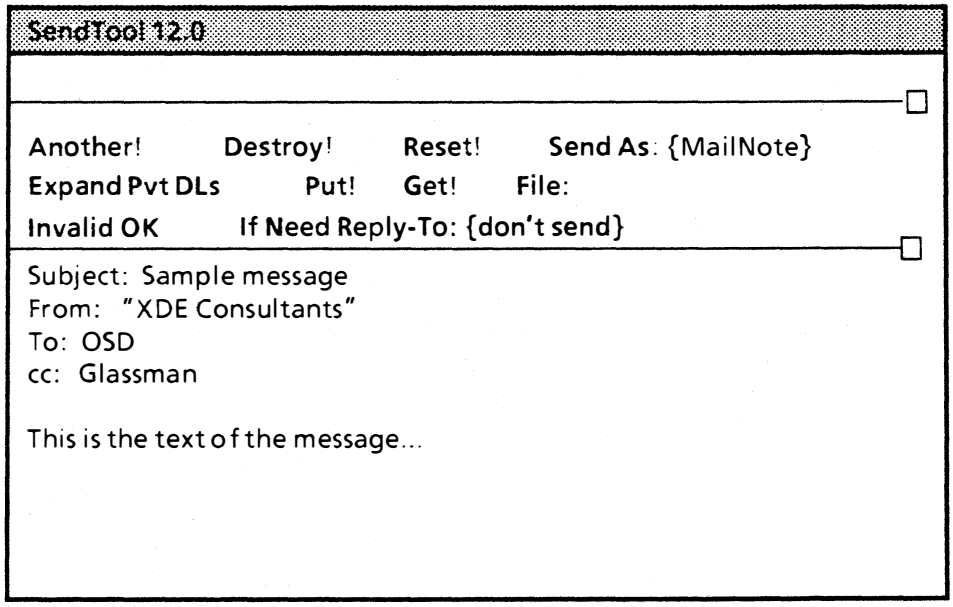


Figure 4.2: The From field

Which type of message?

The Send As: enumerated item is used to choose the kind of message that you are going to send. If you chord in this field, you will see that there are three choices: Text, Mail Note, and Mail Note with attachment. A Mail Note is the simplest of the three, but it is limited to 8,000 characters.

If you select Mail Note with attachment, you will see that two new fields appear at the bottom of the command subwindow, Attachment Type: and Attachment File:. The facilities to implement this choice are not yet fully implemented; you should not worry about the Mail Note with attachment choice for now.

The last choice is Text message. A text message is a more complicated form of a mail note, and requires more overhead, but frees you from the 8,000 character restriction. See the documentation in the XDE User's Guide for more details.

Sending your message

When you have finished filling in the form fields of your Mail Send Tool, you are ready to type your message. Pressing NEXT again will advance to the <<Message>> field, delete the word Message, and set a type-in point in the text subwindow. Now type a message, using the editing techniques that you already know. Type a message that consists of your comments about this tutorial, and your suggestions for its improvement. You can return to edit the form fields (the header) at any time before you actually send the message, if you decide that the information you filled in is inappropriate or incomplete.

Deliver your message now by invoking the Deliver! command in the Mail Send Tool. Notice that the message subwindow at the top of the window gives feedback on the delivery of your message, and that Deliver! changes to say "delivered" when the message has been successfully delivered to the mail service. Notice also that Deliver!" appears only after you have edited a blank form or after you have edited a message that has already been sent.

Public Distribution lists

The mail system also provides a way to address messages to groups of recipients. The Clearinghouse allows the definition of "groups", which are just collections of users (e.g. Software Support:OSBU North:Xerox). There are also some distribution lists that are left over from a previous mail system, and are specified with the format GroupName↑:PA or GroupName↑:ES or the like. These distribution lists are fairly common within Xerox.

There should be a list of existing groups available to you; ask someone in your area where to find it. You should think carefully about your choice of message and list so as not to bother recipients with messages they don't care to read. Check with experienced users to find out which groups should be used for which kinds of messages

You can also create your own private distribution list by typing the recipients as a file, and then sending the message to the file name. For example, the messages you are reading were sent to the private distribution list NewUsers.

The correct syntax for creating a private distribution list is given in your XDE User's Guide.

Other Mail Send commands

Look again at the command subwindow of your Mail Send Tool. Find the commands Another!, Destroy!, and Reset!. Invoke Another!. You will now have a second Mail Send tool on your screen. Type in your own user name in the Subject field.

Now invoke Reset!, and click Point to confirm the command. Notice that Reset! leaves the window open, but clears it of the text you inserted.

Now invoke Destroy!. Destroy! does just that: removes the window and everything in it. If you have destroyed all your Mail Send windows, invoke New Form! in your MailTool window to get another. Destroy the two Mail Send windows on your screen. (MailTool will ask for confirmation of the Destroy command if you only have one Mail Send window open.)

"Put" allows you to store an unfinished message in a file on the local disk. You can specify the name of the file in the File: field. Later, when you want to finish sending your message, you can use Get! to get the unfinished message out of the file and back into a Mail Send.

The Reply-To field

When you are sending a message to a large group of people MailTool sometimes requires that a message have a Reply-To: field. This precaution is to insure that someone who answers the message will direct the answer only to the author of the message, and not to the entire list of recipients.

The If Need Reply-To option on the Mail Send Tool determines what MailTool does when it decides that a message should have a Reply-To: field. Call up a new form and find this option; you should see {don't send} after If Need Reply-to.

A field like this one (the name of an option followed by a colon, followed by something enclosed in curly brackets), signifies that the option is an enumerated type item. This means that the possible values for the option are listed in a menu, and that you specify a value for the option by selecting a choice from the menu. To view the menu for this option, move the cursor to point at the words If Need Reply-To:, and press Chord. The choices listed on the menu are "don't send", "add to form", and "send anyway".

"Don't send" means that if MailTool thinks that a Reply-To field is necessary, it will refuse to deliver the message without one. Instead, the tool will post a message in the message subwindow of your Mail Send Tool informing you of the

problem. When this happens, you can either override the need for a Reply-To: field, or add one to the form. Using the "don't send" value here serves to alert you to the fact that you are addressing a message to a large group of people, and effectively asks you to check that you know what you are doing.

Add-to-form means that MailTool will automatically add the Reply-to field to any message that it thinks should have one, and fill it in with your name.

"Send anyway" overrides the requirement for a reply, and delivers the message without a Reply-To: field. Unless you are certain that you want everyone listed in your To: and cc: fields to receive copies of all replies to your message, it is best to make certain that there is a Reply-To: field, and that it carries your name alone. You should avoid burdening a group of people with copies of other people's answering messages.

Reading your mail

Now that you know how to send messages, it is time to learn how to read the messages which are sent to you. If your mailbox is currently empty, use your Mail Send Tool to send yourself some mail. Figure 4.3 shows what the Mail Tool herald looks like when you have new mail.

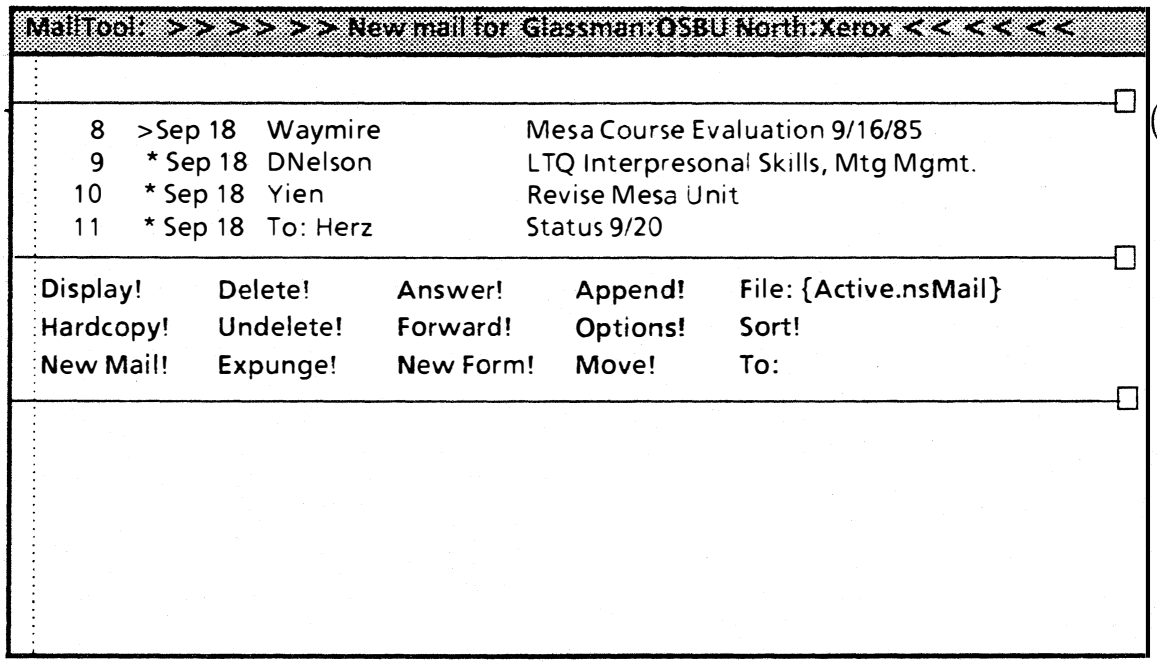


Figure 4.3: The New Mail reminder

Each user of the mail system has a "mail server" in which mail directed to his or her user name is deposited. When you ask to read your mail, the mail messages with your name on them are removed from the remote server and attached to the end of your current mail file.

When your MailTool window tells you that you have mail, invoke "New Mail!" in the command subwindow. Watch the uppermost subwindow (the message subwindow). This subwindow will print the name of the mail service where your

mail is stored, and the number of messages that MailTool has retrieved from that service.

When MailTool has found all your new mail, it will append the new messages to your current mail file (the one that you are reading when you invoke NewMail!), and add a Table of Contents message for each. Notice that your Table of Contents has been scrolled to display the new messages, and that the title of the first new message has been selected. To return to the tutorial, therefore, you will have to scroll your Table of Contents and select the message entitled "FlushRemote".

FlushRemote

Normally, when you use New Mail! to retrieve your mail, the messages are moved from the file server onto your local disk, without leaving a copy on the remote server. Sometimes, however, such as when you are not using your own machine to read your mail, you may wish to keep a remote copy so that you can eventually retrieve the messages permanently onto your own machine.

The FlushRemote option in the MailTool options window can be used to read your mail without removing it from the mail server. (The Options window is illustrated in Figure 4.4. To get this window on your screen, invoke Options! in the command subwindow.) The FlushRemote option is usually turned on, meaning that no copy is kept on the remote server. To keep a copy on the server, turn this option off.

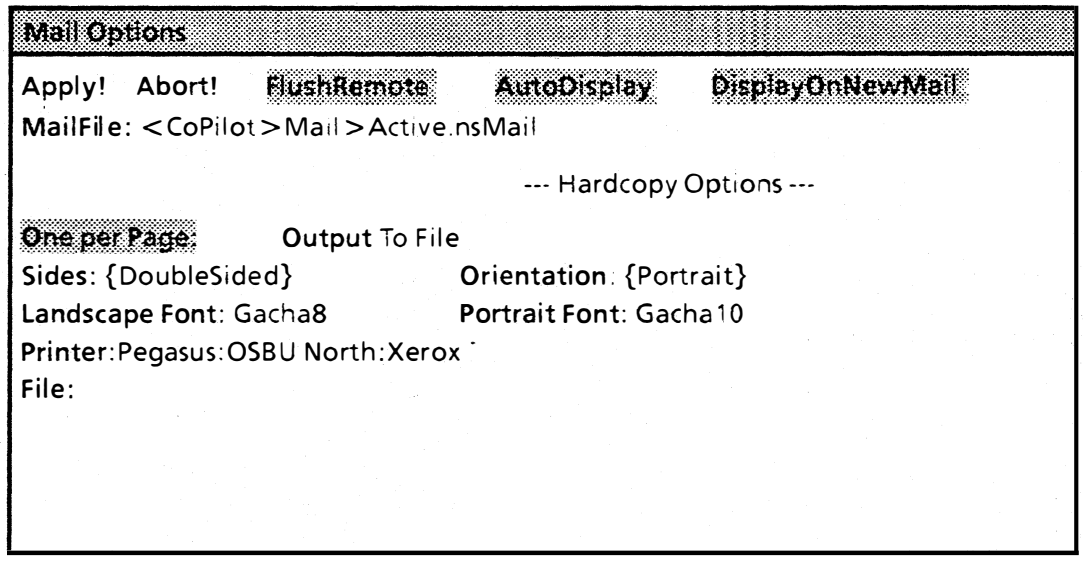


Figure 4.4 Mail Options window

Deleting your mail

When you have read your mail, you may either transfer it into another file, keep it in your mail file, or delete it. This message deals with deleting mail; the next few messages show you how to move your mail

The Delete! command in the MailTool command subwindow (not the DELETE key) deletes a message from your mail file. Select the title of this message in the Table of Contents and invoke Delete!. The message title will be crossed out in the Table of Contents, signifying that the message is marked for deletion, but the message itself will not actually be deleted yet.

If you make a mistake in marking a message for deletion and would like to restore it to good health, select the message title and invoke Undelete!. Do this now.

When you invoke Expunge!, deactivate MailTool, or change mail files, all messages marked for deletion will really be deleted and there will be no way to restore them. Be careful about the messages you delete.

Alternative mail files

You can maintain different mail files for different kinds of messages just as you can maintain different categories in a file cabinet or card file. Using several mail files is a useful way to organize and categorize your mail. For example, you have been reading these messages in several different tutorial mail files, separated by subject.

You can have as many mail files on your local disk as you like; a mail file is just a text file in a special format. However, you can only read one mail file at a time using MailTool.

To see how mail files work, suppose that you want to move the next message in this file to another mail file called temp.nsmail. The To: field in the MailTool command subwindow is used to specify the name of the mail file to which you would like to move a message or messages. Enter temp in this field. (Mail files conventionally have the extension .nsmail; you do not have to type the extension unless you want to name a mail file with an extension other than .nsmail.)

Now select the title of the next message ("Reading another mail file") and invoke Move!. If the file temp.nsmail does not already exist on your local disk, the iconic mouse will appear, asking you for confirmation of the command. Click Point to confirm.

Moving a message to another mail file causes it to be marked for deletion in the current mail file. You should use Undelete! to restore the message. (You will now have a copy of the message in each of two mail files.)

There is also a menu of available mail files atttached to the To: field. Thus, when you want to move a message to a mail file that already exists, you can just chord over the word To: to obtain a list of all the mail files on your disk, and select the one to which you want to move the message. Figure 4.5 illustrates this menu.

