JYACC FORMAKER Configuration Guide

Contents

1 Summary of Configuration Utilities

This manual describes a number of utility programs that fall under the rubric of configuring JYACC FORMAKER itself or applications that use it. One group is for creating and modifying files that tell JYACC FORMAKER how to run on particular computers and terminals; another group of programs enables you to list, reformat, and otherwise manipulate screens.

Hardware Configuration

modkey	A specialized full-screen editor for inspecting, creating, and modifying key translation files.
Key file	Not actually a utility; this section describes how to format key translation files by hand.
key2bin	Converts key translation files to binary format.
Video file	Not actually a utility; this section describes how to create video configuration files for terminals and displays.
vid2bin	Converts video files to binary format.
term2vid	On UNIX and related systems, creates a primitive video file from a terminfo or termcap entry.

Software Configuration

Message file	Not a utility; this section describes how to prepare files of messages for use with the msg2bin utility and the JYACC FORMAKER library.
msg2bin	Converts message text files to binary format.
Setup file	Not actually a utility; this section describes the setup or environment variables supported by JYACC FORMAKER, and tells how to prepare setup files.
var2bin	Converts setup variable files to binary format.
f2r4	Converts Release 3 screen files to Release 4 format.
bin2c	Converts binary files to and from C source code, so that they may be made memory-resident.
bin2hex	Converts binary files to and from an ASCII format for exchange with other computers.
formlib	Collects screen files in a single library file, to simplify the management of large numbers of screens.
lstform	Creates a listing telling everything about a screen.
txt2form	Creates a read-only screen from a text file, for quick construction of help screens and such.

2 Features and Options Common Among Utilities

The following section describes command-line options and file-handling procedures shared by most or all of the JYACC FORMAKER configuration utilities. When a utility deviates from this standard, as a few do, the section describing that utility will make it clear.

Command-line options are identified by a leading hyphen. You can always obtain a usage summary from any JYACC FORMAKER utility by invoking it with the -h option, for instance

```
formlib -h
```

The utility in question will print a brief description of its command line parameters, including the input files and all command options. Utilities that can process multiple input files will also support a -v option. It causes them to print the name of each input file as it is processed.

2.1 Input and Output Files

With a few exceptions, utilities accept multiple input files. Some then combine information from the inputs to create a listing; others perform some transformation on each input individually. No utility will ever overwrite an input file with an identically named output file; if your command calls for such an action, an error message will be the only result. Most utilities will also refuse to overwrite an existing output file; you may force the overwrite with the -f option.

Utilities that create a listing, such as lstform, support a -o option, which directs the output to a named file. For example:

```
lstform -omylist *.frm
```

lists all the screens in the current directory, and places the listing in a file named mylist. A special form of this option, -o-, sends the program's output to the standard output file rather than to a disk file.

Utilities that generate one output file for each input will, by default, give output files the same name as the corresponding input, but with a different extension. Each utility has a different default extension (see the next section for a table); in addition, each one supports a -e option that enables you to specify the output file extension. For example:

```
form2r4 -enew mytop.mnu myscreen.win
```

converts the Release 3 screens mytop.mnu and myscreen.win to Release 4 format, and puts the new screens in mytop.new and myscreen.new. The form -e- makes the output file extension null.

Certain utilities that normally generate multiple output files also support the -o option; it causes them to place all the output in the file named in the option. For instance,

```
f2struct -oscreenrecs.h screen1.jam screen2.jam
```

generates C data structures for screen1 and screen2, and places them both in screenrecs.h. Without the -o option, it would have created two output files, screen1.h and screen2.h.

By default, if an input filename contains a path component, a utility will strip it off in generating the output filename; this usually means that output files will be placed in your default directory. You may supply a -p option to have the path left on, that is, to create the output file in the same directory as the input.

2.2 File Names and Extensions

JYACC FORMAKER runs on several different operating systems, which deal in rather different ways with file naming. We must therefore define a few terms for use in the following sections:

full name	Everything you and the operating system need to know in order to identify a file uniquely.
name	The only truly arbitrary part of the full name, identifying anything at all. May not be omitted.
path	A prefix to the name that tells where (on what device, directory, or user ID) a file resides. If omitted, defaults to a location known to the operating system, such as a working directory.
extension	A prefix or suffix to the name that tells what sort of information is in the file. May be omitted.

JYACC FORMAKER does not attempt to understand or alter paths; it just uses them as you supply them. It knows about a class of path separator characters, and assumes that the path ends at the rightmost such character in the full name.

JYACC FORMAKER, like many other software systems, uses extensions to identify the contents of a file. (Where proper identification is crucial, it puts "magic numbers" in the files themselves.) We have tried to make our conventions flexible: extensions are not required, but are supplied by default, and the default can always be overridden. There are three distinct operations involving file extensions:

1. Finding and modifying files. xform and the JYACC FORMAKER run-time system assume that screen files have a common extension, such as jam. They will add that extension to any filename that does not already contain one before attempting to open it. This rule does not operate if extensions are ignored.
2. Creating new files. Utilities other than xform transform files of one type to another, and must name the output file differently from the input. They do it by replacing the input file's extension, or adding one if there was none. This rule operates even if extensions are ignored, in which case the new extension is always added.
3. Creating data structures. The utilities f2struct and bin2c create data structures from screen files. They name the structures by removing the path and extension from the input filename. If extensions are ignored, only the path is removed.

2.3 Configuring File Extensions and Rules

There are three parameters that control how JYACC FORMAKER uses file extensions:

1. A flag telling whether JYACC FORMAKER should recognize and replace extensions, or ignore them.
2. Another flag telling whether the extension should go at the beginning or the end of the filename.
3. The character that separates the extension from the name (zero means no separator).

The default values for these parameters are recognize, end, and period respectively. You may alter them using the SMUSEEXT setup variable; but be aware that people working on the same project should use the same rules, or confusion is likely to result.

Here is a list of the default extensions used by utility programs.

Utility	Extension
bin2c	language-dependent bin2hex
	none f2r4
	no change f2struct
	language-dependent formlib
	none key2bin
	bin lstform
	lst modkey
	keys msg2bin
	bin term2vid
	vid txt2form
	none var2bin
	bin vid2bin
	bin

2.4 Ordering of Options and Other Arguments

Most utilities take as arguments an output file, a list of input files, and some options. If present, the output file precedes the input file list. Options may be placed anywhere after the utility name; they may be supplied separately (each with its own hyphen), or together (all following a single hyphen); the two commands

```
lstform -fti myscreen
lstform -f -t -i myscreen
```

are equivalent. Option letters may be either upper- or lower-case. On certain systems such as VMS and MS-DOS, where the prevalent "switch character" is / rather than - , both are supported.

2.5 Notation

The rest of this chapter describes each configuration utility individually. There are also a few sections that tell how to prepare input files for some of the utilities. Each section contains the following information:

- The name and purpose of the utility.
- A synopsis of its usage, that is, what you type on the command line to run it. Here, literal input appears in boldface, and parameters that you supply appear in normal type. Optional parameters are enclosed in square brackets []. An ellipsis ... indicates that the previous parameter may be repeated. Command options are simply listed after a hyphen, as -abcdefg; you may select any combination of them.
- A complete description of the utility's inputs, outputs, and processing.
- Where applicable, a list of error conditions that may prevent the utility from doing what you tell it.

NAME

bin2c - convert any binary file to C source code

SYNOPSIS

bin2c [-flv] textfile binfile [binfile ...]

DESCRIPTION

This program reads binary files created by other JYACC FORMAKER utilities, and turns each one into C code for a character array initialized to the contents of the file. Such arrays may then be compiled, linked with your application, and used as memory-resident files. This utility combines the arrays from all the input files in a single output file; each array is given a name corresponding to the name of the input file, with the path and extension stripped off.

Files that can be made memory-resident include the following types:

1. screens (created by xform)
2. key translation files (key2bin)
3. setup variable files (var2bin)
4. video configuration files (vid2bin)
5. message files (msg2bin)

The command options are interpreted as follows:

-f Overwrite an existing output file.
-l Force the array names derived from the input file names to lower-case characters.
-v Print the name of each input file on the terminal as it is processed.

ERROR CONDITIONS

Insufficient memory available. Cause: The utility could not allocate enough memory for its needs. Corrective action: None.

File "%s" already exists; use '-f' to overwrite. Cause: You have specified an output file that already exists. Corrective action: Use the -f flag to overwrite the file, or use another name.

Cannot open "%s" for writing. Cause: An output file could not be created, due to lack of permission or perhaps disk space. Corrective action: Correct the file system problem and retry the operation.

Cannot open "%s" for reading. Cause: An input file was missing or unreadable. Corrective action: Check the spelling, presence, and permissions of the file in question.

Error reading file "%s" Cause: The utility incurred an I/O error while processing the file named in the message. Corrective action: Retry the operation.

Error writing file "%s" Cause: The utility incurred an I/O error while processing the file named in the message. Corrective action: Retry the operation.

NAME

bin2hex - convert binary to and from hex ASCII, for transport

SYNOPSIS

bin2hex -cx [-flv] hexfile binary [binary ...]

DESCRIPTION

The bin2hex utility translates binary files of any description to and from a hexadecimal ASCII representation. It is useful for transmitting files between computers. The utility is very straightforward; no translation of any sort is attempted.

Either the -c or the -x switch is required; the others are optional. Here is a summary:

-c	Create a text file from one or more binary files; the text file's name is the first file argument, and the rest are binaries.
-f	Overwrite any existing output files.
-l	Force the filename arguments to lower case.
-v	Print the name of each binary on the terminal as it is processed.
-x	Extract all the binary files contained in an ASCII source. Selective extraction is not supported.

ERROR CONDITIONS

Error reading %s Error writing %s Cause: The utility incurred an I/O error while processing an input or output file. This message will usually be accompanied by a more specific, system-dependent message. Corrective action: Correct the system-dependent problem, if possible, and retry the operation.

%s already exists %s already exists, it is skipped Cause: The command you have issued would overwrite an existing output file. Corrective action: If you are sure you want to destroy the old file, reissue the command with the -f option.

NAME

f2struct - create program data structures from screens

SYNOPSIS

```
f2struct [-fp] [-ooutfile] [-glanguage] screen
     [screen ...]
```

DESCRIPTION

This program creates program source files containing data structure definitions matching the input files. The output file will contain a single structure bearing the name of the screen.

The language in which the structures are created, and the extension attached to output file names, are both selected by the -g option. The name of the desired language follows the g, and must be in a table compiled into the utility. This option may be placed between file names in the command line to enable files to be created in different languages. Indeed, the same input file can be named twice to create, say, both C and Pascal structures:

```
f2struct -gc address.jam -gpascal address.jam
```

You can modify the conversions or write code to handle more languages, as described in the utility source code; see below. The other command options are interpreted as follows:

-f	Directs the utility to overwrite an existing output file.
-p	Directs the utility to create each output file in the same directory as the corresponding input file.
-o	Causes all output to be placed in outfile.

When a screen name is given to a structure, the screen file's extention is stripped off. Each field of the structure will be named after a field of the screen. If a screen field has no name fldm is used, where m is the field number. The types of the structure fields are derived from the input field data type and character edits, according to the following rules.

1. If a field has one of the following data type edits, it is used.

C data type	mnemonic
omit from struct	FT_OMIT
integer	FT_INT
unsigned integer	FT_UNSIGNED
short integer	FT_SHORT
long integer	FT_LONG
floating point	FT_FLOAT
long floating	FT_DOUBLE
character string	FT_CHAR

2. If a field has no data type edit but has a digits-only or numeric character edit, its type is unsigned int or double respectively.
3. All other fields are of type character string.

Omit from struct is a special type that prevents the field from being included in any structure.

If a field has multiple occurrences, the corresponding structure member will be declared as an array.

ERROR CONDITIONS

Language %s undefined. Cause: The language you have given with the -g option has not been defined in the utility's tables. Corrective action: Check the spelling of the option, or define the language ito the utility.

%s already exists. Cause: You have specified an existing output file. Corrective action: Use the -f option to overwrite the file, or use a different name.

%s has an invalid file format. Cause: An input file is not of the expected type. Corrective action: Check the spelling and type of the offending file.

'%s' has no data to convert. Cause: An input file is empty, or does not have the names you specified. Corrective action: Check the names.

Not enough memory to process '%s'. Unable to allocate memory. Cause: The utility could not allocate enough memory for its needs. Corrective action: None.

At least one form name is required. Cause: You have not given any screen files as input. Corrective action: Supply one or more screen file names.

NAME

f2r4 - convert Release 3 screens to Release 4 format

SYNOPSIS

f2r4 [-ja1udvxfp] [-eext] screen [screen ...]

DESCRIPTION

F2r4 converts Release 3 screens to Release 4 format. It gives each new screen the same name as the old one. It is strongly recommended that you run this utility in a different directory from where your original Release 3 forms reside.

There are a few nontrivial changes involved in this conversion. One is that JYACC FORMAKER control fields may be converted to control strings that do not occupy space on the screen. Another is that jam_first fields, and jam_pf1 fields on read-only screens, have been replaced by screen entry functions and AUTO control strings, respectively. The following options are provided to control the conversion of JYACC FORMAKER control fields to control strings:

-j	Do not convert control fields to control strings.
-a	Do not convert jam_pf1 fields to AUTO control strings.
-1	Do not convert the jam_first attached function to a screen entry function. This is a one, not an ell.
-u	Convert all unprotected fields to menu fields. This is useful for Release 3 item selection screens; in Release 4, item selection fields must have the MENU bit set.
-d	Do not delete jam_d_dflt and jam_f_dflt fields from the screen. These fields were used in earlier releases of JYACC FORMAKER to denote default field and display characteristics.
-v	Print the name of each screen as it is processed.
-x	Delete the input extension.
-f	Overwrite existing output files. Use cautiously: if you do this in the directory where your Release 3 screens reside, the Release 4 screens will be created successfully but your original screens will disappear.
-e	Followed by a character string, makes that string the extension for output files.
-p	Create each output file in the same directory as the corresponding input file. This option is not recommended.

For further information regarding control strings see the Author's Guide.

ERROR CONDITIONS

%s is not a release 3 form Cause: An input file is not of the correct type (this is•determined by a sort of magic number check.) Corrective action: Make sure you haven't already converted•the file, and that it is a screen in the first place.

Unable to allocate memory. Cause: The utility couldn't get enough memory for its needs. Corrective action: None.

File %s already exists. Use '-f' to overwrite or '-e' to append an extension to the output file. Cause: The output file you have named already exists. Corrective action: Be cautious in your use of -f. JYACC suggests•that you create the Release 4 screens in an empty directory, not•in the directory where the Release 3 screens reside.

)

NAME

formlib - screen librarian

SYNOPSIS

formlib -crdxt [-flv] library [screen ...]

DESCRIPTION

Formlib is a screen librarian. It creates libraries of screens that have been created with the JYACC FORMAKER authoring utility. The representation of the screen in the library is the original binary version. This utility enables one to store many screens in a single file and not clutter a directory with many small screen files.

Exactly one of the unbracketed command options must be given; it controls the action of the utility, as follows.

-c	Create a new library, placing in it all the screens named.
-r	Add the screens to the named library, replacing any that are already there.
-d	Delete the screens named from the library.
-x	extract the screens from the named library, placing them in the current directory. If no screens are named, everything in the library will be extracted.
-t	List the current contents of the library.

There is also a -l option which may be used in conjunction with any of the options listed above, and will force the list of screen names to lower case; a -f option that will cause an existing library to be overwritten; and a -v option that will cause the utility to print the name of each screen as it is processed.

To create a new library, use the -c option. For example:

```
formlib forms -c form1 form2
```

This creates a new file called forms containing the same binary representations of form1 and form2 as are in their respective files.

To see what screens are catalogued in the library file, the -t option is used. For example, on the above file forms:

```
formlib forms -t
```

would list:

```
FORMLIB--Librarian for forms created by JYACC FORMAKER.
Copyright (C) 1988 JYACC, Inc.

LIBRARY 'forms' contains:
form1
form2
```

If you wish to add a new screen to the library, or replace one already in the library with a new version, use the -r option. For example, to add the screen form3 to the library forms:

```
formlib forms -r form3
```

Now if you list the contents of forms using the -t option, you get:

```
     FORMLIB--Librarian for forms created by JYACC FORMAKER.
     Copyright (C) 1988 JYACC, Inc.

     LIBRARY 'forms' contains:
     form1
     form2
     form3
```

If you need to obtain one or more of the forms for use by an application or for modification by the JYACC FORMAKER utility, you can extract it from the library file with the -x option. For example:

```
     formlib forms -x form2
```

will create a file called form2 whose contents are the binary representation of that form just as it was created with xform.

If a form is no longer needed and you wish to delete it from the library, the -d option is used. For example:

```
     formlib forms -d form1
```

would delete form1 from the library file forms. Now if you list the contents of forms using the -t option, you get:

```
     FORMLIB--Librarian for forms created by JYACC FORMAKER.
     Copyright (C) 1988 JYACC, Inc.

     LIBRARY 'forms' contains:
     form2
     form3
```

ERROR CONDITIONS

Library `%s' already exists; use `-f' to overwrite. Cause: You have specified an existing output file. Corrective action: Use the -f option to overwrite the file, or use a different name.

Cannot open `%s'. Cause: An input file was missing or unreadable. Corrective action: Check the spelling, presence, and permissions of the file in question.

Unable to allocate memory. Insufficient memory available. Cause: The utility could not allocate enough memory for its needs. Corrective action: None.

File `%s' is not a library. Cause: The named file is not a form library (incorrect magic number). Corrective action: Check the spelling and existence of your library.

`%s' not in library. No forms in library. Cause: A screen you have named is not in the library.

Corrective action: List the library to see what's in it, then retry the operation.

Temporary file `%s' not removed. Cause: The intermediate output file was not removed, probably because of an error renaming it to the real output file. Corrective action: Check the permissions and condition of the files, then retry the operation.

NAME

Key file - keyboard translation table source

DESCRIPTION

JYACC FORMAKER uses a key translation table to map keys you type into a keyboard-independent set of codes, thus relieving applications of the need to know about different terminals. This section tells how to format a text file containing a key translation table.

You can also construct a key translation table using the modkey utility, an interactive program which is documented elsewhere in this chapter. Modkey is recommended if you are defining a key translation file from scratch, or if you are new to JYACC FORMAKER. After creating a key file, either by hand or with modkey, you will need to translate it to binary with the key2bin utility (documented separately) and assign the binary file to the SMKEY setup variable for use by the run-time system.

8.1 Key Translation File Format

The key translation file contains one line for each key. Each line has the following components:

logical-value(label) = character-sequence

Logical-value can either be one of the mnemonics defined in the file smkeys.h , or a hexadecimal value. See Section 8.2 for a table. Only modkey differentiates between the two methods; they operate identically in the run-time system. In modkey, entries specified with hexadecimal values will all appear on the miscellaneous key definition screen, while entries specified with mnemonics will be shown on one of the four screens devoted to specific types of keys. At most 24 entries may be specified in hexadecimal.

The label, which must be enclosed in parentheses, should be a short string that appears on top of the key on your keyboard. It will be stored in the key translation file and can be accessed at run-time through various library functions and the %k escape in status-line messages (see d_msg_line). Key labels, or keytops as they are sometimes called, can be invaluable in user help messages and prompts. The label and parentheses are optional; the following equal sign, however, is required.

The character-sequence is up to six characters that JYACC FORMAKER will translate to the logical value on the left. ASCII control characters may be represented by mnemonics, listed in Section 8.3, or as hex numbers. Displayable characters such as letters can just be typed in. Blanks between characters are ignored; if a space occurs in the sequence, it should be entered as SP.

Lines beginning with a pound sign # will be treated as comments, i.e. ignored, by key2bin. Some representative key translation file entries follow.

```
EXIT(F1) = SOH @ CR
XMIT(Enter) = SOH O CR
TAB  = HT
BACK = NUL SI
BKSP = BS
RARR = ESC [ C
LARR = ESC [ D
UARR = ESC [ A
DARR = ESC [ B
0x108 = DEL
PF2(F2) = SOH A CR
```

If the same mnemonic appears more than once in the file, the last occurrence will appear in the modkey utility. If duplicate right-hand sides appear with different logical values, unpredictable results will occur. Incorrectly formatted lines will cause key2bin to abort.

8.2 Key Mnemonics and Logical Values

The following table lists JYACC FORMAKER's logical key values, their mnemonics, and their actions. Entries followed by "**" are required for xform to work properly; those followed by "*" are strongly recommended.

EXIT 0x103**	exit	SPF1 0x4101*
XMIT 0x104**	transmit	SPF2 0x4201*
HELP 0x105*	help	SPF3 0x4301*
FHLP 0x106	screen-wide help	SPF4 0x4401*
BKSP 0x108*	backspace	SPF5 0x4501*
TAB 0x109*	tab	SPF6 0x4601*
NL 0x10a*	new line	SPF7 0x4701
BACK 0x10b*	backtab	SPF8 0x4801
HOME 0x10c*	home	SPF9 0x4901
DELE 0x10e*	delete character	SPF10 0x4a01
INS 0x10f*	insert character	SPF11 0x4b01
LP 0x110	local print	SPF12 0X4c01
FERA 0x111*	field erase	SPF13 0x4d01
CLR 0x112*	clear unprotected	SPF14 0x4e01
SPGU 0x113	scroll up a page	SPF15 0x4f01
SPGD 0x114	scroll down a page	SPF16 0x5001
LARR 0x118*	left arrow	SPF17 0x5101
RARR 0x119*	right arrow	SPF18 0x5201
DARR 0x11a*	down arrow	SPF19 0x5301
UARR 0x11b*	up arrow	SPF20 0x5401
REFR 0x11e*	refresh screen	SPF21 0x5501
EMOH 0x11f	go to last field	SPF22 0x5601
CAPS 0x110	change shift ind.	SPF23 0x5701
INSL 0x120	insert occurrence	SPF24 0x5801
DELL 0x121	delete occurrence	
ZOOM 0x122	zoom on field	APP1 0x6102
		APP2 0x6202
PF1 0x6101		APP3 0x6302
PF2 0x6201*		APP4 0x6402
PF3 0x6301*		APP5 0x6502
PF4 0x6401*		APP6 0x6602
PF5 0x6501		APP7 0x6702
PF6 0x6601*		APP8 0x6802
PF7 0x6701*		APP9 0x6902
PF8 0x6801*		APP10 0x6a02
PF9 0x6901*		APP11 0x6b02
PF10 0x6a01		APP12 0x6c02
PF11 0x6b01		APP13 0x6d02
PF12 0x6c01		APP14 0x6e02
PF13 0x6d01		APP15 0x6f02
PF14 0x6e01		APP16 0x7002
PF15 0x6f01		APP17 0x7102
PF16 0x7001		APP18 0x7202
PF17 0x7101		APP19 0x7302
PF18 0x7201		APP20 0x7402
PF19 0x7301		APP21 0x7502
PF20 0x7401		APP22 0x7602
PF21 0x7501		APP23 0x7702
PF22 0x7601		APP24 0x7802
PF23 0x7701		
PF24 0x7801		

8.3 ASCII Character Mnemonics

This table lists two- and three-letter ASCII mnemonics for control and extended control characters.

		DLE	0x10			DSC	0x90
SOH	0x01	DC1	0x11			PU1	0x91
STX	0x02	DC2	0x12			PU2	0x92
ETX	0x03	DC3	0x13			STS	0x93
EOT	0x04	DC4	0x14	IND	0x84	CCH	0x94
ENQ	0x05	NAK	0x15	NEL	0x85	MW	0x95
ACK	0x06	SYN	0x16	SSA	0x86	SPA	0x96
BEL	0x07	ETB	0x17	ESA	0x87	EPA	0x97
BS	0x08	CAN	0x18	HTS	0x88		
HT	0x09	EM	0x19	HTJ	0x89		
NL	0x0a	SUB	0x1a	VTS	0x8a		
VT	0x0b	ESC	0x1b	PLD	0x8b	CSI	0x9b
FF	0x0c	FS	0x1c	PLU	0x8c	ST	0x9c
CR	0x0d	GS	0x1d	RI	0x8d	OCS	0x9d
SO	0x0e	RS	0x1e	SS2	0x8e	PM	0x9e
SI	0x0f	US	0x1f	SS3	0x8f	APC	0x9f
SP	0x20	DEL	0x7f				

NAME

key2bin - convert key translation files to binary

SYNOPSIS

key2bin [-pv] [-eextension] keyfile [keyfile ...]

DESCRIPTION

The key2bin utility converts key translation files into a binary format for use by applications using the JYACC FORMAKER library. The key translation files themselves may be generated by JYACC modkey, which is documented elsewhere in this chapter, or created with a text editor according to the rules described in the section on key files in this chapter.

