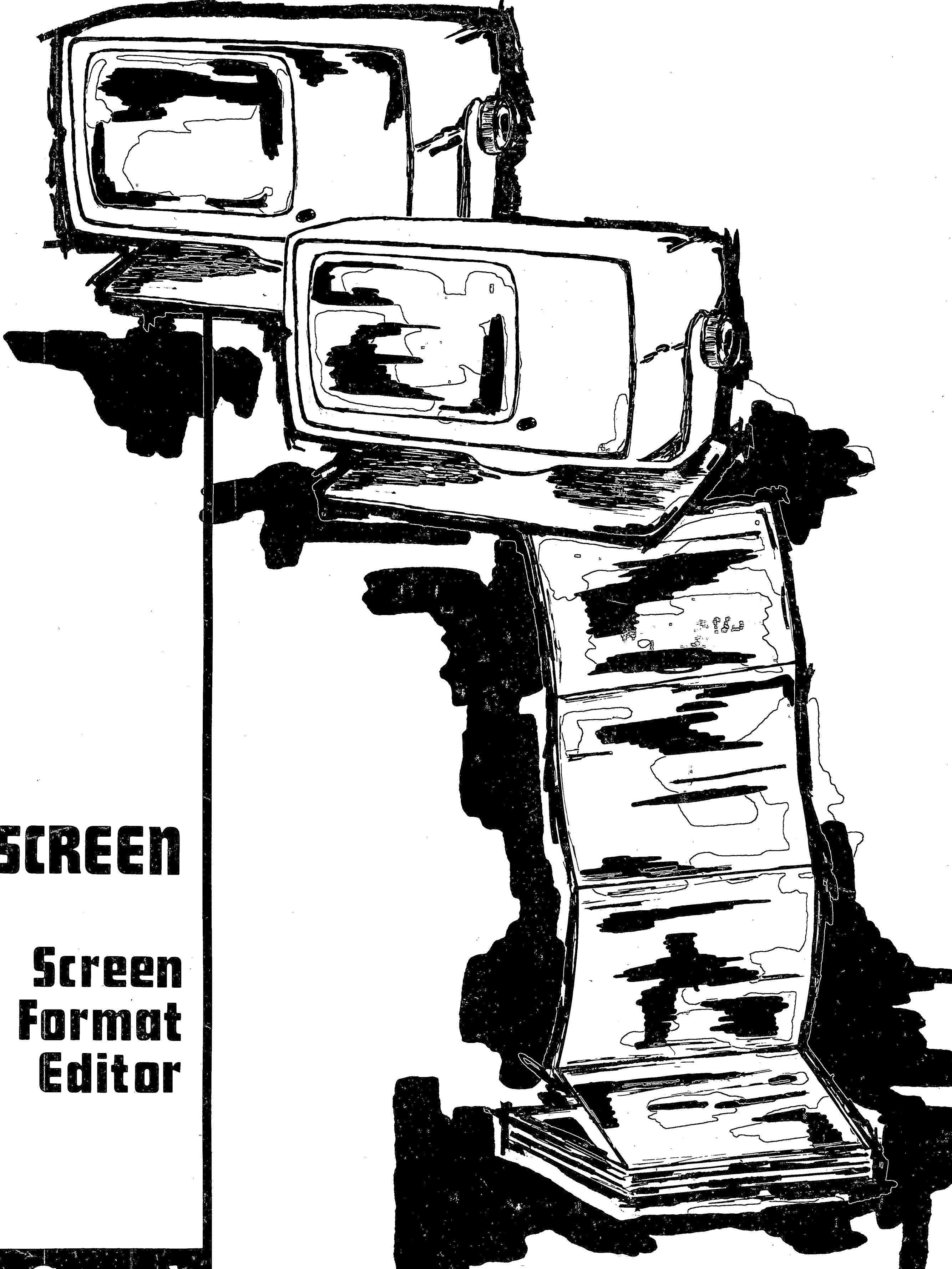
CS COMMERCIAL SYSTEMS

DEVELOPER'S AID SERIES



SCREEN

Screen Format Editor

Data General

SCREEN

screen format editor

A Small Business Systems Publication

055-006-01

Data General Corporation, Westboro, Mass. 01581

NOTES

NOTES

NOTES

031: 0-9,A-D,L,P,R,S,V,Z,Z,*,+,-,COMMA,;,(,),",/,=,SPACE MAY NOT BE CURRENCY SIGN

A character other than those listed in the message was assigned to be the CURRENCY SIGN.

032: NESTED COPY STATEMENTS ARE NOT VALID

Nested COPY statements are not permitted in Interactive COBOL.

033: ILLEGAL FORMAT - INVALID CONSTRUCT AFTER KEY WORD 'COPY'

The keyword COPY must be followed by a filename literal (enclosed in quotation marks).

034: OUTPUT FILE NAME IS THE SAME AS THE IMPLIED INPUT FILE NAME

The ORGANIZE command specified for output the descriptor file from which the current screen image was retrieved ("G" command). SCREEN needs to retain this file in order to build a new file that includes group structure descriptors.

011: NUMERIC ARGUMENT IS OUT OF RANGE
A negative numeric argument was specified with a command that accepts only positive arguments.

012: TOO MANY LOOP LEVELS HAVE BEEN DEFINED
Repetition brackets in the command line were nested more than four levels deep (CLI/SCREEN) or ten levels deep (IC/SCREEN).

013: UNMATCHED RIGHT BRACKET
No left bracket (indicating the beginning of an repetition loop) was included in the command line.

014: ILLEGAL FORMAT - INVALID CONSTRUCT AFTER KEY WORD 'PICTURE' OR 'PIC'
The ANALYZEd file contained a syntax error in a PICTURE statement.

015: THE LEGAL TRANSFER MODES (SEE 'T' COMMAND) ARE A, F, and T.

016: THE LEGAL FORMAT INPUT MODES (SEE 'F' COMMAND) ARE I AND R
The TRANSFER and FORMAT commands are called with two-character commands, as indicated in these error messages.

017: ILLEGAL FORMAT - REPEAT FOR UNSPECIFIED CHARACTER
No character was included with the parenthesis repetition indicator. For example, FR(10)$$.

018: ILLEGAL FORMAT - NON-NUMERIC REPEAT COUNT
A non-numeric repeat count was found inside a pair of parentheses. For example, FRX(P)$$.

019: VALID FORMAT CHARS ARE: 9 A B X Z /,.*+ - CR DB CURRENCY SIGN
The format input command included a character other than those listed in the message.

020: ILLEGAL FORMAT - INVALID CONSTRUCT AFTER KEY WORD 'TO'

021: ILLEGAL FORMAT - INVALID CONSTRUCT AFTER KEY WORD 'FROM'

022: ILLEGAL FORMAT - INVALID CONSTRUCT AFTER KEY WORD 'USING'

023: ILLEGAL FORMAT - INVALID QUALIFIER

024: ILLEGAL FORMAT - INVALID CONSTRUCT AFTER KEY WORD 'ALL'
The ANALYZED file contained the syntax error indicated in the message.

025: NO FILE NAME HAS BEEN SPECIFIED
A filename was not specified in a command that requires one.

026: THE SPECIFIED FILE DOES NOT EXIST
SCREEN could not find the specified source file or descriptor file.

027: THIS IS AN ILLEGAL OPERATION IN COMMAND EDIT MODE
Source and descriptor file commands may not be used to edit a command line.

029: THE SCREEN SECTION SOURCE CONTAINS TOO MANY FIELDS (128 MAX)
SCREEN would have to generate more than 128 source lines (the maximum allowable number) in creating a source file.

030: VALID FORMAT CHARS ARE: 9 A B P S V X Z / , . * + - CR DB CURRENCY-SIGN
The ANALYZED file contained a format character other than those listed in the message.

Appendix D: SCREEN Error Conditions

Whenever SCREEN finds an error in a command line or in an ANALYZEd source file, it suspends processing and displays an error message at the bottom of the screen. The examples below illustrate the error message format:

```
ERROR 025 CMND "W"  ARGS  00002  00003
2,3W
NO FILE NAME HAS BEEN SPECIFIED
```

This error occurred in EDIT mode when the illegal command "2,3W" was issued. To erase an EDIT or ORGANIZE mode error message and resume processing, press **CR**.

```
ERROR 010
        05 LINE 05 COL 30 PIC XXX VALUE IS 345 TO I-CODE.
ILLEGAL FORMAT - INVALID CONSTRUCTION AFTER KEY WORD 'VALUE'
```

This error occurred while SCREEN was analyzing the Screen Section of a source file. (Evidently, this program had never been compiled!) SCREEN may be able to recover from an ANALYZE mode error condition, and return to EDIT mode. In some cases, however, SCREEN passes control back to the CLI or LOGON.

A list of SCREEN's error messages follows. In some cases, IC/SCREEN's error messages are somewhat shorter.

02: **ILLEGAL OR UNDEFINED COMMAND**
SCREEN did not recognize the command you tried to execute.

03: **ILLEGAL FORMAT - A NUMBER WAS NOT EXPECTED**

04: **AN UNEXPECTED END OF FILE WAS READ WHILE READING SOURCE FILE**
The ANALYZEd file contained neither a Screen Section nor the **SCREEN** header inserted by SCREEN when it creates source code.

05: **ILLEGAL FORMAT - INVALID USE OF THE KEY WORD 'SCREEN'**
06: **ILLEGAL FORMAT - INVALID USE OF THE KEY WORD 'SECTION'**

07: **ILLEGAL FORMAT - INVALID USE OF THE KEY WORD 'PLUS'**

08: **ILLEGAL FORMAT - INVALID USE OF THE KEY WORD 'ZERO'**

09: **ILLEGAL FORMAT - INVALID CONSTRUCT AFTER KEY WORD 'BLANK'**

010: **ILLEGAL FORMAT - INVALID CONSTRUCT AFTER KEY WORD 'VALUE'**
The ANALYZEd file contained the syntax error indicated in the message.

Appendix C: List of Figures

Appendix B: SCREEN Command Summary

This appendix provides a brief summary of all SCREEN commands. EDIT and ORGANIZE mode commands are presented alphabetically. Also listed are "immediate-action" keys that may be used in these two modes. Note that the **HOME** key may be used in any SCREEN mode.

EDIT MODE COMMANDS	
A	ANALYZE source file (CRT format)
B	BEGIN again
C	CREATE SOURCE file (CARD format)
D	DELETE characters
E	EXIT from SCREEN
F	Insert/Replace FORMAT characters
G	GET screen image from storage
H	View HELP screen
I	INSERT lieteral characters
J	JUMP cursor to new position
K	Delete lines
L	Move cursor by lines
M	Move cursor by characters
N	CREATE SOURCE file (CRT format)
O	ORGANIZE screen image
P	PRINT a screen image, source file, or descriptor file
Q	ANALYZE source file (CARD format)
R	REPLACE literal characters
S	SPLIT line
T	TRANSFER fields from one line to another line
U	UNITE consecutive lines
V	Assign value of CURRENCY SIGN
W	WRITE descriptor file
Z	Jump cursor to end of a line

ORGANIZE MODE COMMANDS	
A	ADD field to group
D	DELETE leading/trailing spaces
E	EXIT from ORGANIZE mode
G	Create new GROUP
H	Display HELP screen
J	JUMP cursor to new position
L	Move cursor by LINES
M	Move cursor by characters
R	REMOVE field from group
S	SPLIT field
U	UNITE split fields
X	Extend field with blanks
Z	Jump cursor to end of line

IMMEDIATE-ACTION KEYS	
f1	Delete character (terminate direct-entry)
SHIFT-f1	Delete line (terminate direct-entry)
f2	Toggle between INSERT/REPLACE direct-entry-modes
SHIFT-f2	Split line (terminate direct-entry)
f3	Enable LITERAL direct-entry
SHIFT-f3	Enable FORMAT direct-entry
f11	Display HELP screen
HOME	Terminate SCREEN, or (from ANALYZE mode) return to EDIT mode
f9	Erase command line (terminate direct-entry)
f10	Edit command line (terminate direct-entry)
DEL	Erase last command character (terminate direct-entry)
ESC	Command separator and terminator (echoed as "$")
→	Move cursor one column to the right
←	Move cursor one column to the left
↓	Move cursor one line down
↑	Move cursor one line up
CR	Move cursor to beginning of current line
NEWLINE	Move cursor to beginning of next line

GROUP HEADER A screen element that defines and names a group or subgroup. A group header is not associated with any single screen field.

LEVEL NUMBER A number that defines the place of a COBOL statement in a Data Division hierarchical structure. Screen Section level numbers must be between 01 and 49.

LITERAL FIELD A set of consecutive screen positions that displays predefined data, such as headings and messages. The screen element that defines a literal field uses a "VALUE IS" clause to specify the field's contents. Literal fields may contain any displayable ASCII character, including upper/lowercase letters, numerals, and punctuation marks.

ORGANIZE MODE (CLI/SCREEN ONLY) The operational mode in which SCREEN assigns Data Division structure (represented by level numbers) to a screen image.

PICTURE A collection of characters that defines the type(s) of data that a data-item may store. Interactive COBOL uses ANSI-standard PICTURE characters.

SCREEN ELEMENT An Interactive COBOL statement that defines a literal field, a format field, or a group header.

SCREEN IMAGE The purely visual part of a display screen format, consisting of literal fields and format fields.

SCREEN POSITION A location on the 24-line by 80-column DASHER display terminal video screen. A screen position is defined by the "LINE" and "COLUMN" clauses of a screen element.

SCREEN SECTION An Interactive COBOL extension to ANSI standard COBOL that defines the display screen formats that control operator-system communication. SCREEN writes source code for use in the Screen Sections of Interactive COBOL programs, and extracts display screen formats from the Screen Sections of existing source programs.

SOURCE FILE A file that contains Interactive COBOL source program code. SCREEN writes a source file for a screen image after prompting you to specify descriptors for the image.

Appendix A: Glossary

ANALYZE MODE The SCREEN operational mode in which you extract a screen image or an entire display screen format from a source file.

CREATE SOURCE MODE The SCREEN operational mode in which you write source code for a display screen format. This code may be used in the Screen Section of any Interactive COBOL program.

DATA DIVISION STRUCTURE A hierarchical structure of COBOL statements, as defined by the statements' level numbers. File Section items, Working-Storage Section items, and Screen Section items may all be structured in the same way.

DEFAULT FILENAME (1) A new descriptor-file name that corresponds to that of an existing source-file; (2) A new source-file name that corresponds to that of an existing descriptor-file. You may use a default filename: (a) in the command to extract a display screen format from a source file; (b) in the command to create a source file.

DESCRIPTOR (1) A keyword in Screen Section source code that invokes a special function of the DASHER display terminal; (2) A keyword that assigns a screen field to a data-item; (3) A Data Division level number; (4) A COBOL data-name. A complete descriptor list appears below.

Descriptors for display terminal functions and data-item assignments are made in CREATE SOURCE mode. Data Division level numbers are assigned in ORGANIZE mode (CLI/SCREEN only).

DESCRIPTOR FILE A SCREEN "working file" that stores a display screen format (including both screen image and descriptors).

DISPLAY SCREEN FORMAT A screen image, along with additional information that makes the display a functioning part of a COBOL program: calls to terminal functions (e.g. BLINK, BLANK LINE), assignments of fields to data-items, screen names, and level numbers.

EDIT MODE The SCREEN operating mode in which you create and modify screen images. EDIT mode is SCREEN's control center. Passage of control to/from other SCREEN modes, and to/from LOGON or the CLI occurs through EDIT mode.

FORMAT FIELD A set of consecutive screen positions that acts as a "window" through which information flows between the operator and the program. The screen element that defines a format field uses a standard PICTURE clause to specify the type and amount of data that can pass through the window. The format field must pass data to/from a particular File or Working-Storage data-item.

GROUP A collection of screen elements that function as a single unit when invoked by a DISPLAY or ACCEPT statement. Groups define multi-field display screen formats. Groups may possess a hierarchical structure, containing other groups (subgroups). Data Division level numbers define groups and subgroups.

Display Terminal Function Descriptors:

```
AUTO, BELL, BLANK LINE, BLANK SCREEN,
BLANK WHEN ZERO, BLINK, FULL,
JUSTIFIED, REQUIRED, SECURE
```

Data-item Assignment Descriptors:

```
FROM, TO, USING, FROM & TO
```

Data Division Level Numbers:

```
01, 02, 03, . . . , 48, 49
```

COBOL data-names:
Data-names may be up to 30 characters long. The legal data-name characters are: A—Z, 0—9, dash, and dollar sign.

Z Jump Cursor to End of Line

Moves the cursor to the end of a line. Like the JUMP command, "Z" causes an absolute jump.

Format

```
[#] Z
```

Arguments

#	(optional: default value = 24) The line to which the cursor is to be moved.

Example 1

```
Z$
```

Moves the cursor to line 24, column 80.

Example 2

```
13Z$
```

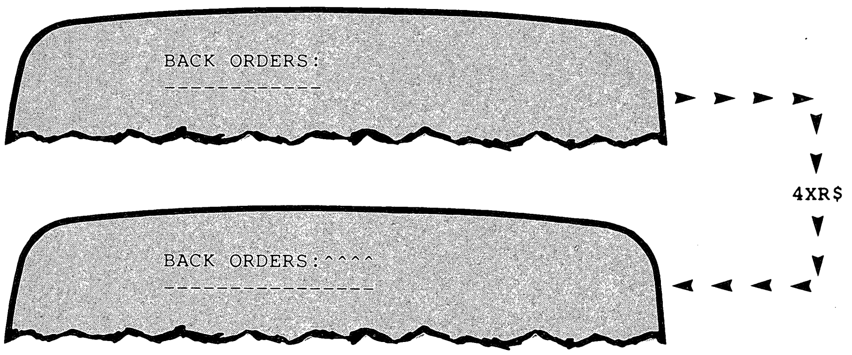
Moves the cursor to line 13, column 80.

Example 3

```
.Z$
```

Moves the cursor to the end of the current line.

Example 1



The source code for this field will include the following clause:

```
VALUE IS "BACK ORDERS:    ".
```

X Extend Field with Spaces

Adds blanks to the beginning or end of a literal field, or creates a literal field consisting entirely of blanks. EXTEND is the only ORGANIZE mode command that adds characters to the screen image—only blanks may be added. These blanks appear on the screen as "^"s, and are coded in the VALUE IS clause when you create source code for the screen image.

Creating Blank Fields

Literal fields consisting entirely of blanks are useful for erasing small portions of the display terminal screen. You must explicitly create such fields with the "X" command, then add them to groups as you do any other field.

To create a blank field, use one of the following procedures:

- Issue the "X" command to create a single-character blank field (VALUE IS " ") at the current cursor position. (The cursor must be pointing to a blank position on the screen, not to a field.) Reposition the cursor and reissue the command to create a blank field longer than one character.
- Alternatively, use the "#XL" and "#XR" commands to create long blank fields. These commands implicitly execute the command "X" first—that is, they create a one-character blank field, then extend it by # spaces to the left or right. The result is a blank field #+1 characters long.

Format

$$[\#]\ \mathrm{X} \begin{Bmatrix} \mathrm{L} \\ \mathrm{R} \end{Bmatrix}$$

Arguments

#	(optional: default value = 1) Number of blanks to be added to the field.
L	Extend with blanks on the left (leading blanks).
R	Extend with blanks on the right (trailing blanks).

U UNITE Fields

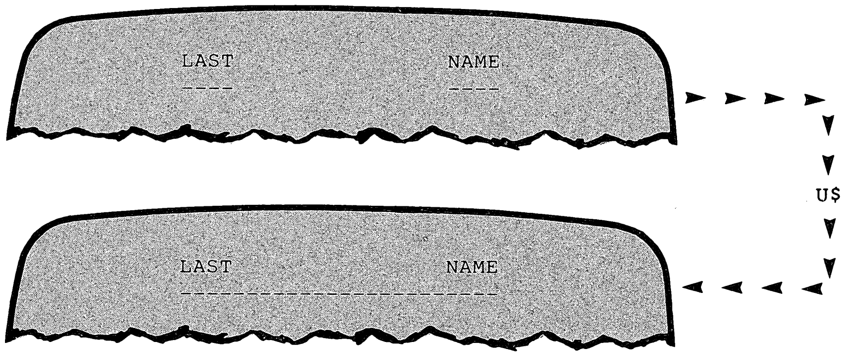
Cancels all SPLITing that you have performed on a field. The field is restored to its state at the beginning of the current ORGANIZE pass. The field's descriptors are restored.

You may place the cursor anywhere within the original limits of the field. The original field is restored *in its entirety*—this command cannot rejoin two parts of a field that has been split into three or more parts.

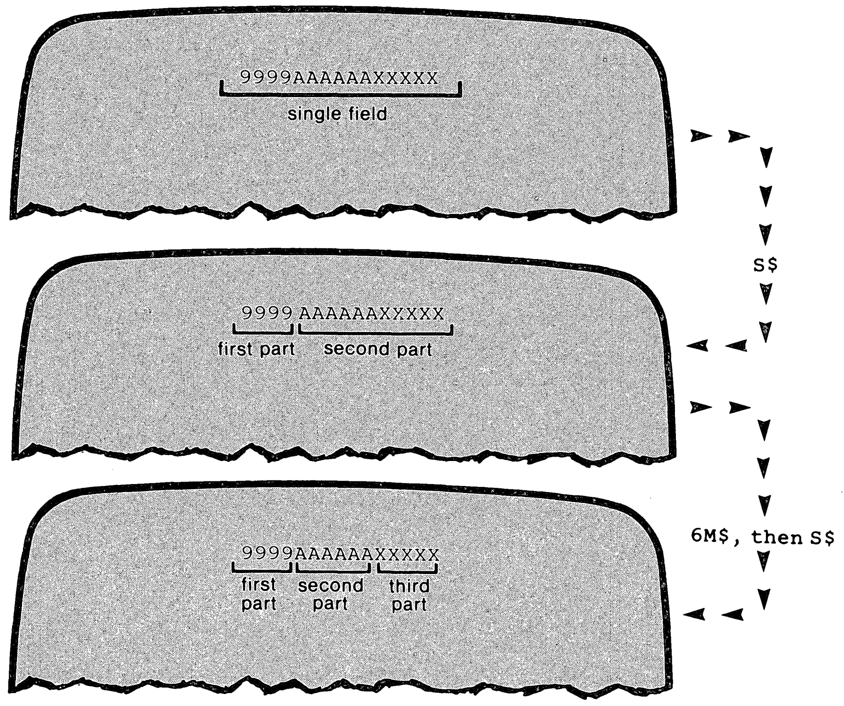
Format

```
U
```

Example 1



Example 2


9999AAAAAAXXXXX
single field
S$
9999 AAAAAAXXXXX
first part
second part
6M$, then S$
9999AAAAAAXXXXX
first part
second part
third part

S SPLIT Field

Splits a literal or format field into two distinct, but contiguous fields. This function is often useful since SCREEN considers two or more consecutive literal fields (no intervening format fields) on the same line to be a single field with imbedded blanks. SPLIT allows you to make a separate field of each of the parts.

Similarly, SCREEN considers any consecutive string of format characters to be a single format field, unless you explicitly SPLIT it.

To split a literal or format field into two parts, place the cursor at the first character of the second part and issue the command "S". SCREEN strips off leading and trailing blanks from the two parts. That is:

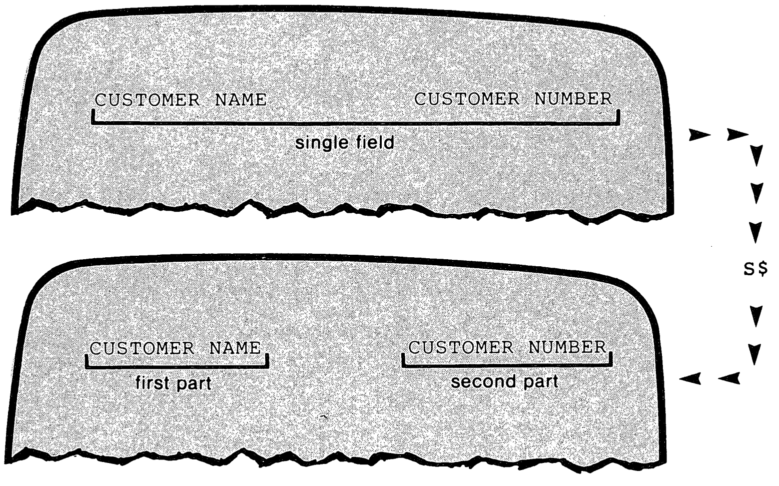
- The first field ends at the first non-blank character before the split location.
- The second field begins at the first non-blank character after the split location.

The two split fields retain none of the descriptor data of the original field, but all the descriptors reappear if the SPLIT fields are rejoined with the "U" command. You may split a field into any number of parts with repeated use of the "S" command at different positions in the field.

Format

S

Example 1



Suppose you began organizing the fields in preparation for writing code in the first structure. After having completed *subgroup-1*, you are adding fields to *subgroup-2* when you decide to switch to the second coding structure. You could use the "R" command to remove the fields on lines 12 and 13 from *subgroup-2*. But there is no way to change *subgroup-1*, which already has been closed. (All is not lost, however—you can bypass unwanted fields in CREATE SOURCE mode.)

R REMOVE Field from Group

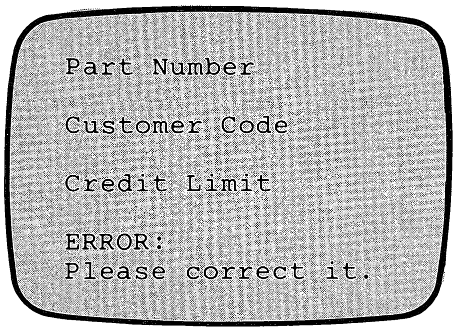
Deletes the field at the cursor position from the currently open group, and stops its blinking. If you have SPLIT a field after adding it to the current group, you may not REMOVE any of its parts unless you first rejoin the parts into the original field. (See the "S" and "U" command descriptions.)

