Dan Bricklin's™

Demo II Program User Manual

By
Dan Bricklin

A Software Garden™ Product

Dan Bricklin's Demo II Program User Manual

Software Garden, Inc., PO Box 373, Newton Highlands, Massachusetts 02161

Table of Contents

Introduction

The Software Garden™ Dan Bricklin's™ Demo II™ Program, called DEMO II, is a professional tool for producing program prototypes, demonstrations, and tutorials. You can use DEMO II to help you conceptualize programs, describe them to others, refine their functionality and human interface, and teach users how to use the final product. It is designed to be used by programmers, documentation writers, graphic designers, product managers, marketing people, and others.

DEMO II is an enhanced version of the award-winning **Dan Bricklin's™ Demo™ Program** (without the "II"). It has all of the features of the older program plus over 75 new features and changes.

DEMO II can be used for a wide variety of applications by an even wider variety of people. In order to appeal to such a wide audience, it is a program rich in many features and capabilities. **This richness may be intimidating at first, but the program is designed so that you can use just the features needed for your application and ignore the others until your needs expand, if ever.** The "How Do You...?" section of this manual is a good place to see how quickly you can create the results you might need.

What EVERYBODY Should Do First!

The first thing you should do to learn about DEMO II, even if you know you can learn it on your own or are familiar with older versions of the product, is to read the on-line information.

To see the on-line information, insert the diskette labeled "***Second Diskette***" in the A: drive, then get to the "A>" prompt. If you do not know how to do that or what it means, you may not know enough about DOS to learn DEMO II on your own from the documentation. In that case, please check your DOS manual or find someone familiar with DOS for assistance.

Execute the command:

```
RUNME
```

This information replaces all of the little printed notices, "Read Me Firsts," and "Last Minute Addenda" that you usually find in a software package. It also is a way of showing off the capabilities of the product. After you see the information shown there, you can go about things the way you normally would (reading the manual cover to cover, ignoring the manual, skimming the manual, or whatever).

Do not forget to backup the diskettes. You will note in the License Agreement and Warranty, whose terms you must have read and accepted before opening the diskette package, that you can make backup copies to protect yourself from equipment and other such failures. We recommend that you do keep backups. Most people just create a "\DEMO2" directory on their hard disk and put copies of all of the files on both of the diskettes there, saving the originals as the backups. That should work fine.

Finally, remember to **send in the registration card**. This is the way Software Garden, Inc., knows who has DEMO II. You should send it in even if you have registered a previous product, such as the older **Dan Bricklin's Demo Program**.

To help you learn about the program and to quickly make effective use of it, the DEMO II documentation includes a variety of learning aids. Your learning style should determine which of the aids you use.

Included in the package are this manual (with both overview and reference sections), keyboard templates, an on-line tutorial, and sample files.

The "Overview" section of the manual, which should be at least skimmed by everybody, is a must for those who like the "big picture." DEMO II has several novel concepts, like "overlays", that may not be obvious without some explanation. This section has several illustrations that help explain those important concepts.

For those who like to try something step by step first, there is a simple tutorial on the Second Diskette. See the on-line information to find out how to start the tutorial.

In this manual, there is a rather complete set of reference descriptions of DEMO II's use of the keyboard, the commands, and the menus. For people who like all of the details, this is where to look. There are also a few sections that cover features, such as running and overlays, in greater depth.

There is also a wide variety of miscellaneous information. In addition to the short examples scattered throughout the reference material, there is the "How Do You...?" section, which lists modes of operation and how to go about getting them. You will find some of the tabular information and other material you may need, also. There are example files on the Second Diskette, described by the on-line information, which show at least one way of implementing a variety of functions and which demonstrate the capabilities of the program.

Function Key templates for the two most common IBM® PC keyboards are provided in the DEMO II package to help you learn DEMO II's use of the function keys.

Finally, there is an index and table of contents.

Overview: Tying It All Together

Introduction

This section of the manual gives an overview of DEMO II. The general concepts needed to understand its operation are presented. In addition, some of the more difficult parts are described in greater detail so that you can find which commands and features are appropriate for your needs.

This overview should be at least skimmed by all users, including those who have used the older **Dan Bricklin's Demo Program**.

This overview section is divided into several subsections. They are: *What is DEMO II All About?*, *Slides and Editing*, *The Command System*, *Variables*, *Overlays*, *Switching From One Slide To Another*, *Running*, and *Save, Load, Print and Other Input/Output*. Finally, there is diagram tying it all together.

What Is DEMO II All About?

DEMO II allows you to create **slide shows**. A slide show is a series of one or more images on the computer's screen that can mimic the appearance of a running program, convey information, entertain, attract the viewer, etc. See Figure 1. Almost anything a normal program can display on the screen can be mimicked by DEMO II, as well as the appearance of many of the actions of a running program. This is useful for prototyping software before it is written, demonstrating existing software, and doing computer based training (CBT).

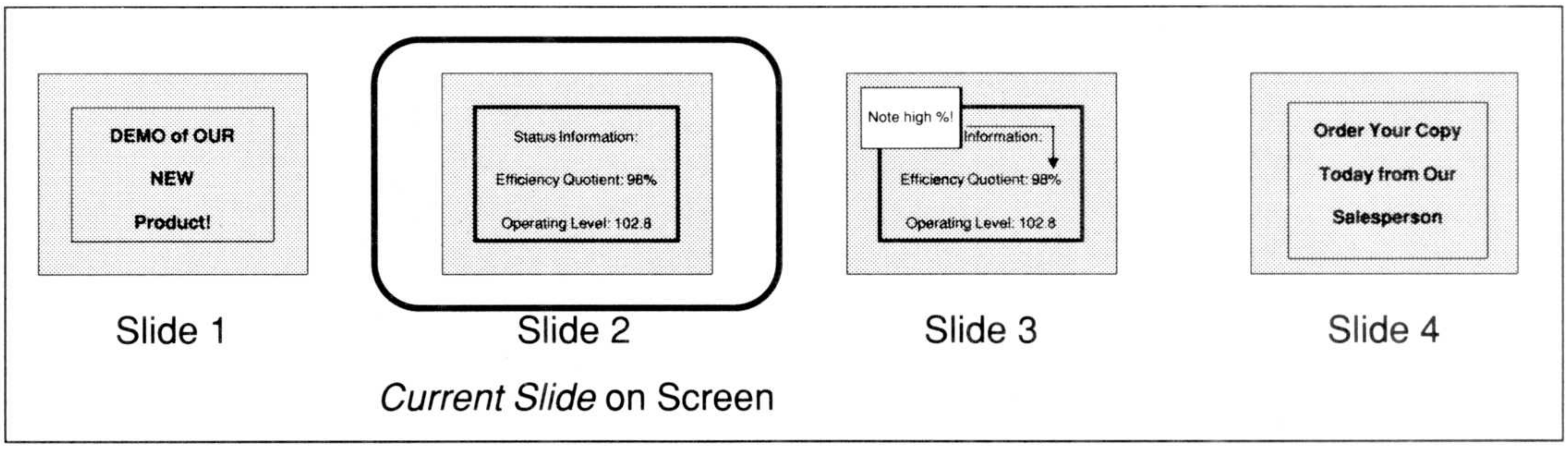


Figure 1. A slide show

DEMO II provides facilities to create the screen images (called **slides**) from scratch, to "capture" images from existing programs while they are running, and combinations of the two.

A powerful "overlay" facility allows you to construct screen images out of other slides and variable information. This lets you reuse the contents of one slide in constructing another, which saves memory and simplifies global changes. It also lets you display variable information, such as that "typed in" by the user, to simulate "fill-in-the-forms" and other systems.

The ability to create slides and show them on the screen is often sufficient for producing screen prototypes. For demonstrations and tutorials that are to be shown to other people or that must simulate a running program more accurately, DEMO II has the ability to automatically switch from slide to slide. This is called **running** the slide show. You can control the sequence of slides, as well as the responses to keys pressed by the user, modify and test variables, and perform a wide variety of other operations. You can run a series of slides within DEMO II, then immediately return to editing to make changes. There is also a "runtime only" version of the program (called RDEMO2) that only runs slide shows and cannot be used for editing.

Slides and Editing

There are two types of slide images: **text** and **bitmapped**. See Figure 2. Text slides consist of 25 rows of 80 character positions. Each character position has two parts: the **character** and the **attribute**. The character is one of the 256 text characters that the IBM PC can display (letters, numbers, box lines, smiling faces, etc.). The attribute is one of the 256 ways in which the IBM PC can display those characters (normal, inverse, blinking, blue on red, green on cyan, etc.). Your computer's display (the CRT or monitor) and display adapter (the part of the electronics in the machine that controls the display, with names such as CGA, EGA, Hercules™ Graphics Card, etc.), determine which attributes actually show. A monochrome system cannot display blue on red, for example. DEMO II is especially tuned to make working with text screens easy and powerful.

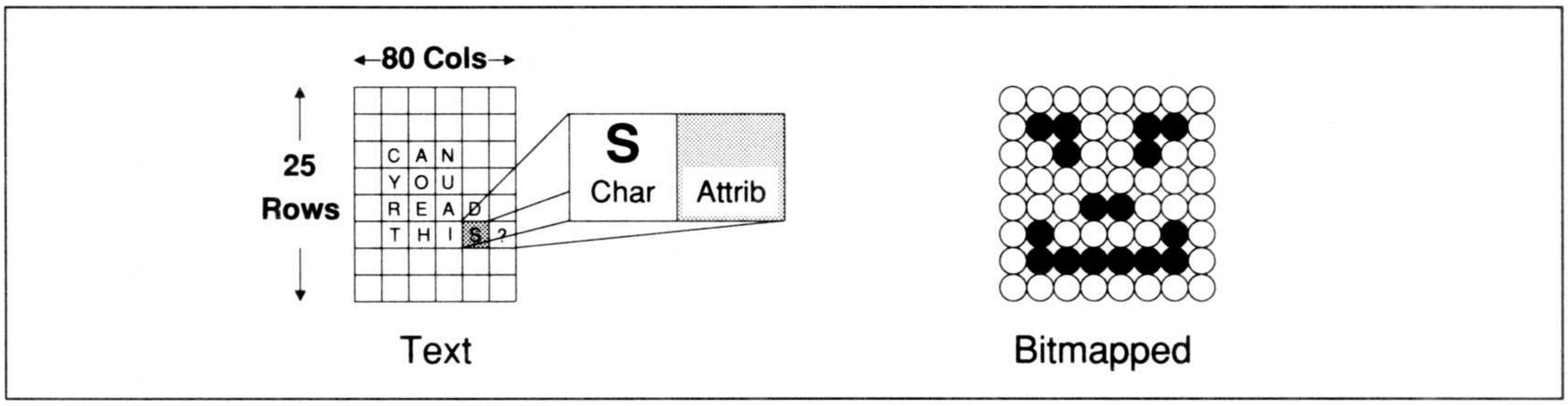


Figure 2. Types of slides

A bitmapped image consists of the specification of all of the dots (or pixels) on the screen. Depending upon your display and display adapter, this can be 64,000 (320x200), 128,000 (640x200), or even 224,000 (640x350) positions, rather than just the 2000 character positions of a text image. DEMO II can display a variety of bitmapped images, but cannot edit them. Bitmapped slide images are produced by another program (such as ZSoft Corporation's PC Paintbrush®) and stored in ".PCX" files. They can also be captured from the screen while another program, which produces bitmapped images, is running. The CAPTURE program (used to capture both text and bitmappped screen images) is included in this package. Captured images are stored in memory along with text slides. You set a slide to have a bitmapped image by retrieving a captured bitmapped image with the I/O Retrieve command. See the section on "Bitmapped Graphics Images" for more information.

You can have many slides in the computer's memory at one time and you can switch between them very quickly and smoothly. They are all automatically stored in a compact manner by replacing repeated characters, attributes, or groups of pixels with a repetition count and one byte. Since most slides have large blocks of the same attribute and often the same character or pixel repeated, the memory saved by compacting can be great. An all-blank slide takes up about 96 bytes, including some status information.

The rest of this subsection is a discussion about editing text slides.

When you type a normal, text-character key (such as X or $), the character is placed at the character position marked by the cursor. The **cursor** is a highlighted character position used to indicate where many operations are to occur. Normally the program is editing in Overwrite mode, and the cursor just moves forward. In Insert mode, the characters to the right of the cursor are pushed over to make room for the new character. You toggle between Overwrite and Insert mode by pressing the Ins key.

There are a variety of editing commands in DEMO II that simplify the creation and modification of slides. Some of the commands are executed by pressing editing and function keys on the keyboard. Others are selected from menus.

The editing keys include the arrow and other **cursor motion keys**. The cursor motion keys move the cursor from character position to character position. There are keys to delete characters and/or attributes from the screen (Del and Backspace), and for copying or moving text. Other keys aid you in drawing lines or boxes, or give you a list of special characters to insert (such as the arrow characters and the smiling faces). A few of the keys and menu commands are for setting and changing the attributes.

There is a setting, accessed from the Block menu, called CAB that affects many of the operations, including typing. CAB stands for "**Character, Attribute, or Both**." The CAB setting specifies whether they operate upon just the characters, just the attributes, or both characters and attributes. For example, if the CAB setting was Character and you moved a block of characters, only the characters themselves would move; the attributes at the source and destination would remain unchanged. If the CAB setting was Both, moving would move both characters and attributes, overwriting the attributes that were at the source and destination before. There is a command on the Block menu, as well as a function key, to change the CAB setting.

Many operations work on a **marked block**. There may be only one marked block active at any given time. The block is a rectangular set of character positions. You mark a block by a variety of methods. The most common one is to position the cursor at one corner of the block and press F9 (or use the Block Begin command). You then move the cursor to the opposite corner. Throughout the operation, the block will be defined as being from that first corner to the cursor, and will be indicated on the screen by an outline around the outside of the block. When a block is marked, the characters immediately outside of the marked block are temporarily obscured by this outline. You do operations on the contents of the marked block by invoking one of the Block commands while a block is marked. The operations usually unmark the block when completed. Block operations include deleting the block, copying, moving, changing the attributes of all the characters in the block, word-wrapping the text in the block, and drawing a box around the block.

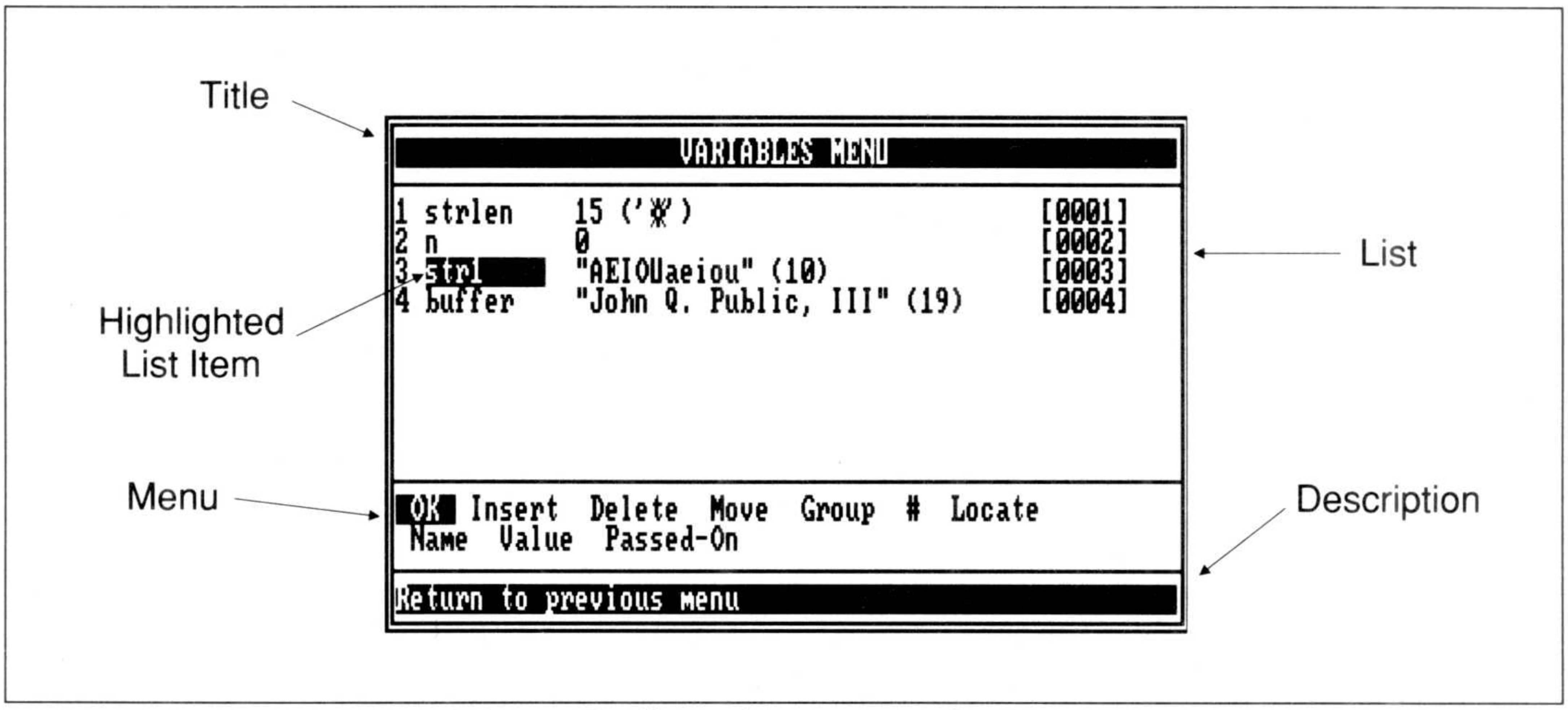
To aid you in typing, there are commands on the Typing menu. Most are also available on single function keys. They let you draw lines and boxes (Typing Lines or F3), insert a special character (Typing Chars or F5), type up or down (Typing Direction), and set horizontal and vertical tabs. You can also set the left margin, provide extra status information on the screen, such as cursor position or slide name/number, and move the cursor to specified characters on this or a following slide (Typing Find or Grey *).

The Command System

The Command System is the main method for communicating with the program. The command system is composed of several parts: **command windows**, **prompts**, **menus/messages**, and **function keys**. Many users will find the operations of the command system very natural and can skim this subsection.

Command Windows

You can invoke many commands from edit mode by pressing the Esc key. A *command window* pops up on the screen in a location away from the cursor. A command window can have one or more of the following parts: *Title*, *List*, one item of which is highlighted, *Menu*, and *Description.*


```
                     VARIABLES MENU
1 strlen     15 ('☼')                          [0001]
2 n          0                                 [0002]
3 strl       "AEIOUaeiou" (10)                 [0003]
4 buffer     "John Q. Public, III" (19)        [0004]

OK  Insert  Delete  Move  Group  #  Locate
 Name  Value  Passed-On
Return to previous menu
```

Figure 3. Sample command window

The title identifies the command window, e.g., MAIN MENU, VARIABLES MENU, or SET BLOCK TO ATTRIBUTE.

The list is a set of items. For example, the list of attributes would have one item for each defined attribute, and each item would be composed of the name of the item and its hexadecimal value. Frequently, the items in a list are prefixed by an identifier (a number or letter) to allow you to select them by typing just one character. You can use the up arrow, down arrow, PgUp, PgDn, Home, and End keys to move a highlight through the list to peruse it, and to find an item to select. These keys are called the List Perusal Keys.

Lists frequently have the six commands, Select, Insert, Delete, Move, Group and #, in the menu. The Select command is frequently first, so that pressing Enter selects the highlighted item and applies the highlighted menu item. The meaning of that selection is dependent upon the particular command window. Insert creates a new item under the highlight, while Delete erases the highlighted one. Move lets you reposition the highlighted item in the list. The order of items has different meanings for different lists. The Group command toggles the start of a group of items. The group extends from the Group Start to the currently highlighted item. The Delete and Move commands (and sometimes some others) operate on all of the items in the group. The # command lets you move to a specific item, for example the 121st item in a list.

A menu is a list of commands, one after another in a row, taking up one or more lines of the window. One of the commands will be highlighted. You choose a command by moving the highlight with the left and right arrow keys to rest upon the command you want, and then pressing Enter. You can also just type the initial character of the command. The Ctrl-Home and Ctrl-End keys can be used to move to the first and last commands, respectively.

Finally, there is often a description area at the bottom of the window. This will either give you instructions for operating on the contents of the window (e.g., a prompt during a list-item move operation), or be a description of the highlighted command. This means that you can see what each of the commands is used for just by moving the highlight from command to command with the arrow keys.

Pressing Esc cancels the window. Pressing Ctrl-Break cancels the window and gets you all the way back to edit mode.

The Main menu is the first command window that pops up when you press Esc while editing. When we make reference to a menu, such as the "Slides Options menu," we mean "press Esc, then select the Slides command to get the Slides menu, then select the Options command there to get the Slides Options menu."

Note that, in terms of commands, the operation is similar to that popularized by Lotus Development Corporation's 1-2-3® program.

Prompts

There are instances when DEMO II displays a message that requires your immediate attention, e.g., when you ask to quit and return to DOS, and DEMO II wants to know if you really want to do it despite unsaved changes; or when you make an erroneous request such as deleting an item in an empty list. To communicate with the user, DEMO II makes use of **prompts**.

A prompt is displayed in a rectangle in the middle of the screen. There are five types of prompts: *away next key*, *space to continue*, *fatal*, *single character*, and *type-in string*.

The simplest prompt is the "away next key" type. An example of this is the message that appears when you create a new slide with Shift-F3 or Shift-F5. These prompts are meant to give you some message, but not to interfere with your work. They are removed from the screen when you press the next key. The key that removes them is treated normally — it *is* executed. You may want to use the left arrow key to make the prompt go away.

Most error conditions are reported in a "space to continue" prompt. These prompts remain on the screen, and all keystrokes are ignored until you press the space bar.

Some errors, such as disk errors that occur while loading data, leave the program in a state such that it would be imprudent to continue. These errors result in a prompt with a highlighted line saying that there was a fatal error. Pressing Enter exits DEMO II.

There are times when DEMO II needs your decision about what to do next, such as whether to truncate an existing file. In those cases a "single character" prompt is used. You need only type in one of the characters listed (whether it is uppercase or lowercase is usually ignored). Either Esc or Ctrl-Break is also acceptable, and they are usually treated as a cancel or a "No."

Finally, when a multiple-character response is needed, such as the new name for a file, a "type-in string" prompt is used. A proposed response may or may not be presented. If you just start typing, the proposed response will be erased. You will be in an insert mode, and you may correct mistakes with Backspace and Del. The left arrow, right arrow, Home, and End keys may also be used. If you start by moving the cursor, or using Backspace or Del, the proposed response will not be erased. When you have finished entering your response, or wish to accept the proposed response shown, press Enter. If you want to cancel the operation, press Esc or Ctrl-Break. You may not type in more characters than the space shown on the screen. If you fill the space, you may have to use the Del key to erase the last character. There is a special version of the "type-in string" prompt for setting some numeric values and variables. This multiple-line prompt is described in greater detail in the subsection on *Variables*, below.

Menus/Messages

There is one other method that DEMO II uses to present information to help you control its operation. It is a cross between command windows and prompts. These menus/messages tell you that you are in a special mode, such as line drawing, and they tell you which keys are active. Like command windows, menus/messages are displayed in the screen corner opposite the cursor. Like single character prompts, only the listed keys may be used.

Function Keys

Some common commands not provided by the normal, labeled keys on the IBM PC (Home, End, Del, etc.) can be executed by pressing function keys. These are the keys labeled F1 through F10. There are 20 such commands defined — 10 for the keys themselves, and 10 for when the keys are pressed in conjunction with Shift. Most of the function key commands are identical to operations accessible through the command windows. They are provided to make use of the program more quickly and easier. Most function keys are only active when you are in edit mode (see the command descriptions) and not when a command window or prompt is displayed, unless indicated. The Shift-F6 key is the only general exception. See the Macro Learn command description for more information about Shift-F6.

A complete description of the function keys is in the "Keys" section of this manual. While editing, you can get a list of the function key definitions by issuing the Help command in the Main menu. A keyboard template is also provided in this package. It is worthwhile to learn to use the function keys, since DEMO II was designed with them in mind for speeding common operations during development of a slide show.

Variables

The presence of variables is one of the features that gives DEMO II the power to be very flexible while running a slide show. Like most programming languages, **variables** are named locations in the computer's memory that can store values. A value is either a number (an integer from -32768 to 32767) or a group of characters called a "string" (0 to 80 characters). A variable has only one value at a time — the value last assigned to it by some operation.

Most values that can be specified with the name of a variable can also be specified with a constant value, so you do not need to understand variables to start using DEMO II.

Variables are optionally used to control a large number of things in DEMO II. They can be used to control the presence and position of text shown on the screen (overlays); they can hold user input (to let the user of a slide show enter field information or answer questions); they can be set and tested to control the display order of the slides in a slide show, and you will find them helpful in many other applications. If you are not going to automatically run your slides, you may be able to skip most of the discussion about variables.

A list of all variables defined in a slide show is displayed in the Variables menu. Whenever you specify a variable, this list is searched. If the name you specify already exists, the variable with that name is used. If not, you are given the option of defining the variable. You must specify whether the variable is to refer to numeric values (N), or string values (S). Once a variable is defined, it stays the same type, unless you change it with the Variables menu Value command. Variables are given an initial value when defined (0 for numbers, no characters for strings). You can change the value with the Variables Value command or by executing actions while running.

Associated with each variable is its name, its value, and its "Passed On" setting. The name may be from 1 to 8 characters, and should be unique. If you change the name, all references to that variable will also have the name changed. The value can be either a number (stored as a 16-bit signed value; that is, a number from -32768 through 32767) or a string (a length value from 0 to 80, and up to 80 bytes of data). The Passed On setting is used when you switch from one slide show saved in a file to another while running, and it determines which variables have their value passed from the first file to the one being loaded.

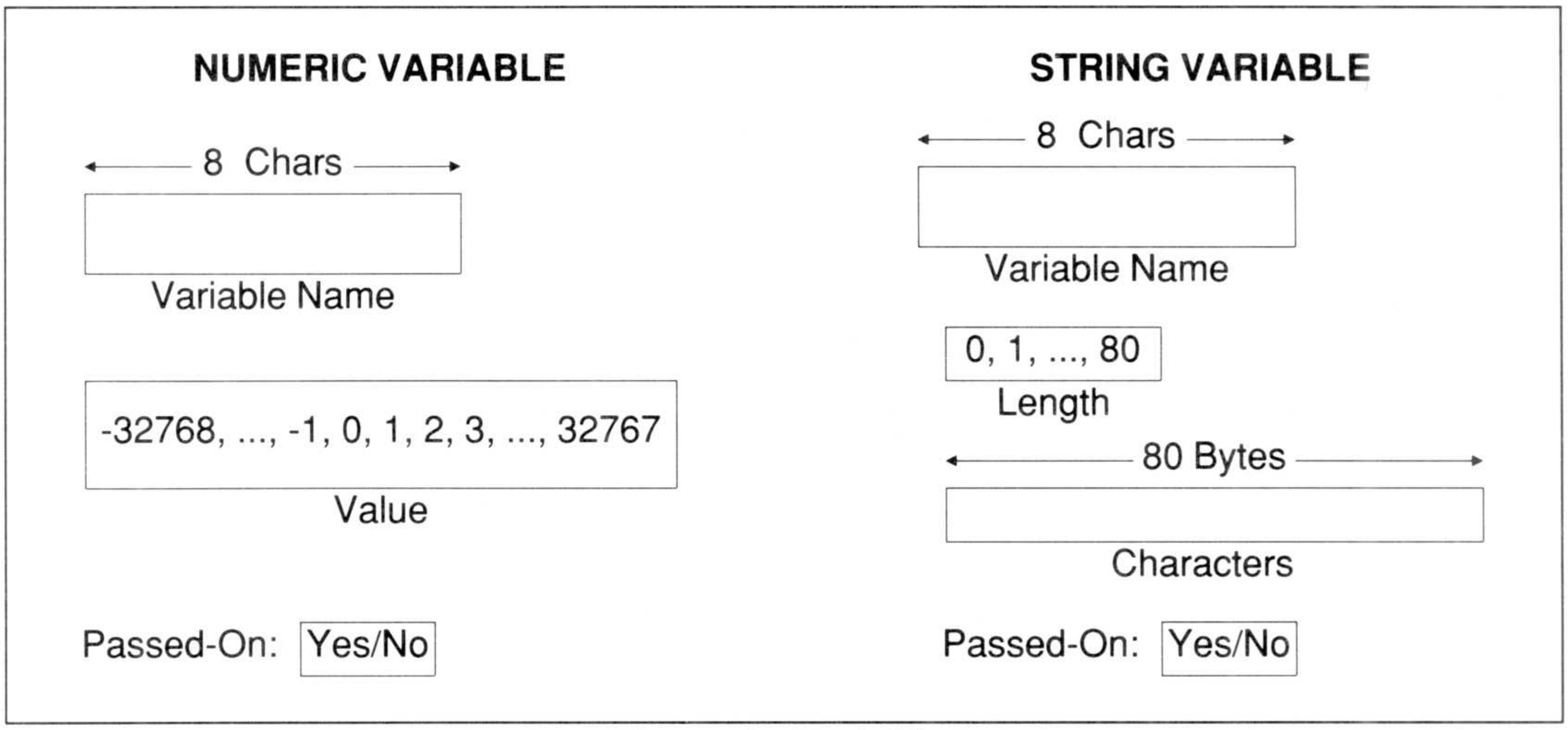


Figure 4. Variables

There are many settings that you can make (such as the name of a bitmapped ".PCX" file or a slide's Run Type) that accept a constant value *or* a reference to a variable. When you try to set or edit these settings, a special, multiple line "type-in string" prompt will popup in the middle of the screen. There will be a descriptive title at the top, an "allowed-input" line, and then the current value at the bottom. The "allowed-input" line will list the types of responses that are allowed.

The response types are:

Yes/No Type **Y** for a 1 or non-zero value, **N** for 0.

Variable Type **V** and then the name of a variable. Used when also Yes/No.

A-Z Just type the name of a variable.

?List Type **?** to choose from the Variables menu.

0-9 Type a numeric value. If the value starts with the characters **x** or **0x**, the value is in hexadecimal.

'Char Type **'** and then press a key to get its value.

"String Type **"** and then up to 80 character string.

End your entry with Enter to accept, Esc to cancel. If you use an editing key first (such as the arrows, Home, End, Backspace, etc.), you can edit the current value.

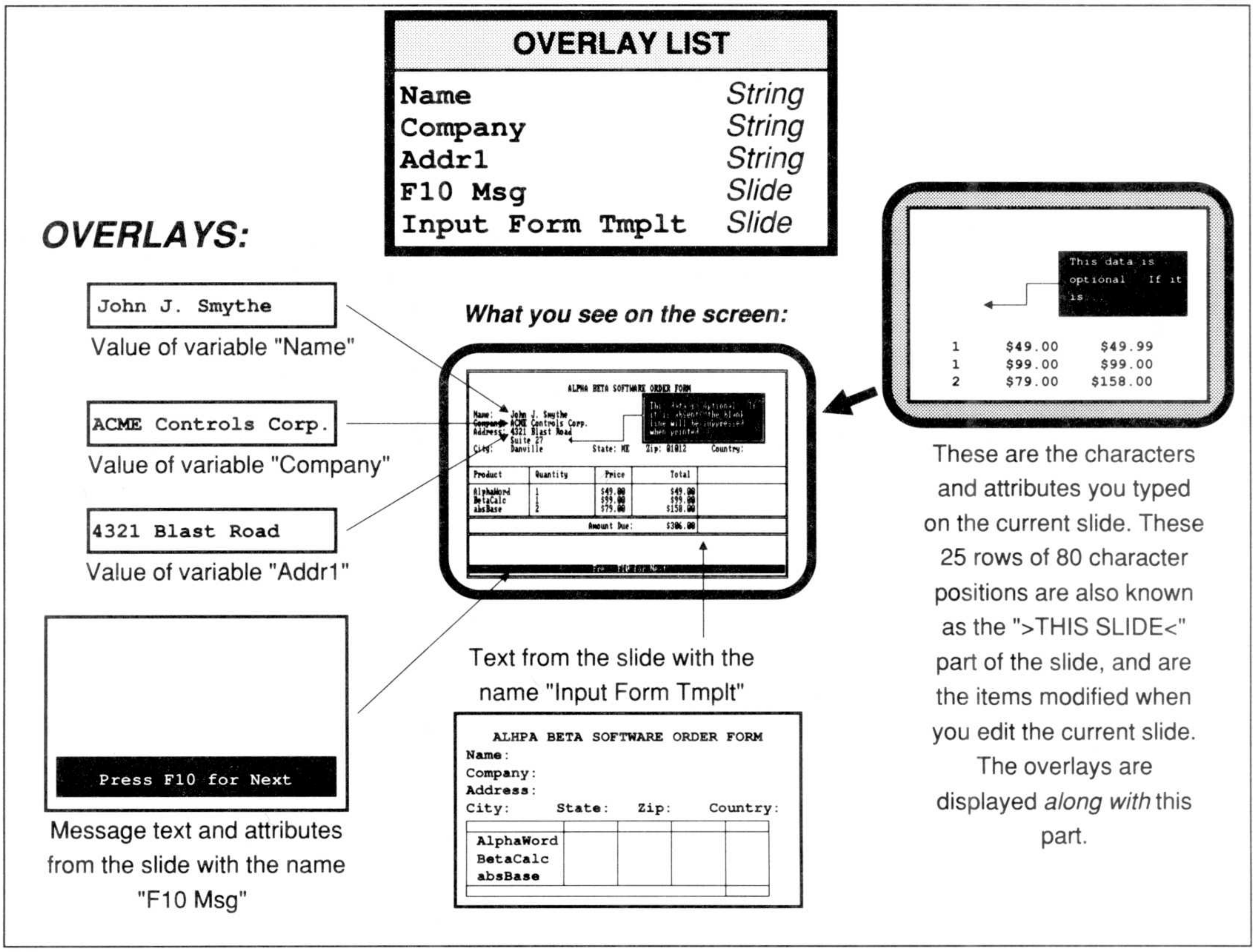
When a string value is used and a numeric value is needed, the first character of the string is treated as a number. That is, the letter "A" would be 65, which is its ASCII value. If the string has a length of 0, it is treated as if no value were set (often defaulting to zero).

When a numeric value is used when a string value is needed, the number is converted to a single-character string. If the value is between 0 and 255, it is used directly (e.g., 65 becomes "A"); otherwise, a zero byte is used (not the character "0", but the value 0). If no value was set, it is treated as a zero-length string.

The bytes (character positions) in string values are numbered from 1 to 80. A number of different operations can be performed on string values while running the slide show. They are listed primarily in the Run Action String menu.

Overlays

DEMO II's overlay feature allows you to have additional items displayed *along with* a given slide. Associated with every slide is an **overlay list**, which can be viewed with the Overlays menu. When a slide is displayed on the screen, its overlay list is examined by DEMO II to find other items to add to the screen. The other items may be



OVERLAY LIST

Name	*String*
Company	*String*
Addr1	*String*
F10 Msg	*Slide*
Input Form Tmplt	*Slide*

OVERLAYS:

John J. Smythe

Value of variable "Name"

ACME Controls Corp.

Value of variable "Company"

4321 Blast Road

Value of variable "Addr1"

Press F10 for Next

Message text and attributes from the slide with the name "F10 Msg"

What you see on the screen:

ALPHA BETA SOFTWARE ORDER FORM

Name: John J. Smythe
Company: ACME Controls Corp.
Address: 4321 Blast Road
Suite 27
City: Danville State: ME Zip: 01012 Country:

This data is optional. If it is absent, the blank line will be suppressed when printed.

Product	Quantity	Price	Total
AlphaWord BetaCalc absBase	1 1 2	$49.00 $99.00 $79.00	$49.00 $99.00 $158.00
		Amount Due:	$306.00

Press F10 for Next

Text from the slide with the name "Input Form Tmplt"

ALHPA BETA SOFTWARE ORDER FORM
Name:
Company:
Address:
City: State: Zip: Country:

AlphaWord
BetaCalc
absBase

This data is optional. If it is...

```
1     $49.00     $49.99
1     $99.00     $99.00
2     $79.00    $158.00
```

These are the characters and attributes you typed on the current slide. These 25 rows of 80 character positions are also known as the ">THIS SLIDE<" part of the slide, and are the items modified when you edit the current slide. The overlays are displayed *along with* this part.

Figure 5. A screen image created using overlays

other slides, values of string and numeric constants or variables, attributes, and the IBM PC's blinking hardware cursor. See Figure 5. Note that the overlay list is only used for text slides; it is ignored on bitmapped slides.

Why would you use overlays? Sometimes you want to reuse the contents of one slide in creating another slide. For example, let's say that the contents of slide 1 is a background and menu template. Once placed on slide 2, it would represent only part of what you would see when you viewed slide 2. You could use the DEMO II editor's copy/paste commands to copy slide 1 onto slide 2, but that would double the amount of computer memory required. Also, if you made a change to slide 1, you would have to make the same change to slide 2 to keep the menus consistent. Another way to do all this is to use overlays. By only making a *reference* to the slide 1, rather than having two copies of the characters, you can save a lot of memory. And, whenever you change slide 1, the image of slide 1 shown with slide 2 will always reflect the changes.

Another, very important use of overlays is to specify which string variables are to be shown along with a slide, and to specify their positions in order to simulate the editing of forms or full-word responses to questions.

If none of these uses for overlays seems that important to your application, you may want to ignore them for now. It is often worthwhile to use DEMO II without them, and become comfortable with the rest of the program first.

What Gets Covered By What?

In order to understand overlays, you must first understand how the screen image that you see is created from the current slide, its overlays, and other items.

There are two types of characters and attributes: **transparent** and **nontransparent**. All characters and attributes are nontransparent, except for the character with value 0 and the attribute with value 0. These are the numeric value 0 (from the range 0 to 255), not the character "0". By default, the slide starts out with all transparent characters and attributes at each character position. All characters that you normally type are nontransparent.

There is a background character and attribute (specified on the Global menu, and defaulting to a space character and a white-on-black attribute). When you view a text slide with no overlays, you are really viewing the slide in front of the background. Any transparent characters on the slide let the background character show through, possibly being "colored" by the attribute in that character position on the screen. Any transparent attributes on the slide let the background attribute show through, possibly "coloring" the character in that position. If both character and attribute are transparent, then the background character is displayed with the background attribute.

This operation is analogous to the background being a blackboard, and the slide being an acetate sheet being held in front of it. The parts of the acetate that are written on would be the nontransparent characters, and the color of any given part of the acetate would be the nontransparent attributes. The lack of writing would be a transparent character, and a clear part of the acetate would be transparent attributes. (This analogy is not perfect, since acetate colors are not opaque and can be mixed when one is in front of another. DEMO II attributes do not mix, but hopefully you get the idea.)

When you are editing a slide, you actually see more than the part you are editing. There is "**>THIS SLIDE<**", which is the 25 rows of 80 columns that you edit. In addition, there are the characters and attributes that show through from the background, as well as any overlays. All together, you are looking at the "Current Slide", as the slide being edited and displayed is called. Editing the current slide edits just the ">THIS SLIDE<" part. Note that when we refer to the "Current Slide", we sometimes mean just the ">THIS SLIDE<" part you edit when you view the current slide.

For example, in Figure 6 we have a simplified picture which assumes that the screen has only four character positions arranged vertically — position-1, position-2, position-3, and position-4. The background character has been set to "X", and the background attribute is medium (out of a possible light, medium, and dark). The ">THIS SLIDE<" part of the current slide has a transparent character and attribute in position-1, the character "A" with a light attribute in position-2, a dark "B" in position-3, and the character "C" in position-4 with a transparent attribute.

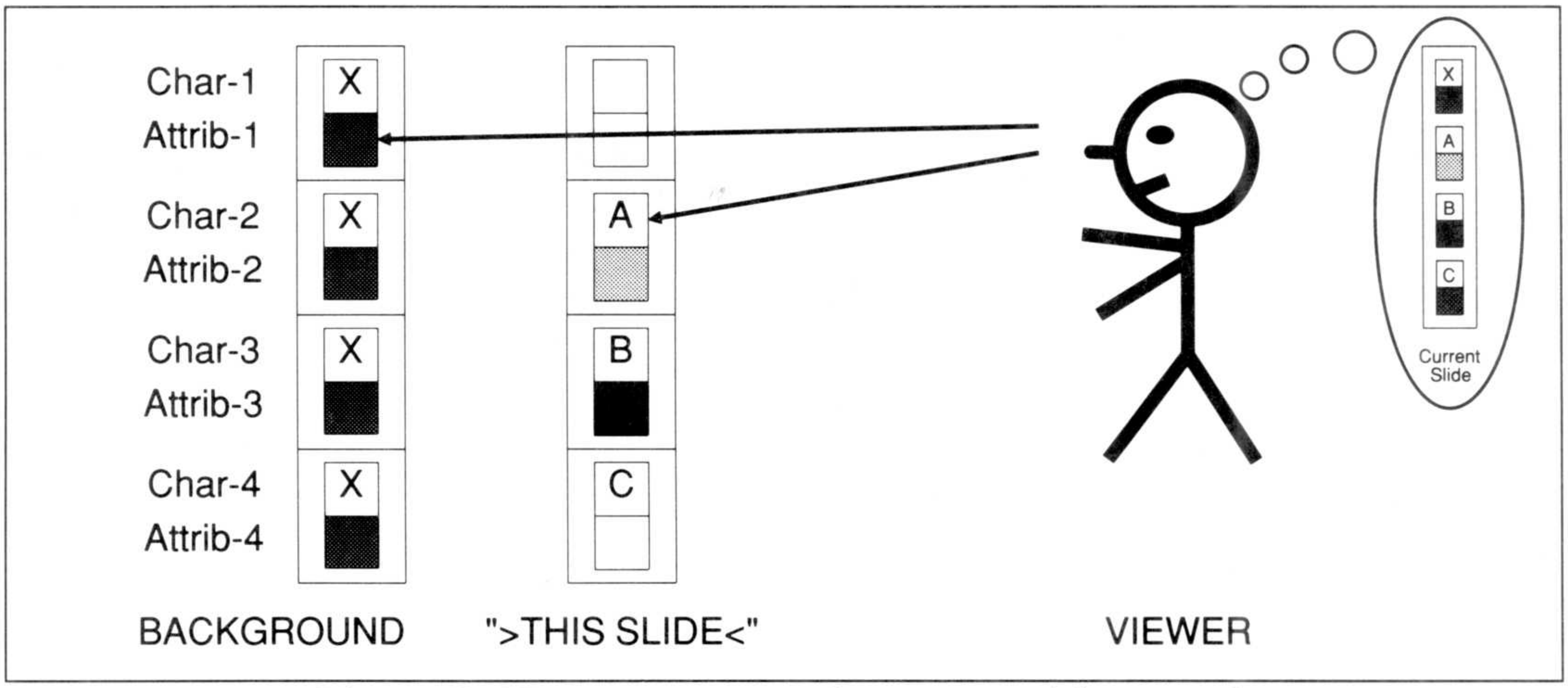


Figure 6. How transparent character positions work

The viewer sees the following:

Position-1 A medium X from the background, since both character and attribute on >THIS SLIDE< are transparent.

Position-2 A light A from >THIS SLIDE<, since both the character and the attribute on the slide are nontransparent.

Position-3 A dark B from >THIS SLIDE<, since both the character and the attribute on the slide are nontransparent.

Position-4 A medium C, with the character from >THIS SLIDE< and the attribute from the background, since only the character on the slide is nontransparent.

Make sure that you understand this example before proceeding. This is one of the most difficult concepts in DEMO II, and it is important to many applications.

Overlays are displayed in front of the background, just like >THIS SLIDE<. The first overlay in the list is displayed immediately in front of the background. The second overlay in the list is displayed in front of that, and so on. The current slide (the ">THIS SLIDE<" part) is always displayed last, by default. If you want another slide last, you can insert the special slide overlay ">THIS SLIDE<" at the point where you want the current slide displayed.

In many cases, "<u>Under</u>lays" may be a better term than "<u>Over</u>lays". Understanding this order is important.

For example, if the overlay list for the current slide was:

```
1   FIRST-ITEM
2   SECOND
3   LAST-ONE
```

they could be represented as in Figure 7.

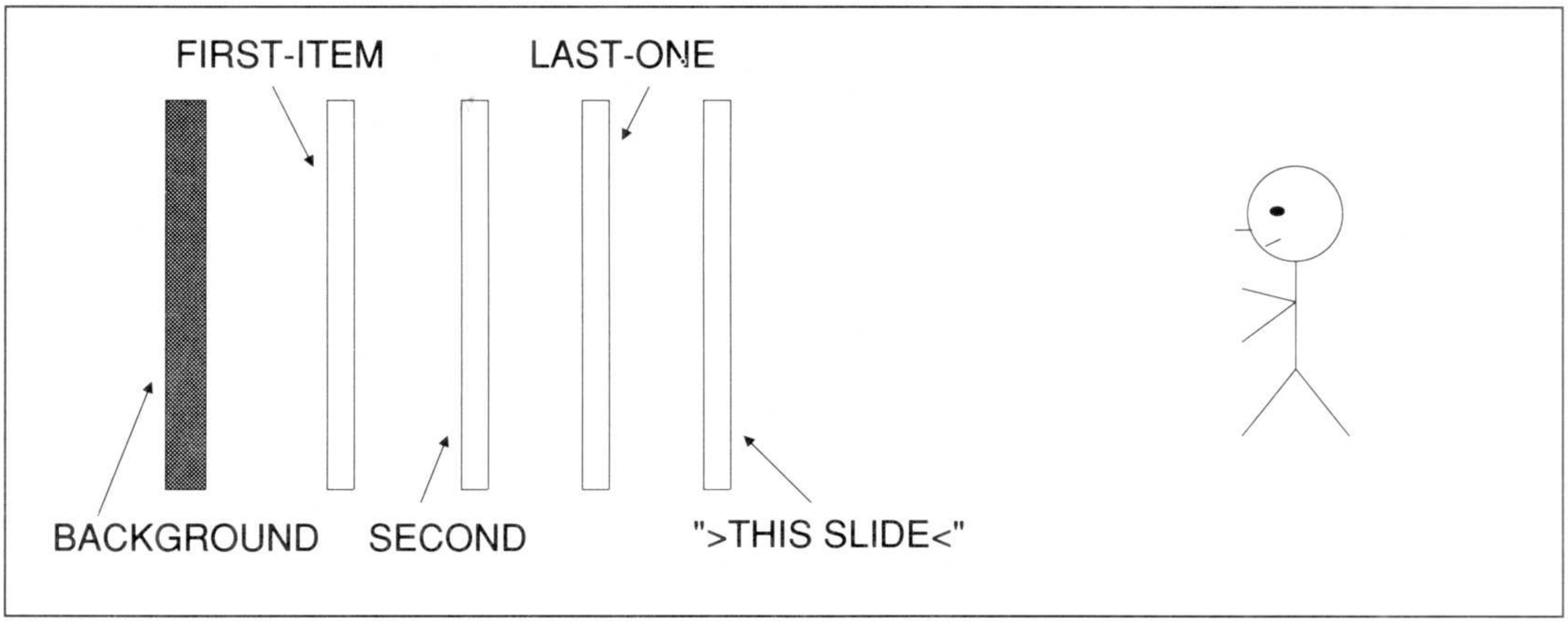


Figure 7. The order of overlays

Nontransparent characters and attributes in the overlays (as well as in >THIS SLIDE<) block characters and attributes in the items before them. Therefore, a nontransparent character in the item "SECOND" blocks the background character and the character in the same position in "FIRST-ITEM." That character, in turn, can be obscured by any nontransparent characters in the same position in "LAST-ONE" or the current slide's >THIS SLIDE<.

If you are using overlays and things are not showing through as you expected, you should check the order of the overlays and see what is transparent and what is not. **Blanks** (the space character produced when you press the space bar) **are not**

transparent. When you capture images of other programs, all of the screen is usually nontransparent. You may have to insert ">THIS SLIDE<" as the first overlay in the list or you may need to "cut holes" in the slide to let overlays behind it show through (using the Block Delete command or F7) in order to get the effect that you want. You can also change the background character or attribute to help you see where the "holes" are. (Use the Global menu, but remember to set them back as they were — the background character is usually a space, not 0!)

Note that only the >THIS SLIDE< part of the current slide is affected by editing. The contents of the overlays are ignored by block deletes, overwriting, etc. To edit the text of a slide displaying as a slide overlay, you must make it the current slide first (by using F1/F2 or the Slides menu — see *Switching From One Slide To Another*).

Global Overlay List

In addition to the overlay list associated with each slide, there is a **Global Overlay list** that is shown on the Global Overlays menu. This list is effectively added onto the end of each slide's overlay list. You can use the Global Overlay list to put an overlay on all slides. This feature is most useful when you use a variable to control when an overlay is visible, and turn the overlay on and off to "pop it up" when needed in front of (or behind) other slides. If the current slide is not listed in either of the lists (as >THIS SLIDE<), it is displayed after all of the overlays. Only the first >THIS SLIDE< takes effect. Any others after the first are ignored.

Types of Overlays

There are several types of overlay items: Slides, Numbers, Strings, Values with the H/W Cursor, Absolute Slide References, Relative Slide References, Attributes, and the H/W Cursor. These are described in detail in the "Types of Overlays" section.

The reason there are so many types of overlays is because there are several types of items that you would want to show along with a slide: images of other slides, referenced in a variety of ways; string values to display viewer typing, computed information, etc.; numeric values to display counts, computed or debugging information, etc.; attributes to simulate cursors and highlighting; and the PC's hardware cursor (the H/W Cursor) to prompt for input or to simulate other program operations. For each of these uses there is an appropriate overlay.

The example files that come in this package, and any others produced by your associates or other DEMO II users, should help to show you how overlays are applied.

Each overlay item in an overlay list takes up 16 bytes of memory. If a slide takes up 1600 bytes, and if you refer to it in the overlay list of 20 other slides, each of those 20 references only takes up 16 bytes. The total of 320 bytes is significantly less than the 32,000 bytes it would take if you copy/pasted the slide onto all the others. Saving memory is reason alone for learning how to use overlays.

Switching From One Slide To Another

Slides are stored in memory in a particular order. Each slide has a **Next** slide and a **Previous** slide (other than the first and last). When you create new slides, you usually insert them in the list after the current slide. You can give each slide a name of up to 16 characters. You can see a list of all slides, along with their names and numbers, on the Slides menu. The Slides menu has commands for inserting new blank slides into the list, for deleting slides, and for changing the order of the slides. Function key Shift-F3 inserts a new blank slide after the current slide; function key Shift-F5 inserts a new slide after the current slide that is a copy of the current slide. The number of a slide always reflects its position in the slide list, and that number is automatically updated whenever the slide list is changed.

When you are editing and you want to make another slide the current slide (the one on the screen and being edited), you can use the commands on the Slides menu. As a shortcut, the F2 key switches the current slide to the next slide in the order of the list, and the F1 key switches to the previous slide in the list. The Shift-F1 key makes the first slide in the slides list the current slide, and Shift-F2 gets you to the last slide. When you switch from slide to slide, the cursor is hidden until you press the next key. In this way you can see how the slide will look without the cursor. (It is also hidden when running).

DEMO II switches from slide to slide quickly and smoothly, so you can use the F1/F2 keys to switch from slide to slide much as you would flip from page to page in an animated "flip book." If each slide were another step in the progression of screens in a simulated program, you could get an idea of how the progression would look. Since you are still editing, you can make changes and get immediate feedback. For prototyping, this is one of the most important features of DEMO II. For simple prototyping, this is all of the switching that you need. For other applications, though, the more sophisticated automatic running facility is required.

When a slide is displayed, it appears on the screen as soon as possible, overwriting what was there before. This switch occurs so quickly and smoothly that only the changes are noticeable from slide to slide. For example, if the only difference from slide A to slide B is the position of a highlight, the highlight will appear to move when you switch from slide A to slide B.

At times you may want some sort of special effect when changing slides. DEMO II provides a feature called **Switch Types**, that gives you different ways to switch from slide to slide. There are switch types for text slides, and switch types for some forms of bitmapped slides. You can set the switch type for a slide by using the Slides Options menu. There is also a Switch Speed. The meaning of the speed value depends upon the slide type and the switch type.

The special effects are only shown when you change slides while running the slide show, not while editing.

The switch types are:

Type 0 Normal. Switch as quickly as possible.

Type 1 Replace the slide already on the screen with the new slide from top to bottom at the speed set.

Type 2 Replace the slide already on the screen with the new slide from bottom to top at the speed set.

Type 3 Replace all characters and attributes that are *different* between the slide already on the screen and the new slide from top to bottom at the speed set. This lets you simulate typing or the appearance of a communications line.

Type 4 The same as Type 3, but make a "click" sound as each changed character is displayed.

See the Slides Options menu description for more information about switch types and switch speeds. The speed is interpreted quite differently for different slide types, so you should check the information in that description before you use this feature.

Running

One of the most powerful features in DEMO II is the ability to run the slides automatically. This means that slides appear on the screen in an order that you determine. They switch at appropriate times, optionally as a response to the viewer pressing keys. (The person watching a slide show is called a "viewer"; the person editing a slide show is a "user".) There is a full programming language that you can use to test and modify variables, make sounds, and control this execution. The language is customized for running slide shows, so processing key presses and changing the screen are very simple to set up. The most common modes of switching from slide to slide were designed so that there is a minimal need to use the programming features.

A knowledge of another programming language such as BASIC, Pascal, or C would be helpful to those who want to take full advantage of all features; however, many applications can be done by using a few simple commands and by following the examples in this manual and on the diskette. If you find that you need the programming features and are not familiar with the general concepts of programming (such as variables, looping, branching, and calling subprograms), you may want to get assistance from someone who does, or study one of the many language systems designed for easy learning (such as Borland's TurboPascal or Microsoft's QuickBasic).

You switch from editing to running by executing the Run command on the Run menu. Running will start with the current slide. When you press Ctrl-Break, execute a Quit action, or an error occurs (such as attempting to switch to the next slide from the last slide), running is stopped and you can resume editing.

By default, running will display a slide, wait for a key to be pressed, display the next slide, wait for another key to be pressed, display the next slide, etc., until there are no more slides to display.

Actually, you can test the key that was pressed, and do different things (such as displaying a slide other than the next slide, or sounding a tone to indicate an error) depending upon which key was pressed, instead of automatically going to the next slide. This testing and execution is called "processing the key." Unless you specify otherwise, though, **the default when processing a key is to move to the next slide**. When you move to another slide, we say that you have "viewed" that slide.

This sequence of viewing slides and processing keys is called Run Type 1. Associated with each slide is its Run Type. The Run Type tells DEMO II what to do when viewing the slide. You can change the Run Type of a slide with the Run Type command. The value is usually a number from 0 to 4.

Also associated with each slide is a Wait Time. The Wait Time is 0 by default, and specifies the amount of time to wait after the slide is displayed before doing anything else. The time is in "clock ticks", which are approximately $^1/_{18}$ of a second. (One clock tick is the same amount of time no matter how fast the PC's CPU runs.) You set the Wait Time with the Run Wait command. In Run Type 1, DEMO II waits the specified amount of time after the slide is displayed before processing any keys.

In Run Type 0, first the slide is viewed. Next the program waits the amount of time specified by the Wait Time for that slide. The next slide is then viewed and the process repeats until there are no more slides, or a slide with a Run Type other than 0 is encountered. On slides with Run Type 0, no keys are read and no keys are processed. This lets you run through a sequence of slides automatically, perhaps simulating a program in operation or moving through descriptive text in a "self-running" demo.