Keyfile is the name of an ASCII key translation file. By convention it is an abbreviation of the terminal's name, plus a tag identifying it as a key translation file; for instance, the key translation file for the Wyse 85 is called W85keys. The utility first tries to open its input file with the exact name you put on the command line; if that fails, it appends keys to the name and tries again. The output file will be given the name of the successfully opened input file, with a default extension of bin.

The command options are interpreted as follows:

-p	Place the binary files in the same directories as the input files.
-v	List the name of each input file as it is processed.
-e	Use the output file extension that follows the option letter in place of the default bin.

To make a key translation file memory-resident, first run the binary file produced by this utility through the bin2c utility to produce a program source file; then compile that file and link it with your program.

ERROR CONDITIONS

File '%s' not found Neither '%s' nor '%s' found. Cause: An input file was missing or unreadable. Corrective action: Check the spelling, presence, and permissions of the file in question.

Unknown mnemonic in line: '%s' Cause: The line printed in the message does not begin with a logical key mnemonic. Corrective action: Refer to smkeys.h for a list of mnemonics, and correct the input.

No key definitions in file '%s' Cause: Warning only. The input file was empty or contained only comments. Corrective action: None.

Malloc error Cause: The utility could not allocate enough memory for its needs. Corrective action: None.

Cannot create '%s' Error writing '%s' Cause: An output file could not be created, due to lack of permission or perhaps disk space. Corrective action: Correct the file

system problem and retry
the operation.

NAME

lstform - list selected portions of screens

SYNOPSIS

```
lstform [-adijmnpstv] [-eext] [-ooutfile] screen
     [screen ...]
```

DESCRIPTION

This program lists selected portions of screen files. By default, all the data about each field in each screen is included. Using command options, however, you can direct that only some of the display be generated. The command options are interpreted as follows:

- -a List default field characteristics for the screen.
- -d List display data.
- -e Generate one output file, with the extension following the option letter, for each input file.
- -i List initial field data, including offscreen data.
- -j List JYACC FORMAKER control strings.
- -m List data relevant to the screen as a whole: border, screen entry function, etc.
- -n Include a snapshot of the screen showing underscores in place of fields.
- -o Send the output to a single file whose name follows the option letter.
- -p Place output files in the same directory as the corresponding inputs.
- -s Include a snapshot of screen showing display data and initial onscreen contents of fields.
- -t List all field edits.
- -v Print the name of each screen on the terminal as it is processed.

ERROR CONDITIONS

Error opening input file. Cause: An input file was missing or unreadable. Corrective action: Check the spelling, presence, and permissions of the file in question.

Error opening output file. Cause: An output file could not be created, due to lack of permission or perhaps disk space. Corrective action: Correct the file system problem and retry the operation.

Unable to allocate memory. Can't allocate memory. Cause: The utility could not allocate enough memory for its needs. Corrective action: None.

Error reading form file. Error writing list file. Cause: The utility incurred an I/O error while processing the file named in the message. Corrective action: Retry the operation.

EXAMPLE

The following is an annotated example of the output of this
program when run on the summary (PF5) window of xform.
Ellipses ... indicate abridgements.

```
FORM 'fm_summ_wi'
-----------------
FORM DATA:
----------
form size         12 lines; 78 columns
Border style 0    REVERSE VIDEO HIGHLIGHTED BLUE
Background color  WHITE
Form help screen  'fm_sum0hlp'

Snapshot with initial data
--------------------------
1                   2         3         4         5         6         7
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678
--------------------------------------------------------------------------------

Field Data Summary

Name                                      Char Edits unfilter
Length        (Max     )  Onscreen Elems      Offset       (Max Items      )

Display Att:
Field Edits:
Other Edits:



Snapshot with underscores
-------------------------
1                   2         3         4         5         6         7
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678
--------------------------------------------------------------------------------

Field Data Summary

Name ____________________________  Char Edits ________                _
Length ___  (Max ___ )  Onscreen Elems ___  Offset ___ _ (Max Items ____ )

Display Att: ____________________________________________________________
Field Edits: ____________________________________________________________
Other Edits: ____________________________________________________________
_____________________________________________________________

--------------------------------------------------------------------------------
```

```
FIELD DATA:
-----------
Field number      1   (line 4, column 8, length = 31)
Display attribute UNDERLINED HIGHLIGHTED YELLOW
Field edits       MENU-FIELD; RETURN-ENTRY;
Help              fm_nam1hlp

Field number      2   (line 4, column 52, length = 8)
Scrolling values  max items      =  7; increment =  1; circular;
Display attribute UNDERLINED HIGHLIGHTED YELLOW
Field edits       PROTECTED FROM:  ENTRY OF DATA;  CLEARING;  VALIDATION;
Help              fm_chr1hlp

initial data:
item  1: unfilter
...
item  7: reg exp

...

Field number      4   (line 5, column 10, length = 3)
Display attribute UNDERLINED HIGHLIGHTED YELLOW
Character edits   DIGITS-ONLY
Field edits       RIGHT-JUSTIFIED; DATA-REQUIRED;
Range             1 TO 255
Help              fm_len1hlp

...

Field number      15   (line 9, column 17, length = 60)
Vertical array    2 elements; offset between elements = 1
Array field numbers : 15 16
Display attribute HIGHLIGHTED YELLOW
Field edits       WORD-WRAP; PROTECTED FROM:  ENTRY OF DATA;  TABBING INTO;
CLEARING;  VALIDATION;

DISPLAY DATA:
-------------
Display text      Field Data Summary
Position          line = 2; column = 31; length = 18
Display attribute CYAN

Display text      Name
Position          line = 4; column = 3; length = 4
Display attribute CYAN

...
```

NAME

Message file - JYACC FORMAKER error message file format

DESCRIPTION

During initialization, the binary message file identified by the environment variable SMMSGS is read into memory. It contains error messages and other text used by the JYACC FORMAKER library, such as the 3-letter abbreviations used for month and day of week names; it can also contain user messages. The binary message file is created by msg2bin, q.v., from a text file. This section describes the text file.

Each line of the message file should have the form

tag = message

The tag is a single word; system message tags have standard prefixes, listed below, and matching identifiers defined in smerrors.h . You may use any tag for your messages that does not begin with a system prefix. The equal sign is required, and the message to its right is completely arbitrary, except that it may not contain newlines (carriage returns). If you have a long message, you may end the first line or lines with a backslash \ and continue it on the next. A pound sign # at the beginning of a line makes it a comment; msg2bin ignores comments.

System messages are identified by one of the following reserved tag prefixes, and have identifiers defined in the system include file smerror.h :

SM	Denotes messages and strings used by the JYACC FORMAKER run-time library.
FM	Identifies messages issued by the screen editor.

Appendix A contains a list of all the system messages as distributed by JYACC, plus explanations and actions recommended for recovery.

The msg2bin utility uses tags only to distinguish user messages from system messages; all user entries are assigned consecutive numbers starting from 0, regardless of their tags. It is the responsibility of the application programmer to maintain the ordering of messages and the assignment of identifiers (manifest constants) for them. Some typical entries are shown below.

SM_RENTRY = Entry is required. SM_MUSTFILL = Must fill field. SM_CKDIGIT = Check digit error. SM_NOHELP = No help text available. US_INSUF = Insufficient funds. RESERVED = US_SUPV = See supervisor.

11.1 Modifying and Adding Messages

The ASCII version of the message file can be modified using a text editor. For example, if the file was modified as follows:

SM_CKDIGIT = Invalid check digit.

the above message would appear in the case of a check digit error, instead of Check digit error. If an application program were to be compiled with the following definitions:

```
#define US_INSUF      0
#define RESERVED      1
#define US_SUPV       2
```

it could issue the calls:

```
sm_quiet_err (sm_msg_get (US_INSUF));

sm_err_reset (sm_msg_get (US_SUPV));
```

If a decision were made later to change the message text, the change could be made by modifying the message file only, without any need to modify and recompile the application code.

If any message is missing from the message file, and a call is made to display the message, only the message number will be shown. Thus, if the file had no entry for SM_RENTRY, and an operator failed to enter data in a field in which an entry was required, the status line would simply display the number corresponding to SM_RENTRY in smerror.h .

User messages may also be placed in separate message files, loaded with calls to msgread, and accessed in the same way as above.

11.2 Embedding Attributes and Key Names in Messages

Several percent escapes provide control over the content and presentation of status messages. They are interpreted by sm_d_msg_line, which is eventually called by everything that puts text on the status line (including field status text). The character following the percent sign must be in upper-case; this is to avoid conflict with the percent escapes used by printf and its variants. Certain percent escapes (%W, for instance; see below) must appear at the beginning of the message, i.e. before anything except perhaps another percent escape.

- If a string of the form %Annnn appears anywhere in the message, the hexadecimal number nnnn is interpreted as a display attribute to be applied to the remainder of the message. The table below gives the numeric values of the logical display attributes you will need to construct embedded attributes. If you want a digit to appear immediately after the attribute change, pad the attribute to 4 digits with leading zeroes; if the following character is not a legal hex digit, leading zeroes are unnecessary.
- If a string of the form %KKEYNAME appears anywhere in the message, KEYNAME is interpreted as a logical key mnemonic, and the whole expression is replaced with the key label string defined for that key in the key translation file. If there is no label, the %K is stripped out and the mnemonic remains. Key mnemonics are defined in smkeys.h ; it is of course the name, not the number, that you want here. The mnemonic must be in upper-case.
- If %N appears anywhere in the message, the latter will be presented in a pop-up window rather than on the status line, and all occurrences of %N will be replaced by newlines.
- If the message begins with a %B, JYACC FORMAKER will beep the terminal (using sm_bel) before issuing the message.
- If the message begins with %W, it will be presented in a pop-up window instead of on the status line. The window will appear near the bottom center of the screen, unless it would obscure the current field by so doing; in that case, it will appear near the top. If the message begins with %MU or %MD, and is passed to one of the error message display functions, JYACC FORMAKER will ignore the default error message acknowledgement flag and process (for %MU) or discard (for %MD) the next character typed.

Note that, if a message containing percent escapes - that is, %A, %B, %K, %N or %W - is displayed before sm_initcrt or after %W is called, the percent escapes will show up in it.

Attribute	Hex value
BLACK	0 BLUE
	1 GREEN
	2 CYAN
	3 RED
	4 MAGENTA
	5 YELLOW
	6 WHITE
	7
B_BLACK	0 B_BLUE
	100 B_GREEN
	200 B_CYAN
	300 B_RED
	400 B_MAGENTA
	500 B_YELLOW
	600 B_WHITE
	700
BLANK	8 REVERSE
	10 UNDERLN
	20 BLINK
	40 HILIGHT
	80 DIM
	1000

If the cursor position display has been turned on (see sm_c_vis), the end of the status line will contain the cursor's current row and column. If the message text would overlap that area of the status line, it will be displayed in a window instead.

Note that the processing of percent escapes in messages is done only when the message is displayed on the status line; they will not be expanded simply by virtue of having been retrieved from the message file. Also, at present, msg2bin does no syntax checking.

NAME

modkey - key translation file editor

SYNOPSIS

modkey [keyfile]

DESCRIPTION

12.1 Introduction

The modkey utility provides a convenient mechanism for specifying how keys on a particular keyboard should operate in the JYACC FORMAKER environment. It provides for defining the function, editing, and cursor control keys used by JYACC FORMAKER, as well as keys that produce foreign or graphics characters. Finally, modkey can store label text corresponding to your keys, for use in prompts and help messages.

The output of modkey is a text file called the key translation file. After being converted into a binary table by the key2bin utility, it is used to translate physical characters generated by the keyboard into logical values used by the JYACC FORMAKER library. By dealing with logical keys, programs can work transparently with a multitude of keyboards.

Refer to the Author's Guide for a table explaining the functions of the cursor control and editing keys. The format of the key translation file generated by modkey is explained in the section of this chapter on key files.

12.1.1 Key Translation

The ASCII character set is comprised of eight-bit characters in the range 0 to 256 (hex FF). It defines characters in the ranges hex 20 to hex 7E and hex A0 to hex FE as data characters, and the rest as control characters. Control characters have mnemonic names; the character hex 1B, for instance, is usually called ESC or escape. See section 8.3 for a list. Note that certain computers, such as PRIME, "flip" the high bit of ASCII characters; on such computers, ESC would be hex 9B and the letter A would be hex C1. In this document, standard ASCII values will be used.

When you press a key, the keyboard generates either a single ASCII data character, or a sequence of characters beginning with an ASCII control code. JYACC FORMAKER converts these characters into logical keys before processing them. Logical keys are numbers between zero and 65535. Logical values between 1 and hex FF represent displayable data; values between hex 100 and hex 1FF are cursor control and editing keys; values greater than hex 1FF are function keys. Zero is never used. For a list of logical values, see Section 8.2.

Data characters received from the keyboard are not translated. Sequences beginning with a control character are translated to a logical value, representing a data character or function key, according to the following algorithm.

When a control character is received, we search the key translation table for a sequence beginning with that character. If there is one, we read additional characters until a match with an entire sequence in the table is found, and return the logical value from the table. If the initial character is in the table but the whole sequence is not, the whole is discarded, on the assumption that it represents a function key that is missing from the table. Finally, if a control character does not begin any sequence in the table, it is returned unchanged; this is useful for machines such as IBM PCs that use control codes for displayable characters. The Programmer's Guide contains a detailed discussion of key translation.

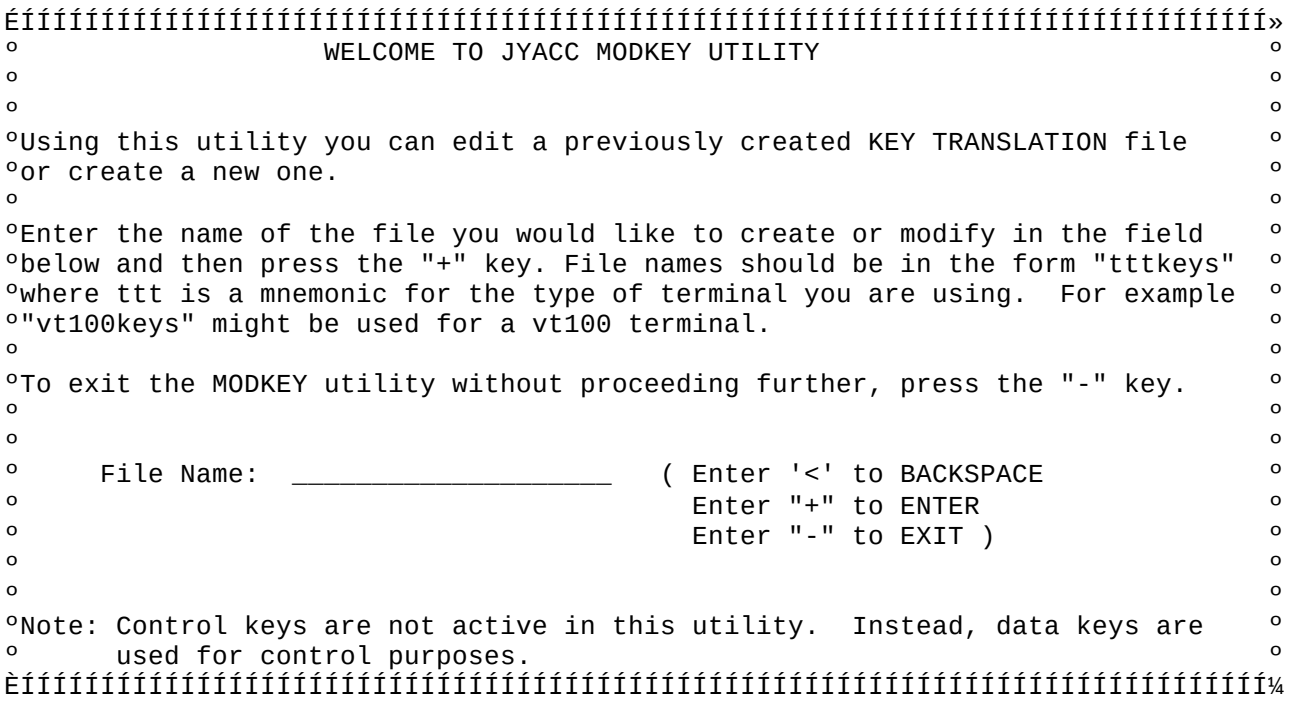

Figure 1: Welcome Screen

12.2 Executing the Utility

You execute modkey by typing its name on the command line, optionally followed by the name of the key file you want to examine or change. If you supply a key file name, the main menu (Figure 2) appears at once. If you do not give a filename, the welcome screen (Figure 1) appears, and you may enter one there.

12.3 Control Keys and Data Keys

Since modkey is used to define the cursor control, editing, and function keys, these keys do not operate in the utility. Instead, displayable data keys are used for these purposes. For example, the TAB key is usually used to move the cursor from one field to the next. But since TAB is one of the keys being defined with this utility, it cannot first be recognized; the data key t is used instead.

Using data keys for control purposes poses no problem since, in this utility, data keys may not begin a control sequence. This will become clearer when the screens in subsequent sections are described. The control functions that are supported in the modkey utility and the keys that are used to provide them are given in the following table:

Control function	Key
TRANSMIT	+ EXIT
	- HELP
	? REDRAW SCREEN
	! BACKSPACE
	< BACKTAB
	b FIELD ERASE
	d ENTER KEYTOP
	k TAB
	t ERASE ALL UNPROTECTED
	z

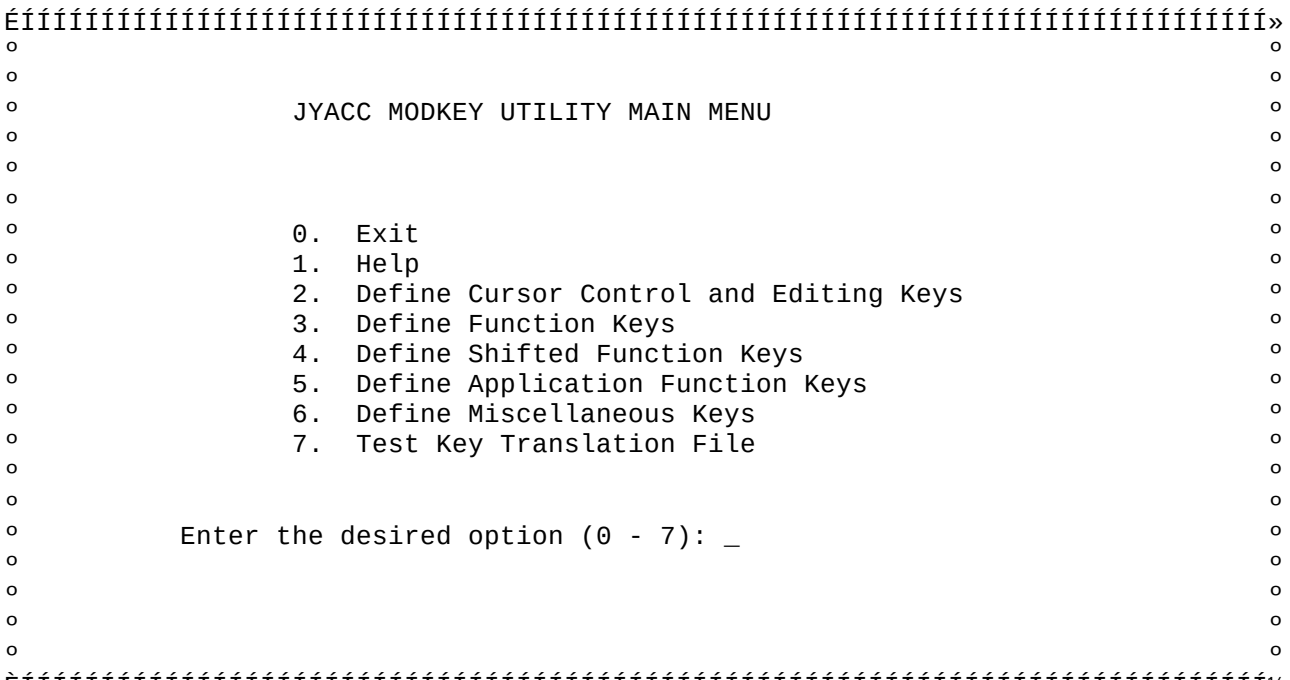

Figure 2: Main Menu

The k key, or ENTER KEYTOP, causes a small window to appear under the cursor in which you may enter the label found on the key in question on your keyboard. This label will be stored in the key translation file; it can be accessed by library functions and in status line messages, and is very useful in help messages telling an operator which key to press. It operates in all the screens below the main menu that are actually used for defining keys.

12.4 Welcome Screen

When you invoke modkey without supplying a key file name, the welcome screen (Figure 1) is displayed. Here you specify the key translation file to be created or modified, by entering it in the field labeled File Name. If you make a mistake, backspace over it using the < key. When finished, complete the screen by pressing the + key.

Key translation file names should begin with a mnemonic for the type of terminal you are using, and end with keys. For example vt100keys might be used for a vt100 terminal. This convention, while not mandatory, helps avoid confusion with video files and with other key translation files; all files distributed by JYACC adhere to it.

If the file already exists, it is read into memory and may be modified; otherwise, you start from scratch. All modifications are made in memory, and file updates are performed only at the conclusion of the program and at your explicit request.

To exit the modkey utility while the welcome screen is displayed, press the "-" key (EXIT).

12.5 Main Menu

The main menu shown in Figure 2 is displayed at entry to the utility, and whenever you return from a lower-level screen. You select an option by typing the corresponding number. For example, to test the key translation file, press "7". If you make an invalid selection, an error message will appear; acknowledge it by pressing the space bar. The functions on the main menu are described in

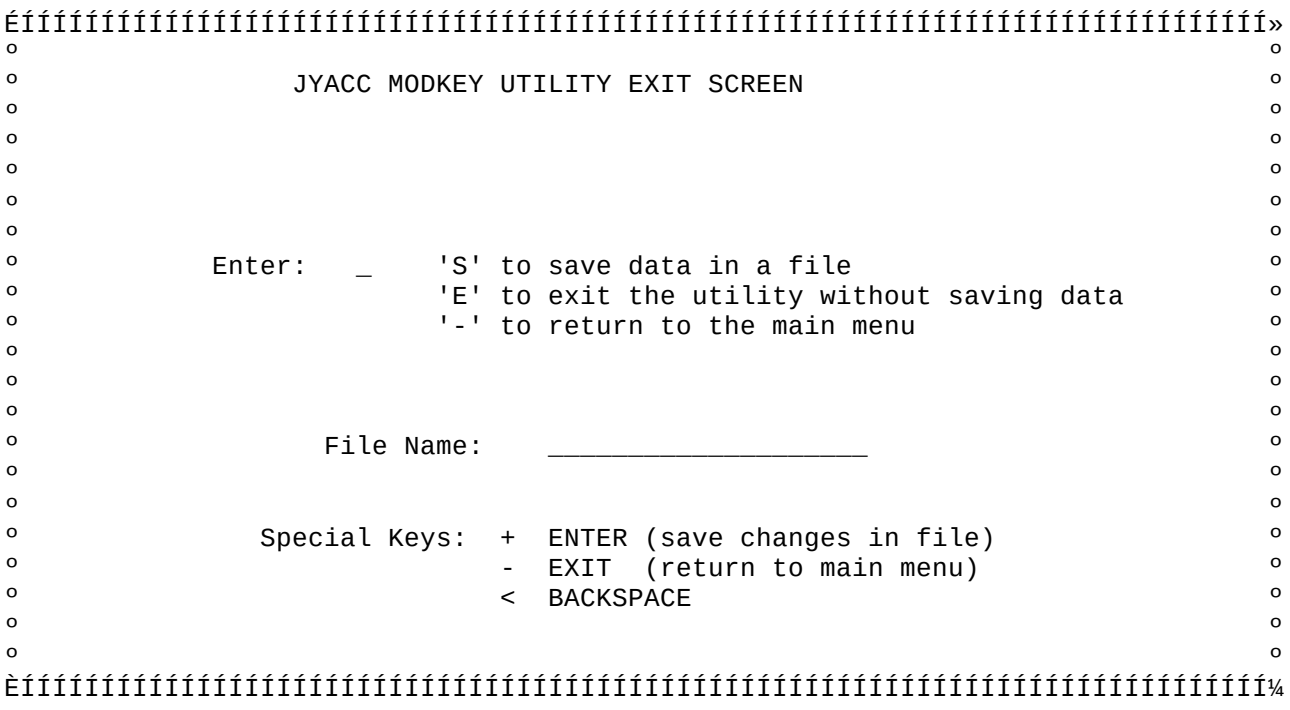

Figure 3: Exit Screen

subsequent sections.