Format

```
R
```

Example 1

Illustrated below are the fields of a screen image, along with two screen-element groups that you might build using the fields:



```
Part Number

Customer Code

Credit Limit

ERROR:
Please correct it.
```

```
01 <overall-group> <--------------------------- Structure-1
   03 <subgroup-1>
      05 LINE 4  VALUE IS "Part Number".
      05 LINE 12 VALUE IS "ERROR:".
      05 LINE 13 VALUE IS "Please correct it.".
   03 <subgroup-2>
      05 LINE 6  VALUE IS "Customer Code".
      05 LINE 12 VALUE IS "ERROR:".
      05 LINE 13 VALUE IS "Please correct it.".
   03 <subgroup-3>
      05 LINE 9  VALUE IS "Credit Limit".
      05 LINE 12 VALUE IS "ERROR:".
      05 LINE 13 VALUE IS "Please correct it."

01 <overall-group> <--------------------------- Structure-2
   03 <subgroup-a>
      05 LINE 4 VALUE IS "Part Number".
   03 <subgroup-b>
      05 LINE 6 VALUE IS "Customer Code".
   03 <subgroup-c>
      05 LINE 9 VALUE IS "Credit Limit".
   03 <subgroup-d>
      05 LINE 12 VALUE IS "ERROR:".
      05 LINE 13 VALUE IS "Please correct it."
```

M MOVE Cursor by Characters

Moves the cursor by characters relative to its current position. This is unlike the "J" command, which provides an absolute jump capability. For purposes of this command, the last column of a line "wraps around to" (is logically followed by) the first column of the next line. There is no "wraparound", however, connecting the first and last positions: you can never send the cursor beyond line 24, column 80 with a forward MOVE, nor before line 1, column 1 with a backward MOVE.

Format

```
[#] M
```

Arguments

#	(optional: default value = 1) Number of characters to move. If # is positive, the cursor moves forward. If # is negative, the cursor moves backward.

Example 1

```
80M$
```

Moves the cursor to the same column of the next line.

Example 2

```
1L-1M$
```

Moves the cursor to the last column of the current line.

L Move Cursor by LINES

Moves the cursor by lines relative to its current position. This is unlike the "J" command, which provides an absolute jump capability.

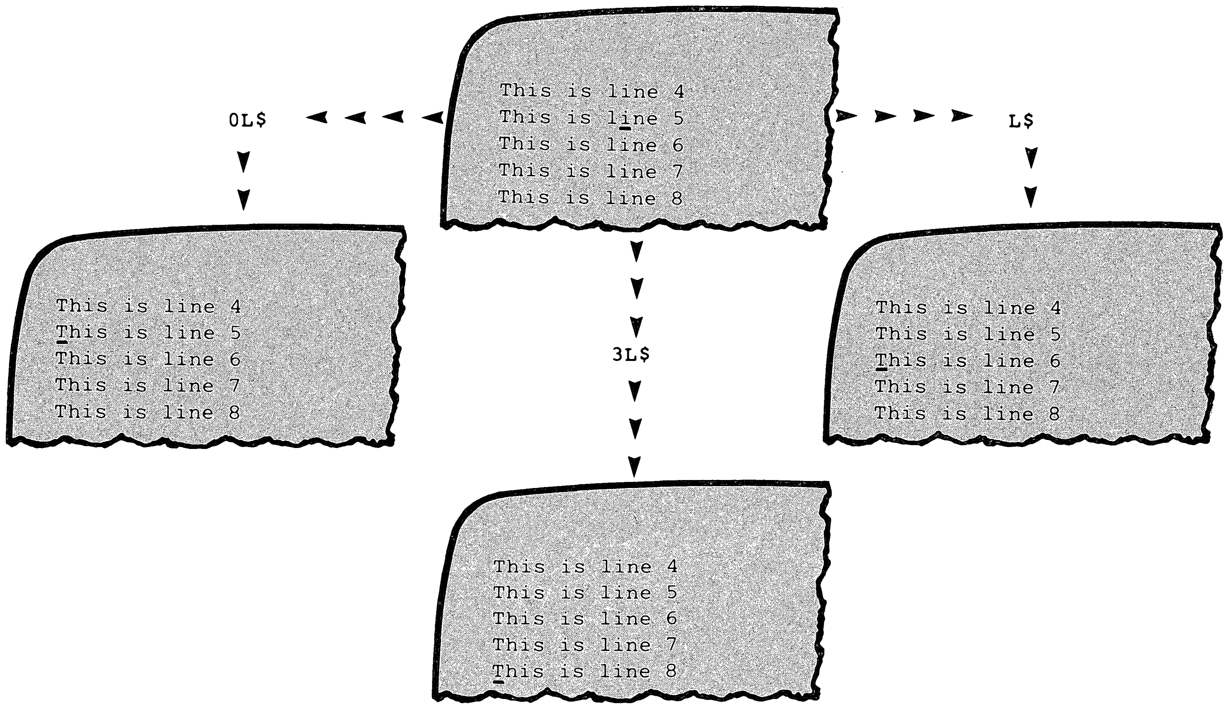
Format

```
[#] L
```

Arguments

#	(optional: default value = 1) If # = 0, the cursor moves to the beginning of the current line. If # is positive, the cursor moves to the beginning of the #th line forward. If # is negative, the cursor moves to the beginning of the #th line backward. There is no "wraparound" effect: you can never send the cursor beyond line 24 with a forward move, nor before line 1 with a backward move.

Examples



J JUMP Cursor to New Position

Moves the cursor to a new position in preparation for inserting text, replacing text, and so on. (You may also use the arrow keys, **CR** and **NEWLINE** to position the cursor.) JUMPing is absolute, not relative: the command has the same effect regardless of the current cursor position.

A column value larger than 80 causes a "wraparound": for instance, the command "3,95J" is equivalent to "4,15J". However, the end of the screen does not wrap around to the beginning: if the argument values would place the cursor beyond the end of the screen, it stops at line 24, column 80.

The variable arguments "." (representing the current line number) and ";" (representing the current column number) are useful with this command. See the examples below.

Format

$$\left[\begin{Bmatrix} \texttt{a} \\ \texttt{a,b} \end{Bmatrix} \right]^{\texttt{J}}$$

Arguments

a,b	(optional: default values: *a* = 1, *b* = 1) The cursor moves to line *a*, column *b*.

Example 1

```
12J$
```
Moves the cursor to the beginning of line 12.

Example 2

```
12,50J$
```
Moves the cursor to line 12, column 50.

Example 3

```
.J$
```
Moves the cursor to the beginning of the current line.

Example 4

```
5,;J$
```
Moves the cursor vertically—to the same column of line 5.

H View HELP Screen

Displays a HELP screen for any of the ORGANIZE mode commands. The "H" and "HH" commands display an alphabetical command summary. (Pressing **f11** does the same.)

To continue processing after viewing a HELP screen, press any key. This key is not interpreted as a command character.

Format

```
H [char]
```

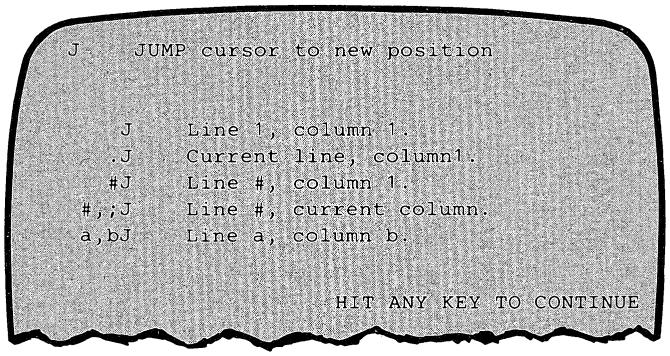
Arguments

char	An ORGANIZE mode command character. If you omit *char*, SCREEN displays an alphabetical summary of all EDIT mode commands.

Example 1

HJ$

Displays a "help" screen for the "J" command.



```
J    JUMP cursor to new position


     J    Line 1, column 1.
    .J    Current line, column1.
    #J    Line #, column 1.
  #,;J    Line #, current column.
  a,bJ    Line a, column b.


                      HIT ANY KEY TO CONTINUE
```

```
01 <screen name to be added in CREATE SOURCE mode>
   03 <screen name to be added in CREATE SOURCE mode>
      05 LINE  4 VALUE IS "Part Number".
      05 LINE 12 VALUE IS "ERROR:".
      05 LINE 13 VALUE IS "Please correct it.".
   03 <screen name to be added in CREATE SOURCE mode>
      05 LINE  6 VALUE IS "Customer code".
      05 LINE 12 VALUE IS "ERROR:".
      05 LINE 13 VALUE IS "Please correct it.".
```

To build this group structure, you:

1. Issue the command "1G" to define the 01-level header.
2. Issue the command "3G" to define the first 03-level header.
3. Add the three fields to the first 03-level group using the "A" command.
4. Issue the command "3G" to define the second 03-level header.
5. Add the three fields to the second 03-level group using the "A" command.
6. Close the group, with by issuing another "1G" command, or with the "Exit" command.

G Create New GROUP

Creates an "empty slot" that will be coded as a non-display screen element (Screen Section source line). Examples of such elements include both names of screen groups and instructions to use the terminal's blanking and BELL functions:

```
01 MASTER-SCREEN.
05 PASSWORD-SCREEN SECURE.
05 BLANK SCREEN.
05 BLANK LINE BELL.
```

The cursor position when you issue the "G" command is irrelevant, but the *order* in which you create groups and add fields is the order in which SCREEN writes code in CREATE SOURCE mode.

The argument # specifies the "slot's" level number. If you do not specify a level number argument, SCREEN computes one itself, as described below.

Creating a group implicitly closes the previously open group. All fields in the group just closed stop blinking.

Format

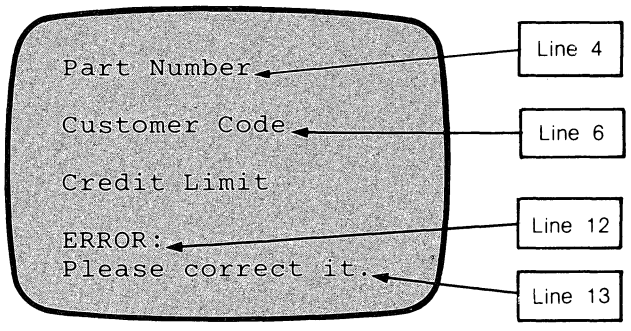
```
[#] G
```

Arguments

#	Level number for the field. If you omit #, SCREEN computes a value according to the following rules: ● If this is the first group, # = 05. ● If other groups have been created, # = previous group's level number.

Example 1

Illustrated below are the fields of a screen image, along with a screen-element group that you might define using the fields:



E EXIT to EDIT Mode

Ends the ORGANIZE session, saving your work in the descriptor file you named. You return to EDIT mode with an empty screen.

Format

```
    E
```

D DELETE Leading or Trailing Blanks

Deletes leading/trailing blanks from a literal field. In ORGANIZE mode, DELETE affects only leading and trailing blanks created with the EXTEND command or hand-coded in an ANALYZEd screen format. DELETE does not erase visible characters from a field.

The command affects the field at the cursor position. You may place the cursor anywhere in the field—the command syntax—not the exact cursor position—indicates where to make the deletion.

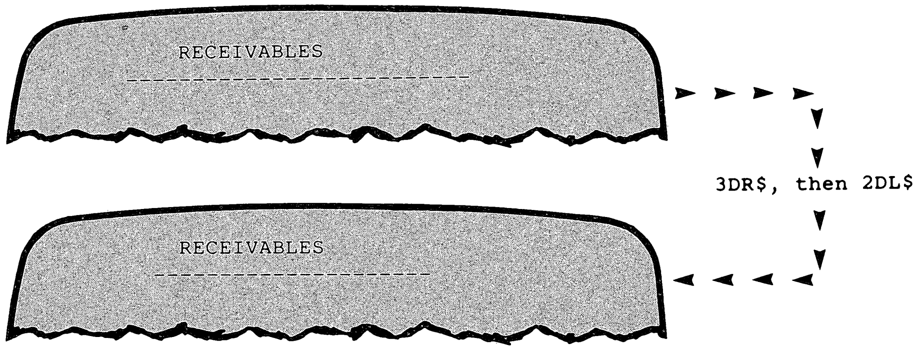
Format

$$[\#]\ \mathrm{D}\begin{Bmatrix}\mathrm{L}\\\mathrm{R}\end{Bmatrix}$$

Arguments

#	(optional: default value = 1) The number of leading/ trailing blanks to be deleted from the field.
L	Delete blanks to the left of the field (leading blanks).
R	Delete blanks to the right of the field (trailing blanks).

Example 1



```
01 <screen name to be added in CREATE SOURCE mode>
   03 <screen name to be added in CREATE SOURCE mode>
      05 LINE 4 VALUE IS "Part Number".
      05 LINE 12 VALUE IS "ERROR:".
      05 LINE 13 VALUE IS "Please correct it.".
```

A *ADD Field to Group*

Includes the field at the cursor position in the currently open group. The field blinks until you remove it from the group or close the group. The argument # specifies the field's level number. SCREEN stores level numbers under the descriptor-file name that you specified upon entering ORGANIZE mode. SCREEN includes the level number in the source line describing the field when you use the descriptor file to CREATE SOURCE for the screen image.

SCREEN will not write a level number larger than 49, since this is the largest level number recognized by the Interactive COBOL compiler. If you do not specify a level number argument, SCREEN computes one itself, as described below.

You may add the same field(s) to several groups. The example below shows how you might organize a screen image to prepare for creating source code that repeatedly uses an error-message field.

Format

```
[#] A
```

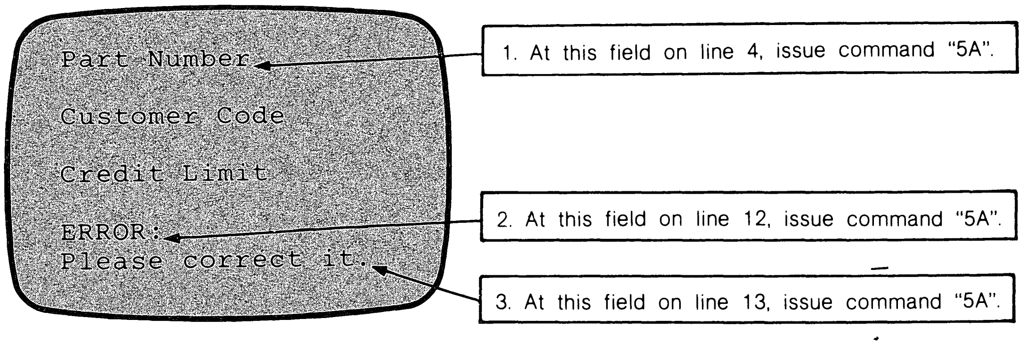
Arguments

#	Data division level number. If you omit #, SCREEN computes a value according to the following rules: ● If no group has been defined, # = 10. ● If this is the first field to be ADDed to a group, # = 2 ⋆ group level number. ● If other fields have been ADDed to the group, # = previous field's level number.

Example 1

Illustrated below are the fields of a screen image, along with one screen-element group that you might define using the fields.

To build the group, you first use the "G" command to define the 01-level and 03-level headers. (You will add screen names later, in CREATE SOURCE mode.) Then you position the cursor to the individual fields, issuing the command "5A" at each one.



Part Number
Customer Code
Credit Limit
ERROR:
Please correct it.
1. At this field on line 4, issue command "5A".
2. At this field on line 12, issue command "5A".
3. At this field on line 13, issue command "5A".

Error Handling

If you issue an incorrect command in ORGANIZE mode, SCREEN responds with an error display, similar to that in EDIT mode. See Appendix D for a listing of SCREEN error messages.

COMMAND REFERENCE

This section presents complete descriptions of the ORGANIZE mode commands, in alphabetical order.

ORGANIZE MODE COMMANDS

ORGANIZE mode commands use command characters, numeric arguments, and filename arguments in the same way as EDIT mode commands. In ORGANIZE mode, however, you terminate commands with a single **ESC** only. This means that you can't string commands together into editing routines. After you execute a command, you may reposition the cursor and reexecute the command by pressing **ESC** again.

At any time, press **f11** to display an alphabetical summary of ORGANIZE mode commands, then press any key to continue ORGANIZE mode processing.

ORGANIZE mode commands fall into the following categories:

- CURSOR POSITIONING COMMANDS move the cursor around the screen image. (These commands behave the same way in ORGANIZE mode as in EDIT mode.)
- GROUPING COMMANDS add and remove fields from groups.
- FIELD MODIFICATION COMMANDS split fields, unite fields, and add or delete spaces from the ends of fields.
- UTILITY CONTROL COMMANDS display help screens and terminate an ORGANIZE pass.

A summary of ORGANIZE mode commands, grouped by category, appears in Figure 5.5.

Immediate-action Keys

Many of the immediate-action keys perform the same functions in ORGANIZE mode as in EDIT mode. Figure 5.5 lists those keys that are enabled in ORGANIZE mode.

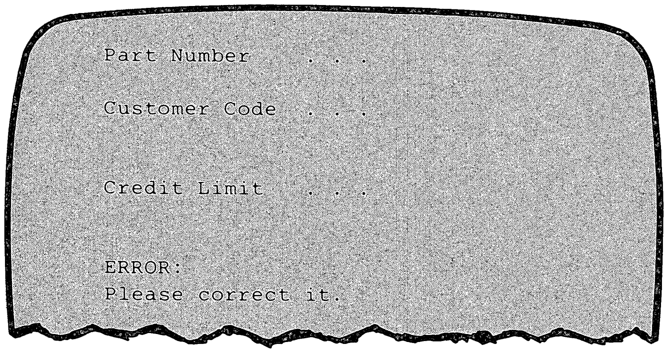
DISPLAY FIELD FORMATTING	
A	Add a field to the currently open group
D	Delete leading/trailing spaces from a field
G	Create and open a new group
R	Remove a field from the currently open group
S	Split a field into two parts
U	Unite the parts of a split field
X	Extend a field with leading/trailing blanks
UTILITY-CONTROL COMMANDS	
E	Exit from ORGANIZE mode; return to EDIT mode
H	HELP
IMMEDIATE-ACTION KEYS	
FN9	Erase command
FN11	Display HELP screen for ORGANIZE mode commands
DEL	Erase last command character
HOME	Terminate SCREEN

FIGURE 5.5 ORGANIZE MODE COMMANDS. The organize mode commands are summarized here, categorized by their function. The use of the cursor-positioning keys and the function keys is also included.

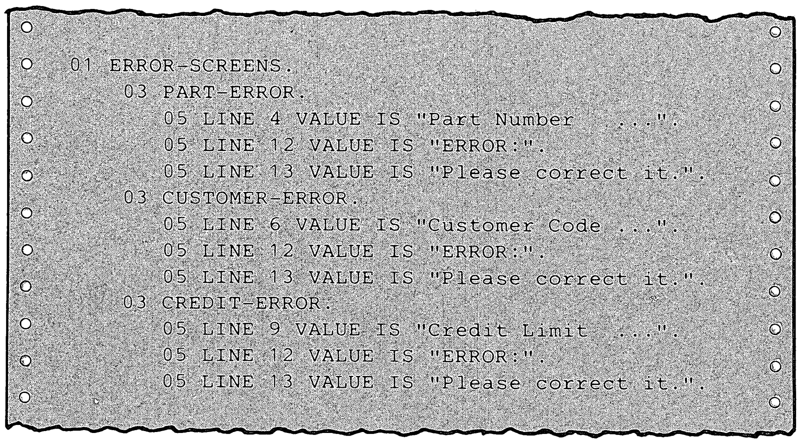
Building a Group—An Example

Figure 5.4 shows a screen image consisting of several literal fields, along with one possible organization of these fields into a structured group. Note that the error message on lines 12 and 13 has been included in three different groups. The group ERROR-SCREENS can be built using the "A" and "G" commands in combination:

1. Issue command "1G" to define a "slot" for the header ERROR-SCREENS.
2. Issue command "3G" to define a "slot" for the header PART-ERROR.
3. Place cursor at field on line 4; issue command "5A".
4. Place cursor at field on line 12; issue command "5A" or "A".
5. Place cursor at field on line 13; issue command "5A" or "A".
6. Issue command "3G" to define a "slot" for the header CUSTOMER-ERROR.
7. Place cursor at field on line 6; issue command "5A".
8. Place cursor at field on line 12; issue command "5A" or "A".
9. Place cursor at field on line 13; issue command "5A" or "A".
10. Issue command "3G" to define a "slot" for the header CREDIT-ERROR.
11. Place cursor at field on line 9; issue command "5A".
12. Place cursor at field on line 12; issue command "5A" or "A".
13. Place cursor at field on line 13; issue command "5A" or "A".

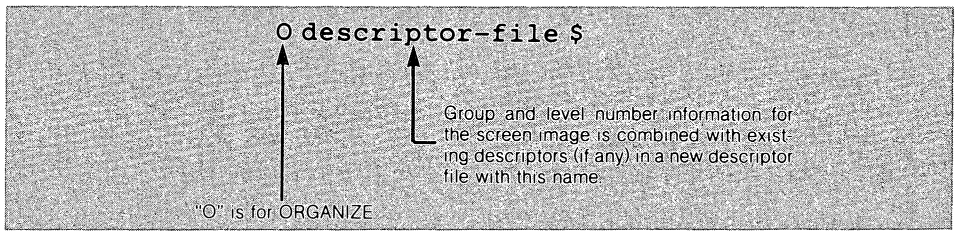


```
Part Number     . . .

Customer Code   . . .


Credit Limit    . . .


ERROR:
Please correct it.
```



```
01 ERROR-SCREENS.
    03 PART-ERROR.
        05 LINE 4 VALUE IS "Part Number     ...".
        05 LINE 12 VALUE IS "ERROR:".
        05 LINE 13 VALUE IS "Please correct it.".
    03 CUSTOMER-ERROR.
        05 LINE 6 VALUE IS "Customer Code ...".
        05 LINE 12 VALUE IS "ERROR:".
        05 LINE 13 VALUE IS "Please correct it.".
    03 CREDIT-ERROR.
        05 LINE 9 VALUE IS "Credit Limit   ...".
        05 LINE 12 VALUE IS "ERROR:".
        05 LINE 13 VALUE IS "Please correct it.".
```

FIGURE 5.4 ADDING THE SAME FIELDS TO SEVERAL GROUPS. The same fields may be incorporated in several parts of a group hierarchy. In this example, the error message on lines 12 and 13 is included in three different groups.



```
O descriptor-file $
```

Group and level number information for the screen image is combined with existing descriptors (if any) in a new descriptor file with this name.
"O" is for ORGANIZE

FIGURE 5.3 ENTERING ORGANIZE MODE. Use the "O" command in EDIT mode to instruct SCREEN to enter ORGANIZE mode.

Creating a Group

The "GROUP" command terminates the currently open group and creates a "slot" for the header of a new group. For example, the command "3G" instructs SCREEN to store the following provisional source line:

```
03.
```

This is not enough information to build an actual source line. When you create source code for this ORGANIZEd screen image, SCREEN will require that you assign a screen name and/or terminal function descriptors to this provisional source line.

Creating Non-Display Elements

You may also use the "G" command to create a slot for a non-display screen element. For example:

```
03 DATA-HEADS.
  05 BLANK SCREEN BELL.<-------------------- This is a non-display element.
  05 LINE 01 COL 20 VALUE IS "CLEARWATER INDUSTRIES".
  05 LINE 02 COL 20 VALUE IS "*********************".
  05 LINE 04 COL 20 VALUE IS "CUSTOMER KEY".
```

The command "5G" creates a "slot" for this element at the 05 level. You can specify the terminal functions BLANK SCREEN and BELL when you create source code for the ORGANIZEd screen image.

Adding a Field to a Group

After you have defined a group header with the "G" command, you may add one or more display fields to the group with the "ADD" command. This command assigns a level number to the field at the cursor position and adds it to the currently open group. (See the "Command Reference" section for details.) SCREEN blinks the added field to confirm its addition to the group; all fields in the current group blink until you terminate the group, either by defining a new group or by exiting from ORGANIZE mode.

You may include both display fields and "non-display elements" (see the previous section) in the same group. After defining the group, execute "G" and "A" commands in order, using the same level number.

You may add the same field to more than one group. The field may have a different level number in different groups.

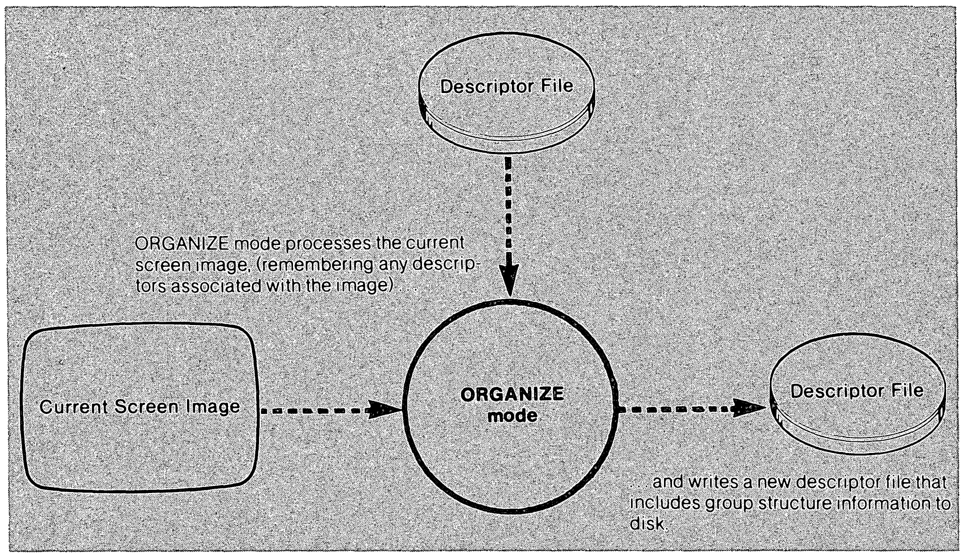

Descriptor File
ORGANIZE mode processes the current screen image, (remembering any descriptors associated with the image) . . .
Current Screen Image
ORGANIZE mode
Descriptor File
. . . and writes a new descriptor file that includes group structure information to disk.

FIGURE 5.2 DATA FLOW IN ORGANIZE MODE. In ORGANIZE mode, SCREEN creates hierarchical structures for the screen image, storing this information in a descriptor file.

THE DISPLAY SCREEN

During an ORGANIZE pass, the current screen image is always displayed, and the cursor is always visible. As you position the cursor to a field, SCREEN underscores the field.

As you create groups of fields, SCREEN *blinks* those fields in the current group. This helps you to keep track of the contents of the current group. Opening a new group ("G" command) implicitly terminates the current group. The fields of the terminated group stop blinking; you may not add any more fields to it during the current ORGANIZE pass.

ENTERING AND EXITING ORGANIZE MODE

To enter ORGANIZE Mode, issue the "O" command in EDIT mode. If you retrieved the current screen image from a descriptor file with the "G" command, you may not use this file's name in the "O" command. Figure 5.3 illustrates the command format.