Run Types 2, 3, and 4 are variations upon Run Type 1. In Run Type 2, any keys pressed before the wait time is over are ignored (called "flushing type-ahead" by

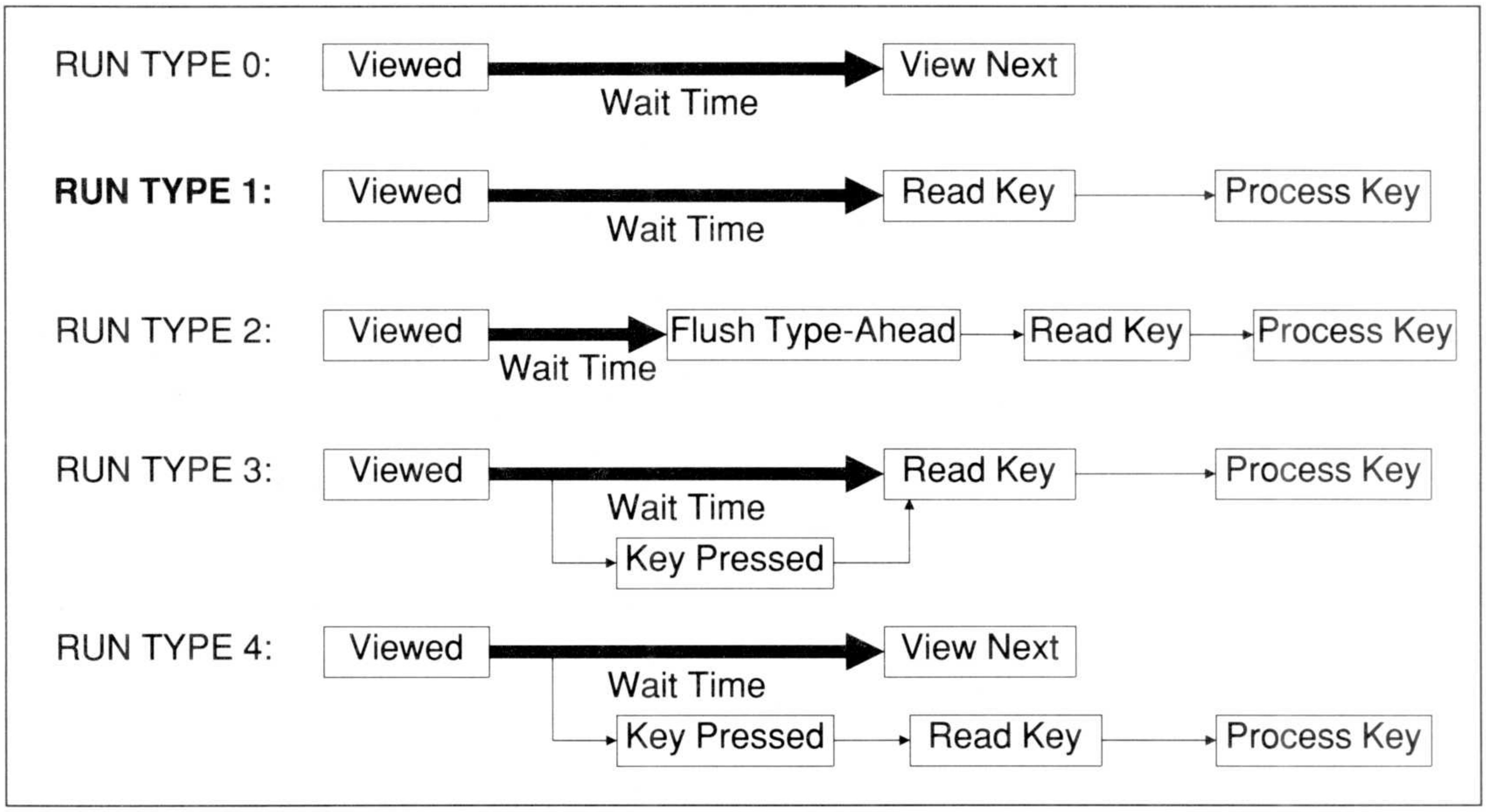


Figure 8. Run Types

programmers). This lets you say "Press Space to Continue" and avoid having a key pressed prematurely by an impatient viewer affect the slide.

In Run Type 3, the wait is terminated as soon as any key is pressed, rather than waiting the full extent of the time period. This lets you process the viewer's keys immediately.

Run Type 4 goes to the next slide after the wait, if no keys have been pressed. If a key is pressed before the wait finishes, the wait is terminated and the key is processed. You can use this to go through a sequence of slides automatically and at a predetermined pace until the viewer presses a key to stop the sequence or to view a different sequence.

A more detailed description of the operation of the Run Types appears in the "How Running Works" section of this manual.

Now that you know how to achieve some control over the automatic displaying of slides, you may be wondering what it means to process a key. One of the most basic concepts that you must understand in order to get control of running is testing and dispatching based upon which key is pressed.

Associated with each slide is a list, called the **Run Action list**. The Run Action list can be viewed on the Run menu. Each line in the list is called an action line. Each action line can have two parts: the Key/Event label, and the Action (see Figure 9, below).

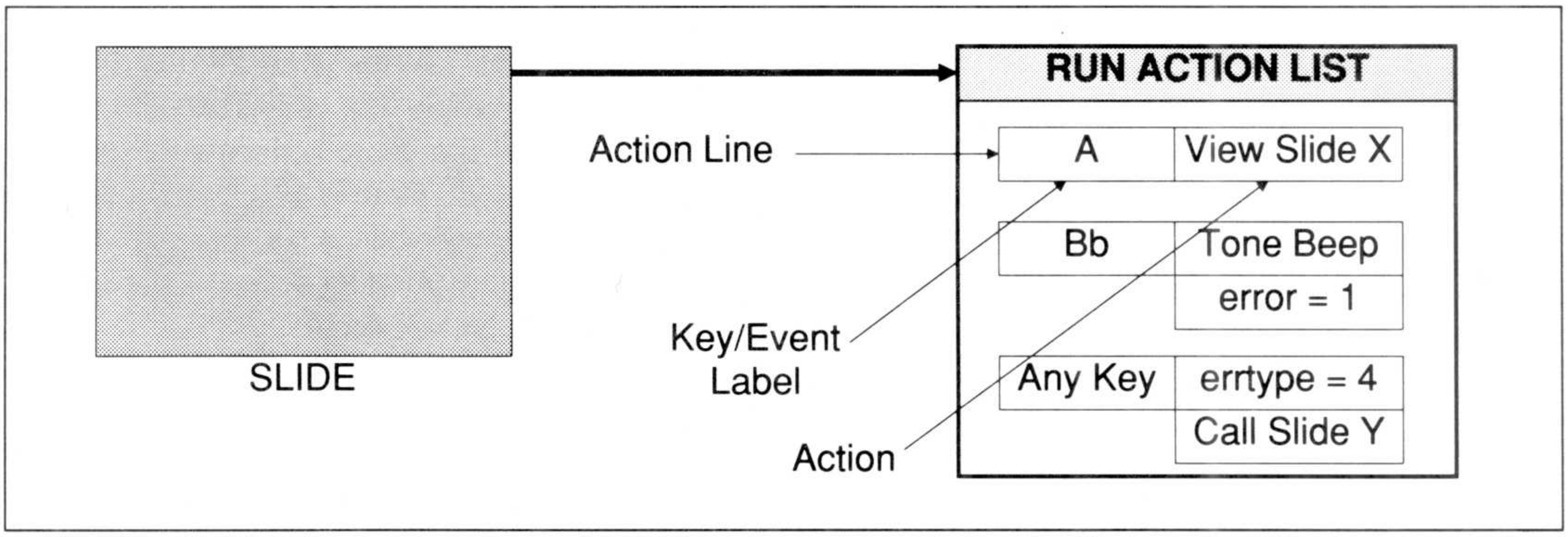


Figure 9. A slide and its Run Action list

The Key/Event label specifies the times when you execute the actions that it labels. When DEMO II processes a key, it starts at the top of the slide's Run Action list and looks for the first Key/Event label that "matches" the key pressed. As you can see from the illustration, in addition to Key/Event labels for each key that you can press, there are labels that can match any one of a group of keys. For example, the Key/Event label "Bb" matches the letter B whether it is uppercase or lowercase ("B" or "b"). The Key/Event label "Any Key" matches any key that is pressed. Since the list is searched from top to bottom, the letter B would match the second Key/Event label, while the letters "C", "c", "D", or even "a", would match the third Key/Event label.

After a match is found, the action on the line with the Key/Event label is executed, along with actions on any succeeding lines up to, but not including, the next line with a Key/Event label. Then DEMO II reads another key and processes it. If an action causes a new slide to be viewed, execution of actions on this slide stops, and whatever is appropriate to the Run Type on the new slide is done.

For example, pressing A in Figure 9 would cause just one action to be executed: the one that viewed slide X. Pressing B or b would sound a tone, set a variable to the value 1 to remember that a viewer error was encountered, and then wait for the user to press another key. (The user wanted the variable "error" set in order test it in a later slide to see if any errors were encountered.) Pressing any other key would set the variable "errtype" to 4, and then view the slide Y, which could return to and re-view this slide at a later point. (Slide Y would probably act differently depending upon the value of the variable "errtype".)

The order of Key/Event labels is important. If you put the set of actions with the Key/Event label "Any Key" first, the other Key/Event labels would never be matched and their actions never executed.

They are called Key/Event labels, and not Key labels, because there are a variety of events that are not triggered by the user pressing keys that you can still match. One of these is called "WaitDone". The Key/Event WaitDone is processed as if it were a key that was pressed immediately after a wait was finished. You can test for this occurrence with a Key/Event label "WaitDone" and take special action (such as sounding a tone to tell the user that it is time to press a key).

Key/Events that do not match any Key/Event label in the Run Action list are processed in a special way. First, a **Global Run Action list** is searched to find a match. If there is a match on the Global Run Action list, those actions are executed. If there is no match on the Global Run Action list as well, there is a Default Action for all Key/Events. The Default Action for keys that are pressed causes the next slide to be viewed. The WaitDone Key/Event reads the next key press by default. The section on "How Running Works" describes these defaults in greater detail.

The Global Run Action list is useful for providing your own defaults. A common Global Run Action list would have the action lines:

```
Esc        Quit
F1         Call Slide Help Slide [0007]
Any Key    Tone Beep
```

This would stop running if you pressed the Esc key; view the seventh slide (which has the name "Help Slide") if you pressed F1, and return to and re-view the current slide at a later point; and just beep at any other keys that do not match Key/Event labels on the slide or in the Global Action list.

In order to create a Run Action list, you use the Run command. To modify the Global Run Action list, you use the Global Run command. In both cases, you insert a new action line with a Key/Event label by using the Run Insert command. By using the Run Line command you insert a new action line with only an action. You can modify the Key/Event label by using the Run Key/Event command, and you can change the action by using the Run Action command. A more complete description of the Run command is in the "Run Commands" section of this manual.

There are over a hundred different actions to choose from. They include the ones in the examples above, as well as programming-type actions such as If, For, While, and Select/Case. There are actions to manipulate strings, read and write files, make a variety of sounds, and control DEMO II internal features. These actions are described in the "Run Actions" section of this manual.

You will find that you rarely use most of the run actions. Which run actions are not used, though, varies from application to application. The most commonly used run actions are: View Slide, Quit, Tone Beep, Call Slide, Slide Return, File, Continue with Default Action, and "=".

Save, Load, Print, and Other Input/Output

DEMO II keeps a lot of information about the slides in memory at one time. All of this can be saved into a single file for later reloading. Some of this information is shown in Figure 10.

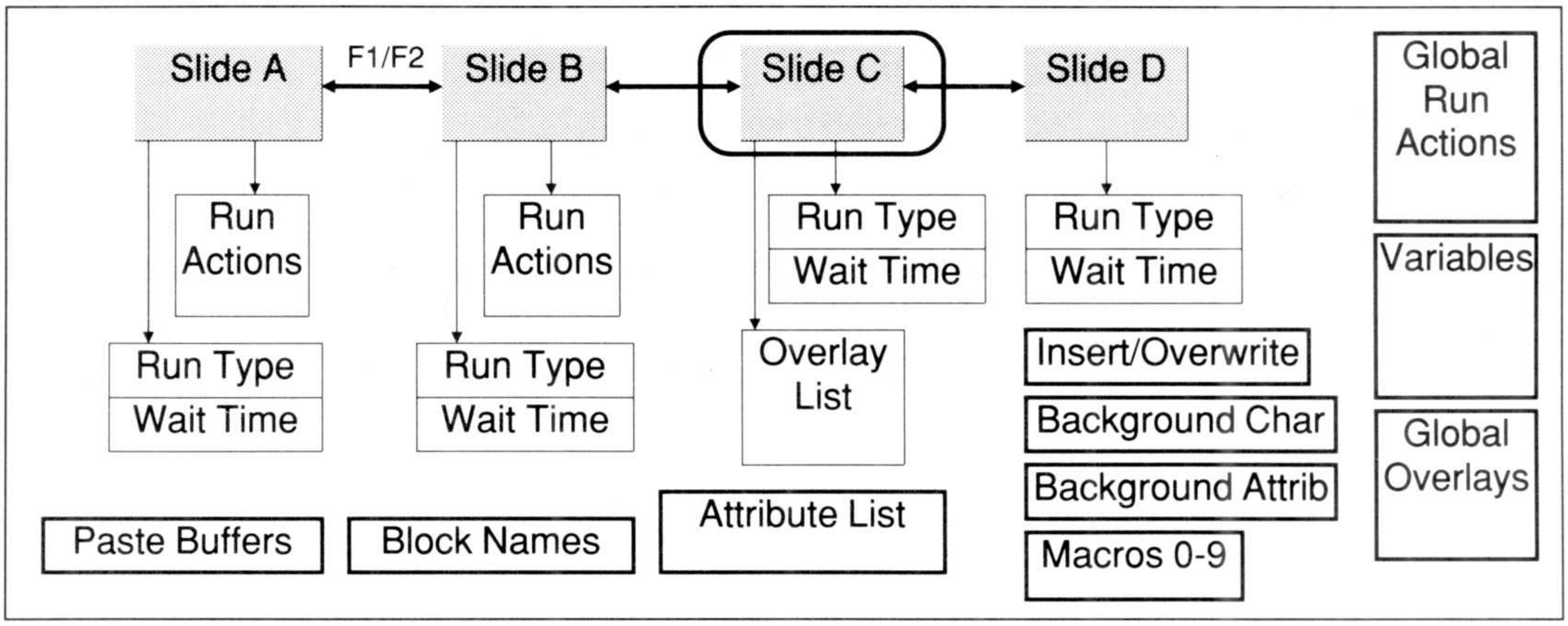


Figure 10. Some of the contents of a saved file

You save all of this with the I/O Save command. When you reload with the I/O Load command, everything comes back as it was when you saved it — the same slide will be on the screen, all the variables will have the same values, and all of the slides will be there as they were.

There is an I/O Add command that adds some of the slides from another save file to the slides currently in memory. As much information as possible about those slides is also loaded, including Run Actions and overlay definitions (if they refer to slides also being added).

An I/O Print command outputs the slides' text and optionally outputs other information (such as Run Actions, Overlay Lists, etc.) to the printer or a file. Any number of slides can be printed, or just the ones that have the Print Flag set (set/cleared with the Slides Print command).

The I/O Retrieve command lets you insert new slides into the current slide show with images that were captured while another program was running (using the "CAPTURE" program). You can also insert new slides with images that are created by reading a normal text file and breaking it into 25 line slides, or you can insert a slide that references a ".PCX" file (the type produced by ZSoft's PC Paintbrush).

The last I/O commands are Code-Read and Write-Code. These commands let you save and load a series of Run action lines. The actions are saved and loaded from the Run Action Paste buffer. You use the Run Delete, Run Copy, or I/O Code-Read commands to put action lines into the buffer. You use the Run Paste command to insert those action lines into a Run Action list. You could save frequently-used program segments with the I/O Write-Code command, and add them to new slide shows with the I/O Code-Read command followed by a Run Paste command.

Tying It All Together

Figure 11 is one of the ways of tying together all of the elements of DEMO II. Your slide shows may not be this complex to start, but hopefully you can see how the run actions are controlled by the keyboard and the passage of time, and in turn how the run actions affect the variables which can change the overlays which can change the screen.

If you don't understand Figure 11, you can ignore it. If it were easy enough to present in one diagram, there would be only one picture in this manual! Some of the early readers of this manual found this diagram helpful, while others did not. It was left in the manual for those of you who do find it helpful.

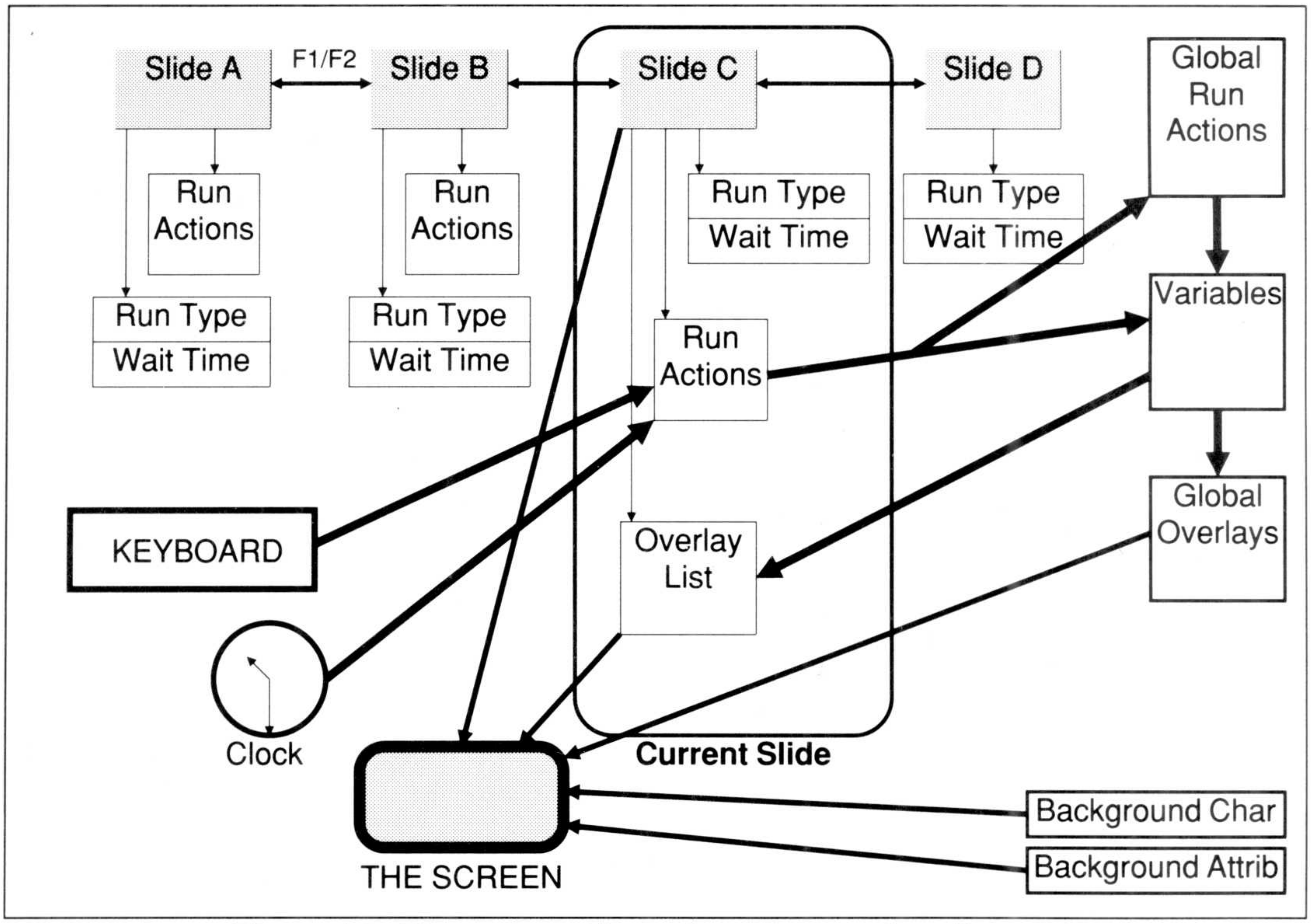


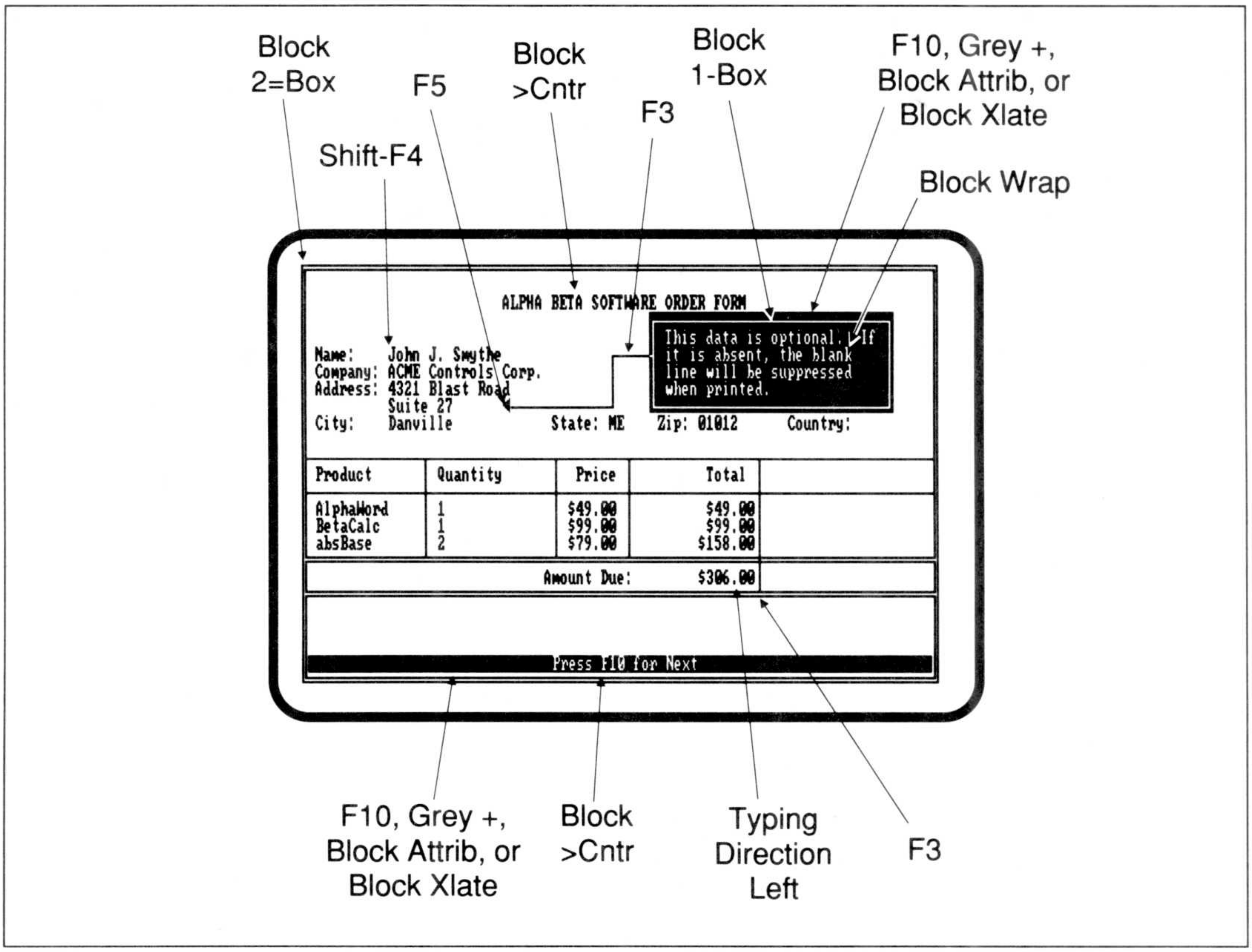
Figure 11. How everything relates (a complex diagram)

The purpose of this section is to point you to the DEMO II features necessary to achieve a variety of commonly desired results. You should be able to get a general idea of the steps you must go through to get to the outcome. You can find further information about the commands and features by looking them up in the Table of Contents and Index.

Some of these techniques are used in the sample files on the Second Diskette. Execute the RUNME command on the Second Diskette for more information.

Screen Mock Up

You can create a mock up of a screen with DEMO II's editing features. Figure 1 is a sample screen with call-outs showing the commands that could have been used to create some of the parts. Detailed descriptions follow.



```
                     ALPHA BETA SOFTWARE ORDER FORM

                                              This data is optional. If
Name:     John J. Smythe                      it is absent, the blank
Company:  ACME Controls Corp.                 line will be suppressed
Address:  4321 Blast Road                     when printed.
          Suite 27
City:     Danville              State: ME    Zip: 01012      Country:

Product     Quantity      Price          Total
AlphaWord   1            $49.00         $49.00
BetaCalc    1            $99.00         $99.00
absBase     2            $79.00        $158.00

                         Amount Due:   $306.00

                           Press F10 for Next
```

Figure 1. A sample screen and how to create it

Block 2=Box
Block 1-Box

You can create a box by marking a block, and then executing a Block 1-Box, Block 2=Box or Block 3_Box command. You start marking a block by pressing F9. You extend the block by using the cursor motion keys. You call up the Block menu, while the block is marked, by pressing either Enter or Esc B. Selecting 2=Box will turn the outline around the block into a double-lined box. Selecting 1-Box will set a single-lined box. Selecting 3_Box will set a heavy-lined box.

Note that a quick way to get to the upper-left corner of the screen is to press Ctrl-Home; a quick way to get to the bottom right is to press Ctrl-End.

Shift-F4

If you are typing a list of items one below the other, you may want to set the left margin. Move the cursor to the column where you want the left margin, and press Shift-F4 or execute the Typing Margin command. From then on, whenever you press Enter while editing, the cursor will move to the left margin on the next row (unless a block is marked, in which case the Block menu will appear).

F5

To type one of the special characters in the PC's character set, such as the arrow heads and international characters, you use the Typing Chars command. Pressing F5 also invokes this command.

Block >Cntr

You can center text between two points by marking a block that contains the entire area and then executing the Block >Cntr command.

F3

You can draw single and double lines with the Typing Lines command. Pressing F3 also invokes this command. You can switch between single and double lines by pressing 1 or 2 while in this command. You "draw" by using the arrow keys, etc. This command knows about line crossings, and it tries to use the appropriate character.

F10, Grey +,
Block Attrib,
or Block Xlate

To have character positions display with inverse and other attributes, you can use any one of several commands. The most common ones are these four. Pressing the F10 key sets all of the character positions within the block (or the cursor position, if there is no block marked) to the first attribute in the Attribute list. By default, this sets the block to inverse video.

The Grey + and Grey - keys cycle the contents of the marked block through the attributes in the Attribute list. See their detailed descriptions in the "Keys" section of this manual.

The Block Attrib command lets you set a block or single character to a specific attribute.

The Block Xlate command lets you change all occurrences of one attribute into another.

Block Wrap

You can word-wrap text within a block with the Block Wrap command.

Typing
Direction Left

The right-aligned figures were typed in by setting the typing direction to "left" and typing the values in backwards, starting at the right-most column. The Typing Direction Down command is useful if you are typing row numbers. You could also right-align text by typing it and then moving it into place. You do this by enclosing it in a marked block and then moving it with the F6 (Block Move) command.

Simple Demo of an Existing Product

A simple demonstration of an existing product would consist of screen images captured while running the existing program, plus additional slides and text created within DEMO II. The slide show would show these images one after another, switching whenever the viewer pressed a key.

CAPTURE

You get the screen images by first running the CAPTURE program, then running the program whose screens you want to capture. Whenever you have a screen image that you want to import to DEMO II, trigger CAPTURE by pressing both shift keys at once. See the "Capture Program" section for more information.

DEMO2
I/O Retrieve

Exit the program being captured and run the DEMO2 program. Execute the I/O Retrieve command, and respond "C" at the "Capture/Text/PCX" prompt. New slides with the images you captured will be inserted after the slide on the screen.

If you need to capture more screens than CAPTURE can handle at once, you can save the slide show (I/O Save), exit DEMO II (Quit), run your program, capture more screens, get back into DEMO II, load the saved, first set of slides, go to the last slide previously retrieved, and execute I/O Retrieve again to get the next group.

Edit
F1/F2

Use DEMO II's editing commands to annotate the captured images and create descriptive text, lines, and boxes. See the "Screen Mock Up" subsection. Use F1/F2 to switch from slide to slide. Note that the retrieved slides will be inserted after the first slide, which starts out blank, so you can put a title on that first slide.

Shift-F3
Shift-F5
Slides Move
Group

You can add new blank slides after any slide by pressing Shift-F3. You can add a new slide that is a copy of a slide by pressing Shift-F5. You can rearrange the order of the slides with the Slides Move command. You may first want to give the slides names with the Slides Name command to make rearranging less confusing. You move a group of slides by moving the highlight in the Slides list to the first slide to be moved, executing the Slides Group command, moving the highlight to the listing for the last slide to be moved, and then executing the Slides Move command.

I/O Save
RDEMO2

To run the slide show, start by putting the first slide on the screen (Shift-F1) and then saving the file (I/O Save). Exit the DEMO2 program and then execute the RDEMO2 program with the saved file's name as an argument (without the ".DBD" extension). After the start-up message, press the Space key. Your first slide should appear on the screen. From then on, to move to the next slide, press any key such as Space or Enter. After the last slide, RDEMO2 will return to DOS.

Copies

To give copies of your slide show to others to run, put a copy of the saved file on a diskette (it has a ".DBD" extension) along with a copy of RDEMO2.EXE. To see the slide show, the viewers must execute RDEMO2 with the appropriate file name argument. You may want to simplify things by creating a batch file to run RDEMO2, or by naming your file "_FILE_0.DBD", which is loaded by RDEMO2 automatically if no file name is given. See the License Agreement at the end of this manual for details about distributing copies of RDEMO2.EXE.

Automatic Running Demo

An automatic running demo shows one slide after another, pausing a predetermined period of time between each slide. Many demonstrations and prototypes have at least some portion set to run automatically.

Create the slides To create an automatic running demo, you start by creating the slides from scratch or capturing them as you would for the simple demo, described in the subsection above.

The next thing you have to do is set each slide's Run Type and Run Wait. You want Run Type 0 which instructs DEMO II to pause the Run Wait amount of time after the slide is displayed, and then automatically to switch to the next slide in sequence.

Run, Type 0 To set a slide's Run Type, call up the Run menu and then execute the Type command. Type the value 0, and then press Enter.

Wait ?? To set the slide's Run Wait value, execute the Wait command on the Run menu. Type the value and then press Enter. The wait is measured in 1/18ths of a second, so a wait of 9 is one half second, 18 is one second, and 36 is two seconds.

If you are not sure which time value you want, you can use the name of a variable instead of the number for the Wait setting. Just type a name with eight or fewer characters when prompted for a Wait value. If the variable has not been previously defined, you will have to indicate its type, which is numeric, by pressing "N". Variables start with a value of zero. You can change the value of a variable by executing the Run Vars command to get the Variables menu, and setting the appropriate value with the Value command on that menu.

F1/F2 OK To switch to another slide in order to change its Run Type and Run Wait settings, use F1/F2 without exiting the Run menu. When you are all done, you can exit the Run menu by pressing Esc or choosing the OK command.

RDEMO2 Put the first slide on the screen (Shift-F1), save the slide show (I/O Save), Quit, and then run RDEMO2, just as you would for a simple demo.

Restarting You often want an automatic demo to restart after the last slide. You can do this by leaving the last slide's Run Type 1, setting the Wait value, and setting a run action to be executed when the wait is done.

You set the run action on the last slide's Run menu by executing the following commands: Insert, press the End key, select "WaitDone", View, move to the first slide (Shift-F1), and then press Enter.

To test the slide show, you can execute the Run command on the Run menu. You can stop the running at any point by pressing Ctrl-Break.

Mixing With Step-By-Step, "Flushing" If you follow an automatic segment in a slide show with one that you advance by pressing a key, you may want to "flush" any keystrokes the viewer inadvertantly made while the automatic part was running. You can do this by setting the Run Type for the first slide in the step-by-step part to 2, instead of the normal 1. Run Type 2 is just like Run Type 1, except that it flushes type-ahead before waiting for input.

Default Run Type If you are about to create a series of slides and you know that they all are going to have the same Run Type, you can use the Global Default Run Type setting. A new slide's Run Type is set to the Default Run Type when it is created. Note that this only affects the Run Type setting when the slides are created — changing the Default Run Type does not change slides already created.

Repeatedly Show a Series of Slides Until a Key is Pressed

Demos that run at trade shows and in stores often need an "attract" mode, where the same series of slides are shown over and over until a viewer comes along and presses a key. At that point more of the demo is shown. The series of slides may be an entire demonstration or just a short eye-catching display.

Run Type 4

To do this, you should set all but the last in the repeating series of slides to have Run Type 4. This will cause DEMO II to switch to the next slide when the wait is done, unless a key is pressed.

Each of those slides should also have a run action of the form:

```
Any Key    View Slide first non-repeating slide
```

Run Type 3

The last slide in the repeating series should have Run Type 3 and the following run actions:

```
WaitDone   View Slide first repeating slide
Any Key    View Slide first non-repeating slide
```

Start/Stop/Backup While Viewing an Automatic Running Demo

While an automatic running demo is useful, you often want to give the viewer the opportunity to stop the demo, back it up, or move ahead quickly.

One specification would be to have PgDn immediately move you ahead to the next slide, PgUp move you to the previous slide, and any other key stop the slide show on the current slide. Whenever the slide show is stopped, a message will pop up telling the viewer how to continue.

Run Type 3,
Wait,
run actions

Each slide that has this effect should have a Run Type of 3, an appropriate Run Wait value, and the following run actions:

```
PgDn       ...                    Do what WaitDone does
WaitDone   stopped = 0            Make sure overlay is off
           View Slide >NEXT<
PgUp       stopped = 0            Make sure overlay is off
           View Slide >PREVIOUS<
Any Key    stopped = 1            Turn overlay on
```

After the AnyKey run action line is executed, DEMO II will wait for another key to be pressed — it does not re-execute the timed wait. See the "How Running Works" section.

Overlays

The Global Overlays list should have the following items:

```
>THIS SLIDE<
Stopped Msg      [????] SLIDE
```

Variable controls overlay

The Overlays Nums Visible setting for the Stopped Msg slide overlay should be the numeric variable "stopped". You can create the list by issuing the following commands:

`Esc G .`	*Call up the Global Overlays list*
`S T`	*Insert a >THIS SLIDE< overlay*
`S`	*Insert a Slide overlay*
`F1/F2, etc.`	*Find the slide with the message to pop up.* *The message should be something like:* *"STOPPED - PgUp Previous, PgDn Next"*
`Enter`	*Select the slide*
`N`	*Get the Overlay Nums menu*
`End`	*Move to the Visible setting*
`Enter`	*Change the value*
`V`	*Set the Visible setting to a variable*
`stopped`	
`Enter`	*If "stopped" wasn't defined, answer "N"*
`Esc Esc`	*Done*

The first and last slides in a sequence will need different run actions so that you do not PgUp/PgDn to a slide that does not exist. You may want to set them to Tone Thud, instead of View Slide >NEXT< or View Slide >PREVIOUS<.

Set the variable "stopped" to 0, before running, by using the Global Variables or Run Vars command to get the Variables menu, and then issuing a Value command.

Have Menu Selections in a Demo

There are a variety of ways to implement menus in a slide show. Three ways will be presented here.

Single Slide Menu

The simplest type of menu has a list of choices, each with a letter. When the viewer presses a key, DEMO II switches to the appropriate slide.

One Slide, Run actions

To implement this type of menu, you have one slide showing all of the choices. The run action list for the slide would have one run action line for each choice, specifying what to do. For example, if the choices were A, B, C, and D, the run actions could be:

Aa	View Slide choice A
Bb	View Slide choice B
Cc	View Slide choice C
Dd	View Slide choice D
Any Key	Tone Beep

Moving Cursor Menu

A more advanced menu would have one of the choices highlighted. The arrow keys would move the highlight. The viewers indicate their choice by highlighting the appropriate item and pressing Enter, or by pressing a key associated with the choice, as you would for the simple menu previously described.

Slide per choice, Shift-F5, F10

For example, to add the moving highlight to the previous simple menu, you would have three more copies of the slide showing the choices (Shift-F5). On the first slide you would highlight the A choice (F10), on the second you would highlight the B choice, etc. You would then add the following run actions *above* the Any Key run action:

Up	View Slide >PREVIOUS<	*Use Tone Thud on first slide*
Down	View Slide >NEXT<	*Use Tone Thud on last slide*

Single Slide With Overlays

Another way of implementing the same type of menu is to have a single slide with all of the text common to the choices. Following this menu slide are choice slides containing the parts that are different for each choice, such as a highlight in the correct place, possibly with a description.

This method uses overlays and makes use of more DEMO II features. Make sure that you understand overlays before trying this method. The advantage of this method is that you only need one slide with run actions no matter how many choices there are. If you understand how this method works, you have probably mastered most of the more difficult features in DEMO II. If you find this method too difficult, you can get the same effect with the previous method, so do not worry.

RELREF Overlay

The appropriate choice slide is shown as an overlay along with the menu slide. Which slide is shown is controlled by the variable "cmdnum". You do that by using a RELREF overlay. The arrow keys are set to modify the value of "cmdnum".

To create the overlay, put the menu slide on the screen and type the following:

```
Esc O V            Create a Value overlay
c m d n u m        The value is variable "cmdnum"
Enter              Press "N" at the prompt if undefined
Enter              Leave it where it is
Enter              Set the numbers for the overlay
End                Change its type
Enter Enter Enter  Switch it to a RELREF-type value overlay
Esc Esc            Done
```

The menu slide should have run actions similar to the following:

```
Viewed     cmdnum = 1                   Initialize cmdnum
           cmdstr = "ABCD"              Get a list of the choices
           cmdmax = Length(cmdstr)      Get maximum number of choices
           Continue With Default Action
Up         If (cmdnum >= 2)             Backup if not on first item
              Decrement cmdnum by 1
           -Else
              Tone Thud
           End-If
Down       If (cmdnum < cmdmax)         Move ahead if not last item
              Increment cmdnum by 1
           -Else
              Tone Thud
           End-If
Enter      key = cmdstr[cmdnum]         Get appropriate choice letter
           Transfer to Key/Event key  Pretend it was typed
Aa         View Slide choice A
Bb         View Slide choice B
Cc         View Slide choice C
Dd         View Slide choice D
Any Key    Tone Beep
```

The variables "cmdnum", "cmdmax", and "key" are numeric; "cmdstr" is a string variable.

You can use DEMO II's Run Copy and I/O Write-Code commands to save a copy of this code after you have entered it once. To use it in another slide show, do an I/O Code-Read and then use the Run Paste command. You will only need to customize the string assigned to cmdstr, the Key/Event labels on the View actions (Aa, Bb, etc.), and the slides referenced by those View actions.

Echo Viewer Typing

Normally DEMO II reacts to one keystroke at a time when running a slide show. Also, the text on the screen is usually all prepared in advance by the slide show creator. There are times when you want to allow the viewer to type in text and have that text displayed. An example would be when you are simulating data entry into a form.

To echo viewer typing you use DEMO II's variable and overlay features. You have the characters pressed by the user accumulate in a string variable and have that string value display with the slide.

String Value Overlay

To have a string variable's value display with the current slide, do the following:

`Esc O V`	*Create a Value overlay*
`i n p u t`	*Give the variable's name*
`Enter`	*Press "S" at the prompt if undefined*
`arrow keys`	*Position the sample where you want the text to appear. If you do not see the "1...VAL.." line, you may need to cut transparent holes — see the "Overlays" subsection in the "Overview" section.*
`Enter`	*Confirm position*
`Enter End Enter`	*Set the type to "w/ H/W Cursor"*
`Esc Esc`	*Done*

Process-Char Run Action

To process the keys pressed by the viewer, use run actions of the form:

`Enter`	`View Slide >NEXT<`	*What to do when done*
`Any Key`	`Process-Char input`	*Otherwise use this action*

The Process-Char run action adds the character that was typed to the string variable's value. If the key pressed was a backspace, the last character in the string is removed. All keys, other than characters and backspace, are ignored.

You can have more than one variable's value displayed this way on a slide, and you can have the same value displayed on more than one slide. In most cases you will want the variable whose value you are typing into to have a "w/ H/W Cursor" overlay, and the others to have just a plain String Overlay.

Test Viewer Typing

You can branch to different slides depending upon what the viewer entered into a string variable by using either the If or the Select actions. The If action can do both uppercase- and lowercase- sensitive or insensitive comparisons. The Select action is uppercase- and lowercase- insensitive.

If you set up a slide as specified in the "Echo Viewer Typing" subsection, you could test for "First", "Second" and "Third" by using the following run actions:

```
Viewed     Erase input        Clear the string before displaying
           Continue with Default Action
Enter      Select (input)
              Case "first":
                 View Slide choice A
              Case "second":
                 View Slide choice B
              Case "third":
                 View Slide choice C
              Otherwise:
                 Tone Beep
           End-Select
Any Key    Process-Char input
```

How Running Works

In order to produce a realistic simulation, a powerful demonstration, or to have the viewer "learn by doing" with computer based training (CBT), there are several things that must be tied together. You want to control which slides are displayed, change the values of variables to reflect data entered, etc. This should be in response to the keyboard and perhaps also to timing. You want the slide show "run" in a manner that produces the effect you desire. This automatic control of the system is called "running the slide show". Figure 1 gives you some idea of the items that are controlled and what controls them.

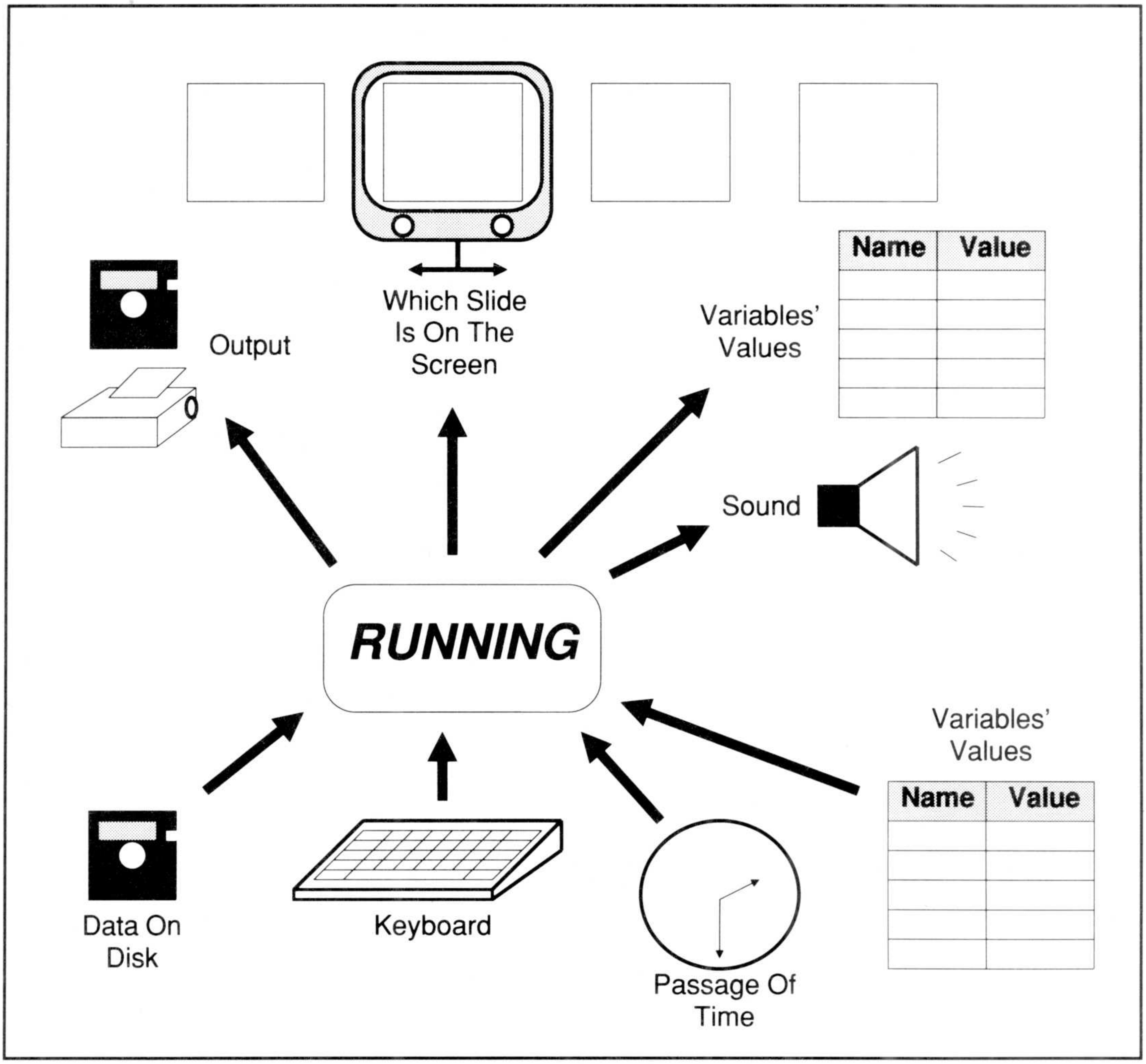


Figure 1. Overview of running a slide show

DEMO II has a method of controlling running that simplifies many of the normal operations needed to produce a slide show. It also has added power, with the help of its built-in programming language, to let you customize the operation to a very fine degree. This section describes running in detail.

Executing Actions

You control running through the use of ***actions***. Actions are commands to DEMO II such as "make a beep sound" or "switch to the startup slide." Actions are stored in action lists that contain one action per line.

When you are running a slide show, DEMO II executes actions. There is usually a set of actions from a run action list (comprising one or more items in the list, all one after another), and those actions are executed sequentially, as in Figure 2. Certain actions cause the program to stop executing this set and to find another set of actions. When the end of the set is encountered, DEMO II also looks for another set to execute.

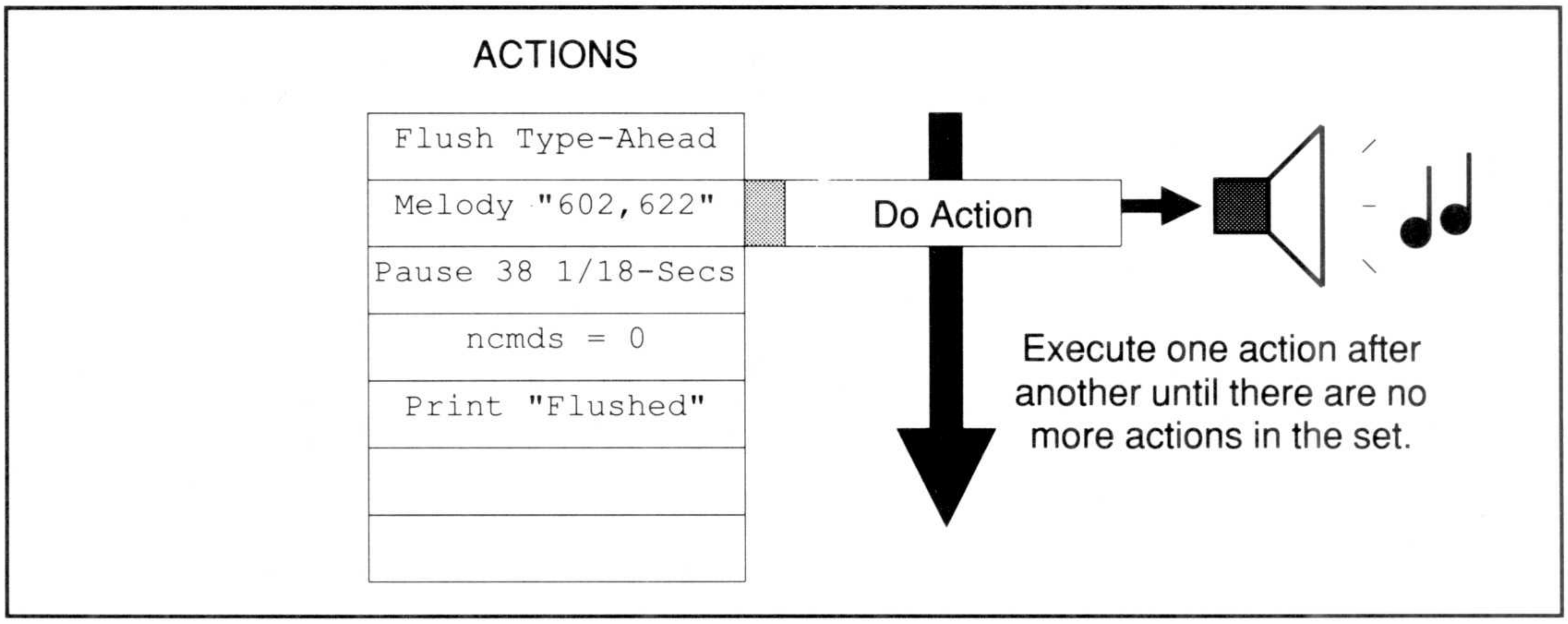


Figure 2. Execute a set of actions

The sets of actions in run action lists are labeled with a ***Key/Event Label***. The label is shown on the left side of the first action line in the set. The set extends until the start of the next set (an action line with a Key/Event label) or the end of the list, whichever comes first. Key/Events correspond to occurrences that happen while running, such as the pressing of a particular key on the keyboard.

An action list might look like Figure 3.

DEMO II finds a set of actions to execute by ***signaling*** a Key/Event and then finding a set of actions whose Key/Event label ***matches*** the Key/Event being signaled. A Key/Event label matches the signaled Key/Event when it is the same as the signaled Key/Event or when it is a superset of the signaled Key/Event.

For example, the Key/Event "A" matches the Key/Event label "A". Both correspond to pressing the uppercase letter "A" on the keyboard. The Key/Event "A" also matches the Key/Event label "Aa" (meaning both uppercase and lowercase "A"), as

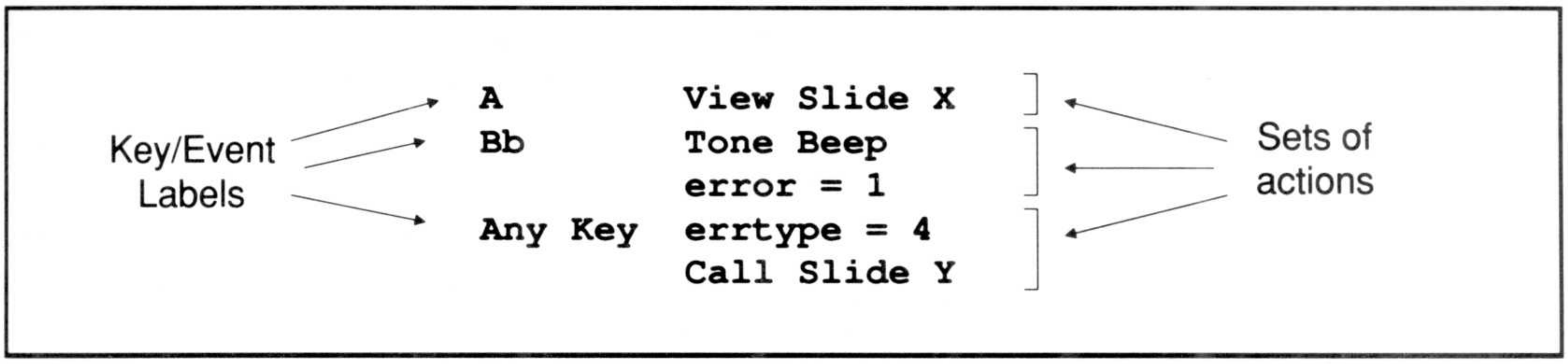


Figure 3. Run Action List with Key/Event Labels

well as the Key/Event labels "Any UPPER", "Any Letter", "Any Ltr/Num", "Any Printing", "Any Key", and "Anything!".

The run action list is searched for a match from top to bottom. The first set of actions whose Key/Event label matches the signaled Key/Event is chosen and executed. See Figure 4. You must insure that the sets of actions are ordered correctly. Superset Key/Event labels that occur above more specific labels will keep the more specific labels from being matched. Be especially careful of Key/Event labels such as "Any Key" and "Anything!". (In general, never use "Anything!" — explanation later).

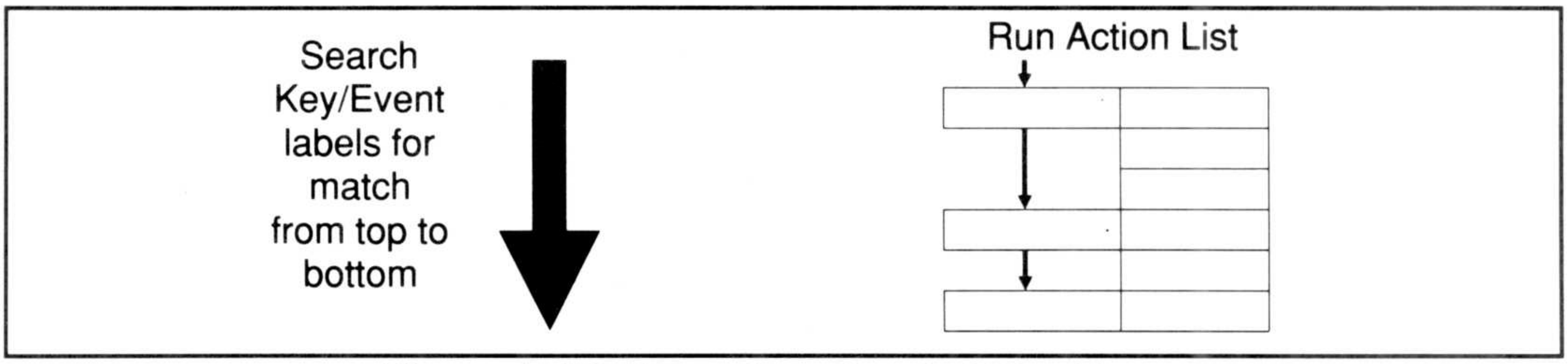


Figure 4. Find match to signaled Key/Event

Let's look at an example. Suppose you want to view the slide "Help" if the viewer presses F1, view the next slide if the viewer presses Enter, and make a "thud" sound if the viewer presses the backspace key. Also, if any other key is pressed, you want to make a "beep" sound, and then increment a count of the number of errors (a variable called "errcount"). You could use the following run action list:

```
F1          View Slide Help [0057]
Enter       View Slide >NEXT<
Bkspace     Tone Thud
Any Key     Tone Beep
            Increment errcount By 1
```

Pressing the F1 key would signal the "F1" Key/Event. That would match the first action in the list. Executing that action would terminate running on this slide and switch to the Help slide. Pressing Enter would view the slide after the current slide by executing the "View >Next<" action. Pressing the backspace key would signal the

"Bkspace" Key/Event and would execute the third action line. Any other key would sound a beep and increment the variable "errcount". Finally, any other Key/Events would not match any of the Key/Event labels.

How does DEMO II decide which run action list to search? It has a very specific method, which is designed to give you the most flexibility.

Associated with each slide is a run action list. When a Key/Event is signaled, this list is always searched first. If no match is found on the slide's run action list, then the Global Run Action list is searched. If no match is found there, then a default operation is performed.

One of the actions that can be executed is "Use Actions On Slide". This causes the search for a match to continue with another slide's run action list. If no match is found there, then the Global Run Action list is searched. There is also an action called "Use Global Actions" that causes the search to continue with the Global Run List. See Figure 5.