12.6 Exiting the Utility

To exit modkey, press 0 on the main menu. This causes the exit screen (Figure 3) to be invoked. This screen initially contains a single field into which you enter s, e, or -. To save the key translation file on disk, enter "S" or "s". When this is done, the file name entered in the Welcome Screen appears; you may change it if you wish, and press + to write it to disk. To exit the utility without saving the file, enter e. If you press -, the main menu will reappear, and you may make additional changes to the key translation file.

12.7 Help Screen

The help screen may be selected from the main menu by pressing "1"; it appears in Figure 4. In addition to displaying useful information, this screen may be used to test out the kinds of keystroke entry that will be required on subsequent screens in this utility. There are two types of keys: those that generate a single ASCII character, and those that generate a sequence of characters. When a sequence is generated, the first character is always an ASCII control character. To see the characters generated by a particular key, type that key twice while the help screen is displayed. (Different keys generate different numbers of characters; when you press the key twice, the program can sense the pattern.)

When the key is pressed the first time, the characters produced will be shown following CHARACTERS GENERATED. When the key is pressed the second time and recognized, the sequence representing the key will appear following KEY STROKE.

It is sometimes desirable to designate a sequence of keystrokes to serve a particular purpose. For example, on a system with a small number of function keys, one may choose to implement the function keys F1 through F9 with the sequence control-F n where n is a single digit. This sequence of keystrokes can be interpreted by JYACC FORMAKER as a single key.

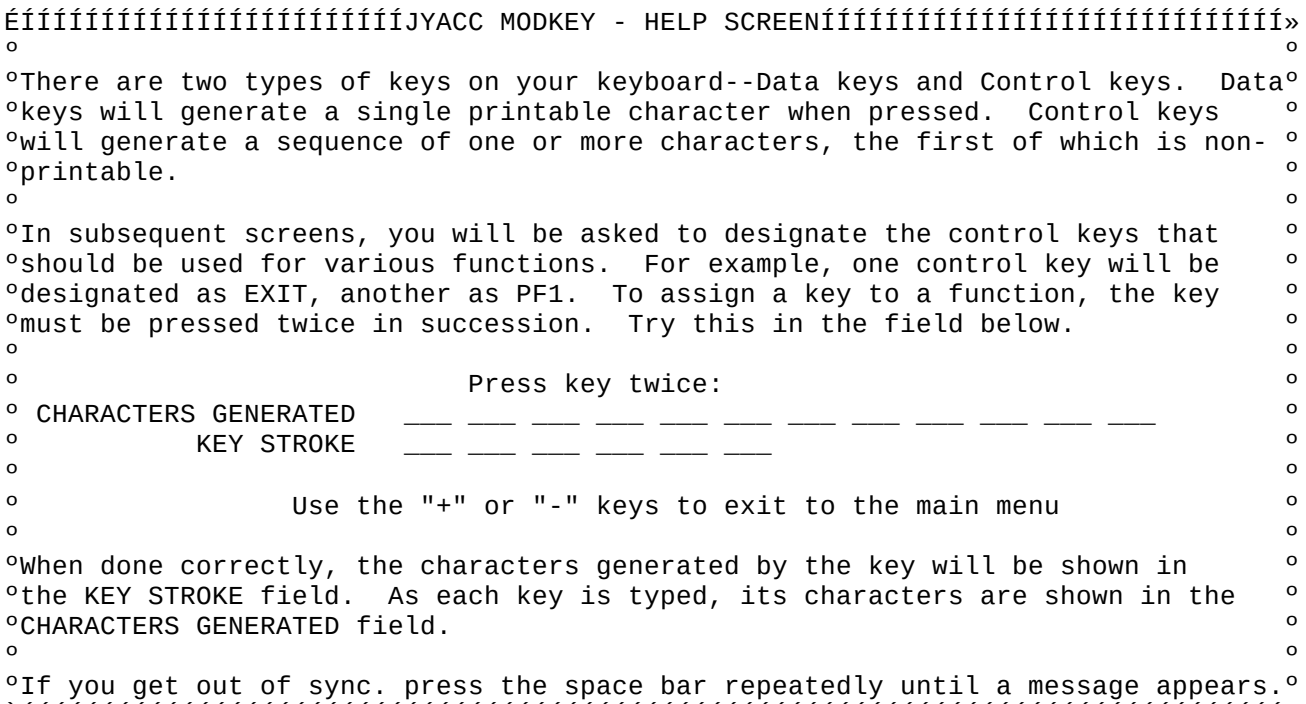

Figure 4: Help Screen

To demonstrate this, type control-F 1 into the help screen, by pressing the F key while holding the CTRL key down, releasing both, and then pressing the 1 key. The sequence ACK 1 will appear following CHARACTERS GENERATED. Repeating the sequence will duplicate the ACK 1 after CHARACTERS GENERATED column, and also display ACK 1 following KEY STROKE.

If a printable ASCII character is pressed as the first key in a sequence, modkey immediately displays it in the KEY STROKE column. If a non-printable character is pressed and then a second, different character is pressed, modkey will assume that a sequence is being tried and will continue displaying these characters in the CHARACTERS GENERATED field. However, if the sequence gets to be longer than six characters without starting to repeat, modkey will display Sequence too long. You must acknowledge this message by pressing the space bar. If you realize you have made a mistake in entering a key or key sequence and do not wish to duplicate it, press any key repeatedly until you see Sequence too long. After acknowledging the message, you can start over.

To exit from the help screen and return to the main menu, press the "-" key (EXIT) as the first character in a sequence.

12.8 Defining Cursor Control and Editing Keys

This function allows the operator to specify the keys that should be used for the various cursor control and editing operations. When 2 is selected from the main menu, the screen shown in Figure 5 appears. This screen has a field for each of the cursor control and editing functions supported by JYACC FORMAKER. Each function has a logical value defined in the file smkeys.h . The purpose of this screen is to allow the operator to specify a sequence of characters for each function key.

12.8.1 Assigning a Key to a Function

To designate a key for a particular cursor control or editing function, position the cursor after that function's name and press the key twice. For example, to designate a key as the EXIT key, press it twice in succession while the cursor is in the EXIT field. When modkey recognizes the second keystroke, the sequence

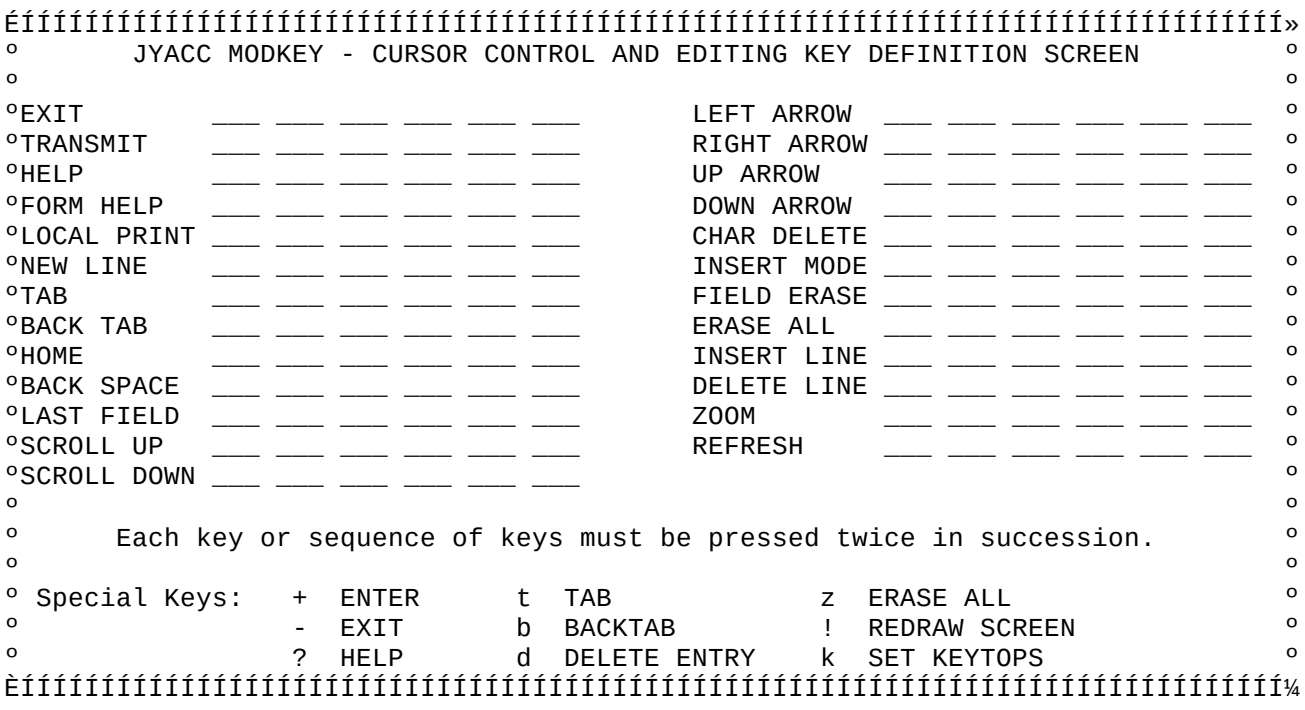

Figure 5: Cursor Key Screen

of characters generated by the key will be displayed, and the cursor will move to the next field.

It is not permissible to define a printable ASCII character as a cursor control or editing key. This means that the sequence of characters generated by the key must start with an ASCII control character. If this is not the case, an error will be displayed. An error will also be displayed if the sequence of characters matches a sequence assigned to another function.

When a field is left empty, its corresponding function will not operate in programs using the Keyboard Translation file being defined. If your program has no use for a particular key (such as GO TO LAST FIELD), you may leave that entry blank on this screen. However, certain keys are required for the proper operation of xform, and should be specified if you are creating a table for use with it. A list of the required keys is given in Section 8.2.

Situations may arise in which you do not press the same key twice in succession. This will be evident because modkey will not display the characters that were generated. To recover, press the space bar repeatedly until the message Sequence too long appears. Then, after acknowledging the message with the space bar, you may enter the correct keystrokes.

To define a key label or keytop for any key on this screen, press k with the cursor at the beginning of the key sequence. A small, borderless window will appear, bearing the word KEYTOP:. In the following field, you should type whatever appears on top of the key on your keyboard, using the < key to rub out mistakes. When done, press + to save the label, or - to discard it.

12.8.2 Assigning a Sequence of Keys to a Function

It is sometimes desirable to designate a sequence of keystrokes to serve a particular purpose. For example, on a keyboard with few function keys, one might implement the function keys PF1 through F9 with the sequences control-F 1 through control-F 9.

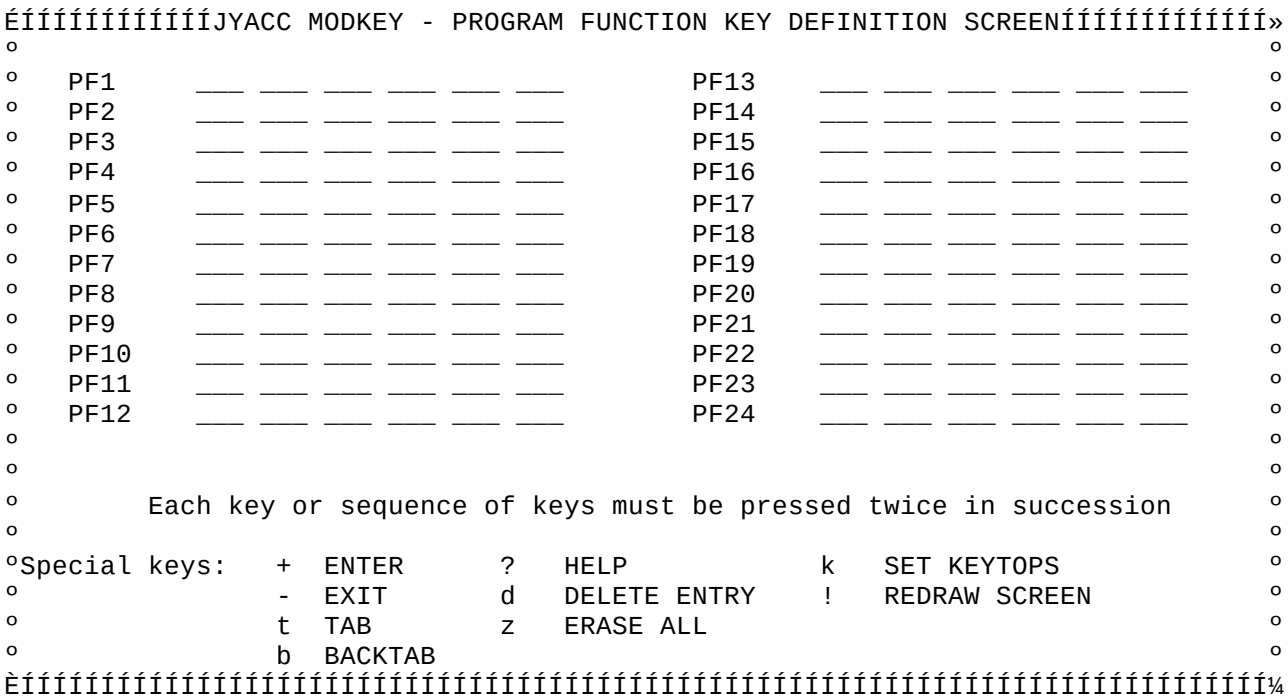

Figure 6: Function Key Screen

One assigns a sequence of keystrokes to a function in much the same way as one assigns individual keys. The sequence is entered once in its entirety and is then repeated. Upon successful completion, the characters generated on behalf of the sequence are displayed.

If you do not press the same key sequence twice, modkey will not display the generated characters. To recover, press the space bar repeatedly until the message Sequence too long appears. At this point, you may enter the correct keystrokes.

12.9 Defining Function Keys

This function allows the operator to specify the keys that should be used as the function keys (PF1 - PF24). When 3 is selected from the main menu, the screen of Figure 6 appears. This function works exactly like its counterpart for defining the cursor control and editing keys described in Section 12.8. You designate a key or key sequence as a function key by pressing it twice, with the cursor in the field to which the sequence applies. For example, to define control-F as the PF2 key, position the cursor to the PF2 field using t and b, and type control-F twice in succession.

To save the changes made in this screen and return to the main menu, press the + key. To return to the main menu without saving changes, use the "-" key.

To define a key label or keytop for any key on this screen, press k with the cursor at the beginning of the key sequence. A small, borderless window will appear, bearing the word KEYTOP:. In the following field, you should type whatever appears on top of the key on your keyboard, using the < key to rub out mistakes. When done, press + to save the label, or - to discard it.

12.10 Defining Shifted Function Keys

This function allows the operator to specify the keys that should be used as the shifted function keys (SPF1 - SPF24). When 4 is selected from the main menu, the screen depicted in Figure 7 appears.

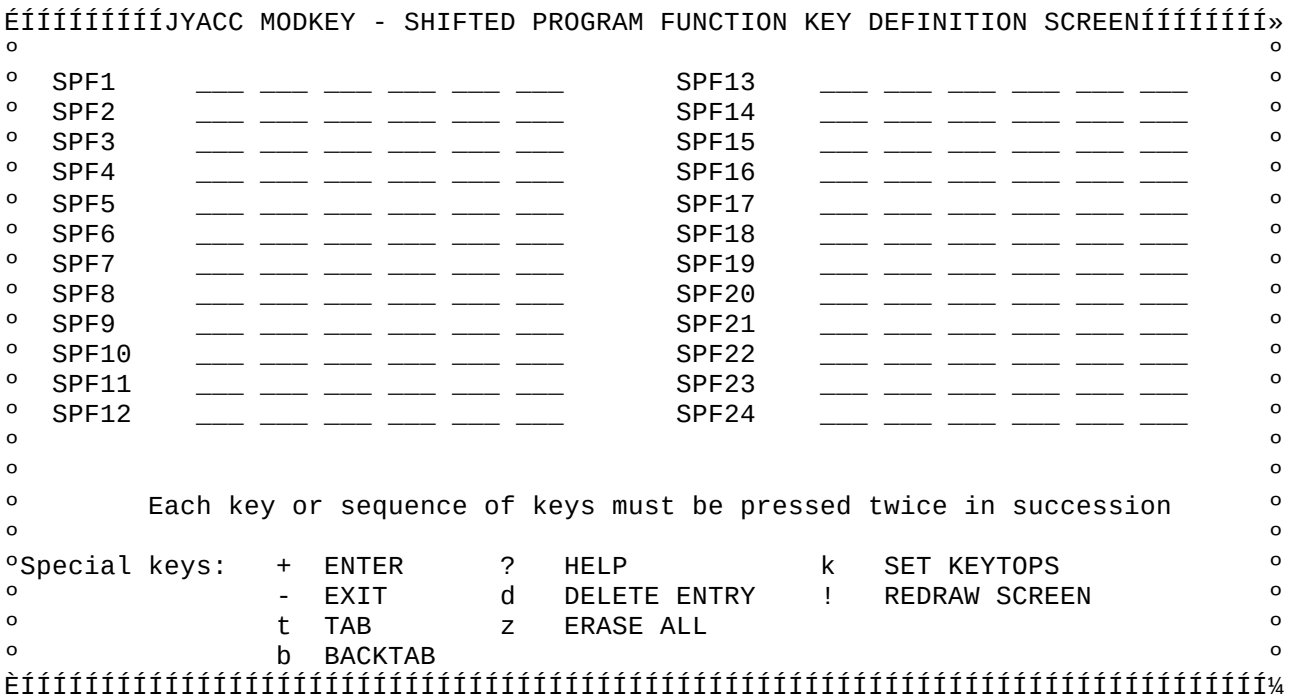

Figure 7: Shifted Function Key Screen

This function works exactly like its counterpart for defining the function keys described in Section 12.9. You designate a key (or key sequence) as a shifted function key by pressing it twice with the cursor in the field to which the sequence applies. For example, to define the sequence of keys control-B 2 as the shifted PF2 key, position the cursor to the SPF2 field, using t and b, and type control-B 2 twice.

To save changes made in this screen and return to the main menu, press the + key as the first character in a sequence. To return to the main menu without saving the changes, use the "-" key.

To define a key label or keytop for any key on this screen, press k with the cursor at the beginning of the key sequence. A small, borderless window will appear, bearing the word KEYTOP:. In the following field, you should type whatever appears on top of the key on your keyboard, using the < key to rub out mistakes. When done, press + to save the label, or - to discard it.

12.11 Defining Application Function Keys

This function allows the operator to specify the keys that should be used as the application function keys (APP1 - APP24). When 5 is selected from the main menu, the screen of Figure 8 appears. This function works exactly like its counterpart for defining the function keys described in Section 12.9.

To define a key label or keytop for any key on this screen, press k with the cursor at the beginning of the key sequence. A small, borderless window will appear, bearing the word KEYTOP:. In the following field, you should type whatever appears on top of the key on your keyboard, using the < key to rub out mistakes. When done, press + to save the label, or - to discard it.

12.12 Defining Miscellaneous Keys

On this screen, you can specify logical keys not present on the other screens, and define alternate control sequences for keys defined elsewhere. When 6 is selected from the main menu, the screen of Figure 9 displayed. This function works in a similar manner to its counterpart for defining the cursor control and

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍJYACC MODKEY - APPLICATION KEY DEFINITION SCREENÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º                                                                               º
º  APP1     ___ ___ ___ ___ ___ ___      APP13    ___ ___ ___ ___ ___ ___       º
º  APP2     ___ ___ ___ ___ ___ ___      APP14    ___ ___ ___ ___ ___ ___       º
º  APP3     ___ ___ ___ ___ ___ ___      APP15    ___ ___ ___ ___ ___ ___       º
º  APP4     ___ ___ ___ ___ ___ ___      APP16    ___ ___ ___ ___ ___ ___       º
º  APP5     ___ ___ ___ ___ ___ ___      APP17    ___ ___ ___ ___ ___ ___       º
º  APP6     ___ ___ ___ ___ ___ ___      APP18    ___ ___ ___ ___ ___ ___       º
º  APP7     ___ ___ ___ ___ ___ ___      APP19    ___ ___ ___ ___ ___ ___       º
º  APP8     ___ ___ ___ ___ ___ ___      APP20    ___ ___ ___ ___ ___ ___       º
º  APP9     ___ ___ ___ ___ ___ ___      APP21    ___ ___ ___ ___ ___ ___       º
º  APP10    ___ ___ ___ ___ ___ ___      APP22    ___ ___ ___ ___ ___ ___       º
º  APP11    ___ ___ ___ ___ ___ ___      APP23    ___ ___ ___ ___ ___ ___       º
º  APP12    ___ ___ ___ ___ ___ ___      APP24    ___ ___ ___ ___ ___ ___       º
º                                                                               º
º                                                                               º
º        Each key or sequence of keys must be pressed twice in succession       º
º                                                                               º
ºSpecial keys:   +  ENTER      ?   HELP            k   SET KEYTOPS              º
º                -  EXIT       d   DELETE ENTRY    !   REDRAW SCREEN            º
º                t  TAB        z   ERASE ALL                                    º
º                b  BACKTAB                                                     º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

Figure 8: Application Key Screen

```
ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍJYACC MODKEY - MISCELLANEOUS KEY DEFINITION SCREENÍÍÍÍÍÍÍÍÍÍÍÍÍ»
º                                                                                º
º      KEY STROKE       LOGICAL VALUE        KEY STROKE       LOGICAL VALUE      º
º___ ___ ___ ___ ___ ___   ___________     ___ ___ ___ ___ ___ ___   ___________º
º___ ___ ___ ___ ___ ___   ___________     ___ ___ ___ ___ ___ ___   ___________º
º___ ___ ___ ___ ___ ___   ___________     ___ ___ ___ ___ ___ ___   ___________º
º___ ___ ___ ___ ___ ___   ___________     ___ ___ ___ ___ ___ ___   ___________º
º___ ___ ___ ___ ___ ___   ___________     ___ ___ ___ ___ ___ ___   ___________º
º___ ___ ___ ___ ___ ___   ___________     ___ ___ ___ ___ ___ ___   ___________º
º___ ___ ___ ___ ___ ___   ___________     ___ ___ ___ ___ ___ ___   ___________º
º___ ___ ___ ___ ___ ___   ___________     ___ ___ ___ ___ ___ ___   ___________º
º___ ___ ___ ___ ___ ___   ___________     ___ ___ ___ ___ ___ ___   ___________º
º___ ___ ___ ___ ___ ___   ___________     ___ ___ ___ ___ ___ ___   ___________º
º___ ___ ___ ___ ___ ___   ___________     ___ ___ ___ ___ ___ ___   ___________º
º___ ___ ___ ___ ___ ___   ___________     ___ ___ ___ ___ ___ ___   ___________º
º                                                                                º
º                     LOGICAL VALUE DISPLAY MODE IS:________                     º
º                                                                                º
ºSpecial keys:  +  ENTER       ?  HELP            v  TAB TO VALUE FIELD          º
º               -  EXIT        !  REDRAW SCREEN   k  SET KEYTOPS                 º
º               t  TAB         d  DELETE ENTRY    <  BACKSPACE IN VALUE FIELD    º
º               b  BACKTAB     z  ERASE ALL       c  CHANGE MODE                 º
ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ¼
```

Figure 9: Miscellaneous Keys Screen

editing keys described in Section 12.8. However, on this screen you must define the logical values as well as the sequences that produce them. (On all other screens, the logical value was implicitly determined by the field with which the sequence was associated.)

The miscellaneous key definition screen has two columns for each key being defined, labeled KEY STROKE and LOGICAL VALUE. You enter a key or key sequence into these fields twice in succession, and modkey displays the generated

characters; then the cursor moves to the LOGICAL VALUE column for that key. Here, you must enter the logical value to be returned when JYACC FORMAKER recognizes the sequence of characters you have just entered. You may get to the logical value field directly by pressing v in the corresponding KEY STROKE field.

12.12.1 Entering the Logical Value

Logical values are numbers, so you will be entering printable ASCII data into this field. This is unlike most other fields, where data characters are not allowed or are given special meaning (such as "b" representing BACKTAB). When entering logical values, three keys are allowed in addition to the data keys necessary to enter the value:

- The + key (TRANSMIT) signifies that the logical value just typed is correct and should be used. When it is pressed, modkey will first check the logical value for errors. If no errors are detected, the cursor will tab to the next field; otherwise, an error message will appear.
- The - key (EXIT) means that the logical value just typed is incorrect and should be ignored. The cursor will go to the next field and the logical value will be reset to what existed before the field was entered. If the logical value field was previously empty, it will be set to zero.
- The < key (BACKSPACE) backs up the cursor one position at a time, so that corrections to the logical value can be made. It erases previously entered data as it moves.