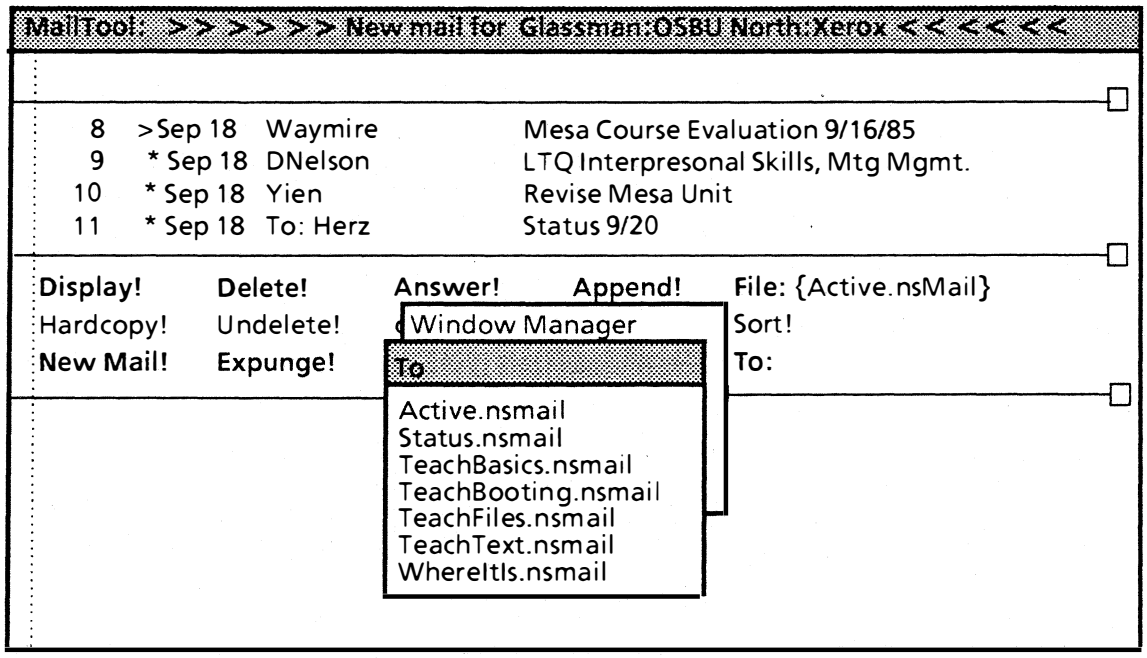


Figure 4.5: A menu of mail files

Reading another mail file

Now that you have a message in the file temp.nsmail, you need to go read temp.nsmail so that you can read the message. You should already know how to read another mail file: just chord over File: and select temp.nsmail from the menu that appears. Go read temp.nsmail now to see that the message is really there, and then return to this mail file. (To return to this mail file, just select TeachMailSystem.nsmail from the menu.)

The long way

Choosing the name of a mail file from the File: field is the accelerated way of switching between mail files. However, if you have a mail file that does not have the extension .nsmail, or a mail file that is for some reason not listed in the enumeration of mail files, you can explicitly specify the name

Bring up your options window by invoking Options! The MailFile: field in this window contains the name of the mail file that is currently being read. The list of mail files is also available here; chord to see that this is so In addition to selecting mail files from the menu, however, you can manually edit the mail file listed here. To do so, just click Adjust over the colon following the MailFile:. This accelerator will delete the current contents of the field and set a type-in point. Type in the name of the mail file that you wish to read, and invoke Apply!.

To get to one mail file from another you can either edit this field or choose the name from the menu. In general, you will want to use the menu, but in special cases you may need to edit this entry manually.

MailTool and the Search Path

Reasonably enough, only mail files that are on the current search path will appear on the Mail File menu. If you want to read a mail file that is in a directory that isn't currently on the search path, you can use the "long method" described in the previous message to view the mail file without changing your search path. (That is, you can enter the fully-qualified name of the mail file in the File: field in the Options window.)

If you have two mail files with the same name on different directories, and both directories are on the search path, the name will appear twice on your File: menu. If you select either instance from the menu, MailTool will load the first instance that it finds; that is, the instance in the directory closest to the top of your search path. Thus, the only way to see the other, identically-named, mail file is to open the Options window and enter the fully-qualified name. Generally, you should avoid having two identically named mail files on different directories: it can cause a great deal of confusion.

AutoDisplay and DisplayOnNewMail

Open the Options Window again (by invoking Options!), and find the word "AutoDisplay".

AutoDisplay controls whether or not the next message will be displayed when you delete a message. If AutoDisplay is video-inverted (on), then invoking Delete causes MailTool to behave as if you had invoked Delete! followed by Display!. In other words, MailTool will automatically display the next message whenever you delete a message. If AutoDisplay is not turned on, then invoking Delete! deletes the current message from the mail file but leaves it displayed in the message subwindow.

When the DisplayOnNewMail option is turned on, retrieving your new mail will automatically display the first new message. Otherwise, when the option is off, the Table of Contents will scroll to the first new message, but it won't be automatically displayed.

The remaining options in the Options window are Hardcopy options. These options are discussed in the printing tutorial, which lets you practice the Hardcopy command.

Answer! and Forward!

Find the Answer! and Forward! commands in your command subwindow.

Invoking "Answer!" allows you to respond to a particular message. Invoking Answer! creates a new Mail Send window, and automatically fills in the "To:" field with the name(s) of the sender(s) of the specified message, or the names in the "Reply-To:" field (if one exists). It also fills in the "Subject:" field and the "In-reply-to:" field. Try "Answer!" to observe its effect. This command operates on the message that is currently displayed in your MailTool window. Thus, if you want to

answer a message, it isn't enough to just select the title in the Table of Contents; you have to display the message first.

"Forward" automatically copies the currently selected message(s) into the message body of a new Mail Send window. Try "Forward" now. Note that you must fill in the "To:" and "cc:" fields, and optionally provide a covering message. Unlike Answer, Forward applies to the message which is currently selected; it is possible to select multiple messages in the Table of Contents, invoke Forward!, and have all the messages copied into one message body. One good time to use the Forward command is when you receive a message that was not intended for you; you should forward it back to the sender so that he realizes that his message was not delivered properly.

Append!

The Append! command allows you to insert text from regular files into your mail file. Thus, you can select text anywhere on your screen and invoke Append!; the selection will be inserted at the end of the mail file, complete with a header and Table of Contents entry, just like all the other messages that you retrieved from the mail service.

The Hardcopy command

The Hardcopy! command is used to print mail messages.

The lower half of your Options window contains options to control the appearance of mail messages that you send to the printer. For example, you can choose the font in which your messages will be printed, and the way that the text will be oriented on the page. The tutorial called TeachPrinting.nsmail discusses the use of these options; for now, you should just leave most of them as they are. (If you have already done the printing tutorial, you should be all set.)

However, there is a Printer: option that allows you to specify the printer to which your output will be sent, and you need to make sure that this item is correct before you can try the Hardcopy! command. If you don't know the name of the printer nearest you, you should check with someone nearby who can help you.

Now fill in the name of the printer in the Printer: field. When you have done so, invoke Apply! to put your changes into effect and destroy the Options window. (Invoking Apply! records changes; invoking Abort! ignores all changes and destroys the window.)

Try using the Hardcopy command on some of your mail messages. The command will apply to any currently selected messages, and not necessarily to the message which is displayed. In other words, if you have a message selected in the Table of Contents that is different from the message currently displayed on your screen, the command will apply to the selected message and not to the displayed one.

Maintain

Maintain is a program that allows you to access the Clearinghouse database. You can use Maintain to inspect and modify information about users and distribution lists. When you run Maintain from the Executive, it will appear as illustrated in Figure 4.6.

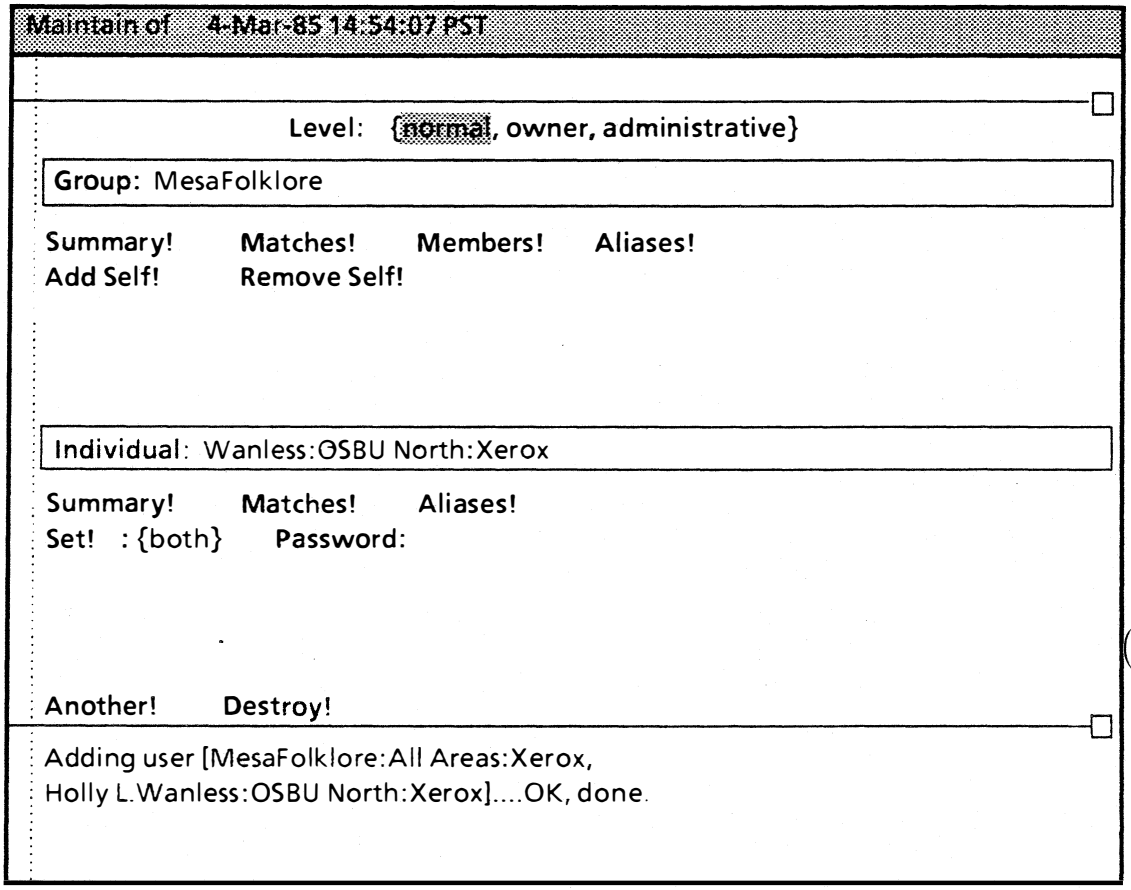


Figure 4.6: Maintain tool window (normal level)

The first thing that you have to set is the **Level** enumerated item, which governs the available commands. There are three possible levels: normal, owner, or administrative. All of the commands available at the normal level are also available at the other levels; owner and administrative simply provide some additional commands not present at the normal level.

In general, the top half of the form subwindow contains items used to manipulate groups, and the bottom half allows you to change items associated with a user. Thus, for example, you can use Maintain to set your password, add yourself to a distribution list, find out the possible aliases for your name, list the members of a particular distribution list, find out the owner of a distribution list (administrative level) and various other information related to the database. For complete documentation on the Maintain commands, see the Mail Tools chapter of the XDE User's Guide.

The Mail File Scavenger

Because a mail file is just a text file, you can edit the content of the messages in your file. To do this, all you have to do is deactivate MailTool, and load the file into a Source window. However, each message in a mail file has a header in a special format to allow MailTool to read the mail file. Among other things, this header includes a count of the number of characters in the message. Thus, if you edit a mail file, you will have to ensure that the header information is correct or MailTool will not be able to read the mail file.

If you have a mail file that you have edited, or that has been damaged in some other way, you can run a tool called MailFileScavenger.bcd to restore the structure of the file. The MailFileScavenger is simple to use; you should read the appropriate chapter in the XDE User's Guide when you want to use this tool.

Because it is fairly rare to have a damaged mail file, you will probably not use this tool very often. However, if you try to read a mail file in MailTool and get a message that the file is "not a properly formatted mail file", you will know to use the MailFileScavenger.

What now?

You should now be able to easily send, receive and print mail in the Xerox Development Environment If you have any questions, you should consult the documentation in the XDE User's Guide. When you feel comfortable with the mail system, you are ready to go on to another tutorial module.

This section discusses how to print local and remote files, and how to change the font and format of the printed material to suit your preferences.

Before you start this section, you will need to know the name and location of your local printer.

Changing your default printer

Before using the Print program, you should set up your user.cm with some default options, such as the name of your favorite printer. The printer specified in your user.cm file is the printer to which your material will usually be sent; you can override it for any individual print command.

Load your user.cm file into a Source Window and invoke Edit. Now scroll through the file until you find a section labelled [Hardcopy]. You should edit your user.cm file so that it has a default printer name. The entry for a default printer may be somewhat unintuitive; it is Interpress, which is a kind of printer. If the name contains spaces, it must be quoted. For example, if you wanted to use a printer called Nevermore, your user.cm might look like this

```
[Hardcopy]
   Interpress: "Nevermore·OSBU North:Xerox"
```

If you are in the same domain as your printer, you do not need to include the domain and organization. For example, if you are in the domain OSBU North, then you could put just Nevermore as the name of your printer. However, if you are in a different domain or organization from your printer (which isn't normally the case), you will have to fully specify the name of the printer.

Other user.cm entries

There are several other aspects of your hardcopy that you can control with user.cm entries:

The **PrintedBy:** entry determines the name that appears on the cover sheet at the printer. If you do not specify anything here, the name that is currently logged in will be printed on the cover sheet. You should put something appropriate in this space so that your printouts are identifiable.

An **Orientation:** entry specifies whether you want your printout to be $8\frac{1}{2}$ by 11 or 11 by $8\frac{1}{2}$. *Portrait* specifies standard orientation ($8\frac{1}{2}$ by 11), *landscape* specifies 11 by $8\frac{1}{2}$.

A **Font:** entry specifies the default font. There should be a font sampler near your printer that illustrates the different fonts available for that printer. You can also have separate **PortraitFont:** and **LandscapeFont:** entries if you like.

For example, your complete hardcopy section might look like this:

```
[Hardcopy]
   PrintedBy: Groucho
   Interpress: Nevermore
   Orientation: portrait
   PortraitFont: Gacha10
   LandscapeFont: Gacha8
```

When you have finished editing your user.cm, invoke Save

Printing a file

Once you have edited your user.cm, you are ready to send something to your default printer.