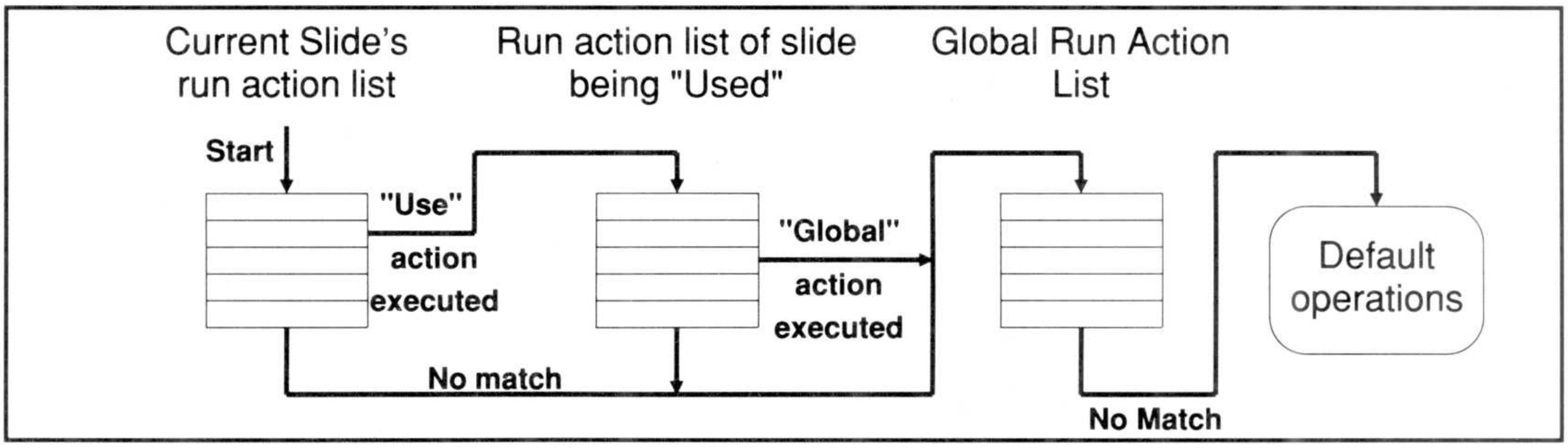


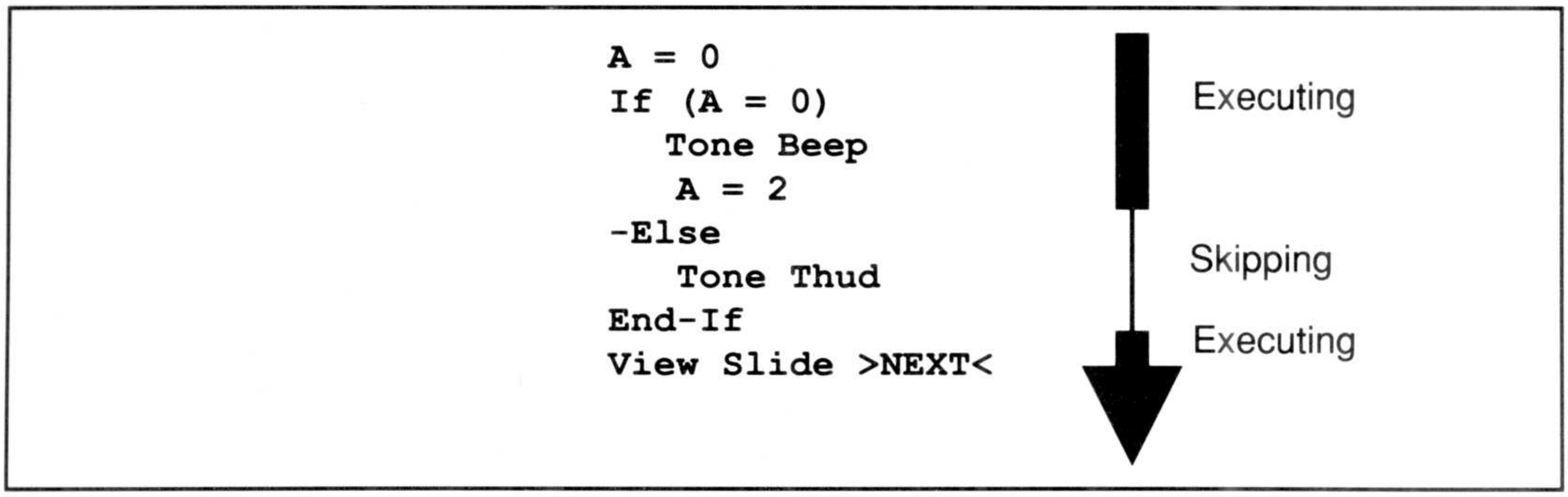
Figure 5. Run action lists searched to find match

Conditionals, Loops, and Tags

Executing a set of actions is similar to executing statements in many programming languages. A difference exists in how the set of actions is chosen (by matching the Key/Event label). We will discuss Key/Events in more detail in a later subsection, but first we must look at the "programming" part of this language.

One of the features that gives programming languages their power is the ability to test conditions and perform different actions, depending upon the results of the test. For example, you might want to switch to a different slide when a key is pressed, depending upon whether the viewer had seen this section of a tutorial previously. Another powerful feature is looping, in which a series of statements is executed repeatedly until some condition is met. For example, you might want to sound a tone every second until a key is pressed.

DEMO II has actions to both test conditions and implement several types of looping. These behave in a manner that should be familiar to most people with a knowledge of programming. There is also a simple "subroutine calling" or "GOSUB" feature called tag-calling, which is described later.



```
A = 0
If (A = 0)
   Tone Beep
    A = 2
-Else
    Tone Thud
End-If
View Slide >NEXT<
```

Figure 6. Skipping actions in a conditional

The "IF" actions all test a condition, such as "is one value equal to another?", or "is the Ctrl key pressed?". If the condition is "true", then the next actions in sequence are executed. If the condition is not "true", then the actions that follow it are skipped until an "End-If" or an "-Else" action is encountered. This is described in greater detail in the "Run Actions" section of this manual. The important concept here is that you can skip over actions depending upon the results of testing a condition. See Figure 6.

While actions are being skipped or executed after an IF action, DEMO II remembers that it is processing an IF action. IFs and other actions, such as the looping actions, can be nested. This means that, within the actions being executed between an IF and and End-If action, you can have other IF/End-If groups of actions. DEMO II remembers when it has not yet encountered an appropriate End action by keeping track of nested conditionals, loops, and Key/Event calls. It remembers this information by using what is known as the **Run Stack**. All nesting information is kept in the one stack. The Run Stack can have a maximum of 99 items at any given time, so you can only nest to a depth of 99, adding all IFs, loops, and Key/Event calls active at once. As soon as the matching End or Return is executed, it is removed from the Run Stack, making room for more.

Nesting must be done carefully, in a manner similar to most programming languages, such as "C". This means that IFs must be entirely contained within the "true" or "false" sections of other IFs. For example:

```
If (a = 0)
   If (b = 0)
      Tone Beep
   End-If
-Else
   Tone Thud
End-If
```

is allowed.

The following is not allowed:

```
If (a = 0)
   If (b = 0)
      Tone Beep
-Else
   End-If
   Tone Thud
End-If
```

Note that the indenting shown above, which is usually provided automatically by DEMO II, only exists to make the listing more understandable to people. The indenting is ignored by DEMO II when determining nesting while running.

Transfering out of an IF or loop with a View action, or by signaling a Key/Event, or by using the Goto Tag action, clears the Run Stack. After signaling a Key/Event or using Goto Tag, you cannot return to where you were in the middle of the IF or loop.

See the Run Actions section for descriptions of the "For", "While", "Block" and "Select" actions, in addition to the description of IF.

In addition to looping and conditionals, you can also explicitly transfer control to running another set of actions. This is similar to "GOTO" and "subroutines" in other programming languages, but with a special DEMO II-flavored twist.

You can transfer to other sets of actions by signaling a Key/Event or using the "tag" mechanism. Signaling a Key/Event causes DEMO II to search for a match to the Key/Event specified. The search is done in the normal way described above. When a match is found, processing continues with the set of actions found there. If no match is found, the normal default is taken for that action.

The "tag" mechanism is similar to the normal Key/Event signaling and matching mechanism. There is a special type of Key/Event called a "**tag**". Associated with each Tag Key/Event is a constant character string of one to eight characters. A Key/Event label for the Tag Key/Event also has a one-to-eight character string, called the "tag name". The label is used like a "statement label" in other programming languages, except that searching is done with the normal DEMO II Key/Event search mechanism. The Key/Event label matches a Tag Key/Event if the character strings are the same or if the label is for "Any Tag" or "Anything!". Note that variable tag names are not allowed — they must be constant strings to result in a match. Tag Key/Event labels always appear alone on an action line; there is a "nothing" action associated with them. When matched, execution continues with the actions following them, as is normal.

There are three ways of transferring to a set of actions with a Tag Key/Event label. You can use the "Goto Tag" action, the "Call Tag" action, and the "View Slide Then Tag" action. The "Call Tag" action adds an item to the Run Stack, and you can return to the action following the "Call Tag" action by issuing a "Return From Tag Call" action at a later point.

Using tags has some advantages over the normal "statement labels" used in other programming languages. DEMO II searches for the tag, not only on the current slide's run action list, but also in the run action lists of any slides that are "Used" and on the Global Run Action list. This lets you share sets of actions between many slides by putting them on the Global Run Action list or Using them on a common slide. You can also have global actions transfer to sets of actions that are different on each slide. This is done by putting the actions on each slide's run action list, with the same tag name on each slide. The searching mechanism will find the correct one for the current slide. See Figure 7.

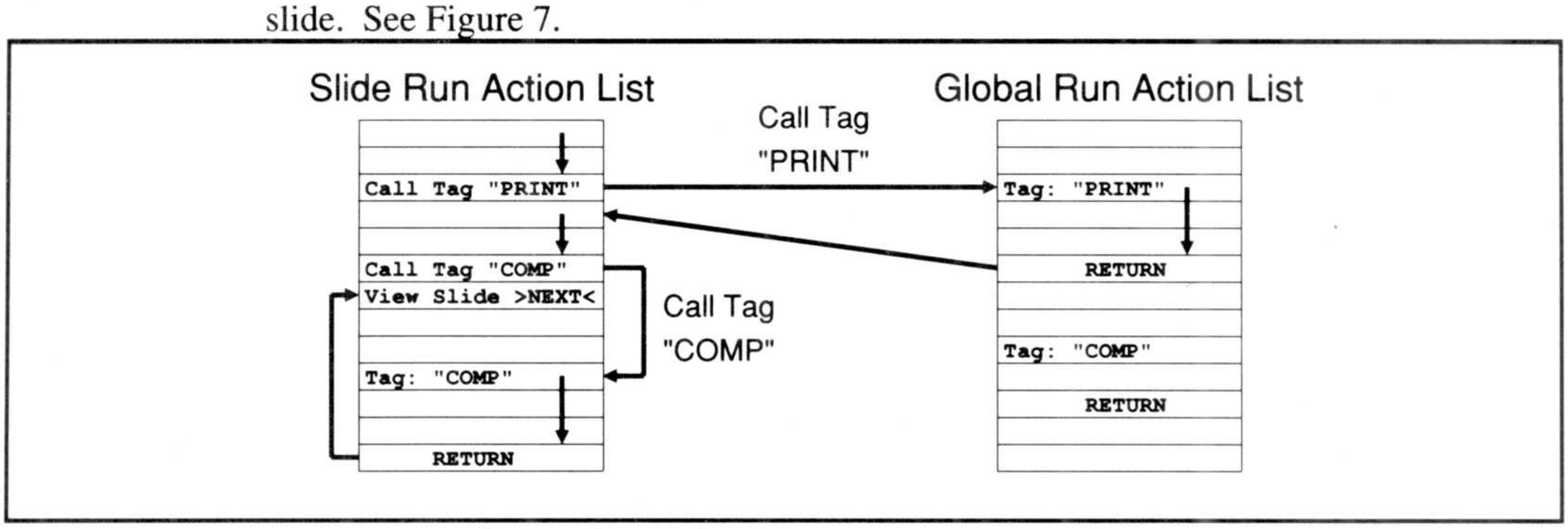


Figure 7. Using Tags to get around

Note that, unlike normal Key/Event labels, Tag Key/Event labels do not end execution of a set of actions when those actions are encountered during execution. Execution continues with the action following the Tag Key/Event label. This is similar to statement labels in most languages and is the way you want the operation to occur in most cases. If you want to end execution after a set of actions that is followed by a Tag Key/Event label, use the "Next Key" action. If you do not want to have a normal Key/Event label end execution of actions preceding it, have the action immediately above the Key/Event label be the "..." action.

There are many examples of conditionals, loops, and tags in the sample files included in the package with DEMO II. Also, some of the examples of actions throughout this manual may aid you in understanding their use.

Interactions With The Viewers: Viewed, Keypress, and More

DEMO II's signaling mechanism is very well suited to the processing of key presses by the viewer. All that you need to do is to have different Key/Event types associated with each of the keys a viewer could press, and to search the run action lists to find out what to do with each keystroke.

Unlike most programming systems, in which you have to write a program to read from the keyboard, process the input, do output, etc., DEMO II has the main "read, process" loop built in. The "output" part is done by just switching the slide that is being shown. This is all implemented using the Key/Event signaling and running mechanism.

A simplified view of running a slide show could be similar to the one in Figure 8, shown below.

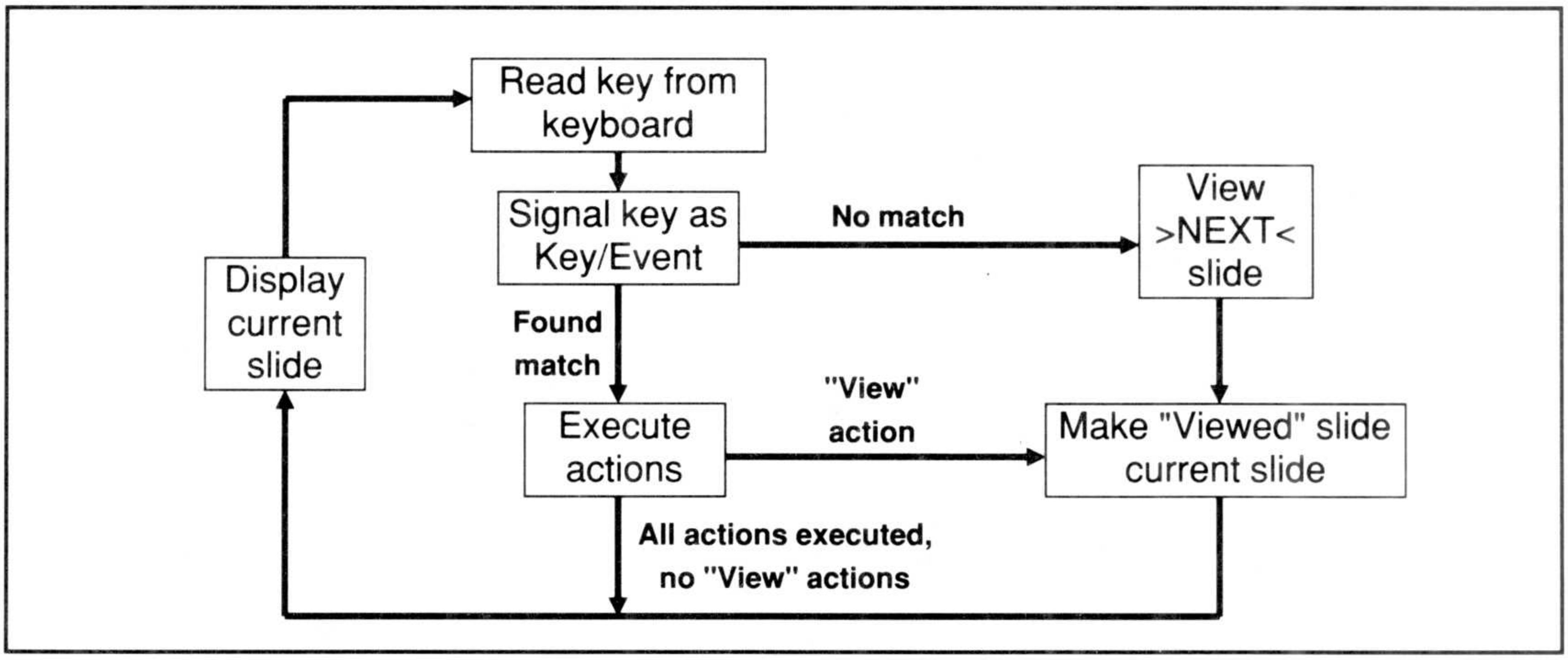


Figure 8. Simplified description of running

This simplified description of running is sufficient for many applications. You can actually use DEMO II assuming that this is how it works, and DEMO II will operate as expected. The default settings will make this "obvious" way of running occur.

Things get a bit more complicated when you start thinking about what operations you want to occur when a slide is viewed. Some operations could be very common, and writing them out as a series of actions each time a slide is viewed could be very tedious. Others involve changing the way the screen looks by modifying variables associated with overlays and then redisplaying the screen.

Think of viewing a slide as having the following standard progression (see Figure 9): When the view action is executed, the slide is made the "current slide" and displayed. Then a key is read, and the key is processed using the Key/Event signaling and action running mechanism. Displaying the slide is also called redisplaying the slide, because its latest version with overlays and associated variables is shown.

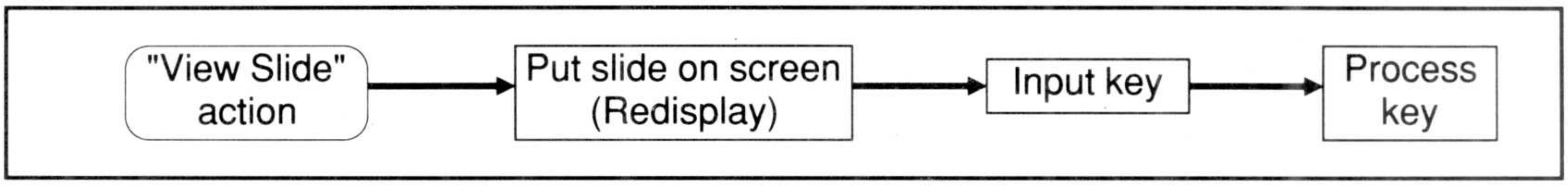


Figure 9. Viewing a slide "standard" progression

Now let's look at what you might want to do at each of the points in this "standard" progression.

Suppose you had a slide with a String Value overlay, and you wanted to give the user up to five seconds to type something after it was displayed. If a key were pressed before the five seconds were up, you would process the key. If not, you would view a slide showing an error message. This could be represented by Figure 10.

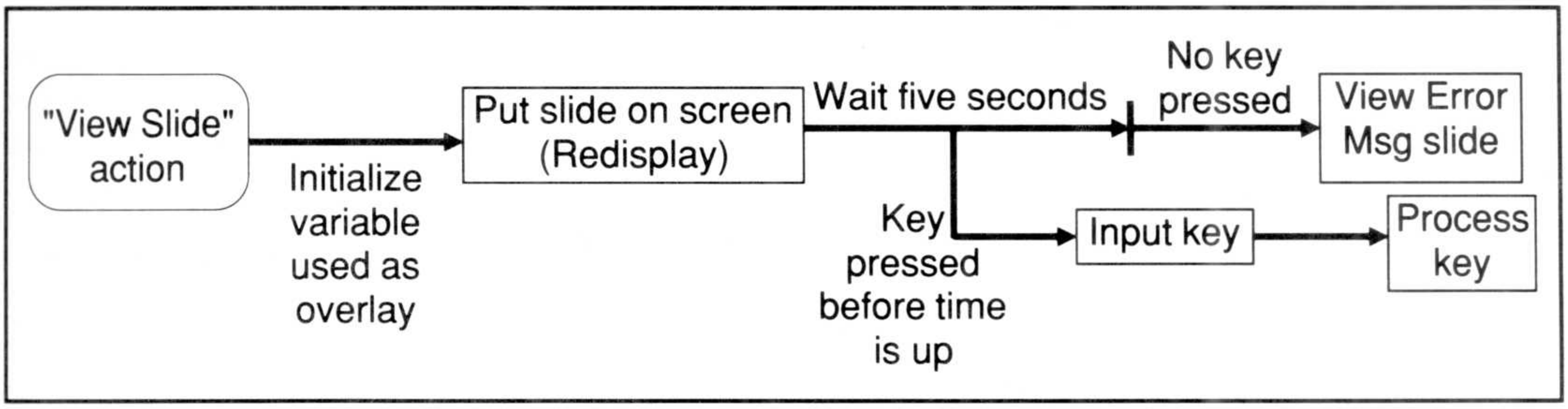

"View Slide" action
Initialize variable used as overlay
Put slide on screen (Redisplay)
Wait five seconds
No key pressed
View Error Msg slide
Key pressed before time is up
Input key
Process key

Figure 10. Typical requirement

DEMO II has broken the viewing process down into six atomic operations. Each operation has a standard default. Each operation can be redefined by you on a global or slide-by-slide basis.

The operations are: Viewed, Displayed, WaitDone, Readkey, Keypress, and Process Key. The Key/Event signaling mechanism is used to switch from one to another. (Their Key/Event names are "Viewed", "Displayed", etc.) The operations and what they do by default (if there is no match) are listed in Figure 11.

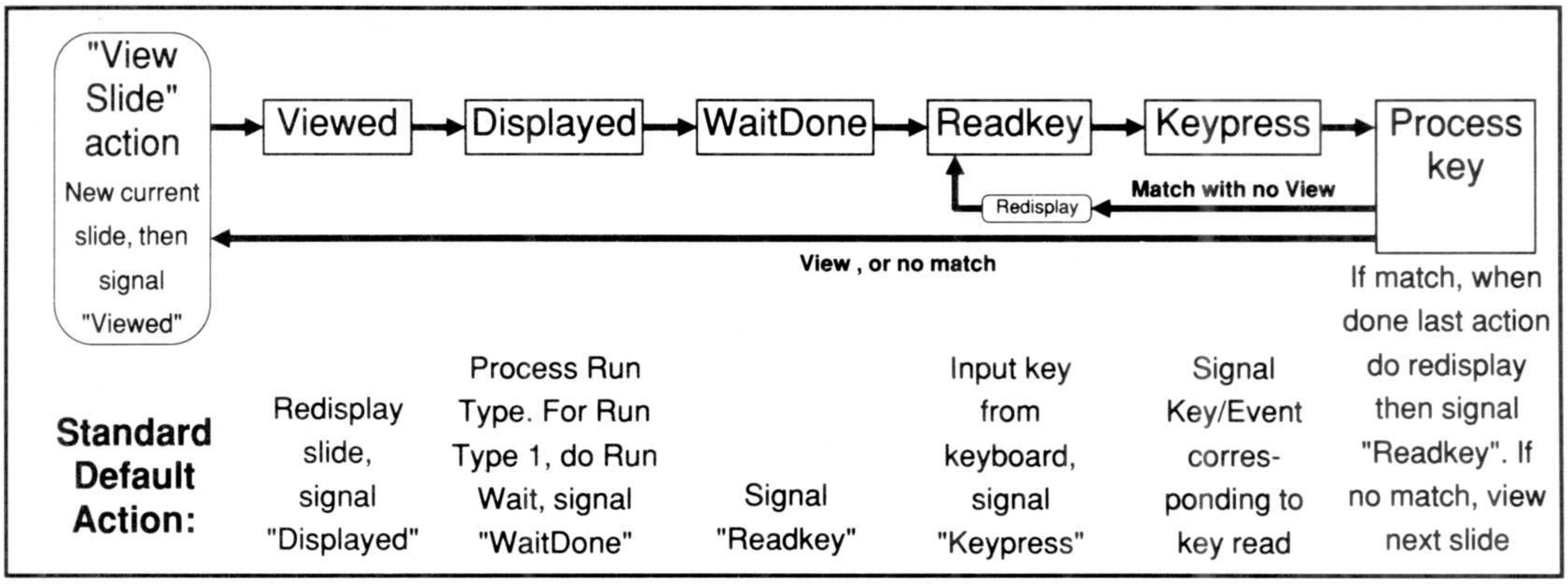

"View Slide" action
New current slide, then signal "Viewed"
Viewed
Displayed
WaitDone
Readkey
Keypress
Process key
Redisplay
Match with no View
View , or no match
If match, when done last action do redisplay then signal "Readkey". If no match, view next slide
Standard Default Action:
Redisplay slide, signal "Displayed"
Process Run Type. For Run Type 1, do Run Wait, signal "WaitDone"
Signal "Readkey"
Input key from keyboard, signal "Keypress"
Signal Key/Event corres-ponding to key read

Figure 11. Atomic operations comprising "Viewing"

There are benefits to having the viewing process work this way. Since the progression from one operation to another is done by using the signaling mechanism, you can get control at any point and insert additional actions or even do things differently.

Normally you do not provide action sets with Key/Event labels for "Viewed", "Displayed", "WaitDone", "Readkey", or "Keypress". You usually let nothing match those Key/Events so that the default actions will occur.

Important! Note that the "Anything!" Key/Event label matches all Key/Events, including those associated with running. If you use the "Anything!" Key/Event label, make sure that you know what you are doing in order not to interfere with viewing. A common acceptable use of "Anything!" would be in a Use action. An action such as Tone Beep, not followed by a "Continue with Default Action" action, could cause the slide show to display strangely. Often what you mean is "Any Key", not "Anything!". For that reason, "Any Key" is at the end of the Key/Event list, with "Anything!" placed where you are unlikely to select it by mistake.

There are times when you do want to use the viewing Key/Event labels. Let's look at a few.

As a very simple example, we will start with the following action line:

```
          Melody "602,622"
```

This action plays two notes on the PC's speaker, and then continues with the next action in sequence.

To play the notes every time a particular slide is viewed but *before* it is displayed, you could put the following on the slide's run action list:

```
Viewed    Melody "602,622"
          Continue with Default Action
```

When the slide is viewed, the "Viewed" Key/Event is signaled. The Key/Event would match the "Viewed" Key/Event label, and the set of actions would be executed. The two notes would sound, and then the second action would tell DEMO II to execute the standard default action, which is to redisplay the slide and then signal "Displayed". If the second action were missing, DEMO II would act as it normally does when it finishes executing a set of actions: do redisplay and signal "Readkey".

You could get the same effect by using the actions:

```
Viewed    Melody "602,622"
          Redisplay Screen
          Transfer To Key/Event 434 (Displayed)
```

Here you have user-defined actions that do the same thing as the built-in, standard default actions. You may want to check the definition of these actions in the "Run Actions" section of this manual. The run actions are presented there in the same order as on the menus. You can find them alphabetically in the Index.

If you want to play the notes after the slide is displayed, you could use:

```
Displayed Melody "602,622"
          Continue with Default Action
```

To play the notes when DEMO II is just about to read from the keyboard, you could use the actions:

```
Readkey   Melody "602,622"
          Continue with Default Action
```

Note that this will make the sound each time a key is to be read. If the key does not result in the viewing of another slide, the "Readkey" Key/Event will be signaled again by the Process Key operation, thus executing these actions again.

Finally, to make the sound each time *after* a key is read, you could use:

```
  Keypress  Melody "602,622"
            Continue with Default Action
```

If you do not have the second action, the key will not be processed. The "Continue with Default Action" action tells DEMO II to do what a "no match" would have done. In this case, not doing the default for "Keypress" would cause DEMO II to do what it normally does when it finishes executing a set of actions: do a redisplay and then signal "Readkey".

Run Types, Run Wait, and "WaitDone"

One part of running remains to be discussed: the "Run Type" setting associated with each slide.

There are a variety of standard ways that you will want to have slides behave when they are viewed. One way has already been described above as the "obvious" way of reading keys, processing them, and viewing the next slide on a "no match". Another way is to display the slide, wait a specified amount of time, and then view the next slide. Yet another would be to view a slide, and wait up to a specified amount of time. If no key is pressed while waiting, you would view the next slide. If a key is pressed before the time is up, then the key would be processed.

Since these are such commonly desired ways of running, it would be a poor design for DEMO II to require you to explicitly put in the actions necessary to implement them on every slide. Instead, DEMO II provides two values associated with each slide: the slide's Run Type and the slide's Run Wait value. The Run Type can be tested to determine how to treat each slide individually. Five different Run Type values have been predefined and are implemented by the default viewing operations.

The different Run Types built into DEMO II are implemented in the default "Displayed" Key/Event. The default "Displayed" code gets the current slide's Run Type value, which can be a numeric constant or variable. It then starts waiting the number of "clock ticks" (1/18 seconds) specified by the Run Wait value and does one of the following:

Type 0 After the wait time has elapsed, it views the next slide.

Type 1 After the wait time has elapsed, it signals "WaitDone".

Type 2 After the wait time has elapsed, it flushes (discards) any keys already typed ahead, and then signals "WaitDone".

Type 3 If a key is pressed before the wait time has elapsed, it terminates the wait and signals "Readkey". (The key is not read by "Displayed", just the fact that it was pressed is sensed.) If no key is pressed before the wait time has elapsed, then it signals "WaitDone".

Type 4 If a key is pressed before the wait time has elapsed, it terminates the wait, and "Readkey" is signaled. If no key is pressed before the wait time has elapsed, then it views the next slide.

Figure 12, below, shows a more explicit version of the illustration about Run Types which appears in the "Overview" section.

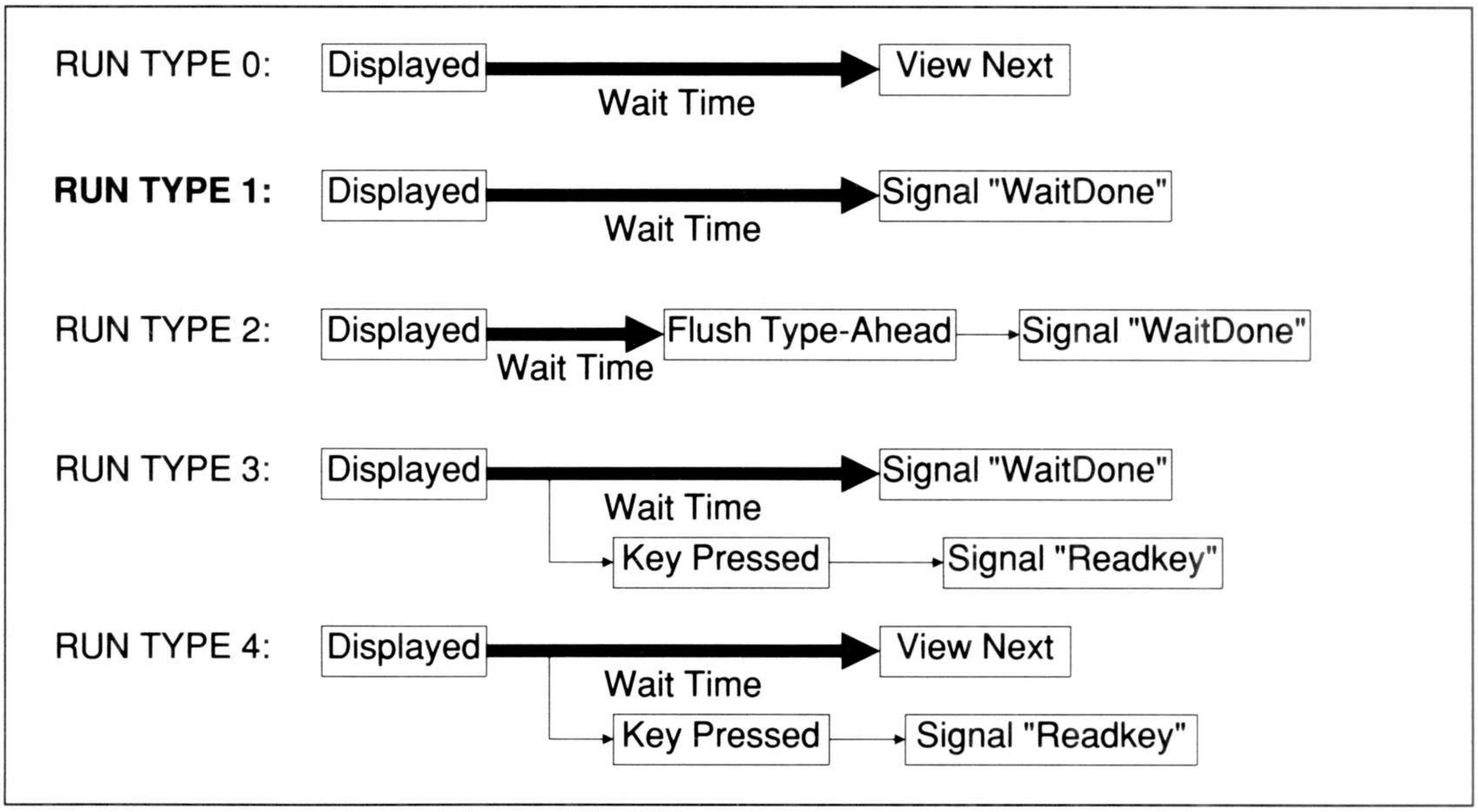


Figure 12. What the Run Types Do

Normally, Run Type 1 is the default. You can change the value that is automatically given to a new slide by changing the "Default Run Type" value on the Global menu.

Run Type 0 is useful for automatically going through a series of slides, perhaps simulating a program in operation or moving through descriptive text in a "self-running" demo.

Run Type 2 lets you say "Press Space to Continue" and not have a key pressed prematurely by an impatient viewer skip past a slide.

Run Type 3 lets you process a viewer's key immediately. You can also distinguish between a time-out and when a user presses a key.

Run Type 4 can be used to go through a series of slides automatically and at a predetermined pace. It also lets the user speed things up by pressing a key, since the default for pressing a key is to view the next slide.

You can create your own Run Type definitions. The Get Builtin action can be used to access various DEMO II values, including the Run Type and Run Wait values of the current slide.

For example, suppose you wanted a Run Type 5 that would have the wait *before* the slide was displayed, not after. This could be used to simulate slow response to the keyboard, such as after the Enter key on an on-line system.

You could put the following actions on the Global Run List:

```
Viewed        runtype = Builtin(-2)
              If (runtype != 5)
                 Continue with Default Action
              End-If
              runwait = Builtin(-1)
              Pause runwait Even If Key Pressed
              Redisplay Screen
              Transfer To Key/Event 393 (WaitDone)
```

These actions get the Run Type value for the slide being displayed. If it is not 5, then the normal, default View operations are done. If it is 5, the Run Wait value is retrieved and the program pauses for that amount of time. After the wait is completed, the slide is displayed. Finally, the "WaitDone" Key/Event is signaled to join the normal processing.

Note that to use the above set of actions, sets of actions on the slides for the "Viewed" Key/Event should end with "Use Global Actions" instead of "Continue with Default Action". This ensures that the new Run Type is implemented.

More commonly, the "Viewed" Key/Event is used to execute actions that initialize values before a slide is displayed. You will frequently see that use in the example files.

There is one last Key/Event that should be mentioned. A "Timeout" Key/Event is signaled if running goes to read a key (during default "Readkey" or with the "Input A Key" action) and the amount of time on the Global menu's "Timeout During Run" item is exceeded. This is another way of doing what Run Type 4 does, but it uses a global value, which can be changed with a Set Builtin action, and maintains compatibility with files from the original **Dan Bricklin's Demo Program**. A Timeout value of 0 means do not check for timeouts. Timeout values are always in whole seconds.

Note

Remember that the screen is only updated whenever a redisplay is done. Any changes to variables by run actions (along with sounds of any Tone Thud, Tone Beep and Tone Note run actions) are not shown to the viewer until that time. Redisplaying occurs, by default, when the "Viewed" Key/Event is signaled, after the last run action in an action list if there are no run actions that view another slide, and when a Redisplay run action is executed. If changes to variables are not showing up when running, but do when running in Debug mode, you may be missing a Redisplay run action after a variable is changed.

Types of Overlays

There are several types of overlay items, each with its own purpose. If you are not planning to use overlays yet, you may want to skim or even skip this section and return at a later time. It is assumed, however, that you have read the subsection about overlays in the "Overview" section.

There are numeric settings associated with overlay items. The settings are on the Overlays Nums menu. All of the settings may be constants or variables. By changing the value of a variable, you can change the way the screen looks the next time it is displayed. This feature can be used to create special effects.

You can change the referenced slide or variable, and change its constant position offsets with the Overlays Adjust command.

Slide Overlays

Slide overlays are direct references to other slides. That is, they refer to the actual slide, not its number. If the order of slides in the slide show is changed, the reference will continue to refer to the same slide, even if its number changes.

The entire image of the other slide is used (the editable ">THIS SLIDE<" part), but not *its* overlays. If you edit the referenced slide, its current updated image will always be displayed as the overlay.

You add Slide Overlays to a slide's overlay list by using the Overlays Slide command.

There is a special type of Slide Overlay called >THIS SLIDE< which refers to the editable part of the current slide. You add >THIS SLIDE< to the overlay list by typing "T" when the Overlay Slide menu/message is shown. The Overlays Slide and Overlays Adjust commands bring up the Overlays Slide menu/message.

Bitmapped images may not be used as overlays. Overlay references to slides with bitmapped images are ignored.

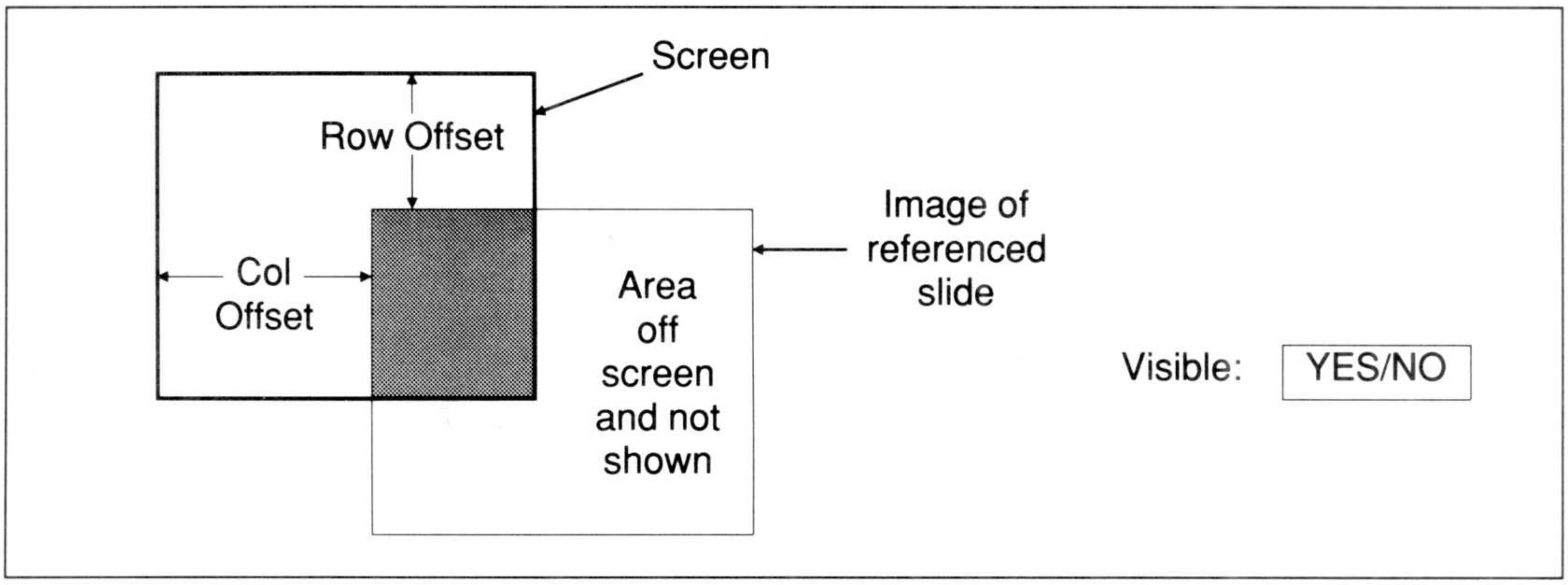


Figure 1. Slide Overlay

There are three settings associated with Slide Overlays: the Row Offset, the Column Offset, and the Visible setting. The Offsets control the positioning of the overlay's image in relation to the screen. A zero value for both Row Offset and Column Offset positions the referenced slide's image exactly over the screen. A Row Offset of 1 would display the top row of the referenced slide on the second line of the screen, and the 25th row of the referenced slide would not be shown. Offsets are ignored for >THIS SLIDE< overlays.

The Visible setting controls whether the referenced slide's image is displayed. If the setting's value is zero, the overlay is ignored. If it is non-zero, the image is displayed. You can use a variable for this setting to turn the overlay "on and off" under control of run actions. By default, the Visible setting has a constant value of 1 (">YES<" displayed).

String Value Overlays

String constants and the values of string variables can be displayed as overlays by creating a String Value Overlay. You add String Value Overlays to a slide's overlay list by using the Overlays Value command and specifying either a string constant (starts with the " character) or a string variable name.

Strings are displayed as a row of characters. The number of characters to be displayed is determined by the current length of the string and the Max Chars Shown setting.

The position of the first character of the string is determined by the Row Offset and Column Offset settings. Offsets 0 and 0 position it in the upper-left corner of the screen. Row Offset 1 would position it on the second row.

The Visible setting controls whether the string is displayed. If the setting's value is zero, the overlay is ignored. If it is non-zero, the characters are displayed. You can use a variable for this setting to turn the overlay "on and off" under control of run actions. By default, the Visible setting has a constant value of 1 (">YES<" displayed).

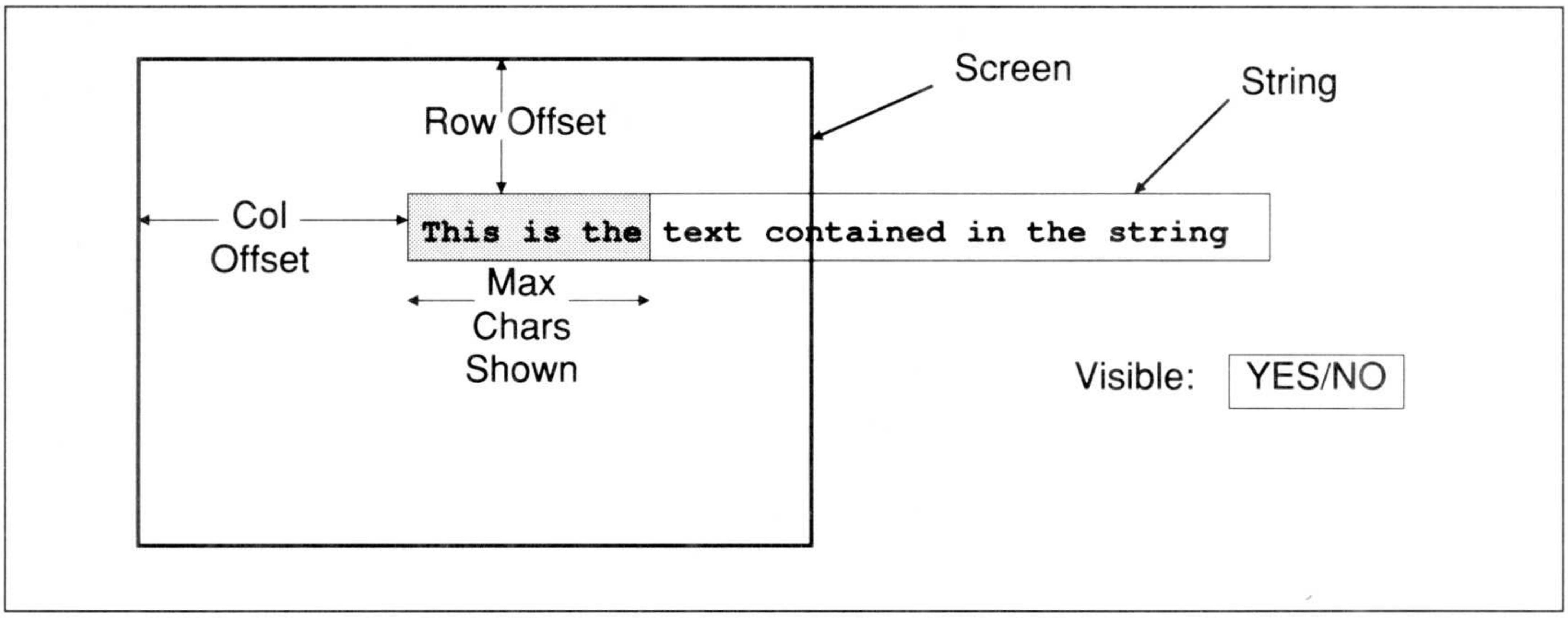


Figure 2. String Value Overlay

The Max Chars Shown setting controls the maximum number of characters that will be displayed from the string. If the setting is less than the current length of the string, only those number of characters will be displayed. You can use a variable for this setting to change what is displayed under control of run actions. By default, the Max Chars Shown setting has a constant value of 80 (the maximum number of characters in a string).

Numeric Value Overlays

Numbers (both constants and variables) can be used as an overlay. The value is displayed in the decimal representation. For example, a variable with a value of -4,321 would display as the five non-transparent characters "-4321". You add Numeric Value Overlays to a slide's overlay list by using the Overlays Value command and then specifying either a numeric constant or a numeric variable name.

The Row Offset, Column Offset, Visible, and Max Chars Shown settings are just like those for String Value Overlays.

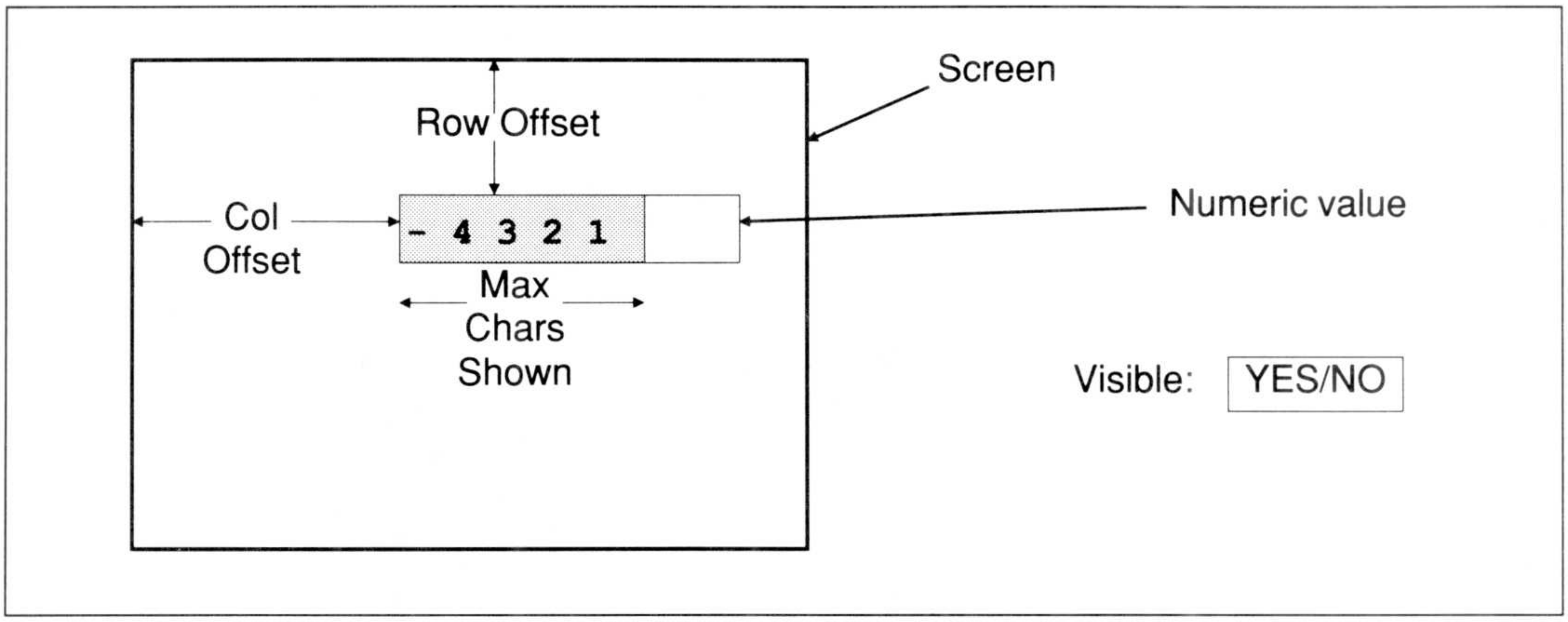


Figure 3. Numeric Value Overlay

String and Numeric Value Overlays with H/W Cursor

Both numbers and strings may be displayed optionally with the blinking hardware (H/W) cursor added after the last character displayed. You can use this feature when you want to have a cursor showing while the user types into a field. This is called a "w/ H/W Cursor" String or Numeric Value Overlay.

You change a String or Numeric Value Overlay into a "w/ H/W Cursor" Overlay by calling up the Overlays Numbers menu associated with the overlay item (using the Overlays Nums commmand) and changing the value of the Type item to "w/ H/W Cursor". See the description of the Overlays Nums Value command for more information.

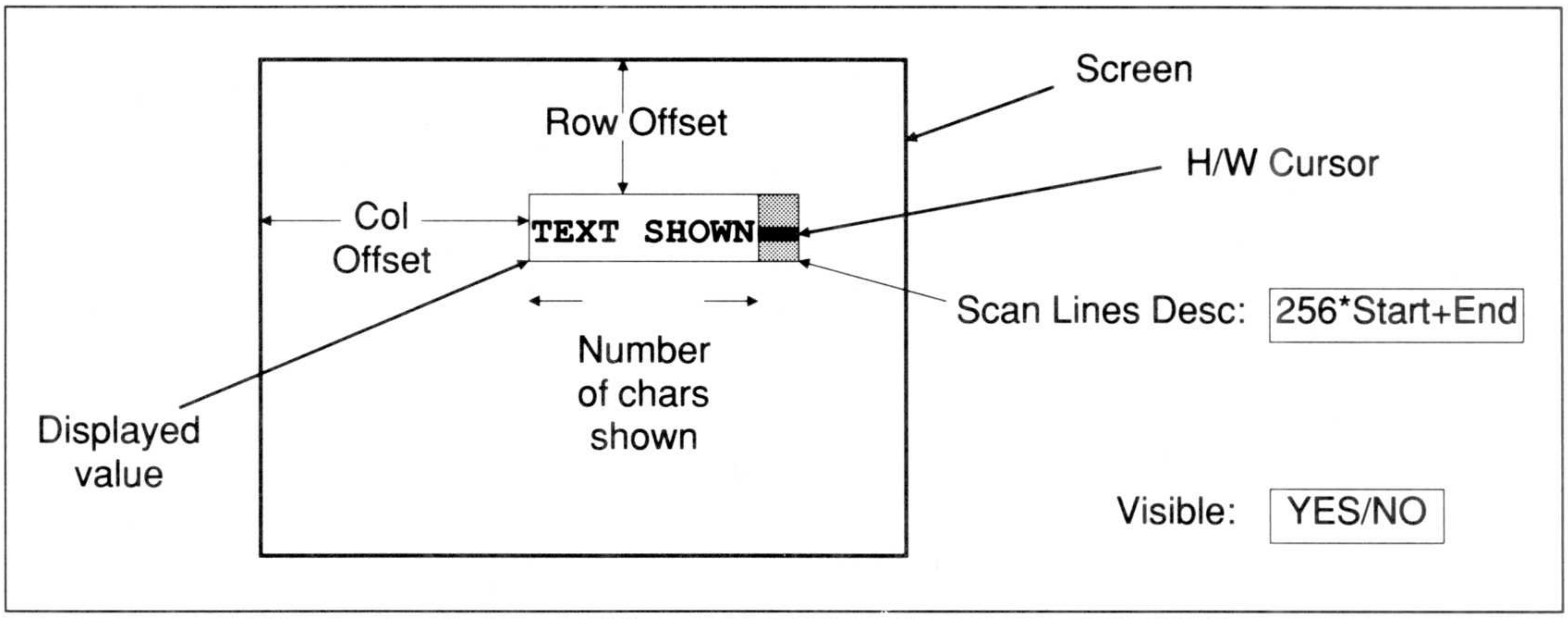


Figure 4. String or Numeric Value Overlay w/ H/W Cursor

Only one H/W cursor can be active at a time, since it is a hardware-implemented cursor, and the hardware can only show one. The last visible one in all of the overlays, including the Global Overlays list, is used. The others are ignored.

These types of overlays have an additional setting called the "Scan Lines Desc" setting. This setting determines the appearance of the H/W cursor. If set to 0 (shown as ">DEFAULT<"), a default cursor is used, like that used by DOS. (It looks like a blinking underline.) Any other value is interpreted as having two parts: the start scan line and the ending scan line. The scan lines are numbered from 0, and the indicated lines blink. The two parts are put into one value by multiplying the starting scan line by 256 and adding the ending scan line. For example, starting at 4 and ending at 5 would be 4*256+5 = 1029.

The H/W cursor is placed immediately following the last displayed character in the overlay. The last displayed character is the smaller of the number of characters to be displayed and the Max Chars Shown setting.

Absolute Slide Reference Value Overlays (ABSREF)

The numeric value of a constant or variable may be used to refer to a slide to be shown. This is similar to a Slide Overlay, except that the absolute position of the slide in the Slides menu list is used instead of a reference to a particular slide. If a new slide is inserted before the referenced slide, a Slide Overlay continues to refer to the same slide with its new number. In the same case, an ABSREF Overlay will continue to refer to the slide that moves into the position.

The value used to refer to a slide must be a number between one and the total number of slides. A string value is converted into a numeric value by using the value of its first character (0-255). The number represents the position of the slide in the Slides menu list. Each slide is automatically numbered in that list. By changing the value of

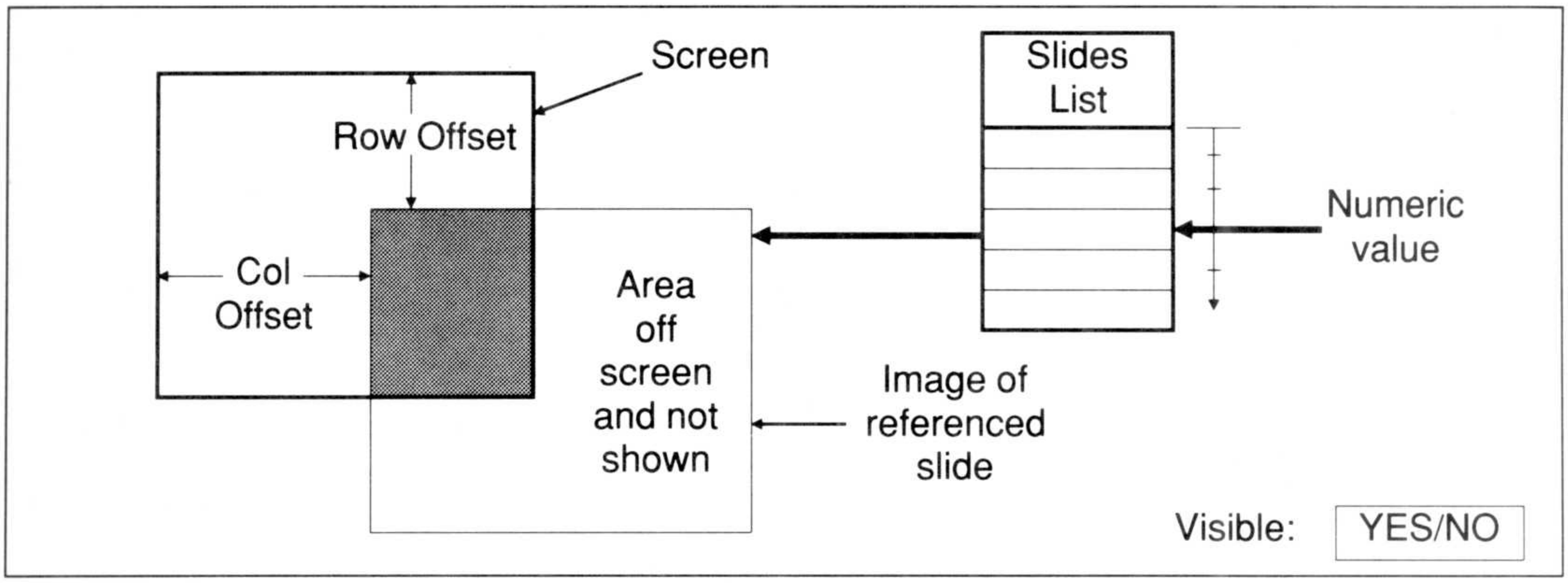


Figure 5. Absolute Slide Reference Value Overlay (ABSREF)

a variable associated with an Absolute Slide Reference Value Overlay, it is possible to change the slide shown the next time the screen is updated.

You can change a Numeric Value Overlay into an ABSREF Overlay by using the Overlays Nums commmand to call up the Overlays Numbers menu associated with the overlay item, and changing the value of the Type item to "Abs Slide Ref". See the description of the Overlays Nums Value command for more information.

ABSREF Overlays can be used to accomplish a variety of functions. For example, you can refer to a slide by name by using the "Slide With Name" run action to get its slide number, and then use a variable with that value in an ABSREF Overlay to display it.

Relative Slide Reference Value Overlays (RELREF)

The numeric value of a constant or variable may be used as the relative position of the slide in the list of slides, as shown on the Slides menu. This may be a positive or negative number. Zero refers to the current slide, 1 refers to the "next" slide, and -1 refers to the "previous" slide. By changing the value of the variable, it is possible to change the slide shown the next time the screen is updated. This is similar to an ABSREF Overlay, but this method uses relative positions in the list instead of absolute positions.

RELREF Overlays are very useful for simulating menus and other applications. For example, through the use of run actions, you could have the arrow keys increase and decrease the value of a numeric variable associated with a RELREF Overlay on the current slide. The menu images could be the slides immediately following the current slide. The variable would start out at 1, showing the next slide, which would show the first menu item selected. Pressing a forward arrow would increment the variable by one, showing the second slide after the current slide as an overlay, and it would have the second menu item selected, and so on.