12.12.2 Logical Value Display and Entry Modes

Logical values are displayed, and may be entered, in any of four modes. The current mode is displayed on the screen following the label LOGICAL VALUE DISPLAY MODE. It may be changed by typing c as the first character of a sequence while the cursor is in any of the KEY STROKE fields on the screen. When the miscellaneous keys screen is first invoked, the mode is hexadecimal. It cycles through all four modes when you press the c key. The four modes are:

decimal	In decimal mode, you enter logical values as decimal numbers. If a non-digit is entered or the logical value is zero, an error will be displayed.
octal	In octal mode, you enter logical values as octal numbers (base 8). If a non-octal digit is entered or the logical value is zero, an error will be displayed.
hexadecimal	In hexadecimal mode, you enter logical values as hexadecimal (base 16) numbers. If a non-hex digit is entered or the logical value is zero, an error will be displayed. The error must be acknowledged by pressing the space bar.
mnemonic	In mnemonic mode, you enter the mnemonic associated with any of the logical values stored in the file smkeys.h . For example, if EXIT is entered into the Logical Value field, the logical value of the EXIT key, hex 103, will be used. If an incorrect mnemonic is entered, an error will be displayed which must be acknowledged by pressing the space bar. For a list of valid mnemonics, press the "?" key while the cursor is in a logical value field.

Entering the logical value as a mnemonic is preferable, as you are less likely to mistake the value you want. Using the numeric modes, it is possible to define logical key values other than those present in smkeys.h , but this should be done cautiously. You should avoid the range 100 hex through 1FF hex, which is reserved for future use by JYACC. Also, for portability's sake, the values

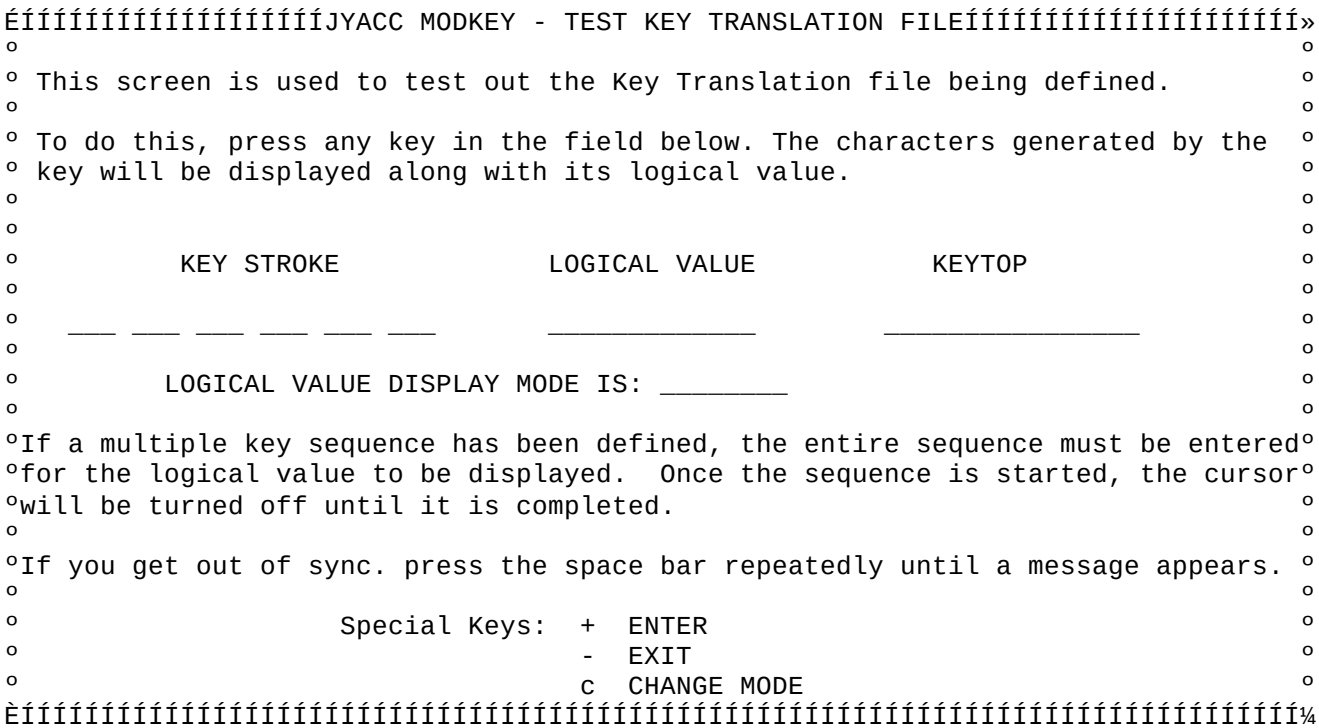

Figure 10: Test Screen

should be small enough to fit in a two-byte integer, i.e. less than 65536 (10000 hex).

To define a key label or keytop for any key on this screen, press k with the cursor at the beginning of the key sequence. A small, borderless window will appear, bearing the word KEYTOP:. In the following field, you should type whatever appears on top of the key on your keyboard, using the < key to rub out mistakes. When done, press + to save the label, or - to discard it.

12.12.3 Returning to the Main Menu

To save changes made in this screen and return to the main menu, press the + key as the first character in a sequence while the cursor is in a KEY STROKE field. To discard the changes and return to the main menu, use the - key.

12.13 Test Keyboard Translation File

This function allows you to test out your new key translation file. When 7 is selected from the main menu, the screen of figure 10 is displayed. This screen has two fields labeled KEY STROKE and LOGICAL VALUE. You enter a keystroke (or sequence of keystrokes) that has been defined in another screen, and modkey will display the logical value of that key. The key or keys need only be pressed once, since the table is being tested for how it will behave when used in a real application. If a key sequence forms only part of a previously specified sequence, modkey will wait for another key until a sequence is matched, or until it determines that no match is possible. In the latter case, the message Key not defined will appear.

The logical value can be displayed in any of the four modes (decimal, octal, hexadecimal, or mnemonic). To change modes, press c as the first character in a sequence. To exit the screen and return to the main menu, use -. Help text can be obtained by pressing "?".

ERROR CONDITIONS

Invalid entry. Cause: You have typed a key that is not on the menu. Corrective action: Check the instructions on the screen and try again.

Key sequence is too long. Cause: You have typed more than six keys wihout repeating any. Corrective action: Key sequences for translation may be at most six characters long. Choose a shorter sequence.

Invalid first character. Cause: A multi-key sequence must begin with a control character. Corrective action: Begin again, using a control character.

Invalid mnemonic - press space for list Cause: In the miscellaneous keys screen, you have typed a character string for logical value that is not a logical key mnemonic. Corrective action: Peruse the list, then correct the input.

Invalid number - enter <decimal>, 0<octal> or 0x<hex> Cause: In the miscellaneous keys screen, you have typed a malformed numeric key code. Corrective action: Correct the number, or use a mnemonic.

Cannot create output file. Cause: An output file could not be created, due to lack of permission or perhaps disk space. Corrective action: Correct the file system problem and retry the operation.

Key sequence does not repeat. Cause: You have typed a key sequence that failed to repeat a string of six characters or less. Corrective action: Retry the sequence, or use a shorter one.

Cannot accept NUL as a key. Cause: The ASCII NUL character (binary 0) cannot be used in a key translation sequence, because it is used internally to mark the end of a sequence. Corrective action: Use another key.

Key previously defined as %s Key conflicts with %s Cause: You have typed a key sequence that has already been assigned to another key, or that is a substring of a previously assigned sequence. Corrective action: Use a different key or sequence, or reassign the other.

NAME

msg2bin - convert message files to binary

SYNOPSIS

```
msg2bin [-pv] [-eextension] [-ooutfile]
     messages [messages ...]
```

DESCRIPTION

The msg2bin utility converts ASCII message files to a binary format for use by JYACC FORMAKER library routines. The command options are interpreted as follows:

- -e Give the output files the extension that follows the option letter, rather than the default bin.
- -o Place all the output in a single file, whose name follows the option letter.
- -p Place each output file in the same directory as the corresponding input file.
- -v Print the name of each message file as it is processed.

The input to this utility files are text files containing named messages, either distributed by JYACC for use with the JYACC FORMAKER library or defined by application programmers. For information about the format of ASCII message files, see the section on message files in this chapter.

The message file and msg2bin utility provide three different services to application designers. First, the error messages displayed by JYACC FORMAKER library functions may be translated from English to another language, made more verbose, or altered to suit the taste of the application designer. Second, error messages for use by application routines may be collected in a message file and retrieved with the msg_get library function; this provides a centralized location for application messages and saves space. Finally, the standard library messages (and user messages) may be made memory-resident, to simplify and speed up the initialization procedure (at some added cost in memory). The bin2c utility converts the output of this utility to a source file suitable for inclusion in the application program.

ERROR CONDITIONS

File '%s' not found. Cause: An input file was missing or unreadable. Corrective action: Check the spelling, presence, and permissions of the file in question.

Unable to allocate memory. Cause: The utility could not allocate enough memory for its needs. Corrective action: None.

Bad tag in line: %s Cause: The input file contained a system message tag unknown to the utility. Corrective action: Refer to smerror.h for a list of tags, and correct the input.

Missing '=' in line: %s Cause: The line in the message had no equal sign following the tag. Corrective action: Correct the input and re-run the utility.

NAME

Setup file - JYACC FORMAKER configuration variables

DESCRIPTION

JYACC FORMAKER supports a number of configuration or setup variables, which provide a convenient way for you to control many operating parameters of the JYACC FORMAKER run-time system and utilities. They include all the environment variables supported in Release 3. The library function smsetup, which is automatically called from initcrt, reads in binary setup files and sets up the run-time environment to correspond.

You can use configuration variables by creating a text file of name-value pairs as described in this section, and then running the var2bin utility to convert it to a binary format.

14.1 The Two Setup Files

There are two files in which you may place setup variables. The first is named by the system environment variable SMVARS. If your operating system does not support an environment, this file will be in a hard-coded location; SMVARS itself may not be put in a setup file. The second file is named by the SMSETUP configuration variable, which may be defined in the SMVARS file or in the system environment.

Any setup variable may occur in either file. If a variable occurs in both, the one in SMSETUP takes precedence. Certain variables may also be specified in the system environment, which takes precedence over any values found in the files; they are noted in the table of Section 14.3. It is possible to specify all the variables necessary to run JYACC FORMAKER in the environment, without constructing a setup file.

Typically, the SMVARS file will contain installation-wide parameters, while the SMSETUP file will contain parameters belonging to an individual or project.

14.2 Input File Line Format

Each line of the input file has the form

```
name = value
```

where name is one of the keywords listed below, the equal sign is required, and value depends on the name. If a line gets too long, it may be continued onto the next by placing a backslash \ at the end. Lines beginning with a pound sign # are treated as comments, i.e. ignored.

Certain variables, notably the JYACC FORMAKER hardware configuration files, have values that depend on the type of terminal you are using. For those variables, there may be many entries in the input file, of the form

```
name = (term1:term2:...:termN)value
```

This signifies that name has value for terminals of type term1, term2, etc. It is not necessary to give terminal names if you are only interested in one file. You may also provide, along with a number of terminal-qualified entries, one entry that is not terminal-qualified; this will serve as the default. It must come last. Variables that are terminal-dependent are noted below.

Certain variables, particularly those that provide parameters for library functions, have keywords to the right of the equal sign. When these keywords are all distinct, they may be separated by blanks, commas, or semicolons, just as you please. But when a certain keyword may appear more than once, so that parameter position is important, then blanks or commas separate the list of

keywords constituting one parameter, while semicolons separate parameters. The semicolon has higher precedence than blank or comma.

14.3 Setup Variables

Broadly speaking, setup variables fall into three classes: those that specify other configuration files; those that are essentially parameters to library routines; and those that specify default file extensions.

Three variables are required: SMMSGS, SMVIDEO, and SMKEY. They specify, respectively, the error message, video configuration, and keyboard translation files that the JYACC FORMAKER run-time system requires in order to function. In the following list, an explanation and example is given for each variable.

14.3.1 Configuration File Setups

SMKEY
: Pathname of the binary file containing a key translation table for your terminal, used by the JYACC FORMAKER run-time system. Refer also to the key2bin and modkey utilities, and the library functions keyinit and getkey.
This variable is terminal-dependent, and may be overridden by the system environment. It may not be omitted.
SMKEY=(vt100:x100)/usr/jyacc/config/vt100keys.bin

SMLPRINT
: Operating system command used to print the file generated by the local print key (LP). It must contain the string %s at the place where the filename should go.
This variable may be overridden by the system environment. It is optional.
SMLPRINT = print %s

SMMSGS
: Pathname of the binary file containing error messages and other printable strings used by the JYACC FORMAKER run-time system and utilities. Refer also to the msg2bin utility and the library functions msg_read and msg_get.
This variable is terminal-dependent, and may be overridden by the system environment. It may not be omitted.
SMMSGS =/usr/jyacc/config/msgfile.bin

SMPATH
: List of directories in which the JYACC FORMAKER run-time system should search for screens and JPL procedures. Place a vertical bar | between directory paths. Refer to the library procedure r_window.
This variable is terminal-dependent, and may be overridden by the system environment. It is optional.
SMPATH=/usr/app/forms|/usr/me/testforms

SMSETUP
: Gives the pathname of an additional binary file of setup variables.
This variable is terminal-dependent, and may be overridden by the system environment. It is optional.
SMSETUP = mysetup.bin

SMVIDEO
: Pathname of the binary file containing video control sequences and parameters used by the JYACC FORMAKER run-time system. Refer also to the vid2bin utility, the Video section of this chapter, and the library function vinit.
This variable is terminal-dependent, and may be overridden by the system environment. It may not be omitted.
SMVIDEO=(vt100:x100)/usr/jyacc/config/vt100vid.bin

14.3.2 Setups for Library Routines

Many of the variables in this class have display attributes as parameters. Here is a table of display attribute keywords:

RED	BLUE	HILIGHT	BLINK
YELLOW	GREEN	UNDERLN	DIM
MAGENTA	CYAN	BLANK	
BLACK	WHITE	REVERSE	

For a single display attribute, you may select from this table one color and any number of other attributes. If a variable has more than one display attribute parameter, separate the parameters with semicolons, but separate the ored attributes for each parameter with blanks or commas. See SMCHEMSGATT, below, for an example.

The explanations of keywords in this section are terse; full details are available on the page in the Programmer's Guide dedicated to the function in question. Mnemonics are the same in both places, except that prefixes may have been deleted for the setup keywords. All these variables are optional, and most cannot be overridden in the system environment.

SMCHEMSGATT — Supplies two display attributes for error messages; see ch_emsgatt. Two parameters, each consisting of one color and any number of other attributes, from the table above.
SMCHEMSGATT = RED; RED, REVERSE

SMCHQMSGATT — Supplies a default display attribute for query messages; see ch_qmsgatt. One parameter consisting of one color and any number of other attributes, from the table above.
SMCHQMSGATT = CYAN, HILIGHT

SMCHSTEXTATT — Supplies a default display attribute for field status text; see ch_stextatt. A single display attribute.
SMCHSTEXTATT=WHITE REVERSE

SMCHUMSGATT — Supplies a border style and three default display attributes for certain JYACC FORMAKER windows. The border style, first, is a number between 1 and 9; the next three are display attributes. See ch_umsgatt.
SMCHUMSGATT = 2; BLUE; BLUE REVERSE; YELLOW

SMDICNAME — Gives the pathname of the application's data dictionary. See the library function dicname. May be overridden in the system environment.
SMDICNAME=/usr/app/dictionary.dat

SMDWOPTIONS — Turns delayed write on or off; passed to the library function dw_options, q.v. Value is either ON or OFF.
SMDWOPTIONS=OFF

SMEROPTIONS — Error message acknowledgement options, as documented at er_options. First comes an acknowledgement character, which you may put in single quotes or as an ASCII mnemonic. Next is the discard keyboard input flag, either DISCARD or USE_KEY. Finally comes the reminder window flag, either YES_WIND or NO_WIND.
SMEROPTIONS=' '; DISCARD; YES_WIND

SMFCASE — Controls the case-sensitivity of filename comparisons when the run-time system searches for files named in JYACC FORMAKER control strings. The keyword INSENS means case will be ignored, and SENS means the search is case-sensitive. The default is SENS. See fcase.
SMFCASE=INSENS

SMFLIBS — A list of pathnames of screen libraries that are to remain open while JYACC FORMAKER is active. The names are separated by blanks, commas, or semicolons. See r_window and l_open.
SMFLIBS=/usr/app/genlib /usr/me/mylib

SMINDSET — Scrolling and shifting indicator options, as for the library function sm_ind_set. The first parameter tells which indicators should be displayed: NONE, SHIFT, SCROLL, or BOTH. The second controls the style of scrolling indicators: FLDENTRY, FLDLEFT, FLDRIGHT, or FLDCENTER.
SMINDSET = BOTH FLDCENTER

SMINICTRL — May occur many times. Each occurrence binds a function key to a control string, which the JYACC FORMAKER run-time system will use in the absence of a control string in the screen. To disable a JYACC-supplied default function key, bind it to a caret function that does nothing.

```
SMINICTRL= PF2 = ^toggle_mode
SMINICTRL = PF3 = &popwin(3,28)
SMINICTRL = XMIT = ^commit all
```

SMININAMES — Supplies a list of local data block initialization file names for use by ldb_init, like the library function ininames. The names are separated by commas, blanks, or semicolons; there may be up to ten of them.

```
SMININAMES=tables.ini,zips.ini,config.ini
```

SMMPOPTIONS — Supplies parameters for the library function mp_options, q.v. These parameters control the behavior of the cursor within menu_proc. Here they are:

Arrow key wrapping: WRAP or NOWRAP
Up- and down-arrow control: UD_TAB, UD_FREE, UD_RESTRICT, UD_COLM, UD_SWATH, UD_NEXTLINE, UD_NEXTFLD
Left- and right-arrow control: LR_TAB, LR_FREE, LR_RESTRICT, LR_COLM, LR_SWATH, LR_NEXTLINE, LR_NEXTFLD

```
SMMPOPTIONS = WRAP; UD_RESRICT,\
UD_NXTLINE; LR_RESTRICT, LR_NXTFLD;
```

SMMPSTRING — Controls the menu item matching actions of menu_proc, by supplying parameters for mp_string; refer to those functions. The single parameter is either STRING or NOSTRING.

```
SMMPSTRING = NOSTRING
```

SMOKOPTIONS — The right-hand side has six parameters, corresponding to those of the library function ok_options, (q.v.). They are, in turn:

Cursor style: BLOCK or NOBLOCK
Arrow key wrapping: WRAP or NOWRAP
Field reset flag: RESET or NORESET
Up- and down-arrow control: UD_TAB, UD_FREE, UD_RESTRICT, UD_COLM, UD_SWATH, UD_NEXTLINE, UD_NEXTFLD
Left- and right-arrow control: LR_TAB, LR_FREE, LR_RESTRICT, LR_COLM, LR_SWATH, LR_NEXTLINE, LR_NEXTFLD
Always-validate flag: VALID, NOVALID
Beep on overstriking last character of no-autotab field: ENDCHAR

```
SMOKOPTIONS = BLOCK; WRAP; RESET;\
UD_RESRICT, UD_NXTLINE; LR_RESTRICT,\
LR_NXTFLD; VALID; ENDCHAR
```

SMZMOPTIONS — Zoom key options, as documented nunder the library function zm_options. The first parameter controls the first step of zooming, and may be either NOSHIFT, SCREEN, ELEMENT, or ITEM. The second controls the subsequent step, and may be NOSCROLL, SCROLL, PARALLEL, or 1STEP.

```
SMZMOPTIONS = ITEM PARALLEL
```

14.3.3 Setups for Default File Extensions

These variables control the default file extensions used by utilities, which are listed below.

SMFEXTENSION — Screen file extension, used by the JYACC FORMAKER run-time system and various utilities. The default in Release 4.0 is none; the default in Release 3 was jam. May be overridden in the system environment. See fextension.

```
SMFEXTENSION=f
```

SMUSEEXT — This variable controls the file extension rules described in Section 2.2. The first parameter is the extension separator character, which may be a quoted character,

number, or ASCII mnemonic. The second controls whether JYACC FORMAKER attemptes to recognize and replace extensions, and is either RECOGNIZE or IGNORE. The last determines whether extensions are placed before or after the filename, and is either FRONT or BACK.
SMUSEEXT = '-'; RECOGNIZE; FRONT

NAME

term2vid - create a video file from a terminfo or termcap entry.

SYNOPSIS

term2vid [-f] terminal-mnemonic

DESCRIPTION

Term2vid creates a rudimentary screen manager video file from information in the terminfo or termcap database. Terminal-mnemonic is the name of the terminal type, the value of the system environment variable TERM, which is used by the C library function tgetent to access that database.

The output file will be named after the mnemonic. The -f option tells the utility it's OK to overwrite an existing output file.

ERROR CONDITIONS

No cursor position (cm, cup) for %s Cause: An absolute cursor positioning sequence is required for JYACC FORMAKER to work, and the termcap or terminfo entry you are using does not contain one. Corrective action: Construct the video file by hand, or update the entry and retry.

Cannot find entry for %s Cause: The terminal mnemonic you have given is not in the termcap or terminfo database. Corrective action: Check the spelling of the mnemonic.

File %s already exists; use '-f' to overwrite. Cause: You have specified an existing output file. Corrective action: Use the -f option to overwrite the file, or use a different name.

)

NAME

txt2form - Converts text files to JYACC FORMAKER screens

SYNOPSIS

txt2form [-fv] textfile screen [height width]

DESCRIPTION

This program converts textfile to a read-only JYACC FORMAKER screen, named screen. It creates display data sections from the input text. It preserves blank space, and expands tabs to eight-character stops; other control characters are just copied to the output. Text that extends beyond the designated maximum output height or width is discarded; if the last two parameters are missing, a 23-line by 80-column screen is assumed.

Txt2form puts no borders, fields, or display attributes in the output screen. However, underscores (or other, user-designated field definition characters) in the input are copied to the screen file; if you subsequently bring the screen up in xform and compile it, those characters will be converted to fields.

The -f option directs the utility to overwrite an existing output file. The -v prints the name of each screen as it is processed.

ERROR CONDITIONS

Warning: lines greater than %d will be truncated Warning: columns greater than %d will be truncated Cause: Your input text file has data that reaches beyond the limits you have given (default 23 lines by 80 columns) for the screen. Corrective action: Shrink the input, or enlarge the screen.

Unable to create output file. Cause: An output file could not be created, due to lack of permission or perhaps disk space. Corrective action: Correct the file system problem and retry the operation.

NAME

var2bin - convert files of setup variables to binary

SYNOPSIS

var2bin [-pv] [-eext] setupfile [setupfile ...]

DESCRIPTION

This utility converts files of setup variables to binary format for use by the run-time system. See pages 5-38ff for a full description of how to prepare the ASCII file.

The -v prints the name of each screen as it is processed. The -p option causes the output file to be created in the same directory as the input file, and the -e option supplies a file extension different from the default of bin.

ERROR CONDITIONS

Error opening %s. Cause: An input file was missing or unreadable. Corrective action: Check the spelling, presence, and permissions of the file in question.

Missing '='. Cause: The input line indicated did not contain an equal sign after the setup variable name. Corrective action: Insert the equal sign and run var2bin again.

%s is an invalid name. Cause: The indicated line did not begin with a setup variable name. Corrective action: Refer to the Configuration Guide for a list of variable names, correct the input, and re-run the utility.

%s may not be qualified by terminal type. Cause: You have attached a terminal type list to a variable which does not support one. Corrective action: Remove the list. You can achieve the desired effect by creating different setup files, and attaching a terminal list to the SMSETUP variable.

Unable to set given values. %s conflicts with a previous parameter. %s is an invalid parameter. Cause: A keyword in the input is misspelled or misplaced, or conflicts with an earlier keyword. Corrective action: Check the keywords listed in the manual, correct the input, and run the utility again.

Error reading smvars or setup file. Cause: The utility incurred an I/O error while processing the file named in the message. Corrective action: Retry the operation.

Unable to allocate memory. Cause: The utility could not allocate enough memory for its needs. Corrective action: None.

At least one file name is required. Cause: You have failed to give an input file name. Corrective action: Retype the command, supplying the file name.

Entry size %d is too large. String size %d is too large. Cause: The indicated right-hand side is too long. Corrective action: Reduce the size of the entry.