Type *Print <filename>* on a command line in the Executive. If you want to print more than one file, separate the list of files with spaces. You can also use * as an expansion character to denote a group of files, as you can elsewhere in the environment. For example, if you wanted to print your user cm and a file called junk.txt, your Executive would look something like this:

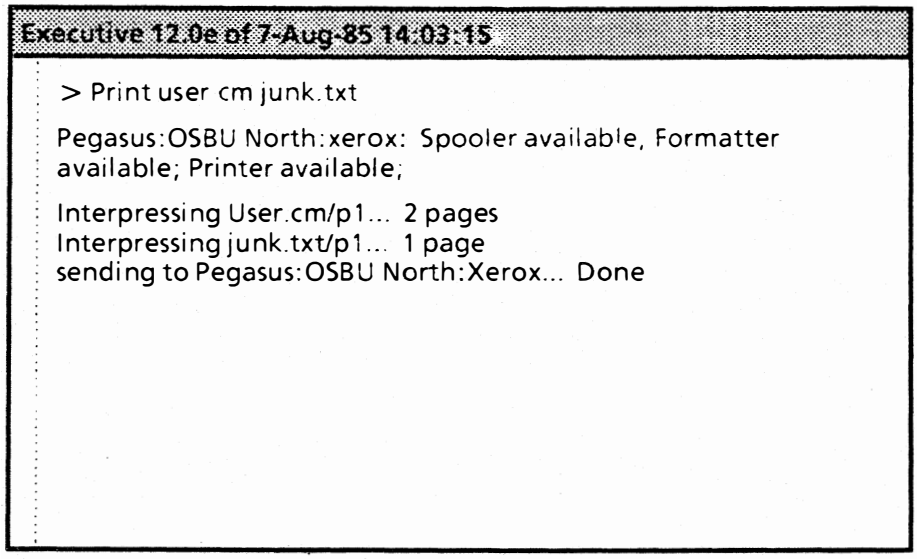


Figure 5.1: Printing a file

When you type a carriage return, your material will be formatted by the Print program into an .interpress file, and then sent to the printer. (Interpress printers only accept files in interpress format; this format is essentially a program that tells the printer what to print and how to print it.)

You will now need to go pick up your hardcopy. Your print-out will probably still be in the printer; it will have a cover sheet with your name, the date and time, and the name of the

printed material on it. If there is other printed material in the printer when you pick up your own, you should remove it from the printer and file it in the shelves near the printer.

$$$ and remote files

If you would like to print only a small part of a large file, or text which is not part of a file, you can type $$$ instead of a file name on your command line. This will print the current selection.

For this convention to work, the text that you want to print must be the current selection on your screen. Thus, you should select the desired text, and set a type-in point in your Executive **using Adjust rather than Point**. (If you try to use Point to enter the Print command, you will lose the selection of text that you are trying to print.) Experiment with printing some selections of text until you get the hang of it.

You can also print a remote file directly (without retrieving it onto your local disk). To do so, just type the fully qualified name. Again, if the name contains spaces, you will have to surround it with double quote marks. For example,

Print "[Hal:]<Hal Training>Mesa Unit>General>User.cm"

Printing switches

The file that you just sent to the printer will have been printed with the default characteristics specified in your user.cm file. You can use printing switches to change these values for a particular instance of the command without changing their permanent setting.

For example, suppose that your standard printer is down for some reason, and that you want to send a document to an alternative printer. You don't want to change the entry in your user.cm file, since you will want to return to the original printer as soon as it is up. Instead, you can use a printing switch to override the default printer for a particular print command.

A sample command line might be "Print Myfile SeaBiscuit/h". The /h switch, which stands for "host", specifies that the document should be sent to the printer named SeaBiscuit instead of to the printer specified in your user.cm.

A printing switch can be used either locally or globally. When used globally, a switch applies to all files being printed; a global switch is placed at the beginning of the command line. A local switch, on the other hand, applies to only one file; it follows the file name that it affects. Thus, the switch used above is a local switch. The command line "Print Seabiscuit/h MyFile" accomplishes the same thing with a global switch. Thus, when you are sending several files to the printer at the same time, you have the choice as to whether they share the same properties or each have different switches.

More Printing switches

Here is a list of some other print switches that you might find useful:

<font>/f Changes the font to <font> for the following files. The default is Gacha8 in portrait; Gacha6 in landscape.

/c<n> Sets number of copies to <n> (default 1).

/t<n> Sets the tab width to <n> spaces (default 8).

Examples:

```
Print Silly

Print Silly/l

Print /l Silly1 Silly2

Print Silly1 Silly2/c3 /p TimesRoman10BI/f Silly3

Print $$$
```

For example, the fourth example could be read "Print the file Silly1 with the default switches; print Silly2 in in triplicate; set the default orientation for the rest of the command line to portrait; set the default font to TimesRoman, 10-point, bold, italic; Print Silly3."

There is a complete list of the switches available, along with detailed descriptions of their effects, in the XDE User's Guide

What now?

You should now be able to obtain a printed copy of anything that you can see on your screen, and anything stored on a remote server, and how to have it look just the way you want it. If you have any questions, consult the documentation in the XDE User's Guide.

6. BOOTING

To **boot** a machine is to load and start a system on the machine. Booting assumes that main memory has undefined contents.

This module discusses how to boot your machine. The material is purely "how-to"; it includes only the information that you will need in order to restart your workstation when you crash, but it does not include detail on what is actually going on.

In Xerox terminology, a hard disk ("physical disk") is divided into several "logical volumes." There can be up to 10 logical volumes on a physical volume, although there are not usually nearly so many. Obviously, if you partition your disk into more pieces, you get less space on each of the pieces.

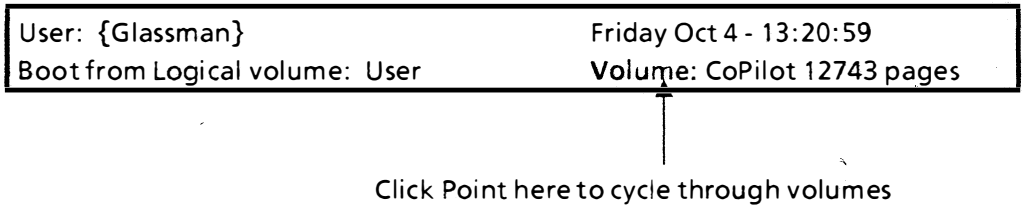
There are three principle systems that you can have on your workstation: ViewPoint, XDE, and InterLisp. Most machines will have XDE and ViewPoint or XDE and InterLisp; unless you have a very large disk (80 Mbytes or more), you probably don't have room for all three.

Generally, each logical volume has a bootfile on it. The bootfile is the file that gains control when the volume is booted. For example, there is a CoPilot bootfile on the logical volume CoPilot; when you boot the CoPilot volume, this bootfile gains control and sets up the environment you know as CoPilot. Not all volumes have a bootfile, however; some of the volumes are used only for data files. If you try to boot a logical volume that does not have a bootfile installed, you will have problems.

Booting a volume will restore it to a pristine state, and will thus cure most software problems. Booting will not destroy any files or mail messages, but any text not saved in a file will be lost.

Booting from the Herald window

Since here are many different ways to set up a machine, you should start by getting an idea of how your machine is set up. To find out what logical volumes are on your disk, and how much space is on each of them, move your mouse to the Herald Window and click your mouse over the word Volume:, illustrated in Figure 6.1.



This field gives the number of free pages on your CoPilot volume. Clicking Point over this field cycles through the logical volumes on the disk, and displays the number of free pages on each. Use Point to cycle through the volumes on your disk, so that you have some idea of the names of the volumes and how large they are. Some of the names will be familiar, such as CoPilot; others may not yet be familiar.

To boot another volume from CoPilot, you use the Boot From: menu, available from the Herald Window. Move your cursor into the Herald and chord to bring up the Boot From: menu. This menu contains the names of the volumes on your workstation, such as Scavenger, User, CoPilot and Tajo, as illustrated in Figure 6.2. This menu can be used to boot any of the volumes listed. For example, to reboot your CoPilot volume, you would select CoPilot from this list. Try it, and notice that the graphic mouse appears, asking for confirmation of the boot command. You can either abort the command with Adjust or confirm in with Point, depending on whether or not you really want to reboot CoPilot.

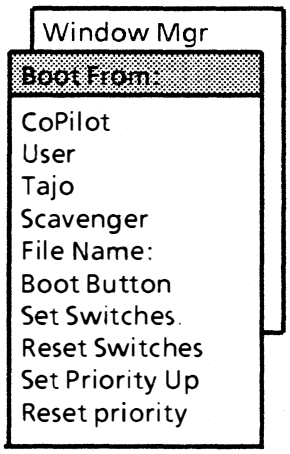


Figure 6.2: Boot From: menu

Setting boot switches in the Herald

The Herald window can also be used to set boot switches for an individual boot. Boot switches are used to control aspects of the initialization of your volumes. Some boot switches are quite important; others are optional In general, there are no required boot switches for CoPilot, but there are several that you must use in order to boot ViewPoint. The person who set up your machine should have set the appropriate default boot switches, so you don't need to worry about them now. However, you should be aware that you can set boot switches from this menu if you want to.

As an example of using the Herald window to set your boot switches, type the letter N somewhere on your CoPilot screen and select it. Now invoke Set Switches from the Boot From menu, and the boot switches will be set to N. This switch will be used the next time you boot from the Herald Window,

regardless of which volume you boot. (The N switch means "don't read the user.cm during booting.")

Reset Switches can be used to restore the boot switches to their default value after you have changed them with Set Switches. Invoke this command. You will not get any feedback from the system when you invoke either Set Switches: or Reset Switches; do not expect the currently applied switches to magically appear anywhere on your screen.

For the uses of the other commands in the Boot From: menu, refer to the chapter on the Herald Window in your *XDE User's Guide*.

Booting from Othello

Othello is a utility from which you can boot any volume. Othello is used for various disk management tasks, such as configuring the disk, installing boot files, and setting default boot switches. For a complete description of the kinds of things that you can do from Othello, see Appendix A of the XDE User's Guide. For now, however, you only need to know how to reach Othello and how to get from Othello back to CoPilot.

Depending on how your machine is set up, you may have a logical volume with the Othello boot file installed, or you may have to boot Othello from the net. We describe how to boot from the net in the next section. If you have an Othello volume on your machine, you can practice the things in this section now. If you don't, you will have to come back to this section after you have learned how to boot Othello from the net.

If you have an Othello volume on your machine, boot it now from the Herald Window. Othello will look something like the illustration in Figure 6.3.

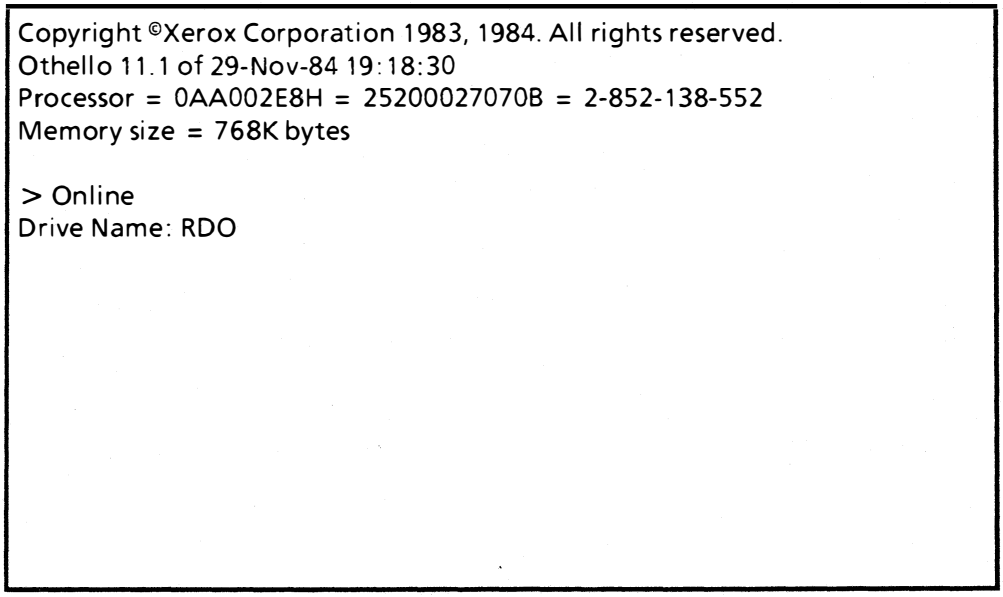


Figure 6.3: Othello

Othello occupies the entire screen; you do not have multiple windows and multiple tools. When you reach Othello, you will see ".Drive Name: RDO". The first thing that you have to do is enter a carriage return to prepare Othello for a command. When you have done so, > will appear as your command prompt.

To get a better idea of how your machine is set up, try invoking the Describe Physical Volumes command. Othello accepts abbreviated commands, so you can type out the above command, or you can just type Desc followed by a return. This command will show you the number of logical volumes that your disk has, the size of each volume, and the name of the bootfile that is installed on it, if any. If you are curious about other Othello commands, you can type a question mark to see what is available.

When you are ready to go back to CoPilot, you need to invoke the Boot command. Type Boot (or an abbreviation), followed by a carriage return. Othello will then ask you for the name of the logical volume that you wish to boot. Type in CoPilot, and carriage return again. (You can't abbreviate the name of the logical volume.)

You will next be prompted for boot switches. You can either enter appropriate boot switches here, or just hit another carriage return to boot with the default switches. Unless you are booting ViewPoint, you can boot without any switches. If you are interested, available boot switches are documented in your XDE User's Guide.

Remember that if you are ever unsure of the necessary command when working in Othello, typing a question mark will provide a list of the available commands. Once you have given an initial command, Othello will provide you with prompts for any additional information that it requires. You need only remember to confirm each of your commands with carriage returns.

Boot buttons

Sometimes you will have to boot your machine with a "hard boot" of the entire machine, rather than just doing a software boot of a particular volume To do this, you will need your boot buttons. The boot buttons are the two buttons beside the maintenance panel on your machine. Locate them. (Note: there is a small door (more like a flap) that covers the maintenance panel. If you do not see the boot buttons, then your door is closed; you will have to find this door and pull it down.) The left button is labelled "B RESET" and the right button is labelled "ALT B."

To do a hardware boot (don't do it yet), press in both buttons, and then release the left one. If you continue to hold in the right button, the maintenance panel numbers should cycle gradually from 0000 to 0010. Each of these numbers represents a different kind of boot. The mapping from number to type of boot varies occasionally from release to release and location to location, so you should check with someone local to find out the mapping. The various boots are referred to as 0 boot, 1 boot, and the like.

When the maintenance panel codes reach the number of the boot that you want, you release the right button, and the boot will be performed. If you make a mistake and pass the number that you want, don't worry about it; the numbers will continue to cycle and you will get another chance.

A 0 boot is the basic boot. It will run hardware diagnostics to make sure your disk is all right, and then it will boot a logical volume. Which logical volume is booted after a hard boot is something that you can specify in Othello; it is usually set to be either CoPilot or Othello.

The numbers that display on the maintenance panel during the boot signify that diagnostics are being executed. If there is a difficulty with the boot, these numbers will stop at a number that indicates the nature of the problem. If this should happen during a boot, you will need help.

990 is the normal readout for XDE volumes (CoPilot, Othello, and Tajo); 8000 is the normal readout for ViewPoint. Any other number usually indicates that something is wrong. (Booting does take a few minutes, so don't panic if a code remains for a little while.)

Alternative boots

As mentioned above, each of the numbers from 0 to 9 represents a different kind of boot. These are called "alternate" boots, since the 0 boot is considered the standard boot.