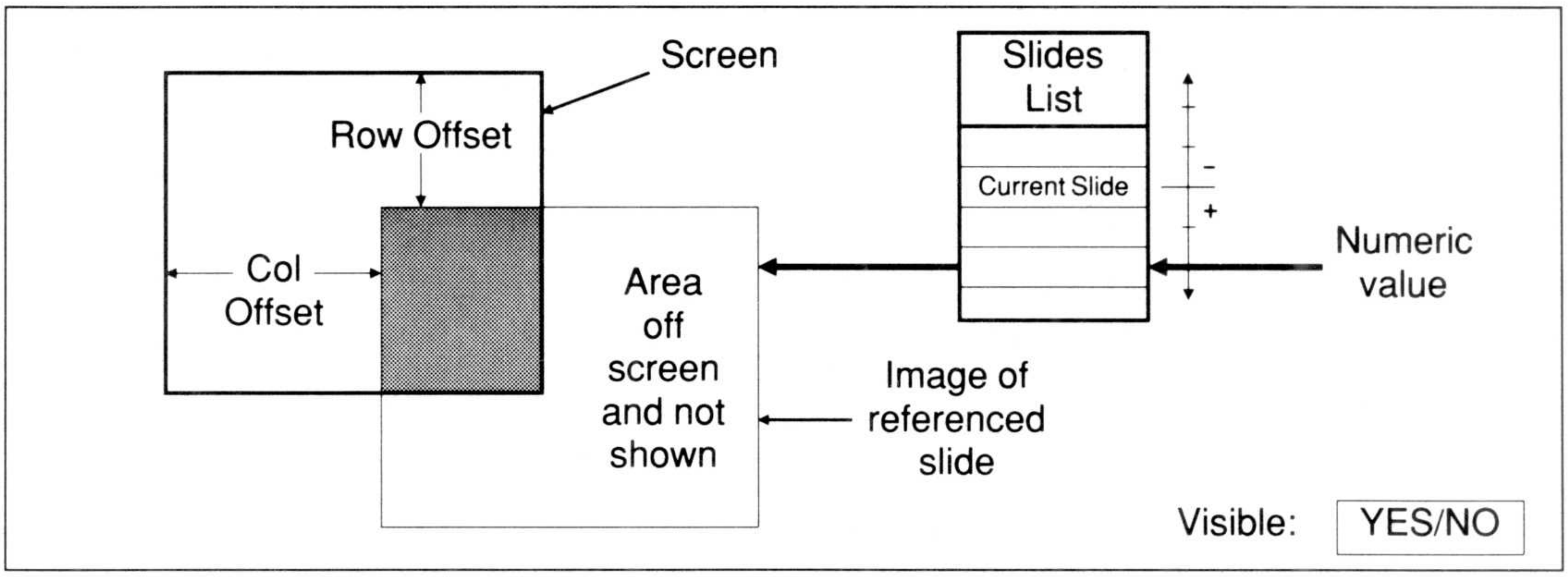


Figure 6. Relative Slide Reference Value Overlay (RELREF)

You can change a Numeric Value Overlay into a RELREF Overlay by using the Overlays Nums commmand to call up the Overlays Numbers menu associated with the overlay item and changing the value of the Type item to "Rel Slide Ref". See the description of the Overlays Nums Value command for more information.

Attribute Value Overlay

The numeric value of a constant or variable can be used to specify an attribute to constitute the overlay. The number of positions covered can also be specified as the Max Chars Shown. This is called an Attribute Value Overlay (ATTRIB).

This type of value overlay displays only attributes. Normal String and Numeric Value Overlays only display characters. One of the uses for Attribute Value Overlays is to simulate cursors and highlights.

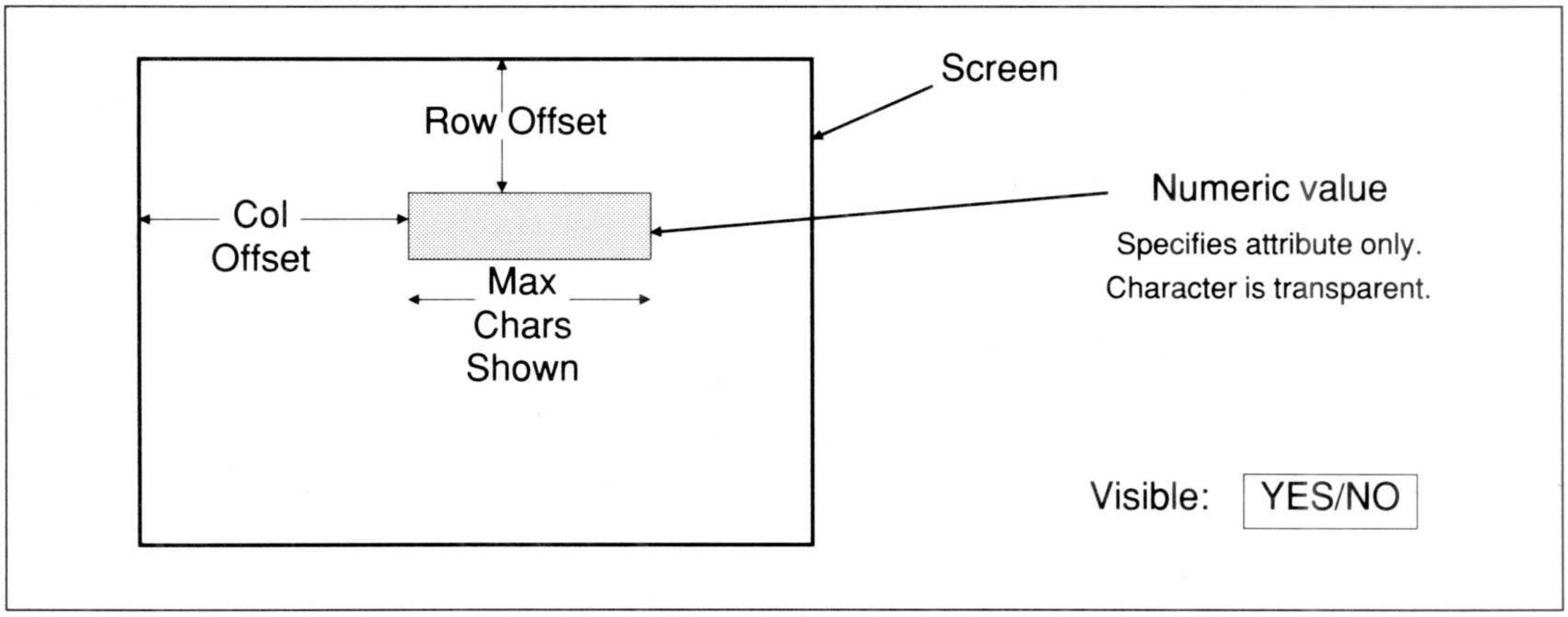


Figure 7. Attribute Value Overlay

To create an Attribute Value Overlay, start with a Numeric Value Overlay and use the Overlays Nums Values command to change the Type item to "Attribute". The value associated with the overlay represents a normal attribute number. For example, 7 would be white on black, 0x70 (hexadecimal 70 or decimal 112 — an "x" at the beginning of a numeric constant signals a hexadecimal value) would be black on white, etc. See the Attribute Chart Appendix for a list of attribute values, or use the Block Attribute or Block Xlate command to see a list on the screen.

H/W Cursor Overlay

You can have an overlay that just specifies the position and appearance of the H/W cursor. This type of overlay is displayed as ">H/W-CURSOR<" in an overlay list.

You can add a H/W Cursor Overlay to a slide's overlay list by using the Overlays Cursor command. You can also change its position with the arrow keys. You can change the position at a later time with the Overlays Adjust command. You can change its settings (Row Offset, Column Offset, Visible, and Scan Lines Desc) by using the Overlays Nums command.

Only one H/W cursor can be active at a time, since it is a hardware-implemented cursor, and the hardware can only show one. The last visible one in all of the overlays, including the Global Overlays list, is used. The others are ignored.

This type of overlay has an additional setting called the "Scan Lines Desc" setting. This setting determines the appearance of the H/W cursor. If set to 0 (shown as ">DEFAULT<"), a default cursor is used, like that used by DOS. It looks like a blinking underline. Any other value is interpreted as having two parts: the start scan line and the ending scan line. The scan lines are numbered from 0, and the indicated lines blink. The two parts are put into one value by multiplying the starting scan line by 256 and adding the ending scan line. For example, starting at 4 and ending at 5 would be 4*256+5 = 1029.

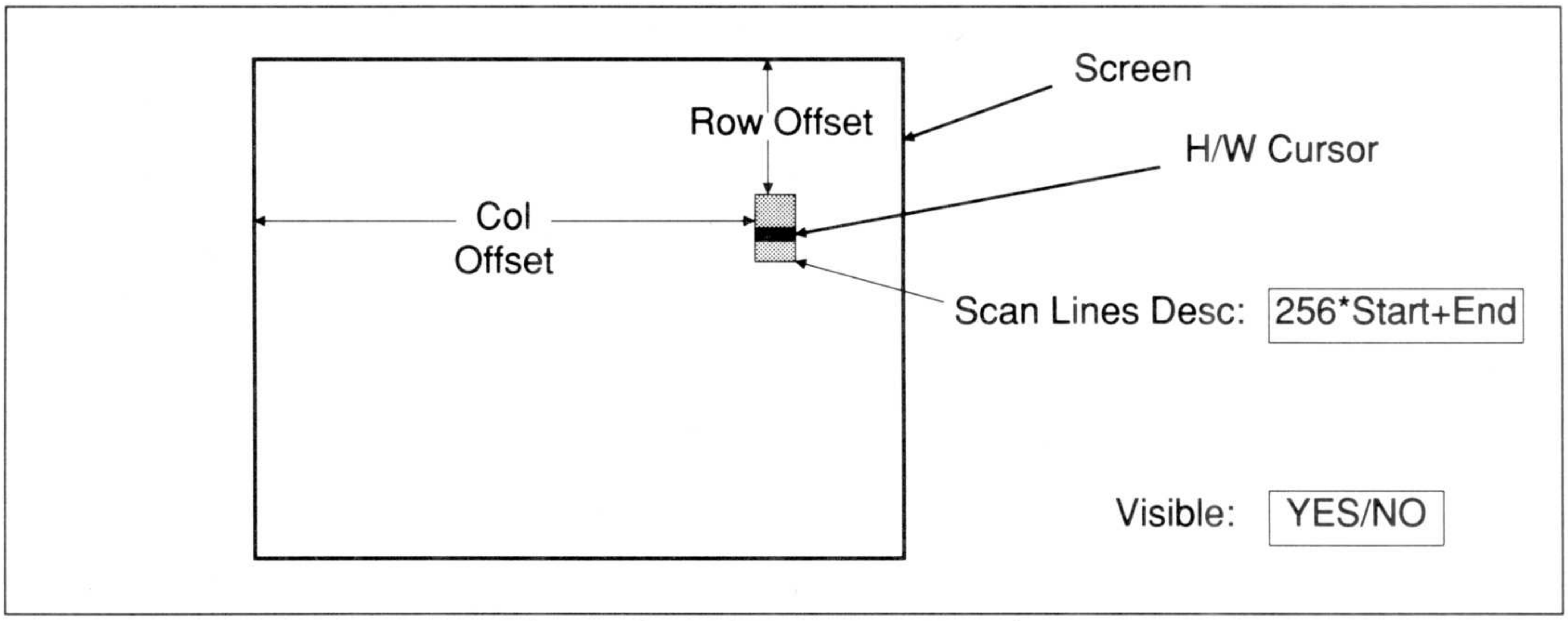


Figure 8. H/W Cursor Overlay

CAPTURE Inserts a H/W Cursor Overlay

A H/W Cursor Overlay is automatically added to each captured text slide. The Offsets and Scan Lines Desc will be of the cursor that was on the screen when the screen was captured. The Offsets and Scan Lines Desc may define a cursor that was not visible at that time. The Visible setting is always set to >NO< (zero) by default. H/W Cursor Scan Lines Desc's are not the same for different displays. For example, the CGA has eight scan lines, while the EGA uses more. Having the information there, but hidden, allows you to use it (by setting Visible to >YES< or 1), or modify it (for example, by setting Visible to >YES<, and Scan Lines Desc to 0 for >DEFAULT<). If you are not going to use the H/W Cursor on the retrieved slides, you may want to delete the H/W Cursor Overlays after retrieving.

Redisplay to Show Variable Changes

When using variables for overlay settings, or in Value Overlays, you must remember that the screen is not updated to show any changes until the next redisplay operation. Redisplays occur when a slide is viewed, when new input is requested on a slide, and when an explicit Redisplay run action is executed. Failure to do a redisplay is a common mistake when using overlays. If the effect that you are trying to produce occurs only when running in Debug mode, you probably forgot a Redisplay somewhere in a loop or after some operation. See the "How Running Works" section for more information about redisplaying.

References to the Current Slide

References to the *same slide* as the one with the overlay list (not ">THIS SLIDE<" references, just ones that treat it like other slides) can cause unexpected behavior. You should rarely need to use such overlays, but they do come up when using global overlays. The image used by overlay references to slides (other than ">THIS SLIDE<") are from the compacted image of the ">THIS SLIDE<" part of the slide. During editing, the compacted image and the actual image will differ until the time when you switch to another slide, start running, or execute a variety of other operations. Therefore, editing changes to the slide will not show up on the overlay references immediately. They will, though, appear when the slide show is run or after F1/F2 is pressed.

The DEMO2 and RDEMO2 Programs

The main DEMO II program is the DEMO2.EXE file on the Program Diskette. This program is loaded once into RAM when you invoke it from the DOS prompt. It then runs completely in memory; there are no program overlays or help files to be read. You do not have to keep a diskette with the program in the drive. DEMO II is *not* a background, "terminate-and-stay-resident" program, like pop-up desktop accessory managers or the CAPTURE program. It is more like most word processors.

The RDEMO2 program (RDEMO2.EXE on the Second Diskette) is the "runtime only" version of DEMO II. It runs completely in memory, just like DEMO2.

Within this manual, the term "DEMO II" is often used to refer to DEMO2.EXE. Other times it will refer to both DEMO2.EXE and RDEMO2.EXE. Which definition is being used should be obvious from the context.

Invoking DEMO2.EXE

To run DEMO2.EXE you execute one of the following command forms from DOS command level:

DEMO2

```
DEMO2
DEMO2 filename
DEMO2 options
DEMO2 filename options
```

The DEMO2.EXE file is on the DEMO II Program Diskette. You should always work from a copy of the original. If you have a hard disk, you should copy it to the hard disk. (See the *Backup* subsection, below.) You do not need any other files in order to run the program. You must be using DOS version 2.0 or a more recent version. DEMO2 is not Microsoft Windows or IBM TopView "aware".

"Filename" is the name of a DEMO2 save file. Use only the name; do not provide the ".DBD" extension. Full pathnames may be used, although DEMO2 restricts path names to 52 characters — about 4 directory levels or so.

"Options" is one or more of the following, all of which are acceptable in uppercase or lowercase:

-snow Use this option when you have a CGA-like display adapter that does not need the program to synchronize its access to the adapter's memory with the retrace signals. DEMO2 may appear to run somewhat faster when this option is used. DEMO2 automatically detects some common display adapters that meet this requirement, such as those on most Compaqs.

-ega Use this option to indicate that you have an EGA or equivalent display adapter. DEMO2 usually can automatically detect an EGA. Check the Global menu to see what display adapter it thinks you have.

-cga Use this option to indicate that you have a CGA or equivalent display adapter that has CGA-type graphics and that needs synchronization for text display.

-herc Use this option to indicate that you have a Hercules Graphics Card or equivalent. DEMO2 usually can automatically detect a Hercules card. Check the Global menu.

-slow There may be times with some display adapters when even the -CGA option produces a small amount of "snow". In those cases, use this option. It also minimizes the amount of time DEMO2 runs with interrupts inhibited. Normally, in -CGA mode, interrupts are inhibited during vertical retrace while the screen memory is updated (a millisecond or so).

-mono Use this option to indicate that you have a Monochrome Display Adapter or equivalent. DEMO2 usually can automatically detect an MDA card. Check the Global menu.

-vga Use this option to indicate that you have a VGA or equivalent display adapter. DEMO2 usually can automatically detect a VGA. Check the Global menu to see what display adapter it thinks you have.

Only the EGA-type features of the VGA are used; e.g., VGA 640x480 graphic images are not supported.

Normally, if DEMO2 finds the video adapter in mode 7 (monochrome text), no bitmapped graphics images are allowed. If a VGA is detected automatically, and it is in video mode 7, it is switched into mode 3 (80 column color text), and graphics are allowed. To disable this feature, use the -MONO option.

-dir This option is followed by a space and then the pathname of the directory you want to use as the DEMO II System Directory. If the -DIR option is not used, the DEMO II System Directory will be set to the DOS Current Directory. The DEMO II System Directory is described below.

-mem This option is followed by a space and then the number of Kbytes to allocate for DEMO2's use for storing slide show information. Normally DEMO2 allocates all the rest of RAM. Extended and Enhanced memory are not used. This means that COMMAND.COM must be reloaded after DEMO2 or RDEMO2 runs. You determine the number of K needed by looking at the Global menu Memory display.

Use the largest value from all of the files that will be run. If you allocate (or there is) too little memory, you will be unable to load the files you want or will be unable to edit. You must use this option if you are using the EXEC action to execute another program while running.

For example:

```
demo2 test5 -mem 55
```

-bitm16

-bitm32 These options are used to reserve memory for bitmapped graphics images. See the discussion in the section "Bitmapped Graphics Images" for more information about when to use these options. If you do not provide these options, the screen will display "Need -bitm16" or "Need -bitm32" when a slide that needs these options is displayed. -Bitm32 is a superset of -bitm16. -Bitm16 reserves 16000 bytes while -bit32 reserves 38400 bytes.

-arg Sets the value accessed by the Builtin(-12) run action. This lets you pass arguments to a slide show.

Using SET DEMO2=

In most cases you will not need to use the options, since DEMO2 can detect the presence of many display adapters. If you need to use options, you can use the DOS Environment String "DEMO2" to pass values to the DEMO2 command. You can do this by using the DOS SET command to give a value to the string "DEMO2". For example,

```
SET DEMO2=-BITM32 -MEM 100
```

would pass the options "-bitm32" and "-mem 100" to DEMO2 before it checked any options on the command line. See your DOS manual for information on the SET command.

The options are the same as are accepted by the RDEMO2 program.

Files And Directories Used By DEMO II

DEMO II saves and loads information from files in a variety of special formats.

The normal DEMO II save file is identified by an extension of ".DBD". DEMO II (DEMO2 and RDEMO2) can read ".DBD" files produced by both the DEMO2 program and the older **Dan Bricklin's Demo Program** Version 1A (releases D1A and D1B). DEMO II can only write the newer ".DBD" files that are not readable by the older program.

"_INIT.DBD" is a special name for a save file called the "Init File". It is a normal DEMO2 save file that is loaded automatically whenever DEMO2 is started and no file name is given, or when a Global ClearAll command is executed. You can save to this file by giving the file name "_INIT" in an I/O Save operation. You use this file to set initial values when you want values other than those that DEMO II provides by default. For example, you may want a different set of attributes for the attribute list, as well as different Background and Menu attributes if you always run on a color monitor. You may want to use a specific printer mapping by default, etc. Note that modifying the Init File only affects future usage — it does not go back and change existing files.

Copies of run actions can be stored in ".DBC" files. These files are saved and loaded by the I/O Write-Code and I/O Code-Read commands, respectively.

The definitions for macros A-Z are stored in a file named "_ALT_A_Z.SG1". These are not compatible with the older Version 1A files. The macros 0-9 are stored with the slides. The "_ALT_A_Z.SG1" file is automatically loaded immediately after start up and Global ClearAll, and after any other files (including the Init File) are loaded in DEMO2 (not RDEMO2). By using this feature, you can provide your own "commands" that are common to all files, yet still be able to produce 10 macros specific to each file. See the Macros command description for more information.

The printer mappings are stored in files of the form "_P??????.SG2", where "??????" is the name of the mapping, e.g., "_PASCII.SG2" for the mapping "ASCII". See the I/O Print Character Mapping setting on the I/O Print menu.

The I/O Print command can produce files instead of printing to the printer, and those files are given the extension ".TXT".

If no file name is provided to the RDEMO2 program, it tries to load the file "_FILE_0.DBD" in the current directory.

The files "_ALT_A_Z.SG1", "_INIT.DBD", and the printer mappings all are assumed to reside in the DEMO II System Directory. The DEMO II System Directory is the DOS Current Directory by default (the one listed when you type the DOS command "cd" or "chdir"). You can set other directories by using the "-DIR" option to the DEMO2 program. By varying the name of the DEMO II System Directory, you can have custom environments for different projects.

RDEMO2

Included in the DEMO II package is a copy of the RDEMO2 program. It is the RDEMO2.EXE file on the Second Diskette. RDEMO2 is the "runtime only" version of DEMO II. You would use this program when you want to give someone else a copy of a slide show you have created. It takes the same command line options as DEMO2.EXE:

RDEMO2

RDEMO2

RDEMO2 *filename*

RDEMO2 *options*

RDEMO2 *filename options*

The "filename" and "options" are interpreted in a manner similar to the DEMO2 program.

The RDEMO2 program loads the slide show named on the command line or "_FILE_0.DBD" if none is specified (that is a zero, not the letter O). It automatically starts the slide show running, as if you had typed Esc Run Run. When the slide show stops running for any reason (e.g., a Quit run action, an error, pressing Ctrl-Break), RDEMO2 returns to DOS. RDEMO2 can be used by people who will only run a slide show without any modifications. It is also substantially smaller in size than the full DEMO2 program, so you can fit more slides on a diskette or in memory.

This package comes with a special license for RDEMO2.EXE to allow you to make copies of it along with slide shows for use by others. Please check the License Agreement and Warranty for more information.

Other, specialized "runtime only" versions of DEMO II become available from time to time for an additional fee. Registered users will usually be notified of these versions (if there is no information included in this package), or you can write to the address on the License Agreement and Warranty for the latest information. These versions include one with no start-up message.

Backup and Copying

You should make backup copies of the files on the DEMO II diskettes. The License Agreement that you accept by opening the DEMO II Diskette Package permits you to make backup copies of the programs, but DEMO2.EXE, CAPTURE.COM, CAPTCMD.EXE and PCXDECOD.EXE may only be running on ***one PC at a given time***. For example, you may use a copy of it at home and another copy at work, but not both copies at the same time on the two machines. This is sometimes called a "like a book" limited license. The diskettes are not copy-protected, of course.

See the License Agreement and Warranty for information about RDEMO2.EXE. The Agreement lets you make an unlimited number of copies of diskettes containing RDEMO2.EXE, subject to conditions and an indemnification listed there of claims arising out of your use of RDEMO2.EXE.

The CAPTURE Program

Included in the DEMO II package is a program called CAPTURE. It is in the file CAPTURE.COM on the Program Diskette. CAPTURE is the screen capturing program for use with DEMO II. It is a background, "terminate-and-stay-resident" program.

When you execute CAPTURE, it installs itself in RAM memory as a background program. It watches the keyboard and, when both shift keys are held down at the same time, it saves a copy of the screen in RAM. More than one screen can be captured if desired. At a later point, you run DEMO2 and execute the I/O Retrieve Capture command. The screens that were "captured" are then inserted into the current slide show.

CAPTURE is a flexible program. It can capture both text and bitmapped screens. You can set a variety of options, including the method by which it is triggered and the amount of memory that it uses. Some options are set by using the CAPTCMD program (CAPTCMD.EXE on the Program Diskette).

DEMO2 and CAPTCMD communicate with the background CAPTURE program by using information stored by CAPTURE in a file. Each time you run CAPTURE, it stores information in a file called "CAPTURE.RDV" in the current directory. The "RDV" is short for "Rendezvous", which is the term for the operation performed with the information. Whenever DEMO2 or CAPTCMD wants to communicate with CAPTURE, it looks at the "CAPTURE.RDV" file in the current directory.

It is important to understand that you must run CAPTURE, DEMO2, and CAPTCMD all with the SAME CURRENT DIRECTORY. If the current copy of "CAPTURE.RDV" is not found in the current directory, DEMO2 and CAPTCMD cannot communicate with CAPTURE.

The "current directory" is the one you set with the DOS "CHDIR" or "CD" command. The programs themselves do not have to be in the current directory, just the "CAPTURE.RDV" file that is created by CAPTURE. After you run CAPTURE, you can change the current directory while you capture the screens. Then you can change back to do the retrieving or to set options with CAPTCMD.

When CAPTURE saves a screen image, it saves it in RAM. The screens are saved in a compacted form, similar to the one DEMO II uses internally. You may save as many screen images as will fit in the memory allocated to CAPTURE. Until the screen images are retrieved by DEMO2 and subsequently saved on disk in a DEMO II save file, they reside in RAM and will be lost when you turn the PC off or do a System Reset (Ctrl-Alt-Del).

CAPTURE can save a variety of screen-image types. It can automatically capture monochrome, black and white, and color text screens, as well as some of the bitmapped screens: 320x200 CGA 4 color (BIOS screen modes 4 and 5), 640x200 CGA monochrome (mode 6), and 640x350 EGA 2 color (mode 10h). It can be set using CAPTCMD to capture 640x350 EGA 16 color (mode 10h, including palette information), and Hercules Graphics Card 720x348 monochrome (no standard mode, either page 0 or 1).

In addition to the "both shift keys" method, you can trigger CAPTURE in other ways. You can change the "hotkey", the key combination that triggers the screen capturing, to almost any other key combination in case the standard one conflicts with other programs. You don't have to use a "hotkey" to trigger CAPTURE; you can have it wait a specified amount of time and then automatically capture the screen. This is helpful when you are capturing from a program that takes control of the keyboard. Finally, you can have CAPTURE watch the screen (in text mode) and trigger automatically whenever the screen changes.

The CAPTURE program is designed for use only with DEMO II and CAPTCMD. It may conflict with other background products. You may have to experiment with the order in which you install various background programs in memory in order to get the operation you desire. CAPTURE does not have to be loaded as the last program. If it is the last program loaded, CAPTCMD can be used with the "-OFF" option to unload it from memory. ***You must not unload CAPTURE with any "TSR Unload" programs, especially on systems with an EGA or VGA.***

Getting a background program like CAPTURE to work in all circumstances with all programs is a very difficult, if not impossible, task. If CAPTURE does not meet your needs after some experimentation, it is probably not appropriate to your situation. It has, though, been used successfully by many people, capturing screen images from spreadsheets, new products, and programs still in development. CAPTURE was used to produce parts of the tutorial on the diskette.

If you use overlays in conjunction with slides that were captured, you should check the note at the end of this section about captured text images.

The CAPTURE Program

CAPTURE

CAPTURE

CAPTURE nnn

You can invoke the CAPTURE command with either no options or a number between 16 and 127. The number specifies how much memory to reserve for the captured images in KBytes. The default is 32K.

A text screen uses approximately 300 to 4000 bytes, depending upon how many different characters and attributes it has. Bitmapped screens take up from a few hundred bytes to 16KBytes for low resolution, up to 32K for higher resolution, and up to 127K for 16-color EGA high resolution. Since most screens have large areas with the same color, character, or pattern, most screens take up less than the maximum. If CAPTURE makes a higher pitched sound than normal when triggered, it has run out of memory or could not capture from that screen mode. To allocate more memory, turn off CAPTURE with CAPTCMD -OFF, or re-boot the PC. Then run CAPTURE again with a larger number.

For example, to capture many text screens, you may want to reserve 64K of memory. You would use:

```
CAPTURE 64
```

Make sure that you do not capture more data than there is memory in DEMO II for retrieving. For instance, do not capture 80K of screens when DEMO II has only 50K free with CAPTURE in memory on a 512K PC.

When executed, CAPTURE writes out a file named "CAPTURE.RDV" in the current directory. Refer to the previous discussion of that file.

To change the Hotkey, method of triggering, and other settings, run the CAPTCMD program after CAPTURE is loaded.

The CAPTCMD Program

CAPTCMD

CAPTCMD

CAPTCMD -OFF

CAPTCMD -FLUSH

This command is used to control the CAPTURE program. It must be run with the same current directory as CAPTURE (see the previous discussion about "CAPTURE.RDV").

The "-off" option turns off CAPTURE, unloading it from memory if possible.

The "-flush" option discards the captured images that have not yet been retrieved by DEMO II. You can then continue capturing new images.

When you run this program, it displays something similar to the following information:

```
3 screen images captured and waiting to be retrieved
5K out of 32K free space used

Hotkey triggering, currently Shift: 3, Key: 0
Using BIOS value for screen mode (normal)
```

This tells you the current status and settings. You are then given the opportunity to change the triggering and screen mode settings:

Trigger

Triggering Method: Hotkey, Timed, Watch Screen (H/T/W)

The current method is the default if you just press Enter. Hotkey means that you trigger capturing by pressing keys. Timed means that capturing is automatically triggered after a specified amount of time repeatedly. Watch screen means that the text screen is checked periodically and, if it has changed, capturing is triggered.

When CAPTURE is triggered, it copies the current screen image into the RAM you reserved for CAPTURE. If successful, it makes a low pitched tone. If it is unable to capture for some reason (not enough memory, unknown screen mode), it will make a higher pitched sound. (For those readers technically minded who need to know, the actual save is done at a 1/18th second timer-interrupt with interrupts not inhibited; saving does not occur during the keyboard interrupt.)

Hotkey Triggering

If you select Hotkey (H), you will be asked:

Trigger
Hotkey
Shift

New Hotkey Shift Value (add: LeftShift=1, Right=2, Ctrl=4, Alt=8)

The value you type describes the shift state necessary for triggering. Pressing Enter gets the default displayed. For example, 3 would mean both shift keys pressed (LeftShift plus Right = 1+2); just the Ctrl key pressed would be 4; Left Shift and Ctrl pressed would be 5, and so on.

Then you are prompted for the key that must be pressed at the same time, if any:

Trigger
Hotkey
Key

New Hotkey Key Value (0=None, or key number)

You specify the number of the key that must be pressed at the same time as any specified shift/ctrl/alt keys to trigger capturing. Pressing Enter gets the default displayed. For example, to make Ctrl-Esc the Hotkey, you would set the Shift Value to 4 and the Key Value to 1. Zero means that no key is needed — just the shift/ctrl/alt. The keys pressed are passed on to the program running, so it is often a good idea to use Hotkey combinations that are ignored by your program (often shift/ctrl/alt combinations alone, or Ctrl-Esc, Ctrl-Pad5, etc.).

The key numbers are:

1-Esc	2-1	3-2	4-3	5-4	6-5	7-6	8-7	9-8	10-9
11-0	12--	13-=	14-Bksp	15-Tab	16-Q	17-W	18-E	19-R	20-T
21-Y	22-U	23-I	24-O	25-P	26-[	27-]	28-Entr	29-Ctrl	30-A
31-S	32-D	33-F	34-G	35-H	36-J	37-K	38-L	39-;	40-'
41-`	42-	43-\	44-Z	45-X	46-C	47-V	48-B	49-N	50-M
51-,	52-.	53-/	54-	55-Gry*	56-	57-Spc	58-	59-F1	60-F2
61-F3	62-F4	63-F5	64-F6	65-F7	66-F8	67-F9	68-F10	69-	70-
71-Home	72-Up	73-PgUp	74-Gry-	75-Left	76-Pad5	77-Rght	78-Gry+	79-End	80-Dn
81-PgDn	82-Ins	83-Del							

Common combinations are: Ctrl-Esc (Shift: 4, Key: 1), Ctrl-NumericPad5 (Shift: 4, Key: 76), Ctrl-SpaceBar (Shift: 4, Key: 57), LeftShift-Ctrl (Shift: 5, Key: 0), Alt-Esc (Shift: 8, Key: 1). Use these examples to help you figure out any others that you might need.

Timed Triggering

If you specified Timed triggering, you would see the prompt:

Trigger
Timed
First

Time to pause before first triggering (1-255 seconds)

The specified amount of time after you exit CAPTCMD, the first triggering will occur. If you press Enter, the value shown will be used.

Trigger
Timed
Between

Time to pause thereafter between triggerings (1-255 seconds)

You can also set the amount of time between each subsequent triggering. Press Enter to use the value shown.

If an error, such as out of memory, occurs during Timed triggering, triggering will revert to Hotkey, thus stopping the auto-triggering. You can also stop Timed triggering by re-executing CAPTCMD.

Watch Triggering

If you specified Watch triggering, you may see the following message. It only shows the first time you set Watch mode in a given CAPTURE run.

```
Removing 4000 bytes from capture free space for doing
                                  comparisons.

Flushing any captured screens.
```

Then the pause prompts are put up, similar to Timed triggering. You can stop Watch triggering by re-executing CAPTCMD.

Trigger
Watch
First

Time to pause before first triggering (1-255 seconds)

The specified amount of time after you exit CAPTCMD, the first triggering attempt will be made. If you press Enter, the value shown will be used.

When a triggering is attempted in Watch triggering, the screen is compared to the saved image of the last screen captured. This last screen starts out undefined and will probably capture the first screen in all cases. If there are any changes, or if the image is not a text mode screen, capturing is triggered. If there are not any changes, then the "between" time is waited, and another triggering is attempted, and so on.

Trigger
Watch
Between

Time to pause thereafter between triggerings (1-255 seconds)

This sets the amount of time between each subsequent triggering attempt. Press Enter to use the value shown.

If an error occurs. such as out of memory, during Watch triggering, triggering will revert to Hotkey, thus stopping the auto-triggering.

To use Watch triggering, you usually set the first time to be long enough to enter the program you want to capture. Then wait for the first tone, which indicates a successful triggering. After you make changes to the screen (or if it changes by itself while the program is running), you should wait long enough for the next triggering to be attempted. Listen for the tone, and then continue. It is often possible to just go through a sample session slowly and have CAPTURE save the whole thing at a very natural pace. Set the between-time longer if you want to have time to make several changes between captures; set it shorter to capture frequently.

The Watch triggering feature is helpful with programs that take control of the keyboard and that do not allow Hotkeys to work.

Screen Mode

After setting the triggering, you can specify how CAPTURE determines the display mode of the screen.

Mode

How to Determine Display Mode: BIOS (normal), Explicit (B/E)

You can specify that CAPTURE checks with the BIOS to find out the screen mode, which is the normal way, or have it always assume an explicit mode. The screen mode indicates whether the screen memory is to be interpreted as text, 320x200 bitmapped, 640x200 bitmapped, etc. Since most programs let the BIOS know the mode, that is the normal way of determining the screen mode. The other option exists to take care of those unusual cases some of us run into. Pressing Enter uses the default shown.

If you answer "E" for Explicit, you will be given the following prompt:

Mode Explicit

Explicit Display Mode Value

The following list of allowable values is displayed:

```
0 = BIOS
1 = CGA text mode
2 = Monochrome Adapter Text Mode
3 = CGA 320x200 4 color graphics
4 = EGA 640x200 2 color graphics
5 = EGA 640x350 2 colors (force settings)
6 = EGA 640x350 16 colors (force settings)
7 = Hercules 720x348 page 0
8 = Hercules 720x348 page 1
```

The first value (0) is the same as BIOS (normal) mode. The second and third (1 and 2) are normal text modes (BIOS modes 3 and 7, respectively). Use the text modes if you have two monitors and you want to get from one explicitly.

The two plain graphics modes can also be used to explicitly capture from a given display adapter. They can also be used if the program you are capturing from does not let the BIOS know that graphics mode is being used (some spreadsheet programs may do this, for example).

The EGA 2-color with forced settings is needed when a program leaves the EGA in a mode such that you must force the EGA registers to read correctly. This is unusual.

The EGA 16 color must be specified explicitly. The BIOS is not used to distinguish between 2 and 16 color. Also, this is always a "force" mode, where the EGA Read Map Select Register and Mode Registers are explicitly set to read the memory planes. This is the common way of running the EGA, but some programs leave the registers set differently and you cannot check their state on an EGA. Those programs that leave the registers set differently may behave strangely on the screen after a screen is captured.

The Hercules modes must be set explicitly, since there is no standard way of finding out about Hercules Graphics Card graphics from the BIOS.

Note

There is usually no need to fully understand all of the CAPTCMD options. For most applications you may not need CAPTCMD at all — the default settings should work. If you are working with bitmapped images or using a program that takes control of the keyboard (like some communications packages) and the default settings are inappropriate, then you may have to experiment a bit to find out which explicit setting is appropriate for your application. The wide variety of settings are provided to help knowledgeable users cope with as many special-case situations as possible. If you cannot figure out a method that works with your program, you may need to speak to the developers of the application you are trying to capture to find out how they used the screen modes.

More information about what is captured is provided in the section on "Bitmapped Graphics Images" and in the discussion of the I/O Retrieve command.

If you make a mistake in CAPTCMD...

You can use Ctrl-Break to exit CAPTCMD. It will not start any Timed or Watch triggerings. Re-execute CAPTCMD to try again.

Using Captured Text Images With Overlays

Programs use various ways to show blank space on the screen. Often they use space characters or zero attributes. A slide produced by retrieving a captured text-screen image that is all text and space characters will obscure any overlays before it. Most slides that you create with DEMO II only have non-transparent characters where you explicitly put them; so this "new" type of slide will act differently with respect to overlays. You may have to insert a ">THIS SLIDE<" overlay as the first overlay in the overlay list in order to let other overlays be in front of the captured image. Alternately, you could cut "holes" in the slide with Del or the Block Delete (F7) command to let overlays behind the retrieved slide show through.

Zero attributes that are used to show blank space can result in other behavior: text that was not visible on the screen when you captured it will appear mysteriously on the slide. This occurs because the program being captured cleared the screen by leaving the text the same and just setting the attributes to zero (black on black). DEMO II treats zero attributes as transparent, so the background, visible attribute shows through.

Bitmapped Graphics Images

DEMO II can display bitmapped graphics images on the screen instead of the normal 25 rows of 80 characters. These are often refered to as "graphics screens". The program can display these screens, but not create them or edit them. The images must be produced by another program and imported by using the CAPTURE program, or shown by reference to images stored in ".PCX" files. While bitmapped graphics slides cannot be edited or used as overlays or with overlays, they can make use of the run facility for controlling the displaying and sequencing of slides.

Associated with each slide is its Slide Type. The Slide Type is displayed on the Slides Options menu which is accessed with the Slides Options command. The type can be either "Normal Text" or "Bitmapped Graphics". With each slide there is also a Switch Type and a Switch Speed. "Bitmapped Graphics" slides, which we will call bitmapped slides, have an additional item. The item says "Bitmap Origin: Capture" or "PCX Filename: *filename*", depending upon the origin of the bitmapped image. Finally, each slide has a "Palette" setting with Video Bits and other information.

How To Create A Bitmapped Slide

To create a bitmapped slide you use the I/O Retrieve command. You will be prompted with:

I/O Retrieve

Retrieve: from Capture, from Text file, or reference PCX file (C/T/P)?

I/O Retrieve Capture

If you have used the CAPTURE program to capture bitmapped screen images, you can retrieve them by responding with a "C". New slides will be inserted after the current slide and given type "Bitmapped Graphics" with "Bitmap Origin: Capture".

I/O Retrieve Text

Responding "T" lets you import from text files and produces normal, text slides. See the description of the I/O Retrieve command for more information.

If you type "P" you will be further prompted:

I/O Retrieve PCX

Filename of PCX file to be referenced (include extension)

To refer to an explicit ".PCX" file press the " key to specify a string constant, and then type the file name. A new slide will be inserted after the current slide and given type "Bitmapped Graphics" with "PCX Filename: *filename*". For example, if you created an image with ZSoft's PC Paintbrush that is in file "c:\pbrush\startup.pcx", you would respond:

```
"c:\pbrush\startup.pcx
Enter
```

Don't forget the quotation mark to start the string constant!

If the name of the file will be in a string variable, you can type the variable's name or choose it from a list by pressing "?". Use a variable if you want to use the run actions to determine which file was used, i.e., to use a different file on a Hercules system than on an EGA system.

The two ways of getting bitmapped images are discussed at length below.

Captured Bitmapped Images

Bitmapped images retrieved from the CAPTURE program are stored along with all of the other slides in RAM and saved in the ".DBD" files.

There are five types of captured bitmapped screens: 320x200 CGA 4 color, 640x200 CGA 2 color, 640x350 EGA 2 color, 640x350 EGA 16 color, and 720x348 Hercules 2 color.

When you execute the Slides Options Palette command, the Video Bits and Palette information captured along with the slide is displayed. This includes the color-set used in 320x200 4 color. To change the information, see the description of the Slides Options Palette command.

Captured bitmapped images display more quickly than PCX images since they are already in memory when viewed. They take up more memory in the DBD file, and about the same amount or slightly more on disk. Larger DBD files take longer to load, so use this form when you want to use CAPTURE, need speed of access, and when you are not concerned about memory requirements or DBD file-loading time.

PCX Bitmapped Images

Bitmapped images can reside in other non-DBD files separate from the other slides and their information. These files must be in the ".PCX" Picture File Format. PCX format is used by a variety of products, most notably ZSoft's PC Paintbrush program and its upgrades. Similar products (sometimes written by ZSoft) are often included when you purchase a mouse for your PC.

The PCX format is fairly general, is able to describe almost any size screen image, and is compacted. DEMO II only recognizes the following types:

Horizontal Dimension	Vertical Dimension	Bits/ Pixel	Number of Planes	Bytes/ Line
320	200	2	1	80
640	200	1	1	80
640	350	1	1	80
640	350	1	4	80
720	348	1	1	90

If you are not sure what type a particular PCX file is, you can use the PCXDECOD program (PCXDECOD.EXE on the Program Diskette). Run PCXDECOD with the file name as an argument, like this:

```
PCXDECOD LOGO.PCX
```

A formatted listing of the PCX file header will be displayed. The header for PCX files is described in the ZSoft Technical Reference Manual "Technical Documentation for PC Paintbrush, PC Paintbrush +, and Frieze Graphics" (ZSoft Corporation, 450 Franklin Road, Suite 100, Marietta, GA 30067; 404-428-0008). The parts of interest are labeled "X1", "X2", "Y1", Y2", "Bit/Pixel", "NPlanes" and "Bytes/Line". "Encoding" is assumed to be 1. Horizontal dimension is X2–X1+1, and vertical dimension is Y2–Y1+1.

When you execute the Slides Options Palette command, the Video Bits and Palette information for this slide is displayed. That information is ignored for PCX bitmapped slides — the information is derived from the PCX header palette information.

PCX file images take longer to display than captured images since they must be read in from disk immediately before being displayed. They take up much less space in the DBD file (about the same as a blank slide) and the same or slightly less on disk. Use PCX images when you do not need to use Capture (such as when you've created the PCX image with a paint program or used a PCX format capture program such as ZSoft's Frieze) and when the size of the DBD file is a concern. If you are using many high-resolution bitmapped images, you may have to use PCX format and a hard disk due to memory constraints on the loaded DBD file.

Removing Bitmapped Information

You can use the Slides Options Text command to turn a bitmapped slide into a normal text slide. Any captured information is erased. References to PCX files are removed, but the files themselves are unaffected.

Display Adapters, -BITM Options, and Switch Speeds for Different Resolutions

The different bitmapped image resolutions and colors can only be displayed on certain monitors and with certain DEMO II command options. Also, the Slide Switching settings affect them differently. The information is contained in the table below:

Image		CGA	EGA	Herc	-BITM		Switch	Switch
Type					16	32	Slow	Fast
320x200	4 color	Y	Y	N	Y	Y	0	7
640x200	2 color	Y	Y	N	Y	Y	0	7
640x350	2 color	N	Y	N	N	Y	0	8
640x350	16 color	N	Y	N	–	–	–	–
720x348	2 color	N	N	Y	N	Y	0	7
Text		Y	Y	Y	–	–	25 (Types 1, 2)	0
Text		Y	Y	Y	–	–	32000 (Types 3, 4)	0

The CGA, EGA, and Hercules (Herc) modes only display on the appropriate monitor. If the monitor is not the appropriate type, the message:

```
Can't display this type bitmapped image on this system
```

will be displayed whenever the slide is shown. The stored image is unaffected by this condition — it will display correctly when the right options and/or adapters are used.

The "-bitm16" and "-bitm32" options to the DEMO2 and RDEMO2 programs reserve memory for a buffer needed to display the images. If the option is needed and is not present, an appropriate message will be displayed whenever the slide is shown. The message will be one of the following:

```
Need -bitm16
Need -bitm32
```

The stored image is unaffected by this condition — it will display correctly when the right option is provided.

The Slide Switch Types 1 and 2 have different minimum and maximum speeds. You should not use any value other than those from the slow to the fast as shown in the previous table (i.e., not less than 0 or more than 7 for 320x200). Bitmapped slides only use Switch Types 0 (immediate, normal), 1 (replace top to bottom) and 2 (replace bottom to top). See the Slides Options menu description and the "Overview" subsection on *Switching From One Slide To Another* for more information about switching.

The 640x350 16-color mode does not need a "-bitm" option and it ignores the Switch Type setting — it is always switched to as quickly as possible. 640x350 16-color images may appear to display a little slower than other images since they are not moved to the screen from an intermediate buffer and are often so large in size.

The information about text slides is provided in the table for completeness. For text slide Switch Types 1 and 2, you will find Switch Speed values most commonly in the range of 0 to 25. Note that values above 25 are allowed — they just run *very* slowly. For Switch Types 3 and 4, the values between 1 and 200 are useful for making text look like it is flowing onto the screen, while the values from 500 to 32,000 are useful for simulating typing.

Keys

DEMO II has uses for most of the keys on the keyboard; this section defines those uses.

The keys may be interpreted differently, depending upon the mode that the program is in. The main modes are edit mode, command mode, prompt/menu/message mode, and run mode.

Edit mode exists whenever there is no command window, prompt, or menu/message on the screen, and you have not switched by command into run mode. This occurs when you are editing the current slide. On bitmapped slides, do not use the editing keys that modify the ">THIS SLIDE<" character positions.

You are in command mode when a command window is on the screen, and there are no prompts or menu/messages on the screen. You can switch to command mode from edit mode by pressing the Esc key.

Prompt/menu/message mode occurs when a prompt or menu/message is on the screen.

Run mode occurs when DEMO II is running a slide show. You enter run mode by executing the Run Run or Run Debug commands, or by using the RDEMO2 program.

Both short and long descriptions of the edit mode keys are provided. You may want to skim the short form of all of the keys, and then read the long descriptions of just some of the keys.

The keys and their definitions are as follows:

Edit Mode Keys

Short Descriptions

character keys	Overwrites or inserts at the cursor position
arrow keys	Moves the cursor one position in the appropriate direction
^<Left **^Right>**	Moves the cursor to the next "change"
Home **End**	Moves the cursor to the "edge" of the line
^Home **^End**	Moves the cursor to the upper-left/bottom-right corner of the screen
PgUp **PgDn**	Moves the cursor to the next vertical tab stop
^PgUp **^PgDn**	Moves the cursor to the next "change"

Tab
S-Tab, F4 Moves the cursor right/left to the next horizontal tab stop

Enter Moves the cursor to the left margin on next line, or calls up the Block menu

^Enter "Types" the character with value 255 (FF hexadecimal) for a word-wrap New Line

Ins Toggles from overwrite mode to insert mode and back

Del Deletes the character under the cursor

Bkspace Deletes the character before the cursor

Grey +
Grey - Cycles through attributes for a block, or a character position and the Typing Attribute

Grey * Executes the Typing Find command to move the cursor to given characters

Esc Calls up the Main menu

F1 Views previous slide

S-F1 Views first slide

F2 Views next slide

S-F2 Views last slide

F3 Starts line drawing

S-F3 Inserts new, blank slide after current slide

F4 Moves left to the next horizontal tab stop

S-F4 Sets margin at cursor column

F5 "Type special characters" command

S-F5 Inserts new slide after current slide with copy of current slide

F6 Invokes the Block Move command

S-F6 Starts learning a macro

F7 Invokes the Block Delete command

S-F7 Invokes the Block Copy command

F8 Invokes the Block Paste command

S-F8 Invokes the Block CAB command, bringing up CAB menu, showing current setting

F9 Toggles the marked block

S-F9 Invokes the Block Last command, re-marking block, or moving cursor corner

F10 Sets the block/character to the first attribute. Default is inverse

S-F10 Sets the block/character to transparent attribute

Alt-0 - Alt-9
Alt-A - Alt-Z Invokes the appropriate macro

Long Descriptions

character keys Typing a letter, number, or punctuation character overwrites or inserts the character at the cursor position, as with a normal word processor. If the Typing Attribute is not 0, and the CAB setting is Attrib or Both, the Typing Attribute is assigned to that character position. Otherwise, the attribute at the character position remains unchanged. The cursor moves one position forward, not moving past the screen edge. See the Ins key, the Global Typing Attribute setting, the Block CAB menu, the Block Wrap command, and the Typing Direction command for more information.

arrow keys: <Left, Right> Up, Down The left, right, up, and down arrows move the cursor around the screen one character at a time, as you would expect. There is no wrap-around from one line to the next. These are "cursor motion" keys.

^<Left ^Right> The ^<Left and ^Right> (Ctrl-Left Arrow, Ctrl-Right Arrow) keys move the cursor left/right to the next "change" on the row. A change is defined as any of the following: a switch from blank or transparent characters to non-blank, non-transparent characters, a switch from non-blank, non-transparent characters to blank or transparent characters, or a change in the attribute. This is something like a "next word" key, but is more appropriate for DEMO II. (Note that all of these changes refer to the ">THIS SLIDE<" part of the current slide, not any overlays or the background.) The ^PgUp and ^PgDn keys work vertically in an analogous way. These are "cursor motion" keys.

Home End The Home and End keys move the cursor left or right, respectively, on the same row. They move to one position past the last non-blank, non-transparent character position or the edge of the screen, whichever comes first. You use these keys to get to the "edge" of your work. Pressing them twice ensures that you are at the edge of the screen. (Note that it checks only the ">THIS SLIDE<" part of the current slide, not any overlays or the background.) These are "cursor motion" keys.

^Home The ^Home (Ctrl-Home) key moves the cursor to the upper-left corner of the screen. This is a "cursor motion" key.

^End The ^End (Ctrl-End) key moves the cursor to the bottom-right corner of the screen. This is a "cursor motion" key.

PgUp
PgDn Pressing the PgUp or PgDn keys moves the cursor up/down to the next vertical tab stop. See the Typing VTabs command. These are "cursor motion" keys.

^PgUp
^PgDn The ^PgUp and ^PgDn (Ctrl-PgUp, Ctrl-PgDn) keys move the cursor up/down to the next "change" in the column. A change is defined as any of the following: a switch from blank or transparent characters to non-blank, non-transparent characters, a switch from non-blank, non-transparent characters to blank or transparent characters, or a change in the attribute. (Note that all of these changes refer to the ">THIS SLIDE<" part of the current slide, not any overlays or the background.) The ^<Left and ^Right> keys work horizontally in an analogous way. These are "cursor motion" keys.

Tab
S-Tab, F4 These keys move the cursor right/left to the next horizontal tab stop. The Tab key moves right, the S-Tab (Shift-Tab) and F4 move left. The F4 function key is defined to be a Shift-Tab for your typing convenience, since it is immediately to the left of the Tab key on many keyboards and you do not have to press a shift key. See the Typing HTabs command. These are "cursor motion" keys.

Enter If no block is marked, the Enter key moves the cursor to the Left Margin on the next row. By default the Left Margin starts as the left edge of the screen.

If a block is marked (i.e., the block outline is shown on the screen), the Enter key invokes the Block menu as if you had pressed "Esc B". This is provided as a typing shortcut and for people who instinctively "finish" a block definition with Enter. In many cases, you can execute block commands by just pressing a function key when a block is marked, instead of calling up the Block menu.

^Enter The Ctrl-Enter key "types" the character with value 255 (FF hexadecimal). Normally displayed as a blank, this character is interpreted as a forced end of line when word wrapping. See the Block Wrap command.

Ins The Ins key toggles from overwrite mode (the default) to insert typing mode, and back again. In overwrite mode, just the character position under the cursor is affected. In insert mode characters and attributes are pushed to the right, depending upon the CAB setting. If the CAB setting is Character, then only the characters on the line are affected; if the CAB setting is Attribute or Both, then characters and attributes are affected. See the character keys (above) and the Block CAB menu.

The Status Indicator shows "Ins" when you are in insert mode. By default, the Status Indicator displays just this information. See the Typing Status command.

To restrict insert to less than the entire line (so it only pushes part of the line to the right), you may want to use the Block Wrap operation. See the Block Wrap command.

Note

DEMO II does not let you insert characters that would push a non-blank, non-transparent character off the right side of the screen. When you are typing and DEMO II just "thuds" at you, not letting you type, check to see if you are in insert mode by mistake.

Del This key deletes the character position under the cursor. In overwrite mode, the character is set to transparent (zero); if the CAB setting is Attrib or Both, the attribute is set to transparent (zero), too. In insert mode, the character positions to the right of the cursor on the line are pulled left by one character position. Attributes are pulled if the CAB setting is Attrib or Both. To restrict deleting to less than the full line, you may want to use the Block Wrap command. See the Block CAB menu and the Block Wrap command.

Bkspace The Bkspace (Backspace) key deletes the character before the cursor. In overwrite mode, the character is replaced by a transparent character; if the CAB setting is Attrib or Both, the attribute is set to transparent. In insert mode, characters (and attributes, if the CAB setting is Attrib or Both) are pulled from the right. To restrict deleting to less than the full line, you may want to use the Block Wrap command. See the Block CAB menu, the Typing Direction command, and the Block Wrap command.

Grey +
Grey - These keys change the attributes for the marked block, or for a single character and the Global Typing Attribute. They let you cycle through a list of attributes by pressing them repeatedly until you see the colors/effect that you want.

The Grey + and Grey - keys are usually on the numeric keypad, to the right of the main keyboard. They should not be confused with the + and - keys, on the main part of the keyboard near the backspace key, that are the same color as the alphabetic keys.

One of two attribute lists is used. If the Global Grey +/- Cycles setting is "Attributes in List" (the default), then the list on the Block Attribute menu is used. If the setting is "All 256 Attributes", then a list of all attributes from 0 to 255 is used.

If a block is marked, the attribute of the character position under the cursor is examined. If no block is marked, the attribute of the character position immediately to the left of the cursor is examined.

The attribute list is searched from top to bottom to find the first attribute that matches the attribute being examined. Then the attribute in the list that is either immediately after (Grey +) or immediately before (Grey -) that attribute is chosen. The list is considered to wrap around from top to bottom.

If a block is marked, the entire block is set to the chosen attribute. The block remains marked, so that you can invoke this command again. You can unmark the block with the F9 key or the Block Unmark command.

If a block is not marked, the character position ***to the left of the cursor*** is given the chosen attribute, as is the Global Typing Attribute setting.

If the chosen attribute is transparent (00), then a "thud" sound is made. This is to act like a click stop on a dial.

See the Global Typing Attribute setting, the Global Grey +/- Cycles setting, and the Block Attribute menu.

Note

The attribute list is always searched from top to bottom for the first match. If an attribute appears more than once, only the first instance will be found. This can lead to your using just part of the list. Be especially careful when you use the Block Attribute All command to add all 255 attributes to the list. Make sure that you delete any redundant items, or use the Global "All 256 Attributes" setting for the Grey +/- commands.

Grey * The Grey * key executes the Typing Find command to move the cursor to the next occurrence of given characters on a single slide or all slides.

The Grey * key is usually on the numeric keypad to the right of the main keyboard. It should not be confused with the * key on the main part of the keyboard that you get when you press the shifted 8 key.

This command prompts you to get the characters to search for. Then it asks whether it should search just the current slide (T or Enter) or continue and search slides after this slide (C).

The cursor moves to the next occurrence of those characters after the current cursor position. Repeated searches will find successive occurrences of the characters. Note that it checks only the ">THIS SLIDE<" part of the current slide, not any overlays or the background.

Esc The Esc key calls up the Main menu and switches to command mode. The commands on the Main menu are: Block, Typing, Slides, Copy, Overlays, Run, Macro, Global, I/O, Help, and Quit. Each command is described in a separate section of this manual.