To exit from ORGANIZE mode, issue the "E" command. SCREEN saves your work in the descriptor file you named in the "O" command.

CREATING GROUPS/ADDING FIELDS TO GROUPS

In ORGANIZE mode, you create groups and add display fields to them. Both the "G" command (to create a group) and the "A" command (to add a field to a group) assign a level number to a screen element. Using these two commands, you can build a hierarchical structure with as much (or as little) complexity as you please. When you create source code for the ORGANIZED screen image, SCREEN will code groups/fields in the order in which you created/added them.

5 Organize Mode

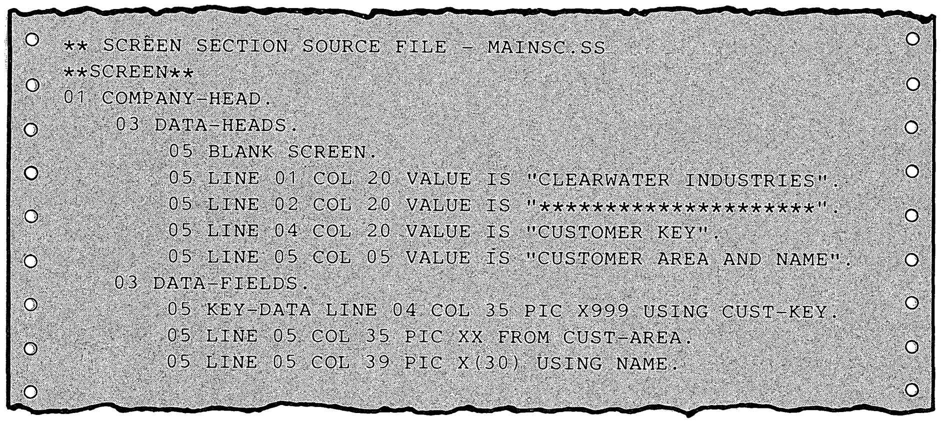
In ORGANIZE mode, you assign COBOL Data Division structure to a screen image. This includes creating groups, selecting fields to include in the groups, and assigning level numbers to screen elements. Figure 5.1 shows two source files created by SCREEN from the same screen image—one was built with an ORGANIZEd image, the other wasn't.

This illustration shows that ORGANIZE mode allows you to create screen elements not associated with any data field: group headers like "01 COMPANY-HEAD" and control elements like "05 BLANK SCREEN".

ORGANIZE mode also allows you to "split" fields into parts that may be addressed independently by DISPLAY and ACCEPT statements within a COBOL program.

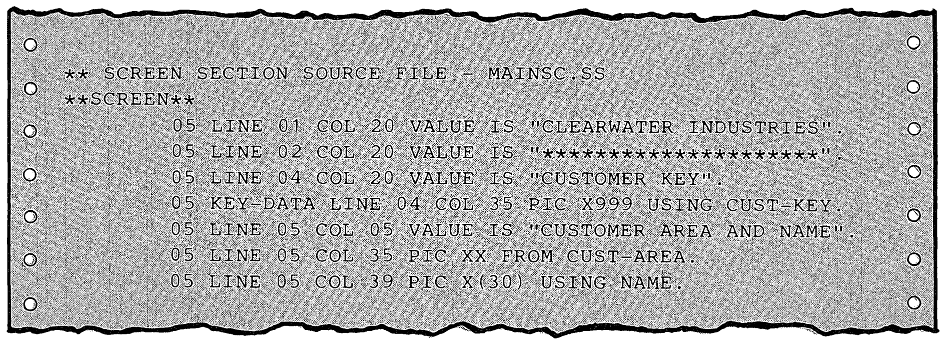
The information that you add to a screen image in ORGANIZE mode is stored in a descriptor file. Figure 5.2 illustrates the flow of information to and from disk files in ORGANIZE mode.

ORGANIZEd Code



```
** SCREEN SECTION SOURCE FILE - MAINSC.SS
**SCREEN**
01 COMPANY-HEAD.
    03 DATA-HEADS.
         05 BLANK SCREEN.
         05 LINE 01 COL 20 VALUE IS "CLEARWATER INDUSTRIES".
         05 LINE 02 COL 20 VALUE IS "*********************".
         05 LINE 04 COL 20 VALUE IS "CUSTOMER KEY".
         05 LINE 05 COL 05 VALUE IS "CUSTOMER AREA AND NAME".
    03 DATA-FIELDS.
         05 KEY-DATA LINE 04 COL 35 PIC X999 USING CUST-KEY.
         05 LINE 05 COL 35 PIC XX FROM CUST-AREA.
         05 LINE 05 COL 39 PIC X(30) USING NAME.
```

unORGANIZEd Code



```
** SCREEN SECTION SOURCE FILE - MAINSC.SS
**SCREEN**
         05 LINE 01 COL 20 VALUE IS "CLEARWATER INDUSTRIES".
         05 LINE 02 COL 20 VALUE IS "*********************".
         05 LINE 04 COL 20 VALUE IS "CUSTOMER KEY".
         05 KEY-DATA LINE 04 COL 35 PIC X999 USING CUST-KEY.
         05 LINE 05 COL 05 VALUE IS "CUSTOMER AREA AND NAME".
         05 LINE 05 COL 35 PIC XX FROM CUST-AREA.
         05 LINE 05 COL 39 PIC X(30) USING NAME.
```

FIGURE 5.1 STRUCTURED GROUPS OF SCREEN ELEMENTS. CLI/SCREEN allows you to create hierarchical groups of screen elements, so that portions of a screen format can function as separate units.

tors) from the source file. In the former case, SCREEN leaves the extracted image on the display screen and returns to EDIT mode. In the latter case, SCREEN stores the image and descriptors under the desciptor filename you specified in the ANALYZE command. The display screen is blanked, and SCREEN returns to EDIT mode. (To retrieve the extracted image for EDITing, issue the EDIT mode command "G*descriptor-file*".)

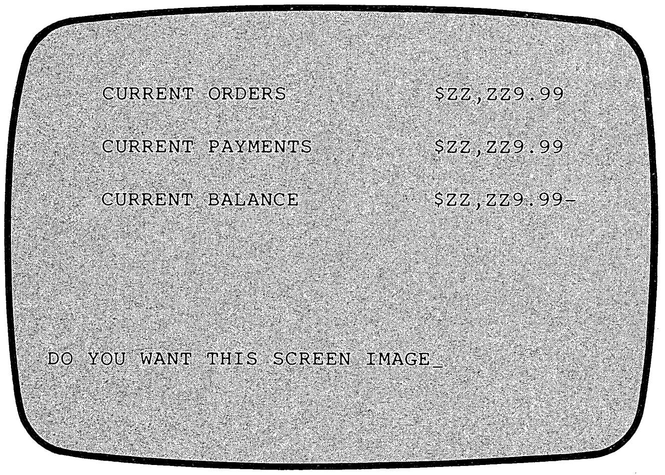
EXITING ANALYZE MODE

ANALYZE mode terminates automatically after you select a particular screen image, or after you have rejected every screen image extracted from the source file. SCREEN returns to EDIT mode with a blank screen if you extracted a complete screen format. If you chose to extract only the screen image, discarding descriptors, the image remains on the screen for editing.

You may also terminate ANALYZE mode when SCREEN asks "DO YOU WANT THIS SCREEN IMAGE"—press the **HOME** key to blank the screen and return to EDIT mode.

ERROR HANDLING

SCREEN is capable of detecting syntactic errors in the screen formats it ANALYZEs. If the source contains such an error, SCREEN suspends processing and displays an error message. Press any key to unfreeze the display; SCREEN returns to EDIT mode. (In some cases, SCREEN may be unable to return to EDIT mode, and the program terminates altogether.) See Appendix D for a list of SCREEN's error messages.



```
   CURRENT ORDERS              $ZZ,ZZ9.99

   CURRENT PAYMENTS            $ZZ,ZZ9.99

   CURRENT BALANCE             $ZZ,ZZ9.99-




DO YOU WANT THIS SCREEN IMAGE_
```

FIGURE 4.2 EXTRACTING A SCREEN FORMAT FROM A SOURCE FILE. SCREEN searches the source file for screen images that begin at the structure level you specify. You may instruct SCREEN to bypass images until it finds the one you want.

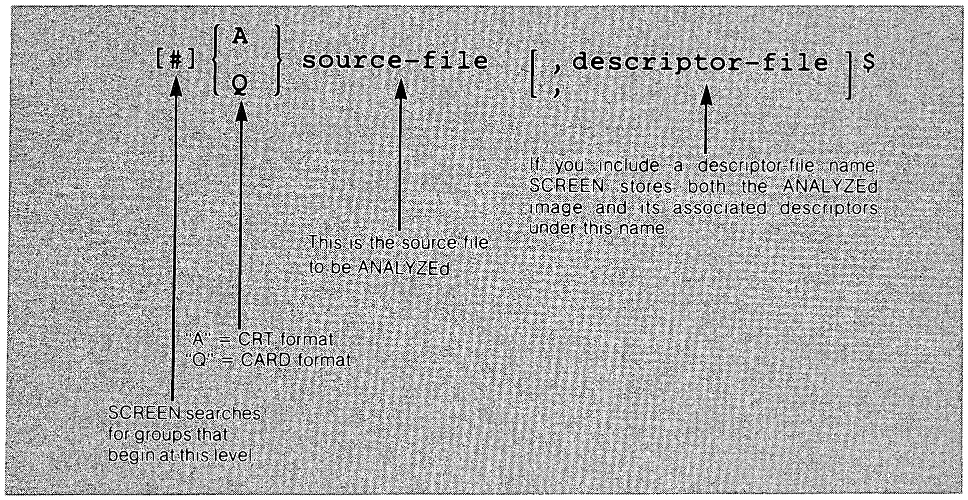
SELECTING A SCREEN IMAGE

In ANALYZE mode, SCREEN displays screen images, one at a time, from the source file named in the ANALYZE command. As it displays a screen image (see Figure 4.3), SCREEN asks:

```
DO YOU WANT THIS SCREEN IMAGE_
```

- Press **Y** to accept the screen image currently displayed.
- Press **ESC** to tell SCREEN to find the next screen image at the specified level.

When you indicate the screen image of your choice by pressing **Y**, SCREEN extracts either the image alone or the entire screen format (image and descrip-



```
[#] { A }  source-file  [ , descriptor-file ] $
    { Q }               [ ,                 ]
```

If you include a descriptor-file name, SCREEN stores both the ANALYZEd image and its associated descriptors under this name.
This is the source file to be ANALYZEd.
"A" = CRT format
"Q" = CARD format
SCREEN searches for groups that begin at this level.

FIGURE 4.3 ENTERING ANALYZE MODE. Use the "A" or "Q" command in EDIT mode to instruct SCREEN to extract a screen image or entire screen format from a source file.

descriptors. In this case, no descriptor file is created. You must supply new descriptor information when you create source code for the (modified) screen image.

EXTRACTING SUBGROUPS: THE LEVEL NUMBER ARGUMENT

The screen image that you want to extract might be defined as part of a larger group. For instance, FIRST-SCREEN and SECOND-SCREEN might be subscreens of MASTER-SCREEN:

```
SCREEN SECTION.
01   MASTER-SECTION.
          03 FIRST-SCREEN.
                  05 ...
                  05 ...
                  05 ...
          03 SECOND-SCREEN.
                  05 ...
                  05 ...
                       07 ...
                       07 ...
01 PRIMAL-SCREEN.
         03 ...
```

To allow you to extract subscreens from a source file, the ANALYZE command accepts a level-number argument. The argument # tells SCREEN to search for a group item that starts at level #. Subsequent lines with a higher level number are considered part of the same group. The first line encountered with a level number equal to or lower than # terminates the group.

If you omit the level number argument, SCREEN computes its own: one less than the first Screen Section level number found in the file.[1] This allows SCREEN to treat as one group the entire contents of a source file that has no group organization. (SCREEN writes such files in CREATE SOURCE mode if you haven't ORGANIZEd the fields into groups.)

ENTERING ANALYZE MODE

To enter ANALYZE mode from EDIT mode, issue the "A" command to process a source file in CRT format, or the "Q" command to process a source file in CARD format. If you wish to extract a complete screen format (image and descriptors), include the *descriptor-file* argument. If you omit this argument, SCREEN extracts only a screen image, discarding the descriptors. Figure 4.2 illustrates the command format.

You may select a default filename for the output descriptor file. This facility allows you to choose corresponding names for source and descriptor files. See "Default Filenames" in Chapter 1.

1 CAUTION: This strategy has the following consequence: If you specify no level number in ANALYZEing a complete COBOL program file, all screen images are extracted and superimposed on the display screen.

4 Analyze Mode

ANALYZE mode provides SCREEN users the capability to extract display screen formats from Interactive COBOL source files. This greatly speeds the process of adapting screen formats from an existing applications system for use in another system.

The ANALYZE command instructs SCREEN to extract a single screen format from the Screen Section of a line sequential source file. This file might contain a complete COBOL program or a COPY section (perhaps code previously written by SCREEN itself). You may extract screen images from both CRT format source files and CARD format source files. Figure 4.1 illustrates the flow of data to and from disk files in ANALYZE mode.

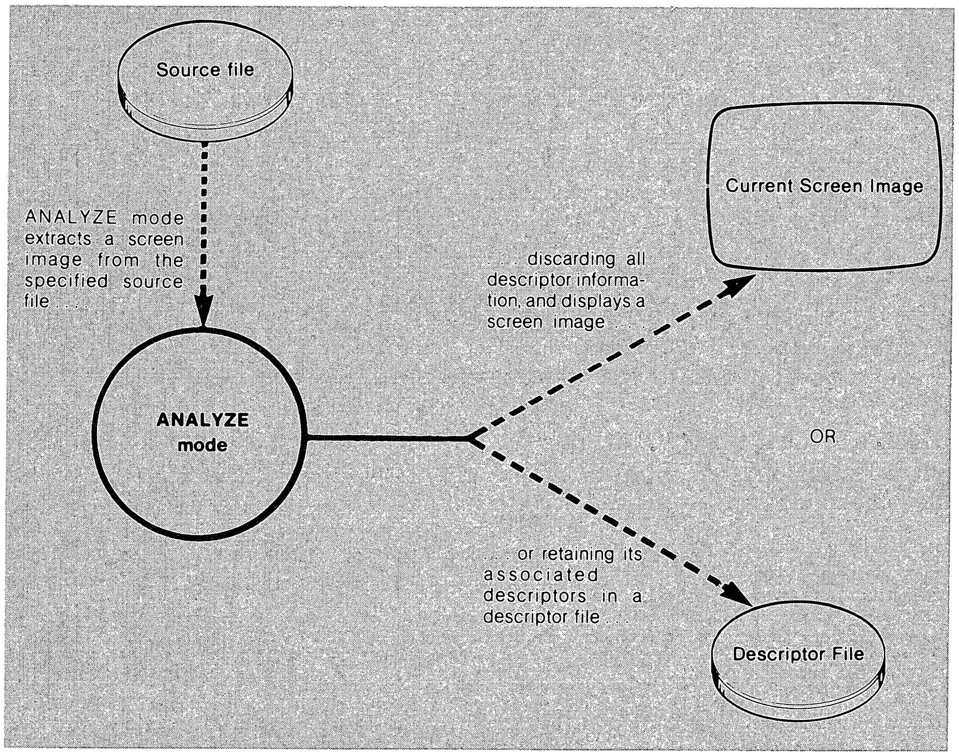

Source file
ANALYZE mode extracts a screen image from the specified source file . . .
ANALYZE mode
. . . discarding all descriptor information, and displays a screen image . . .
Current Screen Image
OR
. . . or retaining its associated descriptors in a descriptor file . . .
Descriptor File

FIGURE 4.1 DATA FLOW IN ANALYZE MODE. In ANALYZE mode, SCREEN extracts a display screen format from an existing Interactive COBOL source file. You may also specify a descriptor file to store the extracted screen format.

SCREEN allows you to extract an entire display screen format, storing both screen image and descriptors in a descriptor file. You may use the descriptor information later, when you create source code for the (modified) screen image.

Alternatively, you may choose to extract only a screen image, discarding the

For example, you may have created a group "slot" at the 05 level in ORGANIZE mode. The provisional source line is:

```
    05.
```

In CREATE SOURCE mode, you might fill the slot in any of the following ways:

- `05 ENTRY-PART-ONE.`
- `05 BLANK LINE.`
- `05 PASSWORD-SCREEN SECURE FULL AUTO.`

EXITING CREATE SOURCE MODE

CREATE SOURCE mode terminates automatically after the last field in the screen image has been coded: the screen is blanked and EDIT mode returns. You may also press the **HOME** key during the source creation dialogue to perform an immediate exit from SCREEN. In this case, the output source file will contain code for the fields you processed before performing the exit.

GETTING HELP

You may press **f11** at any time during the source creation dialogue to view a help screen. Press any key (except a function key) to return to the dialogue.

- Enter "BL" to add both the BLINK and BLANK LINE descriptors to the provisional source line.
- Enter "E-LS" to add the BELL and BLANK SCREEN descriptors, and to delete the BLANK LINE descriptor from the provisional source line.

GROUP	LITERAL FIELD	FORMAT FIELD
BLINK	BLINK	BLINK
BLANK LINE	BLANK LINE	BLANK LINE
BLANK SCREEN	BLANK SCREEN	BLANK SCREEN
BELL	BELL	BELL
SECURE		SECURE
AUTO		AUTO
REQUIRED		REQUIRED
FULL		FULL
		JUSTIFIED
		BLANK WHEN ZERO

FIGURE 3.5 SCREEN FORMAT DESCRIPTORS. SCREEN supports the full range of Interactive COBOL Screen Section descriptors. Certain descriptors are required to make the screen format a functioning part of a COBOL program. Others enhance operator-program communication.

Step 4 (Format Fields Only)

- If the format field has no data-item associated with it, SCREEN demands that you specify both an input/output mode and associated data item(s). You may not bypass this step:[1]

```
I/O MODE: FROM, TO, USING, BOTH FROM & TO
```

  - ☐ Enter one of the code letters and specify a data-item name when SCREEN prompts you.

- If a format field already has an associated data-item, SCREEN allows you to change the assignment:

```
ASSIGN NEW DATA NAMES?
```

  - ☐ Enter "A" to initiate the dialogue that assigns a data-item to the field.
  - ☐ Press **ESC** alone to leave the data-item assignment unchanged.

An empty group "slot" created in ORGANIZE mode must be assigned either a screen element name or a terminal function descriptor before SCREEN will create a source line.

1 Every format field must be assigned to an associated data-item in the FILE or WORKING-STORAGE section, along with an input/output mode: FROM, TO, USING, or FROM & TO. SCREEN will not write a source line until you make this assignment. The data-item may be qualified with an IN group-name or OF group-name clause.

☐ Enter "M" to choose the MODIFY option if the provisional source line currently displayed for the field is not the exact code you want.
☐ Enter "D" to choose the DELETE option, instructing SCREEN not to write a source line for the field. SCREEN moves to the next field, returning to step 1.
☐ Press **ESC** alone to instruct SCREEN to write the provisional source line currently displayed to the output file.
☐ If SCREEN needs more information to build a source line, it does not allow you the the third option. This occurs when a format field has not been assigned a data-item, or a "slot" created in ORGANIZE mode is still empty. In such cases, SCREEN asks:

```
DELETE THIS FIELD ?
```

Your only choices are to delete—not write source code for—the field (enter "D"), or to modify the field (press **ESC** only).

Step 2

- For each display field and group element, SCREEN asks:[1]

```
ASSIGN A SCREEN NAME TO THIS FIELD?
```

☐ To bypass screen name assignment, press **ESC** alone.
☐ To assign a name (or change the current name), enter "A", and then a data-item name at the ENTER SCREEN NAME prompt. (Be sure it's a valid name—SCREEN doesn't perform a validity check.)
☐ To remove a screen name, enter "A", and then press **ESC** alone at the ENTER SCREEN NAME prompt.

Step 3

SCREEN presents several "menus" of Interactive COBOL display terminal functions, tailored to the provisional group, literal, or format field source line. (See Figure 3.5.)

When SCREEN displays a menu, enter one (or more) of the underscored code letters to add the corresponding descriptor to the provisional source line. You may also delete descriptors from the source line, by typing a minus sign (-), followed by the appropriate code letter. SCREEN processes the code letters in order, from left to right.

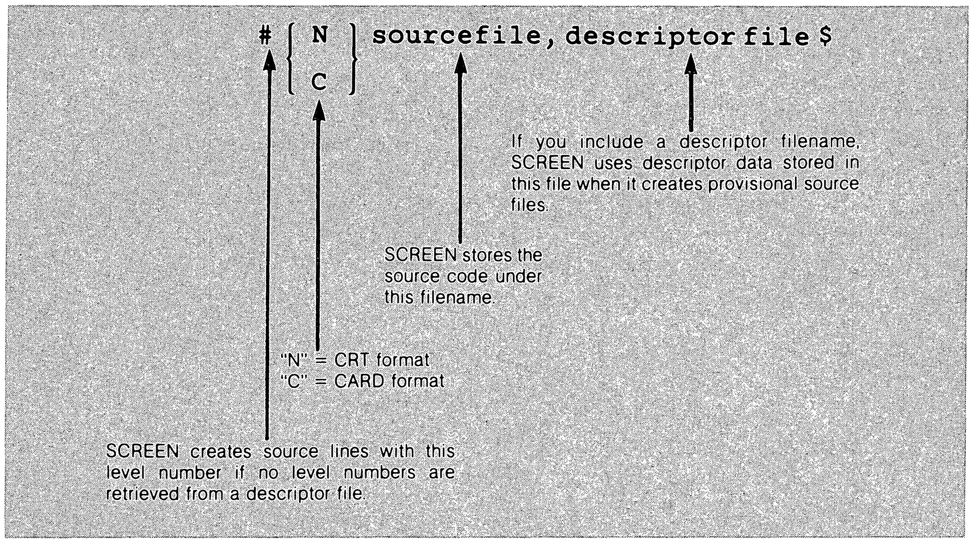
For example, one of the menus is:

```
BLINK, BLANK LINE, BLANK SCREEN, BELL
```

When SCREEN displays this menu, you might:

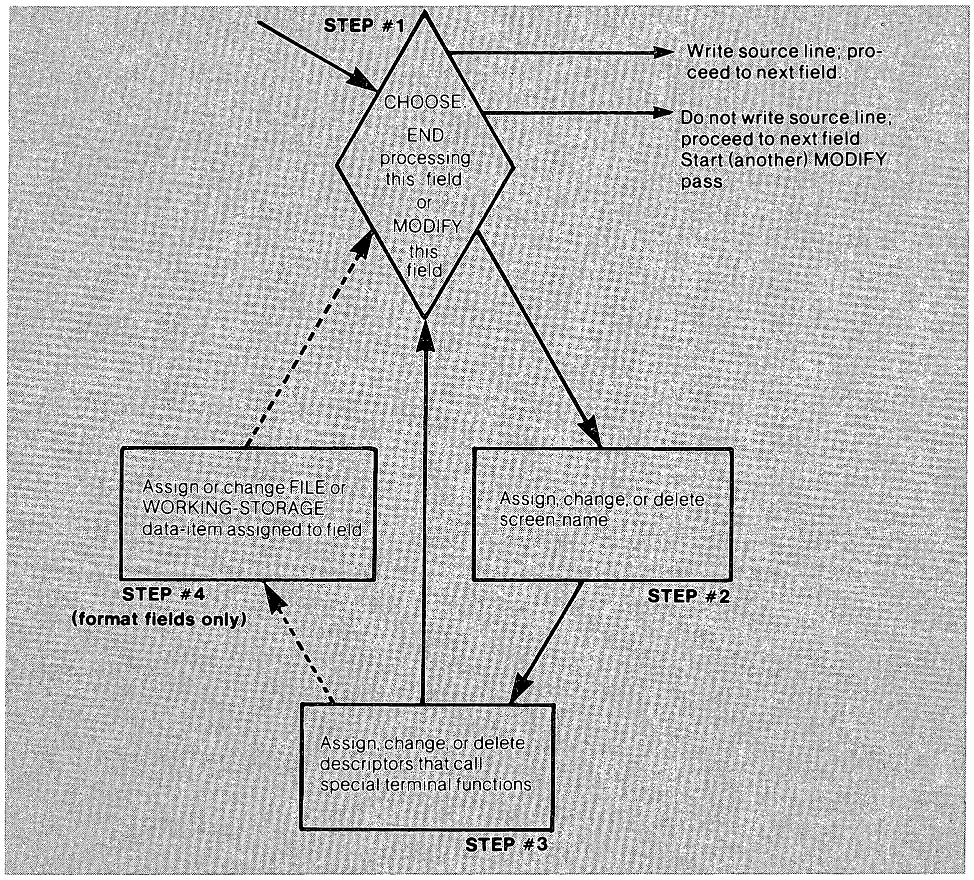
- Enter "E" to add the BELL descriptor to the provisional source line.

1 To reference a literal or format field individually within a source program, you must assign it a screen element name.



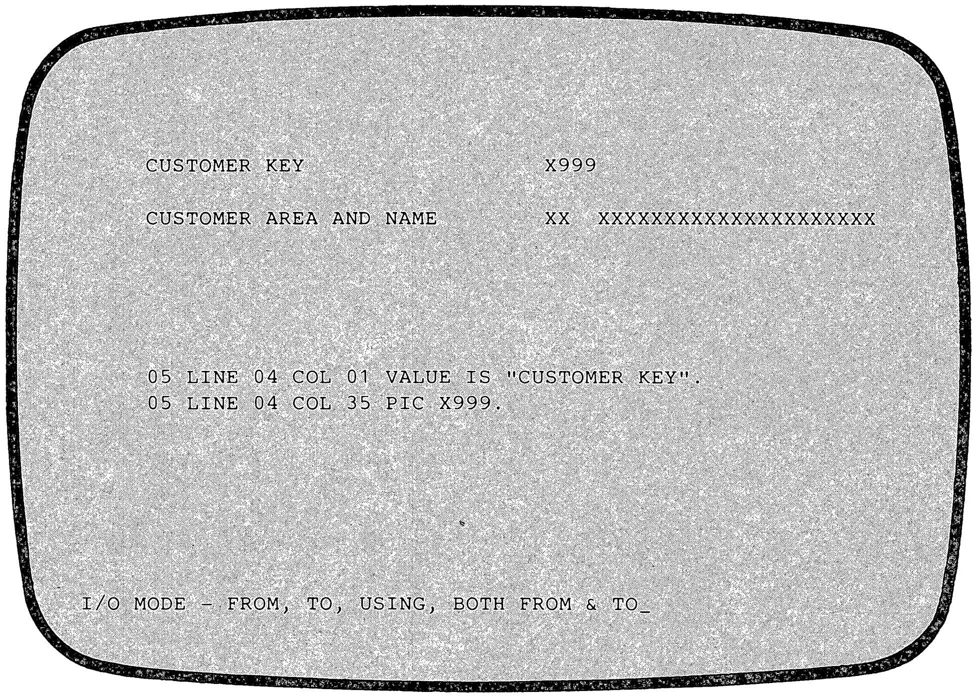
# { N } sourcefile, descriptor file $
  C
If you include a descriptor filename, SCREEN uses descriptor data stored in this file when it creates provisional source files.
SCREEN stores the source code under this filename.
"N" = CRT format
"C" = CARD format
SCREEN creates source lines with this level number if no level numbers are retrieved from a descriptor file.