NAME

vid2bin - convert video files to binary

SYNOPSIS

vid2bin [-vp] [-eext] terminal-mnemonic

DESCRIPTION

The vid2bin utility converts an ASCII video file to binary•format for use by applications with the JYACC FORMAKER library •routines. The video files themselves must be created with a text•editor, according to the rules listed in the video manual•(q.v.).

Terminal-mnemonic is an abbreviation for the name of the terminal •for which the ASCII video file has been constructed. That file,•whose name is conventionally the mnemonic followed by the suffix•vid, is the input to vid2bin. (When opening its input,•vid2bin first tries them mnemonic, then the mnemonic followed•by vid.)

To make a video file memory-resident, run the bin2c utility•on the output of vid2bin, compile the resulting program•source file, link it with your application, and call the library•routine vinit.

The -v option prints the name of each screen as it is processed. -p creates each output file in the same directory as the corresponding input file. The use of the -p option is not recommended.

For information about the format of the ASCII video file, refer•to the video manual and the Programmer's Guide.

ERROR CONDITIONS

Neither %s nor %s exists. Cause: An input file was missing or unreadable. Corrective action: Check the spelling, presence, and permissions of the file in question.

A cursor positioning sequence is required. An erase display sequence is required. Cause: These two entries are required in all video files. Corrective action: Determine what your terminal uses to perform these two operations, and enter them in the video file; then run the utility again.

Unable to allocate memory. Cause: The utility could not allocate enough memory for its needs. Corrective action: None.

Error writing to file '%s'. Cause: The utility incurred an I/O error while processing the file named in the message. Corrective action: Retry the operation.

Invalid entry: '%s'. Entry missing '=': '%s'. Cause: The input line in the message does not begin with a video keyword and an equal sign. Corrective action: Correct the input and re-run the utility. You may have forgotten to place a backslash at the end of a line that continutes onto the next one.

Invalid attribute list : '%s'. Invalid color specification : '%s'. Invalid graphics character specification (%s):'%s'. Invalid border information (%s):'%s'. Invalid graphics type : '%s'. Invalid label parameter :

'%s'.%s Invalid cursor flags specification : '%s'. Cause: You have misspelled or misplaced keywords in the input line in the message. Corrective action: Correct the input, referring to the Configuration Guide, and run vid2bin again.

NAME

Video file - video configuration manual

DESCRIPTION

19.1 Introduction to Video Configuration

JYACC FORMAKER is designed to run on many displays with widely differing characteristics. These characteristics greatly affect JYACC FORMAKER's display of screens and messages. For example, some displays are 80 columns wide, while others have 132; again, the control sequences used to position the cursor and highlight data on the display are hardly the same for any two models. JYACC FORMAKER obtains display characteristics from a video file.

19.1.1 How to Use this Manual

This manual has two purposes. The first is to explain the entries in the JYACC FORMAKER video file, and the concepts used in interpreting them. Although you may well never need to modify or construct a video file, you may wish to know what it does. The second purpose is to provide instructions for modifying existing video files, or constructing new ones, to handle new terminal characteristics.

Creating a video file is not trivial; neither is it a major effort. The easiest way is to use one of the many supplied with JYACC FORMAKER. There are fifty or so as of this writing; you may find a list in an appendix to Chapter One of this manual. It is not much harder to begin with one of the files supplied and modify it, if you can determine that your terminal is similar; this is very often possible because so many terminals emulate others. If your system has a terminfo or termcap database, you can use the term2vid utility (q.v.) to make a functional video file from that information. Finally, if you must start from scratch, you should start with the minimal subset defined in Section 19.1.4, and add entries one at a time.

- Most of this manual should be used for reference only. The sample video file in Section 19.1.5 is suitable for a large number of terminals, and may be all that you need.
- Section 19.1.2 describes the concept of the video file.
- Section 19.1.3 describes the text file format.
- Section 19.1.6 is a must for users on a PC using MS-DOS. It contains a listing of an appropriate video file and special caveats.
- Section 19.2.2 summarizes the keywords. Sections 19.3ff explain parameterized control sequences, which support cursor positioning, attribute setting, etc.
- A separate section of this chapter describes the vid2bin utility, which translates your video file into a binary format the JYACC FORMAKER library can understand.

Details and examples are in Sections 19.4.1ff; the first four are plenty to get you started. Next look at Sections 19.4.5 and 19.4.5.1 for a general description of attributes. Section 19.4.5.2 discusses latch attributes, the most common kind, and Section 19.4.5.3 area attributes. Using color is described in Section 19.4.5.4. The remaining sections discuss less essential topics, such as borders, graphics, help text, etc. The vid2bin utility supplies reasonable defaults for these entries, so worry about them last of all.

19.1.2 Why Video Files Exist

Differences among terminal characteristics do not affect programs that are line oriented. They merely use the screen as a typewriter. Full-screen editors, like emacs or vi, use the screen non-sequentially; they need terminal-specific ways to move the cursor, clear the screen, insert lines, etc. For this purpose the termcap data base, and its close relative terminfo, were developed. Although closely associated with UNIX, termcap and terminfo are also used on other operating systems. They list the idiosyncrasies of many types of terminals.

Text editors use visual attributes sparingly, if at all. Thus termcap contains minimal information about handling them. Usually there are entries to start and end "stand-out" and sometimes entries to start and end underline. Notably missing are entries explaining how to combine attributes (i.e. reverse video and blinking simultaneously). Terminfo can combine attributes; in practice, unfortunately, the appropriate entries are usually missing.

JYACC FORMAKER makes extensive use of attributes in all combinations, and supports color. Rather than extending termcap with additional codes, which might conflict with other extensions, JYACC decided to use an independent file to describe the terminal specific information.

Termcap uses a limited set of commands; notably missing are conditionals. Terminfo uses an extensive set of commands, however the resulting sequences are excessively verbose (103 characters for the ANSI attribute setting sequence without color). Therefore, JYACC developed a set of commands that extend both termcap and terminfo. Both syntaxes are supported with only minor exceptions. All the commands needed in the video file can be written using terminfo syntax; many can be written using the simpler termcap syntax; and a few can benefit by using the extended commands.

A summary of the commands used to process parameters is in Section 19.3; details and examples follow. Refer to those sections if you have trouble understanding the examples elsewhere in the manual.

19.1.3 Text File Format

The video file is a text file that can be created using any text editor. It consists of many instructions, one per line. Each line begins with a keyword, and then has an equal sign (=). On the right of the equal sign is variable data depending on the keyword. The data may be a number, a list of characters, a sequence of characters, or a list of further instructions.

Comments can be entered into the file by typing a hash # as the first character of the line; that line will be ignored by vid2bin. All the video files distributed by JYACC are documented with comments; we recommend that you do likewise, as many of the entries are necessarily cryptic.

It is essential that the instruction formats listed in this guide be followed closely. In order to make run-time interpretation as efficient as possible, no error checking at all is done then. The vid2bin utility checks for things like missing, misspelled, and superfluous keywords, but not for things like duplicated or conflicting entries.

19.1.4 Minimal Set of Capabilities

The only required entries in the video file are for positioning the cursor (CUP) and erasing the display (ED).

In the absence of other entries, JYACC FORMAKER will assume a 24-line by 80-column screen. The 24th line will be used for status text and error messages, and the remaining 23 will be available for forms. It will assume that no attributes are supported by the terminal. Since non-display is supported by software, that attribute will be available. The underline attribute will be

faked by writing an underscore wherever a blank appears in an underlined field. Clearing a line will be done by writing spaces. Borders will be available, and will consist of printable characters only.

Although JYACC FORMAKER will function with those two entries, it will have limited features. The most glaring shortcoming will be the lack of visual attributes. Speed may also be a problem; the sole purpose of many entries in the video file is to decrease the number of characters transmitted to the terminal.

19.1.5 A Sample Video File

The following video file is for a basic ANSI terminal, like a DEC VT-100.

```
# Display size (these are actually the default values)
LINES    = 24
COLMS    = 80

# Erase whole screen and single line
ED       = ESC [ 2 J
EL       = ESC [ K

# Position cursor
CUP      = ESC [ %i %d ; %d H

# Standard ANSI attributes, four available
LATCHATT = REVERSE = 7 UNDERLN = 4 BLINK = 5 HILIGHT = 1
SGR      = ESC [ 0 %9(%? %t ; %c %; %) m
```

This file contains the basic capabilities, plus control sequences to erase a line and to apply the reverse video, underlined, blinking, and highlighted visual attributes. The entries for CUP and SGR are more complicated because they require additional parameters at run-time. The percent commands they contain are explained meticulously in Section 19.3.

19.1.6 An MS-DOS Video File

By default, JYACC FORMAKER displays data on the console by directly accessing the PC's video RAM. On machines that are not 100% IBM-compatible, it will use BIOS calls instead; use the entry INIT = BIOS to effect that. Under no circumstances does JYACC FORMAKER use DOS calls or the ANSI.SYS driver. Video files for both monochrome and color displays are included with JYACC FORMAKER.

Because JYACC FORMAKER contains special code for the PC display, most of the entries that contain control sequences are irrelevant, and are given a value of PC in the video files distributed by JYACC. You should leave these entries alone, since their presence is required but their values are irrelevant. Entries that don't contain control sequences, such as LINES, GRAPH, and BORDER, can be changed as usual. The PC video file, as distributed, follows.

```
LINES = 25
COLMS = 80
ED = PC
EL = PC
EW = PC
CUP = PC
CUU = PC
CUD = PC
CUB = PC
CUF = PC
CON = PC
COF = PC
SCP = PC
RCP = PC
REPT = PC
```

```
    # Next 2 lines give display attributes for monochrome only
    # The INIT line specifies a blinking block cursor
    LATCHATT = HILIGHT = 1 BLINK = 5 UNDERLN = 4 REVERSE = 7
    INIT = C 0 13 2

    # Next 3 lines give display attributes for color only
    # The INIT line specifies a blinking block cursor
    LATCHATT = HILIGHT = 1 BLINK = 5
    COLOR = RED = 1 BLUE = 4 GREEN = 2 BACKGRND
    INIT = C 0 7 2

    SGR = PC
    CURPOS = 1
    GRTYPE = PC
    ARROWS = 0x1b 0x1a 0x1d
    BORDER = SP   SP   SP   SP   SP   SP   SP   SP   \
             0xda 0xc4 0xbf 0xb3 0xb3 0xc0 0xc4 0xd9 \
             0xc9 0xcd 0xbb 0xba 0xba 0xc8 0xcd 0xbc \
             0xd5 0xcd 0xb8 0xb3 0xb3 0xd4 0xcd 0xbe \
             0xd6 0xc4 0xb7 0xba 0xba 0xd3 0xc4 0xbd \
             0xdc 0xdc 0xdc 0xdd 0xde 0xdf 0xdf 0xdf \
             .    .    .    .    .    .    .    .    \
             0xb0 0xb0 0xb0 0xb0 0xb0 0xb0 0xb0 0xb0 \
             0xb2 0xb2 0xb2 0xb2 0xb2 0xb2 0xb2 0xb2 \
             0xdb 0xdb 0xdb 0xdb 0xdb 0xdb 0xdb 0xdb
```

Here the INIT specifies the cursor style; refer to the section on INIT.

19.2 Video File Format

19.2.1 General Information

All white space (spaces and tabs) is skipped, except where noted below. A logical line may be continued to the next physical line by ending the first line with a backslash. (Do not leave a space between the backslash and the newline.) To enter a backslash as the last character of the line, use two backslashes (without spaces). Thus

```
    text \         means a continuation line
    text \\        ends with a backslash
    text \\\       has a backslash and a continuation
```

A double quote " starts a string. The quote itself is skipped; text between it and the next double quote (or the end of the line) is taken literally, including spaces. To include a double quote in a quoted string, use backslash quote \" with no space between. For example,

```
    "stty tabs"    has an embedded space
    stty tabs      does not.
```

The percent sign is a control character; to enter a literal percent sign, you must double it (i.e. %%).

There are three ways to put non-printing characters, such as control characters, in the video file:

1. Any character at all can be entered as 0x followed by two hexadecimal digits. For example, 0x41 can be used for A, 0x01 for control-A, etc. This method is particularly useful for entering codes in the range 0x80 to 0xff.
2. Control characters in the range 0x01 to 0x1f can be represented by a caret ^ followed by a letter or symbol. Either ^A or ^a can represent SOH (0x01). The symbols are ^[for ESC, ^\ for FS, ^] for GS, ^^ for RS and ^_ for US.

3. More control characters can be represented by two- or three-character ASCII mnemonics. This method is particularly useful for entering control sequences to the terminal, since the manuals often list such sequences using mnemonics. Here is a list:

		DLE	0x10			DSC	0x90
SOH	0x01	DC1	0x11			PU1	0x91
STX	0x02	DC2	0x12			PU2	0x92
ETX	0x03	DC3	0x13			STS	0x93
EOT	0x04	DC4	0x14	IND	0x84	CCH	0x94
ENQ	0x05	NAK	0x15	NEL	0x85	MW	0x95
ACK	0x06	SYN	0x16	SSA	0x86	SPA	0x96
BEL	0x07	ETB	0x17	ESA	0x87	EPA	0x97
BS	0x08	CAN	0x18	HTS	0x88		
HT	0x09	EM	0x19	HTJ	0x89		
NL	0x0a	SUB	0x1a	VTS	0x8a		
VT	0x0b	ESC	0x1b	PLD	0x8b	CSI	0x9b
FF	0x0c	FS	0x1c	PLU	0x8c	ST	0x9c
CR	0x0d	GS	0x1d	RI	0x8d	OCS	0x9d
SO	0x0e	RS	0x1e	SS2	0x8e	PM	0x9e
SI	0x0f	US	0x1f	SS3	0x8f	APC	0x9f
SP	0x20	DEL	0x7f				

The rightmost two columns are extended ASCII control codes, which can be transmitted only if the communication line and terminal use eight data bits. If this is not possible, the 8-bit code may be replaced by two 7-bit codes: the first code is ESC (0x1b), the second 0x40 less than the desired 8-bit control character. For example, CSI (0x9b) would be replaced by ESC 0x5b, or ESC [. If a video file contains extended ASCII control codes, JYACC FORMAKER will assume they can be used; it will not transmit the two-character sequence automatically.

Note: PRIME computers, and some others, internally toggle the high bit of a character; ESC on a PRIME is 0x9b and CSI is 0x1b, not vice versa. The numbers given in this document are always standard ASCII.

19.2.2 Keyword Summary

All the video file entry keywords are listed here, arranged by function. Subsequent sections explain each one in detail.

	basic capabilities
LINES	number of lines on screen
COLMS	number of columns on screen
INIT	initialization sequence
RESET	undoes initialization sequence
REPT	repeat following character
REPMAX	maximum number of repeated characters
BOTTRT	last position of screen may be written without scrolling the display
BUFSIZ	number of characters to accumulate before flushing erasure commands
ED	erase entire display
EL	erase to end of current line
EW	erase window
	cursor appearance
CON	turn cursor on
COF	turn cursor off
SCP	save cursor position and attribute
RCP	restore cursor postion and attribute
INSON	insert-mode cursor
INSOFF	overstrike-mode cursor
	cursor position
CUP	absolute cursor position
CUU	cursor up

CUD	cursor down
CUF	cursor forward
CUB	cursor backward
CMFLGS	allowed cursor-motion shortcuts display attributes
COLOR	list of colors
LATCHATT	list of available latch attributes
SGR	set graphics rendition (latch)
AREAATT	list of available area attributes
ASGR	set graphics rendition (area)
ARGR	remove are attribute message line
OMSG	open message line
CMSG	close message line
MSGATT	message line attributes softkey labels
KPAR	function key labels description
KSET	load function key label graphics
MODE0	normal character set sequence
MODE1	locking shift to alternate character set 1
MODE2	locking shift to alternate character set 2
MODE3	locking shift to alternate character set 3
MODE4	non-locking shift to alternate character set 1
MODE5	non-locking shift to alternate character set 2
MODE6	non-locking shift to alternate character set 3
GRAPH	graphics character equivalences
GRTYPE	shortcut for defining graphics characters
ARROWS	shift indicator graphics characters
BELL	"visible bell" alarm sequence borders
BORDER	characters that make up the 10 border styles
BRDATT	available border attributes xform help
FMKRDS	draw-screen mode function keys
FMKRTM	test-screen mode function keys
FMKRCP	copy-field function key
FMKRMV	move-field function key
CURPOS	status line cursor position display

19.3 Parameterized Character Sequences

Certain control sequences cannot be completely specified in advance. An example is the cursor position sequence, which requires the line and column to move to. The commands using these sequences must be passed extra parameters. The following keywords take the indicated number of parameters:

REPT	repeat sequence (2) character and number of times to repeat
EW	erase window (5) start line, start column, number of lines, number of columns, background color
CUP	cursor position (2) line and column (relative to 0)
CUU	cursor up (1) line increment
CUD	cursor down (1) line increment
CUF	cursor forward (1) column increment
CUB	cursor backward (1) column increment
SGR	set latch graphics rendition (11) see section 19.4.5

```
     ASGR set area graphics rendition (11)
          see section 19.4.5.1
```

19.3.1 Summary of Percent Commands

Parameters are encoded in sequences by percent commands, sequences starting with the % symbol. This is superficially similar to the way the C library function printf handles parameters. Some percent commands cause data to be output; others are used for control purposes. Every parameter that is to be output requires a percent command. JYACC FORMAKER uses a stack mechanism as does terminfo; it is described in the next secion. Percent commands are summarized in the list that follows. Examples and more complete descriptions are in subsequent sections.

Since all sequences go through the same processing, even those that do not use run-time arguments, percent signs must be used with care. In particular, to enter a percent sign as a literal, you must use %%.

In the following list, each command is tagged with C, I, or E to indicate whether it is a termcap, terminfo, or JYACC extended command.

Output commands

%%	output a percent sign (C and I)
%.	output a character (C)
%c	output a character (I)
%d	output a decimal (C and I)
%#d	output a #-digit decimal, blank filled (I)
%0#d	output a #-digit decimal, zero filled, like the termcap %2 which is not supported (I)
%+	add and output a character (C)
%#z	output # (decimal number) binary zeroes (E)
%#w	wait (sleep) # seconds (E)

Stack manipulation and arithmetic commands

%p#	push parameter # (1 - 11 allowed) (I)
%'c'	push the character constant c (I)
%{#}	push the integer constant # (I)
%+ %- %* %/ %m	add, subtract, multiply, divide, modulus (I)
%| %^ %&	bit-wise or, exclusive or, and (I)
%= %> %<	logical conditionals (I)
%! %~	logical not, one's complement (I)

Parameter sequencing and changing commands

%#u	discard # parameters (E)
%#b	back up # parameters (E)
%i	increment the next two parameters (C and I)
%r	reverse the next two parameters (C)

Control flow commands

%? expr %t then-part %e else-part %;	conditionally execute one of two command sequences (I)
expr %t then-part %e else-part %;	same effect as previous (E)
%#(... %)	repeat the sequence # times (E)
l(... %)	select operations from a list (E)

Terminfo commands not supported

%s	strings
%P, %g	letter variables
$<#>	padding (use %#z instead)

19.3.2 Automatic Parameter Sequencing

A stack holds all the parameters being processed. It is four levels deep; anything pushed off the end is lost. There are commands that push a parameter or constant onto the stack, but no explicit pop commands. Output commands transmit the value on top of the stack, then remove it. Arithmetic and logical operations take one or two operands from the top of the stack, and replace them with one result; thus they perform an implicit pop.

Arithmetic and logical operations all use postfix notation: first the operands are pushed, then the operation takes place. Thus %p1 %p2 %p3 %+ %* leaves x * (y + z) on the stack, where x, y, and z are parameters 1, 2 and 3. This mechanism is identical to that used by terminfo, so its commands can be used freely.

The simpler termcap commands do not use a stack mechanism. To support them, JYACC FORMAKER uses an automatic parameter sequencing scheme. A current index into the parameter list is maintained. Whenever a parameter is needed on the stack, the current parameter is pushed and the index is incremented. In particular, if an output command is encountered and there is nothing on the stack to output, an automatic push is performed using the current index. The commands %d %d output two decimals; the sequence %p1 %d %p2 %d does the same thing.

The effect of this scheme is that termcap style commands are automatically translated into terminfo style. Most of the examples in this document give both styles. Although it is possible to use automatic sequencing and explicit parameter pushes in the same sequence, this practice is strongly discouraged. Explicit pushes of constants with automatic parameter sequencing, however, is a useful combination, as will be seen.

19.3.3 Stack Manipulation and Arithmetic Commands

Commands are available to push parameters and constants. Only four levels of stack are supported, and anything pushed off the end is discarded without warning.

%p2	push the second parameter
%p11	push parameter 11
%'x'	push the character x
%{12}	push the number 12
%{0}	push binary 0
%'0'	push ASCII 0

Various arithmetic and logical operations are supported. They require one or two operands on the stack. If necessary an automatic push will be generated, using the next parameter.

%'@' %| %| %| %c	or three parameters with @, then output the result.
%'@' %p1 %| %p2 %| %p3 %| %c	same as above

The automatic parameter sequencing mechanism works well in the above example. Since or requires two parameters and there is only one on the stack, a push is performed. Note that no push is required to process %c as an entry already exists on the stack. Thus only three parameters are consumed and the result of the bitwise or is output.

%'SP' %+ %c	output the parameter added to the value of a space. See the next section for an alternate.
%p1 %'SP' %+ %c	same as above

The example above first pushes the first parameter, then pushes a space character (0x20). The %+ command adds these values and puts the answer on the stack. %c then pops this value and transmits it to the terminal.

19.3.4 Parameter Sequencing Commands

With automatic sequencing of parameters, it is occasionally necessary to skip a parameter. The %u command uses up one parameter, by incrementing the parameter index. The %b command backs up, by decrementing the parameter index. Both can be given with counts, as %2u.

```
%d %b %d                output the same parameter twice
%p1 %d %p1 %d           same as above
%p2 %d %p1 %d           output in reverse order
%u %d %2b %d            same as above
```

19.3.5 Output Commands

Because the percent sign is a special character, it must be doubled to output a percent sign. %c and %. output a character, like printf; the latter is supplied for termcap compatibility. %d outputs a decimal. It has variations that allow for specifying the number of digits, and whether blank or zero fill is to be used.

%#z outputs the specified number of NUL characters (binary zero). It is usually used for padding, to insert a time delay for commands such as erase screen.

```
%%          output a percent sign
%d          output a decimal, any number of digits, no fill
%3d         output at most 3 digits with blank fill
%03d        output at most 3 digits with zero fill
%100z       100 pad bytes of 0 are sent to the terminal
```

%S(string %) issues a system command; the string following %S is passed to the command interpreter fpr execution. Since vid2bin strips spaces, this text should usually be enclosed in quotes.

```
%S("stty tabs"%)        System call: stty tabs
%S(stty SP tabs%)       System call: stty tabs
%S( stty tabs %)        Mistaken version of above
%S("keyset \"\""%)      System call: keyset ""
%S("keyset """%)        Mistaken version of above.
```

%#w waits (sleeps) the specified # of seconds. It is not supported on systems where the sleep library routine is unavailable. It is often used as a time delay for INIT and RESET sequences.

```
%2w         sleep 2 seconds
```

Because termcap and terminfo are inconsistent, %+ is implemented in two ways. As described in the section above, %+ can be used to add two operands on the stack and leave the sum on the stack. If the stack has only one entry, an automatic push is generated. However, a special case occurs if the stack is empty: the character following %+ is added to the next parameter, the sum is output as a character, and the parameter index is incremented. This usage occurs often in termcap cursor positioning sequences.

```
%+SP                    output parameter added to the value of space
%'SP' %+ %c             same as above
%'SP' %p1 %+ %c         same as above
```

19.3.6 Parameter Changing Commands

%i increments the next two parameters. It is used almost exclusively in termcap cursor positioning sequences. The parameters passed are line and column, with the upper left being (0, 0). Many terminals expect the line and column to be relative to (1, 1); %i is used to increment the parameters. Note that no output is performed, and no parameters are consumed.

%r reverses the next two parameters. It is unnecessary if explicit parameter pushes are used; in fact, it should be avoided in that case since the numbering of the parameters will be reversed. This command is often used in cursor positioning sequences, where the terminal requires that column be given first and then the line (the default being the other way around).

ESC [%i %d ; %d H	Add 1 to each parameter and send out as decimals
FS G %r %c %c	output column first, then line
FS G %p2 %c %p1 %c	same as above

19.3.7 Control Flow Commands

The general if-then-else clause is %? expr %t then-part %e else-part %; . It can be abbreviated by omitting the if, thus: expr %t then-part %e else-part %; . The expression expr is any sequence, including the empty sequence. %t pops a value from the stack and tests it, executing then-part if it is true (non-zero) and else-part otherwise. Then-part and else-part may be any sequence, including the empty sequence. If else-part is empty, %e may be omitted as well; but %t is always required, even if then-part is empty.