F1 The F1 function key switches the current slide to the slide ***before*** this slide in the order of the Slides menu list. You will then be editing that slide. This is the preferred way to switch from one slide to another. To move a larger number of slides back in the list, you may want to use the Slides menu and the commands there.

S-F1 The S-F1 function key (Shift-F1) switches the current slide to the first slide in the Slides menu list (slide number [0001]). You will then be editing that slide.

F2 The F2 function key switches the current slide to the slide ***after*** this slide in the order of the Slides menu list. You will then be editing that slide. This is the preferred way to switch from one slide to another. To move a larger number of slides back in the list, you may want to use the Slides menu and the commands there.

S-F2 The S-F2 function key (Shift-F2) switches the current slide to the last slide in the Slides menu list. You will then be editing that slide.

F3 The F3 function key starts line drawing by executing the Typing Lines command. You "draw" lines on the slide by using the arrow keys and other cursor motion keys. Some of the keys behave a little differently than when editing (see the Typing Lines command description). You end line drawing by pressing Enter or Esc.

S-F3 Shift-F3 inserts a new blank slide after the current slide, then switches to editing that new slide. It puts up an "away next key" prompt when done.

F4 This moves the cursor left to the next horizontal tab stop. See the Tab description.

S-F4 Shift-F4 sets the left margin to the current cursor column. It is the same as the Typing Margin command. The left margin only effects the Enter key while editing. See the Enter key description.

F5 The F5 function key invokes the Typing Chars command to "type" special characters onto the slide. It puts up the Special Chars menu. See the Typing Chars command.

S-F5 Shift-F5 inserts a new blank slide after the current slide, copies the contents of the current slide to this new slide and then switches to editing that new slide. It puts up an "away next key" prompt when done. The copying is the same as that done by the Copy All command. It copies the contents of the ">THIS SLIDE<" or bitmap information, if present, the overlay references, the run actions, the Run Type, the Run Wait, etc.

F6 The F6 function key invokes the Block Move command to move the contents of the marked block to a new position. What is moved is affected by the CAB setting. See the Block Move command and the Block CAB menu.

S-F6 The Shift-F6 key lets you start learning a macro. The display of "L?" in the upper right corner of the screen prompts you for a letter or number identifying the macro. Type the letter or number, and then you will be "learning" that macro. Use Ctrl-Break or Alt-macro character to finish "learning". See the Macro command section of this manual. This key is active in all modes except run mode, which makes it more useful than the Macro Learn command.

F7 This key is the same as the Block Delete command. It deletes the contents of the marked block or the character position under the cursor, if no block is marked. What is deleted is affected by the CAB setting. See the Block Delete command and the Block CAB menu.

S-F7 Shift-F7 is the same as the Block Copy command. See the Block Copy command.

F8 This key invokes the Block Paste command and "pastes" the last text that was Block Deleted or Block Copied. What is pasted is affected by the CAB setting. See the Block Paste command and the Block CAB menu.

S-F8 Shift-F8 invokes the Block CAB command and brings up the CAB menu. The current setting will be highlighted on the menu (Char, Attrib or Both). The CAB setting affects many editing commands, letting you control whether they affect characters, attributes, or both characters and attributes. See the Block CAB menu description.

Note

This is the only function key that is completely changed from the older **Dan Bricklin's Demo Program**. The older program used S-F8 for Block Fill, which is now merged with the Block menu, and it did not have the CAB setting.

F9 The F9 function key toggles the marked block. If there is no marked block, one is started at the current cursor position. If there is a marked block, it is unmarked.

S-F9 The Shift-F9 key executes the Block Last command. If there is no block marked, it re-marks the last marked block. If there is a marked block, it moves the cursor from corner to corner to let you adjust the block on different edges. The block always extends from the cursor to the opposite corner. The cursor moves and the opposite corner stays stationary.

F10 The F10 key sets the currently marked block (or just the character position under the cursor, if no block is marked) to the first attribute in the Block Attribute menu list. The default attribute in that position is inverse video (black on white, value 70).

If you have an attribute that you want to use often, move it to the top of the list; F10 will set characters to its value. You can make a second common attribute easily accessible by selecting it from the Block Attribute menu list. The next time you need to set a block to that attribute, press Enter (to bring up the Block menu), then Enter again (for the Attrib command), and then Enter one last time (to select the highlighted attribute, which will be the same as selected the previous time).

S-F10 The Shift-F10 key sets the currently marked block (or just the character position under the cursor, if no block is marked) to a transparent attribute (zero).

Alt-0 - Alt-9
Alt-A - Alt-Z The Alt-0 through Alt-9 keys, and Alt-A through Alt-Z keys invoke the appropriate macros. See the Macro command.

Command Mode Keys

character keys Pressing a character key selects the command with that character as its initial character. If there is no command with that initial character, and if there is an item in the list with that character as an identifier (on the left, usually a number), it is selected.

<Left, Right> The horizontal arrow keys move the highlight from one command in the menu to another. They wrap from the last to the first.

Up, Down
PgUp, PgDn
Home, End

These keys, called the "list perusal keys", move the highlight in the list from item to item. The up and down arrows move one item at a time. The PgUp and PgDn keys move one "command box-full". The Home key moves to the first item in the list, and the End key moves to the last item in the list.

^Home, ^End

The Ctrl-Home and Ctrl-End keys move the highlight to the first and last command in the menu, respectively.

Tab
S-Tab, F4

These keys let you select the action arguments to edit in the Run menu only. To edit the first argument to a run action, you press Tab. To edit the comment, you press ";". See the Run command for more information.

Enter

The Enter keys select the currently highlighted command, perhaps applying it to the currently highlighted item in the list.

Ins

This key indents the highlighted action in the Run menu only. See the Run command.

Del

This key decreases the indent of the highlighted run action by one space in the Run menu only. See the Run command.

Esc

This key cancels the current command window, and returns either to whatever invoked this command window or edit mode, depending upon the command window.

F1, F2
S-F1, S-F2

In many command windows (such as the Run menu, the Overlays menu, the Slides menu, and others) these keys have the normal effect of changing the current slide. Information relevant to the new slide is displayed. In the Overlays Nums menu, however, these keys move you through the list of overlays for the current slide and do not change the current slide. See the Overlays Nums menu. In all other cases, these keys are ignored.

S-F6

This key functions the same as in edit mode, starting macro "learn" mode.

Alt-0 - Alt-9
Alt-A - Alt-Z

The Alt-0 through Alt-9 keys, and Alt-A through Alt-Z keys invoke the appropriate macros. See the Macro command.

Ctrl-Break

Pressing Ctrl-Break exits the command window and returns to edit mode.

Prompt/Menu/Message Mode Keys

Different prompts accept different keys. Macros (Alt-character) can usually be used.

The "away next key" prompts disappear when any key is pressed and that key is executed normally (usually in edit mode).

The "space to continue" prompts require you to press the space bar to continue; all other keys are ignored.

"Fatal" prompts require you to press Enter, and then exit the program.

The "single character" prompts tell you which characters to choose (such as Y or N, etc.) Just press one of the keys listed; they are case-insensitive. You can press Esc or Ctrl-Break to cancel.

The "type-in string" prompts require you to type some characters and then press Enter. To accept the default shown (if any), just press Enter. To edit the default, start by pressing an editing key. You can edit your response with the left arrow, right arrow, Home, End, backspace and Del keys. You can cancel the operation with Esc or Ctrl-Break. Multiple-line prompts used to set values, which may be variables or strings, accept some other keys, as shown by the prompt (see the "Overview" section for the discussion of Variables).

The menu/messages always list all of the keys that are acceptable. Ctrl-Break may be used to cancel the operation. Only those keys may be used.

Run Mode Keys

The Ctrl-Break key is special to DEMO II in run mode: it terminates running. All other keys are defined by you. See the "How Running Works" section of this manual. The Shift-F6 and Alt-macro keys are treated as normal keys; they do not invoke the macro-processing operations.

The Main Menu

The Main menu is the first command window you will encounter. You get to the Main menu, while editing, by pressing the Esc key. The commands on the Main menu are:

Block Brings up the Block menu. The Block menu contains commands to manipulate blocks of text and attributes. The Block commands are: Attrib, Begin, Last, Unmark, Delete, Copy, Save, Paste, Retrieve, Move, Wrap, Exchg, Xlate, Fill, 1-Box, 2=Box, 3_Box, 4=NoBox, >Cntr, /CAB, and Names. See the "Block Commands" section of this manual for more information.

Typing Brings up the Typing menu. The Typing menu contains commands that are aids to editing. The Typing commands are: Lines, Chars, Direction, HTabs, VTabs, Margin, Status, and Find. See the "Typing Commands" section.

Slides Brings up the Slides command window, showing the list of all of the currently defined slides, their names, numbers, and relative positions. The Slides commands are: View, Undo-Edit, Insert, !Delete, Print, Name, Options, Locate, #, Group, and Move. See the "Slides Commands" section.

Copy Brings up the Copy command window for copying parts of one slide to another. The Copy commands are: All, Overlays, Run, Slide, and Locate. See the "Copy Commands" section.

Overlays Lists and controls the overlays for the current slide. The Overlays commands are: Nums, Slide, Value, Cursor, Adjust, Group, #, Move, Delete, OK, and Paste. See the "Overlays Commands" section.

Run Displays and manipulates the run actions and the Run Type and Run Wait settings for the current slide. Also used to start running the slides. The Run commands are: Line, Action, 1st, 2nd, 3rd, ;Comment, Wait, Type, Tab/Ins/Del, #, Vars, Insert, Key/Event, Delete, Move, Group, Copy, Paste, OK, *Debug, and Run. See the "Run Commands" section.

Macro Puts up a list of all of the allowable macros, their names, and the number of keystrokes stored in each. The Macro commands are: OK, Run, Save, Learn, Extend, View, Name, and Delete. See the "Macro Commands" section.

Global Lists and sets the current values of various global values, such as the background attribute, size of the marked block, etc. Also used to access the Global Run Action list, the Global Overlays list, the Variables list, and the Global Block Names list, as well as to remove the current slide show from the computer's RAM memory. The Global commands are: Edit, OK, ClearAll, Run, Variables, .Overlays, and Names. See the "Global Commands" section.

I/O Puts up a menu with the various commands to input and output the slides and other data. The I/O commands are: Save, Load, Print, Add, Retrieve, Code-Read, and Write-Code. See the "I/O Commands" section.

Help Puts up a simple help screen, listing the actions of the function and editing keys. This is an "away next key" prompt.

Quit Returns to DOS. If DEMO II thinks that you may have made changes that have not been saved to disk, it will ask you if you want to quit and abandon the changes. You must answer Y or N ("Yes, quit and abandon changes" or "No, don't quit").

The Block Commands

The commands on the Block command menu manipulate rectangular blocks of text on the current slide. You can start the definition of a block by using the F9 function key or the Block Begin, Block Last, or Block Names Block commands. DEMO II shows you what is within the currently defined block (called the ***marked block***) by surrounding it with a single-line box. The cursor always defines one corner of the marked block; when you move the cursor, the block's dimensions change. While a block is marked, the characters immediately outside the block are temporarily obscured by this box. When there is a marked block on the screen, most other operations continue as normal, such as typing, commands, and editing. You do operations on the contents of a marked block by invoking one of the Block commands while a block is marked. The operations usually unmark the block when they are completed.

The Block commands fall into several catagories: block marking, attribute manipulation, cut/paste, text-position adjust, and box drawing.

Only one marked block can exist at any given time. You can restore the last block marking with the Block Last command (Shift-F9), and you can keep many block-marking definitions by using the Block Names commands. The named block definitions can be on a per slide basis and global basis (the latter, by using the Global Names command), and they can include a name and a comment for each block definition.

When character positions are deleted, copied, moved, saved, or exchanged, they go into either the Delete/Paste buffer (Block Delete, Copy, Move, and Exchg commands) or a Named Save Area (Block Save command). They may then be retrieved. The Delete/Paste buffer is useful for temporarily holding something — each new Delete/Copy/etc. replaces its contents. The Named Save Areas are more appropriate for blocks that you will want to retrieve repeatedly or keep through a long edit session. They can be given names to aid in keeping them organized. Both types of buffers are saved and loaded by the I/O Save and I/O Load commands.

The box drawing commands let you turn the block marking outline into a real box.

The CAB setting, which is set by the Block /CAB command, affects many of these operations as well as other editing commands. CAB stands for "Character, Attribute, or Both". The CAB setting specifies whether these operations operate on just the character part of each character position, the attribute part, or both the character and attribute part. There are times when you may need all of the combinations, so the CAB setting is provided to give you control. The description for each command affected by the CAB setting explains how it is affected.

Command Descriptions

Block Attrib

Sets the marked block (or just the character under the cursor, if no block is marked) to a specified attribute or manipulates a list of those attributes. An attribute describes how the character appears on the screen, such as inverse video, blinking, or color. There are up to 256 attributes available, which are determined by your monitor and display adapter.

When this command is invoked, another command window appears showing a list of attributes (often called the Attribute List) and commands to manipulate them. Each item in the list has the optional name of the attribute, which you can set; the hexadecimal value (00 through FF, representing the numbers 0 through 255); and a "+" displayed with that attribute. Those commands follow.

Block Attrib Select

Sets all of the character positions in the marked block (or just the character under the cursor if there is no block marked) to the highlighted attribute. This is the most general way to set character positions to a particular attribute. See also the Grey + and Grey - keys, the F10 and Shift-F10 keys, and the Block Xlate command.

Block Attrib Insert

Inserts a new attribute definition item below the highlighted one. The new item defaults to no name and a transparent (zero) value.

Block Attrib Delete

Removes the highlighted attribute definition. If there is a group defined, all definitions in the group are deleted. Character positions with deleted attributes are unaffected.

Block Attrib Move

Repositions the highlighted definition in the list. If a group is defined, the entire group of items is moved. The first item in the list has special meaning to the F10 function key.

Block Attrib Name

Edits or replaces the attribute name. The name can be used to describe the attribute's appearance, such as "Blue/White"; its purpose, such as "Border" or "Error Msgs"; or whatever else you want. It is optional.

Block Attrib Value

Edits the hexadecimal representation of the attribute displayed to the right of the name. Type the appropriate hexadecimal digits (0-9, A-F) or use the up and down arrows to change the value. Use the left and right arrow keys to switch from one digit to another. The Tab key complements the high order bit of the digit (i.e., adding or subtracting 8), which is useful for setting the bold or blinking bits. The word "Sample" is displayed to the right of the digits using the attribute currently defined. Note that the attribute 00 is special -- it is a transparent attribute (see Overlays commands and the "Overview" section about Overlays). Press Enter when the value is what you want. Esc cancels the command and restores the previous contents.

For a list of attributes and how they display, see the technical reference manual for your display adapter, or try varying the value and watching the "Sample". You can also define them all with the Block Attrib All command, which is described in this section, and see the "+" samples. There is also an appendix containing a list of common attribute values.

Three common attributes are usually automatically defined: inverse, normal, and default. For color displays, the first digit usually affects the character's background color, and the other digit affects the character itself. The high-order bit of the digits, which is changed with the Tab key, affects blinking and intensity. On monochrome adapters, "colors" may be only 0 or 7 (0 is black, 7 is white), with a special case of a background of 0 and a foreground of 1 producing underlined characters.

Block Attrib All

Inserts the 255 attributes from 1 to 255 below the highlighted item. Note that there must be at least one item on the list to use this command. You must confirm that you indeed want to do this. You can define the 0 attribute (transparent) by just inserting an item with the Block Attrib Insert command.

After defining all of the attributes, you may want to delete many of them. The Group command aids you in doing that. Also, see the note with the description of the Grey + and Grey - keys in the "Keys" section for a warning about duplicate attribute definitions. (Duplicates only affect that operation.)

Block Attrib Group

Toggles the group start.

Block Begin

Marks the beginning of a block, starting at the current cursor position. The block is the rectangle delimited by this beginning screen position; the cursor defines the opposite corner. Move the cursor to expand or contract the block. While a block is defined, other commands and editing keys can be used normally. The beginning of a block can also be marked by pressing the F9 key when no block is defined.

Block Last

Restores the last block definition, if no block is marked. This allows you to use multiple Block commands on the same block without having to do a redefinition.

If a block is already marked, this command moves the cursor to the next corner (counterclockwise), so that you can adjust the block's dimensions from any corner.

It is identical to the Shift-F9 key.

Block Unmark

Cancels the block definition. If a block is defined, the F9 key does the same thing.

Block Delete

Saves a copy of the marked block in the Delete/Paste buffer, then deletes it from the current slide. The deleted area's characters are made transparent if the CAB setting is Char or Both, and the area's attributes are made transparent if the CAB setting is Attrib or Both. If no block is marked, just the character under the cursor is deleted. This is the same as the F7 key.

Block Copy

Saves a copy of the marked block in the Delete/Paste buffer, which produces the same result as the Shift-F7 key.

Block Save

Acts like Block Copy, but puts the copy in a Named Save Area. This command puts up a command window with a list of currently defined Named Save Areas. If none are defined, you must create one with the Block Insert command.

The Block Save commands follow.

Block Save Select

Saves a copy of the marked block in the Named Save Area selected by the highlight. Instead of moving the highlight and pressing Enter, you can also type the number to the left of the name in order to select it.

Block Save Insert

Creates a new Named Save Area on the line just below the highlighted one. It also moves the highlight to that line.

Block Save Delete

Deletes the highlighted Named Save Area.

Block Save Move

Repositions the highlighted Named Save Area. Use the list perusal keys to move. Press Enter when the list is in the order you like, or use Esc to cancel the command.

Block Save Name

Edits the name of the Named Save Area. Type the new name or use the editing keys to change the current one. Esc cancels the edit and restores the original contents.

Block Paste

Retrieves a copy of the last deleted/copied block from the Delete/Paste buffer, and uses it to overwrite the area starting at the cursor. A message pops up telling you that you are Pasting. Press Enter if the text to be pasted is where you want it. If not, use the cursor keys to move the cursor and the block being pasted until it is where you want it. You can cancel the command at any time with the Esc key. The F8 key is the same as the Block Paste command. If the CAB setting is Char or Both, characters are pasted. If the CAB setting is Attrib or Both, attributes are pasted. Note that copies of both characters and attributes are always put in the Delete/Paste buffer, so pasting is unaffected by the state of the CAB setting when the Delete/Paste buffer was last filled.

Block Retrieve

Acts like Block Paste, but requests the name of a Named Save Area by putting up a command window similar to that of the Block Save command. When you issue the Select command on that menu, it acts in a manner similar to the Block Paste command, displaying a message and allowing you to confirm the Retrieve and its position.

Block Move

Acts like Block Delete followed immediately by Block Paste, with the cursor position moving to the upper-left corner of the block before the Paste. This is very useful for repositioning blocks of text. Note that a copy of the moved text is left in the Delete/Paste buffer. Also, note that the location definition of the "Last" block (see Block Last command) is moved along with the block, so that the next Block Last command will refer to the new location. The F6 key is the same as the Block Move command.

Block Wrap

Word wraps the contents of a block, and lets you type new text and have that continuously word wrapped as you type. This is a very useful feature for typing comments and text in boxes in tutorials and demonstrations.

When you execute this command, the currently marked block is made into a "word wrap block". The corners of the block are displayed as boxes, rather than corners, to indicate a word wrap block. The text in the block is word wrapped so that there is always a space in the last column of the block. The width of the block always stays the same, but the bottom extent moves up and down as necessary to fit all of the text. If the CAB setting is Attrib or Both, all of the attributes in the block are set the same as the upper left-most character position in the block. All lines are indented from the left side of the block by the same number of spaces as the first line. The character number 255 (FF hexadecimal), which displays as a blank and can be typed by pressing Ctrl-Enter, is treated as a "hard carriage return". This forces the start of a new line.

As long as the word wrap block is defined, any typing, backspacing, or deleting within the block will cause it to be word wrapped again. *The cursor keys may be used.* Typing is always done in insert mode automatically within the block, no matter what the normal mode. To end a word wrap block and leave the text as it is, you unmark the block by pressing F9. You can also use any Block command that unmarks a block or defines a new block.

You can start word wrapping by pressing F9, typing the first line of text, then pressing Enter and a "W" to invoke the Block Wrap command. Then you continue typing until the block is done, making explicit new lines with the Ctrl-Enter key. Finish by either pressing F9, putting a box around the text (Enter "1" or Enter "2"), centering the text (Enter ">"), or adjusting its size by first pressing Shift-F9 to make it a normal marked block again.

Large blocks may be slow to wrap, so you may want to wrap a large block one paragraph at a time, or only after making changes.

Block Exchg

Combines Block Delete and Block Paste to exchange the contents of the Delete/Paste buffer with the marked block. Only the character positions inside of the marked block are affected. The CAB setting determines what is deleted and what is pasted. To undo this operation, re-mark the block with Shift-F9 and do it again (assuming the blocks exchanged were the same size).

Block Xlate

Converts all of one attribute value into another, which is known in programming terms as "translating" the attributes. Executing the Block Xlate command brings up the Block Xlate menu. This command window displays the **Attribute Translate Table**. There are 256 items in the Attribute Translate Table, one for each of the 256 possible attributes. The items specify how to convert each attribute, giving the old and new value on the left and right, respectively. The highlight starts on the value that corresponds to the attribute of the character position under the cursor.

The table is used to translate the attributes on the slides permanently, using the Block Xlate Block, Block Xlate Slide and Block Xlate All commands, or temporarily on the display only, using the Translate Attribute Cmd run action.

The commands on the Block Xlate menu follow.

Block Xlate Value

Edits the hexadecimal value on the right, which is the value of the attribute that all character positions with the value on the left of the item are translated into. Works in a manner similar to the Block Attrib Value command.

Block Xlate Mono

Sets all attributes in the list to convert into normal white-on-black (07), except transparent (0) and inverse (black-on-white, 70), which are left as no translation. This is a way to start converting a full-color slide show into all monochrome If there is a group, only the items in the group are set to monochrome translation.

Block
Xlate
Reset

Sets all attributes in the list to have no translation; that is, to convert to themselves, remaining unchanged after translation. You may want to execute this before specifying a translation after you have done another. If there is a group, only the items in the group are reset.

Block
Xlate
Group

Toggles the group start.

Block
Xlate
#

Lets you move to a specified item. Prompts for the item number, which is the second part of each line.

Block
Xlate
Block

Translates all of the attributes in the currently marked block as specified by the Attribute Translate Table.

Block
Xlate
Slide

Translates all of the attributes on the current slide as specified by the Attribute Translate Table.

Block
Xlate
All

Translates the attributes on all of the slides in the slide show as specified by the Attribute Translate Table. This command requests confirmation.

Block
Fill

Fills the marked block with the character and/or attribute in the upper left corner, overwriting anything else in the block. The CAB setting determines what gets changed: the character, attribute, or both.

Block
1-Box

Outlines the marked block with a single line box, similar to that of the block definition outline. Uses the same logic as the Typing Line command. Type "1" to select this command.

Block
2=Box

Outlines the marked block with a double-line box, similar to that of the block definition outline, but using the double-line drawing characters. This command uses the same logic as the Typing Line command. Type "2" to select this command.

Block
3_Box

Outlines the marked block with a heavy box, similar to that of the block definition outline, but using the character-width block characters. Type "3" to select this command.

Block
4=NoBox

Outlines the marked block with blanks, removing any box around it. Use this command to erase a box around a block, such as before typing in new text and word wrapping it. Type "4" to select this command.

Block
>Cntr

Centers the characters of each row within the block. The attributes are unaffected. The program centers all text surrounded by blanks or transparent (0) characters. The surrounding blank or transparent characters determine whether the "padding" will be done with blanks or transparent characters.

A common operation is to word wrap a block with the Block Wrap command and then center the lines with this command.

Block /CAB

Calls up the CAB menu. The command that corresponds to the current CAB setting will be highlighted. (Normally a menu comes up with the first command highlighted; this is an exception.) Type "/" to select this command. This command is the same as the Shift-F8 key.

The CAB setting affects many editing and block operations. CAB stands for "Character, Attribute, or Both". The CAB setting specifies whether these operations operate on just the character part of each character position, the attribute part, or both the character and attribute part. There are times when you may need all of the combinations, so the CAB setting is provided to give you control. The description for each command affected by the CAB setting explains how it is affected by the CAB setting.

The commands on the CAB menu follow.

Block /CAB Char

Sets the CAB setting to "Character".

Block /CAB Attrib

Sets the CAB setting to "Attribute".

Block /CAB Both

Sets the CAB setting to "Both".

Block Names

Displays the Block Names menu with the block names for the current slide. Each item in the list is a block name definition. On the left is the extent of the block, shown in the form "aa/bb:cc/dd", where aa/bb is the row/column of one corner of the block and cc/dd is the opposite corner. On the right is the optional eight character name of the block. The block name may be followed by a "(C)", meaning there is a comment associated with the block.

Note that there is also a Global Block Names menu with block names for the entire slide show.

Named blocks can be used for several purposes. They can be used to quickly re-mark any of a number of blocks, such as for translating attributes or word wrapping. They can be used to annotate a slide show so that future developers can figure out what things are for — either in a complicated slide show or to explain what a prototype is in greater detail than shows on the screen. Block names can be listed on printouts of the slides by using the I/O Print command. By printing to a file, you can use block names to have field definition information that is used by another program that reads the print file. This program could convert slides into forms definitions for some database system, for example. Block names give you a place to put the extra information you may need.

The Block Names commands follow.

Block Names Block

Uses the highlighted block name definition to set a marked block.

Block Names Comment

Edits the comment associated with the highlighted block. Prompts for a comment of up to 70 characters.

Block Names Position

Changes the highlighted item's block-position definition to be that of the currently marked block.

Block Names Name

Edits the name of the highlighted block.

Block Names Insert

Inserts a new block name definition item below the highlight. The currently marked block's definition is used for the "aa/bb:cc/dd" value.

Block Names #

Moves to the specified item.

Block Names Group

Toggles the group start.

Block Names Delete

Deletes the highlighted item. If there is a group defined, all items are deleted.

Block Names Move

Repositions the highlighted item in the list. If a group is defined, the entire group of items is moved.

The Typing Commands

The Typing commands helps you to use the special characters of the PC character set and position the cursor. The commands are:

Typing Line

Switches into line drawing mode. In this mode, moving the cursor causes the program to "follow" the cursor with line drawing characters. If the cursor "crosses" an already drawn line, the proper crossing character is put on the screen, if it exists. (The PC character set does not have all of the combinations you may need.)

If you press "1", a single line will be drawn. If you press "2", a double line will be drawn. If you press "+", "?" or "=", the character currently under the cursor is used (and will be repeated over and over again as the cursor is moved). The arrows and various tabs move the cursor, leaving a trail of the appropriate number of characters. Backspace works as you would expect, with it knowing the direction you last moved. Del or Space may be used to erase the character under the cursor without moving. Esc or Enter both return you to edit mode. The Typing Lines command is the same as the F3 key.

Note that line drawing "sees" only the characters on the ">THIS SLIDE<" part of the current slide; it does not see characters on overlays.

Note also that the Ctrl-Left and Ctrl-Right arrows move to the next minor tab stop (columns 1, 6, 11, etc.) instead of the normal "next change". The Home key moves to column 1 at the left edge of the screen. The End key moves to column 80, at the right edge. Also, the Ctrl-PgUp and Ctrl-PgDn keys are ignored. Because you usually draw lines where there is no other text, these other definitions are more appropriate.

Typing Chars

Puts up a command window with a list of the special characters available in the PC character set, with their decimal and hexadecimal values displayed to their right. All normal typing characters have been removed from the list. Use the List Perusal keys to scan the list. Pressing Enter to select the highlighted item or typing the character (a digit or letter) to the left of a list item causes the special character in that item to be "typed" at the cursor. The command window is removed from the screen, but if you invoke the Typing Chars command again, the highlight starts on the character you last chose. The Typing Chars command is the same as the F5 key.

The first five items in the list are copies of the five most recently used special characters. They are there to help you find commonly used values quickly.

The Typing Chars commands follow.

Typing Chars Select

"Types" the selected character.

Typing Chars Num

Prompts for a decimal value (0 to 255) representing the character to be "typed". This command is useful for macros that need to select a particular character.

Typing Direction

Changes the direction the cursor moves when you type text. The backspace key is adjusted accordingly. This command puts up a menu to let you choose Right (the normal way), Left, Up, or Down. This is useful for typing titles, boxes, row numbers, etc.

Typing HTabs

Puts up a menu for Setting/Clearing the horizontal tab stops (which are used by the Tab and Shift-Tab keys).

The Typing HTabs commands follow.

Typing HTabs Set

Sets a tab stop at the cursor column.

Typing HTabs Clear

Clears the tab stop (if any) at the cursor column.

Typing HTabs All

Clears all horizontal tab stops.

Typing HTabs Default

Clears all horizontal tab stops, then sets the standard defaults (1, 16, 31, etc.).

Typing VTabs

Displays a menu for Setting/Clearing the vertical tab stops (which are used by the PgUp and PgDn keys).

The Typing VTabs commands follow.

Typing VTabs Set

Sets a tab stop at the cursor row.

Typing VTabs Clear

Clears the tab stop (if any) at the cursor row.

Typing Vtabs All

Clears all vertical tab stops.

Typing Vtabs Default

Clears all vertical tab stops, then sets the standard defaults (1, 6, 11, etc.).

Typing Margin

Sets the column where the cursor positions itself on the next row when you press Enter in edit mode. The margin column is set to the current cursor column. This is useful for typing blocks of text that do not start in column 1. It is independent of the tab stops. The Shift-F4 key is the same as the Typing Margin command.

Sometimes when you are typing aligned text you may find the Block Wrap command useful, instead of using the Typing Margin.

Typing Status

Configures a status display that can appear in the upper-right corner of the screen. The Status Display, by default, is shown with only the Ins/Ovr information displayed (so it is normally blank).

The Status Display is in the form:

```
rr-cc slide-name-number filename Ins
```

where "rr-cc" is the row and column of the cursor separated by an arrow showing the current Typing Direction; "slide-name-number" is the name and number of the current slide; "filename" is the name of the file, if set; and "Ins" is present if you are editing in insert mode (as opposed to overwrite mode).

The Typing Status command puts up another menu with the following commands:

Typing Status Hide

Toggles whether or not the Status Display is shown. Defaults to Yes.

Typing Status Cursor

Toggles the cursor row/column Status Display on and off. Defaults to No.

Typing Status Slide

Toggles the slide-name-number Status Display on and off. Defaults to No.

Typing Status File

Toggles the file name Status Display on and off. Defaults to No.

Typing Status Ins/Ovr

Toggles the Insert/Overwrite Status Display on and off. Defaults to Yes.

Typing Find

Moves the cursor forward to the next screen position with specified characters. This is the same as the Grey * key.

This command prompts you to get the characters to search for. Then it asks whether it should search just the current slide (T or Enter) or continue to search slides after this slide (C).

The cursor moves to the next occurrence of those characters after the current cursor position. Repeated searches will find successive occurrences of the characters. Note that it checks on the ">THIS SLIDE<" part of the current slide, not any overlays or the background.

The Slides Commands

These commands allow you to name, view, create, delete, and modify information about the slides stored in RAM memory. They appear in a command window that displays a list of the names (if present) and the relative order of the slides. The Slides commands are:

Slides View

Changes the screen display to the chosen slide. You either invoke this command while highlighting the desired slide, or type the number to the left of the slide name. A message appears displaying the name of the slide on the screen. You can start editing the slide on the screen by pressing Enter, move to the slide before it or after it on the Slides list by pressing F1 or F2, go to the first or last slides by pressing Shift-F1 or Shift-F2, or return to the Slides list display by pressing Esc.

Slides Undo-Edit

Restores the slide currently on the screen to the way it was the last time you executed a Slides View command, F1, F2, Shift-F1, or Shift-F2 (the Next, Previous, First, and Last slide keys), or any of a variety of commands, such as I/O Save, Print, etc. It will not retrieve a slide deleted with the !Delete command; it just gives you a simple form of Undo. It is good practice to briefly view another slide (with a quick F1/F2 or Esc S Enter) before you do an edit that you may want to undo, such as Block Fill. You cannot undo an Undo-Edit.

Slides Insert

Creates a new slide following the highlighted slide. All slides are renumbered. The Shift-F3 key inserts a blank slide after the current slide in a similar manner.

Slides !Delete

Deletes the highlighted slide. If there is a group defined, then the entire group of slides is deleted. The slides are then renumbered. It is good practice to first Slides View the slide being deleted, Esc out of the view, and then delete it. Note that you have to press ! to invoke this command. The ! is harder to type, so you will be less likely to do it accidentally. The ! is a shifted key, while most other commands are not. You cannot undo a !Delete.

Slides Print

Toggles the Print Flag setting for the highlighted slide. If there is a group defined, the entire group is set like the first item in the group. The Print Flag indicator, "(P)", appears next to the slide's number if it is "On". You can set the I/O Print command to print only slides that have the Print Flag setting "On". It defaults to "Off".

Slides Name

Edits or replaces the name of the highlighted slide.

Slides Options

Displays the Slides Options information. See the Slide Options subsection below for a description of this information.

The Slides Options commands follow.

Slides Options Value

Modifies the values associated with the second, third, and (if present) fourth items. Puts up a variable/value type-in prompt with information about the values that are needed. See the Slide Options subsection for information about the Switch Type and Switch Speed values.

Slides Options Text

Changes a bitmapped slide into a text slide, deleting all of the bitmapped image information. This change cannot be undone.

Slides Options Palette

Displays the Video Bits and EGA Color Palette information. See the Slide Options subsection below.

Slides Options OK

Returns to the Slides menu.

Slides Options F1, F2 S-F1, S-F2

Switches the slide whose options are being shown. This does not change which is the current slide.

Slides Locate

Finds the next slide below the highlighted one whose name starts with given characters. A "type-in string" prompt appears in the middle of the screen showing the last name given. You may replace it by typing new characters, edit it, or use it again. Press Enter to start the search. The highlight will be repositioned over the next slide whose name starts with the characters. The search is uppercase and lowercase sensitive. If no match is found, the highlight will be left on the last slide. If you want it to wrap, you can use the Home key to move to the first slide in the list; then you can reissue the Slides Locate command.

Slides #

Moves the highlight to the slide with the specified position in the list.

Slides Group

Toggles the group setting.

Slides Move

Changes the order of the slides. Use the List Perusal keys to move the highlighted slide (or, if there is a group, the entire group of slides). The slides will be renumbered when you press Enter. The order is important to running and the F1/F2 keys. Cancel the command with Esc.

Slides F1, F2 S-F1, S-F2

To change the current slide, the F1, F2, Shift-F1, and Shift-F2 keys can be used while the Slides Menu is being displayed.

Slide Options

Associated with each slide is some extra information used in displaying the characters or pixels on the screen. This information includes whether a slide has text or bitmapped images (Slide Type); how to switch to this slide from the previous slide and at what speed (Switch Type and Switch Speed); and hardware settings that take fuller advantage of the PC's attributes and color palettes (Palette).

Slide Type

The Slide Type can be either "Normal Text" or "Bitmapped Graphics". Slides are always "Normal Text" unless they are created with the I/O Retrieve command or copied, using Shift-F5 or Copy All, from a bitmapped slide. Bitmapped slides can be made into normal text slides with the Slides Options Text command; they cannot, however, be changed back. Bitmapped slides also indicate whether they were retrieved from the Capture program or if they have had their images read in from a PCX file.

Switch Type
Switch Speed

All slides have a Switch Type. By default the Switch Type is >NONE<, which is the same as zero. Zero Switch Type means to switch to this slide as quickly as possible. Some bitmapped slides can only have Switch Type 0, but most others can have Switch Types 1 and 2, also. Switch Type 1 means to replace the previous image on the screen with the image of this slide from top to bottom at "Switch Speed". Switch Type 2 is the same as 1, but from bottom to top. The information on Switch Speeds for bitmapped slides is in the "Bitmapped Graphics Images" section of this manual.

Text slides can have Switch Types 0, 1, 2, 3, and 4. Switch Types 1 and 2 replace the images from top to bottom and bottom to top, respectively. The Switch Speed is a number from 0 to 25 or more, and specifies the number of vertical retraces to pause between each line of text displayed (a vertical retrace is about 1/50-1/60 of a second). Switch Types 3 and 4 replace all of the characters and attributes that are different between the image already on the screen and the new slide from top to bottom. Switch Type 4 toggles the speaker to make a "click" after every changed item is displayed. The Switch speed can be any number from 0 to 32,000, and it represents the time to wait between displaying the different characters. The time will appear about the same on all machines since the wait mechanism uses the Relative CPU Speed setting (see the Run Actions Builtin(4) description). You should experiment with values (such as 100, 1000, 5000) to see which one is appropriate to your needs.

Switch Types 3 and 4 are very useful for simulating typing, fill-in-the-forms, and the appearance of using a communications line. By starting from a blank screen, you can have the whole slide come on slowly. By starting from a blank form and then switching to a form that is filled in, you can get the appearance of a user filling in the form.

Palette
Video Bits

Associated with each slide is its Palette settings. ***Most of the Palette information is quite technical and if you do not understand it, it may not be necessary for your work. Using the Palette information requires a good understanding of the PC's display adapters from a very programmer-oriented level.***

When you execute the Slides Options Palette command, a type-in prompt will be displayed showing the current setting of the slide's "**Video Bits**". The Video Bits are defined as follows:

Bit 0	Blue 3x9 Register Bit
Bit 1	Green 3x9 Register Bit
Bit 2	Red 3x9 Register Bit
Bit 3	Intensified 3x9 Register Bit
Bit 4	Alternate 3x9 Register Bit
Bit 5	Color Set: 1=Cyan/Magenta/White, 0=Green/Red/Brown
Bit 6	Char Backround Attrib High Bit: 1=Blink, 0=Intensity
Bit 7	Unused, set to 0

You can modify the value if you want, or leave it alone by just pressing Enter or Esc. Note that the value is displayed in hexadecimal (0x??) and you can type in hexadecimal values by leaving the "0x" prefix.

You can change the border colors on text slides for CGA systems by modifying the Blue/Red/Green bits (bit 0 has value 1, bit 1 value 2, bit 2 value 4, etc.). You can change whether the high order bit of a character position attribute specifies blinking or background color intensity by modifying bit 6 (0x40). You can change the color combinations used by 320x200 bitmapped slides by modifying bit 5 (0x20).

The Video Bits for a noncaptured slide are set from the Default Video Bits setting, shown on the Global menu. It is "0x70" by default. Captured slides get the settings that existed when the capturing was done.

If this is a bitmapped image retrieved from Capture and you press Enter, the Video Bits value will be followed by a type-in prompt for "Palette Entry ?". If this is non-zero, then the following type-in prompt values are valid. (If it is non-zero, the other palette information will be ignored.) The following 17 values that appear (press Enter to go from one to another, Esc to stop) are used to set the EGA palette registers. They determine the colors that the EGA actually displays. You should not modify these unless you understand the EGA. They are provided for those who need them.

If you modify any palette settings, including Video Bits, you should check the results on a variety of different systems to see the results — EGA systems act differently than CGA systems sometimes, etc.

Note

Some display adapters (such as the VGA) briefly blank the screen when the Video Bits and other palette settings are changed. Try to keep the Video Bits setting the same so that DEMO II doesn't have to change them when switching from slide to slide. The "0x70" default setting is the same usually used by most programs you will Capture. If you suspect that you are getting this flickering, you can always check by capturing one slide, retrieving it, and then looking at the Video Bits on the slide with the retrieved image. If it is different than "0x70", you may want to change the Default Video Bits setting on the Global menu to be that new value, so that new slides will have that value. You can set the existing slides' Video Bits with the Slides Options Palette command.

The Copy Commands

These commands let you copy the items associated with one slide to another slide. You can copy the ">THIS SLIDE<" part, the overlay references, the run actions and settings, or all of the slide's parts, including bitmapped images. The Copy command displays a list of all slides, with the highlight starting on the slide before the current slide.

Commands

Copy All

Copies all parts of the highlighted slide to the current slide. It pastes the ">THIS SLIDE<" part on the current slide, adds all of the overlay references to the overlay list, adds all of the run actions to the current slide, and copies the Run Type and Run Wait setting. It also copies any bitmapped image, Switch Type, Switch Speed, Block Names and Palette information. The Overlay and ">THIS SLIDE<" part require confirmation (it uses the same program code as the Copy Overlays and Copy Slide commands) and are executed after the other operations.

This mechanism is used by the Shift-F5 key to create a new slide that is a copy of the current slide.

Copy Overlays

Inserts the overlay references from the highlighted slide into the current slide's overlay list. This requires confirmation.

Copy Run

Copies the highlighted slide's Run Type, Run Wait, Switch Type, Switch Speed, Block Names, Video Bits, and Run Actions. The Run Actions are inserted at the end of the slide's run action list.

Copy Slide

Pastes the highlighted slide's ">THIS SLIDE<" part on the current slide. This requires confirmation.

Copy Locate

Finds the next slide below the highlighted one whose name starts with given characters. Similar to the Slides Locate command.

The Overlays Commands

The Overlays commands let you define and manipulate a slide's overlay list. The concept of overlays is described in detail in the "Overview" section of this manual, and the different types of overlays are described in the "Types of Overlays" section. You should be familiar with those sections before using overlays. For some applications, it is also important to understand the use of variables in DEMO II, which is also discussed in the "Overview" section.

The Overlays command puts up the Overlays menu with a list of the current slide's overlays. The order in the list is the order in which the overlays are displayed in front of the background. Each item shows what is being referenced by the overlay and the overlay type.

There are two types of Overlays menus. One is the normal Overlays menu for each slide that you display by pressing "Esc O". The other is the Global Overlays menu that you display by pressing "Esc G .". Note that you need to press the period (".") for the Global .Overlays command.

The Overlays commands are:

Overlays Nums

Puts up the Overlays Numbers menu which shows the settings associated with the highlighted overlay item. See the "Types of Overlays" section for a description of these settings.

The Overlays Nums commands follow.

Overlays Nums Value

Edits the value of the highlighted setting. The "Row Offset", "Column Offset", "Visible", "Max Chars Shown", and "Scan Lines Desc" items put up a type-in prompt for the constant or variable value for the setting. See the "Overview" section's subsection on Variables for more information about how to use that kind of type-in prompt.

This command cycles the "Type" setting for Value slides through the different types: "Normal" (plain String Value or Numeric Value Overlays), "w/ H/W Cursor" (String or Numeric Value Overlays with H/W Cursor), "Abs Slide Ref" (ABSREF), and "Rel Slide Ref" (RELREF). This is how you create those types of overlays, starting from a normal String or Numeric Value Overlay created with the Overlays Value command and then cycling to the desired type.

Note

When you use variables to change the appearance of an overlay (such as modifying its position or making it visible and invisible), changes to the variables' values will not appear on the screen until the next redisplay. This occurs when a slide is next viewed, when new input is requested on the current slide, and when an explicit Redisplay run action is executed. See the "How Running Works" section.

Forgetting to do a redisplay in the middle of a loop is one of the most common mistakes when using overlays with variables. If the effect you are trying to produce looks correct when you run in Debug mode, but not normally, you probably are missing a Redisplay somewhere.

**Overlays
Nums
OK**

Returns to the Overlays menu.

**Overlays
Nums
F1, F2
S-F1, S-F2**

Switches the settings being displayed from one overlay to another ***on the same overlays list***. While this is similar to the normal use of F1/F2 to move through the list of slides, it just moves through the numbers for the current list of overlays and does not change the current slide. This is the only command that has a different definition for F1/F2, but it should feel natural and is very helpful.

**Overlays
Slide**

Inserts a new Slide Overlay definition below the highlight, or creates the first one if none are defined. A menu/message pops up proposing a slide and constant offsets, and it shows how the screen would look with that slide actually used as the overlay. You can then use the arrow keys to change the offsets, press Home to reset the offsets to 0, or F1/F2 to change which slide is referenced by the Slide Overlay. You can also bring up a simplified version of the Slides menu to aid you in finding a slide by pressing "S". Pressing "T" makes the slide reference be to ">THIS SLIDE<". Finally, you can press Enter to use the offsets and slide shown, or use Esc to cancel and return the Overlay to its original status.

**Overlays
Value**

Inserts a new Value Overlay definition below the highlight, or creates the first one if none are defined. A type-in prompt appears so that you can specify the value to be used for the overlay. Then a menu/message pops up showing the value chosen and proposing constant offsets. The overlay's position is represented on the screen by a string of 80 characters with the pattern "1...VAL...11..VAL...21..". The numbers are positioned like a ruler, so you can see where 37 characters would be, for example.

You can then use the arrow keys to change the offsets or press the Home key to reset the offsets to 0. You can also bring up the type-in prompt again to respecify the value by pressing "V". Finally, you can press Enter to use the offsets and value shown, or Esc to cancel any offset changes and return the Overlay to its original state. ("V" operations are not cancelled.)

Note

When you use Value Overlays, any changes to the variables will not appear on the screen until the next redisplay. This occurs when a slide is next viewed, when new input is requested on the current slide, and when an explicit Redisplay run action is executed. See the "How Running Works" section.

Forgetting to do a redisplay in the middle of a loop is one of the most common mistakes when using variable overlays. If the effect you are trying to produce looks correct when you run in Debug mode, but not normally, you probably are missing a Redisplay somewhere.

**Overlays
Cursor**

Inserts a new H/W Cursor Overlay definition below the highlight, or creates the first one if none are defined. A menu/message pops up proposing constant offsets, which starts at the DEMO II cursor location, and shows how the screen would look with those offsets. You can then use the arrow keys to change the offsets or press Home to reset the offsets to 0. Pressing Enter leaves the new offset definitions; pressing Esc cancels them.

Overlays Adjust

Adjusts the constant row and column offsets as well as changes the slide or value referenced by the highlighted overlay definition. It puts up a menu/message similiar to that shown when you create the overlay definition. See the descriptions for the Overlays Slide, Overlays Value, and Overlays Cursor commands.

Overlays Group

Toggles the group setting.

Overlays #

Moves the highlight to the specified item in the overlays list.

Overlays Move

Repositions the highlighted item. If there is a group, then the entire group of overlay definitions is moved. The order of overlays in the list is important. Characters and attributes on overlays lower in the list can obscure characters and attributes above them. See the discussion of overlays in the "Overview" section of this manual.

Overlays Delete

Erases the highlighted overlay definition. If there is a group, the entire group of definitions is erased. This operation has no effect on the referenced slides or values.

Overlays OK

Returns to edit mode.

Overlays Paste

Removes the highlighted overlay definition and "pastes" a copy of the referenced slide in the same position onto the ">THIS SLIDE<" part of the current slide. This converts an overlay reference, which changes in appearance when the referenced slide is modified, into a fixed copy. The fixed copy does not change and can be edited as part of the slide. This command is affected by the CAB setting.

This operation is only applicable to Slide Overlays, and appears only on the normal Overlays menu, not the Global Overlays menu.

Overlays F1, F2 S-F1, S-F2

Switches the current slide to another slide and displays its overlays list. These keys can be used while the Overlays menu is displayed.

The Run Commands

The Run commands let you set the Run Type and Run Wait associated with a slide. In addition, you can create or edit the action lists associated with each slide. Together, these settings and action lists control how DEMO II switches from slide to slide when running a slide show, as well as how it modifies variables, produces sounds, and other effects. Two of the Run commands let you start running the slide show: one normally, and the other in a debugging mode for testing purposes.

Before using the Run commands, you should be familiar with the concept of running and run action lists. These are described in the "Overview" section as well as the "How Running Works" section of this manual. You should also be familiar with how run actions are displayed, which is described in the first subsection below.

In addition to the normal Run commands that operate on the slides' action lists and settings, there is a Global Run command that operates on the Global Run Action list. Both commands work in a similar manner.

How Run Actions Are Entered, Edited, and Displayed

DEMO II displays and manipulates run actions in its own unique way. Because run action lists are similar to programs, you might expect them to be entered and edited by a normal text program "editor"; they are not. Each run action is entered and edited using a menu system similar to the rest of DEMO II. A few extra keys are accepted to make the process more efficient and natural.

The reason that DEMO II uses a menu system for entering and editing run actions stems from the way in which the program is used. Most operations can be done in one or two run actions, such as just viewing another slide or sounding a tone. The selection of which run actions to execute is usually done by the DEMO II Key/Event signaling mechanism, which requires very little program writing — just the setting of Key/Event labels. In most cases you are just choosing Key/Event labels or slides to view out of long lists of alternatives. This method of operation is best served by a menu mechanism rather than a normal program editor. Plus, you do not have to move from one environment to another to change the few actions usually associated with most slides.

Even when you get into more complicated programming, the benefits of a menu system appear to outweigh the benefits of a normal editor for most users. Since large programs are infrequently written for DEMO II, you probably will not get that much practice to help you learn the format of many of the run actions. The particular menu system used ensures that syntactically correct programs are always written and provides the equivalent of an on-line "reference card". The menu system is also very keystroke-efficient, letting you create programs quickly.

Let's look at a typical run action line and see how the menu system is used to create and edit it.

The run action line in Figure 1 has a Key/Event label, the run action Tone Note, which sounds a tone the next time the screen is redisplayed, and a comment:

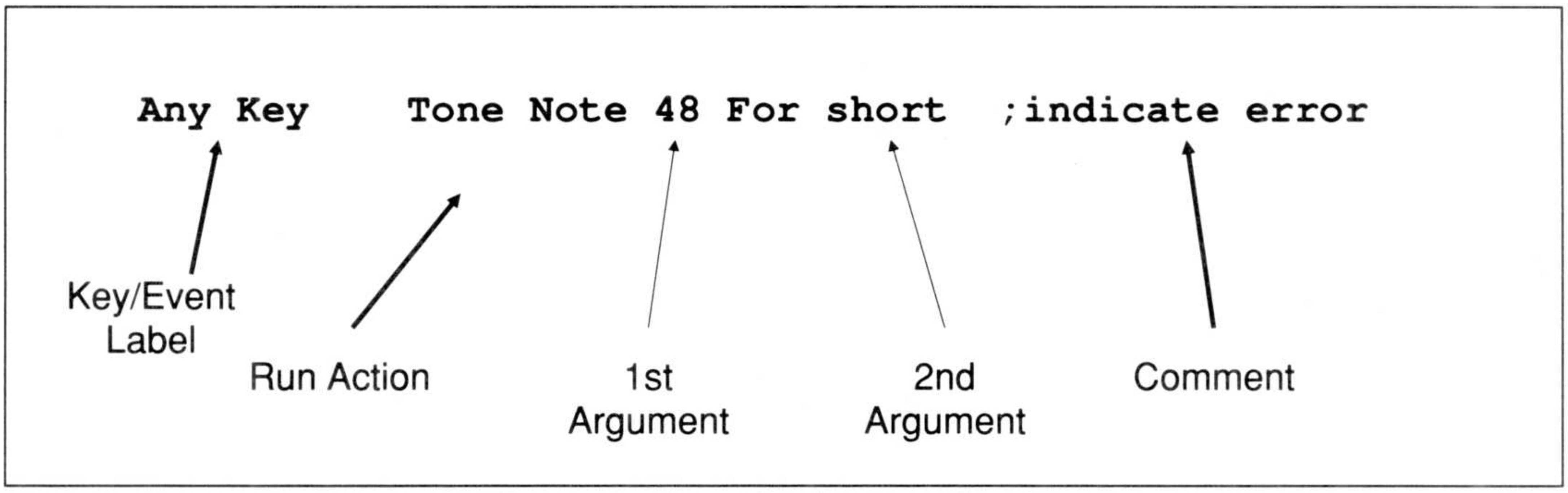


Figure 1. A typical run action line

The "Any Key" on the left is the Key/Event label, the "Tone Note 48 For short" is the run action, and ";indicate error" is the comment.

To create this run action line, you first execute the Insert command. (Note that all of these commands are on the Run menu, so we will not mention the full list of commands to execute when listing the commands.) This brings up a command window with the Key/Event list — the list of all Key/Events. You select the "Any Key" Key/Event in a normal DEMO II-like manner. It is the last item in the list, so you can press End and then Enter. This inserts a new run action line below the line with the highlight (or it creates the first line if none is present) and displays:

```
Any Key
```

A new menu appears at the bottom of the command window and the title of the command window changes to RUN: ACTION MENU. The menu lets you chose a run action.

Some of the more common run actions, such as View and Quit, are listed in this menu. The rest of the run actions are grouped and can be selected on submenus. The run action we want is on the Tone submenu.

The submenus are:

Programming	Run actions for testing values, looping, and transferring
Tone	Run actions for producing sounds
Other-Slide	Run actions for viewing other slides
Key/Event	Run actions to continue executing actions on other action lists
String	Run actions to manipulate string values and variables
Miscellaneous	Run actions very specific to DEMO II
File	Run actions to control file and printer I/O

By selecting the "Note" command on the "Tone" submenu, the run action line changes to:

```
Any Key   Tone Note VALUE? For 1/18-SECS?
```

This is the "form" for the Tone Note run action. The form is the standard text of the command (here, "Tone Note" and "For") along with information about the **arguments** to the run action. Arguments are the parameters to the run actions that you provide, such as numeric values, variables, strings, and slide references. When these arguments are not set, an indication of the type of argument needed is displayed, usually in uppercase letters ending with a "?". In this case, there are two arguments to the run action: the first is a value and the second is an indication of how many 1/18 seconds. The second argument is also a value, but the "1/18-SECS" provides you with reference information about the way in which the argument is interpreted.

All run actions have between zero and three arguments, with descriptive text around the arguments to make the run action readable and understandable.

After chosing this run action, the command window changes back to the "RUN MENU", and the menu changes to one for entering or editing an argument. The prompt at the bottom says "Setting first argument (1)".

To set the note value to 48, you just type "48".

There are other ways of choosing a value that gets you variable names, key or Key/Event values, or references to strings. After setting the value, you press the Tab key to move to the next argument. Here you can type the name of the variable "short" or press "?" to display a list of all variables, and then choose it from the list. Since there are only two arguments to this run action, if you end the name "short" with a Tab, you will be editing the comment field. All run action lines can have comments. They are used for descriptive purposes. Here we just type the comment "indicate error" and then press Enter to get:

```
Any Key   Tone Note 48 for short     ;indicate error
```

The keys we pressed to get this run action were:

`I`	- *Select the Key/Event Label*
`End`	
`Enter`	
`T N`	- *Select the run action*
`4 8`	- *Give the first argument*
`Tab`	- *Move to the second argument*
`S h o r t`	- *Give the second argument*
`Tab`	- *Set the comment*
`i n d i c a t e   e r r o r`	
`Enter`	

This is much shorter than that needed in most editors, and (because of the prompts) easier to produce correctly for the novice.

To create a new run action line with no Key/Event label, you can use the Line command. It is the first command on the Run menu, so that you can get a new run action line by just pressing Enter. Like the Insert command, it then switches to choosing the run action for the line.

For example, to add a "View Slide >PREVIOUS<" run action without a Key/Event label, you would press:

`Enter`	*- Insert a new line*
`V`	*- Select the View run action*
`P`	*- Select the Previous slide*

To change the run action on a line, you use the Action command.

To edit an argument, you can just type the number corresponding to the argument (1, 2, or 3), executing the 1st, 2nd or 3rd commands, or use the Tab and/or Shift-Tab (F4) keys. The Tab keys change which part you are editing, just like when you are originally setting the arguments. This use of the Tab keys is unique to the Run command. You set the comment by executing the ;Comment command or using the Tab/Shift-Tab keys.