FIGURE 3.3 ENTERING CREATE SOURCE MODE. Use the "C" or "N" command in EDIT mode to instruct SCREEN to create source code to define the current screen image.



STEP #1
CHOOSE: END processing this field or MODIFY this field
Write source line; proceed to next field.
Do not write source line; proceed to next field Start (another) MODIFY pass
Assign or change FILE or WORKING-STORAGE data-item assigned to field
STEP #4 (format fields only)
Assign, change, or delete screen-name
STEP #2
Assign, change, or delete descriptors that call special terminal functions
STEP #3

FIGURE 3.4 THE SOURCE CREATION DIALOGUE. Using the screen image at the time you issue the CREATE SOURCE command, SCREEN engages you in a dialogue that fully defines a display screen format.



```
CUSTOMER KEY                    X999

CUSTOMER AREA AND NAME          XX  XXXXXXXXXXXXXXXXXXXX




05 LINE 04 COL 01 VALUE IS "CUSTOMER KEY".
05 LINE 04 COL 35 PIC X999.





I/O MODE - FROM, TO, USING, BOTH FROM & TO_
```

FIGURE 3.2 SPECIFYING DESCRIPTORS FOR A FIELD. For each field in the screen image, SCREEN prompts you to specify descriptors that fully prescribe the corresponding line of source code.

ENTERING CREATE SOURCE MODE

To create source code in CRT format, issue the "N" command. To create source code in CARD format, issue the "C" command in EDIT mode. Figure 3.3 illustrates the command format.

THE SOURCE CREATION DIALOGUE

For each field in the screen image (and for each non-display element input from a descriptor file), SCREEN engages you in a dialogue consisting of several steps. (See Figure 3.4.) During this dialogue, SCREEN prompts you to enter information: one or more code letters, or a COBOL data-name. Enter your responses using a single **ESC** as a terminator. For some questions, pressing **ESC** alone is a meaningful response, indicating that you wish to select the default option.

Note that the dialogue is cyclical; SCREEN always gives you the option of making another "MODIFY" pass on a current field, before writing its source line to the output source file.

At any time, you may press **f11** to display a help screen for CREATE SOURCE operations, then press any key to continue the source creation dialogue.

Each step of the source creation dialogue is further explained below:

Step 1

- If SCREEN has enough information to build a source line, it asks:

```
MODIFY OR DELETE THIS <FIELD,GROUP ITEM>?
```

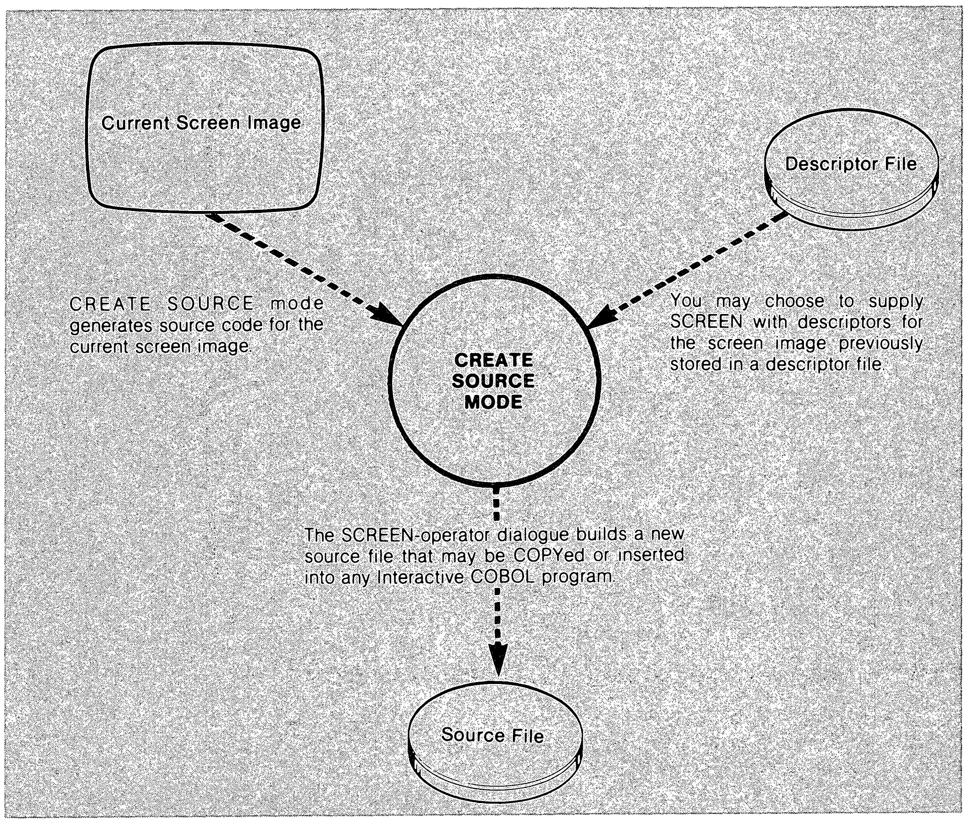

Current Screen Image
Descriptor File
CREATE SOURCE mode generates source code for the current screen image.
CREATE SOURCE MODE
You may choose to supply SCREEN with descriptors for the screen image previously stored in a descriptor file.
The SCREEN-operator dialogue builds a new source file that may be COPYed or inserted into any Interactive COBOL program.
Source File

FIGURE 3.1 DATA FLOW IN CREATE SOURCE MODE. In CREATE SOURCE mode, SCREEN writes source code for the current screen image. The code is written to a source file; you may also specify a descriptor file for input to the source creation process.

When you and SCREEN have determined the coding for every field, SCREEN returns automatically to EDIT Mode, with a blank screen.

THE "MISCELLANEOUS" GROUP

In your "create source" command, you may instruct SCREEN to use a descriptor file associated with the image to be coded. If the descriptor file contains level-number data, SCREEN writes group-structured code. In some cases, though, the descriptor file may contain level-number data for some fields, but not for others. For example, you might have:

1. Created a screen image in EDIT mode.
2. Assigned level numbers in ORGANIZE mode, creating descriptor file DESCRIP.DX.
3. Retrieved the image from DESCRIP.DX, and modified it.
4. Issued the command to create source code for the modified image, using the level numbers in DESCRIP.DX.

In this case, the fields you created or changed in step 3 have no entries in the descriptor file. SCREEN handles this situation by creating a new "miscellaneous" group named "DESCRIP-SS-MISC". It places in the group all fields for which there were no entries in the descriptor file. SCREEN conducts the normal source creation dialogue for the header element and all fields of the miscellaneous group.

3 Create Source Mode

In CREATE SOURCE mode, SCREEN generates Interactive COBOL source code for a screen image, and stores it in a line sequential file. The code may be inserted into any Interactive COBOL program, indirectly with the COPY statement or directly, with a CS editing program.

Through a simple prompt/response dialogue, you and SCREEN build a screen element (source line) for each field in the screen image. SCREEN allows you to include all Interactive COBOL Screen Section descriptors in the code.

You may supply all descriptor information manually, or you may instruct SCREEN to use data from a descriptor file associated with the screen image. During the dialogue, SCREEN uses its knowledge of Interactive COBOL syntax: it will not write code for a field until you have provided sufficient information.

For example, you might create any of the following source lines for a format (data-item) field at line 5/column 13, with PICTURE XXX9:

```
05 LINE 05 COL 13 PIC XXX9 TO CODE-ONE REQUIRED.
05 LINE 05 COL 13 PIC XXX9 TO PART-CODE OF PART-1.
05 BELL LINE 05 COL 13 PIC XXX9 FROM PART-NUMBER.
05 CITY LINE 05 COL 13 PIC XXX9 USING CITY-CODE.
05 LINE 05 COL 13 PIC XXX9 FROM ITEM-1 TO ITEM-2.
```

As you choose descriptors for a field, SCREEN superimposes a provisional source line on the screen image. You may modify descriptors as much as you like—adding some, deleting others—until the provisional source line is exactly as you want it. Then, a single keystroke tells SCREEN to write the source line to the disk file you named in the CREATE SOURCE command line.

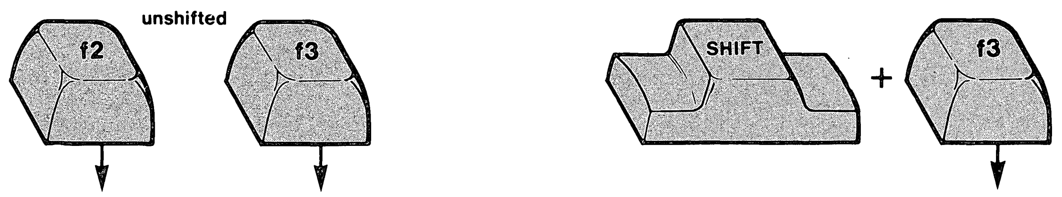
SCREEN produces CRT format source code if you enter CREATE SOURCE mode with the "N" command; it writes source code in CARD format if you issue the "C" command. Figure 3.1 illustrates the flow of information to and from disk files in CREATE SOURCE mode.

THE DISPLAY FORMAT

In CREATE SOURCE mode, SCREEN displays the current screen image and superimposes corresponding lines of source code. The code may appear in the upper or lower portion of the screen, depending on the position of the screen field currently under consideration. The current field and its provisional source line are underscored.

Dialogue questions and your responses are displayed below the provisional source line under consideration. SCREEN offers "menus" of descriptor options, with one letter of each option underscored. To select one or more descriptors, type only the underscored letter(s), then a single **ESC**. To delete a descriptor, type a minus sign followed by the appropriate (underlined) letter. As you select each descriptor option for a current field, SCREEN displays an updated provisional source line, as illustrated in Figure 3.2.

f9	Erases the command line.
f10	Saves the screen image and places the command string at the top of the screen for editing. Pressing **f10** again restores the modified command string. See "Command-String Editing" in this chapter.
f11	Displays an alphabetical summary of EDIT mode commands. Press any key to continue EDIT mode processing.
HOME	Terminates SCREEN.



f2 toggles back and forth between "INSERT" and "REPLACE" mode.

f3 toggles you to "LITERAL" mode — "REPLACE Literal" or "INSERT Literal" — depending on whether the most recent direct-entry mode was "INSERT" or "REPLACE".

SHIFT-f3 toggles you to "FORMAT" mode — "REPLACE Format" or "INSERT format" — depending on whether the most recent direct-entry mode was "INSERT" or "REPLACE".

FIGURE 2.5 DIRECT ENTRY OF SCREEN FIELDS. Function keys allow you to paint the screen image directly, without using EDIT mode commands.

While in either of the FORMAT modes, SCREEN checks the validity of format characters (just as it does with the "F" command). If you enter an illegal format character, SCREEN suspends processing and displays an error message. Press any key to continue direct-entry of format characters. SCREEN always returns the cursor to line 1, column 1 after such an error.

Exiting from Direct-Entry Mode The following keys terminate direct entry and reenable command-driven operations:

DEL ESC f9 f10

f1 SHIFT-f1 SHIFT-f2

These three keys also perform an immediate action. See the next section.

Character/Line-Deletion and Line-Splitting Keys

SCREEN also features IFMT-like usage of function keys to delete characters and lines, and to split lines. If you are in one of the direct-entry modes, these keys perform their function, then *terminate* direct entry and place SCREEN back into command-driven EDIT mode.

f1	Deletes the character at the cursor location, shifting succeeding characters on the line to the left one position. This is equivalent to the command "1D".
SHIFT-f1	Deletes the entire line at the cursor position, moving up all material on succeeding lines. Line 24 is left blank. This is equivalent to the command ".JK".
SHIFT-f2	Splits the line at the current cursor location, placing the right portion on the line below. All material below is shifted down one line, and line 24 is lost off the bottom of the screen. This is equivalent to the command "1S".

Utility-Control Keys

These keys allow you to edit the command line (CLI/SCREEN only), to view an alphabetical listing of EDIT mode commands, and to terminate SCREEN operation.

The Immediate-Action Keys

This section describes the use of keys that affect the screen image immediately—no **ESC** is required. These keys make SCREEN's power more accessible to non-technical users.

Cursor-Positioning Keys

The six cursor-positioning keys function as follows:

→	Moves the cursor one position to the right.
←	Moves the cursor one position to the left.
↑	Moves the cursor up one line.
↓	Moves the cursor down one line.
CR	Moves the cursor to the beginning of the current line.
NEWLINE	Moves the cursor to the beginning of the next line.

These keys obey the following rules at the edges of the display screen:

- The position to the "right" of column 80 is column 1 of the next line.
- The position to the "left" of column 1 is column 80 of the previous line.
- The line "above" line 1 is line 24.
- The line "below" line 24 is line 1.

Direct-Field-Entry Keys

The commands "I", "R", and "F" are used to paint the screen with literal and format fields. Pressing **f2**, **f3**, or **SHIFT-f3** changes the environment from command-driven to immediate-action, enabling the *direct-entry* modes. In these modes, pressing an alphanumeric key enters the character in the screen image. No "insert" or "replace" command is necessary.

Direct-Entry Modes

These four direct-entry modes make SCREEN operation consistent with that of the IFMT program of Data General's IDEA database management system:

- INSERT LITERAL is equivalent to the "I" command.
- REPLACE LITERAL is equivalent to the "R" command.
- INSERT FORMAT is equivalent to the "FI" command.
- REPLACE FORMAT is equivalent to the "FR" command.

Figure 2.5 shows the usage of **f2** and **f3** to place SCREEN into the direct-entry modes, and to switch back and forth among these modes. Essentially, **f2** affects the first word ("INSERT" or "REPLACE") while **f3** and **SHIFT-f3** affect the second word ("LITERAL" or "FORMAT").

To enter a direct-entry mode, press **f3** (for "LITERAL") or **SHIFT-f3** (for "FORMAT") first. Then press **f2** when you need to switch between "INSERT" and "REPLACE". SCREEN verifies its entry into these modes by displaying the mode's name on the command line.

In each of these modes, you may move the cursor with the cursor-positioning keys to the positions at which you wish to enter field characters. If you move to a different line while in either of the INSERT modes, SCREEN automatically changes to the corresponding REPLACE mode.

[and] Define a command loop

EDIT mode command strings can be executed repeatedly by surrounding the code letters with square brackets. The argument preceding the left bracket specifies the number of times the "loop" will be performed. Command loops may be nested up to 10 levels deep in IC/SCREEN and up to 4 levels deep in CLI/SCREEN.

Format

```
# [commandstring][$]
```

The brackets enclosing "commandstring" are part of the command line. Those enclosing "$" merely show that this character is optional.

Arguments

#	Repetition factor. Indicates that *commandstring* is to be executed # times in succession.
`commandstring`	A sequence of EDIT mode commands. Individual commands that include a text argument must be terminated with $.

Example 1

```
5[.+1,10J$FI-ZZZ,ZZ9.99$]$$
```

Inserts the format -ZZZ,ZZ9.99 at column 10 of the next 5 lines.

Example 2

```
J4[3[4[10RX$10R.$]L]3[4[10R.$10RX$]L]]$$
```

Creates a checkerboard. (Try it and see!)

Z *Jump Cursor to End of Line*

Moves the cursor to the end of a line. Like the JUMP command, "Z" causes an absolute jump.

Format

```
[#] Z[$]
```

Arguments

#	(optional: default value = 24) The line to which the cursor is to be moved.

Example 1

```
Z$$
```

Moves the cursor to line 24, column 80.

Example 2

```
13Z$$
```

Moves the cursor to line 13, column 80.

Example 3

```
.Z$$
```

Moves the cursor to the end of the current line.

W *WRITE Descriptor File*

Saves the current screen image, and any associated descriptors[1] in *descriptor-file*. Any existing version of *descriptor-file* is overwritten. This command allows you to save your SCREEN editing session without creating source code.

Format

```
    W descriptor-file$
```

Arguments

descriptor-file	The name of a file in which the screen image is to be stored. IC/SCREEN: You may not include a filename extension; a two-part file is created, *descriptor-file*.AX/*descriptor-file*.SX. CLI/SCREEN: If you do not include a filename extension, SCREEN creates *descriptor-file*.DX.

Example 1

WNEWMENU$$

CLI/SCREEN stores the current screen image and associated descriptors (if any) in NEWMENU.DX. IC/SCREEN stores the image and descriptors in NEWMENU.AX/NEWMENU.SX.

1 When you retrieve a screen image from storage with the GET command, SCREEN also inputs the descriptor data associated with the image. When you issue the WRITE command, SCREEN assumes that you wish to retain the descriptors for fields that you haven't changed. Accordingly, it outputs the unchanged fields' descriptors to the new file.

V Set VALUE of CURRENCY SIGN

Changes the characters recognized by SCREEN as the COBOL "CURRENCY SIGN". At the beginning of a SCREEN session, the CURRENCY SIGN is set to the dollar sign character (**SHIFT-4**).

The FORMAT command and ANALYZE mode recognize only the currently defined value of CURRENCY SIGN. Therefore, be sure to issue a "V" command before operating with a non-standard CURRENCY SIGN.

Format

This is a dollar sign, not ESC.

```
V $char[$]
```

Arguments

char	Any character accepted as a CURRENCY SIGN by the Interactive COBOL compiler. Digits and PICTURE-editing characters may not be used as the currency sign.

Example 1

```
V$M$$
```

Sets the CURRENCY SIGN to "M".

U UNITE Lines

Deletes all characters from the cursor position to the end of the line, and shifts in the same number of characters from the next line. The remainder of the next line is padded on the right with blanks.

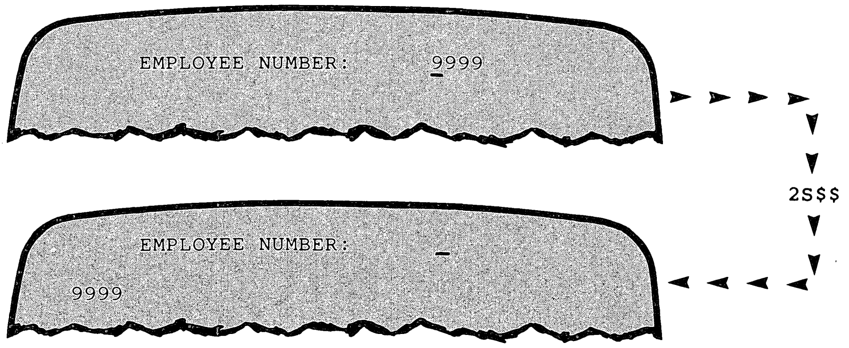
UNITE is the opposite of a single-line SPLIT: to "repair" the effect of a "1S" command, place the cursor at the split location and issue the command "U".

Format

```
U [$]
```

Example 1

To repair a multi-line split in the middle of a line, use both the "K" and "U" commands. In Example 2 of the SPLIT section, reproduced here, the command "2S" created a split between "EMPLOYEE NUMBER:" and "9999". To repair this split:



1. Return the cursor to the split location.
2. Issue the command "2K". This places the two parts on consecutive lines.
3. Issue the command "U".

T TRANSFER Literals and/or Formats

TRANSFER copies fields from any line of the screen to the current line. You may select to copy formats only, literals only, or both. The "from" line remains unchanged; the "to" line is completely blanked before the transfer is made.

Copying material from a line to itself effectively deletes the material not copied. Thus, ".TF" retains all format fields in the current line; all literal fields are deleted. Similarly, ".TT" retains all literal fields in the current line; all format fields are deleted.

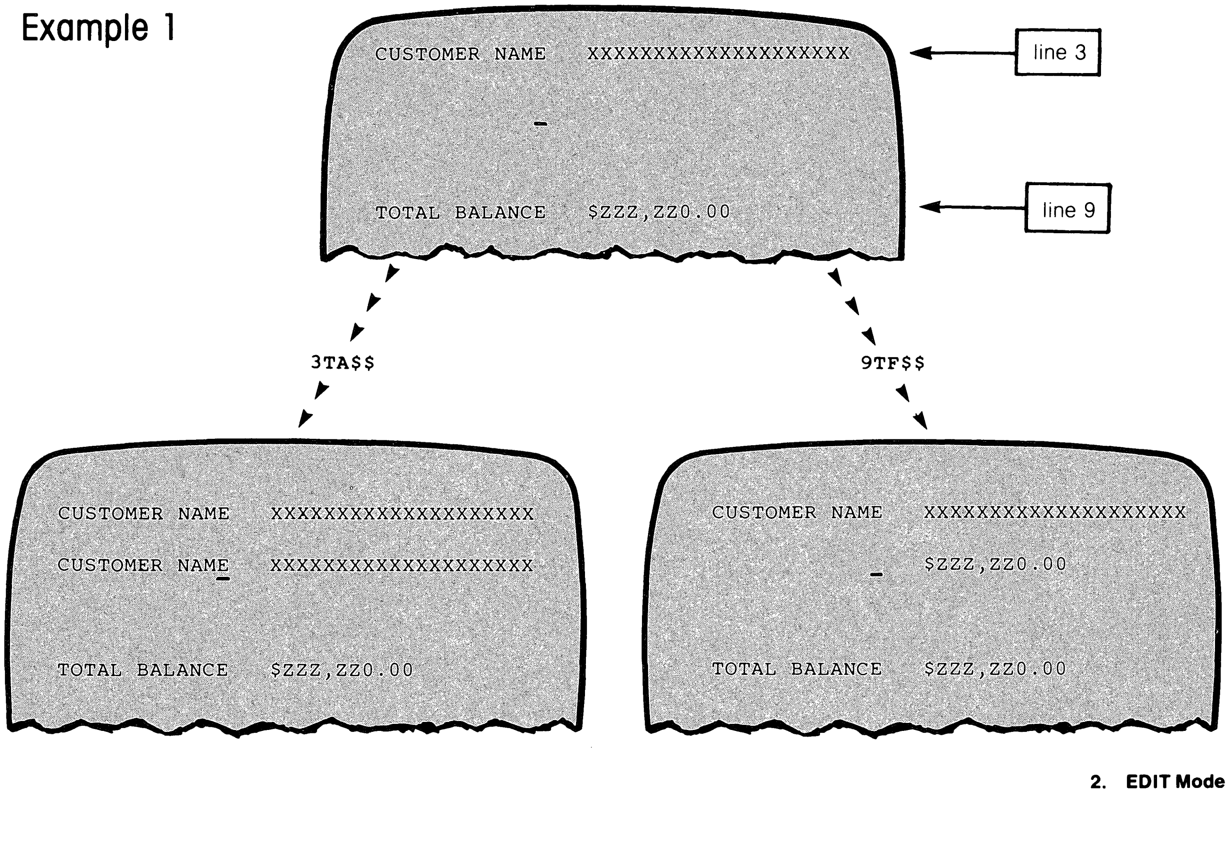
Format

$$[\#]\ T \begin{Bmatrix} A \\ F \\ T \end{Bmatrix} [\$]$$

Arguments

#	(optional: default value = 1) Indicates the line from which material is to be copied.
A	Transfers all literals and formats from line # to the current line.
F	Transfers only formats from line # to the current line.
T	Transfers only literals from line # to the current line.

Example 1


CUSTOMER NAME XXXXXXXXXXXXXXXXXXXX
TOTAL BALANCE $ZZZ,ZZ0.00
line 3
line 9
3TA$$
9TF$$
CUSTOMER NAME XXXXXXXXXXXXXXXXXXXX
CUSTOMER NAME XXXXXXXXXXXXXXXXXXXX
TOTAL BALANCE $ZZZ,ZZ0.00
CUSTOMER NAME XXXXXXXXXXXXXXXXXXXX
$ZZZ,ZZ0.00
TOTAL BALANCE $ZZZ,ZZ0.00

S *SPLIT Line*

SPLIT breaks the material on the current line into two parts at the current cursor position, placing the parts on two lines. The first part remains in place; the second part is shifted down one or more lines, and shifted left to column 1.

CAUTION: The SPLIT command shifts down all material below the split line. The bottom # lines are shifted off the bottom of the screen, and are lost.

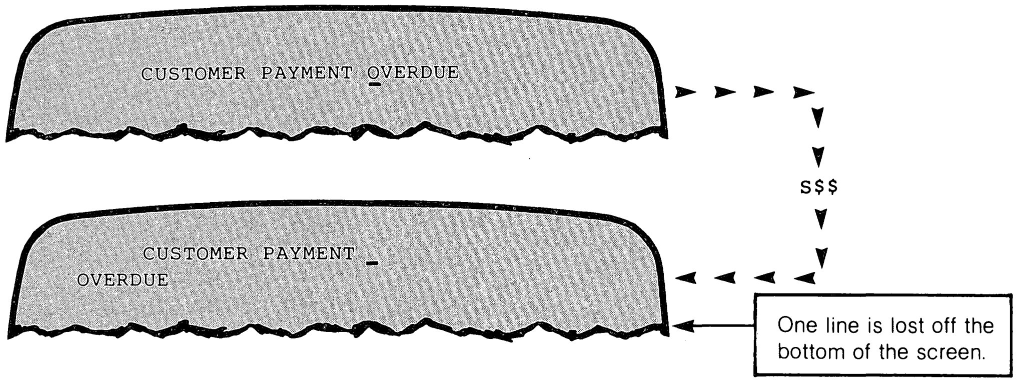
Format

```
[#] S[$]
```

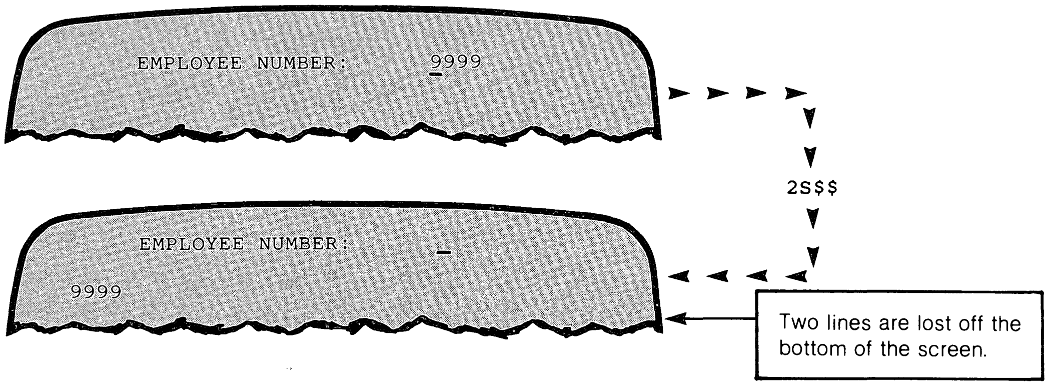
Arguments

#	(optional: default value = 1) SCREEN places the right part of the current line at the beginning of the #th line below, effectively inserting # -1 blank lines between the parts. (Thus, if # = 1, no blank lines are inserted—one line is split into two consecutive lines.) All material below is pushed down, so that # lines are lost off the bottom of the screen.