If %t finds that the stack is empty, it will generate an automatic push of the next parameter as usual. %t consumes one parameter; however, the incrementing of the parameter index is delayed until after the entire conditional has been executed. A conditional always consumes exactly one parameter, regardless of which branch is taken or of the content of then-part or else-part. If either of those use automatic parameter sequencing, they use a local index; thus even if they consume, say, two parameters, at the end of the conditional the parameter index is reset. When the next command is reached, only one parameter has been consumed.

In each of the following examples, one parameter is consumed, even in the last one where no parameter is output.

%t ; %c %;	output ; and a character if the parameter is non-zero, otherwise skip the parameter.
%p1 %t ; %p1 %c %;	same
%? %p1 %t ; %p1 %c %;	same
%? %p1 %t ; %c %;	same
%t ; 5 %;	output ; and 5 if the parameter is non-zero.

In the following two examples, the constant (binary) 1 is pushed, the parameter is compared with 1, and the boolean value is left on the stack. If the value is true, nothing is done; otherwise the parameter is output as a decimal.

```
%? %{1} %p1 %= %t %e %p1 %d %;
%{1} %= %t %e %d %;
```

The following sequence exhibits a simple "either-or" condition that is sometimes used to toggle an attribute on or off. It outputs ESC (if the parameter is non-zero, and ESC) otherwise.

```
ESC %t ( %e ) %;
```

The then-part and else-part may themselves contain conditionals, so else-if can be implemented. This practice is not recommended as it can produce undecipherable sequences. Also, because of the way automatic parameter sequencing is done, the results might be unexpected. It is provided only for terminfo compatibility. The list command, described below, is an alternative.

The repeat command is used to perform the same action for several parameters. It is designed to be used with automatic parameter sequencing, and is almost useless if explicit parameter pushes are used. The count is specified after the percent sign. All the commands between %#(and %) are executed # times.

```
%3( %d %)                              output 3 decimals
%p1 %d %p2 %d %p3 %d                   same as previous
%3( %t %d %; %)                        output whichever of the first three
                                       parameters are non-zero.
%p1 %t %p1 %d %; %p2 %t %d %; %p3 %t %p3 %d %;
                                       same as previous
ESC 0 %9( %t ; %c %; %) m              usual ANSI sequence for SGR.
ESC 0 %? %p1 %t ; 7 %; %? %p2 %t ; 2 ...
                                       same as above, assuming that parameter
                                       1 is 7 and parameter 2 is 2
```

19.3.8 The List Command

The list command is needed very rarely, but is available as an alternate to a complicated if-then-elseif construct. It implements a simple "select" or "case" conditional. The general format is %l(value1: expr1 %; value2: expr2 %; ... %)

The values are single character constants representing the various cases. The expression is executed if the value matches the top of stack. At most one expression is executed, i.e. each case contains a "break". If the value is missing the expression is evaluated as a default. For correct operation, the default case must occur last in the list. Note that the colons do not have a leading percent sign, so no selector may be a colon. The %; after the last element of the list is not required.

The parameter on the stack (automatically pushed, if necessary) is popped and tested against the various cases. The parameter index is incremented by 1 after the entire list is processed, even if the expressions use parameters. The following examples are a bit contrived; see the section on color for a live example.

```
%l( 0:%; 1:ESC%; :FS %)                output nothing if the parameter is
                                       '0'; ESC if it is '1'; FS otherwise.
%'0' %= %t %e %'1' %= %t ESC %e FS %; %;
                                       same result, using "else-if"
%l( 1:2%; 2:1%; %)                     output '1' if the parameter is '2',
                                       '2' if the parameter is '1'; otherwise
                                       do nothing
```

19.3.9 Padding

Certain terminals (or tty drivers) require extra time to execute instructions. Sometimes the terminal manual specifies the delay required for each command, but more often than not it is silent on the subject. If random characters appear on the screen, particularly characters that are part of command sequences, time delays may be required.

Delays can be introduced in two ways. %#w will cause a wait (sleep) for the specified number of seconds; %#z will output the specified number of zeros. The wait command is usually only required in terminal initialization or reset sequences. A "hard reset" of a terminal sometimes requires a sleep of 1 or 2 seconds. The zero command is occasionally needed on the erase display or erase line commands. Very rarely the cursor positioning sequence requires padding. The number of zeros to send range from about 5, for very short delays, to several thousand for longer delays. Usually 100 or so is enough for any terminal.

termcap indicates padding by using a number at the beginning of a sequence, which is the number of milliseconds of pad required. terminfo uses the syntax $<#>. In either case it is easy to convert to the %#z notation, using the fact that, at 9600 baud, one character takes one millisecond to output.

```
ESC c %2w              sleep 2 seconds after terminal reset
ESC [ J %100z          100 pad zeros after clear screen
ESC [ H %1000z         long delay of 1000 pad zeros
```

19.4 Constructing a Video File, Entry by Entry

19.4.1 Basic Capabilities

LINES indicates the number of lines on the display. The default value is 24. In general one line will be reserved for status and error messages so the maximum form size will usually be one less than the number specified here. (See OMSG, below, for exceptions.) COLMS gives the number of columns on the display. The default value is 80.

```
LINES = 25              24 lines for the form, 1 for messages
COLMS = 132             wide screen
LINES = 31              SUN workstation
```

INIT is a terminal initialization sequence, output by the library function initcrt. There is no default; this keyword may be omitted. It is typically used to change the mode of the terminal, to map function keys, select attribute styles, etc. Padding is sometimes required, either with %#z or %#s.

RESET is a reset-terminal sequence, output by the library function resetcrt. There is no default. If given, this keyword should undo the effects of INIT. For many terminals a "hard reset" that resets the terminal to the state stored in non-volatile memory is appropriate.

```
# map 2 function keys, then wait 2 seconds
INIT = %S( "/etc/keyset f1 ^a P ^m" %) \
       %S( "/etc/keyset f2 ^a Q ^m" %) \
       %2w

# load alternate character sets
INIT = ESC)F ESC*| ESC+}

# hard reset, delay, then set tabs
RESET = ESC c %1000z %S("stty tabs"%)
```

On MS-DOS systems only, the INIT and RESET sequences (which are normally not used) may be given a special value to specify the cursor style. With ASCII terminals, escape sequences for setting the cursor style may be included in the INIT and RESET strings in the normal fashion. The format is

```
INIT   = C n1 n2 n3
RESET  = C n1 n2 n3
```

The first two numbers, n1 and n2, specify the top and bottom scan lines for the cursor block; line 0 is at the top. The optional n3 gives the blink rate, as follows:

```
1         no cursor
2         fast blink (the default)
3         slow blink
0         fast blink (Not always valid on non-IBM systems)
```

The standard sequences, for a blinking block cursor, are INIT = C 0 13 0 for a monochrome monitor, and INIT = C 0 7 0 for a CGA monitor (with lower resolution). If RESET is not specified, JYACC FORMAKER saves and restores the original cursor style.

A scan line is the smallest vertical unit on your display (it is one pixel wide).

Two additional special keywords may be used with INIT on MS-DOS systems. BIOS specifies that JYACC FORMAKER should use BIOS calls to do display output rather than writing the video RAM directly. XKEY actually controls keyboard input; it

directs JYACC FORMAKER to use a different BIOS interrupt for keyboard input, one that recognizes the F11 and F12 keys on an extended keyboard.

REPT is a repeat-character sequence. There is no default, since most terminals do not support character repeat. If it is available, it should be given as it can substantially speed up clearing of windows, painting of borders, etc. This sequence is passed two parameters; the character to be repeated and the number of times to display it. The repeat sequence will be used whenever possible, usually for borders and for clearing areas of the screen. If borders do not appear correctly, this sequence may be wrong. A repeat sequence is never used to repeat a control character, and will never extend to more than one line.

REPMAX gives the maximum number of characters REPT can repeat. To check the proper value of REPMAX, first omit it; then, in xform, draw a field that extends the entire width of the screen, and hit the TRANSMIT key. If the whole field changes to the underline attribute, REPMAX is not needed. If it doesn't, experiment by gradually shortening the field to determine the largest possible value of REPMAX.

```
     REPT = %c ESC F %+?        output character, then ESC F
                               and the count with 0x3f (the
                               ASCII value of '?') added
     REPMAX = 64                maximum count for above.
                               Anything over this count will
                               be split into more sequences
     REPT = %p1 %c ESC F %'?' %p2 %+ %c         same as previous
```

BOTTRT is a simple flag, indicating that the bottom right-hand corner of the display may be written to without causing the display to scroll. If this flag is not present, JYACC FORMAKER will never write to that position.

BUFSIZ sets the size of the output buffer used by JYACC FORMAKER. If it is omitted, JYACC FORMAKER calculates a reasonable default size, so you should include it only if special circumstances warrant. If you make extensive use of a screen-oriented debugger, you may want to set BUFSIZ to a large value; that effectively disables the delayed-write feature, which may prove troublesome during debugging.

19.4.2 Screen Erasure

ED gives the control sequence that erases the display. It is required, and must clear all available display attributes, including background color. The correct command can be found in the terminal manual, or in termcap as "cl". Some terminals require padding after this command.

```
   ED = ESC [ J            common for ANSI terminals
   ED = CSI J              ANSI terminals, 8 bit mode
   ED = ESC [ H ESC [ J    "home" may be required too
   ED = ESC [ 2 J          another variation
   ED = ESC [ 2 J %100z    with padding
   ED = ^L                 videotex terminals
   ED = FF                 same as above
```

EL gives a sequence that erases characters and attributes from the cursor to the end of the line. If it is not given, JYACC FORMAKER erases the line by writing blanks. The sequence can be found in termcap as "ce". Padding may be required. EL = ESC [K is common for ANSI terminals; to include padding, use EL = ESC [0 K %100z .

EW gives a sequence that erases a rectangular region on the screen, to a given background color if available. The only known terminal where this is available is a PC using MS-DOS. Five parameters are passsed: start line, start column, number of lines, number of columns, and background color. (If color is not

available, the last parameter can be ignored.) On a PC using MS-DOS, EW should be specified as ESC [%i %d; %d; %d; %d; %c w .

19.4.3 Cursor Position

CUP, absolute cursor position, is required to run JYACC FORMAKER. This sequence appears in termcap as "cm". It takes two parameters: the target line and the target column, in that order and relative to 0. %i (increment) can be used to convert them to be relative to 1. ANSI terminals need the line and column as decimals. Other terminals add a fixed value to the line and column to make them printable characters; %+ is used to implement this. Some typical descriptions follow; all are ANSI standard.

```
CUP = ESC [ %i %d;%d H
CUP = ESC [ %i %d;%d f
CUP = ESC [ %i %p1 %d ; %p2 %d f
CUP = CSI %i %d; %d H
```

Another common scheme is to output the line and column as characters, after adding SP. Terminal manuals tend to obscure this method, as the following excerpt shows:

> Address or load the cursor by transmitting ESC = r c where r is an ASCII character from the table for the row (line) and c is an ASCII character from the table for the column:
>
> | row/column | ASCII code |
> |---|---|
> | 1 | Space |
> | 2 | ! |
> | 3 | " |
> | ... | ... |

Examples of coding in the video file follow.

```
CUP = FS C %+SP %+SP
CUP = FS C %'SP' %p1 %+ %c %'SP' %p2 %+ %c
CUP = ESC = %+SP %+SP
```

CUU, CUD, CUF and CUB perform relative cursor movement. CUU moves the cursor up in the same column; CUD moves it down. CUF moves the cursor forward in the same row and CUB moves it back. All take as a parameter the number of lines or columns to move. If sequences exist to move the cursor by one line or column but not more, do not specify them.

```
CUU = ESC [ %d A        ANSI cursor up
CUD = ESC [ %d B        cursor down
CUF = ESC [ %d C        cursor forward
CUB = ESC [ %d D        cursor back
CUU = CSI %d A          using 8 bit codes
CUU = ESC [ %{1} %= %t %e %d %; A
                        omit the parameter if it is 1
```

The CMFLGS keyword lists several shortcuts JYACC FORMAKER can use for cursor positioning. They are as follows:

CR	Carriage return (0x0d, or ^M) moves the cursor to the first column of the current line.
LF	Linefeed (0x0a, or ^J) moves the cursor down one line, in the same column.
BS	Backspace (0x08, or ^H) moves the cursor one position to the left, without erasing anything.
AM	Automatic margin: the cursor automatically wraps to column 1 when it reaches the right-hand edge of the display.

Most terminals are capable of the first three. The fourth can frequently be found in termcap, as am.

19.4.4 Cursor Appearance

CON turns the cursor on in the style desired. Since an underline cursor is difficult to see in an underlined field, we recommend a blinking block cursor. Note that the INIT and RESET sequences can be used to switch between the cursor style used in JYACC FORMAKER applications and that used on the command line.

COF turns the cursor off. If possible this sequence and CON should be given. Menus (using a block cursor) look better with the regular cursor off. Also JYACC FORMAKER often must move the cursor around the screen to put text in fields, to scroll arrays, etc.; if the cursor is off during these operations, the user is not disturbed by its flickering all over the screen.

Many terminals have no ability to turn the cursor on and off. Although JYACC FORMAKER attempts to minimize cursor movement, some flickering is unavoidable.

CON and COF can sometimes be found in the terminal manual as "cursor attributes" and in termcap as CO and CF. Here are some examples.

```
CON = ESC [             cursor on for videotex terminal
COF = ESC ]             cursor off for videotex
CON = ESC [>5l          cursor on for some ANSI terminals
COF = ESC [>5h          and off
CON = ESC [?25h         another possibility for ANSI terminals
COF = ESC [?25l
CON = ESC [ 3 ; 0 z
COF = ESC [ 3 ; 4 z
```

SCP and RCP save and restore the cursor position, respectively. JYACC FORMAKER must often move the cursor temporarily, as to update the status line. Beforehand, it saves the current cursor position and attribute, and restores them afterwards. Some terminals offer a pair of sequences that perform these two actions, producing less output than the cursor position and attribute setting sequences together. Thus, if they are available, JYACC FORMAKER can run faster. Terminal manuals refer to these sequences in many ways, the most obscure being "cursor description." Here is an example, commonly found in ANSI terminals.

```
SCP = ESC 7
RCP = ESC 8
```

The INSON and INSOFF sequences are issued to the terminal when you toggle JYACC FORMAKER's data entry mode between insert and overstrike, using the INSERT key. They should change the cursor style, so that you can easily see which mode you are in. On many terminals, changing the cursor style also turns it on; in this case, INSOFF should be the same as COF, or you can omit it altogether. If the cursor style can be changed without turning it on or off, you should give both INSON and INSOFF.

19.4.5 Display Attributes

JYACC FORMAKER supports highlight, blink, underline and reverse video attributes. If either highlight or blink is not available, low intensity is supported in its place. The keywords LATCHATT and AREAATT in the video file list the attributes available in each style and associate a character with each attribute.

The set graphics rendition sequences (SGR and ASGR) are each passed eleven parameters. The first nine are the same as used by terminfo; only five of them are actually used by JYACC FORMAKER. The last two specify foreground and background color, and are omitted if color is not available. The parameters, in order, represent:

1. standout not supported, always 0
2. underline
3. reverse video
4. blink
5. dim (low intensity)
6. highlight (bold)
7. blank supported by software, always 0
8. protect supported by software, always 0
9. alternate charsupported in other sequences, 0
10. foreground color
    (if available)
11. background color
    (if available)

If an attribute is desired, the parameter passed is the character associated with the attribute, as explained below. If the attribute is not desired, the parameter passed is (binary) 0. If the video file contains LATCHATT = REVERSE = 7 HILIGHT = 1 BLINK = 5 UNDERLN = 4 , and a field is to be highlighted and underlined, the SGR sequence will be passed (0, '4', 0, 0, 0, '1', 0, 0, 0) . The second and sixth parameters respresent underline and highlight; they are set to the corresponding values from LATCHATT. The rest are zero. To make the field reverse video and blinking, SGR would be passed (0, 0, '7', '5', 0, 0, 0, 0, 0) .

If no attributes are specified in the video file, JYACC FORMAKER will support just two attributes: non-display (done in software anyway) and underline (using the underscore character).

19.4.5.1 Attribute Types

JYACC FORMAKER supports three different kinds of attribute handling. The first is called latch attributes, and is the most common today. The position of the cursor is irrelevant when the attribute setting sequence is sent. Any characters written thereafter take on that attribute. Attributes require no space on the screen. ANSI terminals use this method.

The second is called area attributes. The cursor position is very important at the time the sequence to set the attribute is sent to the terminal. Indeed, all characters from the cursor position to the next attribute (or end of line or end of screen) immediately take on that attribute. Attributes do not occupy a screen position (they are "non-embedded" or "no space"). In this style, JYACC FORMAKER will position the cursor to the end of the area to be changed, set the ending attribute, then position the cursor to the beginning of the area and set its attribute.

The third is called onscreen attributes. They act like area attributes, but occupy a screen position. (They are "embedded" or "spacing".) This style of attribute handling imposes the condition on the screen designer that fields and/or display areas cannot be adjacent, since a space must be reserved for the attribute. Display of windows may be hampered by lack of space for attributes.

A terminal may have several user-settable modes. It is quite common for a terminal to support both area and onscreen attributes. If so, you should select area ("non-embedded" or "no space") over onscreen ("embedded" or "spacing"). Some terminals support one latch attribute and several area attributes simultaneously.

If a terminal has only one attribute style available, it is often user selectable. We recommend that reverse video be selected, since it is attractive in borders. JYACC FORMAKER supports non-display in software, so that attribute need not be available. Underlines will be faked (by writing an underscore character) if that attribute is not available.

Usually attribute information is available only in the terminal manual. However some clues can be found in the termcap data base. The codes "so", "ul" and "bl" specify standout (usually reverse video), underline and bold respectively. The codes "se", "ue" and "be" give the sequence to end the attributes. The standard ANSI sequences are

```
so=\E[7m:se=\E[0m:ul=\E[4m:ue=\E[0m:bl=\E[1m:be=\E[0m
```

If you find something like these you can be quite sure that ANSI latch attributes are available. If you find entries ug#1:sg#1 you can be sure that onscreen attributes are in use.

19.4.5.2 Specifying Latch Attributes

The LATCHATT keyword is followed by a list of attributes equated to their associated character. The possible attributes are:

REVERSE	reverse (or inverse) video
BLINK	blink or other standout
UNDERLN	underline
HILIGHT	highlight (bold)
DIM	dim (low intensity)

The format is LATCHATT = attribute = value attribute = value etc. If the equal sign and value are missing, the attribute is given the value (binary) 1.

Most ANSI terminals use latch attributes and the codes are fairly standardized. The only question is which attributes are supported and how attributes can be combined, if at all. Some ANSI terminals support color, either foreground only or foreground and background. The sequences for color are far less standard.

Terminal manuals often describe the sequence as "set graphics rendition." A common description reads:

```
ESC [ p1 ; p2 ; ... m
where pn = 0    for normal
           1    for bold
           5    for blink
           ...
```

Thus ESC [0 m is normal, ESC [1 m is bold, ESC[1 ; 5 m is bold and blinking. Often setting an attribute does not "erase" others, so it is best to reset to normal first, using ESC[0; 1 m for bold, ESC[0;1;5m for blinking bold, etc. The coding in the video file is as follows:

```
LATCHATT = HILIGHT = 1 BLINK = 5 UNDERLN = 4 REVERSE = 7
SGR = ESC [ 0 %9(%t ; %c %; %) m
```

The meaning of the above SGR sequence is as follows. The sequence is passed 11 parameters, each 0 (if the attribute is not to be set) or the character in the LATCHATT list. First, ESC [0 is output. The %t test, repeated 9 times, causes the zero parameters to be skipped. A non-zero parameter causes a semicolon and the parameter to be output. Finally, the character m is output. If normal attribute is wanted, all parameters will be 0, and the output sequence will be ESC [0 m. If only underline is wanted, it will be ESC [0 ; 4 m. If highlighted, blinking, and reverse video are desired, the output will be ESC [0; 7 ; 5 ; 1 m.

Some terminals (or emulators) will not accept the method of combining attributes used above. In that case, one sequence followed by the next might work, e.g. ESC [1 m ESC [7 m. Some terminals cannot combine attributes at all. Here are some more ANSI and near-ANSI examples:

```
LATCHATT = HILIGHT = 1 BLINK = 5 UNDERLN = 4 REVERSE = 7
```

```
                    "standard" ANSI terminal

     LATCHATT = DIM = 2 REVERSE = 7 UNDERLN = 4 BLINK = 5
                    ANSI with low intensity but no highlight

     LATCHATT = REVERSE = 7
                    only one attribute available

     SGR = ESC [ 0 %9(%t  ; %c %; %) m
                    repeat of previous example

     SGR = ESC [ 0 m %9(%t  ESC [ %c m %; %)
                    attributes not combinable

     SGR = %u ESC [ 0 %5(%t ; %c %; %) m
                    skip parameters that are always 0
```

In the next LATCHATT/SGR example we will use explicit pushes to select the appropriate parameter. The second pair is the same as the first, but the attribute is treated as a boolean. The first uses the optional %?, the second omits it.

```
     LATCHATT = DIM = 2
     SGR = ESC [ m %? %p5 %t ESC [ 2 m %;

     LATCHATT = DIM
     SGR = ESC [ m %t ESC [ 2 m %;
```

The following is suitable for terminals that support all attributes but cannot combine them. It selects one attribute giving preference to REVERSE, UNDERLN, BLINK and HILIGHT in that order. It uses a complicated "if-then-elseif-elseif-elseif" structure. Automatic parameter sequencing cannot be relied on, so explicit parameter pushes are used.

```
     LATCHATT = HILIGHT BLINK UNDERLN REVERSE
     SGR = ESC [ %p3 %t 7 %e %p2 %t 4 %e %p4 %t 5 %e\
          %p6 %t 1 %; %; %; %; m
```

Some terminals use bit-mapped attributes. Terminal manuals are not usually explicit on this. Often they use tables like the following:

```
     n       Visual attribute

     0       normal
     1       invisible
     2       blink
     3       invisible blink
     4       reverse video
     5       invisible reverse
     6       reverse and blink
     7       invisible reverse and blink
     8       underline
     9       invisible underline
     :       underline and blink
     ;       invisible underline and blink
     <       reverse and underline
     =       invisible reverse and underline
     >       reverse, underline and blink
     ?       invisible reverse, underline and blink
```

After poring over the ASCII table for a while, it becomes clear that this is bit-mapped, with the four high-order bits constant (0x30) and the four low-order bits varying, like this:

```
x x x x  x x x x
0 0 1 1  | | | |___ invisible
         | | |_____ blink
         | |_______ reverse
         |_________ underline
```

This can be coded in the video file as follows. The attributes are ored with a starting value of '0' (0x30).

```
LATCHATT = BLINK = 2 REVERSE = 4 UNDERLN = 8
SGR = ESC G %'0' %9( %| %) %c
```

The following gives an example for use with a videotex terminal. All are equivalent: the bits are ored together with a starting value of 0x40, or @, and the result is output as a character.

```
LATCHATT = UNDERLN = DLE BLINK = STX REVERSE = EOT HILIGHT=SP
LATCHATT = UNDERLN = ^P BLINK = ^B REVERSE = ^D HILIGHT = SP
LATCHATT = UNDERLN = 0x10 BLINK = 0x02 REVERSE = 0x04 \
           HILIGHT = 0x20
SGR = FS G %u %'%5( %| %) %c

LATCHATT = UNDERLN = P BLINK = B REVERSE = D HILIGHT = `
SGR = FS G %'%9( %| %) %c
```

Some terminals that use area attributes will support a single latch attribute. It is often called "protected" and is used to indicate protected areas when the terminal is operated in block mode. The following example switches between protected and unprotected modes in order to use low intensity. (Be aware that a terminal might become very slow when using the protect feature.) The SGR sequence depends only on the attribute being non-zero, so no value is necessary:

```
LATCHATT = DIM
SGR = ESC %?%t ) %e ( %;
```

19.4.5.3 Specifying Area Attributes

Area or onscreen attributes are specified like latch attributes. The AREAATT keyword lists the area or onscreen attributes that are available and associates a character with each. As for latch attributes, REVERSE, BLINK, UNDERLN, HILIGHT and DIM are available. In addition, several flags are available to specify how the attributes are implemented by the terminal. The flags are:

ONSCREEN	the attribute uses a screen position
LINEWRAP	the attribute wraps from line to line
SCREENWRAP	the attribute wraps from bottom of screen to top
REWRITE	must rewrite attribute when writing character
MAX = #	maximum number of attributes per line

Area and onscreen attributes modify all characters from the start attribute to the next attribute or to an end, which ever is closer. If there is no end, use SCREENWRAP. If the end is the end of screen, use LINEWRAP. If end is the end of the line, omit both wrap flags. Some terminals allow the user to select the style. For onscreen attributes, screen wrap is best and line wrap a good second best; for area attributes the choices are about the same. If the attribute takes up a screen position, use the ONSCREEN flag.

```
AREAATT = REVERSE = i UNDERLN = _ BLINK = b DIM = l
ASGR = ESC s r %u %5(ESC s %c %)

AREAATT = BLINK = 2 DIM = p REVERSE = 4 UNDERLN = 8 \
          ONSCREEN LINEWRAP
ASGR = ESC G %u %'0' %5( %| %) %c
```

Some terminals have the following misfeature: writing a character at the position where an attribute was set can remove the attribute. Immediately after placing the attribute the character may be written with no problems; however, the next time a character is written there, the attribute will disappear. In this case, use the REWRITE flag to tell JYACC FORMAKER to reset the attribute before writing to that position. The following example is for the Televideo 925:

```
AREAATT = REVERSE = 4 UNDERLN = 8 BLINK = 2 REWRITE
ASGR = ESC G %'0' %9( %| %) %c
```

Yet other terminals restrict the number of attributes that are available on a given line. If so, include MAX = #, where # represents the maximum. If possible, also give a "remove attribute" sequence, ARGR. Changing an attribute to normal is not the same as removing it: a normal attribute will stop the propogation of a previous attribute, but a removed attribute will not. If the maximum number of attributes is small, JYACC FORMAKER's performance may be limited.