The most common alternate boot is the 1 boot, which is just like the 0 boot except that it does not run hardware diagnostics. Thus, you can use this boot when you want to save time by not running the diagnostics. However, you should not always use a 1 boot; the diagnostics performed in the standard boot are a valuable method of checking to make sure that your machine is healthy. For now, since there is nothing wrong with your machine (hopefully), try doing a 1 boot. If this boots Othello, you will then have to boot CoPilot from Othello. If it boots CoPilot, you're all set.

Another important alternate boot is booting Othello from the net. Othello is usually available from a 6 boot; check with someone to find out if this is correct. To reach Othello from the net, just do the appropriate boot, and you will be in Othello. For now, you are not likely to need to do any of the other alternate boots. If you don't have an Othello volume, try booting Othello from the net and then booting CoPilot from Othello, as described earlier.

When not to boot: moving between booted volumes

Although you can always use booting as a means to get from one volume to another, you will not always want to do this. There is a shorthand way of calling the "debugger" volume which allows you to easily move from one previously booted volume to another. The advantage of using this method is that you will not have to reconstruct your screen each time that you

reach a volume; you will be able to leave your desktop as is, much as you do when you invoke DMT.

If you are not familiar with the structure of volumes, you need to know which volumes are serving as debuggers for other volumes. Briefly, CoPilot can serve as a debugger for either the Tajo volume or the User volume, but not for both volumes simultaneously. (To find out more about why some volumes are "debugger" volumes, you will need to read the debugger tutorial.)

Typically, XDE software is installed on Tajo and ViewPoint software is installed on User. You can establish either a Tajo-CoPilot "client volume-debugger volume" relationship or a User-CoPilot client volume-debugger volume relationship, depending on your immediate needs. This relationship can be changed at will.

If you want CoPilot to serve as a debugger for Tajo (as you will for the remaining tutorials), you will boot Tajo from the CoPilot Herald window (don't do it yet.) This will establish a Tajo-CoPilot client-debugger relationship. To return to CoPilot from Tajo once Tajo has booted, simply type SHIFT-STOP (hold down the SHIFT and STOP keys simultaneously.)

Try this now: boot Tajo from the Herald window (confirm with Point), and then interrupt (SHIFT-STOP) back to CoPilot after Tajo has booted. As you can see, this does not affect the state of your CoPilot volume. In particular, you do not have to reconstruct the layout or state of any of your tools.

To reach your previously booted Tajo volume from CoPilot wihout rebooting, you will need to have a debug log on your screen. In CoPilot, the name for this window in the tiny state is "CoPilot 12.0" Find this window now and activate it. To use this window to reach Tajo, you need to type "P" (for Proceed,) and follow with a carriage return. (Note that you must type only P, and not more of the word.) You can Proceed into a volume once you have booted it and SHIFT-STOPped out of it.

Type a "p" in the debug log window followed by a carriage return. This will return you to Tajo; once you are in Tajo, SHIFT-STOP back to CoPilot.

You establish a User-Copilot relationship in exactly the same way: boot User from the Herald window to invoke the ViewPoint software and SHIFT-STOP back to CoPilot once User has booted. (Booting User takes quite a while, so you might not want to experiment with it now.)

Once you have booted a client volume (either Tajo or User) from CoPilot, you can reach the client volume by proceeding. If you wish to change the client volume, you need to boot the new client volume from the Herald window.

The above methods can be used to move about between volumes during your normal operation. You do not normally need to boot the new volume each time that you would like to go from one volume to another.

For more information

You should now feel comfortable with booting a volume from the herald window of a particular volume or from Othello, and with booting your machine from boot buttons. You should also have a general idea of when you will need to boot a volume and when you can just call the debugger volume.

Hopefully, you now have enough information to enable you to survive the first few occasions on which you will need to boot, but remember that there is more information contained in the XDE User's Guide which you will later need to know.

7. COMPILE-BIND-RUN

This tutorial will introduce you to the steps involved in turning a newly written source file into a working program. It is purely "how-to"; that is, it will teach you the mechanics of using programming tools, but will not teach you how to actually write programs in Mesa. When you are through with this tutorial, you probably won't be ready to write your first program in the Xerox Development Environment, but you should have a general idea of the steps and tools involved, so that you will be ready to go as soon as you learn about the Mesa language.

Before starting the next lesson, make sure that your Herald window and your Executive are active. You will also need the files ToolFactorialDefs.bcd, ToolFactorialImplA.mesa ToolFactorialImplB.mesa, ToolFactorial.config, Put.bcd, and Heap.bcd on your local disk. Use your File Tool or your Executive to verify that you have these files and that they are on your search path. If you are missing one or more of them, you will need to ask for assistance.

Compiling

The Mesa compiler does not have a window of its own Rather, it runs from the Executive, or from a tool called Command Central. Running the Compiler from the Executive is just like running any other tool from the Executive: you just type "compiler foo". You can compile a group of files by separating the names with spaces, as in "compile MyProgram MyOtherProgram Fred". Capitalization of the file name does not matter, and you can abbreviate "compiler" if you like. You don't have to include the .mesa extension when you type the file name.

The Mesa compiler is a six pass compiler. Try compiling ToolFactorialImplA.mesa from the Executive. While the compiler is running, look at your Herald window. A small cube appears during compilation. This cube has one side for each pass of the compiler. As the compiler enters a new pass, the cube turns to show the corresponding face In the Executive window, the passes are represented by dots following the command. When errors are found, the number of dots tells you the number of the pass during which the errors were found.

In this case, the compilation will be successful, and you will see a message that gives the number of lines of code, and the number of seconds that the compilation took. When there are errors, you will see how many errors there were, and how long the compiler ran until it encountered them. To see the full compiler log (i.e., any error messages), you will have to load the file compiler log into an Empty window.

The output of the compiler is a file called ToolFactorialImplA.bcd.

Compiling from CommandCentral

You will often run the Compiler from Command Central instead of from the Executive. Command Central is a tool specifically designed to simplify the processes of compiling, binding, and running a program. Bring up Command Central on your screen. The command subwindow has both a Compile! command and a Compile: field, as illustrated in Figure 7.1. To compile a file (or files) from Command Central, just fill in the name of the file(s) to be compiled in the Compile: field, and invoke Compile!. Compile ToolFactorialImplB.mesa. (You don't have to include the .mesa extension.)

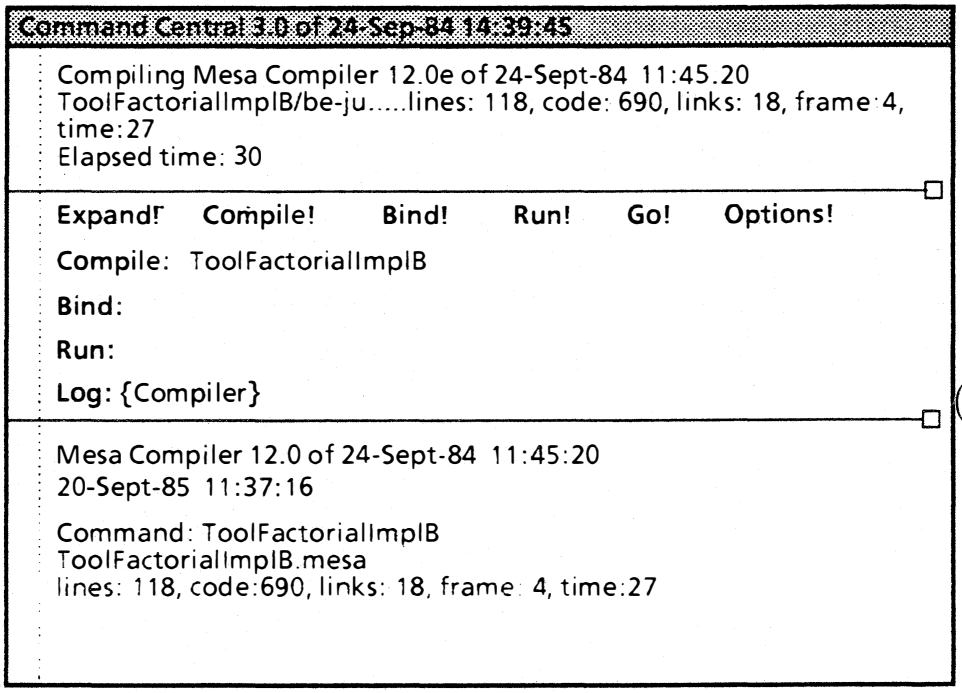


Figure 7.1: Command Central tool window

When the compilation has finished, the complete log file will be displayed in the bottom subwindow of the Command Central window. For your convenience, this file is automatically loaded into the Command Central text subwindow when the Compiler has finished.

When you compile from Command Central, the cube will appear in the Herald Window just as it did when you compiled from the Executive. Other information, such as the number of passes completed, and the code size (for successful compilations) will appear in the message subwindow of the Command Central window.

Compiler switches

There are a number of compiler switches that allow you to specify details of compiler operation. For example, the /b switch tells the Compiler to turn on bounds checking; /e tells it

to include error messages; /u to give warnings about uninitialized variables. The complete list of compiler switches and their default values is in the Compiler chapter of your XDE User's Guide.

You can set default compiler switches in your user.cm file. The entry CompilerSwitches: can be placed in either your [Executive] section or your [CommandCentral] section, or both. (If you do not have a section for these tools in your user.cm, you can add one.) For now, we suggest that you set your switches to beu-j (the j switch activates cross jumping; a - preceding a switch turns it off).

As with other tools, you can override the default values in your user.cm if you like. To specify a different set of switches when running the Compiler from the Executive, just type the list of switches (preceded by a /) on the command line. Compiler switches take the same form as other Executive switches: that is, they can be either global or local. For example, to turn off bounds checking and uninitialized variable checking, you would use the command "Compile/ b-u Myprogram" (global switches) or, equivalently, "Compile MyProgram/-b-u" (local switches).

To set compiler switches in Command Central, just invoke Options! to bring up the options window, and edit the switches that are listed under Compiler Switches:. These switches will override the settings in your user.cm; they will be reset to the default user.cm values each time that you deactivate and reactivate the tool (and each time you boot your system).

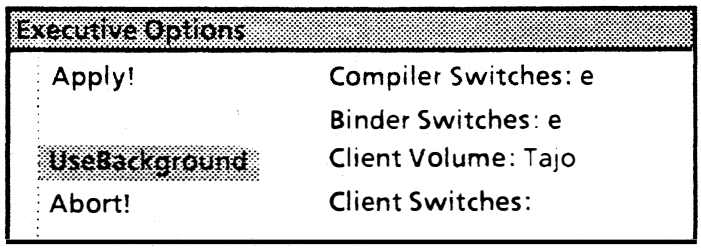


Figure 7.2: Command Central options

Binding

Compiling a program generates an executable object file (if there are no errors). However, in most cases you will not immediately run the object files generated by the compiler. Instead, you will usually need to use the Binder.

The Binder is a tool that groups individual object files together into a single large object file. (The Binder is thus similar to a linker.) Both the Compiler and the Binder produce object files; the Compiler produces simple object files from source files, and the Binder produces complex object files from simple ones.

For example, if you want to use procedures or types that are found in another module, you will need to use the Binder to associate your files with the files from which you would like to use symbols. The Binder also allows you to group files together

in a specific order, so that you can load a single file instead of a large group of files.

The Binder takes as input a file with the extension .config (short for configuration). A .config file is essentially a list of individual object files that you want to group together. You will have to wait until you take the Mesa Course or until you attend a lecture on the Mesa language before you are ready to write a .config file; for now, you will just have to settle for learning what to do with a config file once you have one.

Using the Binder

Running the Binder is much like running the Compiler. To use the Binder from the Executive window, just type "bind foo". You don't need to include the .config extension; this will be assumed. Similarly, to run the Binder from Command Central, fill in the name of the file in the Bind: field, and invoke the Bind! command. Bind ToolFactorial.config now, either from the Executive or from Command Central. This will produce an object file called ToolFactorial.bcd.

The file binder.log contains the log file for the most recent binder command. This file is automatically loaded into the Command Central text subwindow (as was compiler.log); if you bind from the Executive, you will have to load it manually.

You can also set Binder switches, just as you did for the Compiler. The default Binder switches are set in your user.cm file; the complete list of available switches can be found in the Binder chapter of your XDE User's Guide

File naming conventions

Since the Compiler and the Binder both produce files with the .bcd extension, you need to make sure that all the files making up your program have distinct names. For example, if you compile a file called ToolFactorial.mesa, you will get a file called ToolFactorial.bcd. If you then try to bind a file called ToolFactorial.config, you would run into trouble, because the binder would try to create a file called ToolFactorial.bcd and it would find that such a file already existed. If you make this mistake, the Binder will overwrite the .bcd file produced by the Compiler, and you will run into problems

The best way to avoid mistakes is to adopt a consistent naming convention. Thus, program modules that implement a particular interface should be named MumbleImpl.mesa, or MumbleImplA.mesa and MumbleImplB mesa when there are multiple implementation files. Make sure that all modules, whether .mesa or .config, have distinct names, and you will be safe.

You also need to be sure that you include the extension (.mesa or .config) when you name a file.

Logical volumes

Once you have an object file, either a simple one produced by the Compiler or a more complex one produced by the Binder,

you are ready to run it and see if it works. Surprisingly enough, this step is probably the one in which the Xerox Development Environment differs most from the systems that you are accustomed to. The philosophy of the Xerox Development Environment is that programs should be written, compiled, and debugged in a separate logical volume from where you run them.

In the XDE, the word "volume" can mean either a physical volume or a logical volume. A physical volume corresponds to a storage device, typically your hard disk. A logical volume is usually a subset of a physical volume; there are usually several logical volumes on a single physical volume. (There can be no more than 10 logical volumes on a particular physical volume.) Each logical volume is largely protected from actions in other logical volumes. Thus, having a separate logical volume for running your test programs ensures that they have a fresh and clear test area in which to run.

This is called the "world-swap" approach to debugging. When you are ready to run a test program, you start from a "debugger" volume (the one in which you wrote and compiled the test program.) To run a test program, you must first copy the object version of that test program to the test volume. Once the file has been copied, the current contents of real memory are swapped out and written to a file, and then the test volume is booted (replacing the contents of real memory.) This is called a "world swap." Once you have run your test program sucessfully, you can world swap back to your debugger volume. When you swap from the client volume back to the debugger volume, the contents of the debugger volume will be read from the file, and your debugger volume will be in the state it was in before the original world swap.

You will learn more about world swapping in debugger.nsmail and in your documentation. For now, you just need to be familiar with the theory that you will be executing your programs in an entirely different logical volume from where you develop them.

The Tajo/User volume

There are two different possible target environments for new programs: Tajo and ViewPoint. This tutorial discusses Tajo as the target environment, but the process is essentially the same for ViewPoint.

If you don't have a Tajo bootfile installed on your machine, you won't be able to actually try running the test program. The Tajo bootfile is usually stored on a Tajo volume, but it is possible to have it on a different volume. To find out whether you have a Tajo volume, try chording in the Herald and looking at the BootFrom: menu. If there is a Tajo volume listed there, then you are all set. If not, you will either have to boot Othello and see if the there is a Tajo bootfile installed on another volume, or just ask the person who set up your machine whether there is a Tajo bootfile on it.