Example 1



Example 2



To "repair" a SPLIT line, use the UNITE command. (The "K" command may also be useful. See the UNITE command description.)

R REPLACE Literal Text

REPLACE is similar to INSERT in its ability to "paint" literal fields on the screen. The only difference is that REPLACE overwrites existing material on the current line—no material is shifted to the right.

REPLACE places material in the current line only. If the replacement text would extend past column 80, the extra characters are lost.

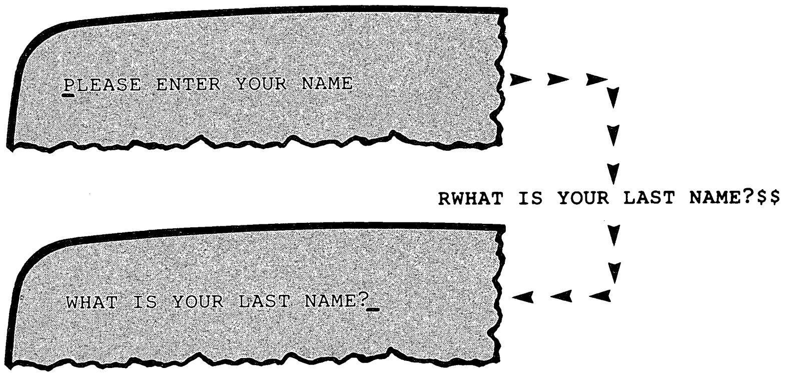
Format

```
[#] Rcharacter-string$
```

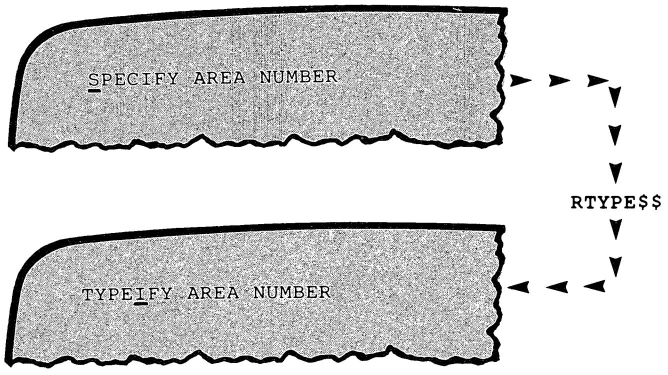
Arguments

#	(optional: default value = 1) Repetition factor. Specifies that # contiguous copies of *character-string* are to be inserted in the screen image.
`character-string`	May include any displayable keyboard character.

Example 1



Example 2



Q ANALYZE Source File in CARD Format

Instructs SCREEN to analyze the Screen Section of a specified CARD-format source file (with line numbers). This file might be a complete COBOL source program, a COPY file, or a file previously created by SCREEN itself. The file must have line sequential organization and contain either the statement "SCREEN SECTION." or both the header line **SCREEN** and the trailer line **SCREEN-END**.

CLI/SCREEN only: The source file may contain COPY statements in its Screen Section. SCREEN can incorporate copy files for analysis the same way that the Interactive COBOL compiler can incorporate them for compilation. Chapter 4 provides a complete description of ANALYZE mode.

Format

$$[\#]\ \texttt{Qsource-file}\left[\left\{\begin{matrix}\texttt{,}\\ \texttt{,descriptor-file}\end{matrix}\right\}\right]\$$$

Arguments

#	(optional: default value = 0) Data structure level. SCREEN extracts screen images that start at this level. If you omit the argument, SCREEN extracts all screen images defined in the file.
source-file	The name of the CARD-format source file to be analyzed.[1] It must be a line sequential file.
descriptor-file	(optional) The name of a descriptor file to store the analyzed image.[1] IC/SCREEN only: No filename extension is allowed; SCREEN creates a two-part descriptor file with extensions ".AX" and ".SX". This argument destroys any existing file named *descriptor-file.*

Example 1

6QMASTERSCRN$$

Retrieves screen images starting at level 06 from MASTERSCRN.SS, leaving the image on the screen for editing.

Example 2 (CLI/SCREEN)

QGREETING,NEWGREET$$

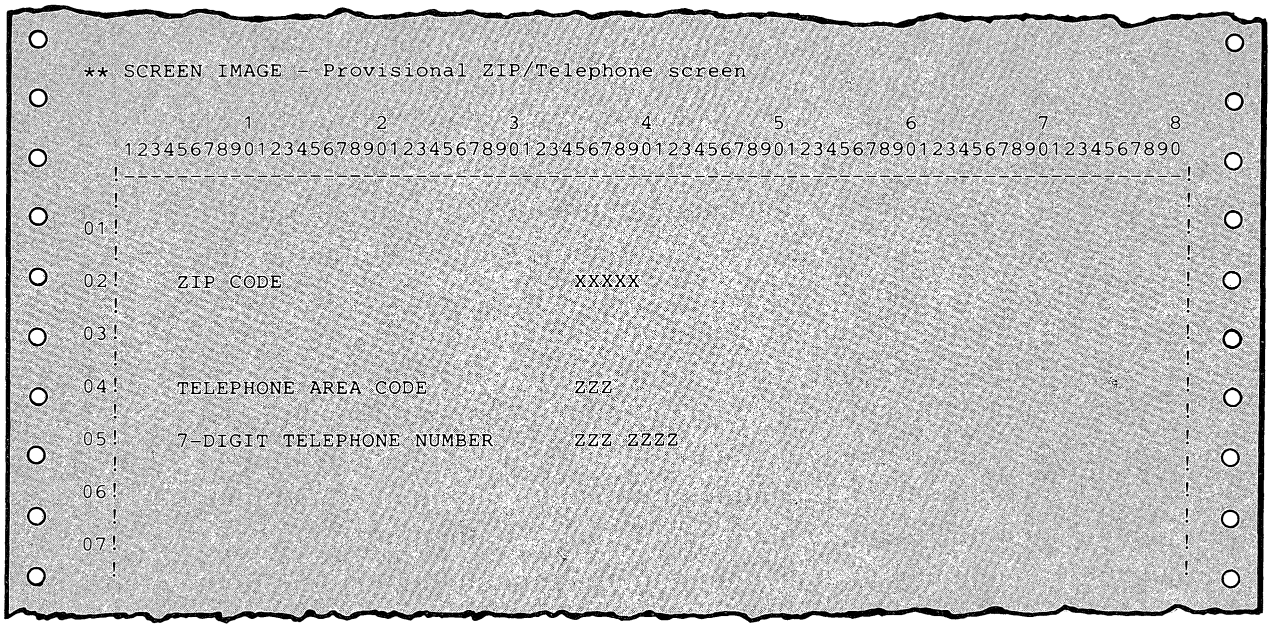
Retrieves all screen images from GREETING, storing the associated descriptors in NEWGREET.DX.

1 See "Using Filenames in Commands" and "Default Filenames" in Chapter 1.

2 These examples reflect SCREEN 's automatic handling of reserved filename extensions. See "Using Filenames in Commands" in Chapter 1.

Example 1

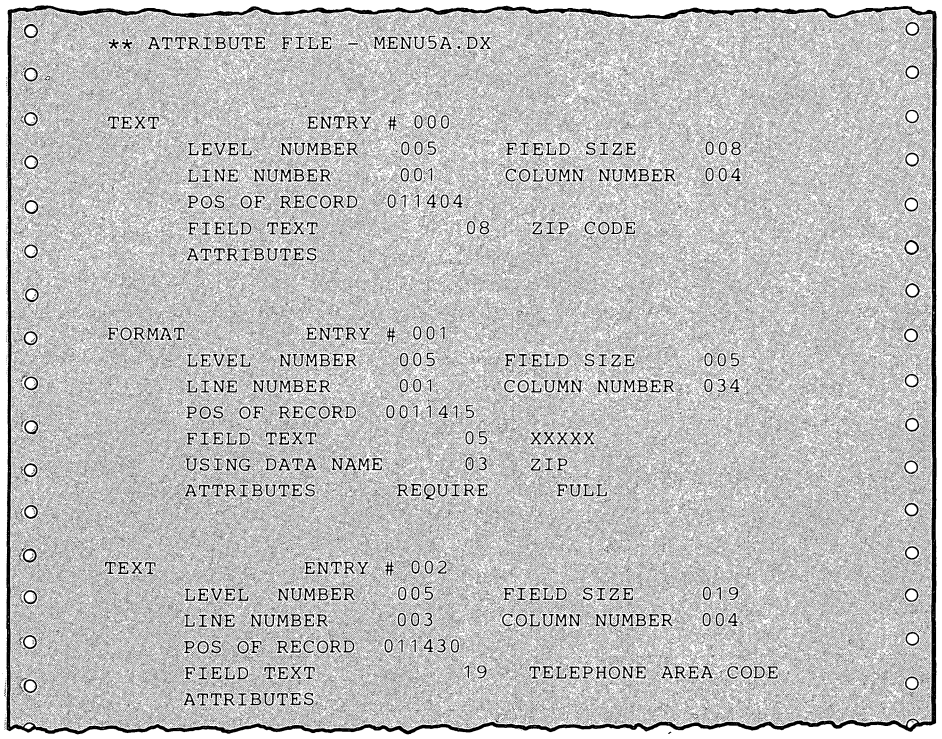
The command "PIProvisional ZIP/Telephone screen" might produce this printout:



```
** SCREEN IMAGE - Provisional ZIP/Telephone screen

             1         2         3         4         5         6         7         8
   12345678901234567890123456789012345678901234567890123456789012345678901234567890
  !--------------------------------------------------------------------------------!
  !                                                                                !
01!                                                                                !
  !                                                                                !
02!    ZIP CODE                      XXXXX                                         !
  !                                                                                !
03!                                                                                !
  !                                                                                !
04!    TELEPHONE AREA CODE           ZZZ                                           !
  !                                                                                !
05!    7-DIGIT TELEPHONE NUMBER      ZZZ ZZZZ                                      !
  !                                                                                !
06!                                                                                !
  !                                                                                !
07!                                                                                !
  !                                                                                !
```

Example 2

The CLI/SCREEN command "PAZIPTEL.01" might produce this printout:



```
** ATTRIBUTE FILE - MENU5A.DX


TEXT           ENTRY # 000
      LEVEL  NUMBER   005     FIELD SIZE     008
      LINE NUMBER     001     COLUMN NUMBER  004
      POS OF RECORD  011404
      FIELD TEXT           08   ZIP CODE
      ATTRIBUTES


FORMAT         ENTRY # 001
      LEVEL  NUMBER   005     FIELD SIZE     005
      LINE NUMBER     001     COLUMN NUMBER  034
      POS OF RECORD  0011415
      FIELD TEXT           05   XXXXX
      USING DATA NAME      03   ZIP
      ATTRIBUTES        REQUIRE      FULL


TEXT           ENTRY # 002
      LEVEL  NUMBER   005     FIELD SIZE     019
      LINE NUMBER     003     COLUMN NUMBER  004
      POS OF RECORD  011430
      FIELD TEXT           19   TELEPHONE AREA CODE
      ATTRIBUTES
```

P PRINT on the System Printer

This command enables you to document and review printed copies of your screen format designs without first having to code them into a COBOL program. SCREEN creates print files and sends them directly to the (first) system printer.

With either IC/SCREEN or CLI/SCREEN, you may use the "PI" command to print the screen image as it currently appears. In constructing a copy of the current screen image for printing, SCREEN includes row and column scales to help you align fields and monitor design consistency across several images. You may also specify a label to be printed at the top of the image. The label is simply printed—it is not stored as an indentifier for the image.

CLI/SCREEN users may also print the contents of a descriptor file with the "PA" command—this includes all descriptor data associated with the literal and format fields in the image.

An additional convenience for CLI/SCREEN users is the ability to print a line sequential source file (in fact, any line sequential ASCII file) with the "PS" command.

NOTES:

- "PI" command: To obtain a printed copy of the screen image that most nearly approximates the proportions of the display terminal screen, set the printer to 8 lines per inch.
- "PA" command: The descriptor file lists row and column numbers that are *one less* than the actual screen positions. (This reflects the internal representation of descriptors.) For instance, a field starting at row 6/column 30 is listed as "LINE NUMBER 005 ... COLUMN NUMBER 029".
- CAUTION: If the system printer is busy when you issue the "P" command, your print job will not be executed. With IC/SCREEN, the "printer busy" condition results in a fatal program error, terminating SCREEN execution. With CLI/SCREEN, execution continues—the command is effectively ignored.

Format

```
P ⎧ I[ image-label ] ⎫ $
  ⎨ Adescriptor-file ⎬
  ⎩ Sline-file       ⎭
```

Arguments

I	Print the current screen image, including row and column scales.
[image-label]	Optional character string to appear at the top of the image printout.
Adescriptor-file	Print the contents of the descriptor file with name *descriptor-file*.
Sline-file	Prints the contents of *line-file* on the (first) system printer. The file must be a line sequential ASCII file.

O *ORGANIZE Screen Image*

Places SCREEN in ORGANIZE mode, wherein you specify Data Division structure for the current screen image. In ORGANIZE mode, you add level number descriptors for literal and format fields, and create "slots" at any level for non-display elements, such as group headers. SCREEN stores the level number data under the descriptor file name you specify in the ORGANIZE command.

ORGANIZE mode has its own commands and HELP-screen. (See Chapter 5.)

Format

```
O descriptor-file$
```

Arguments

`descriptor-file`	The name of a descriptor file in which the level number information is to be stored.[1] If the current screen image was retrieved from a descriptor file, you may not use that filename as this argument.

Example 1

```
OACCTS05$$
```

Instructs SCREEN to ORGANIZE the current screen image. The level number data you specify during the ORGANIZE pass will be stored in descriptor file ACCTS05.DX.

1 See "Using Filenames in Commands" in Chapter 1.

N CREATE SOURCE File in CRT Format

Instructs SCREEN to create source code in CRT format (no line numbers) for the current screen image, storing it under the filename you supply. The source code may be COPYed or inserted into a COBOL program. If you also specify a descriptor file, SCREEN uses the file's stored data (e.g. level 05, BLINK, USING OVER-PAYMENT) in building source lines. Chapter 3 provides a complete description of CREATE SOURCE mode.

In CREATE SOURCE mode, SCREEN engages you in a dialogue that builds a screen element (that is, a line of source code that defines a screen format field) for each field in the current screen image. It also builds code for non-display elements (such as group headers) that you have added in ORGANIZE mode.

Format

$$[\#]\ \mathbf{N}\texttt{source-file}\ \left[\left\{\begin{matrix}\texttt{,}\\\texttt{,descriptor-file}\end{matrix}\right\}\right]\ \$$$

Arguments

#	(optional: default value = 05) The Data Division level number to be assigned to each line of SCREEN code. SCREEN ignores this argument if you specify a descriptor file that includes level-number data.
source-file	The name of the file in which the newly created source code will be stored.[1] SCREEN creates *source-file* as a line sequential file in CRT format (no line numbers).
descriptor-file	(optional) The name of a descriptor file that pertains to some or all the fields in the current screen image.[1]

Example 1[2]

```
7NNO$STOCK$$
```

Source code at the 07 level for the current screen image will be stored in NO$STOCK.SS.[1]

Example 2

```
NENDALL.02,ENDALL.01$$
```

Source code will be stored in ENDALL.02. The data from descriptor file ENDALL.01 is used in the source generation dialogue. If this data does not include level numbers, all source lines are assigned level number 05.

1 See "Using Filenames in Commands" and "Default Filenames" in Chapter 1.

2 These examples reflect SCREEN's automatic handling of reserved filename extensions. See "Using Filenames in Commands" in Chapter 1.

M *MOVE Cursor by Characters*

Moves the cursor by characters relative to its current position. This is unlike the "J" command, which provides an absolute jump capability. For purposes of this command, the last column of a line "wraps around to" (is logically followed by) the first column of the next line. There is no "wraparound", however, connecting the first and last positions: you can never send the cursor beyond line 24, column 80 with a forward MOVE, nor before line 1, column 1 with a backward MOVE.

Format

```
[#] M[$]
```

Arguments

#	(optional: default value = 1) Number of characters to move. If # is positive, move cursor forward. If # is negative, move cursor backward.

Example 1

```
80M$$
```

Moves the cursor to the same column of the next line.

Example 2

```
1L-1M$$
```

Moves the cursor to the last column of the current line.

L *Move Cursor by LINES*

Moves the cursor by lines relative to its current position. This is unlike the "J" command, which provides an absolute jump capability.

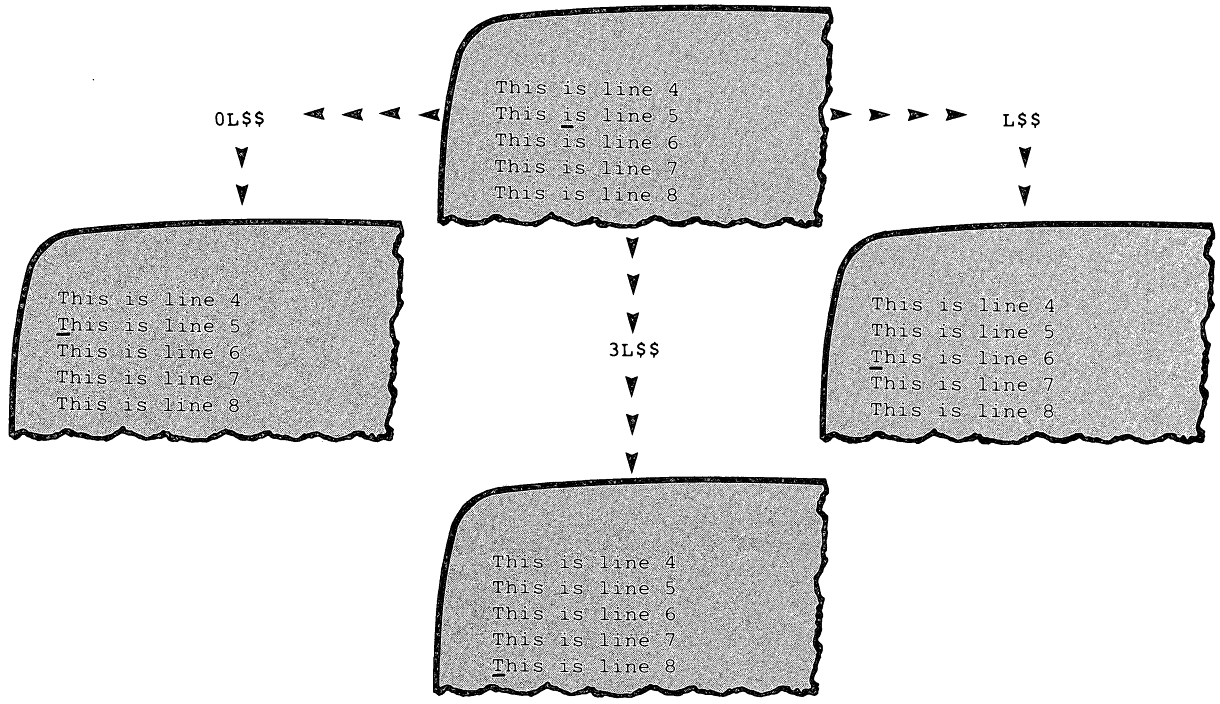
Format

```
[#] L[$]
```

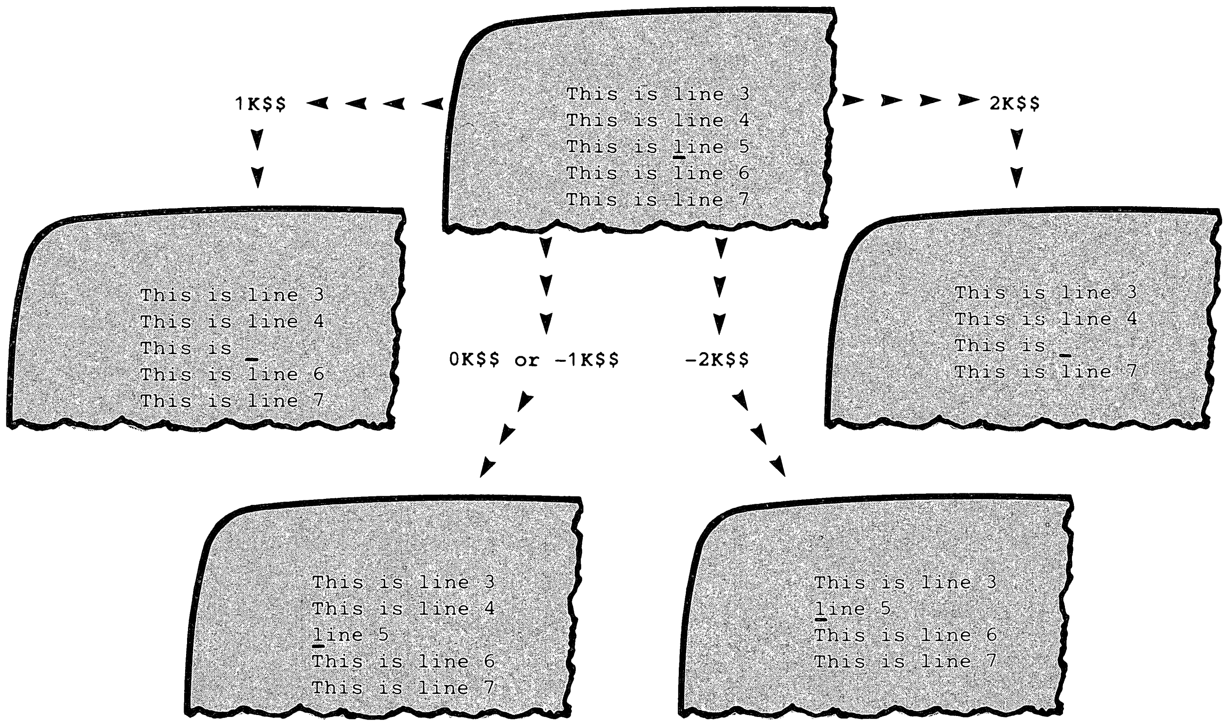
Arguments

#	(optional: default value = 1) If # = 0, the cursor moves to the beginning of the current line. If # is positive, the cursor moves to the beginning of the #th line forward. If # is negative, the cursor moves to the beginning of the #th line backward. There is no "wraparound" effect: you can never send the cursor beyond line 24 with a forward move, nor before line 1 with a backward move.

Examples



Examples



```
This is line 3
This is line 4
This is line 5
This is line 6
This is line 7
```

1K$$

```
This is line 3
This is line 4
This is _
This is line 6
This is line 7
```

2K$$

```
This is line 3
This is line 4
This is _
This is line 7
```

0K$$ or -1K$$

```
This is line 3
This is line 4
line 5
This is line 6
This is line 7
```

-2K$$

```
This is line 3
line 5
This is line 6
This is line 7
```

K Delete Lines

Deletes text by lines. You may delete characters on the current line only, delete one or more entire lines (the material on the lines below moves up to close the space), or both. The exact effect depends on the cursor position and the numeric argument you supply.

- If the cursor is at the beginning of a line:
  - □ # > 0: SCREEN deletes # entire lines, starting at the current one. The material below moves up to fill the space.
  - □ # < 0: SCREEN deletes # entire lines preceding the current one. The material starting at the current line moves up to fill the space.
- If the cursor is not at the beginning of a line:
  - □ # = 1: SCREEN deletes characters from the cursor position to the end of the line. No material moves up.
  - □ # = -1: SCREEN deletes all characters preceding the cursor position from the current line. The remaining material is shifted to the beginning of the line, but no material moves up.
  - □ # > 1: SCREEN performs "1K" at the current cursor position, then deletes the next # -1 lines, moving up the material below to close the space.
  - □ # < -1: SCREEN performs "-1K" at the current cursor position, then deletes the previous #+1 lines, moving up the material below to close the space.

NOTE: Entering SHIFT/fn-1 *deletes the enire line that the cursor is on.*

Format

```
[#] K[$]
```

Arguments

#	(optional: default value = 1) The effects of various values of # are explained above.

J JUMP Cursor to New Position

Moves the cursor to a new position in preparation for inserting text, replacing text, and so on. (You may also use the arrow keys, **CR** and **NEWLINE** to position the cursor.) JUMPing is absolute, not relative: the command has the same effect regardless of the current cursor position.

A column value larger than 80 causes a "wraparound": for instance, the command "3,95J" is equivalent to "4,15J". However, the end of the screen does not wrap around to the beginning: if the argument values would place the cursor beyond the end of the screen, it stops at line 24, column 80.

The variable arguments "." (representing the current line number) and ";" (representing the current column number) are useful with this command. See the examples below.

Format

$$\left[\left\{ \begin{matrix} \texttt{a} \\ \texttt{a,b} \end{matrix} \right\} \right] \texttt{J[\$]}$$

Arguments

a,b	(optional: default values; *a* = 1, *b* = 1) The cursor moves to line *a*, column *b*.

Example 1

```
12J$$
```

Moves the cursor to the beginning of line 12.

Example 2

```
12,50J$$
```

Moves the cursor to line 12, column 50.