Run actions are often displayed in a manner that indents loops and the parts of mulitiple line run actions such as "If". This indenting can be automatic or manual and is ignored by the running mechanism. Its only purpose is to make the run actions more readable. Lines that DEMO II thinks should be indented or moved left are moved by the number of columns listed in the Global Run Action Indent setting on the Global menu when they are created. You can manually indent a run action line by pressing the Ins key, and move it left by one character by pressing the Del key. This

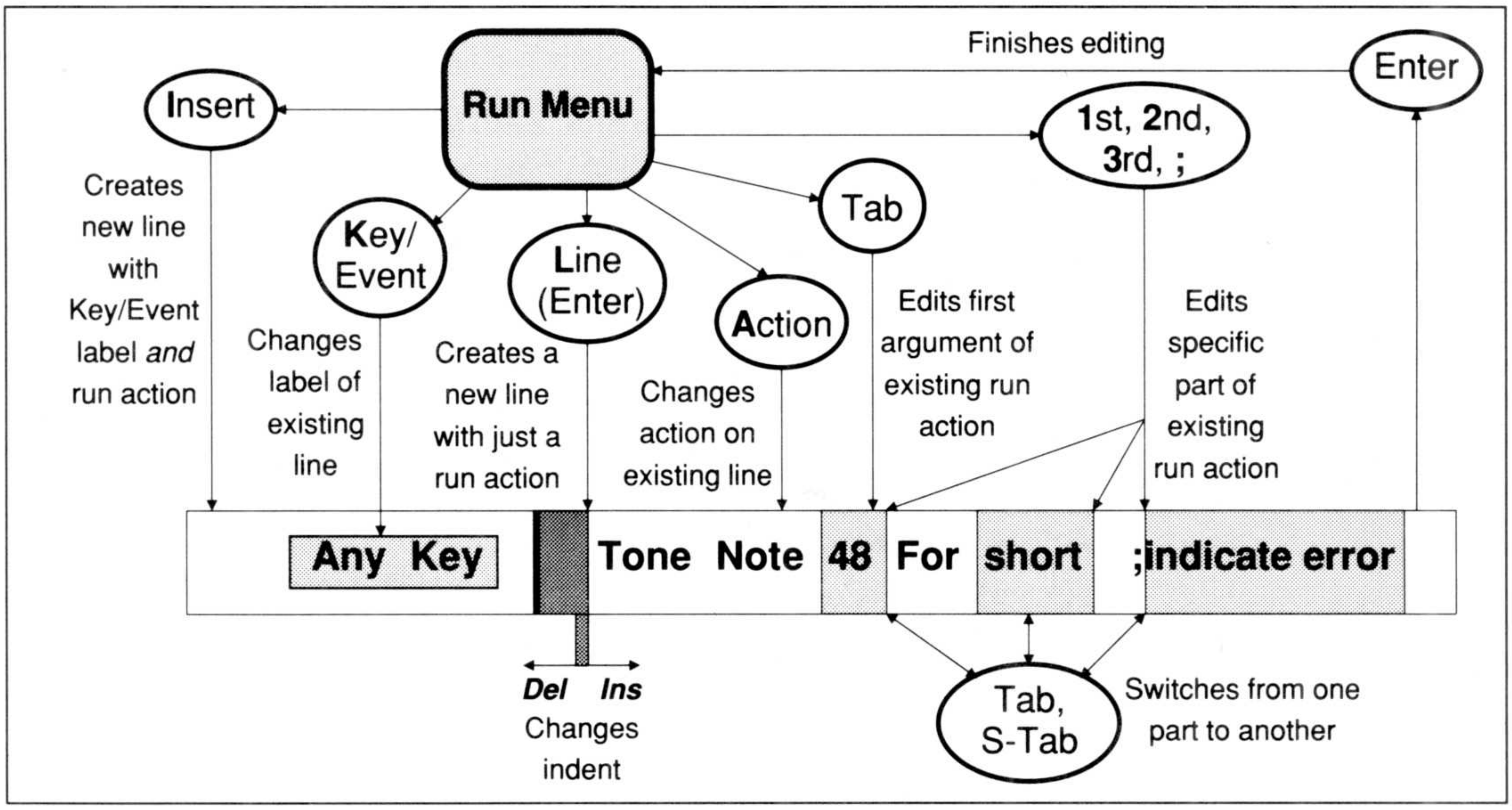

Finishes editing
Enter
Insert
Run Menu
1st, 2nd, 3rd, ;
Creates new line with Key/Event label *and* run action
Key/ Event
Line (Enter)
Tab
Action
Changes label of existing line
Creates a new line with just a run action
Changes action on existing line
Edits first argument of existing run action
Edits specific part of existing run action
Any Key
Tone Note 48 For short ;indicate error
Del Ins
Changes indent
Tab, S-Tab
Switches from one part to another

Figure 2. Editing run actions

use of the Ins and Del keys is unique to the Run command and does not affect the Insert/Overwrite setting; nor does it delete any text or run actions.

You can change, add, or delete Key/Event labels with the Key/Event command.

Figure 2 summarizes the information about creating and editing run actions.

Note

String constant run action arguments only display up to eight characters. If the string constant has more than eight characters, just the first seven are displayed, followed by a ">" character. You can scroll though the entire value when you edit the argument.

Copy/Paste and Other Commands

Individual or groups of run action lines can be deleted or moved with the Delete and Move commands, respectively. The Delete and Copy commands put a copy of the highlighted run action line into the Action Copy/Paste buffer. If there is a group of lines, then copies of all of the group's lines go into the buffer. These copies of run action lines can be pasted into any run action list by using the Paste command. They can also be saved to a file and reloaded later with the I/O Write-Code and I/O Code-Read commands, respectively. All information about the copied lines is preserved, if possible, including references to variables and values, along with their definitions. References to slides (in the View, Call and Use run actions) are only preserved until any save or load is done (including Write-Code) or any slide is deleted. In those cases, the value is undefined, leaving just a "SLIDE-REF?" in the form.

The Type and Wait commands let you set the Run Type and Run Wait values. The Run Wait is zero by default, and the Run Type is the value on the Global Default Run Type setting on the Global menu.

*The Run and *Debug Commands*

The RUN and *Debug commands are used to start the slide show running. RUN starts it running normally, while *Debug starts it running immediately in Debug Run mode.

You can stop running at any point by pressing the Ctrl-Break key. This will bring up a prompt asking you whether you want to quit running or continue running but in Debug mode.

Debug Run mode brings up the Debug menu/message whenever a run action is executed or a key pressing Key/Event is not matched. The menu/message gives you information about the state of running; it then lets you continue with Debug running (also called "single step") by pressing Enter, switch to normal continuous running with "C", examine the list of variables and their values with "V", or stop running with Esc.

Run Line

Inserts a new blank line below the highlighted one (or creates the first line if there is none) and then executes the Run Action command. It is the first command on the Run menu, so you can get a new run action line by just pressing Enter.

Run Action

Sets the run action on the highlighted line. It brings up the Run Action menu. The Run Action menu lets you choose from among all of the available run actions. One of the commands on this menu is the Run Action ?List command, which lists all of the run actions in menu order. Just press Enter to select an item. All of the other commands on the Run Action menu are described in the "Run Actions" section of this manual.

Run Action ?List

Lists the run actions. Press Enter to select an item.

Run 1st

These three commands edit the indicated argument to the highlighted run action. To select one of these commands, you type "1", "2", or "3", respectively.

Run 2nd

Run 3rd

Run ;Comment

Edits the comment part of the highlighted run action line when you press the ";" key. If you then start typing, the current comment is replaced. If you start with an editing key, you can edit the current comment. A comment with no characters causes the entire comment to be removed, including the ";".

Run Wait

Sets the slide's Run Wait setting. This value is used when viewing a slide and usually specifies a time in 1/18 seconds. See the "How Running Works" section for a description of Run Types and how they use the Run Wait setting. When a blank slide is created, this setting has a value of zero by default.

Run Type

Sets the slide's Run Type setting. This value is used when viewing a slide to determine how to process keys and when waiting should be done. See the "How Running Works" section for a description of Run Types. When a blank slide is created, this setting gets the Global Default Run Type then in effect.

Run Tab/Ins/Del

Has no effect. It is provided so that you have a description of what the Tab, Ins and Del keys do. Not all people read manuals carefully, and these are useful functions to know about.

Run
#

Moves the highlight to the specified line number. For your convenience, the lines are numbered on the left. These are not for selecting, like the fixed 1-9/A-F on some other menus, but rather, for identifying all of the lines. Lines are renumbered after Inserting, Deleting, Moving, etc.

Run
Vars

Calls up the Variables menu. This is provided for your convenience.

Run
Insert

Inserts a new line following the line with the highlight (or creates the first line if there is no highlight), then executes the Run Key/Event command followed by the Run Action command. This command inserts a new run action line with a Key/Event Label.

Run
Key/Event

Sets the Key/Event label for the highlighted line. This command invokes the Key/Event list command window, which displays all of the Key/Event names.

The Key/Event menu commands follow.

Run
Key/Event
Select

Selects the Key/Event highlighted. You can also type the character on the left of the desired line, with this menu command highlighted, to select a Key/Event. The last Key/Event selected in this manner is highlighted the next time you execute this command.

Run
Key/Event
#

Prompts for a Key/Event number. That value is used as the selected Key/Event. This can be used in macros, so you do not have to know the position of the highlight.

Run
Key/Event
Key

Prompts for you to press a key. The most specific Key/Event that matches that key will be selected. This is a very useful way of selecting most Key/Events that refer to key presses, except those that are general, such as "Aa" and "Any Punc".

Run
Key/Event
Tag

Sets a Tag Key/Event label, prompting for a tag name of up to eight characters. See the description of Tags in the "How Running Works" section.

Run
Key/Event
Reset

Clears the Key/Event label so that the run action line has no Key/Event label.

Run
Delete

Copies the currently highlighted line (or all of the lines in the group, if there is a group) to the Action Copy/Paste buffer, erasing any lines that were already there. Then it deletes the lines from the run action list. If there is a group, it always copies the lines, even if you cancel the command when it prompts for confirmation.

Run
Move

Repositions the highlighted line. If there is a group, the entire group of lines is repositioned. Note that any indenting is not redone to conform to the program's structure. You may want to use the Ins and Del keys to change the indenting on lines after a move.

Run Group

Toggles the start of a group of lines. Groups can be moved, deleted, and copied.

Run Copy

Copies the currently highlighted line (or all of the lines in the group, if there is a group) to the Action Copy/Paste buffer, erasing any lines that were already there. Turns off the grouping.

Run Paste

Inserts the current contents of the Action Copy/Paste buffer below the highlight (or creates the first lines, if there are none). If more than one line is pasted, it is marked as a group so that you can delete it if you change your mind, move it, etc. You can turn off the grouping by pressing "G" to execute the Run Group command.

Note that any indenting is not redone to conform to the program's structure. You may want to use the Ins and Del keys to change the indenting on lines after a paste.

Run action lines are pasted just as they were when they were copied/deleted. Direct slide references are changed to "SLIDE-REF?" if there was a slide deleted. Direct slide references are also changed if there was any saving or loading of slides or the Action Copy/Paste buffer since the lines were put in the buffer. If the run actions refer to variables that are no longer defined, those variables will be defined as they were when the lines were put in the buffer.

Run OK

Returns to editing.

Run *Debug

Starts running in Debug mode. Puts up the Debug menu/message before run actions are executed or when a key pressing Key/Event is not matched. Debug mode is useful to find out why your slide show is not acting as you expect. Press "*" to select this command.

Run RUN

Starts running the slide show normally.

Run F1, F2 S-F1, S-F2

Switches the current slide to another slide and displays its run action list.

The Macro Commands

DEMO II has learn-type keyboard macros built in. They allow you to perform a series of operations and have your keystrokes recorded. You can then play those keystrokes back by pressing a single key.

There are 36 separate macros, Alt-0 through Alt-9, and Alt-A through Alt-Z. Macros Alt-0 through Alt-9 are stored with the slides when they are saved. These should be those macros that are specific to a particular series of slides. Macros Alt-A through Alt-Z are stored in a system file "_ALT_A_Z.SG1" in the DEMO II System Directory. They are reloaded whenever a file is loaded or memory is cleared, which occurs at start up and after a Global ClearAll command.

To invoke a macro, you type its name (0-9, A-Z) while holding down the Alt key. While the macro is being run, the letter R, followed by the macro name, will appear in the upper-right corner of the screen.

The Macro command puts up a command window listing all of the 36 macros. Each item has the name of the macro (0-9, A-Z), a descriptive name provided by the user, and the current length of the macro in keystrokes.

The following is a list of the Macro commands.

Macro OK

Returns to editing.

Macro Run

Starts running (playing back) the highlighted macro.

Macro Save

Saves the current definitions of macros A-Z in the DEMO II system file "_ALT_A_Z.SG1". Macros A-Z are not saved by the I/O Save command, but macros 0-9 are.

Macro Learn

Starts recording keystrokes for the highlighted macro, erasing any previously stored keystrokes. The letter L and the name of the macro being recorded appear in the upper-right corner of the screen. DEMO II starts recording in that macro definition all keystrokes that you type. The keystrokes will be executed, as well. To stop learning a macro, press Alt-name, where "name" is the macro being learned (e.g., Alt-0), or press Ctrl-Break. You can invoke macros within macros, but illegal combinations, such as invoking a macro that is running, are caught; you will then get an error message. You cannot Save or Load slides in the middle of a macro. (Those commands, as well as Run Run and Run *Debug, turn off any running or learning of a macro.)

The Shift-F6 key prompts for a letter or number by displaying "L?" in the upper-right corner of the screen. It then also "learns" the macro associated with that character. Shift-F6 can be used instead of the Macro Learn command, and it can be used in more places. For example, you can invoke Shift-F6 in the middle of another command, while Macro Learn can be invoked only from Edit mode. As with the Macro Learn command, you stop learning a macro by invoking it or pressing Ctrl-Break.

Macro Extend

Continues recording keystrokes for the highlighted macro, appending them to any previous keystrokes. See Macro Learn.

Macro View

Displays the contents of the highlighted macro in a command window. You can use the list perusal keys to see the stored keystrokes.

The Macro View commands are:

Macro View OK

Returns to the Macro command window.

Macro View Insert

Adds a keystroke to the macro definition. Prompts you to press a key. The key you press is added to the listed macro below the highlight. (Or it becomes the first key, if there are none.)

Macro View Delete

Erases the highlighted keystroke from the macro definition.

Macro Name

Edits the descriptive name of the highlighted macro.

Macro Delete

Erases the definition of the highlighted macro.

The Global Commands

The Global command puts up a command window which displays several Global Settings. You may use the list perusal keys to move the highlight to many of the settings, and then change them with the Edit command. There is also a command to clear memory (erase all slides, etc.), and others to call up the Global Run menu, the Variables menu, the Global Overlays menu, and the Global Block Names.

The following is a list of the Global Settings:

No Marked Block If no block is marked, then the screen position (row, column) and contents of the character under the cursor are displayed. The hexadecimal value of both the text character and attribute are displayed. Characters on only the current slide's ">THIS SLIDE<" are shown — not on any overlays. This is useful for differentiating between transparent characters (00) and blanks (20), as well as looking at attributes of Captured slides. This value may not be changed by the Global Edit command.

Marked Block The size of the marked block, if present, in rows and columns. This is useful for finding out the size of something on the screen. Just mark it as a block, and see the size with the Global command. This value cannot be changed by the Global Edit command.

Memory The amount of memory used and the amount available, both in KBytes (1024 bytes) and chunks (16 bytes each). The internal unit of chunks is provided so that you can get a feel for how DEMO II uses memory when you are trying to minimize the size of a set of slides. Many operations cannot be done when there is less than about 10-20KBytes of memory left. This value cannot be changed by the Global Edit command.

Current Filename The name of the file last loaded, and used as a default. This value cannot be changed by the Global Edit command.

System Files Directory The name of the directory where DEMO II system files are stored. It is set only by the DIR option to the DEMO2 DOS command. If it is not set, the DOS Current Directory is used. The DEMO II System Directory contains the _ALT_A_Z.SG1 (Macros A-Z), _P??????.SG2 (Printer Mappings), and _INIT.DBD (InitFile, loaded after memory is cleared) files.

Display Adapter The type of display adapter operation being used. It is only set automatically or by option to the DEMO2 DOS command. For a list of options to the DEMO2 DOS command, see the "DEMO2 and RDEMO2 Programs" section of this manual.

Changes Made If DEMO II thinks that a change may have been made since the slides were loaded, this is "Yes"; otherwise it is "No". A warning is posted if you try to execute the Quit or I/O Load commands when this value is "Yes". This value is sometimes set even if you just display a menu that could make changes. The Global Edit command will toggle this value.

Typing Attribute

The attribute that is placed on a character when you are typing. If zero (the default), then the screen position's attribute will remain unchanged. Invoking the Global Edit command when this item is highlighted will bring up a command window, similar to that of the Block Attribute command, to be used to define/select an appropriate display attribute. See the Block Attribute command.

The Typing Attribute can be set with the Grey + and Grey - keys, and is only applied if the CAB setting is Attribute or Both. See the "Keys" and the "Block Commands" sections.

Grey +/- Cycles

Determines whether the Grey + and Grey - keys cycle through the attributes in the Block Attribute list or all 256 possible attributes. The Global Edit command switches between these two values. The default value is the Block Attribute list.

Background Attribute

The attribute of the background of all slides. This is the attribute that is shown when all attributes after it are transparent. The Global Edit command works similar to the Global Typing Attribute value.

This value can also be set while running by the Set Builtin(6) run action.

Background Character

The character that is displayed if no characters are overlaid after it. This character is usually the Space character (20 hexadecimal), but for debugging of overlays and special effects you may want some other value. Invoking the Global Edit command when this item is highlighted will bring up a command window with all of the possible 256 characters. Select one.

This value can also be set while running by the Set Builtin(7) run action.

Menu Background Attribute

The attributes of the menu background can be set. The Global Edit command works similar to the Global Typing Attribute value.

Menu Highlight Attribute

The attributes of the menu highlight can be set. The Global Edit command works similar to the Global Typing Attribute value.

Default Video Bits

The initial value used for a slide's Video Bits setting. See the "Slides Commands" subsection, "Slide Options," for more information about the Video Bits setting. The default value is 70 hexadecimal. The Global Edit command prompts for a new value. Precede the value with "0x" or "x" for hexadecimal; otherwise, decimal values will be used.

Default Run Type

The initial value used for a slide's Run Type setting. See the "How Running Works" section for a description of the Run Type. The Global Edit command prompts for a new value.

Run Action Indent Incr

The number of character positions to indent new run actions for each level within a loop, conditional, etc. See the "Run Commands" section. The Global Edit command prompts for a new value.

Overlays Shown If "No", overlays are not shown — just the slides' ">THIS SLIDE<'s" themselves. If "Yes", overlays are shown, too. Toggled by invoking the Global Edit command when this item is highlighted. This is useful when you are checking out slides and their overlays. See the "Overlays Commands" section for more information.

Timeout During Run The number of seconds to wait, while running, for the user to press a key before the Timeout Key/Event is signaled. Invoking the Global Edit command while this item is highlighted will prompt you for a value from 0 (none) through 999 seconds (about 16 minutes). The default is 0 (Timeout not signaled). See the "How Running Works" section for more information.

This value can also be set while running by the Set Builtin(8) run action.

The following is a list of the Global commands:

Global Edit Changes the value of the highlighted item. See the list of Global settings listed above for descriptions of its effect on each setting.

Global OK Returns to editing, remembering the item highlighted so it will be highlighted again on the next Global command.

Global ClearAll Erases all slides, settings, etc., and starts with a clear memory and default settings. The _INIT.DBD file is loaded, if it exists, to set user-defined default values for most settings, etc. Then the Macros A-Z file is loaded (see Macros Save).

Global Run Calls up the Global Run menu, displaying the Global Run Action list. This works in a manner similar to the normal slide Run command. See the "Run Commands" section.

Global Variables Calls up the Variables menu. See the "Variables Menu" section.

Global .Overlays Calls up the Global Overlays menu, displaying the Global Overlays list. See the "Overlays Commands" description. Note that you must press a "." to invoke this command.

Global Names Calls up the Global Block Names menu, displaying the Global Block Names list. See the "Block Commands" section for a discussion of Block Names.

The I/O Commands

The I/O commands are used to save and load what you are working on, print the screen images, save screen images to a file, load portions of another slide show, retrieve images captured from other programs, import text files, and save and load the Action Copy/Paste buffer.

The I/O commands are listed below.

I/O Save

Saves all slides, attributes, overlay definitions, run action lists, named save areas, current Delete/Paste buffer, cursor location, current slide number, tab settings, etc., in a ".DBD" save file. Puts up a command window with the name of the current save file (if set) followed by a directory listing of ".DBD" save files (which may include the current save file, if it already exists). The name of the current default directory that is being listed is shown in the window title, which is set by the I/O Save Dir command. You can move the highlight with the list perusal keys, and then select a file to be used for the Save operation. If the file already exists, you will be asked to confirm that you indeed want to replace its contents. No ".BAK" file is produced.

Note

The current slide at the time you do an I/O Save is important. That slide will become the current slide when the ".DBD" file is reloaded. If the ".DBD" file is loaded by RDEMO2, running will start with that slide as the first slide in the slide show.

The I/O Save commands are:

I/O Save Select

Uses the highlighted file name.

I/O Save New

Prompts for the name of a new file to use in the default directory shown. The file may or may not exist. Do not type an extension — the correct one will be assumed. The name given becomes the new "current" file and is selected.

I/O Save Full

The same as I/O Save New, except you can type a full pathname (with no extension).

I/O Save Dir

Prompts for a pathname of a directory. That directory becomes the new "default" directory and is listed. If the value is all blank, then the DOS Current Directory is assumed. A drive specifier, such as "A:", can be used.

I/O Load

Clears everything, then reloads what was saved in a ".DBD" file by the I/O Save command. DEMO II will then be just as it was when you saved it, with the same slide on the screen, the same cursor position, etc.

This command operates similarly to the I/O Save command, listing files, etc. If changes have been made since the last file was loaded or memory cleared, then you will be prompted for a confirmation. After the file is loaded, the Macros A-Z file is reloaded, if present.

Read errors (such as diskette read errors, but not "file not found" errors) and other errors that occur while reading files, such as out of memory, are often fatal to the operation of the program. They cause the DEMO II program to display an error message and then quit, since the partial read may leave DEMO II in an inconsistent state. These errors should be uncommon.

I/O Print Controls the printing of slide images, information about slides, etc. Puts up a command window with the following Printer Settings:

Output To The filename of the current print file, or the word "[PRINTER]" if output is to go to the PRN device (DOS's standard print device). Editing this value with the I/O Print Edit command brings up a command window similar to the I/O Save command, with the additional command of Printer to set the output to PRN (normally the printer — see your DOS manual) instead of a file.

The assumed extension for files is ".TXT". If you start printing to an existing file, you will be given a choice between replacing its contents and appending to the end.

This value defaults to "[PRINTER]".

Character Mapping The mapping name, or "No Mapping", "C Language" or "Pascal Language".

The default is "No Mapping", but you may want to change that to a printer mapping as explained in the note, below.

If the value is a name, then associated with that name is a file that contains the list of characters to output in place of what would normally be printed.

If "No Mapping", then characters are output with no change.

If "C Language", then characters with a value below space (decimal 32), above ~ (decimal 126), and the characters \ and " are output as octal constants in a form acceptable to the C computer language. For example, the four character "\032" would be output for the right-arrow symbol instead of the single Ctrl-Z character it represents. All other characters are output as themselves. This is as an aid to programmers who want to put screen images with the special characters into their programs.

If "Pascal Language", then all characters are output as #$xx, where xx is the hexadecimal representation of the character, such as #$1A. This is useful to users of some versions of the Pascal programming language, and, with a bit of editing, other languages, too. Note that the lines produced may be longer than some editors or compilers can handle (320 or more characters wide). You may have to edit them appropriately or output small chunks by printing from a marked block.

The "C Language" and "Pascal Language" settings are usually used in connection with printing the screen images to a file, rather than to the printer. You use the "Output To" setting to set a file to receive the data.

Printer Mappings

Often you want to output different characters than those that appear on the screen. The most common case is when the printer does not support that character code. For example, the right-arrow symbol has a value of decimal 26. That is the Ctrl-Z code, and when it is sent to most printer drivers, it indicates the end of a file, and stops printing prematurely. You would want to send a different code in place of the 26 to most printers. The Printer Mapping facility lets you do this.

A printer mapping lists all of the 256 codes that can be output to the printer, and it shows what to output instead. Most codes output as themselves (an "A" for an "A", for example). For others, you can specify up to two replacement codes. If there are two codes, a backspace code (7) is output between them.

Editing the Character Mapping Setting

If the Character Mapping setting is edited with the I/O Print Edit command, a command window similar to the I/O Save command is displayed. The names listed are those of printer mapping files in the DEMO II System Directory. Printer mappings have file names in the form "_P??????.SG2", where ?????? is the printer mapping name. The commands in the Printer Mapping command window are: *Select* and *New*, which are like those in the I/O Save command; *C* and *Pascal*, to use those builtin mappings; *Don't*, to specify no mapping; and *Mapping*, to define or view a user-defined printer mapping.

If *Mapping* is selected, then a command window with a list of all 256 character codes is displayed. Next to each character code (shown as a character, a decimal value, and a hexadecimal value) is the character to output in its place when printing, and a second character, if present, to be overprinted (by outputting a backspace) over the first. The commands there are: *First*, to edit the first character output; *Second*, to edit the second; *Normal*, to make the highlighted character output as itself; *Done*, to save the changes in the file; and *Cancel*, not to save them. The *First* and *Second* commands use the arrow keys, PgUp/Dn, Home, and End to change the value, or you can just type the character you want. Press Enter when you have the correct character. Note that many printers ignore the zero (00) character, so if you output it (such as by having 00 as your Global Background Character), the printout may appear strangely compressed.

Note

If your printer "hangs" while printing slides with special characters and you are not using a printer mapping, you should switch to using one.

To help you set up a printer mapping, two common ones are provided on the Demo II Program Diskette — PRINT and ASCII (_PPRINT.SG2 and _PASCII.SG2, respectively). The PRINT printer mapping is for printers that know about most of the codes, such as the common IBM and Epson printers. The ASCII printer mapping should work with just about all printers, and it only uses the normal 96 printing ASCII characters. For both printer mappings, the unprintable codes are replaced by printable ones such as "-" overprinted by ">" for right arrow, and "?" for many others.

Output: Text

Whether or not to output the text of slides. Editing this value cycles through "No"; "Yes, with CR/LF", which outputs a "Carriage Return, Line Feed" combination after each line; and "Yes, without CR/LF", which outputs one line after another for reading by specially written programs.

The default value is "Yes, with CR/LF".

Output: Attributes

Editing this value cycles through "None"; "Interspersed", which outputs alternating text/attribute bytes like the PC hardware requires; and "Separate", which outputs first all the text and then all the attributes.

The default value is "None".

Outputting attributes is useful for input to custom-written programs, when using the Language mappings, and when you want to see what the attributes are. You may want to do a separate printing of the attributes with a special mapping that gives them understandable symbols (such as "." for normal (07), "I" for inverse (70), and "B" for blinking).

Output: Names/ Numbers

Outputs the slide name and number, as shown on the Slides list, after each slide is printed. Editing toggles between "Yes" and "No".

The default value is "Yes".

Output: Overlay Lists

Outputs the definitions of all overlays for each slide printed as well as the Global Overlay list. The position offsets and other settings of the overlays are also printed. Editing this value toggles between "Yes" and "No".

The default value is "Yes".

Output: Run Info

If "Yes", outputs all run action lists for each slide, outputs the slide's Run Type, Run Wait, Switch Type, and Switch Speed values, and outputs the Global Run Action list after all slides are printed. Editing toggles between "Yes" and "No".

The default value is "Yes".

Output: Variables List

If "Yes", outputs the Variables list after all slides are printed. Editing toggles between "Yes" and "No".

The default value is "Yes".

Output: Block Names

If "Yes", outputs the Block Names list for each slide, and outputs the Global Block Names list after all slides are printed. Editing toggles between "Yes" and "No".

The Block Name, the position, and the comment (if present) are output.

The default value is "Yes".

The Block Names can be used to provide comments about portions of a slide. You can write specially designed programs to read this information from a ".TXT" file produced by the I/O Print command. The information could be interpreted, for example, as field definitions for a forms package or linkage data for a help system.

Output: Slides That Reference Slide

If "Yes", outputs a list of all slides that reference the printing slide in a Key/Event label, a View action, a Use action, a Slide Call action, or a Slide Overlay. This includes references to >NEXT< and >PREVIOUS< that actually refer to the slide. The words "Run" and "Overlay" appear next to each line, indicating what type of reference was made. Editing toggles between "Yes" and "No".

The default value is "Yes".

Page Break After Slide

Whether or not to output a Form Feed character after each slide and its information are output. Editing toggles between "Yes" and "No".

The default value is "No".

Blank Lines After Slide

How may blank lines to output after each slide and its information (follows the Page Break After Slide's Form Feed). If set to zero, you can print slides up against each other, simulating one big, tall screen. You can use this to let DEMO II help you mock up printed reports or print forms. Editing this prompts you for a number.

The default value is 1.

Trim Trailing Blanks

If "Yes", suppresses blanks (hexadecimal 20) and zero (00) characters on the right of each line. This may speed printing on some printers. Editing toggles between "Yes" and "No".

The default value is "Yes".

When outputting to a program that expects exactly 2000 character positions with the Text setting "Yes, without CR/LF", you will probably need this set to "No".

Block Marked

This setting appears if there is a block marked.

If a block is marked, then only the contents of the marked block are output from each slide. This can be useful when putting a message into a program by printing to a file, shortening the output when you mainly want to look at the other information, etc.

This value may not be edited.

How Many Slides to Output

The number of slides to output, starting with the one currently displayed. Editing this value prompts for a number. The letter "A" causes all slides to be printed. The letter "F" prints all slides with the Print Flag set.

The default value is 1.

The I/O Print commands are:

I/O Print Edit

Modifies the highlighted values. See the description with each I/O Print setting.

I/O Print OK

Returns to edit mode.

I/O Print Start

Starts outputting to the printer or a file, as shown in the settings. You can cancel printing, once it has started, by pressing Ctrl-Break.

I/O Add

Loads slides from a ".DBD" file and adds them to the current set of slides. You specify the name of a file from which to load the slides in a manner similar to the I/O Save command. After selecting a file, DEMO II prompts you for the numbers of the first and last slides to reload. The slides themselves are inserted as new slides after the current slide.

This command loads the images of the indicated slides as well as their Run Type, Run Wait, Switch Type, Switch Speed, and Palette settings, and their run action lists and overlay lists. If references are made to variables that are not defined, the definition and value are copied, too. If references are made to slides that are not among those being added, the references are reset to "not set".

I/O Retrieve

Inserts screen images into the current slide show. The images can come from the CAPTURE program, from text files, or are references to ".PCX" files. The images are stored in new slides inserted after the current slide.

The I/O Retrieve command prompts you for the type of retrieving to be done.

"C" means to retrieve from the CAPTURE program. See the "CAPTURE Program" section of this manual.

"T" means to read text images from a file. You will be prompted for a file name *with its extension*. The file will be read into successive slides. The attributes will all be transparent. The characters in the file will be read onto the slides with CR resetting to the first column; LF going to the first column on the next row; FF (^L) going to a new slide; and going past row 25, going to a new slide. All other characters come in as themselves.

"P" means to create a bitmapped image slide that references a ".PCX" file. You will be prompted for the name of the file. The name may be a string variable or a string constant. You start a string constant with the " character and end with Enter. If you use a variable, its current value will be used each time the slide is displayed.

I/O Code-Read

Reads run actions saved with the I/O Write-Code command. The run actions are read into the Actions Copy/Paste buffer for pasting into run action lists by using the Run Paste command. I/O Code-Read puts up a menu similar to the I/O Save command. See the "Run Commands" section for a description of the Actions Copy/Paste buffer.

I/O Write-Code

Writes the contents of the Actions Copy/Paste buffer to a file, and puts up a menu similar to the I/O Save command. See the "Run Commands" section for a description of the Actions Copy/Paste buffer. The files are given the extension ".DBC".

The Variables Menu

The Variables menu lets you view and manipulate the Variables list. You can select variables, set their values, create new variables, and delete them. The Variables menu is accessed from a variety of places, including the Global menu, the Run menu, and type-in prompts that can accept a variable name.

Variables can be used in DEMO II to control many settings, such as those associated with overlays, and the Run Type and Run Wait values. They can also be used to hold viewer input when simulating a database forms system. You can test and modify the values of variables while the slide show is running by using run actions.

The Variables list displays all of the defined variables, listing their names, their current values, their positions in the list, and their Passed On setting.

Types of Variables

Variables can be of two types: numeric or string.

Numeric variables have integer values from -32768 through 32767. In computerese, they hold 16-bit signed values.

String variables have two parts. The first part is space for 80 characters of data. In computerese, that's an array of 80 bytes. Each character can hold a value from 0 to 255, usually representing one of the 256 characters in the PC's character set. The second part of the string is a number from 0 to 80, which specifies the number of chararacters that have valid data. This is called the length of the string. Each time you assign a value to a string variable, the length is set.

When you create a variable, you select its type. You can change its type by using the Variables Value command to assign a new value. Only this type of assignment can change the type of a variable. Assignments made while running, such as with the "=" run action, do not change the variable's type. This is done so that the amount of memory used during running remains constant.

You can determine a variable's type from the way in which its value is displayed in the Variables list. If it begins with a ", then it is a string variable. If it is a number (possibly followed by the character it represents), it is a numeric variable. String variables show the length in parenthesis after the value.

The "Passed On" Setting

Each variable has its name and its **Passed On** setting associated with it. The name may be from 1 to 8 characters long, and should be unique. If you change the name, all references to that variable will automatically have the name changed, too.

The Passed On setting is used to preserve variable values when you switch from one file to another while running. You switch with the File and View Slide In File run actions. The current value of a variable is kept if variables with the identical name and type exist in both the file executing the run action and the file being loaded, and the

Passed On setting for both is set "on". This result ignores the value that was saved in the file being loaded. You would use this feature, for example, to pass an error count from one file to another while running. You could also use this feature to pass on the name of the file to load upon completion, and perhaps to pass on the slide number to view in that file. A maximum of 43 variables may be "Passed On" to the next file.

Miscellaneous

When a string value is used in a run action or in a setting, and a numeric value is needed, the first character of the string is treated as a number. For example, the letter "A" would be 65, its ASCII value. If the string has a length of zero, it is treated as if no value were set, often defaulting to zero.

When a numeric value is used in a run action or in a setting, and a string value is needed, the number is converted to a string with a single character. If the value is between 0 and 255, it is used directly. For example, 65 would become the string "A". Otherwise, a zero character value is used. If no value is set, it is treated as a zero-length string.

The character positions in string values are numbered from 1 to 80, when referenced individually by run actions.

The Variables Commands

Vars OK

Returns to the previous menu or editing, depending upon the situation.

Vars Insert

Inserts a new variable definition below the highlight, or creates the first one if none existed. You will be prompted for the type of the new variable: N for numeric, S for string. The new variable will be given the name ">NONAME<" and a default value. You should give it a name with the Variables Name command.

This is one way of creating a new variable. The more common way is to refer to the variable by name where it is needed, such as in a run action. If a variable with that name is not defined, you will be prompted to see if you want to create one. Variables created in this way are always put at the top of the variables list. You may want to reorder the list with the Variables Move command, if you prefer to see them in a different order.

Vars Delete

Erases the highlighted variable and its value. If there is a group, the entire group of variables is deleted. All references to the variable are changed so that none are set.

Vars Move

Repositions the highlighted variable in the list. If there is a group, the entire group of variables is moved.

Vars Group

Toggles the group start.

Vars #

Moves the highlight to the specified position in the variables list.

Vars Locate

Moves the highlight downward to the next variable that starts with the given characters.

Vars Name

Edits the variable's name. Changes all references to the variable to the new name.

Vars Value

Changes the variable's value. The type of value you enter determines the type of the variable. If you enter a string value (starting with a "), the variable will be switched to a string value if it was a numeric value. If you enter a numeric value (starting with a number or "-"), the variable will be switched to a numeric value if it was a string value.

Vars Passed-On

The highlighted variable's Passed On setting will be toggled. The letters "PO" appear to the right of the variable's position number if set "on".

Run Actions

This section contains a list of all the run actions. Each action is described in detail, along with notes and examples that you may find helpful.

These actions can be inserted in the Run Actions list for a given slide, or in the Global Run Actions list. You insert an action by either using the Run Insert command, which also adds a Key/Event label to the action line, or by using the Run Line action. Note that the Run Line action is the first one in the Run menu, so you can execute it by just pressing Enter on the Run menu.

You can change actions on an action line by using the Run Action command. You can change the Key/Event label on an action line by using the Run Key/Event command.

See the "Run Commands" section for more information about the commands on the Run Menu, and the "How Running Works" section for more information about running in general.

Which Run Actions Do I Use?

There are so many run actions that, at first, you may feel that it is hard to know which ones to use. That is not really the case. Very few of the run actions are used frequently. In fact, the most commonly used run action is the first one, the View Slide run action. Other commonly used run actions are: Quit, Tone Beep, File, and "=".

If you have a programming background, when you are performing many operations the run actions you need should be obvious. For example, if you are testing conditions, you use the If run actions. If you are manipulating strings, you use the run actions on the String submenu. For those without a programming background, the examples in the manual and on the diskette should show you how things can be done.

You may find it helpful to look over the list of run action menu items in the "Command List" section of the manual. Each item has a short, one-line description.

You should also look at the values controlled by the Get Builtin and Set Builtin run actions on the Miscellaneous submenu. These can be very useful for special effects.

Remember that you do not have to use, or understand, all of these run actions. A complete set is provided for those who need them. You may not be one of those people. Users of the older **Dan Bricklin's Demo Program** did very well with only View Slide, a few tones, Quit, File, Use, Global, and just a few other run actions.

Run Actions Listed in Menu Order

The run actions descriptions are presented in this section of the manual in the same order as the Run Action ?List command, which is basically the same order as the menus. A list in that order appears below. The Run menu submenu item and the menu item on the final menu, which are used to access the run action, are listed on the left. You can use the submenu and menu item listings to find the run action description in this section. A list of the run action forms presented in alphabetic order follows this list.

Submenu	Menu Item	Action Form in Menu Order
	View	View Slide SLIDE-REF?
	Quit	Quit
	Nothing	*(nothing)*
	=	VARIABLE? = VALUE?
	+	VARIABLE? = VALUE? + VALUE?
	-	VARIABLE? = VALUE? - VALUE?
	*****	VARIABLE? = VALUE? * VALUE?
	/	VARIABLE? = VALUE? / VALUE?
	%	VARIABLE? = VALUE? % VALUE?
	&	VARIABLE? = VALUE? & VALUE?
	|	VARIABLE? = VALUE? | VALUE?
	Incr	Increment VARIABLE? By VALUE?
	Decr	Decrement VARIABLE? By VALUE?
Tone	**Beep**	Tone Beep
	Thud	Tone Thud
	Sound	Sound STRING? For VALUE? Times
	Note	Tone Note VALUE? For 1/18-SECS?
	Melody	Melody STRING?
Other-Slide	**Call**	Call Slide SLIDE-REF?
	Return	Slide Return
	After	Slide Return After
	File	File FILENAME?
	Slide-File	View SLIDENUM? In File FILENAME?
	Offset-View	Offset View VALUE?
	View-Abs	View Absolute SLIDENUM?
	Name	VARIABLE? = Slide With Name: STRING?
	DispThenTag	View Slide SLIDENUM?, Then Tag STRING?
Key/Event	**Transfer**	Transfer To Key/Event VALUE?
	Call-K/E	Call Key/Event VALUE?
	Return-K/E	Key/Event Return
	Use	Use Actions On SLIDE-REF?
	Global	Use Global Actions
	Default	Continue with Default Action
Prog	**If =**	If (VALUE? = VALUE?) *(upper/lowercase insensitive)*
	If ==	If (VALUE? == VALUE?) *(uppper/lowercase sensitive)*
	If <	If (VALUE? < VALUE?)
	If <=	If (VALUE? <= VALUE?)
	If !=	If (VALUE? != VALUE?) *(not equal; case sensitive)*
	If >=	If (VALUE? >= VALUE?)
	If >	If (VALUE? > VALUE?)
	If In	If (VALUE? In STRING?)
	If Between	If (VALUE? <= VALUE? <= VALUE?)
	If Upper	If (VALUE? Is Upper-Case)
	If Lower	If (VALUE? Is Lower-Case)
	If Alpha	If (VALUE? Is Alphabetic)
	If #Num	If (VALUE? Is Numeric)
	If NumLet	If (VALUE? Is Alphanumeric)
	If Func-Key	If (VALUE? Is Function Key)
	If Edit-Key	If (VALUE? Is Edit Key)
	If Text	If (VALUE? Is Text)
	If Key	If (VALUE? Is Key)
	If WaitKey	If (Waiting For Key)
	If ^Waiting	If (Not Waiting For Key)
	If Shift	If (Shift Key is Down)
	If Ctrl	If (Ctrl Key is Down)
	If *Alt	If (Alt Key is Down)
	-Else	-Else
	For	For VARIABLE? = FIRST? To LAST?
	While =	While (VALUE? = VALUE?) *(case insensitive)*
	While ==	While (VALUE? == VALUE?) *(case sensitive)*
	While <	While (VALUE? < VALUE?)
	While <=	While (VALUE? <= VALUE?)
	While !=	While (VALUE? != VALUE?) *(not equal; case sensitive)*
	While >=	While (VALUE? >= VALUE?)
	While >=	While (VALUE? >= VALUE?)
	While In	While (VALUE? In STRING?)
	While Betwn	While (VALUE? <= VALUE? <= VALUE?)
	While Upper	While (VALUE? Is Upper-Case)
	While Lower	While (VALUE? Is Lower-Case)

	While Alpha	While (VALUE? Is Alphabetic)	
	While #Num	While (VALUE? Is Numeric)	
	While NumLt	While (VALUE? Is Alphanumeric)	
	While Func	While (VALUE? Is Function Key)	
	While Edit	While (VALUE? Is Edit Key)	
	While Text	While (VALUE? Is Text)	
	While Key	While (VALUE? Is Key)	
	While Wait	While (Waiting For Key)	
	While ^Wait	While (Not Waiting For Key)	
	While Shift	While (Shift Key is Down)	
	While Ctrl	While (Ctrl Key is Down)	
	While *Alt	While (Alt Key is Down)	
	Block	Block Repeat	
	Leave	Leave Group	
	Again	Again	
	Select	Select (VALUE?)	
	Case	Case VALUE?:	*(uppercase/lowercase insensitive)*
	Otherwise	Otherwise:	
	Done	Done Select	
	End If	End-If	
	End For	End-For	
	End While	End-While	
	End Block	End-Block	
	End Select	End-Select	
	Goto-Tag	Goto Tag STRING?	*(1 to 8 char constant)*
	Tag-Call	Call Tag STRING?	*(1 to 8 char constant)*
	Return-from	Return From Tag Call	
	...	...	
String	**Set**	STRING-VARIABLE?[POSITION?] = VALUE?	
	Get	VARIABLE? = STRING?[POSITION?]	
	Append	Append VALUE? To STRING-VARIABLE?	
	Backspace	Backspace STRING-VARIABLE?	
	Delete	Delete COUNT? At STRING?[POSITION?]	
	Insert	Insert VALUE? Before STRING-VARIABLE?[POSITION?]	
	Overwrite	Overwrite STRING-VAR?[POSITION?] With STRING?	
	Fill	Fill STRING-VARIABLE? With COUNT? of VALUE?	
	Erase	Erase STRING-VARIABLE?	
	Length	VARIABLE? = Length(STRING?)	
	Process	Process-Char STRING-VAR?	
	Convert	Convert VALUE? To VARIABLE?	
	Where	VARIABLE? = Where In STRING? Is VALUE?	
	Trim	Trim STRING-VARIABLE?	
	Xtract	Extract STRING-VARIABLE? From START? To END?	
	Replace	Replace STRING? With STRING? In STRING-VAR?	
Misc	**Key**	VARIABLE? = Current Key/Event	
	Current	VARIABLE? = Current Slide Number	
	Flush	Flush Type-Ahead	
	Redisplay	Redisplay Screen	
	Pause	Pause VALUE? 1/18-Seconds	
	Long-Pause	Pause VALUE? Even If Key Pressed	
	NextKey	Next Key	
	Input	VARIABLE? = Input A Key	
	Exec	ERRORCODE? = Exec FILENAME? With ARGUMENT-STRING?	
	Mouse	Mouse (COMMAND?, VALUE?, VALUE?)	
	AttribTrans	Translate Attribute STRING? To STRING?	
	TransltCmds	Translate Attribute Cmd VALUE?	
	Debug	Debug	
	Get-Builtin	VARIABLE? = Builtin(VALUE?)	
	Set-Builtin	Builtin(VALUE?) = VALUE?	
File	**Open**	ERRORCODE? = Open File FILENAME? For MODE:R/W/U/A?	
	Close	Close File	
	Read	ERRORCODE? = Read Text Into STRING-VARIABLE?	
	#Read	ERRORCODE? = Read COUNT? Bytes Into STRING-VAR?	
	Write	ERRORCODE? = Write STRING?	
	***WriteNCRLF**	ERRORCODE? = Write STRING? With No CR/LF	
	Seek	ERRORCODE? = Seek To VALUE?	
	Print	Print STRING?	
	&PrintNCRLF	Print STRING? With No CR/LF	

Run Actions Listed in Alphabetic Order

Below is a list of the run actions in alphabetic order as they are displayed in the action form. The Run menu submenu item, and the menu item on the final menu which is needed to access the run action, are listed on the left. You can use the submenu and menu item listings to find the run action description in this section. The descriptions are presented in menu order. A listing appears above of all the run action forms in menu order.

SubMenu	Menu Item	Action Form in Alphabetic Order
	=	`VARIABLE? = VALUE?`
String	**Get**	`VARIABLE? = STRING?[POSITION?]`
String	**Set**	`STRING-VARIABLE?[POSITION?] = VALUE?`
	+	`VARIABLE? = VALUE? + VALUE?`
	-	`VARIABLE? = VALUE? - VALUE?`
	*****	`VARIABLE? = VALUE? * VALUE?`
	/	`VARIABLE? = VALUE? / VALUE?`
	%	`VARIABLE? = VALUE? % VALUE?`
	&	`VARIABLE? = VALUE? & VALUE?`
	|	`VARIABLE? = VALUE? | VALUE?`
Prog	**...**	`...`
Prog	**Again**	`Again`
String	**Append**	`Append VALUE? To STRING-VARIABLE?`
String	**Backspace**	`Backspace STRING-VARIABLE?`
Prog	**Block**	`Block Repeat`
Misc	**Get-Builtin**	`VARIABLE? = Builtin(VALUE?)`
Misc	**Set-Builtin**	`Builtin(VALUE?) = VALUE?`
Key/Event	**Call-K/E**	`Call Key/Event VALUE?`
Other-Slide	**Call**	`Call Slide SLIDE-REF?`
Prog	**Tag-Call**	`Call Tag STRING?` *(1 to 8 char constant)*
Prog	**Case**	`Case VALUE?:` *(case insensitive)*
File	**Close**	`Close File`
Key/Event	**Default**	`Continue with Default Action`
String	**Convert**	`Convert VALUE? To VARIABLE?`
Misc	**Key**	`VARIABLE? = Current Key/Event`
Misc	**Current**	`VARIABLE? = Current Slide Number`
Misc	**Debug**	`Debug`
	Decr	`Decrement VARIABLE? By VALUE?`
String	**Delete**	`Delete COUNT? At STRING?[POSITION?]`
Prog	**Done**	`Done Select`
Prog	**-Else**	`-Else`
Prog	**End Block**	`End-Block`
Prog	**End For**	`End-For`
Prog	**End If**	`End-If`
Prog	**End Select**	`End-Select`
Prog	**End While**	`End-While`
String	**Erase**	`Erase STRING-VARIABLE?`
Misc	**Exec**	`ERRORCODE? = Exec FILENAME? With ARGUMENT-STRING?`
String	**Xtract**	`Extract STRING-VARIABLE? From START? To END?`
Other-Slide	**File**	`File FILENAME?`
String	**Fill**	`Fill STRING-VARIABLE? With COUNT? of VALUE?`
Misc	**Flush**	`Flush Type-Ahead`
Prog	**For**	`For VARIABLE? = FIRST? To LAST?`
Prog	**Goto-Tag**	`Goto Tag STRING?` *(1 to 8 char constant)*
Prog	**If**	`If` *(condition)*
	Incr	`Increment VARIABLE? By VALUE?`
Misc	**Input**	`VARIABLE? = Input A Key`
String	**Insert**	`Insert VALUE? Before STRING-VARIABLE?[POSITION?]`
Key/Event	**Return-K/E**	`Key/Event Return`
Prog	**Leave**	`Leave Group`
String	**Length**	`VARIABLE? = Length(STRING?)`
Tone	**Melody**	`Melody STRING?`
Misc	**Mouse**	`Mouse (COMMAND?, VALUE?, VALUE?)`
Misc	**NextKey**	`Next Key`
	Nothing	*(nothing)*
Other-Slide	**Offset-View**	`Offset View VALUE?`
File	**Open**	`ERRORCODE? = Open File FILENAME? For MODE:R/W/U/A?`

Prog	**Otherwise**	`Otherwise:`
String	**Overwrite**	`Overwrite STRING-VAR?[POSITION?] With STRING?`
Misc	**Pause**	`Pause VALUE? 1/18-Seconds`
Misc	**Long-Pause**	`Pause VALUE? Even If Key Pressed`
File	**Print**	`Print STRING?`
File	**&PrintNCRLF**	`Print STRING? With No CR/LF`
String	**Process**	`Process-Char STRING-VAR?`
	Quit	`Quit`
File	**#Read**	`ERRORCODE? = Read COUNT? Bytes Into STRING-VAR?`
File	**Read**	`ERRORCODE? = Read Text Into STRING-VARIABLE?`
Misc	**Redisplay**	`Redisplay Screen`
String	**Replace**	`Replace STRING? With STRING? In STRING-VAR?`
Prog	**Return-from**	`Return From Tag Call`
File	**Seek**	`ERRORCODE? = Seek To VALUE?`
Prog	**Select**	`Select (VALUE?)`
Other-Slide	**Return**	`Slide Return`
Other-Slide	**After**	`Slide Return After`
Other-Slide	**Name**	`VARIABLE? = Slide With Name: STRING?`
Tone	**Sound**	`Sound STRING? For VALUE? Times`
Tone	**Beep**	`Tone Beep`
Tone	**Note**	`Tone Note VALUE? For 1/18-SECS?`
Tone	**Thud**	`Tone Thud`
Key/Event	**Transfer**	`Transfer To Key/Event VALUE?`
Misc	**AttribTrans**	`Translate Attribute STRING? To STRING?`
Misc	**TransltCmds**	`Translate Attribute Cmd VALUE?`
String	**Trim**	`Trim STRING-VARIABLE?`
Key/Event	**Use**	`Use Actions On SLIDE-REF?`
Key/Event	**Global**	`Use Global Actions`
Other-Slide	**Slide-File**	`View SLIDENUM? In File FILENAME?`
Other-Slide	**View-Abs**	`View Absolute SLIDENUM?`
	View	`View Slide SLIDE-REF?`
Other-Slide	**DispThenTag**	`View Slide SLIDENUM?, Then Tag STRING?`
String	**Where**	`VARIABLE? = Where In STRING? Is VALUE?`
Prog	**While**	`While (`*`condition`*`)`
File	**Write**	`ERRORCODE? = Write STRING?`
File	***WriteNCRLF**	`ERRORCODE? = Write STRING? With No CR/LF`

First Level Run Actions

View

View Slide *SLIDE-REF?*

This action lets you switch from the current slide that is being displayed to another slide. Unlike most of the other actions, it takes one argument which is a **reference** to the slide to view, instead of a variable or value. For example, instead of referring to the fifth slide by having the number 5, you actually show the slide on the screen. If the slide order is changed, the same slide in its new position is still referenced.

When setting the argument, another slide (usually the slide before or after the current slide, if it exists) is shown with a message box in the corner. You can use the F1/F2 keys to change which slide is displayed. Pressing Enter selects the slide shown.

Pressing the S key instead of Enter shows the list of all slides. You can use the Locate command at that point to quickly find a slide that is far from the current slide. Selecting it gets you back to the message box.

Pressing N or P chooses the Next or Previous slide, respectively. This is not an absolute reference to those slides, but one that is resolved at the time the action is executed. If you insert a new slide between a slide and one that was its "Next", the reference will be to the new next slide. If you make an explicit reference to the next slide (showing its name and number), the reference will follow the slide as its position changed with the insert.

Pressing E exits from the Run menu and lets you edit the slide being displayed.

Note 1

View is the normal way to switch explicitly to another slide. The Call Slide action is only used when you want to view one or more slides and then return. Note that when you want to go to the next slide no matter which key is pressed, you may not need any actions at all. See the description of the Run Type value.

Note 2

Next and Previous are always relative to the slide being displayed. If you "Use" the actions on another slide or have Global Actions, Next and Previous are still relative to the slide on the screen, not the slide being used or the slide on the screen when the Global action was defined.

Note 3

When another slide is viewed, a series of operations occur. These involve signaling certain Key/Events in order, and then taking default actions for most of them. This is all described in detail in the section "How Running Works". You may want to skim that section.

Briefly, first the Key/Event "Viewed" is signaled. The default action is to display the slide and then signal "Displayed." The default action for "Displayed" depends upon the Run Type value. It usually waits the Wait Value amount of time. For type 0, it then views the next slide. For type 1, it then signals "WaitDone." That, in turn, signals "Readkey". The default for "Readkey" is to input a key from the keyboard and then signal "Keypress", which by default signals the key itself as a Key/Event. The default for all keys is to view the next slide.

Usually, most of this can be ignored, and you can just check the definition for the Run Types and put in actions for the Key/Events that correspond to keystrokes. There are cases, though, in which you may want to "get control" at a special point: to log keystrokes to a file, for example, or even to implement your own set of Run Types.

Quit

Quit

Stops the program from running the slides and returns to editing. Any variables whose values have been changed retain their new values, and the last slide displayed remains on the screen. Runtime only versions of the program return to DOS.

Note

You can also stop running by pressing Ctrl-Break. You will be asked if you want to quit (the default — press Q, Esc or Enter) or continue running with Debug on (D).

Nothing

Sets the action to nothing — a blank line. This is useful for making a program more readable. You can still have comments on the line, and you can add an action later with the Run Action command.

Note

It is often helpful to make the first action for a Key/Event (the one with the Key/Event labeling the left) blank, so that you can insert other actions between the Key/Event label and the first real action.

= *VARIABLE? = VALUE?*

The value on the right is assigned to the variable on the left. If the variable is a string variable, then a string value is used. If it is a numeric variable, then a numeric value is used.

Remember, when converted to numbers, string values have the value of their first character (e.g., "A" is 65, its ASCII value). When converted to strings, numeric values are a one-character string with the character represented by the value, or 0 if they are >255 or <0 (e.g., 65 is "A").

Note

When string variables are first created, they are given the value of no characters (length is 0). When numeric variables are created, they are given the value 0. It is always good practice, though, to explicitly assign initial values to variables. Otherwise you may get unwanted results the second time you run a series of slides while editing.

String values can be set to null (length 0) by either using this action or erase:

```
strng1 = ""
```

or

```
Erase strng1
```

They both have the same effect.

+ *VARIABLE? = VALUE? + VALUE?*

Add the numeric value of the two values on the right, and assign the result to the variable on the left.

- *VARIABLE? = VALUE? - VALUE?*

The second value on the right of the "=" is subtracted from the first value on the right. The difference is assigned to the variable on the left.

***** *VARIABLE? = VALUE? * VALUE?*

The product of the right two values is assigned to the variable on the left.

/

VARIABLE? **=** *VALUE?* / *VALUE?*

The integer quotient, found by dividing the first numeric value to the right of the "=" by the second value, is assigned to the variable on the left.

%

VARIABLE? **=** *VALUE?* **%** *VALUE?*

The integer remainder, when dividing the first numeric value on the right of the "=" by the second value, is assigned to the variable on the left (e.g., x=15%10 assigns 5 to x).

&

VARIABLE? **=** *VALUE?* **&** *VALUE?*

The variable on the left of the "=" is assigned the computer-type, Boolean AND of the 16 bits of the first value on the right with the 16 bits of the second value on the right.

|

VARIABLE? **=** *VALUE?* | *VALUE?*

The variable on the left of the "=" is assigned the computer-type, Boolean OR of the 16 bits of the first value on the right with the 16 bits of the second value on the right.

Incr

Increment *VARIABLE?* **By** *VALUE?*

The value on the right is added to the current value of the variable, and then the result is assigned to the variable.

Decr

Decrement *VARIABLE?* **By** *VALUE?*

The value on the right is subtracted from the current value of the variable, and the difference is assigned to the variable.

Tone Run Actions

These are the run actions for producing sounds.

Tone Beep

Tone Beep

Makes a "beep" sound immediately after the next time a slide is displayed or redisplayed (with the Redisplay action).