The rest of this tutorial refers to the test volume as Tajo, but you should substitute another volume name if appropriate. You should also edit your user.cm file to reflect the way your machine is set up. Load your user.cm into a file window, and

find the [Executive] section. Make sure that there is a ClientVolume: entry there; it should have the name of the volume where your Tajo bootfile is stored.

(If your user.cm file is wrong, the changes you make will not take effect until you next boot. If you need to change your ClientVolume: entry on the short term (without booting), you can just edit the ClientVolume: entry in the Command Central options window.)

Running the program

You will be running your programs in Tajo, but the object file that you want to run is on the CoPilot volume. Thus, you will have to copy the object file from CoPilot to Tajo, and then boot Tajo, and finally run the program from the Tajo Executive. If you have never booted Tajo, you will have to do so before you can run a program. (Boot Tajo from the Herald Window of your CoPilot volume, wait until you reach Tajo, then press SHIFT-STOP together to return to CoPilot.)

There are two ways to accomplish these steps: an easy way, and a hard way. In this lesson, you will learn the hard way; that is, you will perform all of the steps manually. You will rarely have to do this process manually, but it is important that you do it at least once so that you understand exactly what is happening The next lesson covers the easy way.

Type "open tajo/w" (or "open user/w") in your Executive The /w switch specifies that the volume is being opened for writing

Now copy the file onto the Tajo volume, with the Executive command line "Copy <Tajo>ToolFactorial.bcd ← ToolFactorial.bcd" After copying the file, type "close tajo" to close the Tajo volume.

Now that you have stored the file onto Tajo, you just have to get there and run the file. When you reach Tajo, you will need to load and run the file ToolFactorial from the Tajo Executive This program creates a tool window, with three fields in its form subwindow: Number = , Format. and CalculateFactorial! The number field is used to specify the input to the command; format is used to specify the format (base) of the answer, and CalculateFactorial! actually performs the command You can experiment with this test program if you like. (The input must be between 1 and 12.)

Remember: When you are through running the test program, you can return to CoPilot by pressing SHIFT-STOP.

Boot Tajo from the Herald window Boot from: menu now. Remember to run ToolFactorial from the Executive window when you get there.

CommandCentral's Run! command

The easy way to run a program is via Command Central. To run a program from Command Central, all you have to do is fill in the name of the program in the Run: field, and invoke the Run! command. When you invoke Run!, the object file is copied to the Tajo volume, Tajo is booted, and finally the

program is loaded from the Tajo Executive. Try filling in ToolFactorial in the Run: field and then invoking the Run! command, if you like.

There are also switches that you can specify in the Run: field. For example, if you enter filename/d in this field, the debugger will be called immediately after the tool is loaded (so that you can set breakpoints, or look at the state of things). There are several other switches that can appear here; check the Command Central chapter of your XDE User's Guide for a full list.

CommandCentral's Go! command

We have now covered how to compile, bind, and run a program in the Xerox Development Environment. The easy way to sequence these three steps is with Command Central's Go! command.

The Go! command invokes the Compile!, Bind! and Run! commands sequentially. Thus, if you fill in the name of a file or the Compile: field, the Bind: field, and the Run: field, and then invoke Go!, you will not have to do anything else until you end up in Tajo. Of course, if the compiler or the binder finds errors, the other commands will not be invoked. Thus, for example, if the compilation is not successful, the Go! command will abort after the compilation, and neither Bind! nor Run! will be invoked.

The debugger, CoPilot

You should now know how to compile, bind, and run a test program in the Xerox Development Environment. However, there's a catch to all this. As you know, no test program really runs perfectly the first time. So, to prepare you for the first time you really test a program, you should work through the next tutorial, called TeachDebugger.nsmail.

8. THE DEBUGGER: COPILOT

This tutorial introduces the Xerox Development Environment debugger, called CoPilot. The first dozen or so messages provide a brief overview of the debugger user interface and some debugging commands; the rest provide some hands-on debugging experience. You should read the early messages to get an idea of the available commands, and then refer back to the early material when you are doing the actual debugging.

As you do this tutorial, remember that CoPilot was designed for experienced users, and you shouldn't expect to be completely familiar with it when you finish this tutorial. Rather, we hope that you will get a general idea of what is available, and a good groundwork for learning more.

The World-swap approach to debugging

CoPilot resides on the CoPilot volume; your test programs will be run in Tajo. (There are two possible "target environments" for programs: Tajo and ViewPoint. This tutorial assumes that the target environment is Tajo.) This is called the "world-swap" approach to debugging. When you do a world swap from CoPilot to Tajo to test a program, the memory image of CoPilot is swapped out of main memory and the memory image of Tajo is swapped in. Thus, errors in your test program will not be able to harm the contents of CoPilot in any way.

CoPilot is a source level debugger: it allows you to debug with the same language constructs and concepts used in the original source program. Thus, when you are debugging, you can have the debugger window side by side with a file window that contains your source code. CoPilot provides many other debugging conveniences, such as making procedure calls from the debugger, assigning values to variables during program execution, and evaluating expressions.

To do the tutorial, start out in the CoPilot volume. Bring up your CoPilot window. (If it is on the Inactive menu or tiny it is called CoPilot 12.0; if it is active, the name in the herald is Debug.log.) You will also need the files MiscProcs.mesa, Heap.bcd, and String.bcd. Use your Executive or your File Tool to verify that you have these three files. If you don't have them, check with someone to find out where they are stored.

Entering the debugger

When you are running a program in the client world, there are several ways that you can reach the debugger, some of which are under your control and some of which are not. Each time you arrive in CoPilot from Tajo, a message will appear in the debug window that explains the reason for the world swap.

One way for you to return to the debugger is by an interrupt: you can interrupt the execution of your client program to return to the debugger and look at the runtime state. To manually interrupt from Tajo to Copilot, press SHIFT-STOP Your debug log will look like the one in Figure 8.1.

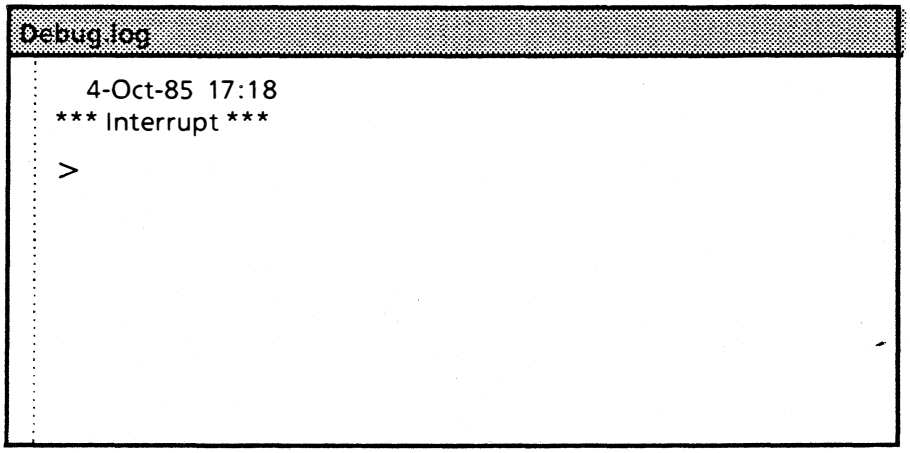


Figure 8.1: The debug log

You can also reach the debugger via a breakpoint. When a client program reaches a breakpoint, a world swap will be executed and you will be back in CoPilot.

Finally, you will be unceremoniously returned to the debugger when certain kinds of errors occur in your client program. We discuss this in more detail later

The debugger user interface

Set a type-in point in the debugger window and type a question mark. You will see a list of possible commands, as illustrated in Figure 8.2.

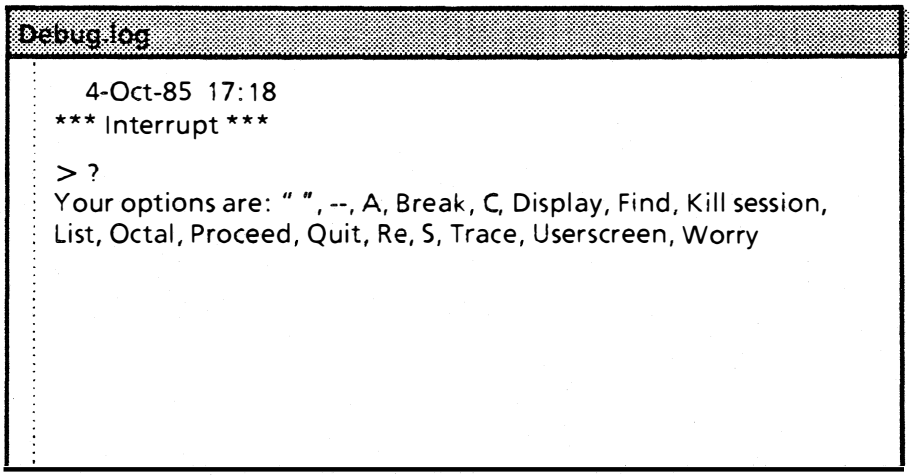


Figure 8.2: The debugger commands

The debugger works in command completion mode; that is, the debugger extends each input character to the maximal unique string that it specifies. Thus, you type only as much of a

command as is needed to uniquely identify it, no more and no less. For example, the list tells you that B stands for Break, but that there is more than one command that starts with the letter A. To see what those options are, type "A?" (Note that the letter is automatically capitalized, regardless of whether you type it in lower or upper case.)

The options that start with the letter A are AScii and ATtach. The capitalized letters tell you how much of the command you need to type. Thus, AS is ascii, and AT is attach. Try typing "at" after the command prompt. As soon as you enter the "t", the debugger fills in the rest of the command for you. If you try to type more, the debugger will interpret the remaining letters as the argument to Attach. To see that this is so, clear out the existing command line by hitting the DELETE key, and then try typing "att". The debugger will not recognize the second t, and will just kill the command.

To see the possible arguments to Attach, type "at?" on a command line. You will see that you can ATtach Condition, ATtach Keystrokes, or ATtach Symbols. (Don't worry; you aren't expected to recognize these commands yet.)

If you type something that you don't mean, you can always use the DELETE key to abort anything you have typed and return you to the top level command processor. As usual, when you give a command or provide input, you will have to enter a carriage return when you are finished

Debugger user interface: input and output conventions

Many of the debugger commands require an identifier (variable, module, etc.) as argument. When you type the name of an identifier to the debugger, you must capitalize it correctly (i.e., the way that it is declared). In Mesa, gamma, GamMa, and GAMMA are three different variables.

You can also control the format (octal or decimal) for debugger output. Invoke the Options command from the CoPilot menu (which you can see by Chording in the debugger window). This will bring up the CoPilot options iwndow, illustrated in Figure 8.3

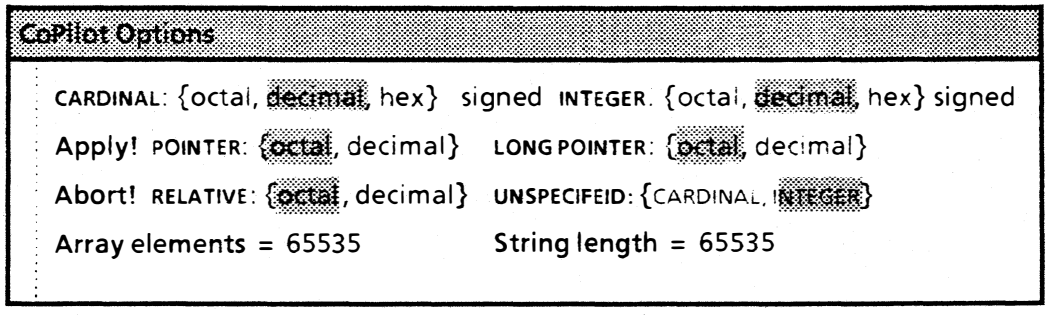


Figure 8.3: The debugger options window

The values in this window are all enumerated types: that is, you can see all of the options available to you, and the value

currently in effect is highlighted. You can change these values by selecting the desired format and invoking Apply!

The values that you specify in this window will apply to all output. However, you can force a particular interpretation of a number by suffixing it: a suffix of D or d forces decimal; b or B forces octal.

Commands for leaving the debugger

Before you get too thoroughly involved in experimenting with the debugger, here is a brief overview of the most "dangerous" debugger commands; that is, the commands that do relatively drastic things. You should know what these commands are so that you don't surprise yourself.

K (ill session) kills the current debugging session; in other words, it boots the physical volume. To prevent you from accidentally booting when you didn't really mean it, this command asks for confirmation before it actuall y executes.

Q (uit) aborts the process that was executing when you entered the debugger; this usually deletes that process. Quit also requires confirmation.

P (roceed) resumes execution of the client program. For example, suppose that you have set a breakpoint, and returned to CoPilot to look at the state of things. When you are ready to return to the client world and continue execution of your program, you can Proceed out of CoPilot into Tajo. If you haven't booted your client volume (Tajo), Proceed will not do anything.

W (orry) is used primarily for debugging the operating system. You don't need to worry about this for now.

Looking at the client world: Userscreen

If you are debugging in CoPilot, and decide that you need to take another look at something in Tajo, you don't have to Proceed back just to take a look at the screen. Instead, the debugger Userscreen (u) command will return you to Tajo for 20 seconds and allow you to look at the current state of the Tajo screen. You can return to the debugger before the 20 seconds are up by pressing STOP, or keep the Userscreen longer by holding STOP.

Inserting comments in the debug log

One of the possible debugger commands is --, which is used to insert a comment into the debugger log. When it sees --, the debugger will ignore all input until you enter a carriage return. Thus, you can insert comments amongst your debugging to serve as reminders of what you were doing.

Try inserting some comments in the debug log, if you like.

The current context

The current context is the domain for symbol lookup: it consists of the current frame and its corresponding process, module, and configuration. For example, when you ask for information on a variable, the debugger will look only in the current context; if it doesn't find the specified variable within the current context, it will look no further. Thus, by making sure that the current context is set to be the code that you are interested in, you can be sure that the debugger will find your variables and not some other variables that happen to have the same name.

Try typing "cu" for current context in the debugger window. This command will give you the name of a module and configuration, as well as the global and local frame number (G and L), and the Process State Block (PSB), as illustrated in Figure 8.4. (You aren't expected to know what all these things are right now: if you do know, that's great; if you don't, you will learn about them in your formal Mesa training, or you can look them up in the documentation.)