Example 3

```
.J$$
```

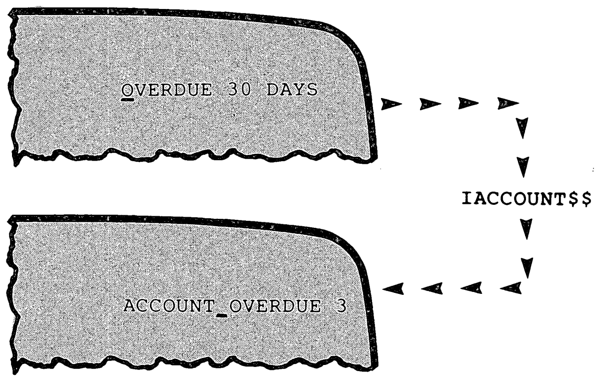
Moves the cursor to the beginning of the current line.

Example 4

```
5,;J$$
```

Moves the cursor vertically—to the same column of line 5.

Example 2



Note that the last few characters were shifted past column 80 and, hence, were lost.

I INSERT Literal Text

INSERT "paints" literal fields on the screen. Literal fields are encoded with the VALUE IS clause when you create source code for the screen image. You may include any displayable characters (upper/lowercase letters, digits, non-alphanumeric characters) in a literal field.

When you perform a literal insertion, SCREEN places the characters on the screen, starting at the current cursor position. Existing material on the line is shifted to the right. Any material that would extend past column 80 is lost.

If you insert more than one literal string on the same line, SCREEN may regard them as a single field with imbedded blanks. With CLI/SCREEN, you can verify this by moving the cursor to one of the inserted strings—the entire current field is always underscored. You may split such a field into parts with the SPLIT command in ORGANIZE mode.

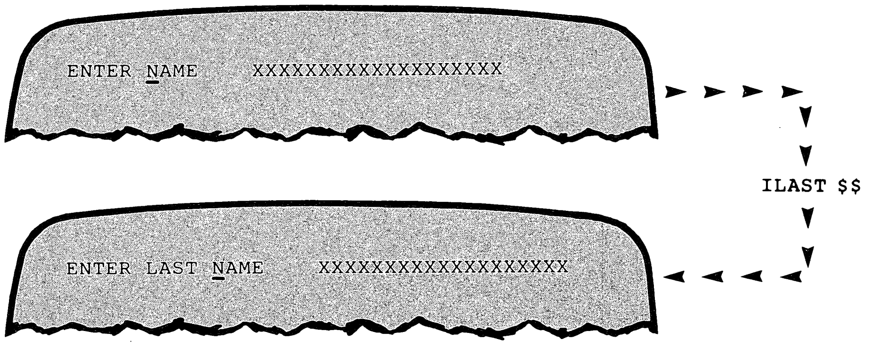
Format

```
[#] Icharacter-string$
```

Arguments

#	(optional: default value = 1) Repetition factor. Specifies that # contiguous copies of *character-string* are to be inserted in the screen image.
`character-string`	May include any displayable keyboard character.

Example 1


ENTER NAME XXXXXXXXXXXXXXXXXXX
ILAST $$
ENTER LAST NAME XXXXXXXXXXXXXXXXXXX

H *View HELP Screen*

Displays a HELP screen for any of the EDIT mode commands. The "HH" command displays an alphabetical command summary. (Pressing **f11** does the same.)

To continue processing after viewing a help screen, press any key (except a function key). This key is not interpreted as a command character.

Format

```
H [char][$]
```

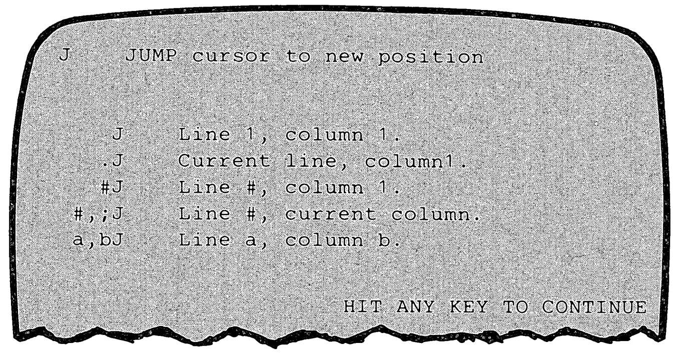
Arguments

char	(optional: default value = H) An EDIT mode command character. If you omit char, SCREEN displays an alphabetical summary of all EDIT mode commands.

Example 1

HJ$$

Displays a HELP screen for the "J" command.



```
J     JUMP cursor to new position


     J     Line 1, column 1.
    .J     Current line, column1.
    #J     Line #, column 1.
  #,;J     Line #, current column.
   a,bJ    Line a, column b.


                           HIT ANY KEY TO CONTINUE
```

G *GET Descriptor File*

Retrieves the information stored in a descriptor file, including both a screen image and any associated field descriptor data. The file may have been created with the WRITE command (EDIT mode), in ANALYZE mode, or in ORGANIZE mode.

This descriptor data is retained until you issue a BEGIN or WRITE command, or enter another mode. The WRITE and ORGANIZE commands retain that portion of the original descriptor information that pertains to the current screen image. These commands create new descriptor files to store this information, so that the original information is not lost.

Format

```
    G descriptor-file$
```

Arguments

descriptor-file	The name of the descriptor file in which the desired screen format is stored.[1] CLI/SCREEN descriptor files may not be retrieved by IC/SCREEN, and vice-versa.

IC/SCREEN: *descriptor-file* may not include a filename extension.

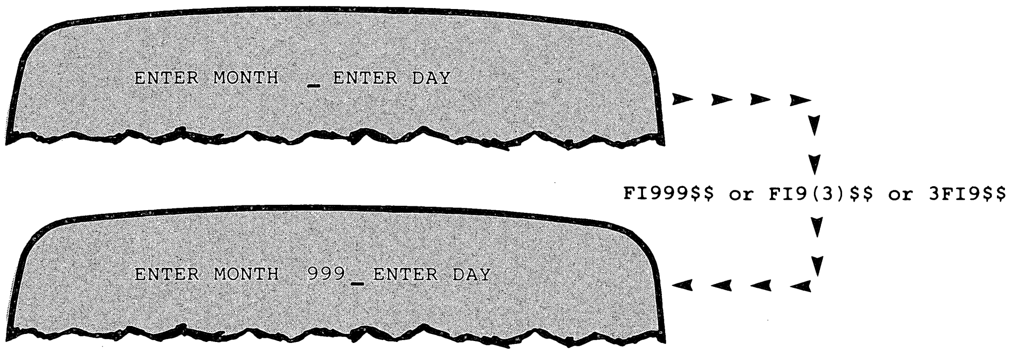
Example 1

```
GMENU54$$
```

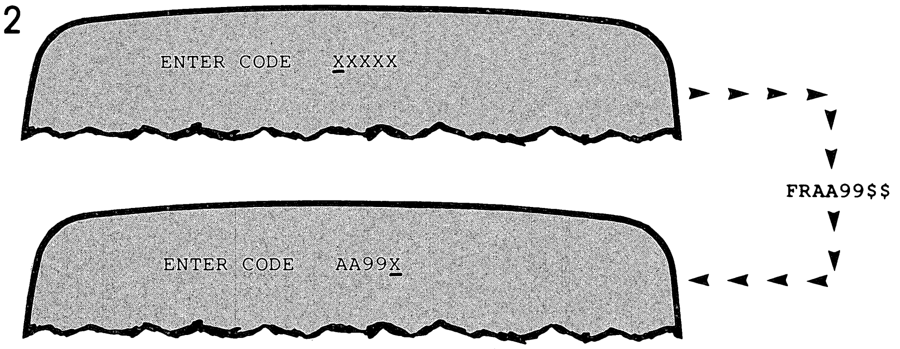
Retrieves the screen format from descriptor file MENU54.AX/MENU54.SX (IC/SCREEN) or from descriptor file MENU54.DX (CLI/SCREEN).

1 See "Using Filenames in Commands" in Chapter 1.

Example 1



Example 2



F FORMAT Insertion or Replacement

"Paints" data-entry/inquiry fields on the screen. These fields are encoded in PICTURE clauses when you create source code for the screen image. SCREEN checks your entry, allowing you to insert only legal PICTURE characters.

You may use the PICTURE clause parenthesis convention:

```
FIXXXXX      is the same as      FIX(5)
FR999.99     is the same as      FR9(3).9(2)
```

The FI (format insertion) command is similar to the INSERT command for literal fields. A format insertion pushes existing characters on the line to the right. Any material that would extend past column 80 is lost.

The FR (format replacement) command is similar to the REPLACE command for literal fields. A format replacement does not perform a shift: the new characters simply overwrite existing material. Replacement characters that would extend past column 80 are lost.

Format

```
[#] F { I }  character-string$
      { R }
```

Arguments

#	(optional: default value = 1) Repetition factor. Specifies that # contiguous copies of *character-string* are to be entered in the screen image.
I	Format characters to be inserted, pushing existing material to the right (and, possibly, off the end of the line).
R	Format characters to replace existing characters. No material is shifted.
character-string	Must consist of legal PICTURE characters: **A B X Z 0 9 * - +. , / ( ) CR DB <currency sign>**

The "F" command won't accept the characters P, S, and V—they are inappropriate for use in a display field PICTURE.

E EXIT

Terminates SCREEN session and returns control to the CLI. The **HOME** key may also be used to terminate SCREEN.

> CAUTION:
> No information is saved at the time you EXIT. To preserve the current state of your work, create a descriptor file with the WRITE command before EXITing.

Format

```
E$
```

Example 1

Nothing you've done today is worth saving. So you throw up your hands, issue the command "E", and head for home.

D DELETE Characters

Removes both format and literal characters from the current line. The remaining material on the line is shifted left to fill the empty space.

This command cannot affect characters on another line or run lines together. You may use the "K" command to delete lines and the "U" command to move material up from the line below to the current line.

NOTE: Pressing **f1** *is equivalent to issuing the command* "1D".

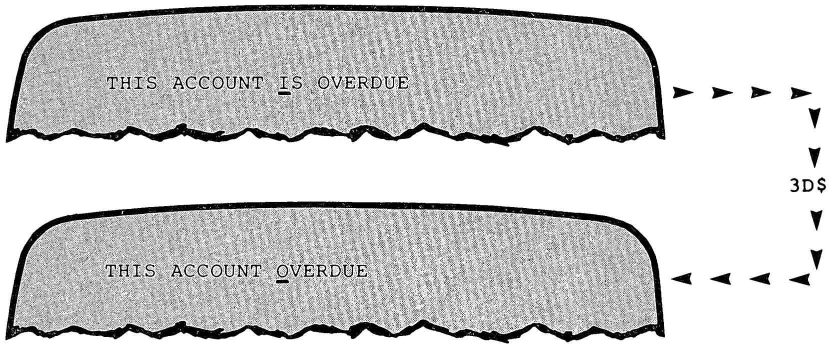
Format

```
[#] D[$]
```

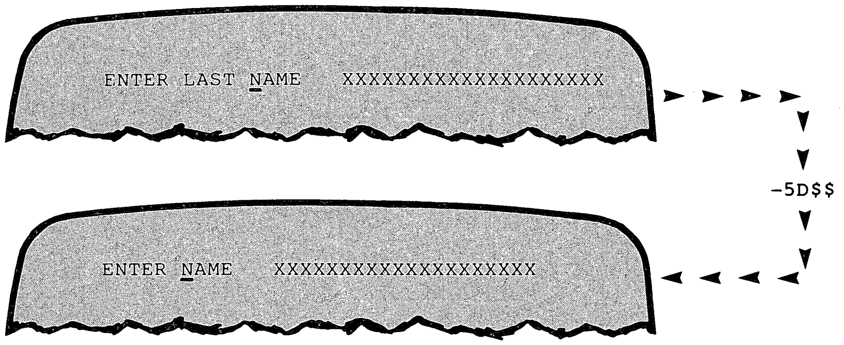
Arguments

#	(optional: default value = 1) Number of characters to be deleted. If # is positive, characters are deleted from the cursor position forward. If # is negative, characters preceding the cursor position are deleted.

Example 1



Example 2


ENTER LAST NAME XXXXXXXXXXXXXXXXXXXX
-5D$$
ENTER NAME XXXXXXXXXXXXXXXXXXXX

C CREATE SOURCE File in CARD Format

Instructs SCREEN to create source code in CARD format (with line numbers) for the current screen image, storing it under the filename you supply. The source code may be COPYed or inserted into a COBOL program. If you also specify a descriptor file, SCREEN uses the file's stored data (e.g. level 05, BLINK, USING OVER-PAYMENT) in building source lines. Chapter 3 provides a complete description of CREATE SOURCE mode.

In CREATE SOURCE mode, SCREEN engages you in a dialogue that builds a screen element (that is, a line of source code that defines a screen format field) for each field in the current screen image. It also builds code for non-display elements (such as group headers) that you have added in ORGANIZE mode.

Format

$$[\#]\ \texttt{Csource-file} \left[\left\{ \begin{array}{l} \texttt{,} \\ \texttt{,descriptor-file} \end{array} \right\} \right] \ \$$$

Arguments

#	(optional: default value = 05) The Data Division level number to be assigned to each line of SCREEN code. SCREEN ignores this argument if you specify a descriptor file that includes level-number data.
source-file	The name of the file in which the newly created source code will be stored.[1] SCREEN creates *source-file* as a line sequential file in CARD format (with line numbers).
descriptor-file	(optional) The name of a descriptor file that pertains to some or all the fields in the current screen image.[1]

Example 1[2]

```
7CNO$STOCK$$
```

Source code at the 07 level for the current screen image will be stored in NO$STOCK.SS.[1]

Example 2

```
CENDALL.02,ENDALL.01$$
```

Source code will be stored in ENDALL.02. The data from descriptor file ENDALL.01 is used in the source generation dialogue. If this data does not include level numbers, all source lines are assigned level number 05.

1 See "Using Filenames in Commands" and "Default Filenames" in Chapter 1.

2 These examples reflect SCREEN's automatic handling of reserved filename extensions. See "Using Filenames in Commands" in Chapter 1.

B *BEGIN Again*

Clears the EDIT mode screen and restarts the SCREEN session. This command is useful when you make a mistake that you'd rather forget than fix.

Format

```
B [$]
```

Example 1

While you are designing a client's display screen format, your boss tells you that the client just went out of business. Issue the "B" command—then start working on screen formats for a more solvent client.

A ANALYZE Source File in CRT Format

Instructs SCREEN to analyze the Screen Section of a specified CRT-format source file (no line numbers). This file might be a complete COBOL source program, a COPY file, or a file previously created by SCREEN itself. The file must have line sequential organization and contain either the statement "SCREEN SECTION." or both the header line **SCREEN** and the trailer line **SCREEN-END**.

CLI/SCREEN only: The source file may contain COPY statements in its Screen Section. SCREEN can incorporate copy files for analysis the same way that the Interactive COBOL compiler can incorporate them for compilation. Chapter 4 provides a complete description of ANALYZE mode.

Format

$$\texttt{[\#] Asource-file} \left[\left\{ \begin{array}{l} \texttt{,} \\ \texttt{;descriptor-file} \end{array} \right\} \right] \texttt{\$}$$

Arguments

#	(optional: default value = 0) Data structure level. SCREEN extracts screen images that start at this level. If you omit the argument, SCREEN extracts all screen images defined in the file.
source-file	The name of the CRT-format source file to be analyzed.[1] It must be a line sequential file.
descriptor-file	(optional) The name of a descriptor file to store the analyzed image.[1] IC/SCREEN only: No filename extension is allowed; IC/SCREEN creates a two-part descriptor file with extensions ".AX" and ".SX". This argument destroys any existing file named *descriptor-file.*

Example 1[2]

```
6AMASTERSCRN$$
```

Retrieves screen images starting at level 06 from MASTERSCRN.SS, leaving the image on the screen for editing.[1]

Example 2 (CLI/SCREEN)

```
AGREETING,NEWGREET$$
```

Retrieves all screen images from GREETING, storing the associated descriptors in NEWGREET.DX.

1 See "Using Filenames in Commands" and "Default Filenames" in Chapter 1.

2 These examples reflect SCREEN's automatic handling of reserved filename extensions. See "Using Filenames in Commands" in Chapter 1.

The immediate-action keys and their functions are listed in Figure 2.4.

TEXT-MANIPULATION COMMANDS

D	Delete Character
F	Insert/replace format characters
I	Insert literal characters
K	Delete line
R	Replace literal characters
S	Split line
T	Transfer format or literal characters from another line
U	Unite lines

INPUT/OUTPUT COMMANDS

G	Get descriptor file (retrieve previous work)
P	Print
W	Write descriptor file (save current work)

UTILITY-CONTROL COMMANDS

A	Analyze CRT format source
B	Restart
C	Create Card format source
E	Exit
H	Help
N	Create CRT format source
O	Organize Screen Image
Q	Analyze Card format source
V	Assign CURRENCY SIGN value

CURSOR-POSITIONING COMMANDS

J	Jump cursor to beginning of a line
L	Move cursor by lines
M	Move cursor by characters
Z	Jump cursor to end of a line

CURSOR-POSITIONING KEYS

→	Move cursor one column to the right
←	Move cursor one column to the left
↓	Move cursor one line down
↑	Move cursor one line up
CR	Return cursor to beginning of current line
NEWLINE	Move cursor to beginning of next line

DIRECT-FIELD-ENTRY KEYS

f2	Toggle between INSERT and REPLACE direct-entry modes
f3	Enable direct entry of LITERAL characters
SHIFT-f3	Enable direct entry of FORMAT characters

CHARACTER/LINE-DELETION AND LINE-SPLITTING KEYS

f1	Delete one character
SHIFT-f1	Delete one line
SHIFT-f2	Split line at cursor position

FIGURE 2.4 EDIT MODE COMMANDS. The EDIT mode commands are summarized here, categorized by their function. The use of the cursor-positioning keys and the function keys is also included.

COMMAND REFERENCE

This section presents complete explanations of the EDIT mode commands, in alphabetical order. Following this listing are descriptions of the immediate-action keys.

when entering a lengthy error message. Command-string editing allows you to insert the missing command letter without having to retype the entire error message text.

To edit the command string, press **f10**. This temporarily saves the screen image and places the command line at the top of the screen. You may edit the command string using EDIT mode commands and the cursor-positioning keys.

When you have completed editing the command line, press **f10** again to restore the saved screen image and reload the modified command line.

NOTE: All **ESC***s in the command line are converted to dollar signs. You may use EDIT mode commands to modify the command line. If you wish to insert an* **ESC** *in the command line, specify it as a dollar sign: for example,* "I$**ESC ESC**"*. When you press* **f10** *again, dollar signs in the modified command line are converted back to* **ESC***s.*

EDIT MODE COMMANDS

EDIT Mode commands perform a range of functions: they "paint" screen images, print, save, and retrieve images, and call other SCREEN modes. At any time you may press **f11** to display an alphabetical summary of all EDIT mode commands, then press any key to continue EDIT mode processing.

EDIT mode commands fall into the following categories:

- CURSOR POSITIONING COMMANDS move the cursor around the 24-line X 80-column display screen.
- TEXT MANIPULATION COMMANDS insert, delete, replace, and duplicate text in a screen image.
- INPUT/OUTPUT COMMANDS retrieve screen format information from disk files, store screen format information in disk files, and print information on the system printer.
- UTILITY CONTROL COMMANDS invoke other SCREEN modes, display help screens, and assign values to the CURRENCY SIGN.

A summary of EDIT mode commands, organized by category, appears in Figure 2.4.

Immediate-action Keys

Several keys have an immediate effect on the screen image. You need not press **ESC** to execute the key's function:

- CURSOR-POSITIONING KEYS move the cursor around the screen. They do not affect the command line. Hence, you may execute the same command string at different screen positions by alternating cursor placement (using cursor control keys) with command reexecution (using the **ESC** key).
- DIRECT-FIELD-ENTRY KEYS enable direct entry of literal and format characters into the screen image. You need not use the "INSERT", "REPLACE", and "FORMAT" commands.
- CHARACTER/LINE-DELETION AND LINE-SPLITTING KEYS delete and move material in the screen image immediately. They perform the same functions as the "DELETE character", "DELETE line", and "SPLIT" commands.
- UTILITY CONTROL KEYS display help screens, allow command-line editing,and terminate program execution.

Numeric Arguments

Many EDIT mode commands accept numeric arguments. Arguments are used to specify repetition factors and display screen positions. CLI/SCREEN allows you to use arithmetic expressions up to 40 *terms* long as arguments. In this context, a *term* is an integer, one of the four arithmetic operators (+, -, *, /), or a parenthesis. SCREEN evaluates these expressions in the following order, according to "algebraic precedence":

First Operations in parentheses.
Second Multiplication (*) and division (/). Quotients are rounded down to whole numbers.
Third Addition (+) and subtraction (-).

If you omit a numeric argument, most commands default to an argument value of one. Exceptions are noted in the "Command Reference" descriptions. For example, these five commands are equivalent, since the arithmetic expression in each evaluates to 20:

```
10+7+3IABC$   2+6*3IABC$   (2+6)*3-4IABC$   103/5IABC$
```

Numeric Argument Variables

The period (.) and semicolon (;) are numeric argument variables. A period denotes the current line; a semicolon denotes the current column.

You may use these symbols by themselves or in expressions as arguments to SCREEN commands. For example, consider the command "#J", which moves the cursor to the beginning of line #. The command ".J" moves the cursor to the beginning of the current line; the command ".+3J" moves the cursor to the beginning of the third line following the current line. the command "12,;J" moves the cursor to the same column of line 12.

Using Repetition Brackets

You may use repetition brackets to repeatedly execute entire command strings or parts of strings—in effect, making miniature editing routines.

To repeat a command string # times, enclose the string in square brackets and precede it with # (or an expression that evaluates to #):

```
#[command-string]
```

You might use this facility to perform the same editing operation on several lines at once. For example, the command "6[L37MK]" erases columns 38-80 of the next 6 lines.

Command String Editing

You may use the **DEL** key to erase one or more characters from the end of the command string as you type it. You may also use **DEL** when SCREEN redisplays a command string it has just executed. Pressing **DEL** causes the redisplayed string *not* to be blanked—this makes it convenient to revise the last few characters of the command string or to build a longer string.

You may use **f9** at any time to blank the entire command string.

CLI/SCREEN only: You may use editing commands to modify an EDIT mode command line. For example, you might have forgotten to type a command letter

- Underscores the entire field at the cursor position, and displays the length of this field at the lower right corner of the screen.
- Displays format fields in bright characters and literal fields in dim characters.
- Underscores the current command line and *blinks* the $s that echo the **ESC** key.

EDIT MODE COMMAND SYNTAX

Figure 2.3 illustrates the syntax for EDIT mode commands. Not every EDIT mode command requires every component shown in the figure. See the "Command Reference" in this chapter for the syntax of each command.

As you type each character of a command, it is displayed on line 1 or line 23. One or more commands on the line (maximum total = 70 characters) form a **command string**. SCREEN ignores spaces in a command string unless they are imbedded in text arguments.

The ESCape Character

In EDIT mode, SCREEN uses the **ESC** character as a command string separator and command terminator. SCREEN echoes **ESC** in the command string as a dollar sign, **$**. One **ESC** separates commands in a command string; two **ESC**s terminate a command.

Within a command string, each command that includes a text argument *must* be separated from the following command by an **ESC**. However, you may use **ESC** to separate commands within a command string, even where it is not required.

Two consecutive **ESC**s signal SCREEN to execute the command string. SCREEN redisplays the command string after executing it, minus one of the terminating **$**s. At this point, you may reexecute the command by pressing **ESC** just once. The cursor-positioning keys do not affect the command string; thus, you may use these keys and **ESC** in combination to execute the same command string at several screen positions.

If you do not wish to reexecute the command, just start typing in your next command—SCREEN automatically clears the command line and echoes the new characters.

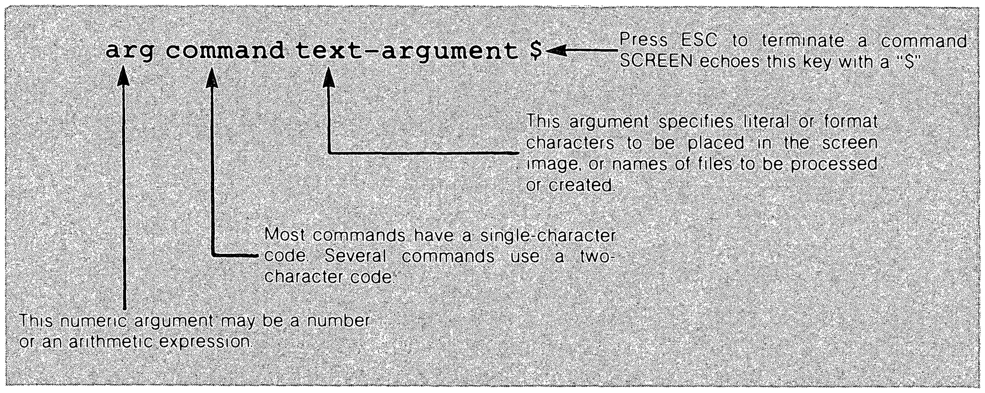

arg command text-argument $
Press ESC to terminate a command. SCREEN echoes this key with a "$"
This argument specifies literal or format characters to be placed in the screen image, or names of files to be processed or created.
Most commands have a single-character code. Several commands use a two-character code.
This numeric argument may be a number or an arithmetic expression.

FIGURE 2.3 EDIT MODE COMMAND FORMAT. EDIT mode commands allow you to design screen images rapidly and efficiently. For simplicity, SCREEN also allows "direct entry" of screen image fields, using the cursor-positioning keys and alphanumeric keys.

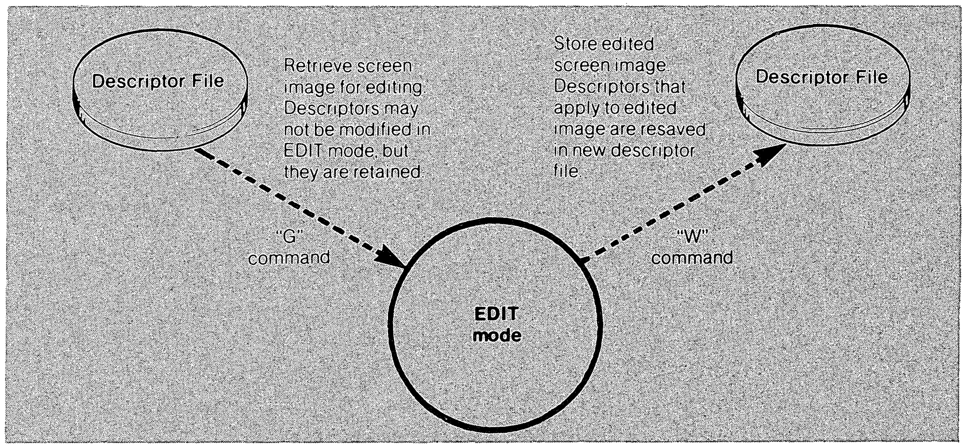

Descriptor File
Retrieve screen image for editing. Descriptors may not be modified in EDIT mode, but they are retained.
Store edited screen image. Descriptors that apply to edited image are resaved in new descriptor file.
Descriptor File
"G" command
"W" command
EDIT mode

FIGURE 2.2 EDIT MODE WORK FILES. In EDIT mode, you may store screen images in "descriptor files", to be retrieved for further work later.