Tone Thud

Tone Thud

Makes a "thud" sound immediately after the next time a slide is displayed or redisplayed (with the Redisplay action).

Tone Sound

Sound *STRING?* **For** *VALUE?* **Times**

Immediately (not waiting for display time) makes the sound described by the string for the appropriate number of times.

The string consists of the letters A-Z (case insensitive), where each character tells the program to wait a specified amount of time and then toggle the position of the speaker. The letter A represents a short time between moving the speaker, B a slightly longer time, ..., and Z the longest time.

Example

To make a "cluck" sound, like a tick-tock clock, you could use:

```
Sound "BBBCFFFHHHHKKKKPPPP" For 1 Times
```

You would get a "zipping" or "ripping" sound from:

```
Sound "QQQQQQQRRRRRRRMMMMMMMHHHHHHHBBBBBBB" 1 Times
```

Note

The time constant used to create the sounds is the "system speed" that is determined at start-up. You can get/set this value with the Builtin(4) actions.

Tone Note

Tone Note *VALUE?* **For** *1/18-SECS?*

Sounds the note, specified by the numeric value, for the specified number of clock ticks ($^{1}/_{18.2}$th seconds). The note numbers are specified in the Tone Chart Appendix. The sound is made immediately after the next time the screen is updated on a display or redisplay.

Tone Melody

Melody *STRING?*

Plays a series of tones on the PC's speaker. The string specifies the notes and their duration. No wait is done for a redisplay — the sound is made immediately.

The string consists of three-digit sequences, optionally separated by spaces. The first two digits in the sequence are the note number (see the Tone Chart Appendix), and the third is the duration of the note in 1/9 seconds. The note 00 is interpreted as a pause (silence). A comma (",") turns the sound off for an instant, which can be useful when two identical notes immediately follow one another.

Example

To play the start of "Mary Had A Little Lamb", you could use the action:

```
Melody "523 501 482 502 522,522 524 502,502 504 522 552
                                    554"
```

Other-Slide Run Actions

These are the run actions for viewing other slides, both in the current slide show, and in other files.

Other-Slide Call

Call Slide *SLIDE-REF?*

Switches from the current slide, which is being displayed, to another slide, and return to this slide (or the one following it) at a later point. Its operation is very similar to the

View action. You return by using either the Slide Return or Slide Return After actions.

See the View action above for more information and a variety of notes.

Note

You may call a slide that continues switching from slide to slide and then calls another slide, etc. The maximum number of Call Slide actions that can be executed, without executing their respective Slide Returns or Slide Return Afters, is 99.

In general there should always be a Slide Return or Slide Return After executed for each Call Slide. More than one slide can Call a given slide, similar to subroutines in a normal computer language. ***If you are not going to return, use the View action.***

Other-Slide Return

Slide Return

Returns from a corresponding Call Slide action by moving back to and "viewing" the most recent slide that has done a Call Slide action and has not been "returned" to.

Note

There are two types of returns, the Slide Return and the Slide Return After. Make sure that you use the one that is appropriate in each case. For example, to go to an error-message slide when an incorrect key is pressed and then return to that same slide where the incorrect key was pressed to let the user try again, use Slide Return. If you want to show a set of slides at the beginning of each chapter and then continue, use Slide Return After.

Other-Slide After

Slide Return After

Functions like Slide Return, but returns to the slide immediately following the slide that made the call.

Note

See the note to the Slide Return action above.

Other-Slide File

File *FILENAME?*

Loads another file in place of the current set of slides, variables, etc., and starts running with the slide that was on the screen when it was saved. The argument is a string with the name of the file to be loaded. It can be a full pathname, and ***includes extensions***. The Attribute Translate Table is *not* loaded, so translating can continue.

Examples

```
  File "lesson2.dbd"
```

or

```
  File "c:\cbt\lesson3.dbd"
```

or

```
filename = "graphic1.dbd"
File filename
```

Note

Any variables that appear both currently and in the file being loaded, which have the Passed On setting set "on" in ***both*** places, will have the current value passed on to the new file, replacing the value loaded from the file in this instance.

Other-Slide Slide-File

View *SLIDENUM?* In File *FILENAME?*

Functions like the File action, but in addition, starts running with the specified slide number in that file.

Other-Slide Offset-View

Offset View *VALUE?*

This is a variation of the View action that uses a value rather than a slide reference. The value specifies a slide relative to the current slide to view.

Example

To view the slide after the next slide, you could use:

```
Offset View 2
```

To view the slide 3 slides before this slide, you could use:

```
Offset View -3
```

Other-Slide View-Absolute

View Absolute *SLIDENUM?*

This is a variation of the View action that uses a value instead of a slide reference. The value is the number of the slide to view, with 1 being the first slide.

You can find out a slide's number with the Slide With Name run action, and with the Current Slide Number run action.

Example

To view the third slide:

```
View Absolute 3
```

Other-Slide Name

VARIABLE? = Slide With Name: *STRING?*

The variable on the left is assigned the number of the first slide with a name that starts with the value of the string on the right. The search starts with slide 1, and if no match is found, the value is 0.

Example

To view the slide with a name that starts with "Main Menu", you could use:

```
mm_slide = Slide With Name: "Main Menu"
View Absolute mm_slide
```

Other-Slide Display-then-Tag

View Slide *SLIDENUM?*, **Then Tag** *STRING?*

This is a variation of the View action that uses the slide number instead of a slide reference. It is similar to the View Absolute action, except that instead of doing the normal operations when the slide is viewed (signaling "Viewed", waiting, checking the slide Run Type, etc.), it immediately does a Goto-Tag of the string. See the Goto-Tag action below.

When you use this action, the slide viewed is not displayed until the next redisplay. Redisplaying occurs when a Redisplay command is executed or when the last action in an action list is executed — see the "How Running Works" section. You may want to have the actions at the tag end by doing a Transfer to Key/Event "Viewed". This is a very programmer-oriented feature, and is not meant for general use.

Key/Event Run Actions

Key/Event Transfer

Transfer To Key/Event *VALUE?*

This action allows you to signal a particular Key/Event. See the section, "How Running Works", for a detailed description of how the lists of actions are searched for a match when a Key/Event is signaled.

You can use this action to have more than one key do some processing, and then start doing what another key would have done.

Note 1

The "last typed" Key/Event value is not affected by a Transfer. The "last typed" Key/Event always refers to the last key typed, which you can get/set with the Builtin(1) actions. The last key typed value is also used by the Process-Char action, not the event last signaled.

Note 2

When a Transfer action is done, like all Key/Event signaling, the searching for a match starts at the top of the list of the current slide, not the slide being Used or the Global Run Action list.

Note 3

Pressing "!" when setting the value brings up a list of Key/Events and their values.

Key/Event Call-Key/Event

Call Key/Event *VALUE?*

This is a variation of the Transfer To Key/Event action. The only difference is that the action line where the call is made is remembered so that it can be used in a corresponding Key/Event Return action. Key/Events may be called to a depth determined by the Run Stack (see "How Running Works").

Key/Event Return-from-Key/Event

Key/Event Return

Returns execution to the action line immediately following the corresponding Call Key/Event action.

Key/Event Use

Use Actions On *SLIDE-REF?*

Searches the run action list of the referenced slide for a match to the Key/Event that got you to the most recent Key/Event label. This lets you say "do whatever that slide would do." This action, unlike most actions, has a slide reference as its argument in a manner similar to the View Slide action.

See the "How Running Works" section for more information about searching and the Use action.

Note 1

The Key/Event label that got you to this action may be more or less explicit than the one that will match on the slide Used. For example, if the Key/Event being matched is "z" and you matched "Any Lower" on this slide, the Key/Event label "Zz" can still match on the slide being Used.

Note 2

If the Key/Event does not find a match on the slide being Used, then the Global Run Actions will be searched. If the Key/Event does not find a match on the Global Run Action list, then the default action for that Key/Event will be executed. For keystrokes, the default is to view the next slide. See "How Running Works" for more information about defaults.

An action on the slide being Used can have Use run actions that refer to other slides. There is no limit to the depth of such references. If there is a loop, you will need to press Ctrl-Break to stop the searching.

Key/Event Global

Use Global Actions

Searches the Global Run Action list for a match to the Key/Event that got you to the most recent Key/Event label.

See the "How Running Works" section for more information about searching and the Global action.

Note 1

The Key/Event label that got you to this action may be more or less explicit than the one that will match on the Global Run Action list. For example, if the Key/Event being matched is "z" and you matched "Any Lower" on this slide, the Key/Event label "Zz" can still match on the Global Run Action list.

Note 2

If the Key/Event does not find a match on the Global Run Action list, then the default action for that Key/Event will be executed. For keystrokes, the default is to view the next slide. See the "How Running Works" section for more information about defaults.

Be careful about having the Use Global Actions run action on the Global Run Action list; you will have to Ctrl-Break out of the loops!

Key/Event Default

Continue with Default Action

Executes the default action for the Key/Event that got you to the *most recent* Key/Event label. This lets you add some processing to the special Key/Events, such as "Viewed", "Displayed", etc., and then continue normally. See "How Running Works" for a list of the default actions. Do not use this action if a Tag Call or Goto Tag is the most recent Key/Event.

Example

To add a "Woodpecker" sound to be played whenever a particular slide is viewed, you could add this set of actions with Key/Event label:

```
Displayed Sound "hhhhhhmmmrrtrrmmmhhhhh" For 10 Times
          Continue with Default Actions
```

Note

In some cases you will want to use a Use Global Actions run action instead of a Continue with Default Action run action. The most common time is on a slide's run action list when the Global Run Action list has a Key/Event label for the Key/Event being continued, and you want those global run actions to be executed.

Programming Run Actions

These are the run actions for testing values, looping, and transferring to other run actions.

Prog If

The If actions allow you to test a variety of conditions. You can execute different actions depending upon the results of the tests.

All of the If actions have a similar syntax. The general form is:

```
If (condition)
   true-action1
   true-action2...
End-If
```

or:

```
If (condition)
   true-action1
   true-action2...
-Else
   false-action1
   false-action2...
End-If
```

The "true" actions are executed if the condition is true; otherwise they are skipped. The "false" actions are executed if the condition is false; otherwise they are skipped. The -Else and the "false" actions are optional. All If actions, though, must have a corresponding End-If.

The If actions are defined by using the Programming If command to show a menu of conditions. The -Else action is on the Programming menu, and the End-If action is on the Programming End menu. Alternatively, you can use the ?List command on the main Run Action menu to get a list of all actions.

The actions that are added to new lines, with the Run Line command, are automatically indented by the amount specified on the Global menu. The default is three spaces. This is a one-time addition of space — it is not recalculated if the If action is removed. You can always increase or decrease the indenting on a given line by pressing the Ins or Del keys, respectively. The indenting only makes the actions more readable; it has no effect on execution. You can turn off the automatic indenting by setting the Global Indent Increment to 0.

The true and false actions can be any type of action, including other If actions. They may not have a Tag or Key/Event label.

The If actions may be nested (along with loops, Call Key/Events, Call Tags, etc.) to a depth determined by the Run Stack (see "How Running Works").

Prog
If
=

If (*VALUE?= VALUE?*)

Compares the first value to the second value. If the values are the same, the condition is "true"; otherwise it is "false."

If both values are strings, then a string comparison is made; otherwise, a numeric comparison is done.

String values are the same if they are the same length and have the same characters. ***This test is uppercase/lowercase insensitive*** — that is, "abcd" and "AbCd" are considered to be the same. Use the If == action if you need a case-sensitive comparison.

Prog
If
= =

If (*VALUE?== VALUE?*)

Compares the first value to the second value. If the values are the same, the condition is "true"; otherwise it is "false."

If both values are strings, then a string comparison is made; otherwise, a numeric comparison is done.

String values are the same if they are the same length and have the same characters. ***This test is uppercase/lowercase sensitive*** — that is, "abcd" and "AbCd" are considered different. Use the If = action if you need a case insensitive comparison.

Note

The initial character of the command is the same as If =. In order to select this comparison, you must move the cursor to the == command and press Enter. If you just typed =, you would get the = comparison, not the == one.

Prog
If
<

If (_VALUE?_ **<** _VALUE?_**)**

Compares the first value to the second value. If the numeric value of the first value is less than the second, the condition is "true"; otherwise it is "false." This is a signed comparison (-1 is less than 0).

Prog
If
<=

If (_VALUE?_ **<=** _VALUE?_**)**

Compares the first value to the second value. If the numeric value of the first value is less than or equal to the second, the condition is "true"; otherwise it is "false." This is a signed comparison (-1 is less than 0).

Note

The initial character of the command is the same as If <. In order to select this comparison, you must move the cursor to the <= command and press Enter. If you just typed <, you would get the < comparison, not the <= one.

Prog
If
!=

If (_VALUE?_ **!=** _VALUE?_**)**

Compares the first value to the second value. If the values are not the same, the condition is "true"; otherwise it is "false."

If both values are strings, then a string comparison is made. Otherwise, a numeric comparison is done.

String values are the same if they are the same length and have the same characters. ***This test is uppercase/lowercase sensitive*** — that is, "abcd" and "AbCd" are considered different. Use the If = action if you need a case insensitive comparison. Note that you may have to put the actions in the -Else section. (You may have an If immediately followed by an -Else, if there is nothing to be done in the "true" part.)

Prog
If
>=

If (_VALUE?_ **>=** _VALUE?_**)**

Compares the first value to the second value. If the numeric value of the first value is greater than or equal to the second, the condition is "true"; otherwise it is "false." This is a signed comparison (-1 is less than 0).

Prog
If
>

If (*VALUE?* > *VALUE?*)

Compares the first value to the second value. If the numeric value of the first value is greater than the second, the condition is "true"; otherwise it is "false." This is a signed comparison (-1 is less than 0).

Note

The initial character of the command is the same as If >=. In order to select this comparison, you must move the cursor to the > command and press Enter. If you just typed >, you would get the >= comparison, not the > one.

Prog
If
In

If (*VALUE?* In *STRING?*)

This condition is "true" if any one of the characters in the string is the same as the value. If the value is numeric, it is treated as the character value. This is case-sensitive.

Example

This would result in a "true" comparison:

```
            If ("e" In "AEIOUaeiou")
```

Prog
If
Between

If (*VALUE?* <= *VALUE?* <= *VALUE?*)

The condition is "true" if the second value is greater than or equal to the first, as well as less than or equal to the third; otherwise it is "false." All values are treated as numeric, and comparisons are on signed values (-1 is less than 0).

Prog
If
Upper

If (*VALUE?* Is Upper-Case)

The condition is "true" if all of the characters in the value are uppercase letters. It is "false" otherwise. The value is treated as a string.

Prog
If
Lower

If (*VALUE?* Is Lower-Case)

The condition is "true" if all of the characters in the value are lowercase letters. It is "false" otherwise. The value is treated as a string.

Prog
If
Alpha

If (*VALUE?* Is Alphabetic)

The condition is "true" if all of the characters in the value are alphabetic (a-z, A-Z). It is "false" otherwise. The value is treated as a string.

Prog
If
#Num

If (*VALUE?* Is Numeric)

The condition is "true" if all of the characters in the value are numeric (0-9). It is "false" otherwise. The value is treated as a string.

Prog
If
NumLet

If (*VALUE?* Is Alphanumeric)

The condition is "true" if all of the characters in the value are alphanumeric (a-z, A-Z, 0-9). It is "false" otherwise. The value is treated as a string.

Prog
If
Func-Key

If (*VALUE?* Is Function Key)

The condition is "true" if the value represents a valid function key Key/Event (F1-F10, Shift F1-Shift F10, Ctrl F1-Ctrl F10, Alt F1-Alt F10). It is "false" otherwise. The value is treated as a numeric value. Pressing "!" when setting the value brings up a list of Key/Events and their values.

Prog
If
Edit-Key

If (*VALUE?* Is Edit Key)

The condition is "true" if the value represents a valid edit Key/Event (Enter, Backspace, ^Backspace, Esc, Ins, Del), or a valid cursor Key/Event (arrows, Home, End, etc.). It is "false" otherwise. The value is treated as a numeric value. Pressing "!" when setting the value brings up a list of Key/Events and their values.

Prog
If
Text

If (*VALUE?* Is Text)

The condition is "true" if all of the characters in the value are text (any letters, any numbers, and any punctuation). It is "false" otherwise. The value is treated as a string.

Prog
If
Key

If (*VALUE?* Is Key)

The condition is "true" if the value represents a Key/Event that is from the keyboard and not a pseudo-key like "Displayed" or "Any Letter". It is "false" otherwise. The value is treated as a numeric value. Only the fact that the value is in the range of legal values is checked — not every value in that range is legal. Pressing "!" when setting the value brings up a list of Key/Events and their values.

Prog
If
WaitforKey

If (Waiting For Key)

The condition is "true" if no keys have been "typed ahead" and none are waiting to be processed. It is "false" if the user has pressed keys that have not been read and processed.

Prog
If
^Waiting

If (Not Waiting For Key)

The condition is "true" if the user has pressed keys that have not been read and processed (i.e., there is "type ahead"). It is "false" if no keys are "typed ahead."

Prog
If
Shift

If (Shift Key is Down)

The condition is "true" if either of the "Shift" keys is currently pressed. The Shift keys switch from lowercase to uppercase, etc. It is "false" if neither of them are pressed.

Prog
If
Ctrl

If (Ctrl Key is Down)

The condition is "true" if the Ctrl key is currently being pressed. It is "false" otherwise.

Prog
If
***Alt**

If (Alt Key is Down)

The condition is "true" if the Alt key is currently being pressed. It is "false" if no Alt key is being pressed.

Prog
-Else

-Else

Marks action as the end of the "true actions" and the beginning of the "false actions." The -Else is paired with the last If action without a corresponding End-If. For each If action there may be only one -Else action. The -Else is optional — if there are no "false actions", the -Else may be omitted. See the discussion above for more information on the If action.

Prog
For

For *VARIABLE? = FIRST?* To *LAST?*

Lets you execute a series of actions a specified number of times. It is analogous to the FOR statement in a language such as BASIC.

The general form of a For loop is:

```
For loopvar = value1 To value2
   loop-action1
   loop-action2...
End-For
```

When the For action is executed, the loop variable is assigned the numeric value1. If value1 is greater than value2, the loop actions are not executed, and execution continues with the action after the End-For action. If value1 is less than or equal to value2, the loop actions are executed.

Each time, after the loop actions are executed and the End-For action is encountered, the loop variable is incremented by 1. That new value is then compared with the current value of value2. If it is still less than or equal to value2, the loop actions are executed again. If the loop variable's incremented value is now greater than value2, execution resumes with the action after the End-For action.

The actions that are added to new lines (with the Run Line command) are automatically indented by the amount specified on the Global menu. (Default is three spaces.) This is a one-time addition of space — it is not recalculated if the For action is removed. You can always increase or decrease the indenting on a given line by pressing the Ins or Del keys, respectively. The indenting only makes the actions more readable; it has no effect on execution. You can turn off the automatic indenting by setting the Global Indent Increment to 0.

The loop actions can be any type of action, including other For actions. They may not have a Tag or Key/Event label. The actions can modify the value of the loop variable.

The For actions may be nested (along with Ifs, Call Key/Events, Call Tags, etc.) to a depth determined by the Run Stack (see "How Running Works").

In addition to the End-For action, you can use the Again action to act as if you encountered the End-For in the middle of the actions. You can also use the Leave Group action to exit the loop at any point and continue executing after the End-For action.

Example

To display the current slide five times, with the variable "c_offset" successively having the values 5, 10, 15, 20 and 25 (for example, to change the column offset of an overlay), you could use the actions:

```
For lv = 1 To 5
   c_offset = 5 * lv
   Redisplay Screen
End-For
```

To do the same thing, but to skip the 20 value (only displaying four times), you could use:

```
For lv = 1 To 5
   If (lv = 4)
      Again
   End-If
   c_offset = 5 * lv
   Redisplay Screen
End-For
```

Prog While

Allows you to test a variety of conditions, and then repeatedly execute a series of actions as long as the condition is "true."

All of the While actions have a similar syntax. The general form is:

```
While (condition)
   while-action1
   while-action2...
End-While
```

If the condition is "false" when the While action is initially encountered, the while actions are skipped, and execution continues immediately following the matching End-While action. If the condition is "true," the while actions are executed, and then the condition is checked again. If it is still "true," the actions are executed again. If it is now "false," execution resumes after the End-While.

The actions that are added to new lines (with the Run Line command) are automatically indented by the amount specified on the Global menu. (Default is three spaces.) This is a one-time addition of space — it is not recalculated if the While action is removed. You can always increase or decrease the indenting on a given line by pressing the Ins or Del keys, respectively. The indenting only makes the actions more readable; it has no effect on execution. You can turn off the automatic indenting by setting the Global Indent Increment to 0.

The While actions can be any type of action, including other While actions. They cannot have a Tag or Key/Event label.

The While actions can be nested (along with Ifs, Call Key/Events, Call Tags, etc.) to a depth determined by the Run Stack (see "How Running Works").

In addition to the End-While action, you can use the Again action to act as if you encountered the End-While in the middle of the actions. You can also use the Leave Group action to exit the loop at any point and continue executing after the End-While action.

Prog
While
=

While (*VALUE?* = *VALUE?*)

Compares the first value to the second value. If the values are the same, the condition is "true"; otherwise it is "false."

If both values are strings, then a string comparison is made; otherwise, a numeric comparison is done.

String values are the same if they are the same length and have the same characters. ***This test is uppercase/lowercase insensitive*** — that is, "abcd" and "AbCd" are considered the same. Use the While == action if you need a case-sensitive comparison.

Prog
While
= =

While (*VALUE?* == *VALUE?*)

Compares the first value to the second value. If the values are the same, the condition is "true"; otherwise it is "false."

If both values are strings, then a string comparison is made; otherwise, a numeric comparison is done.

String values are the same if they are the same length and have the same characters. ***This test is uppercase/lowercase sensitive*** — that is, "abcd" and "AbCd" are considered to be different. Use the While = action if you need a case-insensitive comparison.

Note

The initial character of the command is the same as While =. In order to select this comparison, you must move the cursor to the == command and press Enter. If you just typed =, you would get the = comparison, not the == one.

Prog
While
<

While (*VALUE?* < *VALUE?*)

Compares the first value to the second value. If the numeric value of the first value is less than the second, the condition is "true"; otherwise it is "false." This is a signed comparison (-1 is less than 0).

Prog
While
<=

While (*VALUE?* <= *VALUE?*)

Compares the first value to the second value. If the numeric value of the first value is less than or equal to the second, the condition is "true"; otherwise it is "false." This is a signed comparison (-1 is less than 0).

Note

The initial character of the command is the same as While <. In order to select this comparison, you must move the cursor to the <= command and press Enter. If you just typed <, you would get the < comparison, not the <= one.

Prog
While
!=

While (*VALUE?* != *VALUE?*)

Compares the first value to the second value. If the values are not the same, the condition is "true"; otherwise it is "false."

If both values are strings, then a string comparison is made; otherwise, a numeric comparison is done.

String values are the same if they are the same length and have the same characters. ***This test is uppercase/lowercase sensitive*** — that is, "abcd" and "AbCd" are considered different.

Prog
While
>=

While (*VALUE?* >= *VALUE?*)

Compares the first value to the second value. If the numeric value of the first value is greater than or equal to the second, the condition is "true"; otherwise it is "false." This is a signed comparison (-1 is less than 0).

Prog
While
>

While (*VALUE?* > *VALUE?*)

Compares the first value to the second value. If the numeric value of the first value is greater than the second, the condition is "true"; otherwise it is "false." This is a signed comparison (-1 is less than 0).

Note

The initial character of the command is the same as While >=. In order to select this comparison, you must move the cursor to the > command and press Enter. If you just typed >, you would get the >= comparison, not the > one.

Prog
While
In

While (*VALUE?* In *STRING?*)

This condition is "true" if any one of the characters in the string is the same as the value. If the value is numeric, it is treated as the character value. This is case-sensitive.

Prog
While
Between

While (*VALUE?* <= *VALUE?* <= *VALUE?*)

The condition is "true" if the second value is greater than or equal to the first, as well as less than or equal to the third; otherwise it is "false." All values are treated as numeric, and comparisons are on signed values (-1 is less than 0).

Prog
While
Upper

While (*VALUE?* Is Upper-Case)

The condition is "true" if all of the characters in the value are uppercase letters. It is "false" otherwise. The value is treated as a string.

Prog
While
Lower

While (*VALUE?* Is Lower-Case)

The condition is "true" if all of the characters in the value are lowercase letters. It is "false" otherwise. The value is treated as a string.

Prog
While
Alpha

While (*VALUE?* Is Alphabetic)

The condition is "true" if all of the characters in the value are alphabetic (a-z, A-Z). It is "false" otherwise. The value is treated as a string.

Prog
While
#Num

While (*VALUE?* Is Numeric)

The condition is "true" if all of the characters in the value are numeric (0-9). It is "false" otherwise. The value is treated as a string.

Prog
While
NumLet

While (*VALUE?* Is Alphanumeric)

The condition is "true" if all of the characters in the value are alphanumeric (a-z, A-Z, 0-9). It is "false" otherwise. The value is treated as a string.

Prog
While
Func-Key

While (*VALUE?* Is Function Key)

The condition is "true" if the value represents a valid function key Key/Event (F1-F10, Shift F1-Shift F10, Ctrl F1-Ctrl F10, Alt F1-Alt F10). It is "false" otherwise. The value is treated as a numeric value. Pressing "!" while setting the value brings up a list of Key/Events and their values.

Prog
While
Edit-Key

While (*VALUE?* Is Edit Key)

The condition is "true" if the value represents a valid edit Key/Event (Enter, Backspace, ^Backspace, Esc, Ins, Del), or a valid cursor Key/Event (arrows, Home, End, etc.). It is "false" otherwise. The value is treated as a numeric value. Pressing "!" while setting the value brings up a list of Key/Events and their values.

Prog
While
Text

While (*VALUE?* Is Text)

The condition is "true" if all of the characters in the value are text (any letters, any numbers and any punctuation). It is "false" otherwise. The value is treated as a string.

Prog
While
Key

While (*VALUE?* Is Key)

The condition is "true" if the value represents a Key/Event that is from the keyboard, and not a pseudo-key like "Displayed" or "Any Letter". It is "false" otherwise. The value is treated as a numeric value. Only the fact that the value is in the range of legal values is checked — not every value in that range is legal. Pressing "!" while setting the value brings up a list of Key/Events and their values.

Prog
While
WaitforKey

While (Waiting For Key)

The condition is "true" if no keys have been "typed ahead" and are waiting to be processed. It is "false" if the user has pressed keys that have not been read and processed.

Example

To have a tone produced every second after a slide is displayed and until a key is pressed, then sound a different tone, and then process the key, you could use the following on the slide's run action list with a "Readkey" Key/Event label:

```
Readkey    While (Waiting For Key)
              Melody "481"
              Pause 18 1/18-Seconds
           End-While
           Melody "602"
           Continue with Default Action
```

Note that the Melody action is used, not the Tone Note action, since the Tone Note action does not sound the tone until next time the screen is redisplayed.

Prog
While
^Waiting

While (Not Waiting For Key)

The condition is "true" if the user has pressed keys that have not been read and processed (i.e., there is "type ahead"). It is "false" if no keys are "typed ahead."

Prog
While
Shift

While (Shift Key is Down)

The condition is "true" if either of the "Shift" keys are currently pressed. (The Shift keys switch from lowercase to uppercase, etc.) It is "false" if neither of them are pressed.

Prog
While
Ctrl

While (Ctrl Key is Down)

The condition is "true" if the Ctrl key is currently being pressed. It is "false" otherwise.

Prog
While
*Alt

While (Alt Key is Down)

The condition is "true" if the Alt key is currently being pressed. It is "false" if no Alt key is being pressed.

Prog
Block

Block Repeat

The Block Repeat action allows you to execute a series of actions repeatedly. You only exit the loop when an explicit Leave Group or transfer action of some sort is executed. It is similar to a "While (True)" loop.

The general form is:

```
Block Repeat
   block-action1
   block-action2...
End-Block
```

When the Block Repeat action is initially encountered, the block actions are executed, and then re-executed each time the End-Block action is encountered. Execution continues with the action immediately following the End-Block, if a Leave Group action is executed. The block can also be exited by executing a variety of transfer actions such as View, Transfer, Goto, Return, File, Use, Global, Continue with Default Action, etc.

The actions that are added to new lines (with the Run Line command) are automatically indented by the amount specified on the Global menu. (Default is three spaces.) This is a one-time addition of space — it is not recalculated if the Block Repeat action is removed. You can always increase or decrease the indenting on a given line by pressing the Ins or Del keys, respectively. The indenting only makes the actions more readable; it has no effect on execution. You can turn off the automatic indenting by setting the Global Indent Increment to 0.

The block actions can be any type of action, including other Block Repeat actions. They cannot have a Tag or Key/Event label.

The Block Repeat actions can be nested (along with Ifs, Call Key/Events, Call Tags, Loops, etc.) to a depth determined by the Run Stack (see "How Running Works").

In addition to the End-Block action, you can use the Again action to act as if you encountered the End-Block in the middle of the actions. You can also use the Leave Group action to exit the loop at any point and continue executing after the End-Block action.

Note

Block Repeat loops can be used to let you transfer control around a group of actions. You can have the last action in the block be a Leave Group action, so that the loop is executed, at most, once. Throughout the block, whenever you want to transfer to the action following the block, you execute a Leave Group action. With complicated conditionals, this can sometimes be a more straightforward way of writing the program.

Prog
Leave

Leave Group

Terminates the execution of the enclosing For, While, or Block Repeat loop. Execution resumes with the action immediately following the matching End action.

Prog
Again

Again

Terminates the execution of one loop through the enclosing For, While, or Block Repeat loop. Execution resumes as if the matching End action were encountered (i.e., executing the loop again or exiting, as appropriate).

Prog
Select

Select (*VALUE?*)

Allows you to execute any one of a number of groups of actions, depending upon the value of its argument. It is similar to the Switch statement in the C language.

The general form for using Select is:

```
Select (svalue)
    Case value1:
        select1-action1
        select1-action2...
    Case value2:
        select2-action1
        select2-action2...
    Case value3:...
    Otherwise:
        select0-action1
        select0-action2...
End-Select
```

When the Select action is executed, actions are skipped until a Case action, with a value the same as the svalue, is encountered. Then the actions after the Case action are executed. Execution continues, even if other Case or Otherwise actions are encountered (e.g., the select2 actions would be executed in the example above after the select1 actions, if svalue matched value1). The Done Select action causes execution to skip over all actions that follow until the closing End-Select action. The comparison with the svalue is numeric, if either svalue or value*n* is numeric, and a case insensitive string comparison, if both are strings. The Otherwise action is like a Case action that matches all values.

The actions that are added to new lines (with the Run Line command) are automatically indented by the amount specified on the Global menu. (Default is three spaces.) This is a one-time addition of space — it is not recalculated if the Block Select action is removed. You can always increase or decrease the indenting on a given line by pressing the Ins or Del keys, respectively. The indenting only makes the actions more readable; it has no effect on execution. You can turn off the automatic indenting by setting the Global Indent Increment to 0.

The select actions can be any type of action, including other Select actions. They cannot have a Tag or Key/Event label.

The Select actions can be nested (along with Ifs, Call Key/Events, Call Tags, Loops, etc.) to a depth determined by the Run Stack (see "How Running Works").

Leave Group and Again actions, encountered as one of the Select actions, refer to the containing loop (For, While, Block Repeat).

Example

To view the slide appropriate to a user's response to a question, you could use the following actions:

```
Select (response)
   case "blue":
      View Slide start blue [0014]
   case "red":
      View Slide start red [0038]
   case "green":
      View Slide start green [0053]
   case "orange":
      Melody "481601721"
      Call Slide Not Yet [0007]
      Erase response
      Done Select
   Otherwise:
      Tone Beep
End-Select
```

Prog
Case

Case *VALUE?*:

Specifies the start of a group of actions that are to be executed following a Select action, if the value matches the select value. See the description of the Select action, above.

Prog
Otherwise

Otherwise:

Specifies the start of a group of actions that are to be executed following a Select action, no matter what the select value. See the description of the Select action, above.

Prog
Done

Done Select

Causes the actions that follow to be skipped until the closing End-Select action after a Select action. See the description of the Select action.

Prog
End
If-End

End-If

Marks the end of the group of actions after an If action. For each If action there must be one, and only one, End-If action. See the description of the If action.

Prog
End
For-End

End-For

Marks the end of the group of actions to be executed after a For action. For each For action there must be one, and only one, End-For action. See the previous description of the For action.

Prog
End
While-End

End-While

Marks the end of the group of actions to be executed after a While action. For each While action there must be one, and only one, End-While action. See the previous description of the While action.

Prog
End
Block-End

End-Block

Marks the end of the group of actions to be executed after a Block Repeat action. For each Block Repeat action there must be one, and only one, End-Block action. See the previous description of the Block Repeat action.

Prog
End
Select-End

End-Select

Marks the end of the groups of actions that are affected by a Select action. For each Select action there must be one, and only one, End-Select action. See the previous description of the Select action.

Prog
Goto-Tag

Goto Tag *STRING?*

Allows you to have an action other than the succeeding one execute next. See the section, "How Running Works", for a detailed description of Key/Events and Tags.

Note that the string must be a constant, not a variable.

The Key/Event "Tag", with a name which is the same as the string, is searched for normally. (The search begins from the top of the displayed slides' Run Action list. If it is not there, then the search continues on the Global Run Action list. If a match is not found there, too, it is an error. You can also use the Use action). Execution then resumes with the action following that Tag action.

This action is similar to, though not the same as, a GOTO statement in some other languages. It lets you label an action, and then transfer to that action from one or more places. Note that this all occurs while one slide is on the screen. This does not change the slide being viewed.

The feature of searching the Global actions (and other slides' action lists with Use), lets you share common code among many slides.

Action lines with a Tag Key/Event label have no other action. They just have the one to eight character name of the Tag. You create Tag labels by Inserting a Key/Event-labeled action line in the normal manner, i.e., Insert to make a new one, Run Key/Event to change the current line's Tag. Then select the Tag command. Type the name of the tag in response to the prompt, and then press Enter.

To be a match, Tag names must be exactly the same as the string being searched for. It is case-sensitive.

Note

The Key/Events "Any Tag" and "Anything!" match all tag names. You can have an action that has a Key/Event label of "Any Tag" and an action of Use Actions On Slide.

Example

One application of the Goto Tag action could be to allow several keys to do some specific processing or testing, and then branch to common code, such as indicating a user error. For example, the following code processes the user responses A and O if the variable notyet is not 1. Otherwise, it sounds an error indication, remembers that an error has occurred, and views a slide with an error message:

```
Aa        If (notyet = 1)
             Goto Tag "early"
          End-If
          View Slide Adjust Menu [0205]
Oo        If (notyet = 1)
             Goto Tag "early"
          End-If
          View Slide Options Menu [0349]
Tag: "early"
          Tone Beep
          had_err = 1
          View Slide Early Error [0192]
```

Prog
Tag-Call

Call Tag *STRING?*

Similar to the Goto Tag action, with the added feature of allowing you to return to the next action when a Return From Tag Call action is executed. This is similar to the Call Subroutine operation in many computer languages.

Call Tag actions can be nested (along with loops, Ifs, Call Key/Events, etc.) to a depth determined by the Run Stack (see "How Running Works").

Prog
Return-from-Tag

Return From Tag Call

Returns execution to the action line immediately following the corresponding Call Tag action.

Prog
...

...

Continues execution with the next action line, even if it has a Key/Event label.

Normally, sequential execution of actions stops when a line with a Key/Event label on the left is encountered, and new keyboard input is read and processed. This command inhibits that from happening, i.e., the sequential execution continues.

This command is useful when you want to do the same processing for more than one Key/Event.

Example

The A, B, and C keys all display an error indication, although A only does it in "novice" mode:

```
Aa          If (novice = 0)
               View Slide Advanced A [0303]
            End-If
            ...
Bb          ...
Cc          Tone Beep
            View Slide ABC Error [0499]
```

String Run Actions

These run actions are for manipulating string values and string variables.

String Set

*STRING-VARIABLE?***[***POSITION?***]** = *VALUE?*

Assigns the value to a specified position in the string. The position is a value from 1 (the first position in the string) through the current length of the string. This operation does not extend the string — assigning to a position that does not exist is ignored. If the value is a number >255 or <0, a 0 is used.

Example

If str1 had the value "ABCDE", the action

```
          str1[3] = "x"
```

would change str1 to "ABxDE".

String Get

VARIABLE? = *STRING?***[***POSITION?***]**

Assigns the current value of the specified position in the string to the variable. The string can be a constant or a variable. The position must be a value from 1 through the current length of the string; otherwise a value of zero is produced.

If the variable is numeric, it is set to a number between 0 and 255. If it is a string variable, then it is set to be a string with a single character having the value.

To extract more than one character from a string, use the Extract run action.

Example

If n had the value 2, the action

```
ch = "AEIOU"[n]
```

would set ch to "E".

String Append

Append *VALUE?* **To** *STRING-VARIABLE?*

Adds one or more characters to the end of a string variable's current value. If the value is a string, the string variable is extended in length and the characters are added to the end. If the value is numeric, it is treated as a single character string, and that one character is added to the end of the string. Characters, which extend the string variable past a length of 80 characters, are ignored.

Example

If str1 had the value "Now is" and str2 had the value " the time", the action

```
Append str2 To str1
```

would set str1 to "Now is the time".

String Backspace

Backspace *STRING-VARIABLE?*

Decreases the string variable's length by one if greater than 0, and the last character in the string is removed.

Example

If str1 had the value "12345", the action

```
Backspace str1
```

would set str1 to "1234".

String Delete

Delete *COUNT?* **At** *STRING?*[*POSITION?*]

Removes the number of characters specified by the count from the string variable, starting at the indicated position. The string is shortened by that amount, and the length is adjust accordingly.

Example

If str1 had the value "The real value", the action

```
Delete 5 At str1[4]
```

would set str1 to "The value".

String Insert

Insert *VALUE?* **Before** *STRING-VARIABLE?*[*POSITION?*]

Adds one or more characters into a string variable's current value. If the value is a string, the string variable is extended in length and the characters are added starting at the specified position. This pushes the existing characters to the right. If the value is numeric, it is treated as a single character string and that one character is added.

Characters to be inserted, which extend the string variable past a length of 80 characters, are ignored.

Example

If str1 had the value "The value", the action

```
Insert " real" Before str1[4]
```

would set str1 to "The real value".

String Overwrite

Overwrite *STRING-VARIABLE?***[***POSITION?***] With** *STRING?*

Replaces characters in the string variable with the characters in the string value, starting at the specified position. If the value is a numeric value instead of a string value, it is treated as a single character string; just one character position is changed. If more characters are being replaced than currently exist in the string variable, the length is extended, up to a maximum of 80. (Extra characters are ignored.) You cannot start overwriting past the last existing character.

Example

If str1 had the value "The real value", the action

```
Overwrite str1[5] With "fake"
```

would change str1 to "The fake value".

String Fill

Fill *STRING-VARIABLE?* **With** *COUNT?* **of** *VALUE?*

The current value of the string variable is replaced by a string consisting of the specified number of the value. The value may be a string (the first character is used) or a number (the character number if not <0 or >255; zero otherwise); the count must be between 0 and 80.

Example

If str1 had the value "Testing", the action

```
Fill str1 With 4 of "x"
```

would change str1 to "xxxx".

Note

The Fill action is very useful for initializing strings to 80 blanks, etc., when you need fixed-sized lines for saving to a file or other purposes. You could use an = action with a constant of 80 blanks, but the Fill action might be more readable. Since identical constants are only stored once, there may or may not be an advantage to using Fill for saving space. (No matter how many constant strings you had that were 80 blanks, it would only take up about 112 bytes.)

String Erase

Erase *STRING-VARIABLE?*

Sets the value of the string variable to be a zero-length string.

Note

When string variables are first created, they are given the value of no-characters (length is 0). It is always good practice, though, to explicitly assign initial values to variables. Otherwise, when editing, you may get unwanted results the second time you run a series of slides. The Erase action is useful to do this initialization.

String Length

VARIABLE? = **Length(***STRING?***)**

Assigns the variable the current length of the string value. It will be a number between 0 and 80, inclusive. The length of a number is 1.

Example

If str1 had the value "ABCDEF", the action

```
            n = Length(str1)
```

would set n to 6.

String Process

Process-Char *STRING-VAR?*

Processes the last key typed, and the appropriate action is taken to "type in" to the string variable. If the key was a character (letters, numbers, punctuation, etc.), it is appended to the string. If the key was a backspace, the last character is removed from the string. All other keys are ignored. If there is not room to insert a character, or if there are no characters to erase, a Tone Thud action is automatically executed. This causes a Thud sound at the next screen redisplay.

The Grey +, Grey -, and Grey * keys are converted into their normal key equivalents. This changes the remembered last key to +, -, and *, respectively.

This action is used along with a Value Overlay of a string variable on the current slide to simulate the user "typing in" to a field, etc.

Note

This action uses the "last key typed" value (see the Set Builtin(1) action on the Miscellaneous Run Action menu). This is not necessarily the value of the Key/Event last matched, since a Transfer, Goto Tag, or Call Tag action may have been executed. You can change the value of the "last key typed" with the Set Builtin(1) action.

Example

If the current slide has the string variable "buffer" as a Value Overlay, you can use the following actions to let the user add to its value, remove characters with backspace, and view the next slide upon pressing Enter:

```
  Enter      View Slide >NEXT<
  Any Key    Process-Char buffer
```

Note that the action with the Key/Event label "Enter" appears before the Any Key action. This is necessary because the actions are searched in order from top to bottom for a match, and Any Key would match Enter. You should always put the more specific labels first.

To clear the variable "buffer" before the slide is displayed, you could add the actions:

```
  Viewed     Erase buffer
             Continue with Default Action
```

String Convert

Convert *VALUE?* To *VARIABLE?*

Converts the value to the type of the variable and stores it in the variable. This is used to convert from numbers to strings, and from strings to numbers.

If the variable is a string, the value is interpreted as a signed integer and converted to a string. This string value is stored in the string and replaces whatever value was there previously.

If the variable is numeric, the value is interpreted as a string and converted to a number. Characters other than 0-9 and the minus sign are ignored. Only numbers between -32768 and 32767 are converted correctly, since variables can only hold a 16-bit signed value.

Example

If n was a numeric variable, the action

```
Convert "4321" To n
```

would set n to 4321.

If str1 was a string variable, the action

```
Convert -2,791 To str1
```

would set str1 to "-2791".

String Where

VARIABLE? = Where In *STRING?* Is *VALUE?*

Sets the variable to the first position in the string that has the value. The value can be a number which is interpreted as a single character string, or a string. If the value is a string with a length greater than 1, the position returned is the first position of the match. If there is no match, the variable is set to zero. The comparison is uppercase/lowercase sensitive.

Example

The action

```
n = Where In "This is a test" Is "is"
```

would set n to 3.

String Trim

Trim *STRING-VARIABLE?*

Removes the leading and trailing space characters from the current value of the string variable.

Example

If str1 had the value "   This is centered   ", the action

```
Trim str1
```

would set str1 to "This is centered".

String Extract

Extract *STRING-VARIABLE?* **From** *START?* **To** *END?*

The current value of the string variable will have all but the characters from the start position to the end position removed.

Example

If str1 had the value "ABCDEFGH", the action

```
Extract str1 from 3 to 5
```

would set str1 to "CDE".

String Replace

Replace *STRING?* **With** *STRING?* **In** *STRING-VARIABLE?*

The characters in the string variable, which also occur in the first string, are replaced by the corresponding characters (those in the same position) in the second string. Programmers often call this a translate operation.

This action can be used to convert a string to all uppercase or lowercase, to convert all punctuation to spaces to ease parsing, and so on.

Example

If str1 had the value "Fewor Spelling Errors", the action:

```
Replace "aeiouAEIOU" With "aaaaaAAAAA" In str1
```

would set str1 to "Fawar Spallang Arrars".

Miscellaneous Run Actions

These run actions are very specific to DEMO II. They access internal values, and trigger specific operations.

Misc Key

VARIABLE? **= Current Key/Event**

The value of the last Key/Event which matched a Key/Event label is assigned to the variable. Note that this is not necessarily the last character typed, since there may have been an intervening Goto Tag, Call Tag, or Transfer action.

Example

If you pressed the "x", the action

```
Any Letter  n = Current Key/Event
```

would set n to 120 ('x').

Misc
Current

VARIABLE? = Current Slide Number

Assigns the number of the slide currently being viewed to the variable. This value can be used in subsequent View Absolute or View Slide In File actions.

Misc
Flush

Flush Type-Ahead

Discards and ignores any keys that the user has typed but that have not yet been processed (i.e., keys "typed ahead").

Normally, you want the user to be able to type and have all keys remembered as the computer "catches up" with the typing. There are times, though, when you want to discard any keys that are typed "too early." An example would be after a user error was detected, or after a long pause when the user may have gotten impatient and pressed "Space to continue" too many times.

Note that slides with Run Type 2 automatically flush type-ahead before waiting for keyboard input.

Misc
Redisplay

Redisplay Screen

Displays the slide currently being viewed on the screen along with any changes that may have occurred because of variable-value modifications. Also, any sounds that wait for Redisplay are sounded immediately after the screen is updated. Bitmapped screens are not updated, although the sounds are made. (This is an optimization since bitmapped screens often take a relatively long time to display, and may require a disk access.)

The screen is automatically redisplayed whenever a slide is viewed and whenever new input is requested after run actions are executed. The screen is also redisplayed after every step in Debug running. If the screen updates correctly in Debug, but not normally, you may need Redisplay actions in the middle of your loops, etc.

See the section, "How Running Works", for a more complete description of redisplaying.

Example

If str was a string variable used as a Value Overlay on the current slide, and the following actions were executed:

```
Erase str
For n = 1 To 10
   Pause 9 1/8-Seconds
   Append "x" To str
   Redisplay Screen
End-For
```

you would see first one "x", then "xx", then "xxx", etc., increasing every half second. If the Redisplay action were removed, no changes would appear on the screen until the next redisplay, and then "xxxxxxxxxx" would show.

Misc
Pause

Pause *VALUE?* 1/18-Seconds

Execution pauses for the specified amount of time. As soon as there is any type-ahead, the pause terminates. The system clock is used for the timing, so the pause is independent of CPU speed. The pause is actually about 18.2 units per second.

Misc
Long-Pause

Pause *VALUE?* Even If Key Pressed

Execution pauses for the specified amount of time, regardless of any type-ahead. The system clock is used for the timing, so the pause is independent of CPU speed. The pause is actually about 18.2 units per second.

Misc
NextKey

Next Key

Reads the next keypress, and then processes in the normal way, looking for a matching Key/Event label. It is similar to what happens when there are no more actions to process, or an action with a Key/Event label is encountered. (The full operation is to redisplay the screen, then signal the Readkey Key/Event, which by default inputs a key and then processes it.)

Misc
Input

VARIABLE? = Input A Key

Reads the next keypress from the keyboard. (If there is any type-ahead, it is used first.) The value is assigned to the variable, which should be numeric since keypresses can have values greater than 255. The value is also assigned to the "last key pressed" value used by the Process-Char action.

Example

The following actions sound a tone, and then wait for the user to press a key to continue. The key is ignored.

```
Melody "602"
Flush Type-Ahead
n = Input A Key
```

Misc
Exec

ERRORCODE? = Exec *FILENAME?* With *ARGUMENT-STRING?*

Lets you execute other programs running under DOS and then return to this one. Its operation must be fully understood in order to have it work correctly.

There must be enough room in memory, in addition to DEMO II, to run the requested program. Normally DEMO II uses all memory, so you must restrict the amount used by having the "-MEM" option on the command line when you execute DEMO II. The "-MEM" option is followed by a space and then the number of Kbytes of memory to use for slides, etc. The rest is reserved for the Exec action. The amount of memory to specify should be at least enough to load your largest file. (See the Global menu for an indication of how much memory a given file uses.) The amount left for the Exec action must be enough to hold a new COMMAND.COM environment and invoke the requested program. If there is not enough memory, a suitable code is returned in the errorcode variable.

Failure to use the "-MEM" command option is the most common error when using this action.

The file name is the name of the file to be executed. It can be a full path (starting with optional drive, then "\", etc.), or just a file name. If it has an extension (such as ".EXE"), then the file with that extension is searched for. If there is no "." or extension, first the file with no extension is searched for, and then the name with ".EXE". The search follows the normal DOS conventions, including using the PATH setting.

The argument string is passed to the new program as its argument.

The screen is not changed, so the new program should be aware of that, if necessary. If it changes the screen modes, it should set them back when done.

Upon return, the errorcode variable is assigned the following:

```
-1 - File or pathname not found
-2 - File is not executable
-3 - Not enough memory to execute
-4 - Microsoft C codes E2BIG or EINVAL (rare)
```

The code will be 0 if all is OK, and a positive exit code value if the program has an exit code. (Exit codes are used by ERRORLEVEL in batch files). The "rare" -4 error should never occur — it means a DEMO II system error has occurred.

If the file name is "*" (a single character string consisting of an asterisk), the argument string is passed to COMMAND.COM to execute. This allows you to execute internal DOS commands such as COPY, batch files, etc. Note that exit codes are not returned — successful execution always results in 0.

The Microsoft C library routines SPAWNLP and SYSTEM are used to implement this action.

Example

To print a file on the printer, you could execute the actions

```
status = Exec "*" With "copy file.txt prn:"
If (status != 0)
   View Slide Exec Error [0015]
End-If
```

To get a DOS shell, you could use

```
status = Exec "command.com" With ""
```

and return to DEMO II with the DOS Exit command.

Note

If you are calling a program with no arguments, you need to have a null ("") second argument to the Exec action. You cannot leave it undefined. You get a null argument by pressing the " key and then pressing Enter or Tab.

Misc
Mouse

Mouse (*COMMAND?*, *VALUE?*, *VALUE?*)

Allows you to make use of a compatible mouse or other pointing device while running. It implements a large subset of the Microsoft Mouse Function Calls. The full calls are defined in the Microsoft Mouse Programmer's Reference Guide (from Microsoft Corporation, Document Number 990973002-600-R00-1186), as well as in IBM's Mouse Technical Reference (68X2229 S68X2229-00). You should be familiar with those calls before attempting to use the Mouse actions. ***This is the only DEMO II use of a mouse.***

The Mouse action forms are:

`Mouse (0, arg2, arg3)`

Mouse reset and status (Function 0). Arg2 is set to 0 if no mouse hardware and software are found, -1 if it is. Arg3 is set to the number of buttons on the mouse.

`Mouse (1, arg2, arg3)`

Show Cursor (Function 1). Increments the internal mouse cursor flag. See the Mouse documentation. Note that the cursor is always automatically hidden and then reshown during a redisplay. This can make a very blinky cursor. It is often better to interrogate the mouse position and move an overlay or a H/W cursor overlay by changing a variable. If you are staying on one screen during the processing, and not redisplaying (such as a bitmapped image), there should not be so much of a problem.

`Mouse (2, arg2, arg3)`

Hide Cursor (Function 2). Decrements cursor flag. See Show Cursor, above.

`Mouse (3, arg2, arg3)`

Button Status A (Function 3). Sets arg2 to the state of the two mouse buttons. Bit 0 is set to 1 if the left button is down, 0 if up. Bit 1 is set to 1 if the right button is down, 0 if it is up. For example, 1 means that the left button is currently down, 2 means just the right button is down, and 3 means they are both being pressed.

`Mouse (-3, arg2, arg3)`

Button Status B (Function 3). Sets arg2 to the current mouse column position (horizontal position in units — usually pixels). Sets arg3 to the current mouse row position (vertical).

`Mouse (4, arg2, arg3)`

Set Mouse Cursor Position (Function 4). Arg2 and Arg3 are the horizontal and vertical coordinates, respectively.

`Mouse (5, arg2, arg3)`

Get Button Press Information A (Function 5). Arg2 specifies which button to check. If it is 0, the left button is checked; if 1, the right button is checked. Upon return, arg2 is set to the current status of the buttons (same as Button Status A, above). Arg3 is set to the number of button presses since the last such call.

`Mouse (-5, arg2, arg3)`

Get Button Press Information B (Function 5). Returns additional values associated with the last Mouse Get Button Press Information A call (see above). Arg2 is set to the horizontal coordinate of the last press of the specified button. Arg3 is set to the vertical position.

```
Mouse (6, arg2, arg3)
Mouse (-6, arg2, arg3)
```

Get Button Release Information A & B (Function 6). These are the analogous calls for button release to the Button Press calls, above.

```
Mouse (7, arg2, arg3)
```

Set Min & Max Horizontal Cursor Position (Function 7). Arg2 specifies the mininum position, and arg3 the maximum.

```
Mouse (8, arg2, arg3)
```

Set Min & Max Vertical Cursor Position (Function 8). Arg2 specifies the minimum position, and arg3 the maximum.

```
Mouse (9, arg2, arg3)
```

Set Graphics Cursor Block (Function 9). Arg2 is a specification for the cursor hot spot. It is calculated by: (horizontal * 100) + vertical. Arg3 is a string with the appropriate bytes. A good way to set the arg3 value, since it may have many nonprintable values, is to read it in from a file with a #Read action if you do not want lots of assignment statements.

```
Mouse (10, arg2, arg3)
```

Set Text Cursor Software (Function 10). Arg2 and arg3 specify the screen mask and the cursor mask.

```
Mouse (-10, arg2, arg3)
```

Set Text Cursor Hardware (Function 10). Arg2 and arg3 specify the scan line start/stop values.

```
Mouse (11, arg2, arg3)
```

Read Mouse Motion Counters (Function 11). Arg2 is set to the horizontal mickey count since the last such call. Arg3 is set to the vertical mickey count. A mickey is 1/200 of an inch.

```
Mouse (15, arg2, arg3)
```

Set Mickey/Pixel Ratio (Function 15). Arg2 is the horizontal mickey/pixel ratio, and arg3 is the veritcal ratio.

```
Mouse (19, arg2, arg3)
```

Set Double-Speed Threshold (Function 19). Arg3 (not arg2) is the threshold speed in mickeys/second.

All other command values are ignored.

Misc
Attrib-
Translate

Translate Attribute *STRING?* To *STRING?*

Sets elements of the Attribute Translate Table. This is the same table used by the Block Xlate command. The table specifies how to translate each of the 256 attributes. This action sets the table element specified by each position in the first string to the value in the same position in the second string. The actual translation is done by using the Translate Attribute Cmd action.

Misc Translate-Cmds

Translate Attribute Cmd *VALUE?*

Controls the attribute translation process. The translation allows you to change attributes dynamically before they are displayed on the screen. This lets you show the same set of slides on a variety of monitors and adapters, with color showing as color, and an appropriate translation into black and white.

The values accepted are as follows:

`Translate Attribute Cmd 0`

Turns off translation. Attributes are displayed as themselves.

`Translate Attribute Cmd 1`

Turns on translation. All text screens have their attributes translated by the Attribute Translate Table before being displayed. This has no effect on the actual slides, only the displayed image is changed. To make the translation permanent, use the Block Xlate command.

`Translate Attribute Cmd 2`

Resets the Attribute Translate Table to do no translation. That is, all attributes will be translated into themselves.

`Translate Attribute Cmd 3`

Sets the Attribute Translate Table to do default, monochrome translation. All attributes will be translated into normal video (07) except inverse (70), which stays the same. You may want to set other items explicitly after using this action, such as making some color combinations go to underline or bright (if the monitor/display adapter combination has them).

Note

You should determine the translation that is needed by either asking the user ("Is this in color on your screen?", "Is this underlined?") or by checking the adapter type with the Get Builtin(5) action and guessing what would be appropriate.

Example

The following actions could be used in response to the question "Is this in color on your screen (Y/N)?". They would do a mono translation that would turn all attributes into normal except inverse (70), which would stay inverse, and 41, 47, and 76, which would be turned into underline (01):

```
1  Nn       Translate Attribute Cmd 3   - set to mono xlate
2           Fill str1 With 3 of 0       - what to translate
3           str1[1] = 65                - blue/red (hex 41)
4           str1[2] = 71                - white/red (hex 47)
5           str1[3] = 118               - brown/white (hex 76)
6           Fill str2 With 3 of 1       - into underlined (01)
7           Translate Attribute str1 To str2   - add new 3
8           Translate Attribute Cmd 1   - turn on translation
9           ...                         - join color code...
10 Yy       View Slide >NEXT<
```

Note that if you just wanted the normal color-to-mono translation, you could skip lines 2 through 7, leaving you with:

```
1  Nn       Translate Attribute Cmd 3   - set to mono xlate
2           Translate Attribute Cmd 1   - turn on translation
3           ...                         - join color code...
4  Yy       View Slide >NEXT<
```

Misc Debug

Debug

Starts single-step Debug running. This is like setting a "breakpoint" in other programming systems. You can use this action to start debugging at a particular point or when a particular condition occurs, rather than at the start of running. Note that if you press Ctrl-Break while running, you will be given the option to enter Debug mode or stop running. This is another way to start debugging at a particular point.