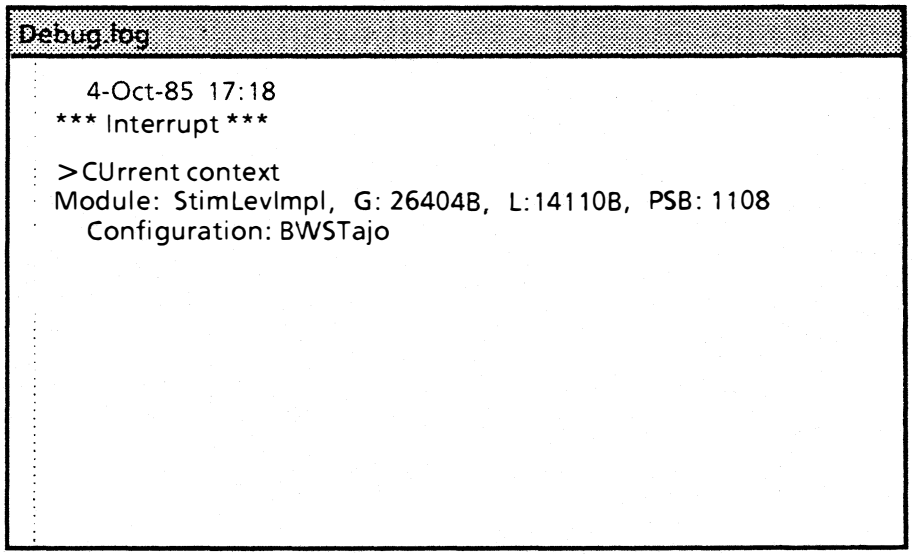


Figure 8.4: Current context

If the current context isn't the one that you are interested in, there are several commands that let you change it to something that you are interested in. In particular, SEt Configuration, SEt Module context, SEt Octal context, SEt Process context, and SEt Root configuration all allow you to control the current context. You can use any of these commands to change the context, depending on whether you are interested in a module, a process, or a configuration.

Try typing "lp" in the debug.log window, to produce a list of currently active processes. Pick any one, and use the SEt Process context command ("sep") to set the current context to that process. (Use the process number, as in "SEt Process context: 76B".) Now take another look at the current context. Your debug log should look something like the one in Figure 8.5.

(Note that we have abbreviated the list of processes; the actual list will be much longer than the one in the figure.)

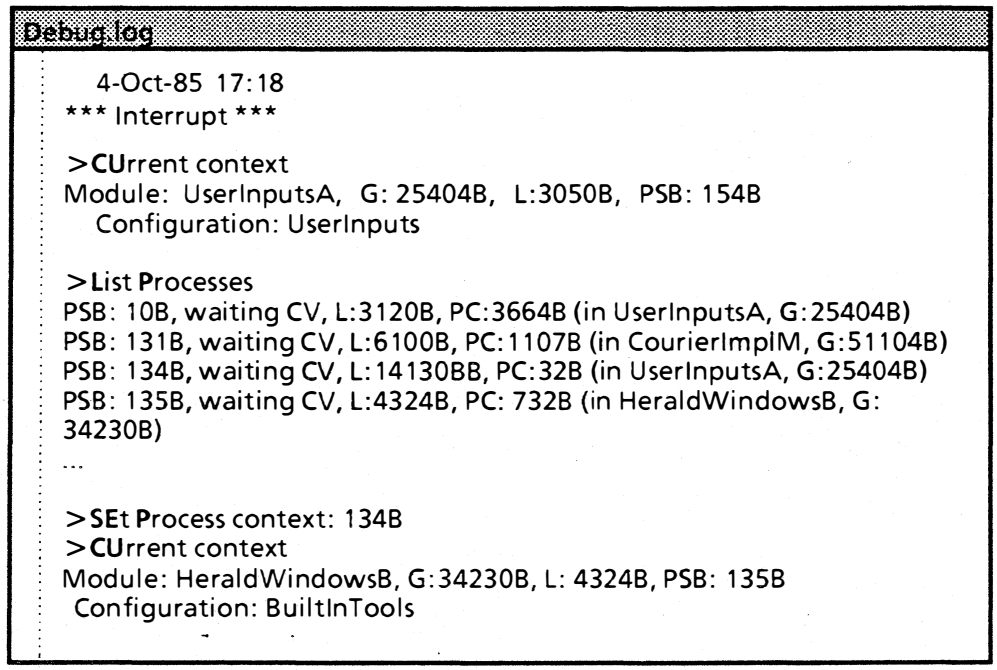


Figure 8.5: Setting the process context

Setting breakpoints

Like all good debuggers, CoPilot allows you to set breakpoints so that you can check up on your program. In the Xerox Development Environment, breakpoints apply to modules that are known within the current context. Thus, you cannot set a breakpoint in your module unless you have first set the current context. (You must also run your program before you can set breakpoints in it.)

You can set breakpoints either from the DebugOps menu or via commands in the CoPilot window. Through the CoPilot window, you can Break Xit procedure (bx), Break Entry procedure (be), Break All Xits module (bax), or Break All Entries module (bae). These commands allow you to set a breakpoint at entry or exit (or both) to a specified procedure, or to set breaks at entry or exit to all procedures in a specified module.

If you want to set a breakpoint on a location other than the entry or exit of a procedure, you will have to use the DebugOps menu of a file window. The DebugOps Break command uses the current selection to set a breakpoint at the closest statement enclosing the selection. Thus, if you load your source file into an Empty window, you can select the line where you would like to set a breakpoint and invoke Break, either from the DebugOps menu or from the EM Symbiote. (Of course, you don't have to recompile after using the source file to set a breakpoint. You don't have to recompile unless you actually edit the file.) Figure 8.6 shows what a source window would look like when you are setting a breakpoint.

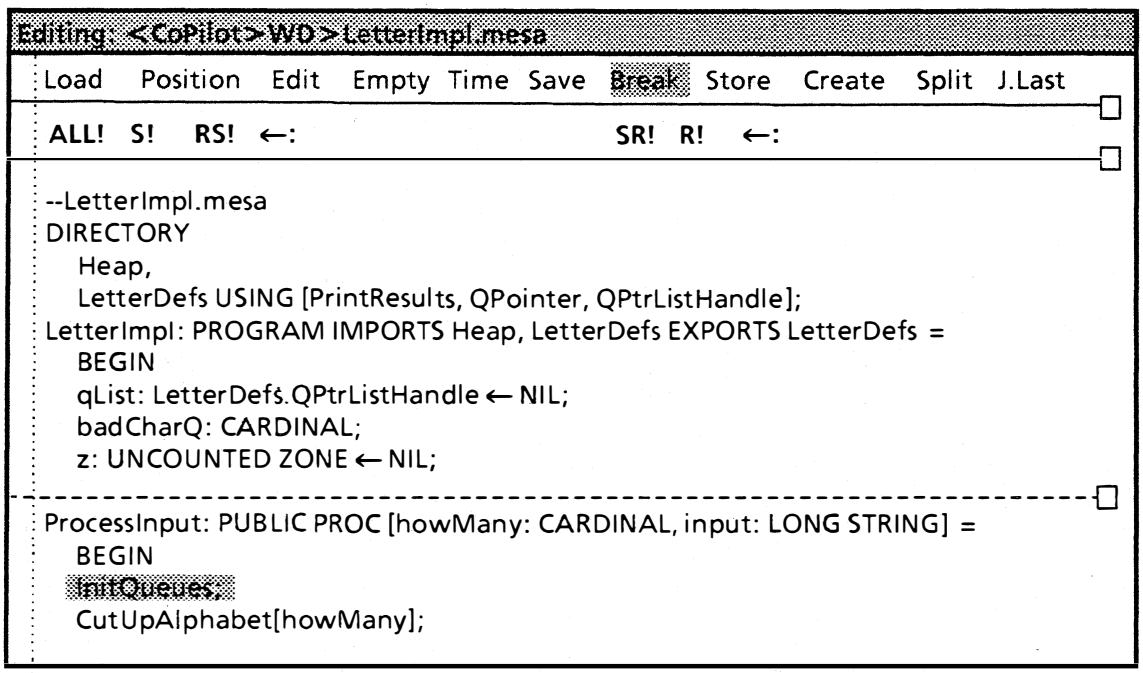


Figure 8.6: Setting a breakpoint in source code

Breakpoints are numbered sequentially throughout a debugging session. Breakpoint numbers are never reused within a session: that is, if you set five breakpoints and then remove them all, the next breakpoint that you set will be #6, and not #1 again.

Conditional breakpoints

Breakpoints may also be conditional; that is, you can attach a condition so that the breakpoint is only taken when the specified condition holds.

The ATtach Condition command in the debugger takes two arguments: a breakpoint number, and a condition to be attached to that breakpoint. (If you don't know the number of the breakpoint that you're interested in, you can find it out with the List Breaks command.)

You can use any of the relational operators (< , >, =, <=, =>) for specifying your conditions. You can also specify the number of times that a break will be bypassed before the debugger is called. For example, "ATtach Condition #: 1, condition: 3".

Clearing breakpoints

There are several commands that allow you to remove a breakpoint. CLear All Breaks, CLear All Entries, and CLear All Xits allow you to remove more than one breakpoint at once.

You can clear a specific break with the Clear Break [#] debugger command, or with the Clear command in the

DebugOps menu. You can also CLear Condition, CLear Entry Break, or Clear Xit Break.

Looking at breakpoints

You can use the List Breaks command or the Display Break command to take a look at your currently active breakpoints. Display Break takes a breakpoint number as argument and lists its type, the procedure or module name in which it is found (for source breakpoints, the source text is also displayed.) Any conditions attached to the breakpoint are also displayed.

List Breaks lists the above information for all currently active breakpoints. You can try it now, if you like, but since you haven't set any breakpoints, there won't be much to see.

Display stack

Once you have set a breakpoint, returned to the debugger, and set the current context, you are ready to start actually probing around and looking at values within the current context. Display Stack is one of the most common commands for looking at the runtime state

Display Stack shows the procedure call stack of the current process. Try typing "ds" in the debugger window now. Notice that the debugger returns with some information (most likely that it doesn't have any symbols for the current module), and then gives another >. Type another question mark, and you will notice that you have a different set of commands available. The Display Stack command gets you into a different command mode, where you have a different group of commands available:

g displays the global variables of the module

j(n) jumps down the stack n levels

l lists the source line that invoked the debugger

n moves to the next frame

b reverse of n; shows the previous frame

p displays the formal parameters

q quits out of Display Stack mode and returns you to the top level of the debugger.(DELETE will also do this.)

r displays the return values

s loads the source file into a window (if it isn't already), and scrolls the source line that invoked the debugger to the top of the window. The source line is also displayed in the debugger window.

v displays the frame's variables

Note: the full set of Display Stack subcommands is not always available For example, when the debugger cannot find a

necessary symbol table, the commands that allow you to look at variables and parameters are all disabled. After all, if the debugger can't find the symbols, how can it let you look at them?

Therefore, you will have to wait until the debugging session at the end of this module before you get to practice Display Stack. When you want to try it, you can look back at this message to see the possible options. Remember that you will have to Quit out of Display Stack mode before you will have access to any of the top-level debugger commands.

The interpreter

CoPilot contains an interpreter that allows you to perform operations such as assigning new values to variables, dereferencing, making procedure calls, indexing, field access, displaying variables and types, and simple type conversion.

If you type a question mark in the debugger again, you will see that the first command choice listed is a space (" "). Typing a space gets you into interpreter mode. (You can also type a space to get into the interpreter from within Display Stack mode.)

For example, let the interpreter evaluate an expression for you: try typing " 2 + 6 MOD 4" in the debugger (don't forget the initial space); the debugger will do the calculation and provide the answer.

The rest of this tutorial consists of a practice debugging session that will allow you exercise the debugger and its interpreter.

Getting started

The "program" that you will be using to experiment with the debugger is called MiscProcs. This program consists of three miscellaneous procedures grouped together in one file: there is no mainline code, so there will be no visible results when you run the program.

Load the file MiscProcs.mesa into a file window, and take a look at the code if you like. (You aren't expected to be familiar with Mesa syntax yet, but there are enough similarities with PASCAL that you shouldn't have too much trouble reading the code.) There are some global declarations at the beginning of the file, and then four procedures, called Factorial, FreeOldNodes, Interchange, and MakeLinkedList.

You will need to compile this program in CoPilot and then run it in Tajo. Once you reach Tajo, you will need to hit SHIFT-STOP to return to CoPilot.

Now use Command Central to Compile and Run MiscProcs. You don't need to bind anything.

Setting the module context

You should now be back in CoPilot; the debug.log will tell you that the reason you are here is that you interrupted (from

Tajo). Type "cu" in the debugger to take a look at the current context. Since MiscProcs doesn't really do anything, the current context will not be the context that you are interested in (look at the module name). So, the first thing that you need to do is set the current context.

To do this, use the SEt Module context ("sem") command, with MiscProcs as argument. You must type the name just as it appears in the internal declaration (MiscProcs: PROGRAM). (Capitalization matters, and the extension should be left off, since it is not part of the internal name.)

(Remember, there are a lot of ways to specify the current context, such as by process number, module name, or configuration name.)

Try CUrrent context again to see the new context. Your debug log should look something like the one in Figure 8 7.

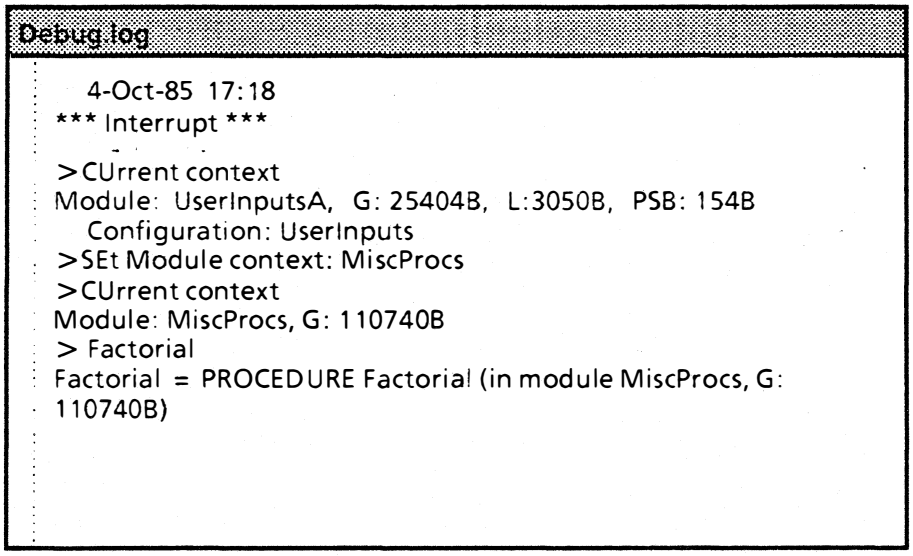


Figure 8.7: The debug log

Making a procedure call from the interpreter

You are now going to make a call to the recursive Factorial procedure. As you can see, there is no global code to make a call to Factorial; this module by itself contains no way to make calls to Factorial.

In the debug.log window, type " Factorial" (don't forget the space, which invokes the interpreter). This doesn't invoke the procedure; it just displays information about it so that you can be sure that you and the debugger are talking about the same procedure Factorial (Again, the name of the procedure must match the internal declaration of the procedure.)

Now, to actually make a call to Factorial, type " Factorial [#]", where # is any number between 1 and 12. This number will be used as the procedure argument. The procedure will only accept input in this range; it will return 0 for the factorial of any number not between 1 and 12. (You might want to read

the next paragraph before you hit a carriage return, so that you will know what to expect.)

Your screen will go grey as control returns to Tajo to execute the procedure call. When the call has finished, control will return to CoPilot, and the result will be printed in the debug.log window.