If there is a remove attribute sequence, JYACC FORMAKER will use it to remove repeated attributes, to avoid exceeding the maximum number of attributes on a line. If there is no maximum, the remove attribute sequence can be omitted. Indeed it often makes the screen "wiggle," which is very unpleasant for the viewer.

```
AREATT = REVERSE = Q UNDERLN = ` MAX = 16
ASGR = ESC d %u %'%5( %| %) %c
ARGR = ESC e
```

19.4.5.4 Color

JYACC FORMAKER supports eight foreground and background colors. The COLOR keyword is used to associate a character with each color, just as LATCHATT associates a character with each attribute. The CTYPE entry has flags that tell JYACC FORMAKER that background color is available. Only the three primary colors need be specified in the video file. If the other colors are not there, they will be generated according to the following rule:

```
BLACK       = BLUE & GREEN & RED
BLUE        must be specified
GREEN       must be specified
CYAN        = BLUE | GREEN
RED         must be specified
MAGENTA     = RED | BLUE
YELLOW      = RED | GREEN
WHITE       = RED | GREEN | BLUE
```

The tenth parameter to SGR or ASGR is the character representing the foreground color; the eleventh is that representing the background color (it is 0 if background color is not available). Many ANSI terminals set foreground color with the sequence ESC [3x m, where x ranges from 0 for black to 7 for white. Background color is often set with ESC [4x m. The order of the colors varies from terminal to terminal.

On color terminals, REVERSE often means black on white. If background color is available, JYACC FORMAKER can do better if REVERSE is not specified in the video file: it will use the specified color as the background, and either black or white as the foreground. The following is often suitable for a color ANSI terminal:

```
LATCHATT = HILIGHT = 1 BLINK = 5
COLOR = RED = 4 GREEN = 2 BLUE = 1 BACKGRND
SGR = %3u ESC [ 0 %3( %?%t ; %c %; %) ; %3u 3%c ; 4%c m
or
SGR = %3u ESC [ 0 %5( %?%t ; %c %; %) m ESC [ 3%c;4%c m
or
```

```
LATCHATT = HILIGHT BLINK
SGR = ESC [ 0 %?%p4%t ;5 %; %?%p6%t ;1 %; m \
      ESC [ 3%p10%c; 4%p11%c m
```

If the terminal has a unique sequence for each color, a list command works well. In the following example, the ANSI attribute sequence (ESC [0 ; p1 ; p2 ; ... m) is used and the values for the colors are:

```
cyan          >1
magenta       5
blue          5 ; > 1
yellow        4
green         4 ; > 1
red           4 ; 5
black         4 ; 5 ; > 1
```

```
LATCHATT = REVERSE = 7 HILIGHT = 2
COLOR = CYAN = 0 MAGENTA = 1 BLUE = 2 YELLOW = 3 GREEN = 4\
  RED = 5 BLACK = 6 WHITE = 7
SGR = ESC [ 0 %p3%t;7%; %p6%t;2%; \
      %l( 0:;>1%; 1:;5%; 2:;5;>1%; 3:;4%; \
      4:;4;>1%; 5:;4;5%; 6:;4;5;>1 %) m
```

Some terminals use ESC [2 ; x ; y m to set color and other attributes. Here x is the foreground color and y is the background color; both numbers range from 0 to 7. If highlight is desired in the foreground, 8 should be added to x. If blink is desired, 8 should be added to y. The following video entries satisfy these requirements:

```
LATCHATT = HILIGHT = 8 BLINK = 8
COLOR = RED = 4 GREEN = 2 BLUE = 1 BACKGRND
SGR = ESC [ 2 ; %p10 %p6 %+ %d ; %p11 %p4 %+ %d m
```

19.4.6 Message Line

JYACC FORMAKER usually steals a line from the screen to display status text and error messages. Thus a 25-line screen (as specified in the LINES keyword) will have 24 lines for the form itself, and one for messages. This use of a normal screen line for messages is the default.

Some terminals have a special message line that cannot be addressed by normal cursor positioning. In that case, the OMSG sequence is used to "open" the message line, and CMSG to close it. All text between these sequences appears on the message line. No assumption is made about clearing the line; JYACC FORMAKER always writes blanks to the end of the line.

```
OMSG = ESC f
CMSG = CR ESC g
```

If the OMSG line keyword is present, JYACC FORMAKER uses all the lines specified in the LINES keyword for forms.

Terminals that use a separate message line may use different attributes on the status line than on the screen itself. JYACC FORMAKER provides some support for this circumstance; for very complicated status lines, the programmer must write a special routine and install it with the statfnc call. (See the Programmer's Guide for details.) The keyword MSGATT lists the attributes available on the message line. This keyword takes a list of flags:

REVERSE	reverse video available
BLINK	blink available
UNDERLN	underline available
HILIGHT	highlight (bold) available
DIM	dim (low intensity) available

LATCHATT	all latch attributes can be used
AREAATT	all area attributes can be used
NONE	no attributes on message line
ONSCREEN	area attributes take a screen position

The attribute for the message line must have been specified as either a latch or area attribute, and the sequence to set it must be given in the SGR or ASGR keyword. For example, if REVERSE is listed in MSGATT and REVERSE is a latch attribute, then SGR is used to set it. Attributes that appear in MSGATT and don't appear in either LATCHATT or AREAATT are ignored.

JYACC FORMAKER must determine the correct count of the length of the line. Thus it is important to know whether area attributes are onscreen or not. It is not uncommon for area attributes to be non-embedded on the screen but embedded on the status line. The keyword ONSCREEN may be included in MSGATT to inform JYACC FORMAKER of this condition.

```
LATCHATT = DIM
AREAATT = REVERSE UNDERLN BLINK
MSGATT = REVERSE UNDERLN BLINK ONSCREEN
MSGATT = AREAATT ONSCREEN
```

The two MSGATT entries are equivalent. They show a case where only area attributes are available on the message line and they take a screen position. The area attributes in the normal screen area do not.

19.4.7 Function Key Labels

Certain terminals set aside areas on the screen, typically two lines high and several characters wide, into which descriptive labels for the terminal's function keys may be written. The KPAR entry gives the number and width of the function key label areas, and looks like KPAR = NUMBER = number of labels LENGTH = width of area The KSET entry gives the character sequence for writing text into a label area. It is passed three parameters:

1. The number of the area to be written.
2. Twice the width of the area (LENGTH parameter of KPAR).
3. The label text, as a null-terminated string.

Here is an example, for the HP-2392A:

```
KPAR = NUMBER = 8 LENGTH = 8
KSET = ESC & f 0 a %d k %d d 0 L %s ESC & j B
```

19.4.8 Graphics and Foreign Character Support

JYACC FORMAKER has support for eight-bit ASCII codes as well as any graphics that the terminal can support in text mode. Bit-mapped graphics are not supported. Just as the key translation tables give a mapping from character sequences to internal numbers, the GRAPH table in the video file maps internal numbers to output sequences. The only character value that may not be sent is 0.

Some terminals have a special "compose" key, active in eight-bit mode. Generally, you would press the compose key followed by one or two more keys, generating a character in the range 0xa0 to 0xff. JYACC FORMAKER can process such characters as normal display characters, with no special treatment in the video file.

Other terminals have special keys that produce sequences representing special characters. The modkey utility can be used to map such sequences to single values in the range 0xa0 to 0xfe. (See the Programmer's Guide for a way to use values outside that range.) The video file would then specify how these values are output to the terminal.

Often, to display graphics characters, a terminal must be told to "shift" to an alternate character set (in reality, to address a different character ROM). The video file's GRAPH table tells which alternate set to use for each graphics character, and how to shift to it. Whenever JYACC FORMAKER is required to display a character, it looks in the GRAPH table for that character. If it is not there, the character is sent to the terminal unchanged. The following section describes what happens if it is in the table.

19.4.9 Graphics Characters

JYACC FORMAKER supports up to three alternate character sets. The sequences that switch among character sets are listed below. Modes 0 through 3 are locking shifts: all characters following will be shifted, until a different shift sequence is sent. Modes 4 through 6 are non-locking or single shifts, which apply only to the next character. You may need to use the INIT entry to load the character sets you want for access by the mode changes.

```
MODE0       switch to standard character set
MODE1       alternate set 1
MODE2       alternate set 2
MODE3       alternate set 3
MODE4       ...
MODE5
MODE6
```

Different modes can be used to support foreign characters, currency symbols, graphics, etc. JYACC FORMAKER makes no assumption as to whether the mode changing sequences latch to the alternate character set or not. To output a character in alternate set 2, JYACC FORMAKER first outputs the sequence defined by MODE2, then a character, and finally the sequence defined by MODE0 (which may be empty, if the others are all non-locking). Here are three examples; the second one is ANSI standard.

```
MODE0 = SI
MODE1 = SO
MODE2 = ESC n
MODE3 = ESC o

MODE0 = ESC [ 10 m
MODE1 = ESC [ 11 m
MODE2 = ESC [ 12 m
MODE3 = ESC [ 13 m

MODE0 =
MODE1 = SS1
MODE2 = SS2
```

Any character in the range 0x01 to 0xff can be mapped to an alternate character set by use of the keyword GRAPH. The value of GRAPH is a list of equations. The left side of each equation is the character to be mapped; the right side is the number of the character set (0, 1, 2, 3), followed by the character to be output. Any character not so mapped is output as itself. For example, suppose that 0x90 = 1 d appears in the GRAPH list. First the sequence listed for MODE1 will be sent, then the letter d, and then the sequence listed for MODE0.

In the following example, 0x81 is output as SO / SI, 0xb2 as SO 2 SI, and 0x82 as ESC o a SI. LF, BEL and CR are output as a space, and all other characters are output without change. This output processing applies to all data coming from JYACC FORMAKER. No translation is made for direct calls to printf, putchar, etc. Thus \n and \r will still work correctly in printf, and putchar (BEL) still makes a noise on the terminal.

```
MODE0 = SI
MODE1 = SO
```

```
MODE2 = ESC n
MODE3 = ESC o
GRAPH = 0x81 = 1 / 0xb2 = 1 2 0x82 = 3 a LF = 0 SP\
        BEL = 0 SP CR = 0 SP
```

For efficiency, we suggest that you use single shifts to obtain accented letters, currency symbols, and other characters that appear mixed in with unshifted characters; graphics characters, especially for borders, are good candidates for a locking shift.

It is possible, though not recommended, to map the usual display characters to alternates. For example, GRAPH = y = 0z will cause the y key to display as z. Graphics characters are non-portable across different displays, unless care is taken to insure that the same characters are used on the left-hand side for similar graphics, and only for a common subset of the different graphics available.

The GRTYPE keyword provides a convenient shortcut for certain common graphics sets, each denoted by another keyword. The format is GRTYPE = type. An entry in the GRAPH table is made for each character in the indicated range, with mode 0. If the mode is not 0, you must construct the GRAPH table by hand. The GRTYPE keywords are:

ALL	0xa0 through 0xfe
EXTENDED	same as ALL.
PC	0x01 through 0x1f and 0x80 through 0xff
CONTROL	0x01 through 0x1f, and 0x7f
C0	same as CONTROL
C1	0x80 through 0x9f, plus 0xff

The GRTYPE keywords may be combined.

19.4.10 Borders

Ten different border styles may be selected when a form is designed. They are numbered 0 to 9, with style 0 being the default (and the one all the JYACC FORMAKER internal forms use). It is usually reverse video spaces, but is replaced by I's if reverse video is not available. Border styles may be specified in the video file. Otherwise the following defaults are used:

```
0.   IIIII          1.    ___
     I   I                |   |
     IIIII                |___|

2.   +++++          3.     ===
     +   +                |   |
     +++++                 ===

4.   %%%%%          5.   .....
     %   %                :   :
     %%%%%                :...:

6.   *****          7.   \\\\\
     *   *                \   \
     *****                \\\\\

8.   /////          9.   #####
     /   /                #   #
     /////                #####
```

The keyword BORDER specifies alternate borders. If fewer than 9 are given, the default borders are used to complete the set. The data for BORDER is a list of 8 characters per border, in the order: upper left corner, top, upper right corner,

left side, right side, lower left corner, bottom, lower right corner. The default border set is:

```
BORDER =  SP  SP  SP  SP  SP  SP  SP  SP \
          SP  _   SP  |   |   |   _   |  \
          +   +   +   +   +   +   +   +  \
          SP  =   SP  |   |   SP  =   SP \
          %   %   %   %   %   %   %   %  \
          .   .   .   :   :   :   .   :  \
          *   *   *   *   *   *   *   *  \
          \   \   \   \   \   \   \   \  \
          /   /   /   /   /   /   /   /  \
          #   #   #   #   #   #   #   #
```

Another example, using the PC graphics character set:

```
BORDER =  SP   SP   SP   SP   SP   SP   SP   SP   \
          0xda 0xc4 0xbf 0xb3 0xb3 0xc0 0xc4 0xd9 \
          0xc9 0xcd 0xbb 0xba 0xba 0xc8 0xcd 0xbc \
          0xd5 0xcd 0xb8 0xb3 0xb3 0xd4 0xcd 0xbe \
          0xd6 0xc4 0xb7 0xba 0xba 0xd3 0xc4 0xbd \
          0xdc 0xdc 0xdc 0xdd 0xde 0xdf 0xdf 0xdf \
          .    .    .    :    :    :    .    .    \
          0xb0 0xb0 0xb0 0xb0 0xb0 0xb0 0xb0 0xb0 \
          0xb2 0xb2 0xb2 0xb2 0xb2 0xb2 0xb2 0xb2 \
          0xbd 0xbd 0xbd 0xbd 0xbd 0xbd 0xbd 0xbd
```

If there is a GRAPH entry in the video file, you can use the graphics character set of the terminal for borders. Choose some numbers to represent the various border parts. The GRAPH option can be used to map these numbers to a graphics character set. The numbers chosen are arbitrary, except that they should not conflict with ordinary display characters. Even if the extended 8 bit character set is used, there are unused values in the ranges 0x01 to 0x1f and 0x80 to 0x9f.

The keyword BRDATT can be used to limit the attributes available in the border. Normally HILIGHT (or DIM) and REVERSE are used; however, if the terminal uses onscreen attributes or can hold only a few attributes per line, it may be better to prohibit attributes in borders. This is accomplished by BRDATT = NONE.

The flags used in MSGATT can also be used with BRDATT; however, the only attributes available are HILIGHT, DIM, and REVERSE.

19.4.11 Shifting Field Indicators and Bell

Shift indicators (ARROWS keyword) are used to indicate the presence of off-screen data in shifting fields. The default characters for this purpose are <, > and X. (The last character is used when two shifting fields are next to each other; it represents a combination of both < and >.) The shift indicators can be changed to any three characters desired.

```
ARROWS = . . .

GRAPH = 0x1b = 0 0x1b   0x1a = 0 0x1a   0x1d = 0 0x1d
ARROWS = 0x1b 0x1a 0x1d

MODE0 = SI
MODE1 = SO
GRAPH = 0x80 = 1a 0x81 = 1x 0x82 = 1&
ARROWS = 0x80 0x81 0x82
```

The BELL sequence, if present, will be transmitted by the library function bel to give a visible alarm. Normally, that routine rings the terminal's bell. Such a sequence can sometimes be found in the termcap file under vb.

19.4.12 xform Status Text

The JYACC FORMAKER authoring utility will display help text on the status line if so desired. There are several different "states" in the utility, each with its own status text; the text to be displayed in each state is listed in the video file. (Logically it belongs in the message file; however, the text mentions keys to use and uses visual attributes. Since the keys and attributes are terminal-dependent, we store the text in the video file.)

Since vid2bin strips spaces, embedded spaces should be entered with the SP mnemonic, or the whole text enclosed in quotes. Attributes can be embedded in the text by using %a as a lead-in; up to four hex digits following define the attribute, using the codes defined in smdefs.h . See d_msg_line in the library manual for a fuller explanation of embedded attributes.

The following is a sample without embedded attributes. Function keys 2 to 9 are used.

```
FMKRDS = "2: DRAW/test 3: form 4: field 5: tmplt "\
         "6: del 7: move 8: copy 9: rept"
FMKRTM = "2: TEST/draw 3: form 4: field 5: tmplt "\
         "6: del 7: move 8: copy 9: rept"
FMKRMV = "MOVE: use arrow keys to position, F7 to release"
FMKRCP = "COPY: use arrow keys to position, F8 to release"
```

The next group is similar except that the numbers are given the reverse video blue attribute. The text is given the normal (i.e. white) attribute. (The color is ignored on monochrome terminals.) The text listed here is the default.

```
FMKRDS = %a11 2: %a07 SP DRAW/test SP \
         %a11 3: %a07 SP form SP      \
         %a11 4: %a07 SP field SP     \
         %a11 5: %a07 SP tmplt SP     \
         %a11 6: %a07 SP del SP       \
         %a11 7: %a07 SP move SP      \
         %a11 8: %a07 SP copy SP      \
         %a11 9: %a07 SP rept

FMKRTM = %a11 2: %a07 SP TEST/draw SP \
         %a11 3: %a07 SP form SP      \
         %a11 4: %a07 SP field SP     \
         %a11 5: %a07 SP tmplt SP     \
         %a11 6: %a07 SP del SP       \
         %a11 7: %a07 SP move SP      \
         %a11 8: %a07 SP copy SP      \
         %a11 9: %a07 SP rept
FMKRMV = %a11 7: %a07 \
   " MOVE:  use arrow keys to position, F7 to release"
FMKRCP = %a11 8: %a07 \
   " COPY:  use arrow keys to position, F8 to release"
```

19.4.13 Cursor Position Display

The utility will display the current cursor position on the status line if desired. When possible, JYACC FORMAKER uses nonblocking keyboard reads. If no key is obtained within a specified time, the cursor position display is updated. This allows fast typists to type at full speed; when the typist pauses, the cursor position display is updated. The keyword CURPOS specifies the timeout delay, in tenths of a second. If the keyword is omitted, or is 0, there will be no cursor position display. Many terminals display the cursor position themselves.

The delay depends on the baud rate and the terminal itself; it should be chosen so that typing is not slowed down. If the terminal has its own display, CURPOS should be omitted.

If there is no non-blocking read, a non-zero value of CURPOS enables the display and zero disables it.

```
CURPOS = 1     - update display every .1 sec
                 (use on fast systems)
CURPOS = 3     - every .3 sec (reasonable for most)
CURPOS = 7     - at low baud rates
CURPOS = 0     - no display, same as omitting keyword
```

Appendix A Error Messages

In this Appendix, all the error messages issued by the JYACC FORMAKER run-time system and utilities appear. Each message is listed, with its tag, as it appears in the message file distributed by JYACC; even if you change the wording of these messages, the tag will remain the same. If you modify the message file extensively, you may want to keep the original around for correlation with this list. Some messages have slots for information determined at run-time; these appear as printf percent escapes, commonly %s for character strings and %d for numbers.

Each message is followed by a less terse description of the error condition and the contexts in which it can arise. If recovery is necessary and possible, you will also find recommendations on how to recover from the error.

The run-time and screen editor messages are currently in message file order, which is perhaps not the most useful. Utility messages are alphabetical by utility.

20 Run-time Messages

SM_BADTERM = Unknown terminal type. SM_ENTERTERM = Please enter terminal type or %KNL to exit. Cause: The library function sm_initcrt cannot find the configuration files it needs to talk to your terminal. Corrective action: Check your SMVIDEO, SMKEY, SMTERM, and SMVARS setup variables. You can proceed by typing the name of your terminal in response to this message, but that's tedious.

SM_MALLOC = Insufficient memory available. Cause: The screen manager uses the C library function malloc() to get memory when needed. It has exhausted the area reserved for dynamic allocation, or perhaps the area has been corrupted. Corrective action: Exit the program.

SM_KEYENV = SMKEY not found. Cause: The file named in the SMKEY setup variable cannot be opened. This will cause initialization to be aborted. Corrective action: Correct the environment variable. Perhaps you need to re-run the key2bin utility.

SM_VIDENV = SMVIDEO not found. Cause: The file named in the SMVIDEO setup variable cannot be opened. This will cause initialization to be aborted. Corrective action: Correct the environment variable. Perhaps you need to re-run the vid2bin utility.

SM_FNUM = Bad field # or subscript. Cause: A field number (following #) or occurrence number (in []'s following a field name or number) is out of range. Corrective action: Correct the math edit or JPL program that contains the errant number.

SM_DZERO = Divide by zero. Cause: Your math expression has caused division by zero. Corrective action: Find the zero. You may need to make a field data-required, as blank fields have a numeric value of zero.

SM_EXPONENT = Exponentiation invalid. Cause: Your math expression has attempted to raise zero to a negative power, or to raise a negative number to a fractional power. Corrective action: Fix the exponential expression.

SM_DATE = Invalid date. Cause: The date in a date field is not formatted according to the field's date edit string. Corrective action: Re-enter the date.

SM_MATHERR = Math error - Cause: Used as a prefix to other math error messages. Corrective action: None.

SM_FORMAT = Invalid format. Cause: The precision expression that precedes a math expression is malformed. Corrective action: It should be %m.n, where m is the total width of the result and n is the number of decimal places.

SM_DESTINATION = Invalid destination. Cause: The destination field expression that begins a math expression is not followed by an equal sign. Corrective action: Supply the equal sign.

SM_INCOMPLETE = Expression incomplete. SM_ORAND = Operand expected. SM_ORATOR = Operator expected. SM_EXTRAPARENS = Right parenthesis unexpected. SM_MISSPARENS = Right parenthesis expected. Cause: The right-hand side of a math expression is missing or malformed. Corrective action: Correct the expression.

SM_DEEP = Formula too complicated. Cause: The internal stack used to store intermediate results in math expression evaluation has overflowed. Corrective action: Simplify the expression, or use an intermediate.

SM_FUNCTION = Invalid function. Cause: The name following the @ in a math expression is not "date", "sum", or "abort". Corrective action: Use one of the built-in functions.

SM_ARGUMENT = Invalid argument. Cause: The argument to @abort in a math expression is not a number. Corrective action: The meaningful arguments to @abort are -2, -1, 0, and 1. Use one of those.

SM_MISMATCH = Type mismatch. Cause: A comparison between numeric and string variables has been attempted in a math expression. Corrective action: Check the types or character edits of the data elements involved.

SM_NOTMATH = Not a math expression. Cause: JAM couldn't get to first base trying to evaluate a math expression edit. Corrective action: Check the Author's Guide for a description of math expression syntax.

SM_QUOTE = Missing quote character. Cause: A math or string expression contains an unclosed quote. Corrective action: Supply the missing quote.

SM_SYNTAX = Syntax error. Cause: Extra characters at the end of a math expression, or a malformed relational operator. Corrective action: Correct the indicated problem.

SM_FRMDATA = Bad data in form. Cause: A file you have attempted to open as a JAM screen is not a screen file, or was created with a different release version of JAM, or has been corrupted. Corrective action: Check the screen name, then try to bring it up in the screen editor.