Example

If you know that there are problems with your actions at a particular point in a slide show, but only when string str has a length of 5, you could add the following actions to start step-by-stepping at that point:

```
slen = Length(str)
If (slen = 5)
   Debug
End-If
```

Misc Get

VARIABLE? = Builtin(*VALUE?*)

Gets the value of a variety of internal settings. Many of them can be set with the Set Builtin action, below.

List of Builtin Options

```
0     Random Number
1     Last Typed
```

2	Elapsed Timer Ticks
3	Elapsed Seconds
4	Relative CPU Speed (10=IBM PC)
5	Display Adapter (0=Mono,1=CGA,2=EGA,3=Herc,5=VGA)
6	Background Attribute
7	Background Character
8	Timeout Value
9	BIOS Shift State (including Num Lock)
-1	Run Wait Value
-2	Run Type Value
-3	Day of Week (0=Sunday)
-4	Year (1987, etc.)
-5	Month
-6	Day of Month
-7	Hours
-8	Minutes
-9	Seconds
-10	Hundredth Seconds
-11	DOS Environment Setting DEMO2V
-12	Command Option -ARG Value
-13	Kbytes Free
-14	Kbytes Total

Full Descriptions

var = Builtin(0)

Gets the next random number, a value between 0 and 32767. To get a number between 0 and n, you use the remainder of the random number with n+1. For example, to get numbers from 0 to 10, you get the remainder with 11:

```
n = Builtin(0)
n = n % 11
```

var = Builtin(1)

Gets the Key/Event value of the key "last typed," which is the value used by the Process-Char action.

var = Builtin(2)

Gets the elapsed time in timer ticks (1193180/65536 = 18.206481 per second). The starting time is power-on of the machine unless you reset it with a Set Builtin action. Since only values from 0-32767 are returned (about 1800 seconds, or slightly under 30 minutes), you should reset the starting time first, and only use it to time short intervals.

var = Builtin(3)

Gets the elapsed time in seconds. The starting time is power-on of the machine unless you reset it with a Set Builtin action. Since only values from 0-32767 are returned (a little over 9 hours), you should reset the starting time first. While most time conversions in this program use 18 clock ticks per second as an approximation, this operation is much more accurate; it should be almost identical to the system clock, since it uses 18.206481.

`var = Builtin(4)`

Gets the relative CPU speed calculated when the program started. This value is used by some of the timing routines, such as the Sound action and some of the Switch display operations. A normal, original, IBM PC has a value of 10. (For those who care, the value is determined by timing a loop with: mov di,[si]; inc di; loop). This value is provided for those times when you need to know the characteristics of the machine you are on, so that you can do something faster on a faster machine and slower on a slower machine, or for whatever other purpose you may want.

`var = Builtin(5)`

Gets the display adapter type. You may need this to translate attributes automatically on a monochrome system (using the Translate Attribute action), or to show the most appropriate bitmapped graphics (higher-res on an EGA, lower on a CGA, and Hercules on a Hercules card). The values are as follows:

```
        0 - Monochrome Display Adapter
        1 - Color Graphics Adapter (CGA)
        2 - Enhanced Graphics Adapter (EGA)
        3 - Hercules Graphics Card (Hercules)
        4 - reserved, treated like CGA
        5 - VGA, treated like EGA
```

`var = Builtin(6)`

Sets the variable to the current background attribute value. This is the value shown on the Global menu as Background Attribute.

`var = Builtin(7)`

Gets the current background character value.

`var = Builtin(8)`

Gets the current timeout value (the value shown on the Global menu).

`var = Builtin(9)`

Sets the variable to the current shift state (BIOS location 40:17). The bits are as follows:

```
        Bit 0 - Right shift key currently pressed (1)
        Bit 1 - Left shift key currently pressed (2)
        Bit 2 - Ctrl key currently pressed (4)
        Bit 3 - Alt key currently pressed (8)
        Bit 4 - Scroll Lock locked (16)
        Bit 5 - Num Lock locked (32)
        Bit 6 - Caps Lock locked (64)
        Bit 7 - Insert locked (128)
```

You can use the computer-type AND and OR actions to manipulate these values, if you know how. See the example in Set Builtin(9).

`var = Builtin(-1)`

Gets the Run Wait value for the currently viewed slide.

`var = Builtin(-2)`

Gets the Run Type value for the currently viewed slide.

`var = Builtin(-3)`

Gets the day of the week, as set in the system: 0=Sunday, 1=Monday, etc.

`var = Builtin(-4)`

Gets the year, as set in the system, e.g., 1987.

`var = Builtin(-5)`

Gets the month, as set in the system. 1=January, 2=February, etc.

`var = Builtin(-6)`

Gets the day of the month, as set in the system.

`var = Builtin(-7)`

Gets the hour of the day, as set in the system.

`var = Builtin(-8)`

Gets the current minute of the hour, as set in the system.

`var = Builtin(-9)`

Gets the current second of the minute, as set in the system.

`var = Builtin(-10)`

Gets the current hundreths of a second, as set in the system.

`var = Builtin(-11)`

Sets var (a string variable!) to the current value of the DOS environment variable "DEMO2V", up to 80 characters.

`var = Builtin(-12)`

Sets var (a string variable!) to the value provided after the "-ARG" option to the DEMO II command. For example, "D2 test -arg convert" sets the variable to "convert". This lets you pass arguments to a slide show from command level or a batch file.

`var = Builtin(-13)`

Gets the number of KBytes free (the same as the KBytes left value on the Global menu).

`var = Builtin(-14)`

Gets the number of KBytes total available to load a file.

Misc
Set

Builtin(*VALUE?*) = *VALUE?*

Sets the value of a variety of internal settings. They can all be read with the Get Builtin action, described above. See also the list of Builtin options there.

An additional Set Builtin (number 10) simulates spinning the diskette on drive A. It is provided for compatibility with **Dan Bricklin's Demo Program**, and may help to make a demonstration more realistic. You can sometimes get the same effect, or better, with the File operations.

`Builtin(0) = value`

Resets the random number generator. A value of 1 resets the set of numbers to the "standard" starting point. Any other value sets it to a different starting point. You can use the Builtin(2) value (the elapsed time in timer ticks) to get a "random" starting value to set Builtin(0).

`Builtin(1) = value`

Sets the "last typed" value, which is used by the Process-Char action.

`Builtin(2) = value`

`Builtin(3) = value`

Both reset the elapsed-time counter to 0.

`Builtin(4) = value`

Sets the Relative CPU speed value. The value must be greater than 0 or else it is ignored. There may be a good reason to set this, but I can't think of too many. It's setable for completeness.

`Builtin(5) = value`

Sets the display adapter that the system thinks it has when it determines what type of bitmapped images are allowed. Setting this can have unpredictable results.

`Builtin(6) = value`

Sets the background attribute. Takes effect the next time the screen is displayed.

`Builtin(7) = value`

Sets the background character. Takes effect the next time the screen is displayed.

`Builtin(8) = value`

Sets the Timeout value, which is the Timeout During Run value shown on the Global menu. It determines how much time to wait for user input before producing the pseudo-key "Timeout". This value must be greater than or equal to 0. Zero means that no checking for timeout will occur.

`Builtin(9) = value`

Sets the BIOS shift-state information (BIOS location 40:17). See the Get Builtin (9) description for a list of the bit assignments. You can use this action to force the Num Lock and Caps Lock bits to be off. (The lights will be turned off on ATs and PS/2s, etc.).

For example, to turn off the Num Lock, you could use the actions:

```
n = Builtin(9)
n = n & 223
Builtin(9) = n
```

The value 223 is the same as hex DF, which is all bits on, except the Num Lock bit 5. You can calculate it by taking 255 and subtracting the value of the bit you want to turn off. Bit 0 is 1, bit 1 is 2, etc.

You can force the bits on by using the OR action. To turn on the Caps Lock bit, you could use the actions:

```
n = Builtin(9)
n = n | 64
Builtin(9) = n
```

Builtin(10) = value

Spins the A diskette drive. The value is ignored. This works even if a diskette is not in the drive. For those readers who are technically minded, it does a ROM-BIOS Verify Diskette operation (int 13H, AH=4) of drive 0, head 0, sector 1, track 0; then another one of sector 1, track 10. If the BIOS returns an error, a reset (AH=0) is done. The interrupt-enable bit in the CPU is set (STI instruction) after each BIOS call to ensure compatibility with some computers from Compaq.

File Run Actions

These run actions control file and printer I/O.

File Open

ERRORCODE? = **Open File** *FILENAME?* **For** *MODE:R/W/U/A?*

Opens a normal DOS file. The file name is a string. The third argument is the mode in which you want to operate on the file. The mode must be a numeric or string value with the value of a single character. The accepted values are R for reading the file; W for writing the file (truncating it first, if it already exists); U for updating the file in place; and A to write starting at the current end of the file (appending). The error code is returned to set the first argument.

The codes returned are:

```
0 - OK
1 - Access violation (write a directory, etc.)
2 - File or pathname not found
3 - Microsoft C codes EEXIST or EMFILE (rare, mean
    system error)
```

Example

To open the file "TEMP.LOG" and add to the end, if it already exists, you could use the actions:

```
status = Open File "temp.log" For "A"
If (status != 0)
   View Slide Open Error [0039]
End-If
```

File Close

Close File

Closes the currently open file. Only one file may be open at any given time. In order to open a second file, you must close the first file before you do the second Open action.

Note

You must close files when you are finished with them. This writes out any last bit of information, and keeps the disk consistent. Turning off power to the computer when a file is open can have an unpredictable effect on the file system.

File
Read

ERRORCODE? = **Read Text Into** *STRING-VARIABLE?*

Reads the next line of text from the currently open file and puts it into the string variable. The error code variable is set to 0 if all is OK, to 1 if an End-of-File has been encountered (no characters read), and to 2 if the file was not open for read or was locked.

A "line" is terminated by a CR (carriage return character, Hex 0D), or a CR/LF (carriage return/line feed combination, Hex 0D 0A). If a line has more than 80 characters, it is broken up into chunks with a maximum of 80. Subsequent Read Text actions retrieve the subsequent characters.

File
Read#

ERRORCODE? = **Read** *COUNT?* **Bytes Into** *STRING-VAR?*

Reads the next specified number of bytes into the string variable from the currently open file. The error code variable is set to 0 if all is OK, to 1 if an End-of-File has been encountered (no characters read), and to 2 if the file was not open for read or was locked.

All characters are treated the same, including CR and LF.

Note

You use this type of Read action when you have a specific number of characters to read from a file. Frequently there are fixed-length "records". You use the Seek action to position the file at the appropriate byte, and then read in the required number of characters.

Use the Read action instead of the Read# action when you are reading a file that is made up of a series of "text lines," each delimited by a CR or CR/LF.

File
Write

ERRORCODE? = **Write** *STRING?*

Writes the characters in the string value to the currently open file, adding a CR/LF at the end. The error code value is set to 0 if the write was successful, to 1 if the file is read only or locked, to 2 if there is no room left on the output device (disk full), and to 3 in the rare event (DEMO II system error) that Microsoft C error code EBADF is returned.

File
*Write-
NCRLF

ERRORCODE? = **Write** *STRING?* **With No CR/LF**

Writes the characters in the string value to the currently open file. No CR/LF is added at the end. The error code value is set to 0 if the write was successful, to 1 if the file is read only or locked, to 2 if there is no room left on the output device (disk full), and to

3 in the rare event (DEMO II system error) that Microsoft C error code EBADF is returned.

You can use this action, along with the plain Write action, to create lines longer than 80 characters. This is especially useful for wide reports for later printing. You can also use this action to write out the parts of a line one at a time, with only the last part having a plain Write with CR/LF. It may be better, though, to build up a line by appending to a string. Then output the entire string as a line when done.

File Seek

ERRORCODE? = **Seek To** *VALUE?*

Moves the "current position" in the currently open file to the specified byte (0-32767, with 0 being the first character). Do not use a value out of this range. The error code variable is set to the value 0 if the seek is successful, and to non-zero otherwise. If a file is opened for Update ("U"), you can read a series of bytes (usually with the Read# action), and then write them back out (usually with *WriteNCRLF) after using the Seek action to move back to the same position.

File Print

Print *STRING?*

Outputs the characters in the string value to the printer, adding a CR/LF at the end. Note that it outputs to the standard print device known as "stdprn" in the DOS technical reference manuals.

File &Print-NCRLF

Print *STRING?* **With No CR/LF**

Outputs the characters in the string value to the printer, with no extra CR/LF added at the end. Note that it outputs to the standard print device known as "stdprn" in the DOS technical reference manuals.

You can use this action, along with the plain Print action, to create lines longer than 80 characters. This is especially useful for printing wide reports. You can also use this action to print the parts of a line one at a time, with only the last part having a plain Print with CR/LF. It may be better, though, to build up a line by appending to a string. Then output the entire string as a line when done.

Command List

This is a list of all of the commands in DEMO II, listed hierarchically, with the prompts that appear below them in the command windows.

BLOCK			Commands to manipulate blocks of text
	ATTRIB		Set block (or one character, if no block marked) to attribute
		Select	Select attribute
		Insert	Make new attribute definition
		Delete	Remove attribute definition
		Move	Reposition attribute in list
		Name	Edit attribute name
		Value	Edit hexadecimal value
		All	Insert ALL 255 attributes
		Group	Toggle group start
	BEGIN		Mark the beginning of a block at the cursor
	LAST		Restore the last block definition
	UNMARK		Stop showing a block definition
	DELETE		Save a copy of marked block, then delete it from screen
	COPY		Save a copy of marked block without deleting it
	SAVE		Save a copy of marked block to a named retrieve area
		Select	Save to highlit save area
		Insert	Make new save area
		Delete	Delete save area
		Move	Reposition save area in list
		Name	Edit save area name
	PASTE		Retrieve deleted/copied block at cursor, then confirm position
	RETRIEVE		Retrieve a named retrieve area at cursor, then confirm position
		Select	Retrieve highlit save area
		Insert	Make new save area
		Delete	Delete save area
		Move	Reposition save area in list
		Name	Edit save area name
	MOVE		Use cursor to move marked block (like cut/paste)
	WRAP		Word-wrap characters in block (until terminated by unmarking)
	EXCHG		Swap contents of delete/paste with block
	XLATE		Translate selected attributes in block or on all slides
		Value	Set value to translate to
		Mono	Set all colors to mono
		Reset	Set all attribs to themselves
		Group	Toggle group start
		#	Go to specified item
		Block	Translate attribs in block
		Slide	Translate attribs on slide
		All	Translate ALL SLIDES

FILL	Fill block with contents of upper-left position
1-BOX	Surround block with single-line box
2=BOX	Surround block with double-line box
3_BOX	Surround block with thick box
4=NOBOX	Remove box around block
CNTR	Center lines within block
/CAB	Set what Delete, Paste, typing, and editing affect: Chars and/or Attribs
Char	Characters Only
Attrib	Attributes Only
Both	Both Chars & Attribs
NAMES	Display list of block names and definitions
Block	Make selected item current block
Comment	Edit block's comment
Position	Redefine item to current block
Name	Edit name of highlighted block
Insert	Insert current block as new item
#	Move to specific item
Group	Toggle start of group
Delete	Delete block name definition(s)
Move	Reposition item
TYPING	Aids for nontext typing: line drawing, special chars, etc.
LINES	Make cursor keys draw lines
CHARS	Choose special character from list
Select	Choose Char
Num	Input Decimal #
DIRECTION	Change typing direction to up, down, left, or right
Right	
Left	
Up	
Down	
HTABS	Set/Clear horizontal tab stops
Set	Set tab stop at cursor
Clear	Clear tab stop at cursor
All	Clear all tab stops
Default	Clear and set to defaults
VTABS	Set/Clear vertical tab stops
Set	Set tab stop at cursor
Clear	Clear tab stop at cursor
All	Clear all tab stops
Default	Clear and set to defaults
MARGIN	Set column, where cursor goes, when Enter is pressed
STATUS	Configure Status Display
Hide	Toggle whether on not to show Status Display
Cursor	Toggle cursor position display
Slide	Toggle slide name/number display

		File		Toggle file name display
		Ins/Ovr		Toggle Insert/Overwrite display
	Find			Move cursor to location with specified characters
SLIDES				View, create, delete, etc., other slides
	VIEW			Show slide
	UNDO-EDIT			Restore slide as when last viewed
	INSERT			Create new slide after this one
	!DELETE			Erase slide(s). Type !, not D.
	PRINT			Toggle Print Flag (all Group like first)
	NAME			Edit name of slide
	OPTIONS			Show/Edit color/bitmap options
		Value		Set value of highlighted item
			Slide Type	
			Switch Type	
			Switch Speed	
			Bitmap Origin/PCX Filename	
		Text		Make slide NOT bitmapped
		Palette		Show/Change video settings
		OK		Return to previous menu
	LOCATE			Find slide with given name
	#			Move to specific number slide
	GROUP			Toggle start of group
	MOVE			Change order of slides
COPY				Copy entire contents of slides
	ALL			Copy overlays, actions, and slide
	OVERLAYS			Copy overlays from slide only
	RUN			Copy run actions from slide only
	SLIDE			Copy slide without overlays only
	LOCATE			Find slide given name
OVERLAYS				Display slides one on top of another, variables on slides, etc.
	NUMS			Set values for row/col offsets, whether or not overlay is displayed, etc.
		Value		Set value of highlighted item
			Row Offset	
			Column Offset	
			Visible	
			Max Chars Shown	
			Type	
			Scan Lines Desc	
		OK		Return to previous menu
	SLIDE			Insert overlay showing another slide at the same time as the current slide
		View		Show slide to select and adjust
		Locate		Find slide with given name

This-Slide	Make reference to THIS SLIDE
VALUE	Insert overlay showing a variable or constant with the current slide
CURSOR	Insert overlay displaying the Hardware Cursor
ADJUST	Change position or item for existing overlay
GROUP	Toggle group start
#	Move highlight to specified overlay
MOVE	Change order (2nd displays after 1st, 3rd after 2nd, etc. Current slide last.)
DELETE	Erase overlay(s)
OK	Return to editing
PASTE	Paste overlay as shown, then remove overlay definition
RUN	Setup and automatically go from slide to slide
LINE	Insert a new blank line below the current line, then set action
ACTION	Set operation for this line
Programming	"Programming" actions such as IF, FOR and SELECT
If	Execute following actions if condition met, else skip to .ELSE or END-IF
=	"True" if first value equals second value (strings not case sensitive)
= =	"True" if first value equals second value (strings case sensitive)
<	Condition met ("True") if first numeric value less than second
<=	"True" if first numeric value less than or equal to second
!=	"True" if first value does not equal second (strings not case sensitive)
=	"True" if first numeric value greater than or equal to second
	Condition met ("True") if first numeric value greater than second
In	"True" if value (single character) is contained in string
Between	"True" if value is >= one value and <= the other
Upper	"True" if value contains only uppercase letters (A-Z)
Lower	"True" if value contains only lowercase letters (a-z)
Alpha	"True" if value contains only uppercase or lowercase letters
#Num	"True" if value contains only digits (0123456789)
NumLet	"True" if value contains only digits or letters
Func-Key	"True" if value is a function key Key/Event number
Edit-Key	"True" if value is an editing/cursor key Key/Event number
Text	"True" if value is a text character (0-9, A-Z, punctuation, etc.)
Key	"True" if value is Key/Event number of key user can press
WaitforKey	"True" if waiting for user to press key (no keys "typed ahead")
^Waiting	"True" if keys "typed ahead"
Shift	"True" if either shift keys are currently being pressed
Ctrl	"True" if Ctrl key is currently being pressed
***Alt**	"True" if Alt key is currently being pressed
-Else	Execute following actions if previous condition not met, else skip to END
For	Repeatedly execute a group of actions for a specified number of times
While	Repeatedly execute a group of actions while a condition is met
=	"True" if first value equals second value (strings not case sensitive)

= =	"True" if first value equals second value (strings case sensitive)
<	Condition met ("True") if first numeric value less than second
<=	"True" if first numeric value less than or equal to second
!=	"True" if first value does not equal second (strings not case sensitive)
=	"True" if first numeric value greater than or equal to second
	Condition met ("True") if first numeric value greater than second
In	"True" if value (single character) is contained in string
Between	"True" if value is >= one value and <= the other
Upper	"True" if value contains only uppercase letters (A-Z)
Lower	"True" if value contains only lowercase letters (a-z)
Alpha	"True" if value contains only uppercase or lowercase letters
#Num	"True" if value contains only digits (0123456789)
NumLet	"True" if value contains only digits or letters
Func-Key	"True" if value is a function key Key/Event number
Edit-Key	"True" if value is an editing/cursor key Key/Event number
Text	"True" if value is a text character (0-9, A-Z, punctuation, etc.)
Key	"True" if value is Key/Event number of key user can press
WaitforKey	"True" if waiting for user to press key (no keys "typed ahead")
^Waiting	"True" if keys "typed ahead"
Shift	"True" if either shift keys are currently being pressed
Ctrl	"True" if Ctrl key is currently being pressed
*Alt	"True" if Alt key is currently being pressed
Block	Repeatedly execute a group of actions (use Leave action to stop)
Leave	Stop executing actions in group and resume after END
Again	Continue executing FOR/WHILE/BLOCK from first action in group
Select	Skip actions until CASE action that matches specified value
Case	Specify value to match last SELECT action
Otherwise	Match last SELECT action no matter what the value
Done	Skip to END of Select group
End	Matching END to IF, FOR, WHILE, BLOCK, and SELECT
If-End	End to match most recent If
For-End	End of For group
While-End	End of While group
Block-End	End of Block group
Select-End	End of Select Case/Otherwise list
Goto-Tag	Continue executing with actions that follow specified Tag Key/Event
Tag-Call	Like Goto-Tag, but execution can be resumed with Return-from-Tag
Return-from-Tag	Resume execution with action following last Tag-Call
...	Continue executing, even if next line has a Key/Event on the left
View	Change display to show a specific other slide
View	Show slide to select
Locate	Find slide with given name
Next	Use next slide in sequence
Previous	Use previous slide in sequence
Tone	Make beep, thud, sound, single note, or melody
Beep	Make Beep sound at next redisplay of screen

Thud	Make Thud sound at next redisplay
Sound	Sound arbitrary sound (see Manual)
Note	Sound particular note for a specified amount of time
Melody	Play a series of notes listed in a string
Other-Slide	Other ways to change display to show another slide
Call	View specified slide, then redisplay this slide after Return action
Return	Redisplay slide that last did a slide Call
After	Display slide following slide that last did a slide Call
File	Load another file and start with default slide
Slide-File	Load another file and start with specified slide
Offset-View	Display a slide the specified number of slides from the current slide
View-Absolute	Display a slide with the specified number
Name	Set variable to number of first slide whose name matches specified string
Display-then-Tag	Display slide with specified number, then go to Tag on that slide
Quit	Stop running (and return to editing if not runtime version)
Nothing	Leave line blank (no action)
Key/Event	Continue executing actions with another Key/Event action list
Transfer	Continue executing with action list that matches specified Key/Event
Call-Key/Event	Like Transfer, but execution can be resumed with Return-from-Key/Event
Return-from-Key/Event	Resume execution with action following last Call-Key/Event
Use	Search for Key/Event match on specified slide's action list
Global	Search for Key/Event match on Global Run Action List
Default	Continue by doing default action for current Key/Event
?List	Choose action from one long list of all actions
String	Actions to manipulate string/character values and variables
Set	Set character position in string to a given value (first is 1, then 2...)
Get	Set variable to the character in the specified position in string
Append	Append the characters in the first string to the end of the second
Backspace	Shorten string by removing last character
Delete	Shorten string by removing character in specified position
Insert	Add character or string into string at specified position
Overwrite	Replace character(s) at specified position in string with new ones
Fill	Make string consist of a specified number of a character only
Erase	Shorten string to have no characters set
Length	Set variable to the left of = with number of characters in string
Process	Have current Key/Event "edit" the string, adding and backspacing
Convert	Convert a string to a number or number to a string, setting variable
Where	Set variable to number of first position in string that matches value
Trim	Remove leading and trailing space characters from string
Xtract	Remove all but specified part of string and leave in variable
Replace	In 3rd string replace chars matching 1st string with corresp. ones in 2nd
Miscellaneous	Miscellaneous actions including Builtin and I/O
Key	Assign current Key/Event value to variable
Current	Assign number of current slide to variable

Flush	Ignore all keys that have just been "Typed-Ahead"
Redisplay	Update screen
Pause	Pause for specified time or until key pressed
Long-Pause	Pause for specified time even if key pressed
NextKey	Get next keypress, then process as Key/Event
Input	Get next keypress, assign to variable, then continue
Exec	Execute external .EXE or .COM file (See Manual!!!)
Mouse	Control optional mouse (See Manual)
AttribTranslate	Set items in Attribute Translate Table
TranslateCmds	Control use of the Attribute Translate Table
Debug	Start step-by-step RUN mode
Get-Builtin	Assign specified internal value to variable (See Manual)
Set-Builtin	Assign new value to specified internal value (see Manual)
File	Actions to Open, Close, Read and Write files, and Print
Open	Open file for Read ("R"), Write ("W"), Update ("U"), Append ("A")
Close	Finish using file
Read	Read next line in file, setting string variable and error code
#Read	Read specified number of bytes into string
Write	Write string into file, followed by CR/LF
***WriteNCRLF**	Write string into file without adding CR/LF
Seek	Have next Read or Write be at specified byte in file (0-32767)
Print	Print string on PRN:
&PrintNCRLF	Print string on PRN: without adding CR/LF
=	Assign value on the right of = to variable on the left
+	Add two values and put the result into variable to the left of =
-	Subtract second value from first and put remainder into variable on left
*****	Multiply two values and put the product into variable on left
/	Divide first value by second and put quotient into variable on left
%	Divide first value by second and put integer remainder into variable
&	Result is a computer AND of the first value with the second
|	Result is a computer OR of the first value with the second
Incr	Add value to variable
Decr	Subtract value from variable
1ST	Define first argument to action
2ND	Define second argument to action
3RD	Define third argument to action
;COMMENT	Set comment for line
WAIT	Set amount of time to wait after displaying this slide
TYPE	Set what to do after the wait (the slide's Run Type)
TAB/INS/DEL	Use the Tab key for 1st Arg, Ins/Del to indent/un-indent current line
#	Move to specific line number
VARS	Show list of variables and their values
OK	Return to previous menu
Insert	Create new variable
Delete	Delete variable(s)
Move	Reposition variable(s) in list

Group	Toggle start of group variable
#	Move highlight to specific position in list
Locate	Find next variable with given name
Name	Edit name of variable
Value	Edit value of variable
Passed-On	Toggle "Passed-On" setting for variable
INSERT	Insert a new line, then set Key/Event label and action
KEY/EVENT	Set Key/Event label for current line
Select	Select Key/Event
#	Prompt for key/event number
Key	Prompt for sample key press
Tag	Set Tag Key/Event and name
Reset	Reset to no Key/Event label
DELETE	Do Copy, then delete line(s)
MOVE	Move line(s)
GROUP	Set/Clear start of group of lines
COPY	Save line(s) for future Pasting or I/O Write-Code
Paste	Insert previously Deleted/Copied lines below current line
OK	Return to editing
***DEBUG**	Do RUN, but stop before each action
RUN	Start displaying the slides automatically
MACRO	Learn and then playback keystrokes
OK	Return to editing
RUN	Execute keystrokes saved in macro
SAVE	Store A-Z (0-9 go with slides)
LEARN	Start recording keystrokes
EXTEND	Continue recording keystrokes
VIEW	Display contents of macro
OK	Return to Macro Menu
Insert	Add item to macro
Delete	Remove item
NAME	Edit name of highlighted macro
DELETE	Erase macro definition
GLOBAL	View/Change global settings
EDIT	Change value of highlighted item
No Marked Block/Marked Block	
Memory	
Current Filename	
System Files Directory	
Display Adapter	
Changes Made	
Typing Attribute	
Select	Select attribute
Insert	Make new attribute definition

		Delete	Remove attribute definition
		Move	Reposition attribute in list
		Name	Edit attribute name
		Value	Edit hexadecimal value
		All	Insert ALL 255 attributes
		Group	Toggle group start
	Grey +/- Cycles		
	Background Attribute		
		Select	Select attribute
		Insert	Make new attribute definition
		Delete	Remove attribute definition
		Move	Reposition attribute in list
		Name	Edit attribute name
		Value	Edit hexadecimal value
		All	Insert ALL 255 attributes
		Group	Toggle group start
	Background Character		
	Menu Background Attribute		
		Select	Select attribute
		Insert	Make new attribute definition
		Delete	Remove attribute definition
		Move	Reposition attribute in list
		Name	Edit attribute name
		Value	Edit hexadecimal value
		All	Insert ALL 255 attributes
		Group	Toggle group start
	Menu Highlight Attribute		
		Select	Select attribute
		Insert	Make new attribute definition
		Delete	Remove attribute definition
		Move	Reposition attribute in list
		Name	Edit attribute name
		Value	Edit hexadecimal value
		All	Insert ALL 255 attributes
		Group	Toggle group start
	Default Video Bits		
	Default Run Type		
	Run Action Indent Increment		
	Overlays Shown		
	Timeout During Run		
OK			Return to editing slide
CLEARALL			Erase all slides, etc., and start anew
RUN			View Global Run Actions
VARIABLES			View Variables List
	OK		Return to previous menu
	Insert		Create new variable

		Delete				Delete variable(s)
		Move				Reposition variable(s) in list
		Group				Toggle start of group variable
		#				Move highlight to specific position in list
		Locate				Find next variable with given name
		Name				Edit name of variable
		Value				Edit value of variable
		Passed-On				Toggle "Passed-On" setting for variable
	.OVERLAYS					Display Global Overlays List
	NAMES					Display Global list of block names and definitions
		Block				Make selected item current block
		Comment				Edit block's comment
		Position				Redefine item to current block
		Name				Edit name of highlighted block
		Insert				Insert current block as new item
		#				Move to specific item
		Group				Toggle start of group
		Delete				Delete block name definition(s)
		Move				Reposition item
I/O						Save, Load, Print, etc.
	SAVE					Save all slides, attributes, save areas, etc., in a file
		Select				Use highlighted filename
		New				Type in explicit filename
		Full				Type in explicit full pathname
		Dir				Change default dir shown above
	LOAD					Clear everything, then reload all from a saved file
		Select				Use highlighted filename
		New				Type in explicit filename
		Full				Type in explicit full pathname
		Dir				Change default dir shown above
	PRINT					Output slides to printer/file in selected format
		Edit				Change value of highlighted item
			Output To			
			Character Mapping			
				Select		Use highlighted name
				New		Type in explicit name
				C		Output in C Language format
				Pascal		Output in Pascal Language format
				Don't		No mapping of characters
				Mapping		Define/View current character mappings
					First	Edit first character to output
					Second	Character to output after Backspace
					Normal	Output character as itself
					Done	Save mapping in file
					Cancel	Return to Printer Menu (doesn't Save)

Output: Text
Output: Attributes
Output: Names/Numbers
Output: Overlay Lists
Output: Run Info
Output: Variables List
Output: Block Names
Output: Slides That Reference Slide
Page Break After Slide
Blank Lines After Slide
Trim Trailing Blanks
(Block Marked -- Output From Within Block, Only)
How Many Slides to Output

OK	Return to editing slides
Start	Output as set above
ADD	Insert slides, etc., from selected slides in other file
Select	Use highlighted filename
New	Type in explicit filename
Full	Type in explicit full pathname
Dir	Change default dir shown above
RETRIEVE	Insert screen images from CAPTURE, PCX files, Text files
CODE-READ	Load Action Copy/Paste Buffer with saved code
Select	Use highlighted filename
New	Type in explicit filename
Full	Type in explicit full pathname
Dir	Change default dir shown above
WRITE-CODE	Save contents of Action Copy/Paste Buffer in a file
Select	Use highlighted filename
New	Type in explicit filename
Full	Type in explicit full pathname
Dir	Change default dir shown above
HELP	Reference for what keys do what, etc.
QUIT	Return to DOS

Differences From the Older Dan Bricklin's Demo Program 1A

This section lists some of the differences between the older **Dan Bricklin's Demo Program** and DEMO II. Almost all of the changes are upwards-compatible, so an experienced user of the older program should be able to start using the new program right away. ***We do, though, recommend at least looking at the "Overview" section of this manual and running RUNME on the Second Diskette to get acquainted with the new features.***

Upgrading

To upgrade from the older program to DEMO II, you just start using DEMO2.EXE instead of DEMO.EXE, RDEMO2.EXE instead of RDEMO.EXE, and the new CAPTURE.COM and CAPTCMD.EXE instead of the older version of CAPTURE and the CAPTOFF program.

DBD files created with the older program will load into DEMO2 and RDEMO2 correctly. They will be converted into the new format and, when written out, will only be readable by DEMO II.

You don't have to learn all the new features to use this new version. The main changes are the *addition* of many new choices to each menu. The old choices are usually still there.

Look through the list of changes and additions that follow in this section.

You will find the **Run menu** structured differently. It now includes what was the Run Handlers menu. You still add handlers, now called "run action lines with Key/Event labels", by executing the Run Insert command. The View run action is almost identical to the old View command. The "Keys: Yes/No" setting has been replaced by the Run Type command, and the Wait setting by the Run Wait command. See the section, "Run Commands", for more information. You may want to try the tutorial on the Second Diskette, which shows how to set run actions.

Note that the items on the Run list are no longer automatically sorted. You must ensure that "a" comes before "Any Key", for example. Also, you should usually use "Any Key" where you used to use "Anything". See the section, "How Running Works", for more information about "Anything!".

The **Overlays commands** have changed a bit. You create a new slide overlay with the Overlay Slide command, not the Insert command.

Since DEMO2 and RDEMO2 use more memory than their older counterparts, some large old files may be too big to fit. You will get "Error reading file" or "Out of Memory" messages when you load these files. You should load them into the older DEMO program instead, write them out as two smaller files, and then convert them separately.

Note that the **macro** format of DEMO II is different from the older format. The Alt-0 through 9 macros are not loaded from old DBD files, and your old "_ALT_A_Z.SG1" file must be replaced by a new one. ***Do not try to use an old "_ALT_A_Z.SG1" file with DEMO II.***

CAPTURE works like the older version. The main difference is that you must run it with the same DOS Current Directory as you will be in when you run DEMO II. It has many new features, but you don't have to learn about them if you don't want to.

How V1 Files Are Converted Into DEMO II Format

Just about everything controlling the running of a slide show in V1 has a counterpart in DEMO II, so the conversion is pretty complete. When V1 files are loaded, they will take up a little more memory due to a few factors. Tones take up an extra run action line or two. "Any NonPrnt", which does not have an exact analogue in DEMO II, is converted into "Any Edit, Any Cursor, Any FuncKey". "Anything", which is less extensive in V1, is converted into "Any Key, Any Tag, Timeout". Events are converted into using Tags "Event_1" through "Event_9". The "Keys: Yes/No" setting is converted to Run Types 1 and 0, respectively. The "I/O: Yes" setting is converted into "Viewed: Builtin(10)=1". The "Clear Buffer: Yes" setting is converted into "Viewed: Erase TIBUFFER". The variable "TIBUFFER" is used instead of the old Type-In Buffer.

You may want to look at the run actions and overlays in a converted file, and remove the unnecessary parts. These include the comments explaining "Anything" and other nondirect conversions, as well as the extra Key/Event labels for old "Anything's" where you really meant just "Any Key".

Changes

There is a new DBD file format. Both the older format and the newer formats can be read, but only the new format is written.

The Ctrl-Left and Ctrl-Right arrows move to the next change, not the next minor tab stop.

The Home and End keys stop the cursor at the edge of what is typed instead of going to the screen edge. If the cursor is beyond the edge of what is typed, it moves to the screen edge. Pressing them twice always moves the cursor to the screen edge.

The items on the old Run menu have been implemented differently. There is a Run Wait command to set the wait value; Run Type command to set the Run Type (replaces Keys: Yes/No); the I/O spin the diskette operation is replaced by the Set Builtin(10) action; and the Clear operation is replaced by the Erase action.

The old Overlays Copy menu has been moved to the Main menu.

The Shift-F5 key uses the Copy All command, and now copies run information as well as overlay information.

The old Block Fill menu has been moved into the Block menu.

The Shift-F8 key calls up the Block CAB menu.

The old Typing Position command has been moved to the Typing Status menu.

There is a new Macro save format, so old macros cannot be used.

The CAPTURE command no longer uses the F1h interrupt, so it should work on a wider variety of machines.

The Run list is no longer sorted automatically. DEMO II keeps the order you set.

Additions

DEMO command

The DEMO2 command can use the DEMO2 DOS Environment String to get options.

The display adapter type is automatically determined in a wider variety of circumstances, including EGA, VGA, Hercules, and Compaq.

The -MEM option has been added to the DEMO2 command.

The -BIT16 and -BITM32 options have been added to the DEMO2 command for use with bitmappped graphics displaying.

The -ARG option has been added to the DEMO2 command to pass arguments to the running slide show.

DEMO2 restores the video mode and cursor when returning to DOS.

General

There are now both numeric and string variables. These variables can affect overlay positioning, turn overlays on and off, be used as multiple "type-in buffers" (to use the old term), be manipulated and tested while running, and much more.

New Bitmapped Image slides can refer to ".PCX" files or have captured information about CGA, EGA and Hercules graphics images.

New overlay types now exist: Numeric and String Value, Absolute and Relative Slide Reference, and Attribute.

The CAB ("Character, Attribute, Both") setting has been added. This determines whether typing, deleting, moving, etc., affect just the characters, just the attributes, or both.

Keys

Delete and Backspace are affected by the CAB setting.

The Ctrl-PgUp and Ctrl-PgDn keys move the cursor to the edge vertically.

Grey + and Grey - cycle through different attributes for a block, or a character position and the Typing Attribute.

Grey * executes the new Typing Find command to move the cursor to given characters.

Menus

Many menus have the new Group command that lets you group consecutive items for deletion, moving, etc.

Many menus have the new # command that lets you move to a specified item.

Block Commands

The Attribute list now displays a sample for all attributes in the list, not just the one being changed.

The new Block Attrib All command defines all 255 visible attributes.

The Block Last and Shift-F9 commands move the cursor from corner to corner, if a block is already marked.

The new Block Wrap command word-wraps text within a block. Text can now be continuously word-wrapped as you type.

The new Block Exchng command exchanges the Delete/Paste buffer with the contents of a block.

The new Block Xlate command converts attributes within a block, on a slide, or on all slides. This can also be done during runtime to convert from color to monochrome temporarily with the Translate Attribute run action.

The new Block 3_Box command creates a heavy line box.

The new Block 4=NoBox command erases a box.

The new /CAB command displays and modifies the CAB settting.

The new Block Names command manipulates named marked block references.

Typing Commands

The new Typing Chars Num command lets you specify a special character by value.

The new Typing Status menu lets you configure a status display (upgrades old Position Indicator). This includes cursor position, slide name and number, current file name, and insert/overwrite indication. The insert/overwrite indication is always on by default.

The new Typing Find command lets you move the cursor forward to the next screen position with specified characters.

Slides Commands

The new Slides Print command toggles the new Print Flag. You can set printing to print all slides with this flag present.

The new Slides Options command views and sets Switch Type, Switch Speed, Video Bits, and Palette.

The new Slide Switch Type and Switch Speed settings control how DEMO II switches from one slide to another, either as quickly as possible, or at the set speed from top to bottom or bottom to top. This applies to both text slides and bitmapped slides. For text slides, you can also just replace the changed characters and attributes at the speed set, with an optional click sound after each changed character position is displayed.

The new Video Bits setting controls border colors, 320x200 color set, and blink vs. intensity for the background attribute color in text mode.

The new Palette setting adjusts EGA color palette for 640x350 16-color bitmapped images.

Copy Commands

The new Copy All command copies overlays and run information.

The new Copy Run command copies run action lists (which used to be called "run handlers"), Run Type, Run Wait, Switch Type, Switch Speed, Block Names, Video Bits, and Palette information.

Overlays Commands

There is a new structure to the Overlays command.

The Overlays command title includes slide name and number.

The new Overlays Nums command associates variables with overlay positioning and/or presence.

The new Visible overlay setting controls whether or not an overlay is shown under control of a variable.

There is a new Max Chars Shown overlay setting.

Run Commands

There is a new structure to the Run command.

The Run command title includes slide name and number.

A Run Actions List concept has been implemented, allowing more than one action when a Key/Event is matched. This allows you to execute simple programs constructed out of run actions when a key is pressed.

Over 100 new run actions have been added, including arithmetic operators to change variables, additional sound-generating actions, Slide Return After, view particular slide in another file, View Absolute, If, While, For, Select/Case, Goto, string manipulation, PC timer access, primitive mouse control, file and printer I/O, and much more.

You will find more Key/Events, including all of the standard PC key combinations, such as Alt-letter, Ctrl-letter, and the Grey +/-/* keys.

The new Tag Key/Event implements Basic Language "GOSUB" and "GOTO"-like run actions.

The new Viewed, Displayed, Readkey, and Keypress Key/Events give you greater control over running.

The new Run Type setting for each slide implements several built-in ways of switching from slide to slide and reading keys, including optional flushing of type-ahead and individual-slide time outs.

There is a menu-driven editor for creating and modifying run action lists.

Comments are now allowed on run action lines to improve readability.

Optional indenting improves the readability of run actions.

The new Run Key/Event # command selects Key/Event by value.

The new Run Key/Event Key command lets you specify a Key/Event by just pressing the key to which it corresponds.

The new Run Delete/Copy/Paste commands manipulate the new Action Copy/Paste buffer. This allows you to replicate run actions or move them from slide to slide or file to file.

An enhanced Debug mode provides more information.

Ctrl-Break can either stop running or start Debug mode.

Access to shift-state information detects whether the Shift, Ctrl, or Alt keys are up or down.

Macro Commands

There are new Macro View Insert and Macro View Delete commands for editing macros.

Global Commands

Additional Global settings have been added.

New Global Overlays list lets you pop up overlays over all slides, possibly controlled by variables.

New Global Block names have been added.

I/O Commands

There are additional Print settings for the new features.

The I/O Add command loads more information, including run actions and overlay definitions.

I/O Retrieve can read normal text files (creating a text slide) and reference ".PCX" files (creating a bitmapped image slide).

New I/O Write-Code and I/O Write-Code commands save and load the Action Copy/Paste buffer.

CAPTURE Program

The new CAPTURE command has setable Hot Key as well as Timed Triggering.

The CAPTURE command captures more information, including H/W cursor position.

The CAPTURE command can capture a variety of bitmapped graphics images.

A new CAPTCMD command controls CAPTURE.

Deletions

The old Run and Run Handlers pair of menus have been replaced by the new single Run menu.

The old "Events" actions and Key/Events have been replaced by multiple run actions per Key/Event and the Tag Key/Event.

The Type-In Buffer has been replaced by a more general use of many variables.

The Cursor Attribute setting has been replaced by an algorithm for inverting the foreground and background colors to create a cursor attribute.

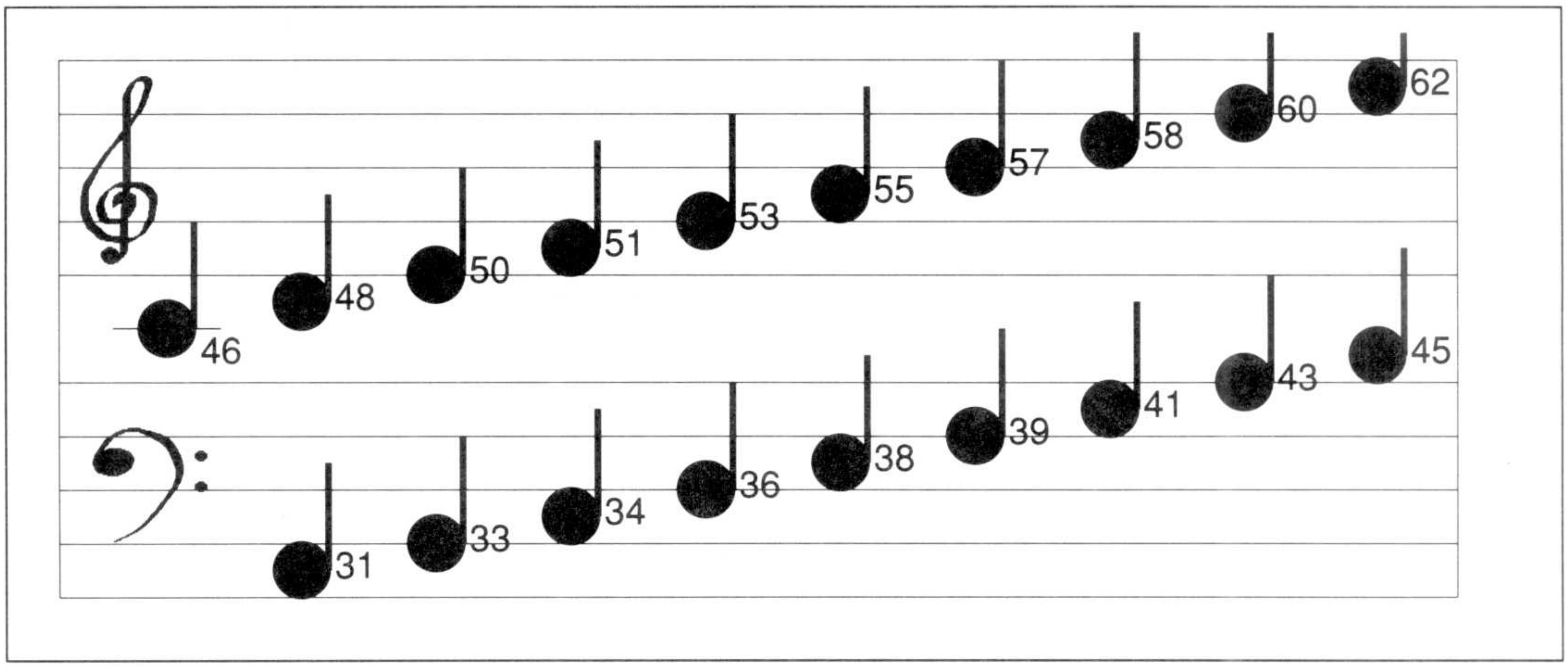
Appendix A: Tone Chart

The Tone Note and Melody run actions require you to use explicit tone values. These values are integers from 0 to 99 for Tone Note and 1 to 99 for Tone Melody. They represent musical notes. DEMO II uses the timer in the PC to produce these tones. Like Run Wait values, they are independent of the clock speed of the CPU (i.e., they will sound the same on an AT as on a PC).

The notes are represented by the values as follows:

OCTAVE:	0	1	2	3	4	5	6	7	8
NOTE									
A		07	19	31	43	55	67	79	91
A#		08	20	32	44	56	68	80	92
B		09	21	33	45	57	69	81	93
C		10	22	34	46	58	70	82	94
C#		11	23	35	47	59	71	83	95
D	00	12	24	36	48	60	72	84	96
D#	01	13	25	37	49	61	73	85	97
E	02	14	26	38	50	62	74	86	98
F	03	15	27	39	51	63	75	87	99
F#	04	16	28	40	52	64	76	88	
G	05	17	29	41	53	65	77	89	
G#	06	18	30	42	54	66	78	90	

For convenience when transcribing tunes from sheet music, a portion of the chart is shown differently below:



Appendix B: Attribute Chart

The charts below list the attribute values for Block Attribute Value settings. They also describe the way that these values will display on a standard IBM Color/Graphics Adapter (and EGA) and monitor, and on the IBM Monochrome Adapter and monitor.

Note that not all monitor/adapter combinations will display the same colors for a given attribute. For example, brown is sometimes orange. To check, use the sample value displayed by the Block Attribute Value command to check how it displays, or load the ATTRIBS.DBD file on the Second Diskette and look at the display there.

Monochrome monitors on CGAs can give undesirable results when displaying colors, and some monochrome adapters (IBM's Monochrome Display Adapter and the Hercules Graphics Card) show underlined or bright in place of some of the colors. If you want the display to look the same on all combinations that viewers may have, the only attributes that should be used are Normal (07) and Inverse (70).

You can use the Block Xlate command to change attributes permanently, and the Translate run actions to do it temporarily while running.

CGA Character		**CGA Character**	
Background	**Value**	**Foreground**	**Value**
Black	0	Black	0
Blue	1	Blue	1
Green	2	Green	2
Cyan	3	Cyan	3
Red	4	Red	4
Magenta	5	Magenta	5
Brown	6	Brown	6
White	7	White	7
Black w/Blink	8	Gray	8
Blue w/Blink	9	Light Blue	9
Green w/Blink	A	Light Green	A
Cyan w/Blink	B	Light Cyan	B
Red w/Blink	C	Light Red	C
Magenta w/Blink	D	Light Magenta	D
Brown w/Blink	E	Yellow	E
White w/Blink	F	Bright White	F

To create an attribute value, use two characters: the first, from the Background column, and the second from the Foreground column. For example, White characters on a Blue background would be 17.

Monochrome Display Adapter Attributes	**Value**
White (Green) Character on Black (Normal)	07
Underlined Normal	01
Black on White (Inverse)	70
Normal w/Blink	87
Underlined Normal w/Blink	81
Inverse w/Blink	F0
Normal Bright	0F
Underline Bright	09
Normal Bright w/Blink	8F
Underline Bright w/Blink	89

Blink vs. Bright

You can use the Video Bits setting (see the Slides Options command) to change the blink attributes into bright. For example, Blinking Blue on Black (81) would turn into Blue on Grey. You do this by setting the Video Bits bit 6 to zero. For example, the normal Video Bits setting is 0x70. You would change it to 0x30 to change blinking to intensity.

Appendix C: Miscellaneous

About the Program and Documentation

DEMO II was designed and written by Dan Bricklin. It is a major upgrade to the original Dan Bricklin's Demo Program. The program consists of over 30,000 lines of C language code, written in Microsoft C version 4.0, and about 5,000 lines of code and constants written for Microsoft Macro Assembler. The CAPTURE program is a major upgrade to the CAPTURE program that came with the original Demo program, and these modifications were also done by Dan Bricklin. The programming was done on a Tandy 3000HD. Some of the testing was done on an IBM PC, a Compaq Portable III, and an IBM PS/2 Model 50.

This manual was written by Dan Bricklin using Xerox Ventura Publisher 1.1. Most of the original typing and almost all of the illustrations were done directly in Ventura. Much of the text on the package itself was produced using Aldus PC PageMaker. Proofing was done on an Apple Laserwriter, with final output on a Linotronic 100. The sample files on the diskettes were written by Dan Bricklin.

The "tomato" cover design was done by Doliber Skeffington Design, of Boston, who also created the identity system for the Boston Computer Society. The design of the inside of the book was done by Dan Bricklin, with some assistance from his father, Baruch Bricklin of Bricklin Graphics in Philadelphia. The parts of the package were designed and purchased with the aid of Jennifer Cushing of Software Garden, Inc.

Acknowledgements

I would like to thank all of you who helped make this product possible.

There are the purchasers of the original Demo program, who showed that this is a valuable and useful tool. Their numerous suggestions and comments were the most influential part in deciding upon the new features.

There are the beta testers of DEMO II. Without their help this product would not have been shipped. They helped valiantly to find numerous bugs, and their experiences and comments had great influence on the documentation. Since many who helped may be barred from using their names by their companies, I will not list their names.

Finally, there are my relatives and friends who put up with my inavailability and preoccupation during the long development period.

Index

License Agreement and Warranty

This software product is licensed by Software Garden, Inc., (SGI) to you, the original purchaser for your use only on the terms set forth below. **BY OPENING THE DISKETTE PACKAGE YOU ARE INDICATING YOUR ACCEPTANCE OF THESE TERMS.** If you do not accept these terms, contact SGI or the place where you purchased the product within 10 days of receipt for a full refund.

In consideration of your accepting the terms of this Agreement and payment of a license fee as pa rice you paid for the product, SGI grants you non-exclusive licenses as follows:

You may use the programs (DEMO2.EXE, RDEMO2.EXE, CAPTURE.COM, CAPT DECOD.EXE), ".DBD" files, and other files on the diskettes in this package on compatible hardware.

You may make Backup copies of the programs and files to protect yourself from e

Only one (1) copy of each of the programs and files may be in operation at any

You may duplicate an unlimited number of "RDEMO2 Diskettes" for use b Diskette" is a diskette that contains RDEMO2.EXE for the purpose of running a ".DBD" .EXE by you or at your request and where that ".DBD" file is on the diskette when it is dur

Other than as stated above, **you may not** copy, alter, translate, d grams or documentation; nor transfer, rent, lease, sub-license or otherwise distribute the ove (or cause not to be displayed) any copyright notices or startup messages contained in

SGI reserves all rights, including copyrights, not expressly g

SGI warrants to you that it is the owner of the copyright to indemnify you against all claims from third parties challenging such ownership, provided that you provid ny such claim and permit SGI to defend or settle the claim.

Refer to license agreement on envelope inside.

Before making copies of RDEMO2.EXE you will ascertain its fitn our use, and check that it is sufficiently free from error or malfunction for use by the intended recipients of the copies. You agree to indemnify SGI and Dan Bricklin against all claims or liability which relate to your use of RDEMO2.EXE, or any third party use of "RDEMO2 Diskettes," provided that SGI gives you prompt notice of any such claim and permits you to defend or settle the claim.

The programs are warranted by SGI to substantially conform to the documentation and SGI's published specifications, provided that they are used as set forth here and in the documentation. Nevertheless, due to the complex nature of computer programs, the programs in this package (like all large programs) will probably never be completely error-free. The physical diskettes and documentation enclosed in this package are warranted by SGI to be free from defects in materials and workmanship that prevent you from operating the programs. These warranties extend for a period of 60 days from the date of purchase. SGI will replace defective diskettes or documentation, or correct substantial program errors free of charge during that period. You must return the defective items to SGI, postage prepaid, with a dated proof of purchase within those 60 days (contact SGI for shipping instructions before sending). In the event that SGI is unable to replace the defective diskettes or documentation, or correct the substantial program errors, SGI will refund your purchase price and the licenses will terminate. These are your sole remedies for any breaches of warranty.

Other than as stated above, the programs are licensed, and the documentation and other files on the diskettes are provided, "AS IS," without any warranty as to performance, accuracy, or freedom from error, or as to any results generated through use of such material, including, without limitation, any implied warranties of merchantability or fitness for a particular purpose. You assume the entire risk as to the results and performance of this product. SGI and Dan Bricklin specifically do not warrant that the program will meet your requirements or operate without interruption or error. SGI's obligation to replace defective media, documentation and programs as provided above shall be your sole and exclusive remedy for any and all claims against SGI or Dan Bricklin arising out of or in connection with this product, whether made or suffered by you or any other person and whether based in contract or tort. Under no circumstances, whether in contract or in tort, shall SGI or Dan Bricklin be liable for indirect, consequential, special, or exemplary damages such as but not limited to loss of revenue, data, or anticipated profits, lost business, or other economic loss arising out of or in connection with this Agreement or your use or inability to use the diskettes, the documentation or the programs. In any event, any liability of SGI or Dan Bricklin arising out of or in connection with this Agreement or your use or inability to use this product, whether based in contract or tort, shall not exceed the amount you paid, if any, for the product.

These licenses shall terminate upon your failure to comply with the terms and conditions of this Agreement.

This Agreement is to be interpreted by the laws of the Commonwealth of Massachusetts.

This Agreement may not be assigned by you without the written consent of SGI, which consent will not be unreasonably withheld. This Agreement shall be binding upon the respective successors and assigns of the parties.

This Agreement sets forth the entire agreement of the parties and supersedes all prior understandings and agreements, written or oral.

Software Garden, Inc. / PO Box 373 / Newton Highlands, Massachusetts 02161 / U.S.A. / 617-332-2240