Setting breakpoints

Now suppose that you would like to get a closer look at this recursive procedure. By setting a breakpoint as you exit Factorial, you can watch the recursion unwind. Use the debugger "bx" command to set a breakpoint at the exit of Factorial ("Break Xit procedure: Factorial".) This will be Breakpoint #1.

Call Factorial through the interpreter again (" Factorial [4]"). When you return to CoPilot, type " j" in the interpreter to look at the value of the variable j, which is the value returned by Factorial. (Don't forget the carriage return after the j.) Since this is the first time through, j will be 1, the factorial of 0. Now type p to Proceed back to Tajo and continue execution.

The second time that you hit your breakpoint, the value of j will still be 1 (the factorial of 1) Use the interpreter to check this. Now try the Display Stack command. This command will tell you that you are in the procedure Factorial, which is in the module MiscProcs When you are in Display Stack mode, type a question mark to see the commands that are available to you within this mode Try L, P, R, S, V, and G. When you are through experimenting with the Display Stack subcommands, use Q(uit) or DELETE to return to the upper level command processor

If you are now convinced that the program works, use Clear All Breaks, or Clear Break #1 to remove the breakpoint. Then Proceed back to Tajo; the procedure will finish executing and tell you that the factorial of 4 is 24.

Procedure Interchange

Take a look at the procedure Interchange in MiscProcs. To see that the array (A) is currently empty, type " A" in the debugger; you will see that the array has 13 elements, and that they are all currently 0.

Suppose that you want to change some of the values in this array. For example, you can set the first element to 7 by typing " A[0] ← 7". Change the values of the first three elements of this array to be 7, 8 and 9. The array should now look like this: A = (13)[7, 8, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0].

Now call Interchange, and interchange the values of the first and last elements (" Interchange [0, 12]"). When you return to CoPilot, take another look at the array to verify that your switch has in fact been made.

MakeLinkedList

The last procedure in MiscProcs is called MakeLinkedList. This procedure builds a simple linked list. For example, if you specify a list of 5 elements, headNode will point to an element that contains the letter E; headNode ↑.next to the letter D, and so on. To see the first element, type " headNode ↑, to see the second, type " headNode ↑.next ↑ ", etc.

You are undoubtedly familiar with linked lists, so we will leave you to experiment with the debugger as much as you like. Set some breakpoints in MakeLinkedList; display the stack; change some values if you like; make a breakpoint conditional. Remember, the fact that your program runs in Tajo and your development environment runs in CoPilot means that you cannot harm any of your development tools by experimenting.

If you run into trouble, you can always look back at earlier messages in this file, or reference the debugger chapter of the XDE User's Guide.

When you are through experimenting, there is one additional tutorial that you might want to look at. This final tutorial, called ToolBuilding, discusses how to build new tools for the XDE.

This tutorial provides an introduction to building new XDE tools. We first provide a brief overview of the "philosphy" behind XDE tools and then discuss the mechanics of creating a new tool. To do this tutorial, you should be familiar with the compile-bind-run cycle of tool development, but you don't have to be familiar with the Mesa programming language.

Subwindows

Part of the XDE philosophy is that all tools should have a consistent user interface; this makes life easier for the user as well as for the programmer. The user interface is based on subwindows; there are various standard subwindow types, and most tools have a user interface built from those basic subwindow types. You will learn more about subwindows as you start programming in the XDE; for now, you just need to be familiar with the three most common types: the message subwindow, the form subwindow, and the file subwindow.

A message subwindow is used to post feedback from the program to the user, and a form subwindow is used to simplify parameter collection. A file subwindow is just a text subwindow, and can have various uses. For example, the File Tool has four subwindows: a message subwindow, two form subwindows, and a file subwindow, as illustrated in Figure 9.1.

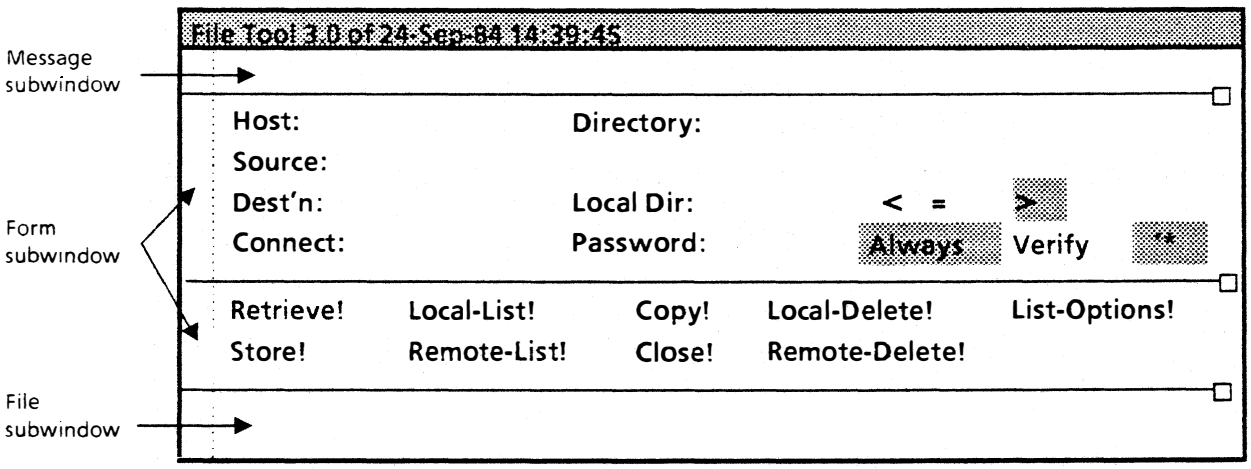


Figure 9.1: File Tool window

The first form subwindow lists various parameters, and the second form subwindow contains the tool's commands. The file subwindow is used for keeping a log of filing operations. (There is no functional reason for having two form subwindows; they could equally well have been combined.)

The FormSWLayoutTool

One of the advantages of standard subwindows is that they make it very easy to create a new tool. To build a tool, you compose it of various subwindows, depending on the kind of functionality that you want. With the exception of a form subwindow, the standard subwindow types are independent of the particular tool. Thus, a message subwindow in one tool is just like a message subwindow in another tool, and the code to create and manage that subwindow is identical.

This makes it possible for you to create a new tool interface from an existing one just by changing the layout of the form subwindow. XDE provides a tool called the FormSWLayoutTool that supports this approach; it allows you to graphically specify the form subwindow that you want your tool to have, and it then generates code to produce a window with your new form subwindow, a message subwindow, and a file subwindow. Thus, you just "draw" the form subwindow that you want your tool to have, and let the layout tool generate code to create that user interface.

The layout tool always generates a window with three subwindows: a message subwindow, a form subwindow, and a file subwindow; the only thing you specify is the format of the form subwindow. However, once the code has been generated, you can edit the code to add or remove subwindows, or reorder the existing ones.

Plagiarize

Run the FormSWLayoutTool (Form subwindow layout tool) from your Executive window. The FormSWLayoutTool window has three subwindows: a message subwindow, a form subwindow, and a file subwindow, as illustrated in Figure 9.2.

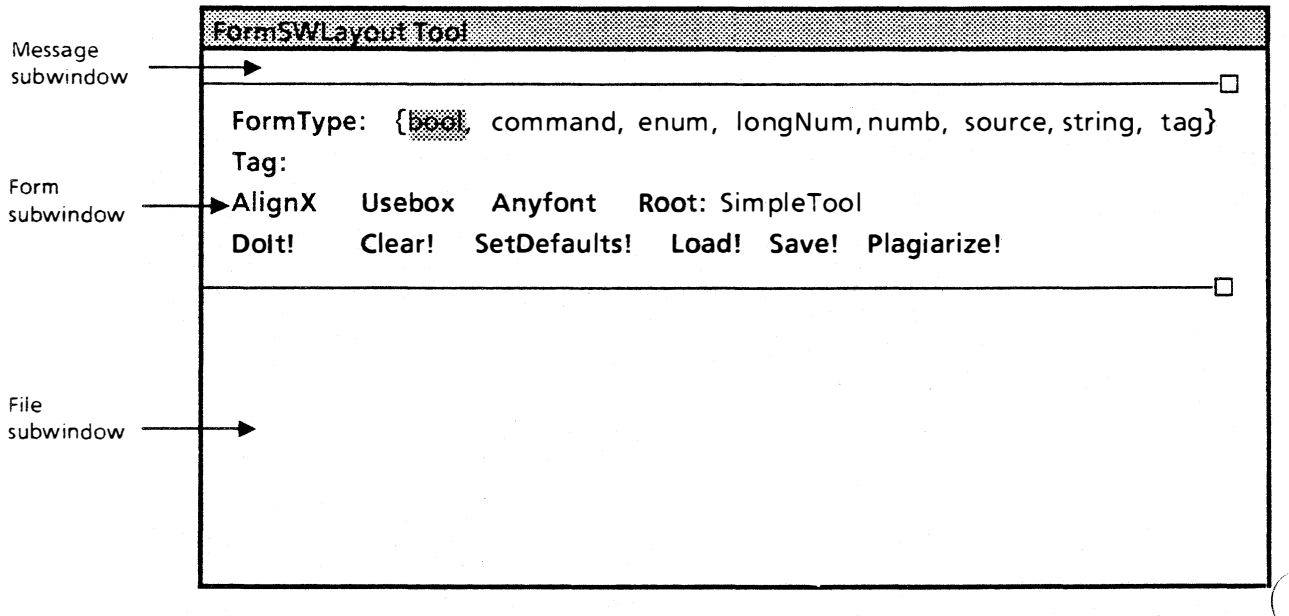


Figure 9.2: FormSWLayoutTool

To use this tool, you "draw" the layout of a new form subwindow in the file subwindow of the layout tool. There are two ways to put a form item on your new form subwindow: you can add them individually or you can "plagiarize" from another form subwindow on your screen.

To see how plagiarizing works, bring up your Command Central window. Now invoke the Plagiarize! command, and then click Point over the form subwindow in Command Central. (The cursor will change into an "eyeball" while you are in plagiarize mode.) When you click Point over the form subwindow that you want to plagiarize, a copy of that subwindow will appear in the bottom subwindow of the FormSWLayoutTool, as illustrated in Figure 9.3.

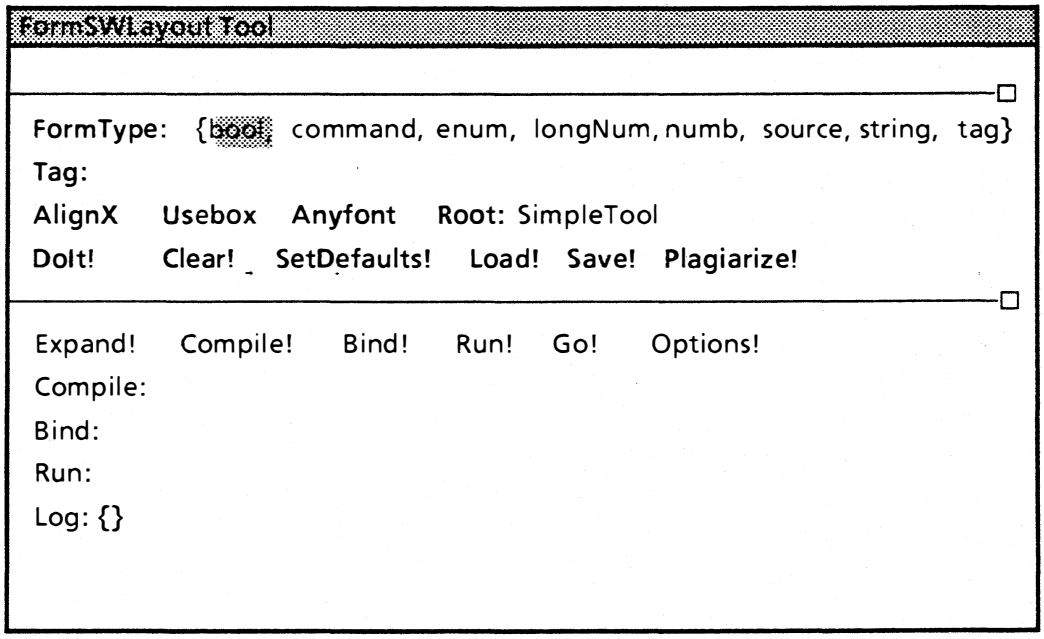


Figure 9.3: Plagiarizing a form subwindow

Once you have plagiarized a subwindow, you can edit the plagiarized copy using the DELETE, MOVE, STOP, and UNDO keys. MOVE lets you move a selected form item around the file subwindow; DELETE deletes a selected item. UNDO brings back the last form item that you deleted; STOP lets you abort in the middle of a MOVE command. Try using these keys to change the items that you just plagiarized.

When you are through using the editing keys, invoke the Clear! command to clear the bottom subwindow.

Layout mode

You can also create a form subwindow "from scratch" by adding items one at a time. To add a form item to your new form subwindow, you first select the type of item that you want from the FormType: enumeration in the layout tool, and then you enter a tag for it in the Tag: field. (If you aren't familiar with the various form items, such as boolean and enumerated type, check your XDE User's Guide for details.)

For example, suppose that you want build a form subwindow just like the one in the MailTool. (Normally, of course, you would just do this with Plagiarize!, but for now do it one item at a time instead.)

To start off, create the Display! command. To do this, select "command" as the FormType in the layout tool's form subwindow, and enter the word Display in the Tag: field, as illustrated in Figure 9.4.

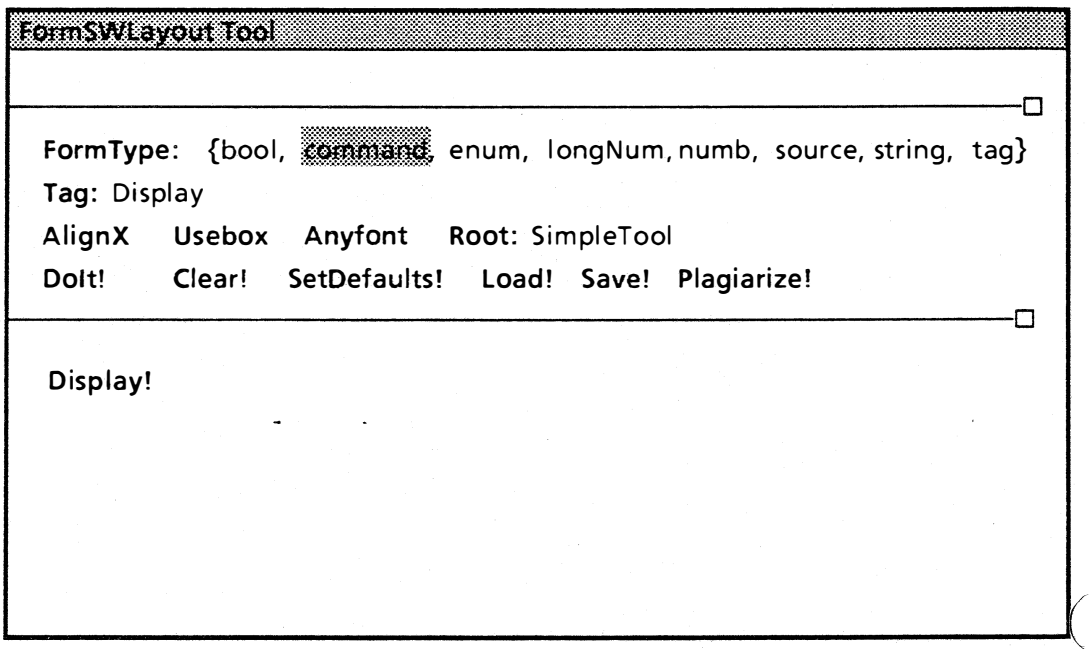


Figure 9.4: Adding an item to the tool

Now move your cursor into the bottom subwindow; notice that the cursor changes form as you do so. Move the cursor around until you find a spot that you like, and then press Point; your new command item will now be in the bottom subwindow. As long as you have a value in the Tag field, you are in layout mode. Thus, you can continue to move the cursor around in the bottom subwindow and place as many copies of the Display! command as you like.