SM_NOFORM = Cannot find form. Cause: JAM cannot open the form file you have requested. Corrective action: Check that the file exists, and/or that proper entries have been made in the SMPATH directory list, the memory-resident form list, and the SMFLIBS library list.

SM_FRMERR = Error while reading form. Cause: This refers to I/O errors in reading a form file from disk. Corrective action: Retry the operation.

SM_BIGFORM = Form has fields that extend beyond screen size. Cause: You have tried to display a form that won't fit on your terminal. Corrective action: Reduce the screen's size, using the screen editor.

SM_SP1 = Please hit the space bar SM_SP2 = after reading this message. Cause: These two lines appear in a prompt window when an error message has been displayed and you have not acknowledged it by pressing the space bar, but by pressing some other key. Corrective action: Press the space bar. If you don't want to acknowledge the message, set the SMEROPTIONS setup variable.

SM_RENTRY = Entry is required. Cause: You have failed to enter data in a required field. Corrective action: Enter something. In digits-only fields, you must enter at least one digit.

SM_MUSTFILL = Must fill field. Cause: You have failed to fill a must-fill field. No blanks whatever are allowed there. Corrective action: Fill out the field.

SM_AFOVRFLW = Amount field overflow. Cause: You have typed a number that is too big for the field's currency format to accommodate. Corrective action: Reduce the number or increase the precision.

SM_TOO_FEW_DIGITS = Too few digits. SM_CKDIGIT = Check digit error. Cause: A number has failed check-digit validation. Corrective action: Re-enter the number.

SM_FMEM = Insufficient memory for data entry field. Cause: In trying to construct a field for data entry in a help screen, available memory was exhausted. Corrective action: Exit the program.

SM_NOHELP = No help text available. Cause: You have pressed the HELP key in a field where no help was available. Corrective action: Define a help screen for the field or screen.

SM_MAXHELP = Five help levels maximum. Cause: You have nested help windows too deeply. Corrective action: Restructure the help windows.

SM_FRMHELP = No form-level help text available. Cause: You have pressed the FORM HELP key in a screen with no form-wide help. Corrective action: Define a help screen for the form.

SM_OUTRANGE = Out of range. Cause: The string or number you have entered violates a range edit. Corrective action: Enter a correct value, or relax the range restrictions.

SM_SYSDATE = Use clear for system date or enter in format: Cause: The date in a system date field is not formatted according to the field's date edit string. Corrective action: Re-enter the date, or clear the field to get the current date.

SM_DATFRM = Invalid format; enter date in format: Cause: The date in a date field is not formatted according to the field's date edit string. Corrective action: Re-enter the date.

SM_DATCLR = Invalid date; clear gets system date. Cause: The date in a system date field is not formatted according to the field's date edit string. Corrective action: Re-enter the date, or clear the field to get the current date.

SM_DATINV = Invalid date; enter a valid date. Cause: The date in a date field is not formatted according to the field's date edit string. Corrective action: Re-enter the date.

SM_SYSTIME = Use clear for system time or enter in format: Cause: The time in a system time field is not formatted according to the field's time edit string. Corrective action: Re-enter the time, or clear the field to get the current time.

SM_TIMFRM = Invalid format; enter time in format: Cause: The time in a time field is not formatted according to the field's time edit string. Corrective action: Re-enter the time.

SM_TIMCLR = Invalid time; clear gets system time. Cause: The time in a system time field is not formatted according to the field's time edit string. Corrective action: Re-enter the time, or clear the field to get the current time.

SM_TIMINV = Invalid time; enter a valid time. Cause: The time in a time field is not formatted according to the field's time edit string. Corrective action: Re-enter the time.

SM_MOREDATA = No more data. Cause: You have attempted to scroll past the beginning or end of a non-circular scrolling field. Corrective action: Warning only.

SM_SCRLMEM = Insufficient memory for scrolling. Cause: Ran out of memory for scroll buffers. Corrective action: Exit the program.

SM_NOTEMP = Cannot open temporary file. Cause: The local print function failed to open its scratch file. Corrective action: Check write permissions in your directory.

SM_NOFILE = '"%s" not found' Cause: A file needed by the screen manager was missing. Corrective action: Supply the file, or correct the environment variable that points to it.

SM_NOENV = "'%s' missing" SM_NOSECTOR = section '%2.2s' not found SM_FFORMAT = bad file format in "%s" SM_FREAD = file read error in "%s" Cause: There was a problem initializing one of the configuration files (the key file, video file, msgfile, smvars or setup). Corrective action: Check the contents of the text file, compile it again (with key2bin, vid2bin, msg2bin or var2bin), and try again.

SM_RX1 = Invalid character. Cause: The character you have typed is not allowed by the field's regular expression. Corrective action: Type an allowed character, or relax the expression.

SM_RX2 = Incomplete entry. Cause: The field's regular expression demands more data than you have entered. Corrective action: Supply the missing characters.

SM_RX3 = No more input allowed. Cause: The opposite problem: the field's regular expression demands fewer characters than you have entered. Corrective action: Shorten your input.

SM_TABLOOK = Invalid entry. Cause: The contents of a field have failed the table-lookup validation. Corrective action: Correct the

input (perhaps through item selection), or add the missing item to the table-lookup screen.

SM_ILLELSE = Illegal Else Cause: In a JPL program, an else has appeared without a preceding if. Corrective action: Correct the program's syntax.

SM_EOT = unexpected End Of File Cause: At the end of JPL program text, there are unclosed blocks. Corrective action: Supp;ly the missing right curly braces.

SM_BREAK = BREAK not within loop Cause: A JPL program contains a break command that is not inside a for or while loop. Corrective action: Remove the break.

SM_NOARGS = Verb needs arguments Cause: A JPL command that requires arguments has been given none. Corrective action: Supply the arguments; see the JPL Programmer's Guide.

SM_HASARGS = Illegal arguments Cause: A JPL command has excess arguments. Corrective action: Remove the excess.

SM_EOL = Source line too long Cause: A JPL program contains a logical line that is too long (currently, a couple of thousand characters). Corrective action: Figure out how to do it in multiple lines.

SM_EXCESS = Extra data at end of line Cause: In certain JPL commands, there is superfluous stuff following the command. Corrective action: Get rid of it.

SM_FILEIO = System File I/O error Cause: An I/O error has occurred while reading a JPL program file. Corrective action: Exit the program.

SM_FOR = USAGE: FOR varname = Value WHILE (expression) STEP [+-]value Cause: A JPL for command has a syntax error. Corrective action: Recast the command according to the given format.

SM_LINE_2_LONG = Line too long after expansion Cause: A line of a JPL program is too long after colon expansion (more than about 2000 characters). Corrective action: Check for missing subscripts: a name with multiple occurrences but no subscript in the expression is replaced by all the occurrences.

SM_NOFILE = Could not open file Cause: A JPL program source file was missing or unreadable. Corrective action: Create the file, correct its spelling in the program, or add its directory to your SMPATH.

SM_NONAME = Expected variable name Cause: An entry in a JPL vars command does not begin with a letter, $, ., or _. Corrective action: Fix the name.

SM_NOTARGET = Target does not exist Cause: The field to be assigned to in a JPL math or cat command is not in the screen or LDB. Corrective action: Create the field or change the command.

SM_NUMBER = Illegal Number Cause: The argument to a JPL return statement was invalid. Corrective action: It must be an integer constant, variable name, or LDB name - no expressions.

SM_RCURLY = Ended block not begun Cause: A JPL program has too many right curlies. Corrective action: Remove some.

FM_BADENTRY = Bad entry. Cause: In the field size window, you have specified a vertical array without giving an offset. Corrective action: Supply the offset.

FM_MXSCRN = Maximum number of %s on the screen is %d. Cause: You have tried to make your screen bigger than the display, using the PF3 window; the maximum possible values are in the message. Corrective action: Specify a smaller screen.

FM_MNBRDR = Minimum number of %s to hold form data and a border is %d. Cause: In the PF3 window, you have tried to make the screen smaller than the existing data plus border. Corrective action: Make the screen larger, or move or delete some of the contents.

FM_MNFORM = Minimum number of %s to hold form data is %d. Cause: In the PF3 window, you have tried to make the screen smaller than the existing data. Corrective action: Make the screen larger, or move or delete some of the contents.

FM_NOOPEN = Cannot create form %s. Cause: The editor was unable to create the file whose name is in the message, probably for lack of permission or space. Corrective action: Write the screen to a different file, or escape to the command interpreter and correct the problem.

FM_WRFORM = Error writing form '%s'. Cause: The editor incurred an I/O error while writing out the screen file. Corrective action: Try writing to a different file.

FM_NOFROOM = Insufficient memory for new fields. Cause: No fields can be added because the editor has run out of memory. Corrective action: Write the screen out at once, exit the editor, and re-edit the screen.

FM_ARHROOM = No room for horizontal array. Cause: In the field size window, you have specified a horizontal array that will fall outside the screen. Corrective action: Make the screen bigger, or the array smaller.

FM_ARVROOM = No room for vertical array. Cause: In the field size window, you have specified a vertical array that will fall outside the screen. Corrective action: Make the screen bigger, or the array smaller.

FM_ARHVSEL = Enter v or h. Cause: I'm not sure this error message is correct. Corrective action:

FM_AROVERLAP = Overlaps existing field. Cause: You have specified an array that would overlay part of an existing field. Corrective action: Change the array size or move the field.

FM_UCSET = Set upper or lower case. Cause: You have specified both upper- and lower-case in the field edits window. Corrective action: Type 'n' for one or the other.

FM_SHRNG = The shifting increment must be at least 1, but no more than %d.
Cause: You have specified a shifting increment of zero, or greater than the onscreen width of the field. Corrective action: Change the shift increment to a value in the proper range, indicated in the message.

FM_FLDLEN = Length must be non-zero and no greater than %d. Cause: In the field size or summary window, you have tried to make a field so long that it reaches out of the screen. Corrective action: Make the field shorter or move its origin to the left.

FM_GRNONE = Graphics not available on this terminal. Cause: You have pressed the graphics key (PF10 or SPF5), but your display's video file contains no definitions for graphics characters. Corrective action: Put the appropriate entries (GRAPH, GRTYPE, MODE1-6) in your video file.

FM_OVERLAP = Overlaps field or border. Cause: In creating a JAM control field or moving an ordinary field, you have placed it so that it would overlap another field or the screen's border. Corrective action: Reposition the new field.

FM_NAMEINUSE = Name already assigned to another field. Cause: You have tried to give a field a name that already belongs to another field. Corrective action: Rename one of the fields.

FM_FLDNO = Invalid field number. Cause: In specifying a next-field edit, you have given a target field number (using #) that is out of range for the screen. Corrective action: Change the field number to refer to an existing field.

FM_INCR = Invalid increment. Cause: In specifying a next-field edit, you have given a field increment (using + or -) that results in an occurrence number out of range for the field. Corrective action: Reduce the increment.

FM_FNUMB = Field number must start with #. Cause: In specifying a next-field edit, you have typed # for field number but have not put a number after it. Corrective action: Supply the field number.

FM_ELEMENT = Invalid element. Cause: In a next-field edit, your element specification contains a syntax error. Corrective action: The proper syntax is field-id[element].

FM_1FMT = Enter one format only. Cause: You have entered both a date and a time format string. Corrective action: Remove one of them; a field can be either date or time, but not both.

FM_CLCMIN = Minimum digits should not exceed length of field, which is %d.
Cause: In the math/check digit window, you have specified a minimum number of digits for the check-digit that is too large. Corrective action: Reduce the minimum below the number in the message, or make the field longer.

FM_AZNAME = Name must start with letter. Cause: You have typed a field name that does not begin with a letter. Corrective action: Change the field name.

FM_A9NAME = Must be alpha, number or '_'. Cause: You have typed a character elsewhere in a field name that is neither alphanumeric nor an underscore. Corrective action: Remove the offending character from the name.

FM_INBORDER = Bad entry -- field in prospective border. Cause: You have requested a border on a screen that has fields at the very edge of the screen, where the border should go. Corrective action: Move the offending field or fields.

FM_DUPDRAW = Duplicate draw character. Cause: In the draw-field/field defaults window, you have specified a draw-field character twice. Corrective action: Pick another character.

FM_IFORMAT = Invalid format. Cause: In specifying a window name and coordinates for a help screen, sub-menu, or other edit, you have deviated from the prescribed format screen-name (line, column) Perhaps you have left out a parenthesis, or omitted the comma. Corrective action: Correct the format.

FM_INVRC = Invalid menu return code. Cause: You have specified a menu return code that does not evaluate to an integer. Corrective action: Allowable return codes are: decimal numbers; hexadecimal numbers; quoted printable ASCII characters, as 'q'; ASCII control character mnemonics, as ESC; and JAM logical key mnemonics from smkeys.h.

FM_WRMSK = A word wrap field may not have a regular expression edit. Cause: You have attempted to create a field with both word wrapping and a regular expression edit. Because word wrap interprets certain characters specially, this is not allowed. Corrective action: Choose one. As word-wrapped fields are generally used for large quantities of text, they are best left unfiltered.

FM_RX1 = Regular expression too long. Cause: When compiled, the regular expression you have typed is to long to be stored as a special edit. Corrective action: Try to simplify the expression.

FM_RX2 = Unbalanced '[' bracket. Cause: A left bracket that begins a character class has no matching right bracket. Corrective action: If you want a literal left bracket, quote it: \[. If you really wanted a character class, insert the corresponding right bracket.

FM_RX3 = Too many '(' brackets. FM_RX4 = Too many ')' brackets. FM_RX8 = Closing '}' brace expected. FM_RX11 = Previous '(' bracket not yet closed. Cause: Various cases of bracket imbalance. Corrective action: As above, the usual cause is forgetting to quote a bracketing character when you want it literally.

FM_RX5 = Expecting number between 0-9 or '\}'. Cause: You have put something other than a number inside a subexpression repeat count. Corrective action: Remove it.

FM_RX6 = Range may not exceed 255. Cause: You have specified a repeat count greater than what will fit in a field (fields are limited to 255 characters in width). Corrective action: Reduce the count.

FM_RX7 = Too many commas in specifying range. Cause: You have put two consecutive commas in a range expression. Corrective action: Remove one.

FM_RX8 = Closing '}' brace expected. Cause: You have followed a comma in a range expression with a closing curly brace. Corrective action: Remove the comma, or put another number after it.

FM_RX9 = First number exceeds second in specifying range. Cause: You have got the range of a range backwards. Corrective action: Reverse or correct the range limits.

FM_RX10 = \digit out of range. Cause: You have entered a backslash followed by a number to re-match a subexpression, but the number exceeds the number of parenthesized subexpressions. Corrective action: Reduce the number or parenthesize the correct subexpressions.

FM_RX12 = Unexpected end of regular expression. Cause: Your regular expression ends with a backslash. Corrective action: If you want a literal backslash, double it.

22 Utility Messages

These messages are also listed in the Configuration Guide with their utilities; they are repeated here for convenience.

bin2c Messages

Insufficient memory available. Cause: The utility could not allocate enough memory for its needs. Corrective action: None.

File "%s" already exists; use '-f' to overwrite. Cause: You have specified an output file that already exists. Corrective action: Use the -f flag to overwrite the file, or use another name.

Cannot open "%s" for writing. Cause: An output file could not be created, due to lack of permission or perhaps disk space. Corrective action: Correct the file system problem and retry the operation.

Cannot open "%s" for reading. Cause: An input file was missing or unreadable. Corrective action: Check the spelling, presence, and permissions of the file in question.

Error reading file "%s" Cause: The utility incurred an I/O error while processing the file named in the message. Corrective action: Retry the operation.

Error writing file "%s" Cause: The utility incurred an I/O error while processing the file named in the message. Corrective action: Retry the operation.

b2hex Messages

Error reading %s Error writing %s Cause: The utility incurred an I/O error while processing an input or output file. This message will usually be accompanied by a more specific, system-dependent message. Corrective action: Correct the system-dependent problem, if possible, and retry the operation.

%s already exists %s already exists, it is skipped Cause: The command you have issued would overwrite an existing output file. Corrective action: If you are sure you want to destroy the old file, reissue the command with the -f option.

f2struct Messages

Language %s undefined. Cause: The language you have given with the -g option has not been defined in the utility's tables. Corrective action: Check the spelling of the option, or define the language ito the utility.

%s already exists. Cause: You have specified an existing output file. Corrective action: Use the -f option to overwrite the file, or use a different name.

%s has an invalid file format. Cause: An input file is not of the expected type. Corrective action: Check the spelling and type of the offending file.

'%s' has no data to convert. Cause: An input file is empty, or does not have the names you specified. Corrective action: Check the names.

Not enough memory to process '%s'. Unable to allocate memory. Cause: The utility could not allocate enough memory for its needs. Corrective action: None.

At least one form name is required. Cause: You have not given any screen files as input. Corrective action: Supply one or more screen file names.

formlib Messages

Library `%s' already exists; use `-f' to overwrite. Cause: You have specified an existing output file. Corrective action: Use the -f option to overwrite the file, or use a different name.

Cannot open `%s'. Cause: An input file was missing or unreadable. Corrective action: Check the spelling, presence, and permissions of the file in question.

Unable to allocate memory. Insufficient memory available. Cause: The utility could not allocate enough memory for its

needs. Corrective action: None.

File `%s' is not a library. Cause: The named file is not a form library (incorrect magic number). Corrective action: Check the spelling and existence of your library.

`%s' not in library. No forms in library. Cause: A screen you have named is not in the library. Corrective action: List the library to see what's in it, then retry the operation.

Temporary file `%s' not removed. Cause: The intermediate output file was not removed, probably because of an error renaming it to the real output file. Corrective action: Check the permissions and condition of the files, then retry the operation.

key2bin Messages

File '%s' not found Neither '%s' nor '%s' found. Cause: An input file was missing or unreadable. Corrective action: Check the spelling, presence, and permissions of the file in question.

Unknown mnemonic in line: '%s' Cause: The line printed in the message does not begin with a logical key mnemonic. Corrective action: Refer to smkeys.h for a list of mnemonics, and correct the input.

No key definitions in file '%s' Cause: Warning only. The input file was empty or contained only comments. Corrective action: None.

Malloc error Cause: The utility could not allocate enough memory for its needs. Corrective action: None.

Cannot create '%s' Error writing '%s' Cause: An output file could not be created, due to lack of permission or perhaps disk space. Corrective action: Correct the file system problem and retry the operation.

lstform Messages

Error opening input file. Cause: An input file was missing or unreadable. Corrective action: Check the spelling, presence, and permissions of the file in question.

Error opening output file. Cause: An output file could not be created, due to lack of permission or perhaps disk space. Corrective action: Correct the file system problem and retry the operation.

Unable to allocate memory. Can't allocate memory. Cause: The utility could not allocate enough memory for its needs. Corrective action: None.

Error reading form file. Error writing list file. Cause: The utility incurred an I/O error while processing the file named in the message. Corrective action: Retry the operation.

modkey Messages

Invalid entry. Cause: You have typed a key that is not on the menu. Corrective action: Check the instructions on the screen and try again.

Key sequence is too long. Cause: You have typed more than six keys wihout repeating any. Corrective action: Key sequences for translation may be at most six characters long. Choose a shorter sequence.

Invalid first character. Cause: A multi-key sequence must begin with a control character. Corrective action: Begin again, using a control character.

Invalid mnemonic - press space for list Cause: In the miscellaneous keys screen, you have typed a character string for logical value that is not a logical key mnemonic. Corrective action: Peruse the list, then correct the input.

Invalid number - enter <decimal>, 0<octal> or 0x<hex> Cause: In the miscellaneous keys screen, you have typed a malformed numeric key code. Corrective action: Correct the number, or use a mnemonic.

Cannot create output file. Cause: An output file could not be created, due to lack of permission or perhaps disk space. Corrective action: Correct the file system problem and retry the operation.

Key sequence does not repeat. Cause: You have typed a key sequence that failed to repeat a string of six characters or less. Corrective action: Retry the sequence, or use a shorter one.

Cannot accept NUL as a key. Cause: The ASCII NUL character (binary 0) cannot be used in a key translation sequence, because it is used internally to mark the end of a sequence. Corrective action: Use another key.

Key previously defined as %s Key conflicts with %s Cause: You have typed a key sequence that has already been assigned to another key, or that is a substring of a previously assigned sequence.

Corrective action: Use a different key or sequence, or reassign the other.

msg2bin Messages

File '%s' not found. Cause: An input file was missing or unreadable. Corrective action: Check the spelling, presence, and permissions of the file in question.

Unable to allocate memory. Cause: The utility could not allocate enough memory for its needs. Corrective action: None.

Bad tag in line: %s Cause: The input file contained a system message tag unknown to the utility. Corrective action: Refer to smerror.h for a list of tags, and correct the input.

Missing '=' in line: %s Cause: The line in the message had no equal sign following the tag. Corrective action: Correct the input and re-run the utility.

term2vid Messages

No cursor position (cm, cup) for %s Cause: An absolute cursor positioning sequence is required for JYACC FORMAKER to work, and the termcap or terminfo entry you are using does not contain one. Corrective action: Construct the video file by hand, or update the entry and retry.

Cannot find entry for %s Cause: The terminal mnemonic you have given is not in the termcap or terminfo database. Corrective action: Check the spelling of the mnemonic.

File %s already exists; use '-f' to overwrite. Cause: You have specified an existing output file. Corrective action: Use the -f option to overwrite the file, or use a different name.

txt2form Messages

Warning: lines greater than %d will be truncated Warning: columns greater than %d will be truncated Cause: Your input text file has data that reaches beyond the limits you have given (default 23 lines by 80 columns) for the screen. Corrective action: Shrink the input, or enlarge the screen.

Unable to create output file. Cause: An output file could not be created, due to lack of permission or perhaps disk space. Corrective action: Correct the file system problem and retry the operation.

var2bin Messages

Error opening %s. Cause: An input file was missing or unreadable. Corrective action: Check the spelling, presence, and permissions of the file in question.

Missing '='. Cause: The input line indicated did not contain an equal sign after the setup variable name. Corrective action: Insert the equal sign and run var2bin again.

%s is an invalid name. Cause: The indicated line did not begin with a setup variable name. Corrective action: Refer to the Configuration Guide for a list of variable names, correct the input, and re-run the utility.

%s may not be qualified by terminal type. Cause: You have attached a terminal type list to a variable which does not support one. Corrective action: Remove the list. You can achieve the desired effect by creating different setup files, and attaching a terminal list to the SMSETUP variable.

Unable to set given values. %s conflicts with a previous parameter. %s is an invalid parameter. Cause: A keyword in the input is misspelled or misplaced, or conflicts with an earlier keyword. Corrective action: Check the keywords listed in the manual, correct the input, and run the utility again.

Error reading smvars or setup file. Cause: The utility incurred an I/O error while processing the file named in the message. Corrective action: Retry the operation.

Unable to allocate memory. Cause: The utility could not allocate enough memory for its needs. Corrective action: None.

At least one file name is required. Cause: You have failed to give an input file name. Corrective action: Retype the command, supplying the file name.

Entry size %d is too large. String size %d is too large. Cause: The indicated right-hand side is too long. Corrective action: Reduce the size of the entry.

vid2bin Messages

Neither %s nor %s exists. Cause: An input file was missing or unreadable. Corrective action: Check the spelling, presence, and permissions of the file in question.

A cursor positioning sequence is required. An erase display sequence is required. Cause: These two entries are required in all video files. Corrective action: Determine what your terminal uses to perform these two operations, and enter them in the video file; then run the utility again.

Unable to allocate memory. Cause: The utility could not allocate enough memory for its needs. Corrective action: None.

Error writing to file '%s'. Cause: The utility incurred an I/O error while processing the file named in the message. Corrective action: Retry the operation.

Invalid entry: '%s'. Entry missing '=': '%s'. Cause: The input line in the message does not begin with a video keyword and an equal sign. Corrective action: Correct the input and re-run the utility. You may have

forgotten to place a backslash at the end of a line that continutes onto the next one.

Invalid attribute list : '%s'. Invalid color specification : '%s'. Invalid graphics character specification (%s):'%s'. Invalid border information (%s):'%s'. Invalid graphics type : '%s'. Invalid label parameter : '%s'.%s Invalid cursor flags specification : '%s'. Cause: You have misspelled or misplaced keywords in the input line in the message. Corrective action: Correct the input, referring to the Configuration Guide, and run vid2bin again.

skipsomething

Index

In this Index, library functions are displayed in boldface, without the prefixes specific to the language interface. Video and setup file entries appear in ELITE CAPS, while utility programs and JPL commands are in elite lower-case. Function key names are in ROMAN CAPS.