Command-driven operation allows you to manipulate screen images with efficiency and speed. Command-repetition arguments enable repeated execution of a command in a single step. You may also "string" several commands together and execute them as a unit—a miniature editing routine. SCREEN remembers the most recently executed command string, allowing you to perform the same command or editing routine at different screen locations.

To complement the power of command-driven operation, SCREEN also offers the simplicity of a non-command, *immediate-action* environment. In this mode of operation, every keystroke affects the screen image directly—you may insert, overwrite, and delete characters, delete and split lines, and move the cursor around the screen with immediate-action keys.

THE DISPLAY SCREEN

SCREEN makes full use of the display terminal screen: In EDIT mode, the full 24-line x 80-column display area is available for manipulating screen images. Superimposed on the screen image is the *command line* that echoes the commands you enter. If you enter an incorrect command, SCREEN responds with an *error display.*

The position of the command line is not fixed: it moves so that the entire line at the current cursor position is always visible. The command line may appear on line 1 or line 23.

Error messages always appear in a three-line display at the bottom of the screen. SCREEN displays the faulty command string, the particular command character and arguments that caused the error, an error code, and an error message. SCREEN error messages are listed in Appendix D.

A *cursor* always indicates the focus of SCREEN's attention. EDIT mode commands that alter the screen image always take effect at the cursor position. Other commands have the same effect regardless of the cursor position. SCREEN's cursor is *not* an underscore, as with many other CS programs; instead, the character at the cursor position blinks. If the character is a blank, a blinking vertical mark (|) is displayed. You may position the cursor using commands or the terminal's cursor-control keys.

As you position the cursor, SCREEN displays updated values for the current line and column at the lower right side of the display screen. To provide additional aids and visual cues, CLI/SCREEN:

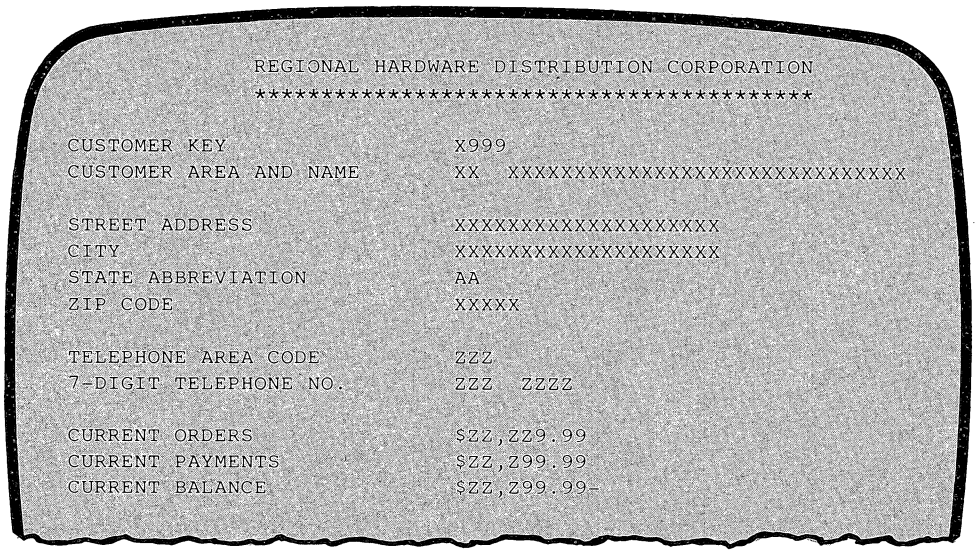
2 Edit Mode

EDIT mode is SCREEN's principal operating environment and central control point. During a SCREEN session, you can invoke SCREEN's other modes through EDIT-mode commands. Normal program entry and exit paths are routed through EDIT mode.

In EDIT mode, you perform the basic operations that "paint" a screen image: inserting new literal and format fields, moving fields around the screen for best visual effect, modifying and removing fields. (See Figure 2.1.) You may create a new screen image or modify an image that SCREEN has extracted from an existing source file. As you proceed, you may print a copy of the current screen image, complete with row and column reference scales.

During a SCREEN session, you may work on several screen images. You don't need to create source code immediately—SCREEN lets you save work-in-progress in a descriptor file, then retrieve the image for additional editing at another time. Figure 2.2 illustrates the flow of information to and from disk files in EDIT mode.

SCREEN's text manipulation capabilities are invoked through a simple command language. You move SCREEN's cursor to the location on the screen where you wish to perform an operation. (Do this either with commands or with the terminal's cursor control keys.) Then, you issue a command to perform an editing function.



```
             REGIONAL HARDWARE DISTRIBUTION CORPORATION
             ******************************************

CUSTOMER KEY                 X999
CUSTOMER AREA AND NAME       XX  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

STREET ADDRESS               XXXXXXXXXXXXXXXXXXXX
CITY                         XXXXXXXXXXXXXXXXXXXX
STATE ABBREVIATION           AA
ZIP CODE                     XXXXX

TELEPHONE AREA CODE          ZZZ
7-DIGIT TELEPHONE NO.        ZZZ  ZZZZ

CURRENT ORDERS               $ZZ,ZZ9.99
CURRENT PAYMENTS             $ZZ,Z99.99
CURRENT BALANCE              $ZZ,Z99.99-
```

FIGURE 2.1 A SCREEN IMAGE. In EDIT mode, you paint a new screen image or modify an existing one. "Direct-entry" makes it easy for non-technical personnel to design screen images.

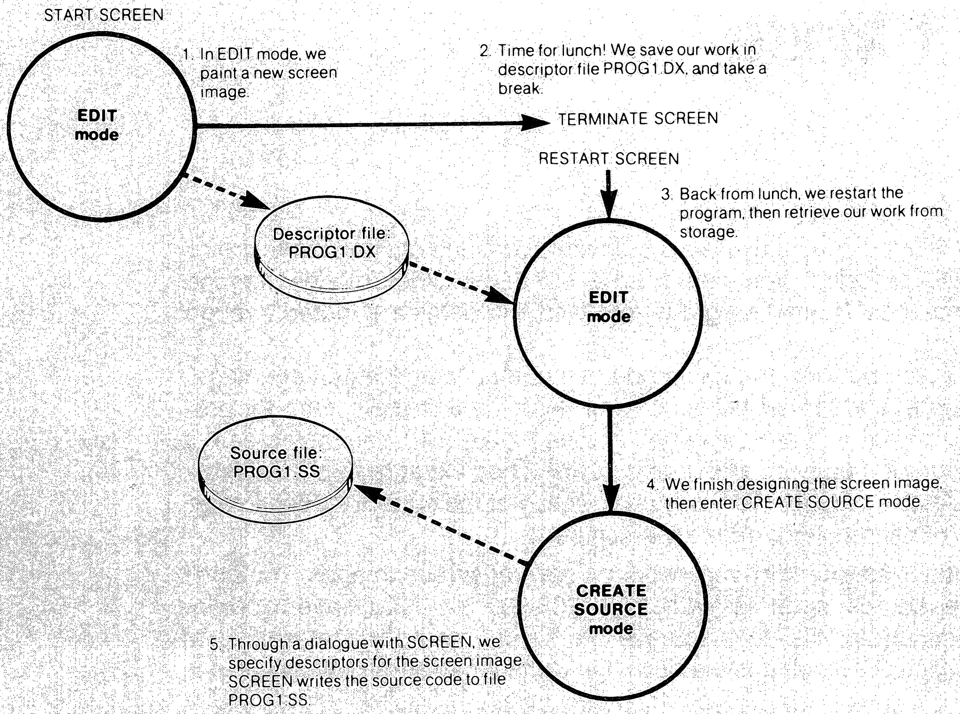

START SCREEN
EDIT mode
1. In EDIT mode, we paint a new screen image.
2. Time for lunch! We save our work in descriptor file PROG1.DX, and take a break.
TERMINATE SCREEN
RESTART SCREEN
3. Back from lunch, we restart the program, then retrieve our work from storage.
Descriptor file: PROG1.DX
EDIT mode
Source file: PROG1.SS
4. We finish designing the screen image, then enter CREATE SOURCE mode.
CREATE SOURCE mode
5. Through a dialogue with SCREEN, we specify descriptors for the screen image. SCREEN writes the source code to file PROG1.SS.

FIGURE 1.9 STORING WORK IN PROGRESS. **You can interrupt a SCREEN session, storing your work in a descriptor file on disk.**

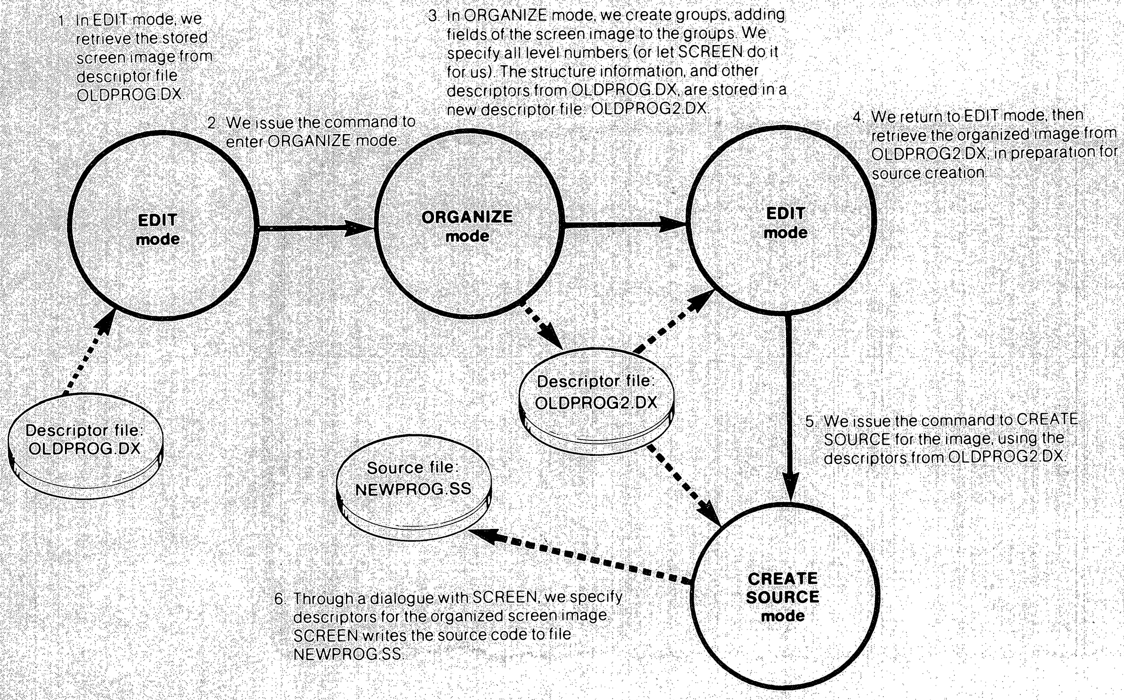

1. In EDIT mode, we retrieve the stored screen image from descriptor file OLDPROG.DX.
3. In ORGANIZE mode, we create groups, adding fields of the screen image to the groups. We specify all level numbers (or let SCREEN do it for us). The structure information, and other descriptors from OLDPROG.DX, are stored in a new descriptor file: OLDPROG2.DX.
2. We issue the command to enter ORGANIZE mode.
4. We return to EDIT mode, then retrieve the organized image from OLDPROG2.DX, in preparation for source creation.
EDIT mode
ORGANIZE mode
EDIT mode
Descriptor file: OLDPROG2.DX
5. We issue the command to CREATE SOURCE for the image, using the descriptors from OLDPROG2.DX.
Descriptor file: OLDPROG.DX
Source file: NEWPROG.SS
6. Through a dialogue with SCREEN, we specify descriptors for the organized screen image. SCREEN writes the source code to file NEWPROG.SS.
CREATE SOURCE mode

FIGURE 1.10 CREATING A GROUP. **In CLI/SCREEN's ORGANIZE mode, you can structure the fields of a screen image into hierarchical groups.**

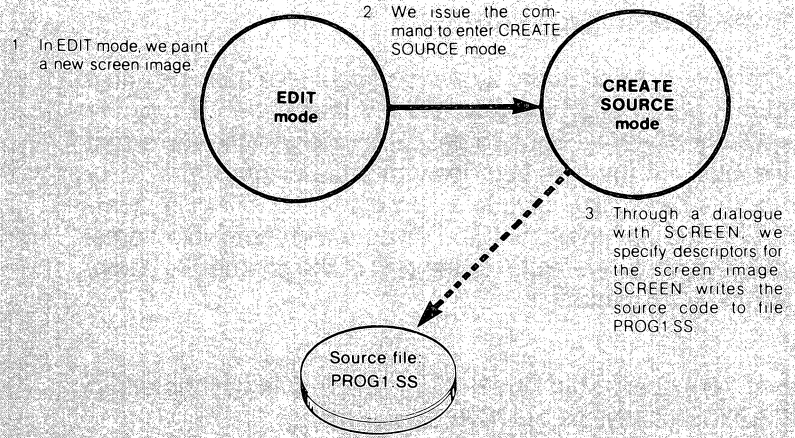

1. In EDIT mode, we paint a new screen image.
2. We issue the command to enter CREATE SOURCE mode.
EDIT mode
CREATE SOURCE mode
3. Through a dialogue with SCREEN, we specify descriptors for the screen image. SCREEN writes the source code to file PROG1.SS.
Source file: PROG1.SS

FIGURE 1.7 CREATING A NEW FORMAT. **To define a new display screen format, you can use SCREEN to "paint" an image and generate source code for the image.**

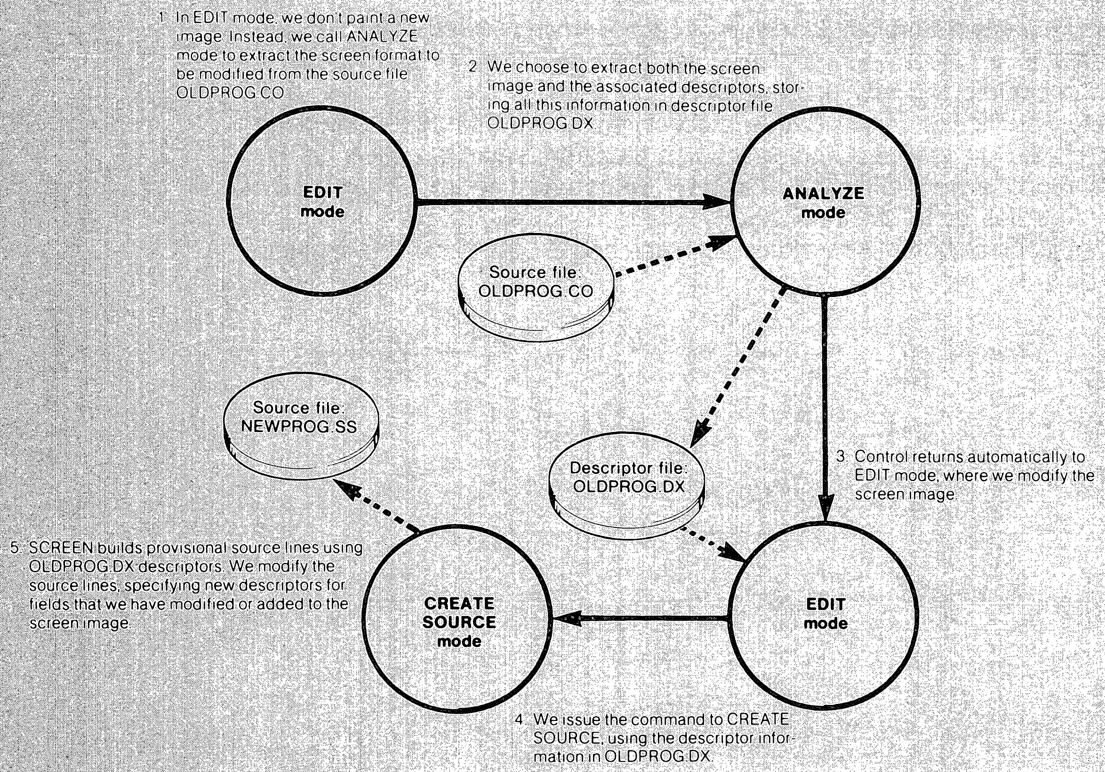

1. In EDIT mode, we don't paint a new image. Instead, we call ANALYZE mode to extract the screen format to be modified from the source file OLDPROG.CO.
2. We choose to extract both the screen image and the associated descriptors, storing all this information in descriptor file OLDPROG.DX.
EDIT mode
ANALYZE mode
Source file: OLDPROG.CO
Source file: NEWPROG.SS
Descriptor file: OLDPROG.DX
3. Control returns automatically to EDIT mode, where we modify the screen image.
5. SCREEN builds provisional source lines using OLDPROG.DX descriptors. We modify the source lines, specifying new descriptors for fields that we have modified or added to the screen image.
CREATE SOURCE mode
EDIT mode
4. We issue the command to CREATE SOURCE, using the descriptor information in OLDPROG.DX.

FIGURE 1.8 ADAPTING AN EXISTING FORMAT. **You can modify a display screen format that is part of one application system for use in another system. SCREEN can extract a screen format from an Interactive COBOL source file.**

☐ CLI/SCREEN searches for *descriptor-file*.DX.

DEFAULT FILENAMES The ANALYZE command accepts a source file for input and a descriptor file for output. SCREEN allows you to identify these two files by the "same" name. More precisely, SCREEN creates a descriptor file with the same root name as the source file, but with the reserved descriptor-file extension.

To take advantage of this capability, include a comma in the ANALYZE command line after the root source-file name, but don't type a descriptor-file name. The root of the descriptor-file name "defaults" to the root of the source-file name. For example, the CLI/SCREEN command:

```
AOLDPROG,$$
```

extracts a screen format from source file OLDPROG.SS, storing its descriptors in descriptor file OLDPROG.DX. Similarly, the CLI/SCREEN command:

```
AOLDPROG.06,$$
```

extracts a screen format from source file OLDPROG.06, storing its descriptors in descriptor file OLDPROG.DX.

You may also use the default filenaming facility with the CREATE SOURCE command.

SOME EXAMPLES OF SCREEN USAGE

Let's examine a few examples of the flow of control among SCREEN's modes: the flow of data to and from source files and descriptor files in each mode is illustrated in Figures 1.7, 1.8, 1.9, and 1.10.

We supply all filenames for SCREEN processing, but we'll let SCREEN append its reserved extensions to distinguish scource files from descriptor files. (In this case, let's suppose we are using CLI/SCREEN: source files get a ".SS" extension and descriptor files get a ".DX" extension.)

Note that all these SCREEN sessions begin and end in EDIT mode, the "master" mode.

SCREEN permits you to name both descriptor files and source files as you wish, including filename extensions. If you do not include extensions in your filenames, SCREEN appends certain reserved extensions automatically. (See Figure 1.6.) Letting SCREEN "do it all for you" simplifies your bookkeeping. For example, you could obtain a printout of all CLI/SCREEN-created source files with the CLI command "LIST/L -.SS". This facility also allows you to use the "same" name for both a source file and a related descriptor file—SCREEN adds the extensions to make unique names that distinguish the files. See "Default Filenames" below.

NOTE: IC/SCREEN does not allow you to specify descriptor file extensions—it always appends the reserved extension to the name you specify.

	FILE TYPE	RESERVED FILE EXTENSION
CLI/SCREEN	Source File	.SS
	Descriptor File	.DX
IC/SCREEN	Source File	.SS
	Descriptor File	.SX/.AX[1]

FIGURE 1.6 SCREEN FILE EXTENSIONS. You can allow SCREEN to use its own filename extensions when it creates files. This simplifies your bookkeeping chores.

Using Filenames in Commands

The filenaming conventions explained above make it unnecessary for you to specify extensions when you name files in SCREEN commands. SCREEN can tell from the command syntax whether you are specifying a descriptor file or a source file.

Specifically, SCREEN uses the following procedures to search for files:

- If you include an extension (possibly just a "."), SCREEN searches only for the file with the exact name you specify. (The operating system regards *filename* and *filename.* as the same name.)
- If you do not include an extension in the source file name *source-file*, SCREEN assumes that you have specified the root name only. It searches for files in the following order, until it finds one:

  1. *source-file*.SS
  2. *source-file*.SR
  3. *source-file*

- If you do not include an extension in the descriptor file name *descriptor-file*, SCREEN assumes that you have specified the root name only. It searches for the following files:

  ☐ IC/SCREEN searches only for the two-part file *descriptor-file*.SX/ *descriptor-file*.AX [1]

1 IC/SCREEN implements the descriptor file as two operating-system files, treating them as a logical unit.

tents of these workfiles with the program's "PRINT" command.) The descriptor files created with CLI/SCREEN may not be used with IC/SCREEN, and vice versa.

- *Source* files are line sequential files[1] that contain Interactive COBOL code. SCREEN's *raison d'etre* is the creation of source files that define display screen formats. SCREEN can also extract screen formats from existing source files, making it convenient to revise an existing program-operator interface. SCREEN can create and read files in both CARD format (with line numbers) and in CRT format (no line numbers). The source files created with CLI/SCREEN and IC/SCREEN are fully compatible.

  Included in each chapter that treats an individual SCREEN *mode* is a figure illustrating how SCREEN reads and writes *descriptor* files and *source* files in that mode.

Descriptor Files

SCREEN descriptor files save the results of "intermediate processing"—operations that do not directly produce source code. The descriptor file contains both the screen image and descriptors (e.g. AUTO, REQUIRED, USING ACCOUNT-OVERDUE) that apply to the image. This data is stored without regard to the eventual coding format—CARD or CRT.

You may create several descriptor files for a single screen image, since many sets of descriptors can apply to the same image.

Source Files

A source file on disk is not directly accessible to EDIT mode processing. Before it can be edited, you must extract a screen format from the source file in SCREEN's ANALYZE mode. You may choose to extract only the screen image from a source file, or the entire format—image and descriptors. SCREEN is compatible with the Interactive COBOL compiler in its ability to retrieve COPY files and bypass comment lines.

The CARD-format or CRT-format source file from which you extract a screen image (or complete screen format) must meet one of the following requirements:

1. It contains the COBOL source line: "SCREEN SECTION."[2]
2. It contains both the header comment line "**SCREEN**" and the trailer comment line "**SCREEN-END**".

In CREATE SOURCE mode, SCREEN writes to disk a source file that defines a screen format. The file includes the header and trailer comment lines mentioned above, so that you may retrieve and revise the format at another time.

You may include SCREEN-created code in a program either with the COPY statement or by inserting the code in the program file with one of the CS editors.

Filenaming Conventions

CS system filenames consist of a *root* name of up to ten alphanumeric characters ("$" is also permitted), and an optional *filename extension.* For SCREEN's purposes, a filename extension consists of a period only, or a period followed by one *or* two alphanumeric characters.

1 SCREEN does not process the indexed source files created by ICEDIT.

2 A complete COBOL program using display screen formats must include this line.

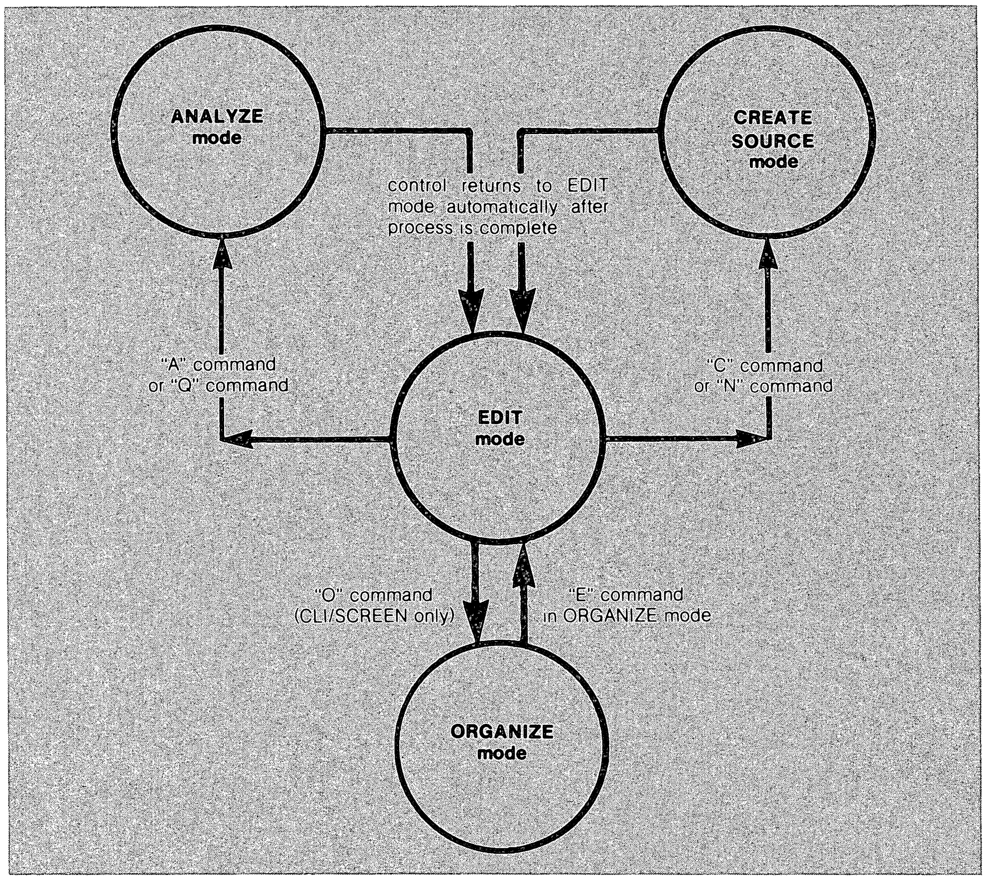

ANALYZE mode
CREATE SOURCE mode
control returns to EDIT mode automatically after process is complete
"A" command or "Q" command
EDIT mode
"C" command or "N" command
"O" command (CLI/SCREEN only)
"E" command in ORGANIZE mode
ORGANIZE mode

FIGURE 1.5 SCREEN MODES. SCREEN operations take place in four modes. The various parts of a line of source code are defined in EDIT, ORGANIZE, and CREATE SOURCE modes. Screen formats are extracted from existing programs in ANALYZE mode.