Note: You don't have to include the punctuation that normally follows a form item, such as ! or :; the tool deduces the necessary punctuation from the type of the item and adds it automatically.

Now move your cursor back into the layout tool's form subwindow, and change the value of the Tag: field to Delete, and then move back into the lower subwindow, position the cursor, and click Point. Continue until you have copied each item from the MailTool into your own new form subwindow.

When you are through, or if you wish to edit any of the items that you have already put in the bottom subwindow, **remove the item from the Tag: field before doing any editing**. As long as there is a value in this field, the tool is in layout mode, and you won't be able to edit the items in the bottom subwindow

Enumerated items

One of the items that you just created is an enumerated item, which means that it should contain a list of all possible values for that entry. However, reasonably enough, the tool is not able to deduce the values that you want your enumerated type to have. Therefore, whenever you create an enumerated item, it is followed by the words {fix the enums}. This should serve as a reminder to you that you need to specify a list of the values that this item can take on.

To do this, remove the value in the Tag field (if there is one), move your cursor into the lower subwindow, and select your new enumerated type. Now press the PROPS key, and a property sheet for that item will appear. This property sheet contains various pieces of information about the item, most of which refer to the names of variables inside the code. For the most part, you will not have to edit any of the items in this sheet. For enumerated items, however, you need to put the values for your enumerated type in the Choices: field. The values in this list can be single words or quoted strings of multiple words; individual entries are separated by spaces. For example, if you wanted to list your favorite kinds of bagels, your Choices entry might look like Figure 9.5

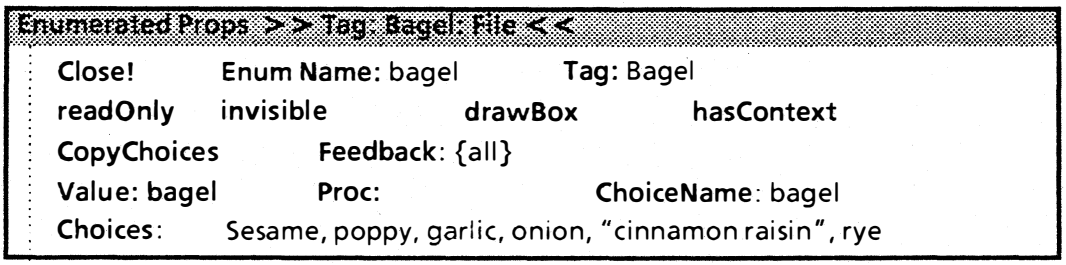


Figure 9.5: Property sheet for an enumerated type

When you have specified the choices, you also get to specify whether all of the choices will be displayed, or just the one that is currently in effect. If you select the Feedback: {one} option, only the current choice will be displayed. If you choose Feedback: {all}, all the choices will be dipslayed in the form subwindow, and the choice currently in effect will be highlighted.

When you have set up the property sheet to reflect your enumerated type, invoke Close!.

Note: There is a property sheet associated with every item that you create. Normally, you will only have to edit the property sheet for your enumerated items, but you can call up a property sheet on any of your form items.

FormSWLayoutTool booleans

The form subwindow for the layout tool also has several important booleans: AlignX, Usebox, and Anyfont.

AlignX controls the spacing between form items. When AlignX is on, each column will start on multiples of a specific distance

from the previous column. Using this boolean will help you to keep the items in your form subwindow cleanly aligned.

Usebox specifies that the generated tool will have the same window box as the current size of the layout tool. Since the user can always control the size of any tool on the screen, this boolean just allows you to control the initial size of a new tool.

Anyfont causes the layout tool to generate code that will have proportioned space on the form subwindow, regardless of the system font being used. Turning on this boolean is a good idea, since you don't know what system font the user will choose.

Saving a form

If you are working on a complicated form, you might want to save an intermediate version of it and later retrieve it. If you crash your machine without saving the form, you will lose the items in the bottom subwindow of the layout tool.

To save a form, enter the name of your tool in the Root: field and then invoke Save!. For example, if you invoke Save! with the value SimpleTool, the layout tool will create a file named SimpleTool.by. You can then later use the Load! commmand to load this file back in and continue editing.

Set Defaults!

The SetDefaults! command allows you to set various default characteristics for your form items. Most of these characteristics correspond to those in the property sheet for a form item; you don't have to worry about these defaults if you don't want to. Later, when you become more familiar with the code that the tool generates, you may want to modify some of these defaults. Try SetDefaults now so that you have an idea of the kind of default values that this command allows you to set.

Creating a simple tool from scratch

To illustrate how this all works, you are now going to do a very simple tool from scratch. This tool performs operations such as blinking the display and "beeping" the sound generator.

The first step is to use the FormSWLayout tool to create the user interface for this tool. You will need three command items (Beep, SetBackground, and Blink), two numbers (Frequency and Duration) and one enumerated type (Background: {white, black}). Set up the form subwindow any way you like, but be sure that you include these six items. You should also make sure that you turn on the AnyFont boolean. Your FormSWLayoutTool should look something like the one illustrated in Figure 9.6.

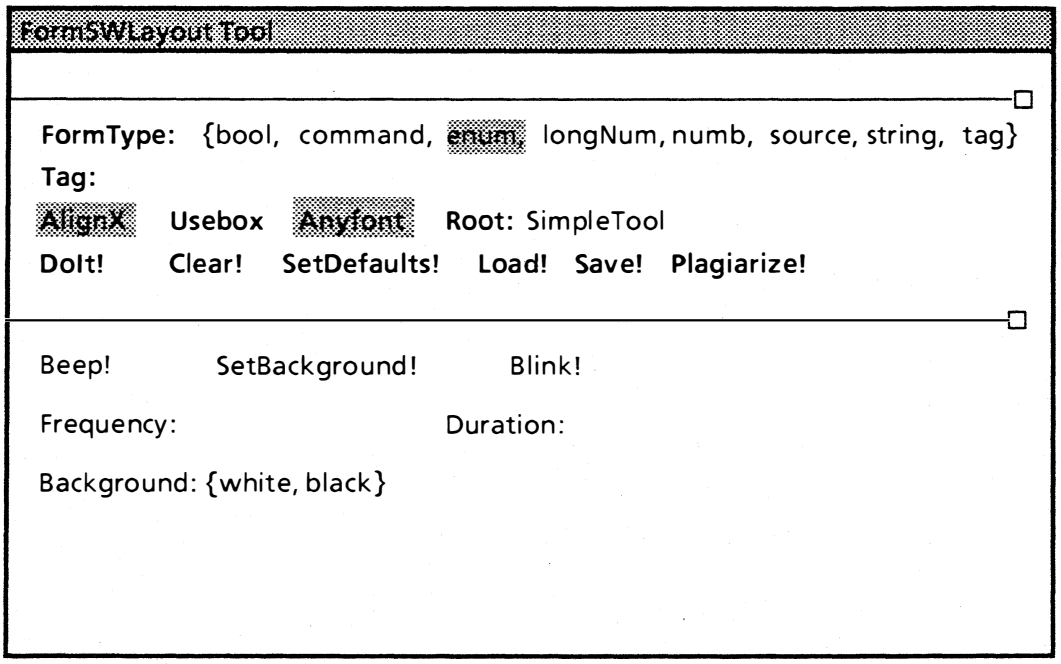


Figure 9.6: Laying out the tool

Frequency and Duration are the arguments to the Beep command. Frequency specifies the frequency of the beep in hertz; this value must be a positive number. Duration specifies the length of time that the beep should sound; this value is specified in milliseconds.

Background is the argument to SetBackground; white and black are the two possible colors for the background. Note that you will have to edit this item to fix the choices. Make sure that you put White and then Black rather than the other way around. (This matters only because you will be inserting some code that has already been written, and assumes that the order of items is white, black.)

When you have the tool laid out, change the name in the Root: field to be the name that you want your tool to have, and then invoke the Doit! command. This command will create a .mesa file, that has as its base name the value that you put in the root field. For example, if you put DumbTool in the root field, the FormSWLayoutTool would generate a file called DumbTool.mesa

Once you have invoked Doit!, bring up an Empty Window and load your new .mesa file into the window. This is the source file for your tool.

Running the skeleton

The code in this file will compile as is. Try it: put the name of your tool in the Compile field and in the Run field of Command Central, as illustrated in Figure 9.7.

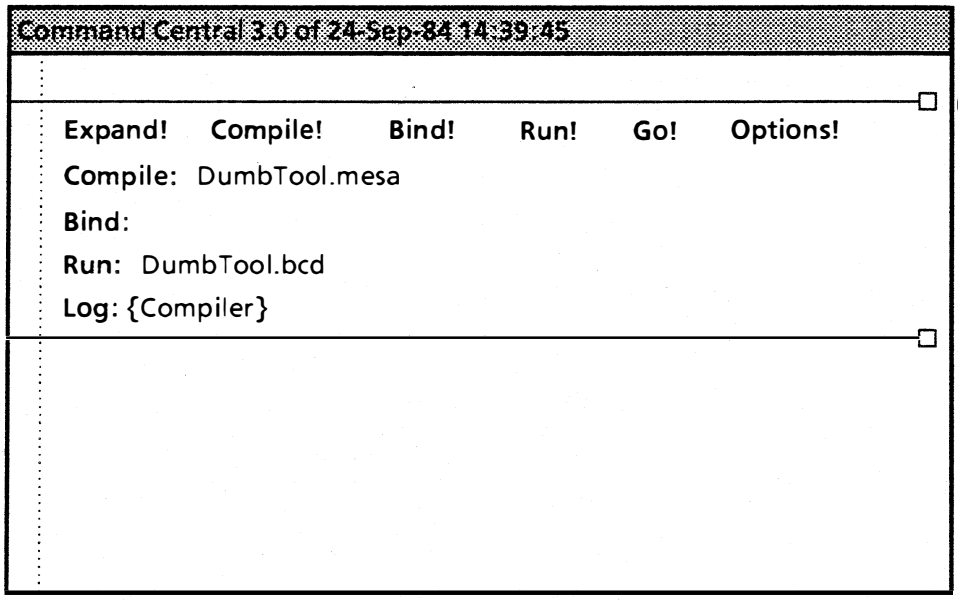


Figure 9.7: How to set up Command Central

When you invoke Go!, your file will be compiled, and then run in the Tajo volume. It won't do anything, of course; it will just present the user interface. You should try changing the values of the Duration and Frequency fields to see if it affects your subwindow at all. If you have the items too close together, putting a new value in for a number or string may obscure some of the keyword for the next item. If this happens, just redo the subwindow so that the items are farther apart.

When you are ready to continue with the tutorial, press SHIFT-STOP to return from Tajo to CoPilot.

Invoke Go!

The code

As you saw by executing the program, the code that the FormSWLayoutTool generates is just the skeleton of a tool. It creates the user interface, but does not provide any of the actual functionality of the tool. Once you have created a user interface for your tool, you still have to add the code that performs the actual operations.

Take a look at the code in your source file. To make this program work, you don't need to understand how this code works; you just need to be able to make a couple of minor changes.

Somewhere near the top of the file you should find "templates" for the commands that you put in your form subwindow. These "templates" are just procedure declarations without any code. They will look like this:

```
Beep: FormSW.ProcType = {
    Put.Line[data.fileSW, "Beep called"L]};

SetBackground: FormSW.ProcType = {
    Put.Line[data.fileSW, "SetBackground called"L]};

Blink: FormSW.ProcType = {
    Put.Line[data.fileSW, "Blink called"L]};
```

Delete these templates, and insert the following code:

```
Beep: FormSW.ProcType = {
    UserTerminal.Beep[data.frequency, data.duration]};

SetBackground: FormSW.ProcType = {
    [] ← UserTerminal.SetBackground[data.background]};

Blink: FormSW.ProcType = {
    UserTerminal.BlinkDisplay};
```

Editing the Directory list

The final change that you need to make is to edit the DIRECTORY and IMPORTS lists at the top of your file. Basically, these lists specify that this module is calling procedures from other modules. You will learn more about these lists as you start programming in Mesa. Now, however, just delete the word Put in both of these lists and replace it with the word UserTerminal. (**Capitalization is important, and make sure that you don't delete the trailing comma**) The beginning of your program should now look like this;

```
DIRECTORY
 FormSW,
 Heap,
 UserTerminal,
Tool,
ToolWindow,
Window,
WindowFont;

DumbTool: PROGRAM

IMPORTS
  FormSW, Heap, UserTerminal, Tool, WindowFont = {
```

Running the tool

The tool is now ready to go. Compile it and run it again from Command Central, and experiment with it in Tajo. Remember, the values for frequency should range from 50 to about 20000 (depending on how good your hearing is), and the duration parameter is in milliseconds. (10 might be a good initial value.)

When you are through, you are ready to go on to the next part of your Mesa training, whatever that might be.

The following files are required for doing the tutorials. One way to retrieve these files is to execute the command file Tutorials.cm. If you are a Xerox employee, you can retrieve this command file from [Tundra:OSBU North:Xerox]<XDE Documentation>Tutorials. If you are not a Xerox employee, you should consult a local expert to find out where these files are stored.

Necessary files

TutorialFiles	TeachBasics.nsmail TeachFiles.nsmail TeachText.nsmail TeachBooting.nsmail TeachPrinting.nsmail TeachMailSystem.nsmail TeachCompile-Bind-Run.nsmail TeachDebugger.nsmail TeachToolBuilding.nsmail	MiscProcs.mesa ToolFactorial.config ToolFactorialDefs.bcd ToolFactorialDefs.mesa ToolFactorialImplA.mesa ToolFactorialImplB.mesa User.cm
Released software	Binder.bcd CommandCentral.bcd Compiler.bcd FTP.bcd MailTool.bcd Maintain.bcd Print.bcd FONTS.WIDTHS Format.bcd FormSW.bcd Heap.bcd NSFilingConfig.bcd Put.bcd Tool.bcd ToolWindow.bcd Traps.bcd UserTerminal.bcd Window.bcd	
Unsupported utilities	FontMonster.bcd Fetch.bcd FSWindowTool.bcd FormSWLayoutTool.bcd RemoteBrowser.bcd SmoothScroll.bcd Scrollbar.TIP	