Calling SCREEN/Exiting From SCREEN

- To call CLI/SCREEN from the Command Line Interpreter, type: SCREEN **CR**.
- To call IC/SCREEN from LOGON, type: ICSCREEN **CR**.
- To exit normally from SCREEN in EDIT Mode, issue SCREEN'S "E" command.

SCREEN returns you to the CLI or LOGON. You may also use the **HOME** key to terminate SCREEN in EDIT, ORGANIZE, and CREATE SOURCE modes. In ANALYZE mode, pressing the **HOME** key returns you to EDIT mode.

- To abort your EDIT mode work at any time, issue SCREEN'S "B" command.

This results in the loss of current information not saved in a descriptor file. You remain in EDIT mode.

SCREEN File Handling

SCREEN saves screen format information in two types of files: *descriptor* files and *source* files.

- *Descriptor* files are for SCREEN's use only—they are workfiles that cannot be directly used by COBOL programs. (CLI/SCREEN users can examine the con-

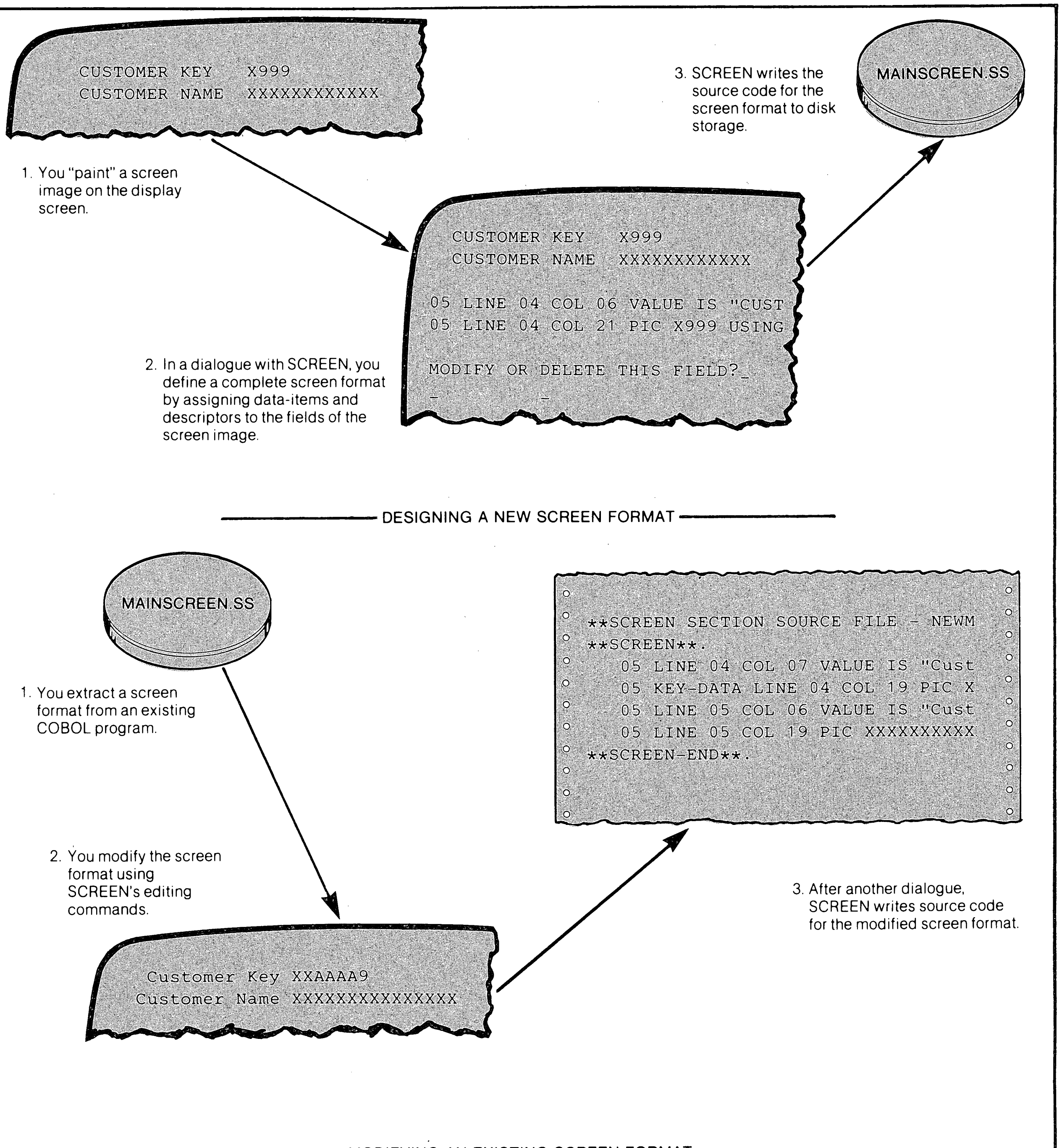

CUSTOMER KEY X999
CUSTOMER NAME XXXXXXXXXXXX
1. You "paint" a screen image on the display screen.
CUSTOMER KEY X999
CUSTOMER NAME XXXXXXXXXXXX
05 LINE 04 COL 06 VALUE IS "CUST
05 LINE 04 COL 21 PIC X999 USING
MODIFY OR DELETE THIS FIELD?_
2. In a dialogue with SCREEN, you define a complete screen format by assigning data-items and descriptors to the fields of the screen image.
3. SCREEN writes the source code for the screen format to disk storage.
MAINSCREEN.SS
DESIGNING A NEW SCREEN FORMAT
MAINSCREEN.SS
1. You extract a screen format from an existing COBOL program.
2. You modify the screen format using SCREEN's editing commands.
Customer Key XXAAAA9
Customer Name XXXXXXXXXXXXXXX
**SCREEN SECTION SOURCE FILE - NEWM
SCREEN.
05 LINE 04 COL 07 VALUE IS "Cust
05 KEY-DATA LINE 04 COL 19 PIC X
05 LINE 05 COL 06 VALUE IS "Cust
05 LINE 05 COL 19 PIC XXXXXXXXXX
SCREEN-END.
3. After another dialogue, SCREEN writes source code for the modified screen format.
MODIFYING AN EXISTING SCREEN FORMAT

FIGURE 1.4 SCREEN OPERATIONS. SCREEN allows you to create new display screen formats, and to adapt existing formats to new applications. CLI/SCREEN users can create Screen Section groups with hierarchical structure.

A **screen element** is an Interactive COBOL statement that defines a field on the display terminal screen. This field may be used to handle *literal* data or a COBOL *data-item*.

A **group** is a structured collection of screen elements that defines a multi-field screen format. The first element in a group is a **group header**—it identifies the group, and is not associated with any single screen field.

SCREEN OPERATING CONCEPTS

SCREEN supports different development strategies. For example, in a single session you might design, document, and code complete display screen formats. On the other hand, you might prefer to design and print a system's screen images initially, postponing source generation until the screen designs are reviewed and the associated data-items are named. Another alternative is to adapt existing screen formats for a new application.

Using SCREEN, you perform all these operations in a series of interactive steps, typing instructions at the keyboard and responding to SCREEN's prompts. (See Figure 1.4.)

SCREEN Modes

You perform each of the operations illustrated in Figure 1.4 in one of SCREEN's operating *modes*.

Use EDIT mode to:

- Create and save a new screen image.
- Edit an existing screen image and save it.
- Call ORGANIZE, ANALYZE, and CREATE SOURCE modes.

Use ANALYZE mode to:

- Extract a screen image from an existing source file.
- Save the extracted screen image for future editing, without destroying the original.

Use ORGANIZE mode (CLI/SCREEN only) to:

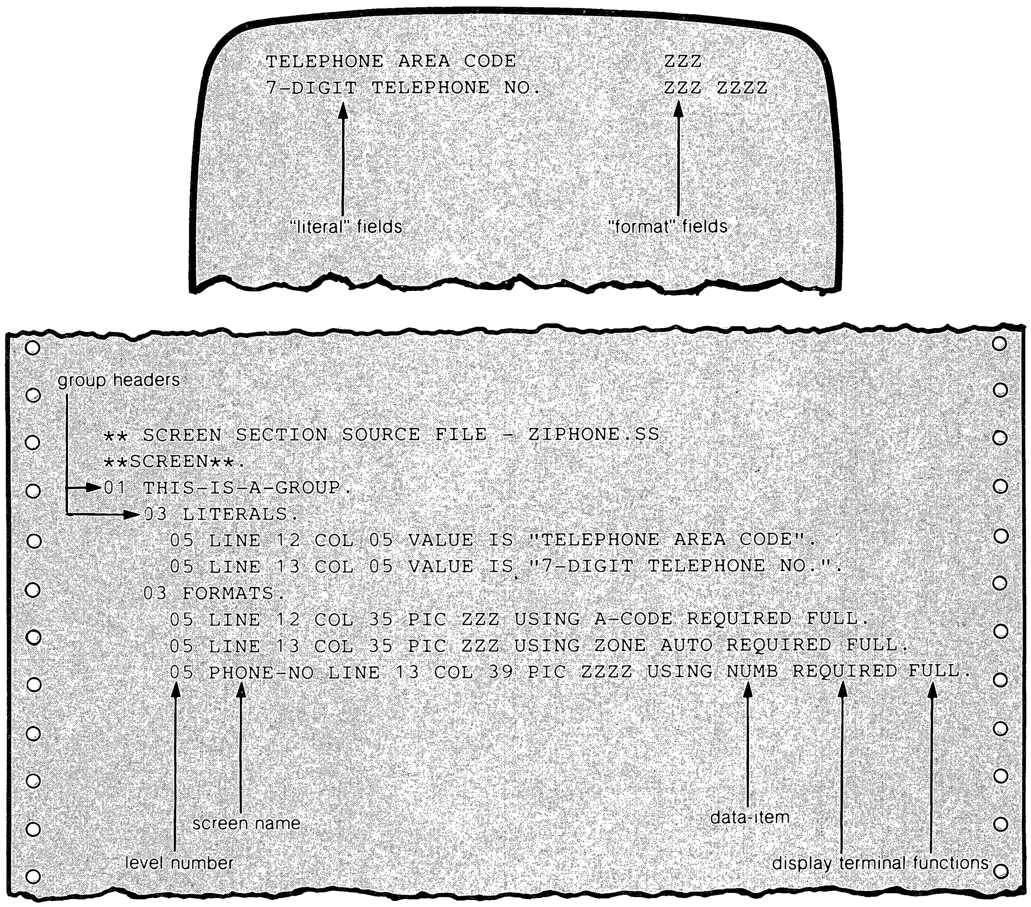
- Group and order the fields in a screen image by creating COBOL Data Division structures.
- Create non-display elements (e.g. group headers) for later coding.

Use CREATE SOURCE mode to:

- Create source code for the current screen image.
- Assign screen names, data names, and display terminal functions to screen fields.

EDIT mode is the "master" mode: all other modes are called from and return control to, EDIT mode. Figure 1.5 illustrates the flow of control among SCREEN modes, and shows how the modes share the task of defining a screen element.

ORGANIZE and CREATE SOURCE modes process a *current* screen image, that is, an image displayed on the terminal screen. A *current* screen image may consist of *new* screen fields just typed in at the keyboard, or a screen format retrieved from storage.



```
TELEPHONE AREA CODE                ZZZ
7-DIGIT TELEPHONE NO.              ZZZ ZZZZ
```

```
** SCREEN SECTION SOURCE FILE - ZIPHONE.SS
**SCREEN**.
01 THIS-IS-A-GROUP.
 03 LITERALS.
   05 LINE 12 COL 05 VALUE IS "TELEPHONE AREA CODE".
   05 LINE 13 COL 05 VALUE IS "7-DIGIT TELEPHONE NO.".
 03 FORMATS.
   05 LINE 12 COL 35 PIC ZZZ USING A-CODE REQUIRED FULL.
   05 LINE 13 COL 35 PIC ZZZ USING ZONE AUTO REQUIRED FULL.
   05 PHONE-NO LINE 13 COL 39 PIC ZZZZ USING NUMB REQUIRED FULL.
```

FIGURE 1.3 SCREEN SECTION SOURCE CODE. Each "literal field" and "format field" is defined by a line of code. "Descriptors" in the code make the screen format a functioning part of an Interactive COBOL program.

A format also includes additional instructions:

- How data is to be displayed or accepted (e.g. BLINK, BLANK WHEN ZERO, REQUIRED).
- What COBOL data-items are to be displayed or updated with the format (e.g. USING ACCT-BALANCE).

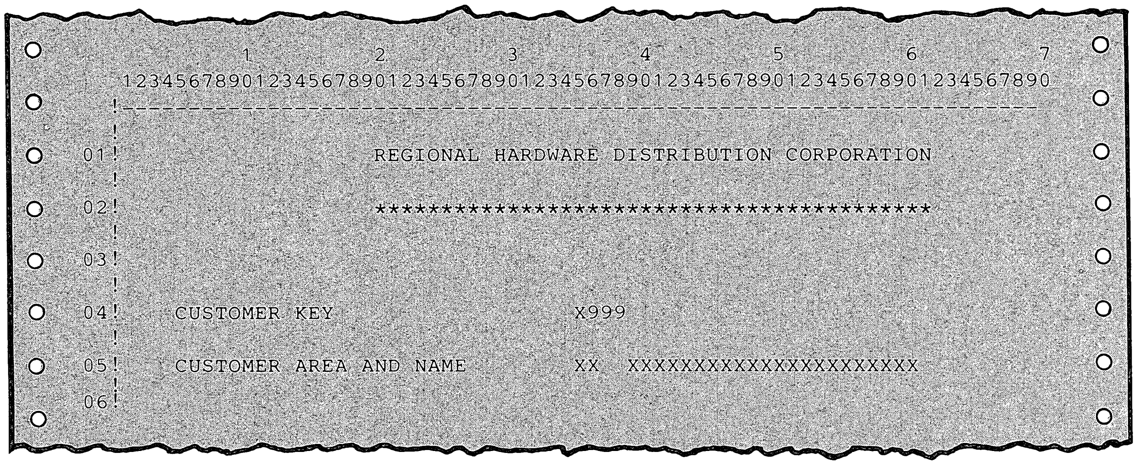
We refer to the purely visual part of the format as the *screen image*, and to the additional information that must be coded as *descriptors*.

A **literal field** is a screen field that displays predefined data, such as headings and messages. The screen element that defines a literal field uses a "VALUE IS" clause to specify the field's contents. Literal fields may contain any displayable ASCII character, including upper/lowercase letters, numerals, and punctuation marks.

A **format field** defines a "window" through which information flows between the operator and the program. The statement that defines a format field uses a standard PICTURE clause to define the type and amount of data that can pass through the window. The statement must also include a clause that assigns the format field to a particular File or Working-Storage data-item and specifies the allowable direction(s) of data flow.

SCREEN's editing and source coding capabilities can help you to maintain and enhance existing Interactive COBOL applications systems. You may extract any screen format defined in an existing source file on disk, edit it, and generate new source, without destroying the original.

SCREEN can provide a printout of any screen format as you are designing or modifying it. (See Figure 1.2.) Because most of an interactive system's functionality is expressed in its screen designs, such printouts serve as excellent system design documentation initially, and as guides for business users after development is complete.



```
            1         2         3         4         5         6         7
   1234567890123456789012345678901234567890123456789012345678901234567890
  !--------------------------------------------------------------------
  !
01!                   REGIONAL HARDWARE DISTRIBUTION CORPORATION
  !
02!                   ******************************************
  !
03!
  !
04!    CUSTOMER KEY                  X999
  !
05!    CUSTOMER AREA AND NAME        XX  XXXXXXXXXXXXXXXXXXXXXX
  !
06!
```

FIGURE 1.2 DOCUMENTING A SCREEN IMAGE. SCREEN's ability to print a screen image makes it easy to review visual designs before coding and to enforce design standards.

SCREEN includes an on-line help facility consisting of an information screen for each of its commands. The "help" displays provide a valuable quick-reference facility.

SCREEN may be executed under the Interactive COBOL runtime system (IC/SCREEN) or from the CLI (CLI/SCREEN). The two versions are very similar operationally. The few differences are noted in the relevant sections of this manual.

SCREEN TERMINOLOGY

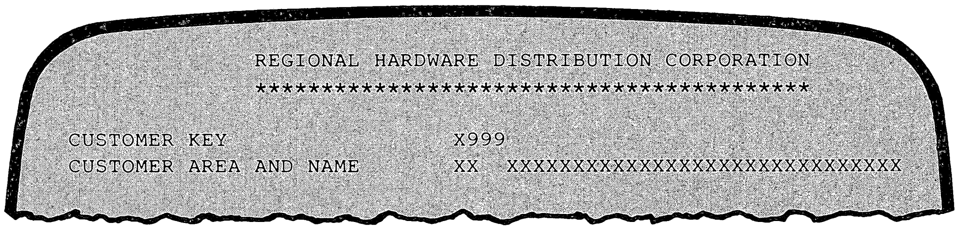
SCREEN writes source code for use in the Screen Section of an Interactive COBOL program. The Screen Section, an Interactive COBOL extension to ANSI standard COBOL, defines the *display screen formats* that control operator-system communication.

The following paragraphs define terms that we will use in discussing SCREEN and its operation. Figure 1.3 applies the terms to a simple display screen format and its corresponding source code.

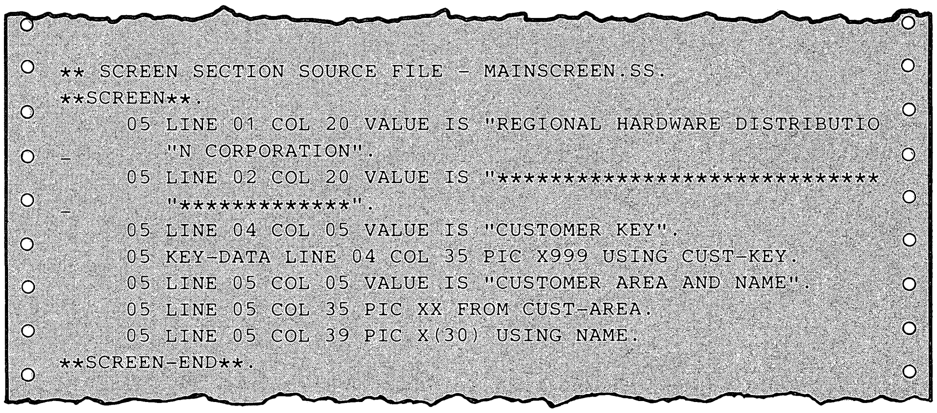
A **display screen format** (abbreviated *screen format*) is a set of information to be displayed to the operator as a single unit. Individual parts of a format are called **fields.** Fields may be used to display data to the operator or to allow the operator to enter data for processing and storage. Position is part of the screen format: each field appears in the same row/column location every time the format is used.

1 Introduction to SCREEN

SCREEN is a computerized tool for designing, coding, and documenting Interactive COBOL display screen formats. Using SCREEN, you *paint* the screen image, typing in literal and data fields exactly as they will appear to the program operator. Then, you instruct SCREEN to write source code for the image, thus defining a *display screen format*. (See Figure 1.1.) SCREEN's direct-printing facility enables you to document both a screen image and its corresponding source code. You may use SCREEN-written source files in any Interactive COBOL program.



```
               REGIONAL HARDWARE DISTRIBUTION CORPORATION
               ******************************************

CUSTOMER KEY                 X999
CUSTOMER AREA AND NAME       XX  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```



```
** SCREEN SECTION SOURCE FILE - MAINSCREEN.SS.
**SCREEN**.
     05 LINE 01 COL 20 VALUE IS "REGIONAL HARDWARE DISTRIBUTIO
-       "N CORPORATION".
     05 LINE 02 COL 20 VALUE IS "*****************************
-       "*************".
     05 LINE 04 COL 05 VALUE IS "CUSTOMER KEY".
     05 KEY-DATA LINE 04 COL 35 PIC X999 USING CUST-KEY.
     05 LINE 05 COL 05 VALUE IS "CUSTOMER AREA AND NAME".
     05 LINE 05 COL 35 PIC XX FROM CUST-AREA.
     05 LINE 05 COL 39 PIC X(30) USING NAME.
**SCREEN-END**.
```

FIGURE 1.1 A DISPLAY SCREEN FORMAT. Using SCREEN, you enter and position screen fields exactly where you want them to appear to the program operator. Then, you instruct SCREEN to write code to define the fields, assigning screen names and display terminal functions.

With a full range of editing commands and function keys as your tools, you can create and modify many screen formats with speed and efficiency. You might design a collection of diverse formats, creating each individually. Or you might design a visually consistent set of formats for an entire applications system by modifying a single "master" format. Consistent screen design makes an important contribution to ease of operation.

Preface

SCREEN, the CS Commercial Systems Screen Format Editor, is a valuable tool for developers committed to delivering easy-to-operate Interactive COBOL applications systems. Programmers will find that SCREEN increases their speed and consistency in coding Interactive COBOL display screen formats.

This manual presents a complete description of SCREEN. Chapter 1 introduces SCREEN operations and provides conceptual information about file handling, the flow of information, and SCREEN's command language conventions. Succeeding chapters treat major SCREEN operations in detail, including complete command references and examples. Also included are appendices, a glossary, and an index.

Typographic Conventions

- The names of keyboard characters appear in boldface type. For example, **CR** and **ESC** identify the "Carriage Return" and "Escape" keys, respectively.
- In examples of program displays, the **ESC** character is represented by a dollar sign ($). SCREEN echoes **ESC** on the display screen as a dollar sign.
- # indicates a numeric argument. SCREEN commands use both positive and negative numbers as arguments, and complex arithmetic expressions as well.
- Lowercase words (*this kind of type in running text*, **`this kind of type in examples and command formats`**) indicate that a name or character string is to be entered. The word conveys the function of the name or string: e.g. *source-file, character-string.*

SCREEN command syntax is such that some parts of a command are required, while other parts are optional. In some cases, you must enter one element from a specific set of choices. In the command reference sections of this manual, we use the following conventions to communicate these ideas:

- Square brackets enclose optional elements of a SCREEN command.
- Braces (curly brackets) enclose a set of elements from which you must choose a single character.

For example, the "A" command format follows:

```
[#] Asource-file [ { ,                } ]$
                 [ { ,descriptor-file } ]
```

In this command format:

1. The numeric argument # is optional.
2. "A" is the code for a SCREEN command (ANALYZE).
3. You should enter the name of a source file where source-file occurs.
4. The remainder of the command line is optional. If you choose to continue the command line, you have a two-way choice:

   □ Enter a comma only.
   □ Enter a comma followed by the name of a SCREEN descriptor file.

Table of Contents

CS Commercial Systems Documents

Overview and Planning

Concepts and Facilities 055-001

Gives an overview of the entire CS computer line. The capabilities of the system for business use are described. A separate chapter is devoted to file management. Special attention is paid to CS's unique screen extension.

Planning Guide 055-002
Addendum June 1979 055-003

Administrators and managers will find this guide useful for selecting a CS System. Basic system components are discussed. Options for future upgrading and expansion are presented so that the best possible decision can be made. The book contains complete information on planning and installation. Also included: site preparation checklist, a floor planning kit including punchouts of system components.

Developer's Tools

ICEDIT: Interactive COBOL Editor 055-004

This manual describes ICEDIT, a line-oriented source program editor specifically created for the writing of Interactive COBOL programs. Included are details of ICEDIT's interactive operation that makes it particularly easy to learn and use. ICEDIT features an interface with other CS system utilities, including the Interactive COBOL compiler.

SCREEN: Screen Format Editor 055-006

This manual describes SCREEN, a special-purpose editor for the design and coding of display screen formats. These formats define the communication between program and operator in Interactive COBOL applications systems. SCREEN users can code new formats and adapt existing formats to different uses.

Language and System Operation

Interactive COBOL
Programmer's Reference 045-011

In this book, COBOL applications programmers will find all the information needed to write COBOL programs and use the Interactive COBOL screen extension. The book is laid out for quick reference, with an index on the back cover. COBOL verbs are described alphabetically. The use of the compiler is explained, with a list of error messages. Information on the use of the debugger is included, with a list of commands. There are many examples throughout the text, and a chapter containing five sample programs. This is a valuable reference book. Also included: a glossary of Interactive COBOL and standard terms, and a list of COBOL reserved words.

Converting COBOL Programs:
CS Interactive COBOL to AOS COBOL 045-009

This manual contains guidelines for converting existing Interactive COBOL code to equivalent AOS COBOL code. It includes procedures for exporting data and program files from a CS System to an AOS Eclipse System.

Operating CS Systems:
A Guide for the System Developer 045-012

This book documents in detail the software tools that system analysts and system developers will use to operate CS Commercial Systems. File structures and the Interactive COBOL runtime systems are explained. The operating system's command language is defined. Use of utilities, including JOBS batch stream processor, disk initialization and hardware diagnostics is described.

IC/RJE80 Remote Job Entry
Communications Utility User's Guide 055-040

IC/RJE80 implements the nearly-universal inter-computer communications protocol—2780/3780. The manual describes its features and provides detailed operating instructions. It has a tutorial, an alphabetical command listing, a glossary and error messages.

APPENDIX E: SCREEN Function Key Templates

SHIFT	Delete Line	Split Line	Format					
	Delete Character	Insert / Replace	Literal					

DASHER D2, D3, D4, D5 Terminals
Function Keys 1-8

SHIFT	Delete Line	Split Line	Format					
	Delete Character	Insert / Replace	Literal					

DASHER D2, D3, D4, D5 Terminals
Function Keys 1-8

SHIFT			
	Erase Command	Edit Command	Help

D2, D3, D4, D5
Functions Keys 9-11

SHIFT			
	Erase Command	Edit Command	Help

D2, D3, D4, D5
Function Keys 9-11

Delete Line	Split Line	Format			SHIFT
Delete Character	Insert / Replace	Literal			

DASHER D200
Terminal

Function Keys 1-5

					SHIFT
			Erase Command	Edit Command	

Function Keys 6-10

					SHIFT
Help					

Function Keys 11-15

Index